MCSD Training Kit

Analyzing Requirements and Defining Solution Architectures

For Exam 70-100

Scott F. Wilson
Bruce Maples
Tim Landgrave

Microsoft Press

Принципы проектирования и разработки программного обеспечения

Учебный курс MCSD

Официальное пособие Microsoft для самостоятельной подготовки к экзамену 70-100

Скотт ***Ф.*** *Уилсон*
Брюс *Мэйплс*
Тим *Лэндгрейв*

Издание второе, исправленное

Москва 2002

РУССКАЯ РЕДАКЦИЯ

УДК **004.45**
ББК 32.973.26-018.2
М59

Microsoft Corporation

М59 Принципы проектирования и разработки программного обеспечения. Учебный курс MCSD/Пер. с англ. — 2-е изд., испр. — М.: Издательско-торговый дом «Русская Редакция», 2002. — 736 стр.: ил.

ISBN 5–7502–0213–5

Настоящий учебный курс посвящен разработанной компанией Microsoft методологии проектирования и разработки программного обеспечения Microsoft Solutions Framework. Эта методология вобрала в себя последние достижения в области проектирования и разработки приложений масштаба предприятия. Значительное внимание уделено вопросам управления проектами разработки программного обеспечения.

Книга адресована менеджерам проектов и разработчикам программного обеспечения. Кроме того, этот учебный курс рекомендован корпорацией Microsoft для самостоятельной подготовки к обязательному экзамену №70-100 по программе сертификации разработчиков Microsoft Certified Solution Developer (MCSD).

Книга состоит из 14 глав, 10 практикумов, приложения и предметного указателя. К учебному курсу прилагается компакт-диск с учебными и демонстрационными материалами словарем терминов, электронной версией учебного курса и программным обеспечением для проведения экзаменационного теста.

УДК 004.45
ББК 32.973.26–018.2

Подготовлено к изданию по лицензионному договору с Microsoft Corporation. Редмонд, Вашингтон, США.

ISBN 0–7356–0854–7 (англ.)
ISBN 5–7502–0213–5

Содержание

Об этой книге

Мы рады представить вам учебный курс «Принципы проектирования и разработки программного обеспечения. Учебный курс MCSD», предназначенный для подготовки к экзамену 70-100 по программе сертификации специалистов Microsoft Certified Solution Developer (MCSD).

Примечание Дополнительную информацию о программе сертификации разработчиков по программе Microsoft Certified Solution Developer см. в разделе «Программа сертификации специалистов Microsoft».

Кому адресована эта книга

Настоящий учебный курс адресован всем, кто хочет изучить методы анализа требований к программным системам и проектирование архитектуры решений. В книгу включены следующие темы: разработка распределенных приложений с помощью методики Microsoft Solutions Framework (MSF), разработка многоуровневых приложений и приложений архитектуры клиент-сервер, создание компонентов для работы под управлением сервера транзакций Microsoft Transaction Server (MTS) и специализированных интерфейсов *модели компонентных объектов* (Component Object Model, COM).

Требования к читателям

Для успешного изучения материала и понимания концепций и задач, изложенных в учебном курсе, требуется предварительная подготовка. Как минимум, от вас требуются:

- основные навыки программирования;
- умение создавать и компилировать простые приложения;
- понимание основ реляционных баз данных.

С чего начать

Настоящий учебный курс поможет вам подготовиться к экзамену 70-100: *Analyzing Requirements and Defining Solution Architectures.* Для изучения материала вам не потребуется ни компьютер, ни программное обеспечение. Однако, если вы захотите познакомиться с учебным приложением «Система управления ресурсами» (Resource Management System, RMS), активно используемым в практикуме, вам понадобится аппаратное и программное обеспечение, о котором подробнее рассказано в следующих разделах.

Требования к аппаратному обеспечению для работы с учебным приложением

Конкретные требования к аппаратуре, необходимой для работы Microsoft Windows NT Server, Microsoft Internet Information Server, Microsoft Transaction Server и Microsoft SQL Server приведены в документации к этим программным продуктам, Минимальные рекомендуемые конфигурации сервера и клиентской рабочей станции таковы:

- Pentium II 266 МГц или другой процессор эквивалентной мощности;
- 128 Мб памяти;
- жесткий диск 4 Гб;
- сетевой адаптер и другие компоненты, необходимые для подключения к сети;
- дисковод для компакт-дисков.

Требования к программному обеспечению для работы с учебным приложением

Для работы с учебным приложением, помещенным на компакт-диск учебного курса, необходимо следующее программное обеспечение.

Программное обеспечение, устанавливаемое на сервере:

- Windows NT Server 4.0 с установленным сервисным пакетом 4;
- Windows NT Option Pack 4, Internet Information Server 4.0 и Microsoft Transaction Server 2.0:
- Microsoft Exchange Server 5.5 с сервисным пакетом 2;
- Microsoft Collaborative Data Objects 1.21 и Microsoft Outlook 98/2000 (на компьютере, где установлен Microsoft Transaction Server и бизнес-компоненты приложения RMS):
- Microsoft SQL Server 7.0;
- Microsoft Data Access Components 2.1;
- Microsoft Internet Explorer 4.01 (или следующая версия).

Программное обеспечение, устанавливаемое на клиентской рабочей станции:

- Windows NT Server 4.0 с установленным сервисным пакетом 4, Windows 98 или Windows 95 (с установленной поддержкой DCOM95);
- распространяемые компоненты Microsoft Data Access 2.1 устанавливаются одновременно с клиентским ПО учебного приложения; дополнительно необходимо установить распространяемые компоненты ADOR 1.5;
- Internet Explorer 4.01 или более поздняя версия;
- Microsoft Outlook 98/2000.

Обзор глав и приложений

Занятия, упражнения и проверочные вопросы учебного курса помогут вам освоить методы анализа требований к приложениям и проектирования их архитектуры. Конечно, учебный курс предназначен для последовательного изучения, однако вы можете работать с ним так, как вам удобнее, скажем, проштудировать лишь отдельные главы.

Ниже кратко описаны главы и приложения учебного курса.

- Во вводной главе «Об этой книге» вы найдете информацию предварительного характера и сведения о содержании учебника. Внимательно прочитайте ее: это поможет эффективнее изучать материал или быстро выбрать интересующую вас тему.
- Глава 1 «Производственная архитектура»: в этой главе вы узнаете, почему разработку приложений и развитие инфраструктуры надо ориентировать на базовые концепции уровня предприятия. Здесь рассмотрена концепция разработки систем с позиции «приоритета архитектуры», созданные компанией Microsoft принципы разработки приложений Microsoft Solutions Framework (MSF), рассматривающие архитектуру с точки зрения бизнеса, приложений, информации и технологии, организация процесса разработки производственной архитектуры и создание систем и приложений, когда производственная архитектура еще не сформирована.
- Глава 2 «Приложения масштаба предприятия» начинается с описания функций современных производственных приложений и проблем, связанных с ними. Затем вы познакомитесь с десятью принципами успешного создания программных продуктов и проектированием крупномасштабных распределенных производственных приложений. Кроме того, здесь описаны методы снижения сложности таких проектов и модель производственных приложений, разработанная компанией Microsoft. В заключение обсуждается каркас архитектуры приложений, представленный в модели разработки приложений MSF.
- Из главы 3 «Проектные группы» вы узнаете, кто же непосредственно выполняет проект, познакомитесь с принципами создания проектных групп в контексте модели проектной группы MSF и с обязанностями членов проектной группы. Мы начнем с поиска кандидатов на роль руководителей и расскажем о шести основных направлениях работы группы. Затем мы познакомим вас с обязанностями участников проекта и их распределением среди членов группы. Далее мы расскажем о способах анализа требований к проекту с точки зрения его участников, методах масштабирования группы в соответствии с его значением и размером и, наконец,

перечислим требования к руководителям и проектной группе, которые позволят добиться эффективного управления проектом.

- Глава 4 «Процесс разработки» посвящена модели процесса разработки приложений MSF (MSF Process Model for Application Development), или, короче, модели процесса разработки. Эта модель - гибкий компонент обшей модели процесса MSF, с успехом применяемый в индустрии разработки программного обеспечения для повышения управляемости проектов, минимизации рисков, повышения качества продукции и ускорения разработки. Кроме того, в этой главе вы познакомитесь с практическими рекомендациями по проведению формального рецензирования, с критериями отбора участников этой процедуры и методами ее организации. Мы покажем, как организовывать и проводить соответствующие совещания, как собирать и документировать информацию. В завершение мы обсудим создание и роль специальной обзорной группы в больших проектах.
- В главе 5 «Предпроект» мы опишем динамику фазы «Анализ» модели процесса разработки MSF. Мы обсудим, какую информацию необходимо получить от участников проекта и как создавать концепцию продукта, расскажем о партнерстве различных ролей в проектной группе, созданной в соответствии с моделью группы разработчиков MSF, и выясним, какова область ответственности каждой роли на фазе «Анализ». Мы также посмотрим, как процесс исследования развивается во времени, и наконец, детально обсудим процесс управления рисками, базирующийся на модели управления рисками MSF.
- В главе 6 «План проекта» в общих чертах описан переход от «теории» к «практике» и рассказано о задачах проектной группы на стадии «Планирование». Мы подробно рассмотрим модель процесса разработки MSF наряду с концептуальной, логической и физической архитектурой приложения и интеграцию различных уровней модели приложения MSF (пользовательского, прикладного и уровня данных) в физическую архитектуру приложения. Мы остановимся на основных результатах рассматриваемого этапа: функциональных спецификациях, плане и графике проекта. И наконец, обсудим принципы разработки графика работ и контроль рисков, входящие в компетенцию руководителя проекта. В этой главе мы в основном рассказываем о трех моделях MSF: процессе разработки, процессе проектирования и разработке приложения.
- Из главы 7 «Технологии пользовательского уровня» вы узнаете, как создавать эффективный пользовательский интерфейс (User Interface, UI) и пользовательские сервисы. Мы расскажем об уже существующих технологиях, влияющих на дизайн пользовательского

уровня модели приложения MSF, об их перспективах и о воздействии Web-технологий на современные методы проектирования приложений. В начале главы обсуждаются проектирование пользовательских сервисов и разные типы пользовательского интерфейса. Вы узнаете о влиянии требований к приложению на выбор методов его реализации, о создании и применении приложений со стандартным пользовательским интерфейсом и об особенностях Web-интерфейсов. В заключение мы рассмотрим некоторые конфигурации и методы, которые разработчики могут применять при объединении пользовательского и прикладного сервисных уровней.

- В главе 8 «Технологии прикладного уровня» обсуждаются вопросы, решение которых необходимо для обеспечения корректной работы бизнес-сервисов приложения, включая использование контекста объекта для управления состоянием, использование явно определенных интерфейсов, функциональный состав компонентов, сохранение состояния в границах транзакции, контроль ошибок и программное управление зашитой. Кроме того, в этой главе описана компонентная объектная модель (Microsoft Component Object Model, COM) и ее применение в бизнес-сервисах приложения.
- Глава 9 «Технологии уровня данных» посвящена особенностям проектирования, связанным с данными, включая реализацию сервисов доступа к данным. Мы познакомим вас с характеристиками различных технологий доступа и рассмотрим способы их использования, обсудим нормализацию данных, вопросы целостности, влияние бизнес-правил на данные и поясним, где эти правила лучше реализовать. Потом мы обсудим технологии доступа к данным, хранящимся в традиционных системах и приложениях управления ресурсами предприятия (Enterprise Resource Planning, ERP), например, SAP R/3 фирмы SAP AG. В заключение мы обсудим базу данных в памяти (In-Memory Database, IMDB), которая позволяет повысить производительность работы с данными.
- Глава 10 «Тестирование и производственный цикл» — первая из нескольких, посвященных созданию среды поддержки жизненного цикла проекта. Мы опишем этот цикл, именуемый производственным, на реальных примерах, а также расскажем о проведении тестирования приложения в контролируемой среде без «нанесения ущерба» производственной среде организации. Мы рассмотрим особенности тестирования, обсудим некоторые методы выполнения тестов и контроля их результатов, а также методы расширения производственной среды посредством добавления дополнительных серверов. И наконец, вы узнаете о классификации отказов и сбоев приложений, об основных проблемах, связанных с ошибками, и о методах их контроля, классификации и устранения.

- Мы начнем главу 11 «Защита приложения» с описания нескольких протоколов, обеспечивающих защиту. Затем расскажем об аутентификации — безопасном и надежном методе идентификации пользователей при регистрации *к* системе или при обращении к ресурсам — и шифровании, обеспечивающем защиту информации от несанкционированного доступа. Мы также обсудим контроль доступа, который необходим для определения прав пользователей системы. и аудит, позволяющий протоколировать работу пользователей и их доступ к ресурсам. В заключение мы познакомим вас с протоколами, журналами событий и распределенными средами.
- Глава 12 «Стадия «Разработка» и ее результаты» посвящена переходу от фазы «Планирование» к фазе «Разработка» на основе функциональных спецификаций и утвержденного проекта приложения. Мы рассмотрим процесс разработки и участие к нем сотрудников проектной группы, детально опишем тестирование, выявление ошибок и стратегию ориентации на отсутствие дефектов, а также рассмотрим компромиссы, необходимые на этой стадии. Кроме того, мы обсудим реализацию многоуровневых приложений, а также окончательный этап стадии «Разработка» и его результаты.
- Глава 13 «Стабилизация продукта» посвящена фазе «Стабилизация» модели MSF, когда проектная группа заканчивает разработку продукта и переходит к завершающим этапам его выпуска. Мы обсудим переход от этапа «Завершение разработки» фазы «Разработка» к этапу «Выпуск продукта» фазы «Стабилизация», в том числе четыре основных промежуточных этапа этого процесса: устранение проблем, синхронизацию результатов, выпуск продукта и его исчерпывающее тестирование. Кроме того, вы познакомитесь с промежуточными этапами, которые предстоит пройти проектной группе на пути к завершающему этапу «Выпуск продукта», и с некоторыми рекомендациями по развертыванию продукта после его выпуска. Мы расскажем о предварительном планировании, пилотном выпуске и его тестировании, сопровождении и устранении проблем.
- Глава 14 «Обсуждение проекта» посвящена методам организации обсуждения завершенных проектов. Мы покажем, что практика рецензирования завершенных проектов позволяет повысить качество управления этими проектами, рассмотрим взаимосвязь обсуждения завершенных проектов с моделью зрелости разработки программного обеспечения (Capability Maturity Model for Software, CMM) и поясним, насколько велико значение рецензирования проектов для выработки оптимальных приемов работы. Кроме того, в этой главе вы познакомитесь с практическими рекомендациями по проведению формального рецензирования, с критериями отбора участников этой процедуры и методами ее организации.

Мы покажем, как организовывать и проводить соответствующие совещания, как собирать и документировать информацию. В завершение мы обсудим создание и роль специальной обзорной группы в больших проектах,

- Приложение «Вопросы и ответы» содержит ответы ко всем вопросам из упражнений и разделов «Закрепление материала» всех глав учебного курса.

Структура книги

- Структура книги отражает последовательность разработки приложения.
- Каждая глава начинается с раздела «В этой главе», где кратко перечислены обсуждаемые темы.
- В каждой главе приведен список дополнительных материалов по теме занятия.
- В разделе «Резюме» обобщены основные вопросы, затронутые в данной главе.
- Каждая глава завершается разделом «Закрепление материала», содержащим контрольные вопросы. Они помогут вам оценить, насколько вы усвоили материал. Ответы на вопросы вы найдете в приложении «Вопросы и ответы».
- В практикуме описана разработка многоуровневого распределенного приложения сотрудниками вымышленной фирмы «Фергюсон и Барделл». Авторы постарались проиллюстрировать эффективность работы в группе. Действующие лиица практикума и само приложение подробно описаны в разделе «Практикум» этой главы.

Соглашения, принятые в учебном курсе

Прежде всего вам надо усвоить терминологию и обозначения, принятые в учебнике, а также разобраться в логике построения книги.

- *Курсивом* выделены новые термины; курсивом же даны цитаты из других книг.
- Имена файлов, папок и каталогов набраны с прописных букв. Кроме особо оговоренных случаев, для ввода имен файлов и каталогов в диалоговом окне или в командной строке можно использовать строчные буквы.
- Расширения имен файлов набраны строчными буквами,
- Аббревиатуры набраны прописными буквами.
- Моноширинным шрифтом набраны фрагменты кода и примеры текста на экране. Названия классов интерфейсов даны **полужирным начертанием**.

- Необязательные элементы синтаксических операторов взяты в квадратные скобки []. Если, например, в синтаксисе команды приведен параметр [*имя_файла*], то это значит, что вы можете (но не обязаны) задать имя файла в качестве параметра команды. При этом достаточно ввести только имя файла — сами скобки набирать не нужно.
- Обязательные элементы синтаксических операторов выделены фигурными скобками {}. Набирать следует только информацию, взятую в скобки — сами скобки набирать не нужно,

Содержимое компакт-диска

Компакт-диск учебного курса содержит электронную версию книги, а также программное обеспечение для сдачи тестового экзамена (он аналогичен реальному экзамену 70-100: *Analyzing Requirements and Defining Solution Architectures*). Воспользовавшись программным обеспечением, разработанным компанией Self-Test Software (STS), вы проверите знания и навыки, которые вам придется продемонстрировать на сертификационном экзамене. Для каждого вопроса приведен ответ со ссылками на материал учебного курса и другие справочные материалы. Полный список сертификационных экзаменов, к которым можно подготовиться средствами программного обеспечения компании Self-Test Software (STS), вы найдете на Web-узле компании по адресу *www.selftestsoftware.com*.

Кроме того, компакт-диск содержит учебное приложение RMS и необходимую документацию (подробнее об этом приложении см. следующий раздел), а также словарь терминов.

Практикум

Концепции, излагаемые в учебном курсе, иллюстрируются в главах практикума. Это рассказ о вымышленной компании «Фергюсон и Барделл», сотрудники которой анализируют требования, проектируют архитектуру и разрабатывают конкретное приложение, применяя концепции, описанные в учебном курсе. Назначение практикума - проиллюстрировать применение концепций процесса разработки на практике и помочь разобраться в том, как использовать его в проектах. Сама компания и ее сотрудники описаны в следующем разделе. Внимательно прочитайте его, прежде чем приступать к изучению учебного курса.

Учебное приложение «Система управления ресурсами» (Resource Management System, RMS), о котором говорится в практикуме, — это реальное многоуровневое распределенное приложение, работающее с двумя хранилищами данных и использующее для управления ресурсами компании основные технологии корпорации Microsoft (Microsoft Visual Basic 6.0, Active Server Pages, Dynamic HTML, Microsoft Outlook 98.

Microsoft Transaction Server 2.0, Microsoft Exchange Server 5.5, Microsoft SQL Server 7.0 и т. д.). Помимо исполняемых модулей и исходных кодов, на компакт-диске вы найдете документацию, которая служит примером документов, создаваемых проектной группой в процессе разработки. Учебное приложение — это система управления ресурсами начального уровня, отслеживающая занятость сотрудников компании и обеспечивающая доступ к информации об их квалификации и работе над проектами. Это приложение создано двумя разработчиками за два месяца, что свидетельствует в пользу как их квалификации, так и технологий, избранных для создания решения.

Требования к аппаратному *и* программному обеспечению для работы с учебным приложением описаны ранее в разделе «С чего начать».

Действующие лица

Дабы заинтересовать читателей, мы решили, припомнив весь свой опыт работы, описать в практикуме почти реальную историю, которая произошла в выдуманной нами компании. Такой способ изложения материала, как мы надеемся, позволит вам лучше понять, как применять в своих проектах методы, описанные в книге.

Сейчас мы познакомим вас с историей и специализацией компании и ее сотрудниками.

Компания

«Фергюсон и Барделл» — чикагская компания, занимающаяся архитектурными и проектными решениями в сфере строительства. В настоящее время годовой оборот компании, основанной двумя ветеранами второй мировой войны в 1948 году, составляет около 230 млн. долларов; в ней работает около 800 сотрудников. Штаб-квартира компании занимает семь этажей в престижном небоскребе в центре Чикаго; у компании есть отделения в Детройте, Милуоки, Цинцинати, Индианаполисе и Луисвилле.

Компания «Фергюсон и Барделл» давно и активно использует информационные технологии, однако ее подход к развитию информационной инфраструктуры не отличался последовательностью. Это привело к чрезмерной «пестроте» форматов хранения информации в региональных отделениях, откуда данные поступают в центральный офис по модему или с курьером. Кроме того, до самого последнего времени в компании использовалось три программы редактирования текста, в том числе давно устаревшее приложение для мэйнфреймов.

В 1998 году совет директоров пришел к выводу, что информационная инфраструктура компании никуда не годится. Один из директоров, хорошо разбирающийся в современных технологиях для эффективной организации бизнеса, связался с компанией-консультан-

том и заказал исследование информационной инфраструктуры «Фергюсон и Барделл». Через два месяца консультанты представили совету директоров свой отчет и рекомендации.

Больше всего споров вызвал совет изменить структуру руководства, отказавшись от поста директора по информационным технологиям (ИТ), подчиненного финансовому директору, и ввести пост исполнительного директора по ИТ в составе совета директоров. Некоторые члены совета, включая финансового директора, настаивали на сохранении статус-кво, но консультанты не сдавались. «Пока директор по ИТ будет выполнять роль еще одного бухгалтера, вы не сможете использовать технологии максимально эффективно. Кроме того, пока за ИТ отвечает человек, не входящий в совет директоров, руководство компании будет недостаточно информировано и не сможет принимать эффективные решения по этим вопросам. Если вы хотите преодолеть информационный хаос, царящий в компании, переведите управление ИТ на уровень совета директоров, а все решения совета принимайте с учетом возможностей и направления развития информационной инфраструктуры».

Новый директор по ИТ, приступивший к исполнению своих обязанностей в октябре 1998 года, представил блестящие рекомендации от своего предыдущего работодателя — местной юридической компании, где он за пять лет прошел путь от администратора сети до директора по ИТ. Прежний директор «Фергюсон и Барделл» по ИТ, который предпочел покинуть компанию, несколько лет пытался создать новую инфраструктуру и связать все региональные подразделения «Фергюсон и Барделл» через Интернет. В результате новому директору понадобилось некоторое время на ознакомление с ситуацией, прежде чем он смог заняться новыми проектами. Первые три месяца он знакомился с сотрудниками, изучал особенности бизнеса и делопроизводства в компании и потихоньку вводил в практику новые методы работы.

В январе он пригласил сертифицированного инструктора, чтобы познакомить своих основных сотрудников с методологией разработки Microsoft Solutions Framework (MSF). Реакция сотрудников была далеко не единодушной — кто-то проявил энтузиазм, кто-то реагировал скептически, а часть сотрудников решила, что новому директору просто нечем заняться. Однако директор не отступал, поскольку был твердо уверен, что лишь четкая методика реализации проектов, основанная на постоянном обмене информацией между ИТ-отделом и бизнес-подразделениями, позволит компаниидобиться положительных результатов.

Группы

Новый директор по ИТ отлично понимал, что для создания производственной архитектуры необходимо создать группу, представляющую разные отделы компании, а для разработки приложений в контексте корпоративной архитектуры понадобятся проектные группы, объединяющие инициативных и опытных сотрудников, которым можно доверить выполнение конкретных работ в рамках проекта. Создание таких групп — непростая задача, поскольку каждый сотрудник привнесет не только свой опыт, но и свою точку зрения.

Для построения корпоративной архитектуры директор по ИТ решил собрать группу в следующем составе.

- **Дэн Шелли, директор по ИТ.** Ему чуть больше сорока, он влюблен в свое дело, информационные технологии и «Чикаго Булле» (не обязательно в указанном порядке). Его любят и уважают. Члены совета директоров ценят его понимание дела, а коллегам в отделе импонирует то, что он прошел все ступени служебной лестницы. Как сказал один из инженеров, «Дэн знает, что такое вызов на работу в три часа ночи». Дэн по-прежнему восхищается потенциалом новых технологий в решении бизнес-проблем, но уже достаточно опытен, чтобы понимать, как трудно их внедрить.
- **Дженни Сакс, сетевой инженер.** Она год назад закончила колледж и с тех пор работает в группе сопровождения сети компании «Фергюсон и Барделл». За это время она приобрела репутацию человека, документирующего каждый шаг. Первое в компании руководство по работе с пользователями появилось, как в сказке, в понедельник утром после того, как она провела всю пятницу в беседах с раздосадованными пользователями. Дженни очень въедлива и никогда не упускает шанс научиться чему-то новому.
- **Кевин Кеннеди, специалист по менеджменту.** Кевин — член комитета по долгосрочному планированию. Весь прошлый год он работал вместе с исполнительным директором и занимался оптимизацией процессов планирования и финансирования. За глаза его называют нахалом, но все понимают, что он быстрыми шагами идет к должности исполнительного директора, которая должна освободиться через четыре года. Он поработал на всех управляющих должностях компании и заработал репутацию человека, делающего свое дело несмотря ни на что,
- **Джозефина (Джо) Браун, заместитель директора по производству.** В прошлом году, когда никто не хотел браться за провальный проект проверки готовности компании к 2000 году, Джо взяла эту неприятную обязанность на себя и отлично справилась с ней. За-

кончив проект на пять месяцев раньше срока, Джо вернулась к своим обязанностям — обеспечению бесперебойной работы компании. Ее последняя работа — создание руководства, которое должно помочь упростить делопроизводство. Все знают, что лучше придерживаться этого руководства — иначе вал гневных электронных писем от заместителя директора им обеспечен.

- **Дик Каплан, аналитик.** В каждой компании есть свой сумасшедший; в «Фергюсон и Барделл» это Дик Каплан. В прежней жизни Дик был профессором философии; за его плечами диссертация и несколько книг. Однако в начале 80-х он решил начать новую жизнь и занялся консультированием. Десять лет назад он пришел в «Фергюсон и Барделл», где до сих пор и работает — аналитиком. Дик — желанный участник всех проектных групп, где ценят общительность, добродушие и поразительную способность находить простые решения сложных проблем.

  Хотя корпоративная архитектура все еще находится в стадии разработки, Дэн решил приступить к новому проекту — созданию системы учета ресурсов — по методике MSF. Для этого он создал проектную группу.

- **Билл Парди, заведующий отделом разработки.** Билл работает в компании с тех пор, как демобилизовался в 1978 году. Он медленно. но верно шел вверх по служебной лестнице, начав с работы на устройстве по подготовке перфокарт, и наконец добрался до разработки приложений баз данных для персональных компьютеров на базе dBase III и Paradox. Он стал заведующим отделом в 1996 году после того, как в течении двух лет руководил группой из 35 разработчиков, «с нуля» создававших для «Фергюсон и Барделл» новый бухгалтерский пакет. Хотя проект затянулся на полгода и обошелся на 40% дороже, чем планировалось, все были уверены, что, если бы не Билл, все было бы гораздо хуже.
- **Джейн Клэйтон, главный бухгалтер.** Пожалуй, именно Джейн лучше всех знает, как работает компания. Она пришла сюда десять лет назад простым бухгалтером, а четыре года назад заняла должность главного. Джейн любит хороших работников и всегда защищает их от любых проблем, будь то другие сотрудники, политика компании или технология. Она не специалист по компьютерам, но работает с ними достаточно долго, чтобы хорошо понимать, что полезно в бухгалтерии, а что нет.
- Тим **О'Брайан, администратор сети.** Тим выглядит на десять лет моложе своих 28. Его интересует все, что связано с компьютерами, но он выбрал для себя технологии Microsoft и получил сертификат MCSE (помимо диплома инженера, который ему выдали в

Северо-западном университете). Тим — компанейский, всегда улыбающийся парень с отличным чувством юмора. Его безотказность в работе и неприязнь к разного рода собраниям вошли в легенду, как и знание информационной инфраструктуры компании и умение быстро и эффективно устранять любые проблемы.

- **Мэри-Лу Морис, инструктор.** Независимый инструктор из Чикаго и любимица всех сотрудников «Фергюсон и Барделл». Она читает лекции в разных компаниях и в центрах подготовки по всему северо-востоку США. Ее курсы неоднократно слушали и сотрудники Джейн, причем за это время Мэри-Лу с Джейн стали подругами. Именно Джейн порекомендовала ее Дэну, узнав, что в проектной группе требуется инструктор.
- **Марта Вольф-Хелен, инженер.** Марта — самый молодой член проектной группы. Она только-только пришла в компанию, однако все уже оценили ее ум и умение работать. Она молчалива (иногда это принимают за признак слабости, пока не убедятся в обратном), однако лишь до тех пор, пока со всеми не познакомится, — после этого все замечают ее остроумие.

Материалы для подготовки к экзамену

В приведенных ниже таблицах перечислены темы сертификационного экзамена 70-100: *Analyzing Requirements and Defining Solution Architectures* с указанием глав учебного курса, где обсуждаются соответствующие вопросы.

Анализ требований

Тема	Главы
Анализ предметной области, включая существующие решения, возможные изменения среды, время жизни решения, возможные компромиссы между затратами времени и средств и характеристиками решения	1, 2, 4, 5
Анализ бизнес-требований	
• выявление требований	5
• определение типа решения (например, система сообщений или коммуникационное приложение)	5
• выявление требований к качеству	5
• снижение совокупной стоимости владения	3, 5, 6, 13
• обеспечение быстрого возврата инвестиций в проект	3. 5, 6, 13
• анализ имеющихся платформ и инфраструктуры	I

(продолжение)

Тема	Главы
• учет инфраструктуры и платформ в проектировании решения	6
• выявление и учет необходимости перехода на новые технологии	I
• выявление физических требований, включая требования к инфраструктуре	6–9, 11
• планирование среды, необходимой для работы приложения, включая аппаратные платформы, операционные системы и вопросы сопровождения	6–9, 11
• выявление ограничений, вызванных организационной структурой, включая финансовую ситуацию, политику компании, уровень технической подготовки персонала и необходимость в дополнительной подготовке	1, 3, 5
• разработка графика реализации решения	6
• анализ пользовательской аудитории	5
Анализ требований к защите	
• анализ ролей и групп пользователей решения	11
• ограничения, накладываемые существующей средой	11
• требования к отказоустойчивости	10
• требования в отношении сопровождения решения	11
• распространение базы данных системы зашиты	11
• требования к системе защиты	5, 11
• планирование аудита	1]
• анализ существующих механизмов зашиты	11
Анализ требований к производительности, включая: необходимую скорость обработки транзакций, полосу пропускания, число обслуживаемых пользователей, соответствие стандартам, соотношение между пиковой и средней нагрузкой, необходимое и нынешнее время отклика, ожидаемые проблемы	2, 10
Анализ требований к сопровождению, включая: распространенность приложения, методы распространения, ожидаемые затраты на сопровождение, местонахождение и необходимый уровень квалификации службы поддержки, учет существующих соглашений с внешними службами технической поддержки	3, 10, 13

(продолжение)

Анализ требований к расширению, включая: учет расширения функциональных возможностей продукта	10, 3, 7
Анализ требований к доступности, включая: планируемый распорядок использования, степень доступности и последствия простоя	10
Анализ человеческого фактора, включая: пользовательскую аудиторию, необходимость локализации, средства доступа для инвалидов, наличие мобильных пользователей, справочную службу, требования по подготовке пользователей и ограничения, накладываемые физической средой	3, 7
Анализ требований, накладываемых интеграцией с существующими приложениями, включая: традиционные приложения, формат и местонахождение накопленных данных, совместимость с существующими приложениями, необходимость преобразования и более **эффективного** использования данных	**9**
Анализ бизнес-процессов и накладываемых ими ограничений, включая: юридические ограничения, практику ведения дел, структуру организации, управление процессами, бюджет, методики развертывания решений и подготовки пользователей, требования контроля качества и пожелания пользователей	1, **3, 4**
Анализ требований к масштабированию, включая: возможное расширение аудитории, рост организации, увеличение объема данных и особенности использования	1, 3-5

Выбор архитектуры решения

Тема	Главы
Выбор типа решения (одноуровневое, двухуровневое или N-уровневое) в конкретной ситуации	**1, 2**
Выбор технологий для реализации решения с учетом следующих вопросов: технологические стандарты (EDI, Интернет, OSI. COMTI, POSIX), технологии, требующие лицензирования, нынешняя и планируемая технологическая инфраструктура компании, выбор средств разработки	2, 7–9, 11
Выбор архитектуры хранилища данных с учетом следующих вопросов: объем, число транзакций в единицу времени, число подключений или сеансов, требования бизнес-среды, необходимость расширения, число пользователей и тип базы данных	1, 2, 5, 6, 9

(продолжение)

Тема	Главы
Проверка реализуемости выбранной архитектуры, включая: демонстрацию соответствия бизнес-требованиям, демонстрацию соответствия сценариям использования, демонстрацию учета технологических ограничений, учет последствий невозможности реализации отдельных требований	5, 6, 10
Выбор стратегии разработки	**3.** 6, 13

Концептуальное и логическое проектирование

Тема	Главы
Разработка концепции, учитывающей различные сценарии, контекст, бизнес-процессы, последовательность выполнения отдельных задач и модели среды.	
Типы решений: однооконные, многооконные, консольные и диалоговые приложения для настольных систем; двухуровневые, клиент-серверные и Web-приложения; N-уровневые приложения и приложения для коллективной работы	5, 6
Применение принципов модульного проектирования к концепции для **выделения компонентов и** сервисов, **образующих логический проект**	**5, 6**
Учет **бизнес-требований при проектировании объектов**	**5, 6**
Оценка влияния логического проекта на **производительность, простоту сопровождения, возможность расширения, масштабируемость,** доступность **и защиту**	10, 11

Проектирование моделей данных

Тема	Главы
Организация данных с применением правил **нормализации**	**9**
Описание связей между элементами	**9**
Выбор внешних ключей для обеспечения целостности элементов и ссылочной целостности	**9**
Учет бизнес-требований и ограничений в модели **данных**	10, 11
Выбор необходимого уровня денормализации	**9**
Разработка базы данных с учетом стандартов и практики разработки БД	**9**

Разработка пользовательского интерфейса и сервисов пользовательского уровня

Тема	Главы
Выбор методов навигации	**6, 7**
Выявление процедур проверки вводимой информации, которые надо интегрировать в пользовательский интерфейс	**9**
Оценка и выбор архитектуры справочной системы (строка состояния, советы и справочные файлы)	7
Создание прототипа пользовательского интерфейса с учетом бизнес-требований, правил проектирования интерфейса и стандартов, принятых в организации	1, **4, 5**
Анализ необходимости использования элементов управления на базе меню	7
Анализ необходимости использования клавиатурных сокращений для ускорения выполнения различных функций	1, 4, 5

Физическое проектирование

Тема	Главы
Оценка влияния физического проекта на производительность, простоту сопровождения, возможность расширения, масштабируемость, доступность и защиту	**6, 9-12**
Оценка необходимости объектной инкапсуляции доступа к БД	**8, 9**
Проектирование свойств, методов и событий для конкретных компонентов	**6, 8**

Программа сертификации специалистов Microsoft

Программа сертификации специалистов Microsoft (Microsoft Certified Professional, MCP) — отличная возможность подтвердить знание современных технологий и программных продуктов этой фирмы. Компания Microsoft, лидер в области сертификации, разработала современные методы тестирования. Они убедительно подтвердят вашу квалификацию разработчика или специалиста по реализации решений на основе технологий и программных продуктов Microsoft. Профессионалы, сертифицированные компанией Microsoft, квалифицируются как эксперты и пользуются огромным спросом на рынке труда.

В апреле 2002 г. вниманию соискателей предлагались следующие типы сертификации.

- *Сертифицированные специалисты по продуктам Microsoft* (Microsoft Certified Professional) — требует досконального знания как мини-

мум одной операционной системы Microsoft. Кандидаты могут слать дополнительные экзамены, что подтвердит их право на работу с продуктами Microsoft BackOffice, инструментами или прикладными программами.

- *Сертифицированные специалисты по продуктам Microsoft и Интернету* (Microsoft Certified Professional + Internet, MCP + Internet) - планирование систем защиты, установка и конфигурирование серверных продуктов, управление ресурсами сервера, расширение возможностей сервера средствами сценариев интерфейса общего шлюза (Common Gateway Interface, CGI) и интерфейса прикладного программирования сервера Интернета (Internet Server Application Programming Interface, ISAPI), мониторинг работы сервера, анализ его производительности и устранение неисправностей.
- *Сертифицированные специалисты по продуктам Microsoft и проектированию Web-узлов* (Microsoft Certified Professional + Site Building, MCP + Site Building) — проектирование, создание, управление и поддержка работы Web-узлов с использованием технологий и продуктов корпорации Microsoft.
- *Сертифицированные системные администраторы Microsoft* (Microsoft Certified Systems Administrator, MCSA) — реализация, управление и устранение неполадок в существующих системах на основе Windows 2000, включая Windows .NET Server.
- *Сертифицированные разработчики Microsoft* (Microsoft Certified Solution Developer, MCSD) — разработка и создание прикладных приложений с применением инструментов, технологий и платформ Microsoft, а также архитектуры Windows DNA.
- *Сертифицированные системные инженеры Microsoft* (Microsoft Certified Systems Engineer, MCSE) — проектирование и реализация инфраструктуры для бизнес-решений на базе платформы Windows 2000 и серверных продуктов Microsoft.
- *Сертифицированные администраторы баз данных Microsoft* (Microsoft Certified Database Administrator, MSDBA) — разработка физической структуры баз данных, проектирование логических моделей БД, создание БД и сервисов данных средствами Transact-SQL, администрирование и сопровождение БД, настройка и сопровождение системы защиты, установка и конфигурирование Microsoft SQL Server.
- *Сертифицированные инструкторы Microsoft* (Microsoft Certified Trainer, MCT) — теоретическая и практическая подготовка для ведения соответствующих курсов в авторизованных учебных центрах Microsoft.

Набор доступных сертификации постоянно обновляется в соответствии с требованиями компьютерной индустрии. Для получения самой свежей информации о них и о требованиях, предъявляемых к соискателям, обращайтесь на Web-узел корпорации Microsoft по адресу *http://www.microsoft.com/rus/mcp*.

Достоинства сертификации

Программа сертификации Microsoft — один из самых строгих и полных тестов оценки знаний и навыков в области проектирования, разработки и сопровождения ПО. Сертифицированным специалистом Microsoft становится лишь тот, кто продемонстрировал умение решать конкретные задачи, применяя продукты компании. Программа тестирования позволяет не только оценить квалификацию специалиста, но и служит ориентиром для всех, кто стремится достичь современного уровня знаний в этой области.

Индивидуальная сертификация

Вот какие преимущества дает звание Microsoft Certified Professional:

- официальное признание корпорацией Microsoft вашего высокого профессионального уровня в использовании и поддержке программных продуктов фирмы, а также в разработке решений на их основе;
- доступ к новейшей технической информации непосредственно от Microsoft; в зависимости от выбранной программы сертификации, вы получите подписку на различные издания Microsoft, содержащие ценную техническую информацию о продуктах и технологиях компании;
- эмблемы, соответствующие выбранной вами программе сертификации, а также другие материалы, которые позволят вам проинформировать своих коллег и клиентов о вашем статусе сертифицированного специалиста;
- доступ к специальным форумам MSNTM, The Microsoft Network и CompuServe, которые позволяют сертифицированным специалистам контактировать с Microsoft и друг с другом;
- приглашения на конференции, практикумы и специальные мероприятия Microsoft, предназначенные для специалистов;
- сертификат «Microsoft Certified Professional».

Кроме того, в зависимости от типа сертификации и страны, сертифицированные специалисты получают:

- годичную подписку на ежемесячно распространяемые компакт-диски Microsoft TechNet (Technical Information Network).

- годичную подписку на программу бета-тестирования продуктов Microsoft; в результате вы бесплатно получите до 12 компакт-дисков с бета-версиями новейших программных продуктов.

Организованная сертификация

Сертификация позволяет организациям извлечь максимум прибыли из затрат на технологии Microsoft. Исследования показывают, что сертификация сотрудников по программам Microsoft:

- очень быстро окупается за счет стандартизации требований к обучению специалистов и методов оценки их квалификации;
- позволяет увеличить эффективность обслуживания клиентов, повысить производительность и снизить расходы на сопровождение;
- обеспечивает надежные критерии для найма специалистов и их продвижения по службе;
- предоставляет методы оценки эффективности труда персонала;
- обеспечивает гибкие методы переподготовки сотрудников для обучения новым технологиям;
- позволяет оценить партнеров — сторонние фирмы.

Дополнительную информацию о том, какую пользу ваша компания извлечет из сертификации, см. на странице *http://www.microsoft. com/rus/mcp/mcp_benef.html.*

Техническая поддержка

Мы постарались сделать все от нас зависящее, чтобы и сам учебный курс, и прилагаемый к нему компакт-диск не содержали ошибок. Издательство Microsoft Press публикует постоянно обновляемый список исправлений и дополнений к своим книгам по адресу: http://mspress.microsoft.com/support.

Если у вас все же возникнет вопрос или комментарий, обращайтесь в издательство Microsoft Press по одному из указанных ниже адресов.

- Электронная почта: tkinput@microsoft.com
- Обычная почта:

  Microsoft Press
  Attn: Analyzing Requirements & Defining Solutions Architecture
  Training Editor
  One Microsoft Way
  Redmond, WA 98052-6399

Указанные адреса не предназначены для поддержки программных продуктов, упомянутых в учебном курсе. Ее вы найдете на узле http:// suppon.microsoft.com или обратившись в службу технической поддер-

жки компании Microsoft по телефону (800) 936-3500, За пределами США обращайтесь в местное представительство компании Microsoft.

Об авторах

- Тим **Лэндгрейв** (Tim Landgrave)

  В 1991 году Тим Лэндгрейв основал компанию KiZAN, специализирующуюся на разработке клиент-серверных приложений на базе технологий Microsoft. С тех пор Тим занимается проектированием, разработкой и развертыванием программных систем на базе ключевых технологий Microsoft. Тим — не только исполнительный директор компании KiZAN, но и директор регионального отделения Microsoft. В этом качестве он организует различные мероприятия, например, встречи разработчиков, и пропагандирует платформы Microsoft как основу разработки надежных многоуровневых приложений. Прежде Тим работал директором по технологиям компании Cobb Group, где основал несколько журналов для разработчиков. Кроме того, он занимал пост главного редактора таких журналов, как *Inside Visual Basic*, *Inside Visual* C++ и *Microsoft Networking Journal.*

- Брюс **Мэйплс** (Bruce Maples)

  В 1984 у Брюса появился первый Apple II, на котором он учился копаться в операционной системе и понемножечку программировать. С тех пор он успел многое: писал приложения на множестве языков от dBase до Visual Basic, читал лекции, консультировал, стал автором нескольких книг. Тим живет в Луисвилле, шт. Кентукки, с женой Ниной и сыновьями Гриффином и Бенджамином, Помимо увлечения компьютерами, Брюс активно участвует в делах местной церковной общины.

- Скотт Ф. **Уилсон** (Scott F. Wilson)

  Скотт помог превратить KiZAN в успешную программистскую компанию, специализирующуюся на разработке многоуровневых приложений и сетевых сервисов на базе технологий Microsoft. Скотт занимался проектированием производственных архитектур, инфраструктуры сетей и приложений масштаба предприятия для клиентов KiZAN. Последние три года он помогает корпоративным заказчикам создавать и развертывать Web-приложения на платформе MCIS и Microsoft Site Server Commerce. Скотт — не только директор по технологиям в компании KiZAN, но еше и сертифицированный инструктор и системный инженер Microsoft (с 1995 года). С ним можно связаться по адресу scottw@kizan.com.

Благодарности

Я признателен моей семье, сотрудникам компании KiZAN и моим соавторам Брюсу и Тиму за постоянную поддержку. Спасибо Эрику, Венди и Вики из Microsoft Press за помощь в издании этой книги, Коду Фергюсону за вдумчивые комментарии, а группе MSF и Мэри Киртлэнд — за возможность воспользоваться их материалом.

Эта книга не появилась бы на свет без группы редакторов издательства OTSI. Особая благодарность Джойсу Коксу и Джоан Лэмберт. Если что-то в этой книге правильно, благодарите их, а все что неверно — на моей совести.

Я бесконечно благодарен моей жене Сэнди за постоянную поддержку и любовь. Ты не даешь моей жизни рассыпаться на части, пока я совершаю разнообразные безумства, например пишу книги — кстати, с этой я наконец справился.

— *Скотт Ф. Уилсон*

Прежде всего, спасибо Скотту — работа с ним доставила мне огромное удовольствие. Писать большую книгу — чаще изматывающий труд, чем удовольствие, однако благодаря Скотту удовольствия было больше. Я хочу поблагодарить сотрудников KiZAN за их безотказную помощь, профессионализм и за то, что с ними всегда приятно иметь дело. Спасибо Алану Макгаффи, который научил меня не быть пассивным.

Кроме того, я признателен всем, кто работает в кафе Steak-and-Shake (хотя они об этом и не узнают) — без них материал, над которым я работал поздним вечером, остался бы неоконченным. И наконец, спасибо двум самым главным персонажам моей жизни: Господу Богу и моей жене Нине. Благодаря им я счастлив, как никто другой.

— *Брюс Мэйплс*

Спасибо сотрудникам KiZAN — они всегда делают работу «на отлично». Спасибо моей жене Лауре и нашим дочерям Тэйлор и Ханне за любовь, терпение и поддержку.

— *Тим Лэндгрейв*

Часть I. Методология

ГЛАВА 1

Производственная архитектура

В этой главе

Решение современных задач из области информационных технологий (ИТ) требует применения принципов и методов, которые мы совокупно называем «Производственной архитектурой». В этой главе рассказывается, почему разработку приложений и развитие инфраструктуры надо ориентировать на базовые концепции уровня предприятия. Прежде всего мы рассмотрим концепции разработки систем с позиции «приоритета архитектуры». Далее мы опишем выработанные компанией Microsoft принципы создания приложений Microsoft Solutions Framework (MSF), в которых архитектура рассматривается с четырех точек зрения (назовем их *перспективами):* бизнеса, приложений, информации и технологии. При использовании метода MSF решения получаются комплексными, итерационными, работоспособными, с четко определенными приоритетами. Наконец, мы обсудим организацию процесса разработки производственной архитектуры и покажем, как создавать системы и приложения, когда производственная архитектура еще не сформирована.

При работе над этой главой мы использовали свой собственный опыт проектирования архитектуры приложений и их реализации, а также материалы MSF.

Изучив материал этой главы, вы сможете:

- ✓ охарактеризовать достоинства разработки проектов с позиций приоритета архитектуры;
- ✓ продемонстрировать значимость разработки архитектуры для успешного создания приложений;
- ✓ описать четыре перспективы — составные части модели производственной архитектуры MSF;
- ✓ описать структуру каждой перспективы;
- ✓ описать преимущества планового полхода к созданию производственной архитектуры.

Что такое архитектура

В контексте этой книги архитектурой называется скоординированный, единый технологический план. Применительно к ИТ архитектура позволяет сосредоточить внимание на целостности технологического процесса и на комплексном подходе, четко направленном на достижение конечных целей. Архитектор информационной системы — это человек, проектирующий ее и руководящий составлением технологического плана, цельного, ясного и последовательного. Другими словами, архитектор и разрабатываемая им архитектура определяют направление создания и развития информационной системы. При этом упор делается на главные элементы, определяемые приоритетом архитектуры, что позволяет достичь результатов с максимальной эффективностью и минимумом проблем.

Отличный пример важности выбора направления движения — диалог Алисы и Чеширского кота из книги Льюиса Кэрролла «Алиса в стране чудес»*:

«— Скажите пожалуйста, куда мне отсюда идти? — спросила Алиса.

— А куда ты хочешь попасть? — ответил Кот.

– Мне все равно... — сказала Алиса.

– Тогда все равно, куда и идти, — заметил Кот».

Учитывать приоритет архитектуры требуется практически в любом виде деятельности, в частности, при планировании, создании и сопровождении какого-либо продукта. Ведь множество компаний, подобно Алисе, стараются достичь результата, не имея четкого представления о том, что он из себя представляет.

* Цитируется по изданию: Л. Кэрролл. «Алиса в стране чудес» (перевод и подготовка текста Н.М. Демуровой), М.: Наука, 1990. — *Прим. ред.*

Метод, который мы определили как «приоритет архитектуры», основан на следующем; прежде чем начать любой проект, надо ответить на простые вопросы.

- Как мы пришли к существующему положению?
- Знаем ли мы, куда движемся?
- Зачем мы туда идем?
- Какие задачи нужно решить, чтобы достичь желаемого?
- В каком порядке следует решать эти задачи?
- Как мы поймем, что цель достигнута?
- Что еще следует принять во внимание?

Согласно концепции «приоритета архитектуры», на эти вопросы нужно ответить еще до начала проекта, а затем руководствоваться полученными ответами в течение всей работы над ним. Кроме того, этот метод поможет вам достичь желаемого баланса между:

- целями и требованиями, определяемыми бизнесом;
- важнейшими проектными решениями;
- затратами человеческих ресурсов;
- финансовыми затратами организации.

От концепции к практике

К сожалению, многие организации поддерживают концепцию приоритета архитектуры и планирования проектов лишь на словах. Они просто «плывут по течению», надеясь, что, когда работа будет закончена, все прояснится само собой. Многие руководители полагают, что пока идет планирование, работа стоит.

Единственный шанс вырваться из этого порочного круга — на деле применить метод, ориентированный на приоритет архитектуры. Для этого надо понять простую вещь: затраты времени на разработку плана в конечном итоге экономят время, затраченное на весь проект.

Замечание Это не означает, что работу нельзя начать до тех пор, пока полностью не определена архитектура. В этой и следующих главах мы продемонстрируем обоснованность и реалистичность *планирования во время разработки*. Другими словами, планирование, разработка и сопровождение — итерационные шаги, часто выполняемые параллельно.

Поддержка архитектурно-ориентированного метода означает, что все три составляющие ИТ-проекта — планирование, создание и сопровождение продукта — базируются на ясно и четко определенной архитектуре высокого уровня, что эта архитектура выработана до на-

чала разработки, и наконец, что именно эта архитектура определяет направление работы. Прежде чем применять подобный метод к конкретным приложениям, необходимо полностью определить архитектуру на уровне организации.

Когда планирование и организацию работ начинают проводить в соответствии с этим методом, сотрудники информационной службы совершенно естественно полагают, что они способны начать разработку и полностью определить производственную архитектуру организации. Энтузиазм в этот период очень высок. Затем, по мере осознания объема возникающих проблем, числа факторов и связей, которые необходимо учитывать, настроение изменяется и перспективы проекта кажутся все более сомнительными. Наконец, возникает вопрос: «Действительно ли мы получим от этого какую-то вы году? А учитывая объем работы, стоит ли пытаться?»

В этой книге мы попробуем ответить главным образом на первый вопрос, поэтому сейчас коротко остановимся на втором. Почему столь занятые и квалифицированные люди, как профессионалы в области информационных систем, должны тратить время на разработку производственной архитектуры организации и выполнение связанных с этим работ? Ответ очень прост: любое предприятие обладает в большей или меньшей степени сложившейся архитектурой. Организация может оценивать и планировать ее, а может использовать стихийно сложившуюся архитектуру, которая совсем не обязательно соответствует задачам бизнеса. Только занявшись анализом создавшегося положения и разработкой продуманной производственной архитектуры, можно взять этого монстра под контроль.

Задачи информационной инфраструктуры

Давайте на минуту отвлечемся и рассмотрим проблемы, стоящие перед современным бизнесом. Ситуация на внутренних и внешних рынках стремительно меняется, что требует постоянного предложения разнообразных и сложных новых продуктов и услуг. Руководство ожидает, что цикл разработки новых продуктов будет сокращаться во имя ускорения их появления на рынке. Чтобы сохранить конкурентоспособность, организации вынуждены осваивать постоянно изменяющиеся технологии. Справляться с этой сложной задачей помогает следующее (рис. 1.1):

- рационализация производственных процессов;
- совершенствование структуры организации;
- внедрение новых технологий в сжатые сроки.

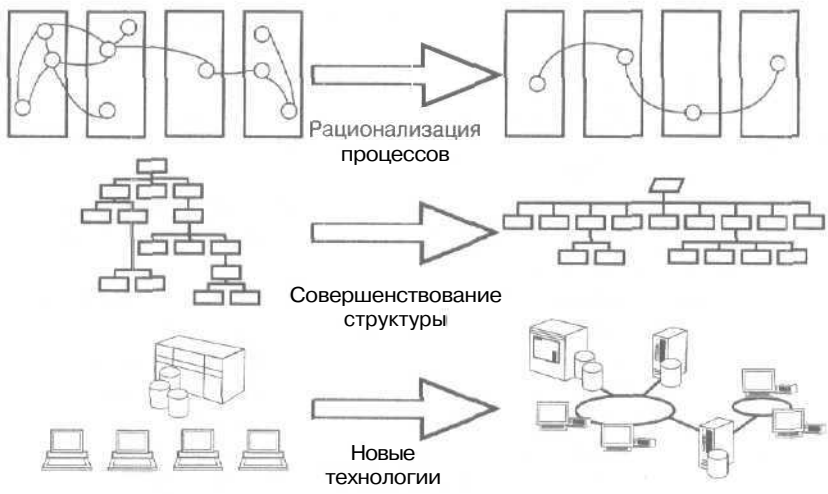


Рис. 1.1. Развитие организации

Поскольку потребность в изменениях постоянно растет, от информационных систем требуется гибкость, достаточная, чтобы поддерживать эти изменения. Рассмотрим, чего ожидает типичная организация от своей информационной инфраструктуры в быстро меняющихся условиях:

- **двойственности** — инфраструктура должна удовлетворять нужды клиентов и в то же время поддерживать задачи производства;
- **гибкости** — необходимость приспосабливаться к постоянно изменяющемуся технологическому пейзажу и в то же время не нарушать любимого правила руководства: «Лучше, быстрее, дешевле — и срочно!».

Чтобы организация выдержала конкуренцию, информационная служба должна уметь меняться в соответствии с запросами производства и клиентов. Кроме того, она должна быть готова быстро создавать новые приложения и быстро модернизировать их. Ключ к успеху — применение гибких, постоянно совершенствуемых методов, позволяющих создавать повторно используемый код и компоненты.

Большинству организаций трудно быстро адаптироваться к нуждам производства, сохраняя относительно низкий уровень финансовых затрат. Какие бы способы ни использовала организация в конкурентной борьбе — повышение уровня обслуживания и скорости реакции на запросы клиентов или выпуск более дешевых продуктов без потери качества — роль информационных технологий огромна. Для решения таких ключевых вопросов, как выпуск продукции, анализ

рынков, финансы и продажи требуется ИТ, как основа принятия решений. Реализация этих функций также требует использования ИТ для нормальной интеграции и кооперации между различными подразделениями компании и выполняемыми задачами,

Кроме того, современная ситуация предъявляет чрезвычайно сложные требования к приложениям.

- **Продвижение товаров** — специальные программы способны по компьютерным сетям мгновенно распространять информацию о продукте буквально по всему миру.
- **Возможность привлечения огромного числа клиентов** — пользователями таких приложений потенциально являются миллионы человек. Если не использовать возможности, предоставляемые современными ИТ, эти потребители не станут клиентами вашей компании.
- **Ограниченные возможности связи** — потребители часто работают с приложениями лишь временно. Не исключено также, что их возможности ограничены пропускной способностью соединения. Например, сотрудники могут использовать портативные компьютеры, которые не связаны постоянно с корпоративной сетью, а клиенты обращаются к приложениям компании через Интернет с помощью не слишком быстрого модема.
- **Возможности хранения** — необходимые для работы приложения данные находятся на нескольких компьютерах. Возможно, что эти компьютеры расположены не на одной территории или не доступны постоянно.
- **Аппаратные ограничения** — пользователи зачастую применяют весьма разнообразные программные и аппаратные средства, например, работают на разных типах компьютеров с различными возможностями или хранят данные в разных базах данных. Новому программному обеспечению, вероятно, придется взаимодействовать с существующими приложениями, работающими на различных платформах.

Учтите совокупность этих факторов, необходимость взаимодействия систем, и вы согласитесь, что разработка приложений, учитывающих все эти обстоятельства, действительно требует серьезных усилий. Сочетание организационных и технологических проблем с уже работающим в организации программным обеспечением ставит руководителей информационных служб в очень сложное положение, Подводя итог, мы можем сформулировать три правила для руководителей ИТ-подразделений:

- **следовать целям бизнеса** — необходимо тесно увязывать ИТ с целями бизнеса;

- **контролировать расходы** — следует обосновывать малейшие отклонения от запланированного уровня расходов и тщательно планировать будущие инвестиции;
- **чувствовать и реагировать** — необходимо совершенствовать связи между подразделениями внутри и за пределами организации, чтобы сделать работу с заказчиками, поставщиками и партнерами более эффективной.

Производственная архитектура и задачи ИТ

Внедрение современных технологий может как способствовать, так и препятствовать адаптации организации к изменению конъюнктуры рынка. Современные информационные решения должны полностью отвечать требованиям бизнеса — с одной стороны, быть достаточно гибкими, чтобы легко взаимодействовать как с новыми, так и с устаревшими технологиями, а с другой — не подвергать риску бизнес-процессы в уже сложившейся производственной архитектуре. С учетом этих соображений от метода создания производственной архитектуры требуется:

- считать приоритетным учет нужд бизнеса;
- предусматривать такие технические решения, которые делают простые веши легкими, а сложные — возможными и выгодными;
- обеспечить достаточную гибкость при адаптации к неизбежной эволюции технологии и организации производства.

Ключевым моментом в достижении этих целей является создание комплексной высокоуровневой производственной архитектуры. Именно она определяет, как будет происходить анализ существующего состояния и пути его совершенствования. Этот процесс представляет собой разработку целой серии проектов, цель которых — полностью обеспечить переход информационной инфраструктуры и программных продуктов к намеченному состоянию. Таким образом вы создадите прочную основу для согласования стратегии развития ИТ и каждодневной работы по ее реализации со стратегическими целями компании.

Задача производственной архитектуры

Несколько ранее мы заметили, что для нового проекта важнее всего определить направление или цель. Цель разработки архитектуры можно сформулировать так;

> *«Выработать логически связанный план рутинных работ и скоординированных проектов, управляющих развитием структуры информационных систем и приложений организации. В плане должен определяться последовательный переход от настоящего к намеченному будущему состоянию на основе текущих и перспективных задан и процессов».*

Попытаемся пояснить это утверждение.

- **Логически связанный** — все части общего плана рассматриваются вместе, они должны быть логически связаны.
- **Рутинные работы и скоординированные проекты** — задачи архитектуры касаются как каждодневной деятельности, так и самостоятельных проектов.
- **Развитие сложившейся структуры информационных систем и приложений организации для достижения намеченного будущего состояния** — архитектура должна не только описывать текущую ситуацию, но и предлагать перспективную концепцию. Очень важно, когда она определяет ясный и оптимальный путь от текущего состояния к достижению долгосрочных целей.
- **Текущие и перспективные задачи и процессы** — проект будет бесполезным, если в нем не учитываются как текущее положение дел, так и перспективы развития бизнеса и производственных процессов. С другой стороны, бизнес-планы часто формируются под влиянием достижений ИТ, например, развитие доступа в Интернет заставило многие компании срочно создавать подразделения электронной коммерции.

Помните об этих моментах, когда мы будем рассматривать детали производственной архитектуры. Их необходимо учитывать и при оценке степени завершенности проекта создания производственной архитектуры.

Принципы разработки приложений MSF

Принципы разработки приложений MSF — это набор моделей, принципов и методов, которые помогают организации более эффективно создавать и использовать ИТ для решения проблем бизнеса. Обеспечивая ощутимый прогресс и четкое руководство, MSF позволяет сделать приложение гибким и способным реагировать на изменяющиеся потребности организации. Ядро этой системы составляют шесть основных моделей:

- модель производственной архитектуры;
- модель проектной группы;
- модель процесса разработки ПО;
- модель управления рисками;
- модель процесса проектирования;
- модель приложения.

Ниже мы кратко опишем каждую из этих моделей. В следующих главах мы покажем, как применять их на практике в конкретных проектах разработки программного обеспечения.

Модель производственной архитектуры

Эта модель предлагает набор принципов, обеспечивающих быстрое создание производственной архитектуры посредством выпуска версий, При этом информационные технологии приводятся в соответствие с требованиями бизнеса с четырех точек зрения: бизнеса, приложения, информации и технологии. Использование этой модели позволяет сократить затраты времени на разработку производственной архитектуры.

Модель проектной группы

Эта модель относится к группе, работающей над проектом: описывает роли, обязанности каждого участника, распределение ответственности и порядок работы. Гибкость позволяет привести модель в соответствие с характером проекта, размером группы и квалификацией участников. Использование этой модели и ее основных принципов помогает сформировать неравнодушную, энергичную и эффективную команду.

Модель процесса разработки ПО

Эта модель описывает организационную структуру процесса разработки и руководство им в течение всего времени выполнения проекта. Отличительные особенности модели — поэтапность, итеративность и гибкость. Она описывает фазы, этапы, виды деятельности и результаты процесса разработки приложения и их связь с моделью проектной группы MSF. Использование этой модели обеспечивает контроль за ходом разработки проекта, минимизацию рисков, повышение качества и сокращение сроков выполнения проекта.

Модель управления рисками

Эта модель предлагает организованный путь активного управления рисками проекта. Она описывает порядок и условия реализации упреждающих решений и мер для постоянного выявления потенциальных проблем, позволяет обнаружить наиболее существенные риски и реализовать стратегии их устранения. Использование этой модели и ее основных принципов помогает команде сосредоточиться на наиболее важных моментах, принимать верные решения и лучше подготовиться к тому времени, когда будущее откроет свои тайны.

Модель процесса проектирования

Эта модель описывает трехфазный, ориентированный на конечного пользователя, непрерывный процесс разработки, характеризующийся параллельным и итерационным выполнением проекта и таким образом способствующий его эффективности и гибкости. Три фазы разработки — концептуальное, логическое и физическое проектирование — реализуют точку зрения на проект трех аудиторий: конечных

пользователей, проектной группы и разработчиков, Продвижение от концептуального проекта к физическому превращает набор сценариев использования в совокупность компонентов и сервисов, образующих приложение, реализующее требования заказчика и пользователей. Таким образом, приложение разрабатывается не ради демонстрации технологических возможностей, а для решения насущных проблем бизнеса и пользователей.

Модель приложения

Эта модель реализует логичный. трехуровневый, ориентированный на сервисы метод проектирования и разработки программного обеспечения. Применение пользовательских сервисов, бизнес-сервисов и сервисов данных позволяет реализовать параллельную разработку, обеспечивает оптимальное использование технологии, облегчает эксплуатацию и сопровождение и обеспечивает максимальную гибкость развертывания, поскольку сервисы, составляющие приложение, могут располагаться и на единственном персональном компьютере, и на различных серверах и клиентах, установленных в разных странах,

MSF в этой книге

В этой книге мы используем основные концепции MSF как основу для обсуждения создания производственной архитектуры и разработки приложений. Модели MSF и их основные принципы разработаны группой MSF компании Microsoft. В этой книге мы в значительной степени использовали материалы, предоставленные этой группой. Однако, полагая, что лучше не ограничивать себя только ими, мы обратились к другим опубликованным источникам, а также учли собственный опыт разработки приложений. Это позволило нам достаточно полно описать эти концепции и показать, как они применяются на практике. При рассказе о концепциях MSF мы старались отмечать информацию, почерпнутую из других источников, во всех случаях, когда это не отвлекает внимания читателя.

Модель производственной архитектуры MSF

Как сегодня функционирует предприятие? Многие руководители способны ответить на этот вопрос лишь приблизительно. Модель производственной архитектуры MSF — это работоспособная система, в контексте которой возможен анализ существующей инфраструктуры и разработка перспективной архитектуры. Кроме того, эта модель закладывает фундамент для реализации бизнес-решений с использованием современных технологий. Она позволяет системам не только реализовать свои функциональные возможности, но и логически

объединить их в целое, возможности которого много больше, чем сумма частей. Именно такая целостность является основной целью разработки производственной архитектуры, которую предполагает модель MSF.

Модель производственной архитектуры MSF является структурной и состоит из четырех элементов (перспектив): бизнеса, приложения, информации и технологии (рис. 1.2).

Рис. 1.2. Элементы (перспективы) производственной архитектуры

Бизнес-перспектива

Бизнес-перспектива состоит из множества стратегий и планов, цель которых — переход организации от сложившегося состояния к желаемому. Она описывает организацию работ в компании. Модель включает:

- глобальные цели и задачи организации;
- виды продуктов и услуг, производимых организацией;
- бизнес-процессы, реализующие основные функции организации и связь между ними;
- основные структуры;
- взаимодействие всех перечисленных элементов.

Прикладная перспектива

Прикладная перспектива — это услуги, информация и функции, которые не вписываются в организационную структуру фирмы, связывай пользователей, имеющих разные обязанности и квалификацию, для достижения общих целей. Эта перспектива описывает комплект приложений, использующихся в организации, и включает:

- описание сервисов, которые поддерживают бизнес-процессы, представленные в бизнес-перспективе;
- описание взаимодействий и взаимозависимостей корпоративных приложений;
- приоритеты для совершенствования существующих и развития новых приложений на основе бизнес-перспективы.

Информационная перспектива

Информационная перспектива описывает то, что должна знать организация для решения своих задач и обеспечения нормального функционирования. В нее входят:

- стандартные модели данных;
- методы организации данных и управления ими;
- описание структуры «производства» и «потребления» информации в организации.

Информационная перспектива также описывает использование данных в производственных процессах. Это — данные, хранящиеся в базах данных, и неструктурированные данные — документы, таблицы и презентации, созданные в процессе работы организации. Зачастую жизненно важная для организации информация хранится не только на серверах баз данных, но и на тысячах настольных компьютеров, образующих информационную инфраструктуру организации.

Технологическая перспектива

Технологическая перспектива представляет аппаратное и программное обеспечение, необходимое для работы организации. Она включает:

- персональные компьютеры и серверы;
- операционные системы;
- сетевые компоненты;
- принтеры;
- использование Интернета;
- другое периферийное оборудование.

Технологическая перспектива обеспечивает логичное, независимое от производителя описание инфраструктуры и системных компонентов, которые необходимы для поддержки прикладной и информационной перспектив. Она определяет перечень технологических стандартов и сервисов, необходимых для выполнения задач организации. Вот некоторые необходимые стандарты и сервисы:

- топологии;
- среды разработки;
- прикладные интерфейсы;
- средства защиты;

- сетевые сервисы;
- сервисы баз данных;
- технические спецификации.

Четыре перспективы — одна архитектура

Модель производственной архитектуры MSF включает четыре перспективы, но это единая модель. Когда организация создает производственную архитектуру, важно помнить, что ценность архитектуры не только в одной из индивидуальных перспектив, но в связях, взаимодействиях и зависимостях между ними.

Развивая эти четыре перспективы и исследуя их индивидуальные и коллективные взаимосвязи, вы выявите приоритеты, проекты, политику, стандарты и способы руководства организации, то есть именно ту информацию, которая необходима для эффективной работы компании. Эти данные необходимы и при реализации принятых решений и их обеспечении ресурсами. Кроме того, они позволят обеспечить взаимодействие бизнес-подразделений и информационных подразделений организации.

Как соотнести цели бизнеса и ИТ

При разработке программного обеспечения и развертывании приложений в подразделениях, непосредственно занятых бизнесом, руководителям ИТ-подразделений приходится учитывать две важные особенности:

- **сложившееся положение** — когда информационные подразделения приступают к работе, основные бизнес-процессы уже налажены и функционируют, что не позволяет этой группе максимально эффективно применять имеющиеся в ее распоряжении технологии;
- **оторванность от процесса принятия решений** — поскольку ИТ-подразделения не участвуют в решении стратегических вопросов бизнеса, число неудачных технологических решений велико.

Советуем очень серьезно относиться к организации сотрудничества бизнес- и ИТ-подразделений. Новые технологии должны действительно помогать в решении задач бизнеса.

Скорость появления новшеств в технологиях так велика, что некоторые отделы ИТ, изо всех сил стараясь не отстать от них, теряют из виду задачи и цели развития организации. В результате руководство и сотрудники утрачивают доверие и к ИТ-подразделению, и к новым технологиям. Чтобы сохранить доверие, ИТ-отделы должны помнить, что новшества важны не сами по себе, а лишь как средство организации и бизнеса.

Модель производственной архитектуры MSF — это инструмент, который гарантирует, что деятельность информационных структур предприятия будет ориентирована именно на бизнес. Менеджерам подразделений организации не надо давать повод усомниться в том, что ИТ-отдел и используемые им технологии служат именно целям бизнеса. С другой стороны, не следует ожидать, что ИТ-отдел будет координировать свои планы с другими подразделениями, не имея соответствующих возможностей. Существенно, чтобы между информационной службой и менеджерами подразделений, непосредственно занятых бизнесом, установились взаимовыгодные отношения сотрудничества.

«Искусственная стена»

Часто между ИТ и другими подразделениями организации возникает «искусственная стена». Зачастую она появляется в результате неверных представлений обеих сторон.

- **Бизнес-подразделения: «Им ни к чему все знать».** Многие бизнесмены думают, что могут квалифицированно оценивать ситуацию и вне сфер бизнеса, а ИТ-отдел должен основываться исключительно на этих (зачастую нечетко сформулированных) заданиях, Масса печальных примеров свидетельствует, что такого рода требования очень трудно удовлетворить вне контекста целей и задач бизнеса. Поэтому важно, чтобы сама процедура выработки требований к ИТ была организована с учетом разных точек зрения, и особенно той, которая исходит от информационных отделов.
- **ИТ-подразделения: «Мы уже знаем».** Это другое распространенное заблуждение. Многие руководители ИТ-подразделений полагают, что их отстраненность от бизнеса не мешает им создавать эффективные решения. И снова жизнь доказывает обратное; люди, не соприкасающиеся непосредственно с бизнесом, не могут понять всех его нюансов.

Чтобы ИТ-подразделения успешно поддерживали бизнес-подразделения, обеим сторонам придется прилагать усилия к устранению непонимания. Ведь их основная цель — решение задач организации. Ключевыми при этом являются следующие соображения.

- **Приоритет требований бизнеса** — бизнес-перспектива модели производственной архитектуры MSF предлагает такую структуру, которая способствует осознанию потребностей бизнеса и их приоритетного влияния на проекты в области ИТ. Только эти цели могут служить обоснованием любого ИТ-проекта.
- **Понимание задач бизнеса ИТ-подразделениями** — до ИТ-отделов надо довести четко сформулированные цели и механизмы веде-

ния дел, чтобы информационная инфраструктура организации развивалась рационально и обоснованно.

- **Понимание бизнес-подразделениями стоимости ИТ** — руководители бизнес-подразделений должны при обсуждении проектов осознавать стоимость совершенствования существующих процессов и организации новых. Стоимость использования ИТ надо обязательно учитывать в расчетах общих затрат.
- **Вовлечение руководителей ИТ-подразделений в принятие бизнес-решений** — это позволит эффективно совершенствовать существующие бизнес-процессы и планировать новые.
- **Вовлечение руководителей бизнес-подразделений в принятие решений в сфере ИТ** — инвестиции в развитие информационной инфраструктуры должны быть обоснованы деловыми, а не техническими соображениями.

Как руководителям бизнес-подразделений, так и ИТ-менеджерам придется осознать, что не только они способны верно понимать свои задачи. Чем лучше организован обмен опытом и информацией между подразделениями, тем меньше вероятность того, что они станут воспринимать друг друга как противников, и тем легче они найдут общий язык.

Основные опасности при разработке производственной архитектуры

При выработке производственной архитектуры вас подстерегают некоторые опасности, способные понизить шансы успешной реализации проекта.

Избыточное внимание закупкам аппаратного и программного обеспечения

Идея производственной архитектуры не нова. Большинство крупных компаний полагают, что имеют производственную архитектуру, но часто в действительности речь идет о списке приобретаемого аппаратного и программного обеспечения. Однако эффективная производственная архитектура не ограничивается этим списком. Точно так же организация не может ограничиться им при определении стратегии своего развития и при организации работы своих проектных групп.

Отсутствие четкой формулировки целей

Без компетентного планирования и ясного видения целей развития организации никакая производственная архитектура не поможет.

Вспомним определение, приведенное ранее. Единственная возможность *прогресса* — это *движение к четко сформулированной цели,* Противоположностью прогрессу может быть регресс или хождение по кругу. Производственная архитектура должна включать ясную формулировку цели, к которой стремится компания.

Слишком сложно и масштабно, чтобы быть реальным

Другая общая ошибка — попытка максимально широко и глубоко выявить все детали производственной архитектуры. Описание производственной архитектуры в результате становится настолько детальным, что проекты «задыхаются» под ее тяжестью. Более того, поскольку при таком подходе разработка архитектуры требует много времени, проблемы изменятся еще до того, как будет сформулировано их решение.

Типичный план производственной архитектуры — это документ объемом в сотни и даже тысячи страниц, на его создание требуется один-два года. Причем это время, необходимое для собственно формулирования архитектуры, но не для ее реализации. Размер и сложность проекта, а также затраченное на его разработку время зачастую не позволяют выявить приоритетные задачи развития информационной инфраструктуры организации. В результате проект не позволяет решить реальные проблемы бизнеса, и ИТ-подразделению приходится все начинать сначала.

Отсутствие обратной связи и коррекции курса

Создание производственной архитектуры должно быть общим процессом, в который вовлечены пользователи архитектуры (проектные группы), владельцы бизнеса и разработчики архитектуры. Производственные архитектуры часто создаются без учета необходимости обратной связи с проектными группами. Всегда наступает момент, когда планы пора претворять в жизнь, а внедрением архитектуры занимаются проектные группы. Без обратной связи с людьми, выполняющими конкретную работу, архитектура, хорошо выглядевшая на бумаге, на деле оказывается неудобной или просто не работает.

Еще одна распространенная проблема: условия бизнеса и приоритеты меняются именно в то время, когда разработка производственной архитектуры идет полным ходом. Кроме того, более новые и более мощные технологии часто вытесняют технологии, первоначально предлагавшиеся для решения проблем, перечисленных в плане. В отсутствие обратной связи и периодической корректировки курса усилия по разработке архитектуры не дают желаемого результата.

Недостаток интеграции и **стабильности**

Даже когда отдельные ИТ-решения уже работают, они часто не интегрированы в общую производственную архитектуру. Часто квалифицированные пользователи проявляют инициативу, используя новые технологии и приложения, применение которых ничем не оправдано и может дестабилизировать архитектуру. Это вызывает хаос и затрудняет работу других приложений, которые могли бы значительно повысить эффективность производственных процессов.

Недостаточное внимание реализации

Производственная архитектура включает не просто проект, но и возможность его реализации. Приложения должны быть созданы, а инфраструктура — развернута. Все, что определено в архитектуре, следует выполнить в разумные сроки с разумными затратами. Однако часто архитектура оказывается оторванной от реальности.

Производственная архитектура может содержать перспективные задачи, но приоритет должен оставаться за основополагающими вопросами и практическими способами их реализации. Например, если архитектура требует консолидации систем на определенной платформе, должно быть показано, как именно выполнить этот переход и как при этом стимулировать усилия проектной группы по реализации архитектуры.

Неспособность справиться с текущими потребностями бизнеса

Многие проекты не учитывают тот факт, что на время разработки производственной архитектуры жизнь не останавливается, а производственные задачи должны решаться и в период работы над проектом.

Если информационная инфраструктура не обеспечивает решения насущных проблем бизнеса, подразделения компании будут выбирать технологии и системы по своему разумению, пытаясь выжить любым способом, Каков бы ни был подход к производственной архитектуре. он должен включать разработку приложений, поддерживающих основные бизнес-процессы при разработке архитектуры.

Задачи модели производственной архитектуры MSF

Модель производственной архитектуры MSF состоит из рекомендаций, технических методов и ключевых принципов. Назначение модели — в короткий срок получить эффект и сохранить его в постоянно изменяющихся условиях. Эти задачи можно реализовать посредством выпуска версий. Динамичность модели производственной архитектуры позволяет использовать обратную связь для реализации рациональных методов принятия решений. Этот способ повышает способность организации применять в своей работе инновационные технологии, позволяющие успешно решать задачи бизнеса.

Замечание Когда организация реализует модель производственной архитектуры MSF, важно помнить, что это общий метод. Его можно и должно уточнять с учетом ситуации и конкретными потребностями организации.

Модель производственной архитектуры MSF характеризуется;

- **интеграцией** — интересы владельцев бизнеса, архитектурной группы и проектных групп сбалансированы, сотрудники проектных

групп понимают устройство как системы в целом, так и отдельных составляющих информационной инфраструктуры организации, а руководители бизнес-подразделений знают, как технологические решения соотносятся с задачами бизнеса;

- **итерационным** подходом — архитектура решения создается посредством последовательного выпуска версий;
- **работоспособностью** — основная цель разработки производственной архитектуры — быстро создать промежуточную, но вполне работоспособную версию. Далее проектные команды могут продолжать работу над совершенствованием архитектуры. Таким образом обеспечивается обратная связь и коррекция курса;
- **учетом приоритетов** — работа над совершенствованием информационной инфраструктуры предприятия не мешает сосредоточить усилия на тех направлениях, где возможен наибольший эффект с точки зрения бизнеса. Разработка архитектуры всегда учитывает необходимость обеспечения поддержки основных бизнес-процессов.

Создание производственной архитектуры

Ранее мы описали четыре перспективы модели производственной архитектуры MSF (бизнес, приложение, информация и технология) и некоторые трудности, возникающие при создании архитектуры. Мы рассмотрели четыре задачи производственной архитектуры (интеграция, итерация, работоспособность и учет приоритетов); мы также указали на вероятность возникновения «искусственных стен» между группами, непосредственно организующими бизнес, и ИТ-отделами. Вы узнали многое и наверняка у вас возникли вопросы: «Как достичь этих целей? Какие шаги следует предпринимать и в каком порядке?» Как воскликнул один раздраженный ИТ-менеджер: «Просто скажите мне, как загрузить файл проекта и соответствующие шаблоны Excel, и я начинаю работать!»

К сожалению, все не так просто. Процесс разработки производственной архитектуры — нечто большее, чем просто перечень последовательных действий для достижения цели. Вам надо овладеть *ключевыми принципами,* которые позволят организации приспосабливать архитектуру к своим бизнес-целям. Тогда проект производственной архитектуры будет и ценен, и оценен.

Миф о всеохватывающей и всепроникающей архитектуре

Не ждите, что вам сразу удастся создать проект производственной архитектуры, который точно описывает все уровни детализации — от бизнес-процессов до выбора технологии и функциональных возможностей приложений, — и охватывает все проекты и корректирует работу всего предприятия.

Несмотря на очевидную несостоятельность таких попыток, они регулярно повторяются. Многие компании нанимают архитекторов и консультантов, которые запираются на год-два и затем торжественно преподносят совету директоров свои «Ответы на все вопросы». Проблема в том, что к этому времени их труд уже устарел. Попытки определить «все и для всех» серьезно компрометируют ценность подобного метода.

Мы рекомендуем учитывать эти ограничения, когда вы решите построить производственную архитектуру методом *последовательных итераций*. Только при правильном применении метода вам удастся достичь конкретных целей бизнеса и своевременно корректировать проект.

Поэтапный процесс

Модель производственной архитектуры MSF предполагает итерационный, поэтапный выпуск набора последовательных версий. При этом организация постепенно достигает желаемой цели, проходя через различные стадии (рис. 1.3), которые мы детально обсудим позже,

Рис. 1.3. Стадии разработки производственной архитектуры

Движение от существующей архитектуры к желаемой

Руководству компаний приходится постоянно переосмысливать бизнес-процессы и переписывать бизнес-правила. Прикладные системы должны легко реагировать на эти изменения и адаптироваться к ним. Поэтому архитектурным группам не следует при корректировании производственной архитектуры применять метод «штурма и натиска». Мы убедились, что успеха чаше добиваются группы, которые исполь-

зуют постепенный, итерационный подход, предполагающий, что разработка проекта никогда не заканчивается, а выполняется постоянно.

Прежде всего надо ясно понимать нынешнее состояние организации (что есть) и четко определить цели (что будет). После расстановки приоритетов выполняются отдельные проекты, цель которых— постепенное решение поставленных задач, По мере выполнения этих проектов производственная архитектура постоянно корректируется под воздействием обратной связи и изменений в технологии и бизнесе (рис. 1.4)

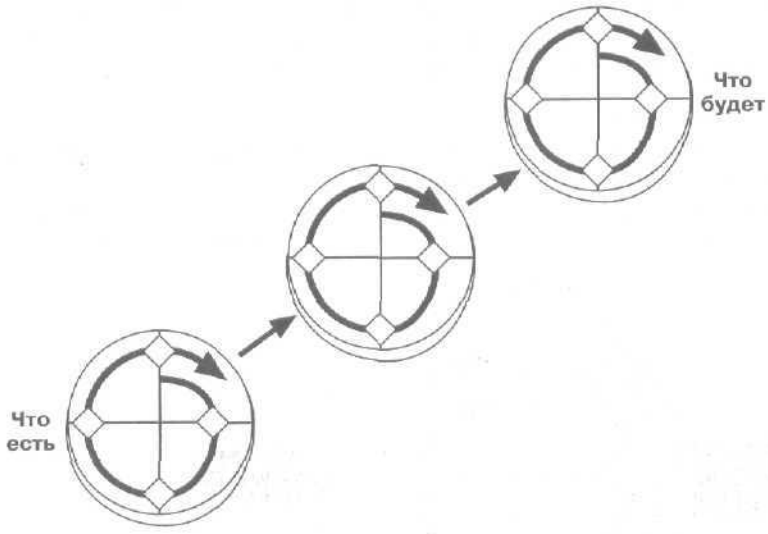


Рис. 1.4. Эволюция через последовательные версии

Внимание! Осознание того, что разработку производственной архитектуры нельзя закончить, может убить и надежду дожить до ее реализации, и удовлетворение от выполнения определенного этапа. Применение концепции последовательного выпуска версий позволяет избежать этой безысходности. Каждый выпуск представляет собой логически последовательный и достижимый этап, совершенствующий инфраструктуру организации.

Достоинства итерационного подхода

Метод выпуска версий предполагает, что производственная архитектура, как правило, слишком масштабна, чтобы ее удалось создать сразу. Правильнее будет расставить по приоритетам разработку последовательных версий, удовлетворяющих потребности бизнеса, и затем методично создавать работоспособные подмножества.

Выпуск версий имеет несколько преимуществ.

- **Исключение из плана лишних требований** — поскольку подразделения, занимающиеся бизнесом, и проектные группы знают, что будет следующий выпуск, они не чувствуют необходимости в первой же версии «собрать в кучу все, кроме умывальника». В каждой версии можно решать ключевые проблемы, что позволит оперативно внедрять ценные для организации решения.
- **Способность реагировать на изменения в каждом следующем выпуске** — требования и приоритеты для «полной» архитектуры часто меняются как раз в тот момент, когда рабочая версия архитектуры уже построена, часть приложений подготовлена и некоторые вопросы инфраструктуры решены.
- **Поиск ответов** — многие организации так боятся неопределенности, что стараются избежать ее, объявив неизвестные факторы известными. Такой самообман иногда имеет самые печальные последствия и приводит к неверным решениям. Единственный способ избежать его — двигаться вперед последовательно, ничего не выдумывая и ничего не принимая на веру.

Важность динамики

Архитектура никогда не бывает статична. Это органичная система, в которой каждый элемент имеет свой жизненный цикл:

- создание (малые функциональные возможности, низкая стабильность);
- развитие (быстрый, но не полностью осознанный рост);
- зрелость (работа выполняется ровно, стабильно);
- закат (борьба за поддержание темпа);
- смерть (отставка, ликвидация).

Чтобы разумно изменять производственную архитектуру, необходимо постоянно обращать внимание на изменения в бизнесе и технологиях:

- понимать, какие элементы архитектуры нуждаются в изменении;
- выявлять участки архитектуры, зависящие от этих элементов.

Выпуск версий позволяет осуществлять постоянную корректировку архитектуры с учетом изменений. Реализация отдельных элементов архитектуры в каждом очередном выпуске уже способствует повышению эффективности управления и инвестиций, оставляя возможность для усовершенствований в следующих выпусках.

Философий выпуска версий

Чтобы подобный метод заработал, группа должна подготовить первую версию производственной архитектуры, а затем регулярно выпускать следующие, реагирующие на изменение задач организации.

Основные лозунги подхода, связанного с выпуском версий, таковы: «Лучше меньше, да лучше», «Понимание лучше незнания», «Прогресс лучше обещаний».

Прогнозирование и реагирование

Одно из наиболее важных препятствий, возникающих в процессе разработки производственной архитектуры, — это то, что все основные бизнес-процессы уже сформированы. В результате зачастую упускаются возможности использовать технологии эффективнее или по-новому.

Разработка производственной архитектуры — это не просто реагирование. Под реагированием мы понимаем однонаправленный поток решений и информации, который определяет методы использования технологии в организации. Такой поток включает следующие действия (рис. 1.5):

- выявление бизнес-процессов;
- подбор приложений и информации;
- реализацию технологии.

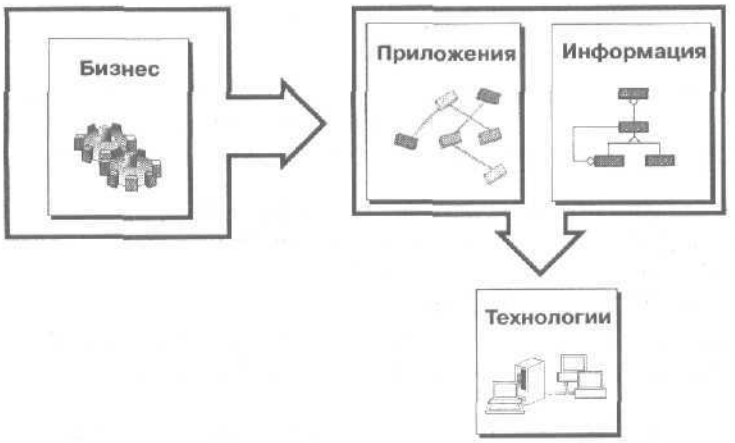


Рис. 1.5. Поток решений и исходной информации

Поток решений важен, но он должен сочетаться с прогнозированием — встречным потоком исходной информации, идущим от технологии к бизнес-процессам. Как показано на рис. 1.6, встречный поток состоит из:

- анализа технологии;
- определения приложений и выявления информации;
- реконструкции бизнес-процессов.

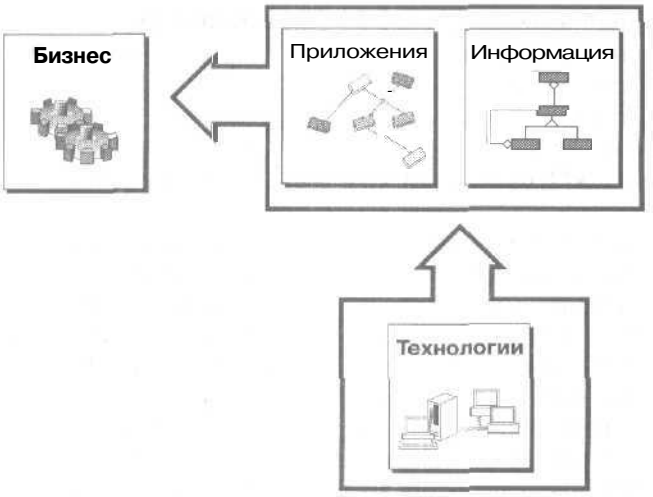


Рис. 1.6. Встречный поток решений и информации

Замечание Технология сама по себе не создает продукта, но ее применение в инновационной сфере для изменения возможностей бизнеса может иметь большое значение. Решение вопроса о том, где и как применять технологию, — еще одна сфера, где кооперация бизнеса и ИТ необходима. Электронная коммерция — хороший пример встречного потока, идущего от технологии.

Все внимание бизнесу

Модель производственной архитектуры MSF позволяет сконцентрировать усилия ИТ-отделов на выявлении основных бизнес-процессов, которые особенно важны для организации, и именно им уделить максимум внимания. Если же при создании архитектуры не выявляются важнейшие бизнес-процессы, то развиваются, как правило, отделы, в большей степени поддерживающие второстепенные и административные процессы, а не действительно важные для бизнеса. Основные процессы при этом часто остаются «в загоне».

Модель производственной архитектуры MSF позволяет сосредоточиться на основных процессах в важнейших областях, а значит, сформировать верное направление развития, поскольку архитектура сначала создается, а затем начинает развиваться. Каждый очередной выпуск работоспособен — это важное требование. Такой подход позволяет не тратить время на мечты в «башне из слоновой кости», а смотреть в будущее и планировать работу, исходя из реальных условий.

Производственная архитектура и конкретные проекты

Производственная архитектура координирует все проекты, которыми занимается организация (рис. 1.7). Она позволяет;

- добиваться логической целостности проектов;
- разумно планировать и использовать ресурсы;
- привести инфраструктуру и приложения в соответствие с целями организации.

Таким образом, производственная архитектура становится основой стратегического планирования развития информационной инфраструктуры. Она позволяет выделить важнейшие бизнес-процессы и технологии, поддающиеся автоматизации, и исходя из этого определить сферу применения приложений и направление развития инфраструктуры. Такой «взгляд сверху» особенно важен в случае, когда проекты выполняются одновременно или задействуют общие ресурсы.

Производственная архитектура — это больше, чем инструмент планирования. Она также определяет направление развития и развертывания инфраструктуры. Это может принести серьезную пользу организации за счет того, что:

- проектируемые приложения приведены в соответствие с целями бизнеса;
- применяются только одобренные технологии;
- после реализации технологии приложения эффективно эксплуатируются.

Производственная архитектура определяет административные и технологические рамки разработки конкретных проектов.

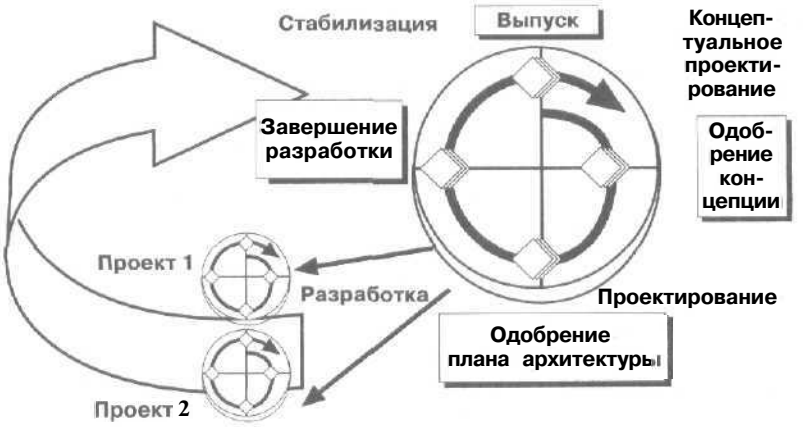


Рис. 1.7. Координация отдельных проектов производственной архитектурой

Планирование во время разработки и разработка во время планирования

Организации по своей иерархической природе часто признают единственно возможным нисходящий способ принятия решений, особенно когда дело касается определения основных задач бизнеса и стратегического планирования. Во время разработки производственной архитектуры руководство может навязывать стандарты сверху, но, как показано на рис. 1.8, архитектура должна также совершенствоваться и развиваться снизу, на основе информации, поступающей от разработчиков проектов.

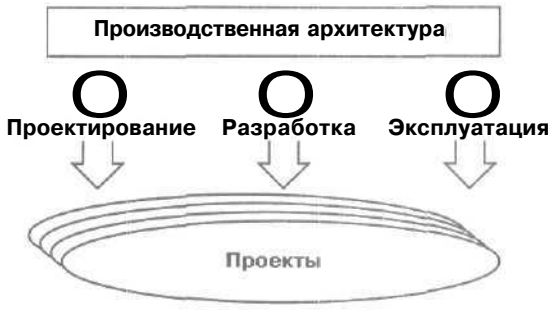


Рис. 1.8. Структура планирования производственной архитектуры

Поначалу это было просто требование здравого смысла — сначала создать модель предприятия, а затем на основе этой модели заниматься отдельными ИТ-проектами. Чтобы гарантировать согласованность определений данных и интерфейсов и поддержку бизнес-процессов в рамках организации, каждый проект должен был «выводиться» из модели предприятия. Однако этот метод планирования архитектуры часто невыполним, так как требует, чтобы к началу процесса планирования были известны все детали. Теперь мы уже знаем, что такое требование ошибочно. Производственная архитектура не может быть определена в вакууме, а обязательно должна реагировать и на ту новую информацию, которая выявляется в процессе создания решений. Метод последовательных версий, который подразумевает обратную связь с проектными группами и пользователями, позволяет постепенно улучшать архитектуру. В противном случае быстро меняющиеся условия сделают бессмысленными все усилия как по созданию производственной архитектуры, так и по ее применению в конкретных проектах.

Резюме

Эту главу мы начали с описания *архитектуры* как последовательного и целостного технологического плана. Мы отметили, что архитектура должна быть обращена к целям и задачам бизнеса, и на этой основе определять направление развития информационной инфраструктуры. Затем мы обсудили важность приверженности руководителей как бизнес-подразделений, так и ИТ-отделов подходу, ориентированному на приоритетархитектуры.

Вы узнали о том, как влияет на бизнес быстро изменяющаяся ситуация, и, надеемся, осознали, что единственный способ преодолеть это влияние — хорошо спланировать производственную архитектуру. Мы определили производственную архитектуру как «логически связанный, цельный план работ и скоординированных проектов, необходимых для преобразования сложившейся структуры информационных систем и приложений организации к намеченному будущему состоянию».

Мы представили принципы разработки приложений MSF и обсудили модель производственной архитектуры MSF, которая базируется на четырех перспективах (бизнес, приложение, информация и технология) и имеет четыре цели (интеграция, итерационность, работоспособность и поддержка приоритетов). Вы убедились, как важно привести производственную архитектуру в соответствие с задачами бизнеса, и познакомились с серьезной опасностью — возникновением «стены» между подразделениями, занимающимися вопросами бизнеса, и ИТ-подразделениями.

Наконец, мы обсудили связь между стратегическими планами организации, ее производственной архитектурой и отдельными проектами.

Закрепление материала

1. Какие проблемы развития информационной инфраструктуры решает производственная архитектура?
2. Каковы основные задачи производственной архитектуры?
3. Перечислите фазы применения производственной архитектуры в проектах разработки программного обеспечения.
4. Перечислите четыре перспективы модели производственной архитектуры MSF.
5. Как выпускать приложения в период разработки проекта производственной архитектуры?

Практикум 1. Разработка производственной архитектуры

Тим О'Брайан, начальник сетевого отдела компании «Фергюсон и Барделл», заглянул в офис директора по информационным технологиям.

– Ты хотел видеть меня, Дэн? — начал он, но вдруг обнаружил, что офис пуст.

Он уже собрался уходить, когда услышал шуршание, затем ворчание и какую-то возню. Либо Дэн Шелли поймал непонятно откуда взявшегося крупного зверя, например, гориллу, либо в офисе все-таки был человек.

Тим медленно прошел в комнату, стараясь держаться поближе к выходу, поскольку вариант с гориллой ему вовсе не улыбался. Он снова услышал ворчание и понял, что оно доносится из закутка между столом и шкафом. Тим обошел стол кругом и увидел сидящего на полу босса.

Дэн пытался вытащить картонную коробку со дна шкафа: потянул ее, крякнул, с трудом подняв стопку толстых папок, и уронил их себе на колени. Он был так увлечен, что, не замечая Тима, начал просматривать заголовки папок, но, очевидно, ему не удавалось отыскать то, что нужно. С видимым раздражением он отбросил последнюю папку в кучу напротив и взял другую стопку.

Тим не мог припомнить, что когда-нибудь видел шефа в таком положении. Он прислонился к стене и усмехнулся:

— Я уже вытащил отсюда все лотерейные билеты, если ты именно их ищешь.

Не глядя, Дэн ответил:

— Ха! То, что я ищу, найти так же сложно, как выигрышный лотерейный билет.

Он просмотрел последнюю папку в коробке и бросил ее в кучу на полу, снова застонав от раздражения. Вставая и отряхиваясь, он обернулся к Тиму:

– Есть какое-нибудь еще место в этом здании, где может быть припрятана информация о ПА?

Тим гордился своими профессиональными познаниями и уж тем более считал, что знает все аббревиатуры, но эта оказалась неизвестной. Что за ПА? И никакой путеводной нити. Он не хотел признаваться в некомпетентности и решил потянуть время.

—Так, и которая же это ПА, Дэн?

– Разумеется та, которая для «Фергюсон и Барделл», — ответил Дэн. Он убрал папки и запихнул коробку обратно в шкаф. Выпрямляясь, он увидел сконфуженное лицо Тима и пояснил без тени высоко-

мерия: — Производственная архитектура, Тим, Мне нужны материалы по производственной архитектуре компании «Фергюсон и Барделл».

— Ах, вот оно что, — Тим был доволен, что не нужно больше ломать голову над расшифровкой аббревиатуры, но по-прежнему не понимал, о чем идет речь, хотя конечно виду не подавал, — не думаю, чтобы у нас была одна из них.

Дэн засмеялся.

– Нет, Тим, она у нас точно есть. Похоже, мы ее просто не задокументировали. Я, кстати, давно это подозревал. Поэтому и попросил тебя зайти.

Он закрыл дверцу шкафа и сел за стол, жестом предлагая Тиму сесть напротив.

– Понимаешь, мы собираемся в ближайшее время запустить несколько крупных проектов — в следующем году или около того. Прежде чем начать, однако, нам надо бы получить представление о том, что мы уже имеем и почему. Короче говоря, мы должны описать нашу производственную архитектуру.

Дэн потянул папку со стола и открыл ее.

– Я собираю команду, которой придется приблизительно за два месяца сделать первый набросок нашей ПА и рекомендовать компании первый набор проектов, — он заглянул в календарь и продолжил: – Сегодня 22 февраля. Начнем в следующий понедельник, 1 марта. Мне нужен человек из твоей группы, сообразительный и хорошо знающий нашу технологическую инфраструктуру, причем такой, которому ты мог бы поручить важный проект. Кроме того, хорошо бы он (или она) любил решать сложные задачи и хотел приобрести соответствующий опыт. Подумай, кого ты можешь порекомендовать?

Тим усмехнулся:

– Знаю я одного парня, который точно подходит под твое описание — сообразительный, знает организацию, любит сложные задачи — это я!

Но Дэн покачал головой:

– Нет, Тим, ты неподходящая кандидатура, — и, увидев удрученное выражение на лице Тима, продолжил более благосклонным тоном: — Не огорчайся, Тим. У меня на подходе другой проект — большой и важный, и твоя фамилия уже в списке.

Услышав это, Тим успокоился:

– Это хорошо. Мне бы не хотелось, чтобы вся слава досталась молодым пижонам!

Оба засмеялись: молодость Тима была в отделе постоянным предметом шуток.

Тим немного подумал и продолжил:

– Есть одна женщина, она пришла к нам на работу прошлой осенью, почти одновременно с тобой. Ты знаешь, о ком я говорю — та, чью фамилию ты никогда не можешь вспомнить.

– Ты имеешь в виду Дженни? Дженни Флейта, или Дженни Альт, или ... ах, черт, как же ее фамилия?

— Дженни Сакс, Дэн, как саксофон. Она очень хорошо управляется с сетью в свою смену, и я заметил, что она уже запомнила почти все адреса прерываний. Видно, что она имеет вкус к деталям и к работе с документацией, и — что там говорить — похоже, что ее *мозг* просто лучше организован, чем у большинства из нас. Если я правильно понял твои пожелания, то она отлично подойдет.

— Похоже на то, — сказал Дэн. — Можешь освободить ее для этой работы? Она будет занимать примерно десять часов в неделю, может, и больше.

– Придется приспосабливаться, но мы с этим справимся. Надо учесть, что такой опыт сделает ее более ценным работником, и потом — это ведь всего два месяца.

Ответ Тима удовлетворил Дэна. Это было именно то, что он хотел бы услышать от молодого коллеги. Тим уже не раз показывал, что «заботится о своих людях», что только прибавляло ему очков, как начальнику отдела.

Дэн проводил Тима до двери офиса.

— Спасибо за проявленную добрую волю, Тим. Пусть Дженни заглянет ко мне немного попозже — мы обсудим задачу в деталях. И просматривай свою электронную почту. Я дам тебе знать о начале большого проекта где-нибудь на следующей неделе.

Когда Тим попрощался и вышел, Дэн задумался. «Один подходящий человек в команде уже есть. Хотелось бы надеяться, что и другие менеджеры проникнутся идеей этой работы и тоже дадут мне хороших людей. Иначе, — он повернулся к столу, — этот проект с производственной архитектурой превратится в те, что мне доводилось видеть раньше — сплошная технология и никакого бизнеса».

Начало

На первой же встрече группы разработчиков производственной архитектуры фирмы «Фергюсон и Барделл» Дэн перестал волноваться о том, удалось ли собрать хорошую команду. По-видимому, начальники других отделов решили не разочаровывать нового директора по информационным технологиям и прислали Дэну лучших и самых ярких людей. Теперь приходилось волноваться о другом — смогут ли все «эго» и все мнения уместиться за одним столом в его офисе.

Дженни пришла первой. «Не то что ее начальник!» — подумал Дэн с усмешкой, приветствуя ее. Следующим в дверях появился Кевин

Кеннеди, выскочка из управления, посланный исполнительным директором. Кевин работал в комитете долгосрочного планирования фирмы, где заработал репутацию проницательного аналитика и человека, который способен претворять в жизнь свои рекомендации.

Сразу за Кевином в дверях появилась Джо Браун, заместитель директора компании по производству. Она основательно занималась проверкой готовности к 2000 году и наверняка знала все или почти все о приложениях, используемых в компании. Кроме того, она славилась своим острым умом и ехидными записками, которыми пыталась образумить тех, кто не следует инструкциям.

Когда участники группы уже рассаживались за столом, появился доктор Ричард Каплан, загадка фирмы «Фергюсон и Барделл». Дик пришел в группу консультантов компании в конце 80-х; он занимался анализом организации хранения данных. Что-то в нем произвело сильное впечатление на тогдашнего директора по информационным системам, и позже он пригласил Дика на работу и пристроил его в качестве «аналитика» в отдел разработки. Дик продолжил совершенствоваться в области, связанной с моделированием данных и технологиями хранения, его часто привлекали к проектам на ранних стадиях разработки. Он отличался необщительностью и многим казался из-за этого неинтересным, скучным человеком. Дэн, заинтригованный личностью Дика, однажды попросил, чтобы ему принесли диссертацию Дика, которая называлась: «Сочинения Канта: анализ на основе поздних писем». Прочтя всего несколько страниц, Дэн раз и навсегда убедился, что Дик Каплан совсем не остолоп.

- Всем доброе утро. — сказал Дик, машинально опустив трубку в карман, когда ему понадобилось взять стул. — Извините, если я вас задержал.

- Нет, мы только начали, Дик. Здорово, что вы смогли придти, — ответил Дэн, посылая Дику по кругу последний блокнот. — Легко ли Билл отпустил вас к нам?

- Ну, поворчал немного, уровень недовольства был примерно 1 по шкале Билла, так что, я думаю, все в порядке.

Дэн засмеялся:

— Джо и Кевин, я надеюсь, что у вас тоже не было проблем с начальством.

Джо ответила первой:

- Нисколько, Дэн. Мне просто сказали появиться здесь в 10 утра по поводу одного важного проекта и посвятить этому делу столько времени, сколько потребуется.

Кевин согласно кивнул.

- Отлично, — сказал Дэн, заняв место за столом и открыв блокнот. — Так как никто из вас не представляет, зачем мы собрались, я

хотел бы объяснить в общих чертах, о чем идет речь. Если вы откроете свои блокноты, то найдете в них повестку дня. Сверху вы видите название: «Проект разработки производственной архитектуры компании», или ПА для краткости. Короче говоря, наша задача — выявить нынешнее состояние производственной архитектуры компании, придать ему формализованный вид и разработать направление развития нашей ПА на будущее.

Фергюсон и Барделл

Строительство · Проектирование · Управлениепроектами

Проект разработки производственной архитектуры компании

Повестка дня

Дата: 1 марта 1999 г. Тема: запуск проекта

I. Что такое производственная архитектура

II. Четыре перспективы
- Бизнес
- Приложения
- Информация
- Технология

III. Два подхода: превентивный и реагирующий

IV. Версии

V. Вопросы и ответы

VI. Задания и ответственные

Чикаго · Детройт · Милуоки · Цинцинати · Индианаполис · Луисвилль

Кевин слушал Дэна внимательно и кое-что записывал. Воспользовавшись паузой, он спросил:

—Хотелось бы знать — что же все-таки такое *производственная архитектура*?

Дэн улыбнулся.

– Загляните в повестку дня. Наша первая задача как раз и состоит в том. чтобы дать определение ПА. Я хочу, чтобы каждый из вас сделал это самостоятельно. Так определение лучше уляжется в головах, а мы обсудим его по ходу дела,

Он встал и прошел к доске на стене.

— Я буду выписывать определение по одной фразе и хотел бы, чтобы вы переписывали их к себе. Вот первое, — Дэн повернулся к доске и написал; *«логически связанный, цельный план действии и скоординированных проектов»*. Дэн подождал, пока остальные перепишут.

– Прежде всего, — сказал он, — это план. В этом документе не просто указано, что мы делаем, но и определено, что мы собираемся делать. Все внутренние части такого плана логически связаны. Другими словами, если мы возьмем любые два случайно выбранных раздела этого плана, их комбинация должна иметь смысл. Это относится к различным видам нашей деятельности — тем задачам, которыми мы занимаемся постоянно, и к разовым проектам, Все уловили?

Пока все записывали, Джо спросила:

— Дэн, это слишком широкое определение. Мы собираемся документировать все виды деятельности и все проекты компании?

Дэн покачал головой:

– ПА прежде всего относится к планам, которые связаны с информационными технологиями. Мы не будем пытаться документировать все виды деятельности и все проекты компании.

Кевин спросил с явным непониманием на лице:

– Тогда почему все мы здесь? Я не очень-то разбираюсь в этой области,

– Как ты сейчас увидишь, Кевин, — сказал Дэн невозмутимо, — каждый из вас имеет свою точку зрения, которая очень ценна и даже необходима для реализации этого проекта, — он повернулся к доске. — Хорошо, пошли дальше.

Он добавил: *«необходимых для преобразования сложившейся структуры информационных систем и приложений организации»* к определению, написанному на доске.

– Видите, в этом смысл плана. Мы будем использовать документ, который создаст наша команда, как руководство для работы в информационном отделе. Этот документ пригодится нам, когда придется разрабатывать или покупать приложения, развивать инфраструктуру. Вопросы есть?

Вопросов не было, и Дэн написал следующую часть определения на доске: «*к намеченному будущему состоянию ...*». Он остановился и подождал, пока все закончат.

– Это важный пункт. Наша задача — описать, где мы находимся сейчас и где хотим оказаться через некоторое время. И кроме того, нам придется определить ту последовательность действий, которая позволит достичь этого состояния (нечто вроде расписания); другими словами, мы должны определить несколько целей для достижения того состояния, которое мы наметили. Звучит осмысленно? — все кивнули в знак согласия, и Дэн продолжил: — Хорошо. Эта последняя фраза начинает проясняться, почему вы все находитесь здесь.

Он повернулся и дополнил определение: *«базирующийся на текущих и планируемых бизнес-целях и бизнес-процессах»*. Он повернулся к группе.

– Как видите, — сказал он. положив маркер и возвращаясь за стол, — мы в отделе информационных технологий не можем самостоятельно подготовить этот документ. Нам нужна информация из остальных подразделений компании, чтобы документ отражал реальные производственные и административные процессы и особенности бизнеса, а не наши собственные представления о том, что надо делать.— Он повернулся к Кевину: — Теперь ты, наверное, понял, Кевин: ты здесь, потому, что работаешь в комитете по стратегическому планированию компании. Мы должны познакомиться с этими долгосрочными планами прежде, чем сможем разрабатывать наши собственные планы, как стратегические, так и тактические.

– А как насчет остальных, Дэн? — сказала Джо, а Дик Каплан добавил; — Да, Дэн, я вижу, что Кевин тут действительно нужен, но мне неясно, чем будут полезны заместитель директора и аналитик — мастер на все руки.

Дэн ответил:

— Хорошие вопросы... Я думаю, что следующий пункт нашей повестки дня поможет каждому из вас лучше понять свою роль. Взгляните на первый рисунок в блокнотах.

Четыре модели с перспективой

На первом рисунке был изображен треугольник, разделенный на четыре части — «Бизнес», «Приложение». «Информация» и «Технология».

– Полная архитектура предприятия базируется на четырех моделях, каждая из которых определяет свою точку зрения, или *перспективу,* — сказал Дэн. — Описания этих моделей вы найдете на следующих страницах. Давайте разберем их и решим, за что каждый из вас будет отвечать.

Он повернулся к Кевину:

– Бизнес-перспектива — это твое, Кевин. Мы хотим знать, какова стратегия компании и каковы планы ее реализации. По существу, именно от тебя мы будем узнавать о том, как фирма работает сегодня и как группа стратегического планирования представляет себе ее работу завтра. Ты также должен помочь нам понять, какие процессы происходят в компании при текущей деятельности. Это твой пункт списка на этой странице. Понятно, о чем идет речь?

Читая в своем блокноте страницу, посвященную бизнес-перспективе, Кевин становился все более серьезным. Наконец он поднял глаза и сказал:

– Это большая задача, но я думаю, что опыт работы в комитете планирования поможет. Я могу это сделать.

Дэн кивнул:

– Задача действительно большая, но тебя не пригласили бы в эту группу, если бы сомневались в твоих возможностях. Надеюсь, уже сегодня мы сократим задачу до приемлемых размеров.

Он повернулся к Джо:

– Твоя область, Джо, это прикладная перспектива. Твоя задача — выявить и описать все приложения и функциональные возможности, которые применяются в компании для реализации процессов, перечисленных Кевином. Кроме того, ты должна описать все взаимодействия и интерфейсы *между* приложениями и их функциями. Обо всем этом сказано на соответствующей странице. Тебе все ясно?

Джо кивнула:

– Теперь я понимаю, почему меня выбрали на *эту* роль. Когда мы готовились к 2000 году, мы провели инвентаризацию всех приложений — этот опыт нам наверняка пригодится.

– Точно, — сказал Дэн. — Но не думай что дело только в этом. Даже без учета этого опыта твои рекомендации говорят сами за себя. Я сильно подозреваю, что твое знание делопроизводства оказалось решающим при выборе кандидатуры в нашу группу.

Дэн повернулся к Дженни. — Держу пари, что ты уже догадалась, какая часть твоя, Дженни, — сказал он с улыбкой.

Дженни улыбнулась в ответ.

— Думаю, да, — она посерьезнела и продолжила: — На этой странице я просмотрела ту часть, что относится к технологической перспективе. Это больше похоже на работу для комиссии по инвентаризации, которой не помешал бы какой-нибудь инструментарий автоматизации. Мне, что, собрать эти данные вручную?

Дэн почувствовал огорчение в ее голосе:

— Дженни, ты права. С таким инструментом было бы проще. Но я думаю, тебе надо скооперироваться с Джо — она проводила инвентаризацию аппаратного обеспечения, и ее результаты весьма пригодят-

ся тебе. Кроме того, ты могла бы собрать многие нужные тебе данные по электронной почте — с помощью хорошо оформленного опросника. И еще: я переговорю с Тимом, чтобы он внес в заявку в бюджет этого года на приобретение подобного инструментария. — Он облокотился на стул и продолжил: — Однако это еще не все — ты не должна ограничиваться только аудитом текущей ситуации. Взгляни еще раз на определение. Твоя задача — описать текущее состояние среды, но не только. От тебя требуется также разработать план ее развития. Обдумай предложения по следующему вопросу: какие технологии нужны, чтобы компания смогла достичь намеченных стратегических целей. Конечно, тебе еще придется убедить нас всех, что эти технологии действительно требуются, и обосновать свои доводы, опираясь на задачи бизнеса компании. Тем не менее, по-моему, тебе можно позавидовать — у тебя самая «веселая» роль в нашей команде.

Дженни обдумала все это и уже собиралась что-то сказать, когда Джо повернулась к ней и начала напевать «Голубые небеса»*. Дженни засмеялась и сказала:

– Хорошо, хорошо, вы меня купили. Я буду делать вашу занудную работу в надежде, что когда-нибудь будет и работа для души. — Она посмотрела на Дэна и улыбнулась: — А когда мы сегодня закончим, я скажу Тиму, что ты хотел его видеть по поводу корректировки бюджета.

– Это честно, — сказал Дэн. Затем он повернулся к Дику, который наблюдал за происходящим с удивленным выражением на лице: – Итак, Дик, тебе досталась последняя задача. Как ты отнесешься к предложению обеспечить нас информационной перспективой?

Дик изучил свою страницу и затем сказал:

– Что ж, она кажется мне наиболее абстрактной из всех четырех, Моя задача — описать то, что фирма «должна знать», чтобы осуществлять все свои процессы и операции. Я полагаю, что лучше всего пойти следующим путем: взять то, что получится на выходе у Кевина и у Джо, и трансформировать это в модели, показывающие, какая информация и какие данные нужны, чтобы модели стали работоспособны.

Полагаю, мне также придется определить все места, в которых хранится информация, и выяснить, что представляет собой наша политика организации данных. В основе моих изысканий до некоторой степени лежат данные, добытые Дженни. В конечном счете, начало моей работы зависит от результатов работы остальных. Но, раз уж мы строим планы на будущее, их работа также должна в какой-то степе-

* Знаменитая мелодия Дюка Эллингтона «Blue Sky»; Джо намекает на безоблачные перспективы. — *Прим. ред.*

ни зависеть от моей — ведь им придется принять к рассмотрению модели и способы хранения данных, которыми мы пользуемся, и учесть, как они влияют на их перспективы. Для начала хватит?

Дэн восхищенно покачал головой и улыбнулся:

— Дик, я рад, что ты участвуешь в работе группы, — твоя способность составлять единое целое из вроде бы несвязанных частей просто поражает. Ты снова объяснил все лучше, чем официальные материалы. Хотя ты и сказал, что твоя работа будет базироваться на результатах остальных, я все-таки полагаю, что ты и сейчас уже знаешь достаточно о бизнесе, процессах и технологии в компании, так что можешь начать работу над своим разделом. Ты согласен?

Дик кивнул и затем спросил:

– Надо использовать для этой работы универсальный язык моделирования?

— В конечном счете, да, — ответил Дэн. — Сейчас, однако, начнем с нескольких простых таблиц со структурами данных и перечнем процессов, к которым они относятся. Так нам будет проще осознать все это на первом этапе.

Прогнозирование, реагирование и версии

Дэн перевернул страницу блокнота:

— Давайте присмотримся к направлению процессов и приоритетам нашей команды. Взгляните в начало страницы 6.

Там оказалась схема: от бизнес-перспективы стрелки через прикладную и информационную перспективы вели к технологической.

— Обычно работа над производственной архитектурой организуется таким образом, — сказал Дэн, — сначала мы выявляем и документируем стратегические планы и процессы, затем выявляем приложения и способы хранения информации, необходимые для реализации этих планов, Наконец, мы выбираем технологии, необходимые для поддержки приложений и информации, в которых нуждаемся.

Кевин откинулся в своем кресле и сказал:

– Классно! Итак, я приношу евангелие от бизнеса, и все остальные используют его, чтобы делать свою работу, пока я сижу в сторонке и отдыхаю.

Он самодовольно усмехнулся и добавил:

– Теперь этот проект нравится мне больше.

Дэн подумал: «А вот ты теперь нравишься мне меньше», и сказал Кевину:

– Это только часть картины. На следующей странице отражена другая точка зрения.

Все перевернули страницы и увидели рисунок, во всем подобный первому, за исключением того, что стрелки теперь тянулись от технологической перспективы через информационную и прикладную к бизнес-перспективе.

Дэн продолжал:

– В современном мире технология меняется так стремительно и с такими серьезными последствиями для бизнеса, что только безрассудный руководитель игнорирует ее влияние на планы развития компании. Короче говоря, мы хотим иметь направленный в обе стороны поток информации, проходящий через все четыре перспективы, но особенно важна связь между технологией и бизнесом. — Он посмотрел прямо на Кевина: — Одна из твоих задач — выбрать те технологические возможности, которые мы выявим, и учесть их в процессах долгосрочного планирования компании. Чтобы это сделать, тебе придется очень внимательно прочитать и изучить все то, что сделают остальные; то же сделают Дженни, Джо и Дик со своей стороны. Как думаешь, справишься?

Уловив тон, которым Дэн произнес последнее предложение, Кевин ответил кротко:

– Я постараюсь.

– Хорошо! — сказал Дэн решительно. Он смотрел на остальных: —Надеюсь, что все поняли, как ваши задачи влияют на целое и насколько важен каждый из вас для успеха этого проекта. Есть вопросы о ролях и перспективах?

Вопросов не было, поэтому Дэн перевернул страницу блокнота и сказал:

– Тогда все в порядке, посмотрим на график.

График вызвал недоуменный гул. Дженни взглянула на Дэна;

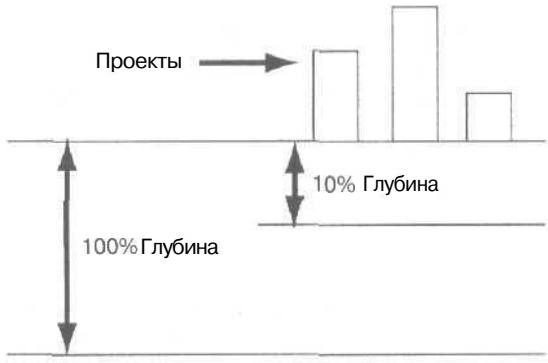
— Дэн, насколько я помню, ты сказал, что проект рассчитан на два месяца, но список индивидуальных задач, а также масштабность общей задачи заставил меня усомниться, что мы сможем сделать это за два месяца. Два года — может быть, но два месяца? Исключено.

Остальные согласно кивали. Дэн подождал немного и ответил:

— Дженни, если бы мы решили за один раз сделать этот проект целиком, ты была бы абсолютно права. Но, видите ли, это ловушка, в которую уже попались многие до нас. Нельзя сразу решить все задачи на 100% как в ширину, так и в глубину: когда проект закончен, в большинстве случаев реальная архитектура уже изменилась, и приходится изучать весь процесс снова и снова. А поскольку при этом тратится много времени на документирование текущего состояния, то перейти к планированию будущего не удается. — Он помолчал, и затем сказал твердо: — Наша команда избежит обеих ошибок. И вот каким образом,

Он подошел к доске, вытер ее, и затем нарисовал линию посередине.

— Сейчас я попробую нарисовать то, что сказал. Эта линия — полный охват по ширине. — Он нарисовал еще одну линию внизу доски, затем — стрелки между двумя линиями: — А этот прямоугольник — 100-процентный охват по глубине. — Затем он нарисовал линию на дюйм ниже верхней: — Таким образом мы избежим опасности погрязнуть в деталях. Мы будем «охотиться» только за 10% глубины. — Он опустил маркер и продолжил: — При документировании текущей ситуации мы постараемся соблюдать довольно высокий уровень абстрагирования — только основные процессы, основные инфраструктурные ресурсы, основополагающие стратегические планы. Это закончится для каждого из нас идентификацией тех ключевых элементов наших областей, на которые мы просто обязаны обратить внимание. — Вернувшись к доске, он нарисовал три прямоугольника выше средней линии. — Когда мы получим общую картину нашей текущей ситуации, то сможем выбрать два или три больших проекта или процесса, которые хотим осуществить. Они станут нашими приоритетами в этом плановом «сезоне». На этом создание первой версии производственной архитектуры будет закончено.



— Как мы сможем, охватывая только 10%, все же выполнять наши задания? — спросила Джо. — Я только что начала видеть ценность этой работы, почувствовала радость от того, что стала участником этой команды. Но если мы собираемся выпускать какой-то полусырой документ, слишком общий для того, чтобы кто-то мог им реально воспользоваться, я лучше уйду прямо сейчас. И если все, что мы делаем, — это просто туманное оправдание для проектов, которые вы все равно будете делать, я не хочу в этом участвовать.

Дэн понял, что репутация Джо как человека прямого заслуженна. С минуту он думал, как отреагировать на ее комментарий. Наконец, сказал:

— Джо, я совершенно согласен. Мое время слишком дорого, чтобы растрачивать его на ерунду, будь то проект или документ. Мне также не нужно придумывать оправдания для моих проектов. Я уже достаточно взрослый, чтобы получать то, что мне нужно, без посторонней помощи.

Он заметил, что Джо не обиделась. «Отлично, она не только умеет говорить, что думает, но и выслушивать других», — подумал он.

Дэн вернулся на свое место за столом.

— Посмотрим, поможет ли эта аналогия. Предположим, вам нужно выполнить сложную работу по реконструкции нашего дома. Как вы станете действовать? Для начала вы, вероятно, попробуете выявить общую картину. Вы сделаете набросок той площадки, которую предстоит перестраивать. Это текущее состояние. Затем вы составите список того, что хотите изменить, и выберете то, что покажется вам наиболее важным и рентабельным, Таким образом, вы поймете, что надо делать в первую очередь. Этот метод мы применим и сейчас. Мы выберем два или три проекта из нашего списка — те, которые можно выполнить немедленно или в разумное время, и постараемся реализовать их. Мы получим необходимую дополнительную документацию. Осуществляя процесс таким образом, мы будем двигаться в нужном направлении *разумно*. В этом ключ. Мы хотим начать с достаточно высокого уровня, чтобы первая версия описания архитектуры была настолько полной, насколько это возможно за разумное время. Подробное описание текущего состояние позволит нам планировать будущее.

Он повернулся к Джо:

— Это звучит разумно, Джо?

Она медленно кивнула:

— Да. Это в самом деле отличная аналогия. Реконструкция — это не строительство нового дома. Чтобы начать ее, нам не нужен план всех работ. Мы уже *живем* в нашем доме. Он уже построен. Мы можем, конечно, создать планы строительства нового жилья, но зачем? Гораздо лучше, если мы достаточно хорошо поймем, что у нас уже есть, чтобы сформировать предстаачение о будущем нашего дома, основываясь на нынешнем состоянии. Затем мы решим, как попасть отсюда туда, принимая во внимание советы профессионалов — нашей соседки, строительного подрядчика и ее друга-архитектора. Это будет наша обратная связь, — закончила она, улыбнувшись Дженни и Дику.

— Ей-богу, ты отлично схватываешь суть! — сказал Дэн, оглядывая остальных. — Все ясно с нашими задачами для первой версии?

Когда все кивнули в знак согласия, Дэн вернулся к повестке дня в начале своего блокнота:

— Хорошо, тогда мы готовы к заданиям.

Первые задания

Дэн вытащил несколько листков из блокнота и пустил их по кругу:

- Я даю вам распечатку перечня вопросов, которые, как я полагаю, являются частью вашей перспективы ПА. Некоторые из них — из диаграмм, которые вы уже видели; другие я добавил сам, основываясь на опыте нескольких месяцев работы в «Фергюсон и Барделл». Возможно, что-то покажется вам ненужным, а что-то захочется добавить. Мы рассмотрим любое предложение, если оно аргументировано.

К нашей следующей встрече я прошу вас подготовить список всех первоочередных вопросов; в этом списке вы можете указать все, что считаете нужным. Но затем, помня о нашем 10-процентном подходе, вернитесь к этому перечню и пронумеруйте вопросы по степени важности. Другими словами, вы должны быть уверены, что включаете в список только главные вопросы, необходимые для продвижения компании вперед. Те вопросы, которые не кажутся столь важными, но, тем не менее, очень существенны, вы также можете включить.

Наконец, в первом приближении оцените те связи, которые вы увидите между перечисленными пунктами. Чем больше связей вы выявите, тем, по-видимому, более важен этот вопрос для **ПА.** Конечно, существует возможность того, что один-два процесса окажутся ненужными целиком. Надеюсь, это мы выясним по мере работы над проектом.

— Дэн, когда мы встречаемся в следующий раз? — спросил Дик, вытаскивая электронную записную книжку, чтобы зафиксировать дату и время.

- В следующий понедельник, через неделю. Этого времени должно хватить на подготовку первого варианта. Я бы хотел, чтобы вы обменялись этими документами к полудню четверга, чтобы таким образом к понедельнику все было просмотрено. Возражений нет? Хорошо, — Дэн замолчал и обменялся взглядом с каждым: — Я знаю, что вы прониклись важностью этой работы. Я с нетерпением жду, с чем вы придете в следующий раз. Жду вас в следующий понедельник, а до этого, если есть вопросы или нужна моя помощь, просто звоните или заходите,

— Да, чуть не забыл, — добавил Дэн. когда все уже собрались уходить. —Учтите, что мы не будем останавливать разработку проектов в ожидании окончания нашей с вами работы. Мы будем создавать новые системы и совершенствовать работающие. Другим группам придется согласовывать с нами архитектуру решений, чтобы наши взгляды на организацию производства не разошлись. Не забывайте, что нам придется постоянно совершенствовать архитектуру, чтобы достичь желаемого состояния, и в течение всего времени разработки проекта корректировать наше видение проблем.

ГЛАВА 2

Приложения масштаба предприятия

В этой главе

Одна из основных трудностей, с которыми сталкиваются ИТ-отделы, - необходимость количественной оценки успешности проекта приложения и его реализации. Способы определения этого параметра вызывают постоянные разногласия между экспертами, а в различных журналах бесконечно печатаются статьи с новыми, с каждым разом все более сложными методиками.

Мы начнем эту главу с описания функций современных производственных приложений и связанных с ними проблем. Затем сформулируем десять принципов успешного создания программных продуктов. Далее мы обсудим проектирование крупномасштабных распределенных производственных приложений и расскажем о необходимости снижения сложности таких проектов, По нашему мнению, упростить проект позволяет абстрагирование: похожие требования надо разделить на несколько абстрактных категорий. В качестве средства реализации этого метода мы предлагаем модель производственных приложений, разработанную компанией Microsoft. Мы обсудим каркас архитектуры приложений, представленный в модели разработки приложений MSF.

Эта глава основана на нашем опыте проектирования и реализации архитектуры приложений и на следующих материалах:

- Microsoft Solution Framework;
- Уокер Ройс (Walker Royce) «Software Project Management: A Unified Framework» (Addison-Wesley, 1998);
- Адель Голдберг (Adele Goldberg). Кеннет Рубин (Kenneth S. Rubin) «Succeeding with Objects» (Addison-Wesley, 1995);
- Уильям Браун (William Brown), РафаэльМальво (RaphaelMalveau), Хэйз Маккормик III (Hays McCormick III), Томас Моубри (Thomas Mowbray) «AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis» (John Wiley and Sons, 1998);
- Грэйди Буч (Grady Booch), ДжеймсРамбо (James Rumbaugh), Ивар Джекобсон (Ivar Jacobson) «The Unified Modeling Language User Guide» (Addison-Wesley, 1998);
- Мэри Киртлэнд (Mary Kirtland) «DesigningComponent-Based Applications» (Microsoft Press, 1998).

Изучив материал этой главы, вы сможете:

- ✓ описать основные характеристики производственных приложений;
- ✓ разобраться в современных методах создания архитектур приложений;
- ✓ выбрать архитектуру для типичного проекта производственного приложения;
- ✓ описать основные принципы разработки производственных приложений;
- ✓ перечислить характеристики модели разработки приложений MSF.

Характеристики приложений масштаба предприятия

Приложения масштаба предприятия отличаются размером: это *большие* бизнес-приложения. Как правило, они сложны, масштабируемы, распределены, часто разделены на компоненты и жизненно важны для работы всей организации. Их можно развернуть на различных платформах, представленных в корпоративных сетях, интрасетях и в Интернете. Главное для них — данные, поэтому такие приложения должны удовлетворять самым строгим требованиям к безопасности, администрированию и сопровождению. Короче говоря, это — сверхсложные системы.

При проектировании и разработке производственных приложений нужно учитывать множество различных требований. Кроме того, каждое решение по одному требованию влияет на другие, причем в таком влиянии непросто разобраться и еще труднее его предсказать. Таким образом, несоблюдение *всего одного* требования способно «провалить» весь проект.

Как и все современные программы, производственные приложения должны быть достаточно надежными и производительными, а также обладать интуитивно понятным и хорошо организованным пользовательским интерфейсом. Однако, помимо перечисленного, производственным приложениям присущи еще три особенности:

- Сложность — это многопользовательские многокомпонентные приложения, разработанные большим коллективом программистов и предназначенные для обработки огромного количества данных, параллельного выполнения множества процессов и интенсивного использования распределенных ресурсов. Естественно, что используемые в них алгоритмы очень сложны. Такие приложения, как правило, развертываются на различных платформах, взаимодействуют с другими приложениями и эксплуатируются в течение продолжительного времени.
- Ориентация на бизнес — цель таких приложений — решение конкретных задач бизнеса; они реализуют правила, процессы и объекты, присущие производственным процессам конкретной организации. Именно для этого их разрабатывают, развертывают и эксплуатируют.
- Важность — производственные приложения должны быть достаточно надежными, чтобы выдерживать непрерывную эксплуатацию. От них требуется исключительная гибкость в вопросах, касающихся масштабирования и развертывания, а также наличие средств обслуживания, мониторинга и администрирования.

Требования к приложениям постоянно ужесточаются, и это делает разработку производственных приложений еще более трудной. Из-за постоянной модернизации оборудования и программного обеспечения, а также наличия конкуренции, необходимо, чтобы бизнес-системы быстро реагировали на любые изменения и обладали исключительно высокой производительностью. С ростом требований организациям приходится наращивать долю автоматизированных процессов, быстрее создавать программное обеспечение, справляться с обслуживанием все большего числа пользователей и без задержек обрабатывать огромное количество данных.

Однако сложность технологий, используемых при создании корпоративных решений, и частота их изменения еще больше усложняют разработку. Поэтому при проектировании производственных приложений необходимо тщательно взвесить и сбалансировать множество требований к приложению, включая:

- его бизнес-цели;
- как быстро оно должно быть создано;
- бюджет разработки;
- количество разработчиков, тестеров и сопровождающего персонала;
- поддерживаемое количество пользователей;
- важность производительности и простоты работы с приложением;
- оборудование, на котором оно будет работать;
- где приложение будет развернуто:
- требования к защите данных;
- продолжительность эксплуатации.

Не разобравшись во взаимосвязях между этими сложными и часто конфликтующими требованиями, трудно определить, с чего начать. Поэтому прежде всего нужно выбрать способ разработки приложения и метод организации проектных работ, которые позволят наилучшим образом учесть множество требований.

Архитектура производственных приложений

Как мы уже упоминали в главе 1, один из принципов разработки современных приложений требует предварительного определения архитектуры, на основе которой создается система. В термин «архитектура приложения» разные люди вкладывают разное значение. Для одних — это наука, для других — искусство, для третьих — что-то еще. Основоположники современного учения об архитектуре приложений, в том числе Грэйди Буч в своем докладе «Software Architecture and the UML» на конференции UML World 1999 предложили считать, что это — набор основополагающих проектных решений относительно организации программной системы. К ним относятся:

- выбор структурных элементов и интерфейсов, образующих систему;
- поведение системы, определяемое взаимодействием этих элементов;
- объединение структурных и поведенческих элементов в более крупные подсистемы;
- стиль организации системы.

Чтобы получить более полное представление о том, что такое архитектура, следует объединить определение архитектуры, предложенное Бучем, и данное Мэри Шоу. В книге «Software Architecture: Perspectives on an Emerging Discipline» (Prentice Hall, 1996) она заметила,

что архитектурный стиль характеризует семейство систем с помощью структурной модели. В архитектурный стиль входят:

- словарь типов компонентов;
- набор ограничений, определяющий их объединение;
- одна или несколько семантических моделей, позволяющих определить свойства системы по свойствам ее частей,

К концепции архитектуры, разработанной Бучем и Шоу, можно добавить выделенные Бучем характеристики удачного архитектурного решения:

- гибкость;
- простота;
- реализуемость;
- четкое разграничение проблем;
- сбалансированное распределение ответственности;
- сбалансированность экономических и технологических ограничений.

Несомненно, постоянное обновление технологии, быстрый рост организаций и рынков, постоянно сокращающиеся бюджеты ИТ-отделов и невероятная популярность Интернета ставят новые проблемы и изменяют приоритеты разработчиков архитектур приложений. В этом разделе мы расскажем об этих трудностях и способах их устранения.

Повторное использование компонентов

Представив себе размер производственных приложений, вы поймете, почему их архитектура должна повторно использовать как можно больше компонентов. Повторное использование лежит в основе большинства приложений, поэтому не стоит считать его новой технологией или техническим прорывом, Оно применялось с самого возникновения программирования. Повторно применяемые объекты разделяют на следующие категории:

- сегменты исходного кода;
- библиотеки кода;
- библиотеки ресурсов;
- интерфейсы объектов;
- объекты, расширяемые объекты и т. д.

Принцип повторного использования не стоит ограничивать только исходным кодом. Он пригодится при стандартном процессе разработки сразу нескольких проектов или применении структур одного проекта в другом. Один из лучших шаблонов повторного использования подразумевает переход от внутренних приложений к коммерческим продуктам, таким, как библиотеки разработчика, различные сис-

темы и среды. Именно так были созданы COM и COM+, сетевые службы каталога и некоторые продукты Microsoft — например, ActiveX Data Objects (ADO), Microsoft Transaction Server (MTS), Microsoft Message Queuing Services (MSMQ), Exchange и SQL Server. Теперь эти продукты существенно облегчают труд разработчиков.

Однако затраты на создание повторно используемых объектных компонентов довольно велики, а дополнительные расходы на их длительное сопровождение для большинства некоммерческих организаций делают повторное использование дорогим удовольствием. Поэтому часто берет верх мнение «купить, а не разрабатывать».

На рис. 2.1 [он основан на диаграмме из книги Уокера Ройса «Software Project Management: A Unified Framework» (Addison-Wesley, 1998)], изображен график, показывающий затраты средств и времени на создание повторно используемых объектных компонентов.

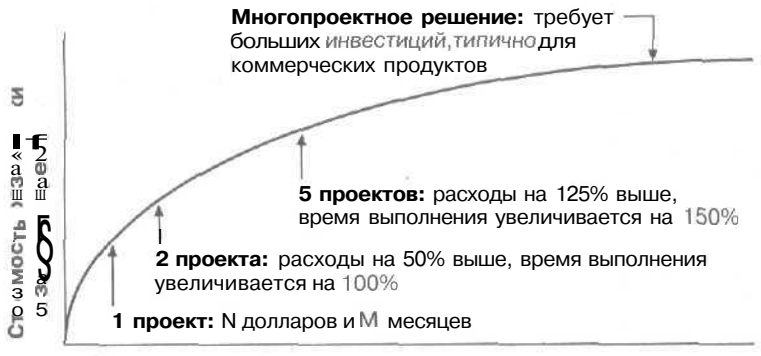


Рис. 2.1. Затраты средств и времени на создание повторно используемых объектных компонентов

Повторное использование возможно и на уровне приложения. Чуть позже в этой главе, а также в части 3 этой книги мы опишем *многоуровневую* модель архитектуры, закладывающую основу для повторного использования внутри приложений. Распределение сервисов по нескольким уровням приводит к появлению точек взаимодействия между разными частями приложения, которыми может воспользоваться любой компонент. Одно из достоинств такого метода заключается в том, что для изменения определенного сервиса не требуется модификация остальных составляющих приложения.

С появлением новых методов моделирования приложений возникла еще одна возможность — повторного использования проекта.

К алгоритмам и программной логике такая техника применяется уже долгие годы. Любой программист помнит, как он реализовал свой первый алгоритм поиска. Но ведь он не изобретал его, а просто воспользовался результатами интеллектуального труда других людей. Однако код используется повторно уже после создания архитектуры приложения. Разработка понятия *шаблона* сделала возможным повторное использование проектов и классов, что, кстати, не всегда влечет за собой повторное использование кода (это зависит от шаблона).

Размер приложения

Размер современных производственных приложений, которые в массе своей очень велики, необходимо контролировать. Самый эффективный способ уменьшить размеры приложения — сократить объем кода. При этом важно различать объем кода, написанного человеком, и общий объем кода. Применение современных средств разработки часто увеличивает число строк кода, а следовательно, и время выполнения. Однако современные компонентные модели разработки способны значительно уменьшить объем кода, который должны писать программисты, благодаря повторному использованию объектов и кода и применению высокоуровневых языков, объектно-ориентированных средств и механизмов автоматической генерации кода.

По мере распространения языков программирования высокого уровня число строк кода, написанных человеком, сокращается. Уокер Ройс в своей книге сравнил объем кода, необходимый для реализации одной и той же программы на разных языках:

> *«Эти значения показывают относительную выразительность различных языков, коммерческих компонентов и автоматических генераторов кода».*

Результаты этого сравнения представлены в табл. 2.1.

Табл. 2.1. Количество строк кода при применении разных языков и средств программирования

Язык программирования	Количество строк кода, созданного человеком
Ассемблер	1 000 000
C	400 000
Ada **83**	**220000**
C++ или Ada 95	175000
C++ или Ada 95 в сочетании с коммерческими системами и/или компонентами **сторонних разработчиков**	75 000

Замечание Объектно-ориентированные технологии и средства визуального моделирования также сокращают объем кода, который должен написать разработчик. Довольно долго обсуждалась эффективность сложных языков моделирования, например, *универсального языка моделирования* (Unified Modeling Language, UML), и новых систем визуального моделирования. Хотя такие системы способны сократить объем кода, их применение требует значительных вложений в обучение программистов.

Производительность приложения

При создании архитектуры приложения нужно учитывать требования к его производительности и предпринять все возможное, чтобы эти требования соблюдались как в настоящем, так и в будущем. Для этого приложение следует протестировать, а в случае неудачи найти способы повышения производительности. Требования к производительности редко удается удовлетворить без дополнительной оптимизации приложения, поэтому их нужно принимать во внимание как до разработки приложения, так и в ее ходе.

Для повышения производительности не нужно оптимизировать все компоненты системы — ведь пользователю важна эффективность приложения, а не каждого компонента в отдельности. На производительность распределенных приложений влияет множество факторов: оборудование, соединения, конфигурация системы, топология приложения и т. д., не зависящих от методов кодирования компонентов и приложений, При этом всегда возможен компромисс между простотой разработки, развертывания, сопровождения и эффективностью приложения. Главное при повышении производительности — с минимальными затратами найти и устранить «узкие» места в работе приложения.

При анализе производительности, особенно на ранних стадиях разработки проекта, редко удается точно воссоздать условия, в которых он будет эксплуатироваться.

Поэтому часто единственная возможность оценить производительность в реальной среде — экстраполировать результаты тестирования. Чем точнее тестовая среда воспроизводит эксплуатационную, тем ниже вероятность ошибки экстраполяции, однако затраты на адекватное моделирование реальной среды иногда превышают выгоды от снижения риска некорректности результатов тестирования. Для достижения требуемой производительности желательно выполнить только действительно необходимую работу. Таким образом, проверить эффективность приложения — значит уменьшить риск того, что

приложение не сможет поддерживать требуемый уровень производительности, а не повышать этот уровень до предельных значений.

Масштабируемость приложений

Чтобы удовлетворить требования пользователей, архитектура приложения должна описывать не только размер программы, но и расположение оборудования, или *топологию* приложения. Топология определяет тип, количество и конфигурацию серверов, на которых будет выполняться приложение. Ключевой принцип создания масштабируемых надежных приложений— независимость от топологии. Продукты, обладающие таким качеством, при увеличении нагрузки смогут работать на нескольких серверах без потери производительности. Кроме того, для повышения надежности системы приложение или его составляющие можно реплицировать в рамках кластера (*кластер* — это группа компьютеров, логически работающих как один). Выбор топологии определяется корпоративной политикой, существующей инфраструктурой, результатами тестирования производительности и ограничениями, связанными с реализацией.

Большинство распределенных приложений развертывают на уже существующей инфраструктуре, поэтому некоторые особенности топологии нужно определить заранее. Например, программный продукт может обращаться к уже имеющейся базе данных, расположенной на выделенном сервере, где, согласно корпоративным правилам, не допускается установка приложений. В таком случае архитектура приложения должна разрешать установку программного обеспечения на другом сервере и гарантировать наличие соответствующего коммуникационного протокола и соединения между серверами, а также возможность подключения к базе данных с необходимыми полномочиями.

Корпоративные правила оказывают большое влияние и на развертывание приложений, доступ к которым осуществляется через Интернет. Обычно серверы установлены за брандмауэром, предотвращающим несанкционированный доступ к конфиденциальной информации, причем иногда возможно несколько уровней таких брандмауэров, отличающихся ограничениями на допуск и протоколы. Web-серверы, серверы баз данных и серверы приложений могут быть защищены разными брандмауэрами. И в этом случае на всех компьютерах должны поддерживаться соответствующие коммуникационные протоколы, физические соединения и меры защиты.

Хотя существующая корпоративная инфраструктура и правила накладывают ограничения на масштабируемость приложения, результаты тестирования производительности способны повлиять на топологию. Например, в некоторых случаях необходимая производитель-

ность достигается на одном сервере, удовлетворяющем минимальным требованиям к производительности процессора, памяти и диска, но тестирование при этом показывает, что приложение будет работать эффективно только при *масштабировании,* то есть при добавлении в топологию новых серверов и распределения по ним нагрузки. Приложения можно масштабировать также, добавляя серверы бизнес-объектов и баз данных или разделяя доступ к информации.

Довольно часто такое *горизонтальное масштабирование* — оптимальный путь повышения производительности решения, поскольку оно не требует изменения кода приложения. К тому же затраты на новое оборудование, как правило, меньше расходов на разработку и тестирование, связанное с модификацией программного обеспечения, что особенно заметно в сравнении с полученным таким образом приростом производительности. Масштабирование увеличивает расходы на администрирование, но прибыль от роста эффективности системы окупает их.

Виды архитектуры

Одна из трудностей разработки архитектуры приложений — применение единой терминологии для обсуждения новых и применения существующих концепций. Эта сложность существует как на уровне разработки, так и на уровне организации. Сейчас мы опишем средства, позволяющие устранить их.

Универсальный язык моделирования

Как мы уже отмечали, один из важнейших факторов успешной разработки приложения заключается в способности передать всем участникам проекта информацию о процессах и бизнесе, а также всю техническую информацию. Этой цели служит *универсальный язык моделирования* (Unified Modeling Language, UML), основное назначение которого — облегчить визуализацию, определение, создание и документирование всех элементов программной системы.

UML основан на нескольких языках моделирования, получивших распространение в конце 80-х и в 90-х годах. Его версия 0.8 объединяла в себе методику Грэйди Буча и метод OMT Джеймса Рамбо, в версию 0.9 был добавлен метод OOSE Ивара Джекобсона, и наконец, версия 1.0 стала результатом работы большой группы исследователей, дополнивших этот язык новыми функциями. Дальнейшее совершенствование UML, проводившееся *группой объектного проектирования* (Object Management Group, OMG), окончательно утвердило его в статусе отраслевого стандарта.

Полное описание UML выходит за рамки данной книги, поэтому мы лишь перечислим его основные характеристики. UML можно разделить на 4 части.

- **Элементы моделирования** — эти элементы подразделяются на четыре группы:
  - структурные;
  - поведенческие;
  - группирующие;
  - другие.
- **Связи** — эти элементы также разделяют на четыре группы:
  - зависимость;
  - объединение;
  - обобщение;
  - реализация.
- **Расширяемые механизмы** — позволяют добавлять к моделям новые возможности.
- **Диаграммы** — представляют систему графически. В каждую диаграмму входит один или два типа представления: статический или динамический (табл. 2.2). Различные представления выражают разные точки зрения на проблему, причем для каждого из них существует специальная модель UML.

Табл. 2.2. Статические и динамические представления диаграмм UML

Представление	Модель	Характеристики
Статическое	Схемы использования	Создаются на ранних стадиях разработки для выявления необходимых пользователям функциональных возможностей. Их назначение — определить контекст системы, требования к ней, а также проверить архитектуру системы и создать тесты. Создаются, как правило, аналитиками или экспертами в данной предметной области
	Классы	Описывают словарь системы. Создаются один раз, но на протяжении всей разработки дорабатываются. Они необходимы для именования и моделирования концепций, взаимодействия составляющих и описания логических схем баз данных. Создаются системными аналитиками, дизайнерами и разработчиками

(*продолжение*)

	Объекты	Реализуют конкретные элементы и их связи с другими элементами. Создаются на стадии анализа и проектирования. Описывают структуры данных и объектов. Создаются, как правило, системными аналитиками, дизайнерами и разработчиками
	Компоненты	Описывают физическую структуру реализации. Создаются на стадии проектирования архитектуры до начала разработки, реализуя приоритет архитектуры. Они необходимы для организации и структурирования исходного кода и определяют физическую структуру базы данных. Создаются системными архитекторами и программистами
	Развертывание	Описывает топологию системы и оборудования. Необходимо для распределения компонентов и выявления непроизводительных составляющих системы. Создается как часть архитектурного процесса системными архитекторами, сетевыми инженерами и системными инженерами
Динамическое	Последовательность	Описывает динамическое поведение, характеризующее работу приложения с течением времени и его действия в типичных сценариях
	Сотрудничество	Описывает динамическое поведение на базе обмена сообщениями. Также представляет логику работы приложения и демонстрирует скоординированное поведение структур объектов
	Карта состояний	Описывает повеление, связанное с событиями. Может представлять жизненный цикл объектов, а также их реакцию на события. Часто используется при моделировании пользовательского интерфейса

(продолжение)

Представление	Модель	Характеристики
	Деятельность	Описывает поведение, связанное с действиями. Обычно используется при моделировании бизнес-процессов, их взаимодействия с приложением и основных операций

Для разработки архитектур приложений просто необходимо хорошо разбираться в UML. Советуем вам прочесть несколько книг, в том числе цитированную в начале главы «The Unified Modeling Language User's Guide» (Addison-Wesley, 1998), а также «Tried and True Object Development: Practical Approaches with UML» (Cambridge University Press, 1999), написанную Ари Яакси (Ari Jaaksi), Юха-Маркусом Аалто (Juha-Marcus Aalto), Ари Аалто (Ari Aalto) и Киммо Ватто (Kimmmo Vatto).

Шаблоны проектирования

Еще один способ описания сложных архитектур приложений основан на применении *шаблонов проектирования.* Этот метод предложили Кристофер Александер (Christopher Alexander), Сара Ишикава (Sara Ishikawa) и Мюррэй Сильверстайн (Murray Silverstein) в своей книге «A Pattern Language: Towns, Buildings, Construction» (Oxford University Press, 1977). В 1995 году Эрик Гамма (Erich Gamma), Ричард Хелм (Richard Helm), Ралф Джонсон (Ralph Johnson) и Джон Влиссидис (John Vlissides) описали принципы этого метода применительно к проектированию программного обеспечения. В книге «Design Patterns: Elements of Reusable Object-Oriented Software» (Addison-Wesley, 1995) они указали, что этот метод

> *«... определяет основные особенности общей структуры проектирования, что делает его полезным для создания повторно используемых объектно-ориентированных проектов».*

Сегодня термин «шаблон проектирования» известен любому разработчику. Каждый дизайнер объявляет свой архитектурный проект шаблоном. Определение этого термина, вошедшее в практику, можно найти в статье Дирка Риле (Dirk Riehle) и Хайнца Цуллигхофена (Heinz Zullighoven), напечатанной в журнале «Theory and Practice of Object Systems»:

> *«Шаблон — это информация, описывающая структуру удачного семейства проверенных решений некоторого класса проблем, возникающих в некоторых условиях».*

В этом определении указана одна из основных характеристик шаблона. Джеймс Коплин (James Coplien) в книге «Pattern Languages of

Program Design» (Addison-Wesley, 1995) отметил, что шаблоны обладают следующими свойствами;

- решают поставленную задачу;
- решение не очевидно;
- решение доказательно;
- описывают взаимосвязь.

Еще одна отличительная черта шаблона проектирования — он должен быть повторяющимся и подчиняться *«правилу трех»,* то есть его можно наблюдать как минимум на трех различных системах или приложениях, решающих данную проблему.

Шаблоны проекта бывают *порождающими* и *нейтральными.* Первые можно использовать при решении практических задач, вторые — только соблюдать. В своей книге «The Timeless Way of Building» (Oxford University Press, 1970) Кристофер Александер размышляет о различии этих терминов:

> *«...они отличаются в одном: шаблоны просто существуют вокруг нас, однако в нашей голове они динамичны. Они обладают силой. Они порождают решение. Они говорят нам, как решить задачу. А иногда они просто требуют, чтобы мы ими воспользовались. Каждый шаблон — это инструкция, рассказывающая, что мы должны сделать, чтобы создать объект».*

К счастью, уже идентифицированы шаблоны для многих обших задач. Некоторые из них имеют отношение к определенным сегментам производства, другие служат для решения специальных задач проектирования. Хотя организации способны определить шаблоны для своих приложений сами, мы бы порекомендовали для начала воспользоваться проверенными решениями. Применяя к проекту шаблон, следует первым делом определить технические и бизнес-требования к приложению. После этого не составит труда выбрать наилучший для вашего проекта шаблон (в этом смысле шаблоны проектов составляют основу повторного использования всей архитектуры системы). Хороший шаблон проекта не только содержит логическое обоснование решения, но и само это решение. Однако большинство шаблонов проекта ограничиваются созданием архитектуры, не предлагая реализацию задачи в виде кода.

Как и в любой еще не окончательно сформировавшейся области, при определении шаблонов возникают разногласия относительно элементов, включаемых в них. Несмотря на это, в разных шаблонах имеются общие элементы, которые изучил Брэд Эпплтон в статье «Patterns and Software: Essential Concepts and Terminology» *(www.enteracl.com/~bradapp/docs/patterns-intro.html).* Он выделил следующие элементы.

- **Название** — осмысленное слово или короткое словосочетание, определяющее шаблон и описываемую им структуру. В некоторых случаях шаблоны классифицируются не только по имени, но и по типу. Названия формируют словарь, необходимый для обсуждения концептуальных абстракций. Если у шаблона несколько названий, дополнительные названия заносят в документацию как псевдонимы.
- **Задача** — описание цели, которой необходимо достичь в данном контексте с учетом различных привходящих обстоятельств. Часто обстоятельства противодействуют достижению цели и конфликтуют друг с другом.
- **Контекст** — исходные условия, в которых формулируется задача и реализуется ее решение. Контекст можно рассматривать как начальную конфигурацию системы до применения к ней шаблона,
- **Силы** — описание всех действующих сил и ограничений, а также их взаимодействий и конфликтов друг с другом и с целями задачи. Действующие силы характеризуют сложность задачи и описывают возможные компромиссы.
- **Решение** — описание метода получения результата. Иногда показано в виде рисунков, диаграмм, чаще всего — текста, Решение должно описывать не только статическую структуру (форму и организацию шаблона), но и динамику его реализации, что помогает избежать ошибок.
- **Примеры** — одно или несколько применений шаблона в конкретном контексте. Предпочтительны несложные примеры для известных систем,
- **Окончательный контекст** — описывает состояние системы после применения шаблона, в том числе последствия (как положительные, так и отрицательные) и проблемы, связанные с изменением контекста. Если образец использован на промежуточной стадии решения крупной задачи или выполнения крупного проекта, результирующий контекст часто служит исходным контекстом других шаблонов.
- **Обоснование** — описание отдельных действий и шаблона в целом, демонстрирующих решение задачи и подтверждающее его соответствие принятым принципам и поставленным целям. Обоснование позволяет исследовать внутренние структуры и основные механизмы системы, разобраться в работе шаблона и понять, как и почему он работает и насколько он хорош.
- **Связанные шаблоны** — статические и динамические взаимосвязи между шаблонами, не выходящими за рамки одного языка или системы. Связанным шаблонам часто присущи общие черты, а их исходные и результирующие контексты обычно совместимы. Они

могут представлять разные решения одной задачи и зависеть друг от друга.

- **Примеры применения** — примеры использования шаблона в существующих системах. Доказывают правильность решения повторяющейся задачи. Часто используются в качестве учебных примеров.

Типичный шаблон проектирования кратко описывает все эти элементы и дает ясное представление об условиях и решении конкретного класса задач,

Антишаблоны

Шаблоны предоставляют полные решения технических и бизнес-задач. Они предназначены для создания новых проектов. Но существуют еще и *антишаблоны* — для исправления ситуации в задачах, уже обладающих неадекватным решением. Различить их помогут следующие определения;

> «*Шаблоны содержат решения, основанные на определенных критериях и действиях. Антишаблоны позволяют построить новое решение, когда существующее не работает*».

Таким образом, шаблоны служат для создания проекта «с нуля», а антишаблоны — для исправления неработающих решений. Эта концепция иллюстрируется на рис. 2.2, заимствованном из книги Уильяма Брауна (William Brown), Рафаэля Мальво (Raphael Malveau), Хэйза Маккормика III (Hays McCormick III) и Томаса Моубри (Thomas Mowbray) «AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis» (John Wiley and Sons, 1998).

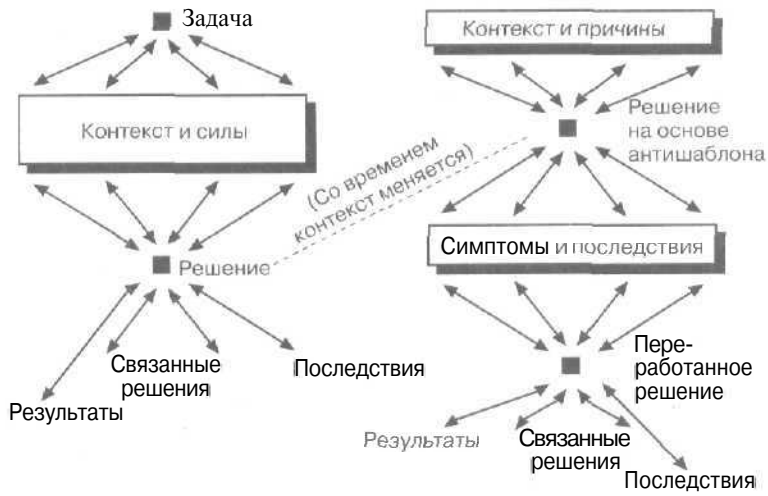


Рис. 2.2. **Шаблоны и антишаблоны**

Шаблоны и антишаблоны похожи по структуре и основополагающим принципам. Таким образом, антишаблоны:

- обладают структурированным описанием, которое служит языком общения для всех участников разработки приложения;
- удовлетворяют «правилу трех»;
- повторяемы.

Антишаблон можно сразу принять в качестве решения или разработать на его основе новый антишаблон. Кроме того, они помогают разработчикам изучать методы решения сложных задач.

Основные принципы разработки приложений

Разработка программного обеспечения за последние 20 лет сильно изменилась. С учетом накопленного за долгое время опыта обновились традиционные методы управления. Раньше считалось, что нужно разработать как можно больше функций за наикратчайшее время. и именно по этой причине современные программные продукты так сложны. Однако давление конкуренции, растущее влияние пользователей, постоянное изменение ситуации на технологическом рынке и непрестанно развивающиеся технологии привели к тому, что организации стремятся упростить процесс разработки. Главное же, что побудило их к этому, — провал огромного количества проектов. Сегодня цель разработчиков — создать как можно лучшую программу при минимуме затрат, а их девиз — «выпустить нужный продукт в нужное время»,

В этом разделе мы опишем основные принципы управления созданием программных продуктов, на которых основана разработка архитектур производственных приложений. Порядок, в котором они перечислены, не имеет особого значения, так как все они важны для успешного создания приложений. Благодаря им вы сможете выпустить нужный продукт в нужное время.

Соответствие целям бизнеса

Каждый проект должен быть оправдан, единственное же оправдание — соответствие задачам бизнеса. Таким образом, цели любого проекта определяются всеми его участниками.

Возврат инвестиций в проект не всегда выражается конкретными суммами. Проект может обладать стратегической ценностью, которую невозможно измерить деньгами. Однако и в этом случае цели проекта должны соответствовать задачам организации. При их изменении соответственно надо корректировать и цели проекта — по крайней мере, они не должны конфликтовать с новыми задачами бизнеса.

Поскольку следование бизнес-целям проекта — основа его успеха, члены проектной группы должны разбираться не только в техни-

ческой стороне проекта. Множество технически совершенных продуктов пылятся на полке из-за несоответствия запросам заказчика. Чтобы избежать этого, мы советуем проектной группе всесторонне изучать решаемую задачу с позиций бизнеса.

Ориентация на продукт

Этот принцип относится не к специфике вашего приложения — будь то коммерческое программное обеспечение, которое выпускает компания Microsoft или приложение для сотрудников вашей компании, — а к тому, что нужно рассматривать результаты вашей работы как продукт. Это значит, что следует концентрироваться на результате работы, а не на процессе. Конечно, процесс разработки тоже важен, но он лишь позволяет достичь поставленной цели, а сам по себе не имеет никакой ценности. Важно, чтобы все члены проектной группы чувствовали ответственность за свою работу, понимая, что их основная задача — создать приложение, а все остальное — второстепенно.

Приоритет архитектуры

В своей книге «Software Project Management» Уокер Ройс (Walker Royce) отмечает, что при управлении разработкой программного обеспечения наиболее важен принцип приоритета архитектуры. Его применение позволяет сбалансировать бизнес-требования, значимые для архитектуры решения, ожидаемую потребителями дату выпуска продукта и план выполнения проекта до получения окончательной информации о необходимых ресурсах. Принять соответствующие решения помогают прототипы и испытательные системы. В результате проектная группа должна не только согласовать со всеми заинтересованными сторонами набор функций приложения и дату его выпуска, но и создать общее описание проекта, удовлетворяющее все заинтересованные стороны.

Контекстно-зависимое проектирование

Современные приложения не изолированы — они зависят от потребностей бизнеса, других приложений, инфраструктуры сети и системы. Не только приложения взаимодействуют с информационными системами, но и системы взаимодействуют с приложениями. Как следствие, при проектировании приложений приходится учитывать необходимость их интеграции в информационную инфраструктуру предприятия и просчитывать возможное влияние программного продукта на информационную инфраструктуру и ее ответную реакцию.

Средства, используемые на разных фазах проекта

В современной индустрии программного обеспечения сложилась практика применения специализированных средств для каждой фазы

разработки. Например, для сбора бизнес-требований служат бумажные и электронные формы. Превосходным средством для формулирования требований и моделирования проекта является UML. А современные языки программирования, такие как Visual Basic, C++ и Java, образуют надежную платформу для разработки приложений.

Слагаемые успешного проекта

У каждого проекта — свои приоритеты и ограничения. Очевидно, что и цели, определяющие успех приложения, тоже отличаются. Поэтому, не выяснив намерений всех участников, нельзя определить направление развития проекта.

Коллективная работа

Современные производственные приложения слишком сложны, поэтому один человек создать их не может. Чтобы решить все необходимые проблемы, необходимо распределить их среди членов группы. Такой метод значительно повышает шансы на успех проекта.

Понимание целей проекта

Успех проекта основан на понимании членами проектной группы его целей и задач. Все должны уяснить их, согласиться с ними и сказать: *«Мы* все сделаем, и точка!» Кроме того, разработчики и заказчики должны выработать общий взгляд на проект, то есть четко определить его задачи. Часто у членов проектной группы и заказчиков есть сформировавшиеся представления о ходе проекта и графике его выполнения. Чтобы снизить возможность конфликта, проектная группа и заказчики должны согласовать представления о проекте. В конечном счете и та, и другая сторона заинтересованы в выпуске требуемого продукта в установленные сроки.

Периодическая демонстрация продукта заказчику

Проектная группа должна, соблюдая интересы заказчика, помочь ему выявить необходимые функции приложения и упорядочить их по степени значимости, а следовательно, и по порядку их создания и выпуска.

Чтобы уже на ранних этапах разработки у заказчика и пользователей сложилось представление о приложении, стоит применять прототипы, которые демонстрируют основные особенности продукта и отражают текущее состояние проекта. Прототипы помогают заказчику и исполнителям уточнить требования к продукту и приоритеты реализации функций. Помните, что изменение технического задания — не следствие жадности заказчика, а результат того, что он стал лучше понимать, что и когда может быть реализовано.

В ходе выполнения проекта архитекторы и разработчики могут создавать прототипы для проверки наиболее важных функций до написания больших сегментов кода. И наконец, альфа- и бета-версии

приложения, выпущенные на ранних стадиях разработки, позволяют собрать мнения пользователей о продукте. Особенно важна бета-версия приложения, с помощью которой изучаются проблемы интеграции продукта в инфраструктуру предприятия и его развертывания.

Управление рисками

Риск — это возможная потеря, в том числе падение качества продукта, рост затрат на разработку, отставание от графика или неудача в достижении целей проекта. Борются с рисками либо превентивно, либо по факту их проявления. Естественно, грамотное управление подразумевает превентивные меры по предотвращению рисков на всех этапах разработки. Управление рисками повышает осведомленность членов проектной группы о факторах риска и предоставляет средства по борьбе с ними всем участникам проекта.

Рассмотрим этапы процесса управления рисками:

- выявление риска;
- определение степени его влияния на проект;
- определение вероятности его возникновения;
- понимание того, как риск может проявиться в проекте;
- принятие превентивных мер по его предотвращению;
- выработка плана на случай реализации риска.

Если применять управление рисками с самого начала работы над проектом, вероятность их осуществления на поздних стадиях значительно сократится,

Компонентный подход

Наиболее полно компонентная разработка изложена в цитированной в начале главы книге Мэри Киртлэнд. Традиционно прикладные сервисы создавались с помощью *интерфейсов прикладного программирования* (Application Programming Interface, API), изучение которых весьма трудно.

Еще один популярный метод — объектно-ориентированные системы, но и он требует от программистов достаточно высокого мастерства. Кроме того, объектно-ориентированные системы, как правило, ограничены рамками одного языка программирования.

Компоненты — стандартная модель объединения сервисов, чтобы предоставить их услуги приложениям. Компоненты являются своего рода «черными ящиками», скрывающими подробности реализации. Сервисы таких компонентов доступны через открытые интерфейсы и общую программную модель. Еще одно преимущество такого метода — возможность взаимодействия компонентов, созданных с применением разных языков программирования и расположенных в разных местах.

Управление изменениями

Каждому проекту необходима система контроля за изменениями, Разработчики используют такие системы (системы контроля исходного кода) уже не один год, но они, как правило, выполняют только проверку некоторых документов проекта, например, планов, функциональных спецификаций, файлов с исходным и компилированным кодом. Такая проверка — только часть управления изменениями. Кроме того, система должна отслеживать воздействие изменений на компоненты, определять, кто и как вносил поправки, изучать их воздействие на требования к проекту и создавать новый стандарт проекта, учитывающий внесенные изменения. Этот процесс значительно уменьшает энтропию проекта,

Зависимость продукта от ожиданий и приоритетов пользователей

Независимо от того, насколько быстро продвигается работа над проектом, рынок, технологии, конкуренция и бизнес заказчика развиваются еще быстрее. Поэтому последовательный выпуск версий приложения позволяет проектной группе оперативно реагировать на изменение целей, графика и рисков проекта. Часто обновляя версии, разработчики получают возможность не только взаимодействовать с заказчиком, но и собирать пожелания относительно следующих выпусков продукта, основанных на использовании текушей версии.

Необходимо, чтобы основной набор функций появился в самой первой версии приложения, к которой в дальнейшем будут постепенно добавляться новые функции, формирующие окончательный продукт. Естественно, новые версии приложения должны отражать изменения требований к нему.

После выработки подхода и принятия основных проектных решений рекомендуется как можно быстрее начать выпуск версий.

Гибкость метода

Универсального метода для всех проектов не существует. Поэтому метод, применяемый на практике, должен быть достаточно гибким и учитывать потребности разных проектов. Кроме того, он должен быть масштабируемым, то есть годиться как для крупных, так и для мелких проектов.

По мере выполнения проектов на основе выбранного метода следует собирать удачные решения, повышающие качество процесса разработки и увеличивающие вероятность выпуска приложения в назначенное время.

Модель производственного приложения

Архитектура приложения представляет собой высокоуровневый план создания продукта, решающего конкретные бизнес-задачи. *Модель приложений масштаба предприятия* (Enterprise Application Model,

EAM) упорядочивает все требования к реализации таких приложений, разделяя их на шесть категорий (подмоделей). В табл. 2.3 перечислены задачи, формулируемые или решаемые каждой из этих подмоделей.

Табл. 2.3. Требования подмоделей модели производственного приложения

Подмодель	Требования
Бизнес	Бизнес-цели Стоимость разработки Возврат инвестиций Требуемые ресурсы Сроки Безопасность и сопровождение Инвестиции в существующую инфраструктуру Бизнес-правила
Пользователи	Пользовательский интерфейс Требования к простоте использования Обучение и документация Поддержка приложения Конфигурация пользовательских компьютеров и сетевых соединений
Логическая	Логическая структура приложения Моделирование объектов и данных Бизнес-объекты и сервисы Определение интерфейса
Технологическая	Разработка или повторное использование компонентов Средства разработки Платформы Системные технологии и технологии доступа к данным Технологии кластеризации, создания пулов и передачи сообщений
Разработки	Группа разработки Процесс разработки Управление проектом Контроль за исходным кодом Основные направления и результаты тестирования
Физическая	Физическая архитектура приложения Распределение и взаимосвязь компонентов Окончательный продукт на основе данных других подмоделей

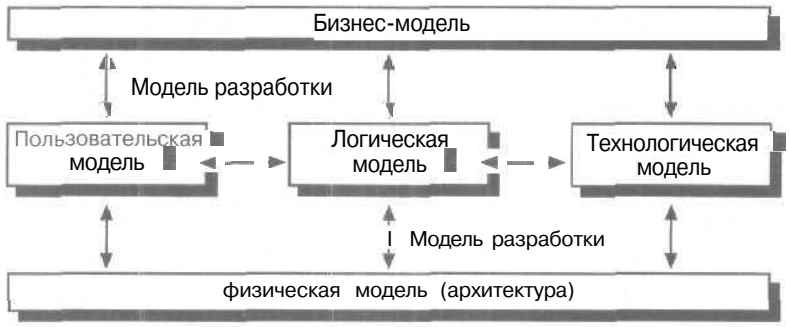


Рис. 2.3. Модель производственного приложения

На рис. 2.3 проиллюстрированы не только категории требований к производственному приложению, но и взаимосвязи между ними. Стрелками показано направление реализации успешного проекта: он начинается с определения бизнес-требований и заканчивается созданием физической архитектуры завершенной системы. Между этими двумя этапами реализуются пользовательские, логические и технологические требования, причем каждая из подмоделей зависит как от бизнес-требований, так и от других подмоделей. К тому же все модели вносят свой вклад в окончательную реализацию физической архитектуры. Остальные требования координируются с помощью *модели разработки* (Development Model), определяющей процессы и проектные группы, необходимые для работы над приложением.

- **Важность взаимосвязей** — понимание взаимосвязей между различными требованиями позволяет проектировать и разрабатывать приложение, не пренебрегая зависимостями между различными элементами проекта.
- **Важность требований** — требования каждой подмодели суммируются и составляют требования полной модели производственного приложения. От этой «суммы» зависит успех или провал проекта независимо от того, были ли осознаны ее составляющие во время разработки.
- **Важность подмоделей** — каждую подмодель можно рассматривать по отдельности, подобно компоненту программы. Все они обладают собственными наборами концепций, требований, методов, процессов, средств, хранилищ состояния и входных/выходных данных. В большинстве случаев результаты всех подмоделей становятся частью функциональной спецификации, которая, в свою очередь, определяет физическую архитектуру проекта и план его тестирования.

Эти соображения подсказывают, что процесс разработки должен быть итерационным и постепенным, а не линейным. При соблюде-

нии этих условий работа над каждым набором требований часто прерывается для изучения воздействия одной полмодели на все остальные. Такой метод помогает довольно рано выявить конфликтующие требования, а значит, внести коррективы *до того,* как это потребует серьезного пересмотра дизайна или переделки кода.

Проектирование с помощью модели производственного приложения

Модель производственного приложения — это средство проектирования, с помощью которого создается архитектура крупномасштабных приложений. Она организует все требования проекта в подмодели, описывая тем самым их взаимодействие. Кроме того, она позволяет сбалансировать конкурирующие требования. В результате создание приложения разбивается на небольшие этапы, выполняемые в любом имеющем смысл порядке, причем все решения плавно вливаются в общую архитектуру продукта.

Проектирование с помощью подмоделей

Большим достоинством модели производственного приложения считается возможность начать процесс проектирования с любого этапа. Это обусловлено тем, что порядок следования подмоделей определяется контекстом и приоритетами проекта, а не произвольной линейной последовательностью. В новых проектах предпочтительно двигаться сверху вниз, формируя из потребностей бизнеса подмодели среднего уровня, а из них — физическую архитектуру. Однако по некоторым причинам начальная точка разработки может сместиться — например, в случае повторного развертывания приложения или необходимости устранения проблем.

Изображенные на рис. 2.3 стрелки показывают, что требования и результаты соседних моделей влияют друг на друга. При создании функциональных спецификаций эти соображения служат для устранения проблем, вызванных конфликтующими требованиями.

Независимо от порядка работы над подмоделями процесс проектирования состоит из следующих стадий:

- в одной из подмоделей принимаются проектные решения;
- изучается воздействие этих решений на соседние подмодели, после чего вносятся необходимые поправки;
- предыдущий пункт повторяется до тех пор, пока изменения не будут изучены в масштабе всей модели производственного приложения.

Предположим, что пользовательская модель системы онлайновых продаж должна учитывать двадцатикратное увеличение числа покупателей в праздничные дни. Как это требование повлияет на осталь-

ные модели? На все эти вопросы вам поможет ответить рис. 2.3 и наш комментарий.

Примечание В приведенном ниже примере работы подмоделей используются концепции распределенных компонентных архитектур (например, создание пулов и масштабирование), описанных в следующих главах книги.

- **Бизнес-модель** — обосновывает необходимость реализации изменений. Следует определить степень влияния на группы сопровождения, эксплуатации и другие подразделения. Необходимо изучить влияние технологии на все системы. Если проект не интегрируется в бизнес-модель организации, продолжать работу над ним не стоит.
- **Пользовательская модель** — описывает отношение пользователей к проекту. Методы работы каждого приложения должны быть согласованы и понятны тем, для кого оно создавалось. Понятный и удобный интерфейс повышает эффективность работы с программным продуктом.
- **Логическая модель** — отвечает за инкапсуляцию сервисов, непосредственно взаимодействующих с пользователем, и за определение интерфейсов этих сервисов, что позволяет реализовать их в виде компонентов, которые можно организовать в пул.
- **Технологическая модель** — позволяет, основываясь на изменениях, внесенных в физическую, логическую, и бизнес-модели, создать помещаемые в пул компоненты и выбрать соответствующую технологию масштабирования для координации их работы (например, Microsoft Transaction Server).
- **Модель разработки** — необходима для организации групп, создающих и тестирующих компоненты, для распределения ресурсов и для создания графиков работы. В общей модели производственного приложения модель разработки служит «клеем», объединяющим подмодели. Она определяет ресурсы, требующиеся для проекта, и распределение работ между подмоделями.
- **Физическая модель** — предназначена для быстрого масштабирования приложения с целью обслуживания большого числа пользователей без падения производительности системы. Выполнение данного требования, как правило, требует применения кластеризации или пулов, благодаря чему удается увеличить число экземпляров критичных компонентов простым добавлением серверов.

Все важные результаты подмоделей — бизнес-требования, пользовательские требования, алгоритмы, реализующие бизнес-правила,

схема базы данных, информация о производительности и т. п. — заносятся в спецификации. Этот документ постоянно изменяется и обновляется во время цикла разработки. На каждой стадии проекту особенно важно «провести» любое изменение через все подмодели в соответствии со стрелками между ними и выяснить воздействие этих изменений на модель приложения в целом. До окончательного утверждения спецификаций этот процесс повторяется не один раз,

Сбалансированность подмоделей

Физическая модель производственного приложения — конечный результат процесса проектирования. Ее формируют, исходя непосредственно из требований остальных подмоделей. Таким образом, сам процесс проектирования представляет собой нахождение баланса между всеми подмоделями.

Например, уменьшение затрат на администрирование системы, заложенное в бизнес-модели, в некоторых случаях выливается в использование в технологической модели обработки данных на централизованных серверах. Однако, несмотря на это, достаточная эффективность пользовательской модели достигается только при локальной обработке информации на рабочих станциях. Естественно, при создании физической модели необходим компромисс между этими требованиями.

Разобравшись во взаимодействии подмоделей, вы сможете определить порядок принятия решений и предотвратить болезненные изменения проекта, отрицательно влияющие на требования, описанные в других подмоделях.

Примечание Так как модель разработки относится ко всему производственному приложению, в ее взаимодействии с другими подмоделями нельзя выделить «типичные» составляющие. Таким образом, все решения, касающиеся проекта и его реализации, должны быть учтены в модели разработки.

Б следующих разделах мы расскажем о взаимодействии подмоделей модели производственного приложения. Для каждой из них указаны:

- основные элементы проекта приложения, за которые отвечает подмодель;
- основные взаимосвязи с другими подмоделями;
- пример влияния сети Интернет на подмодель;
- разделы в этой книге, где также рассказано об этой подмодели.

Примечание Приложения масштаба предприятия, доступные в Интернете, появились сравнительно недавно. По этой причине многие

разработчики знакомы с ними хуже, чем с технологиями компонентного проектирования. Поэтому в каждом разделе мы кратко упоминаем влияние Интернета на рассматриваемую подмодель. Это не значит, что Интернет — лучшая платформа для производственных приложений, хотя иногда это действительно оптимальный выбор.

Бизнес-модель

Бизнес-модель описывает цели организации и причины инвестиций в разработку проекта. Ниже перечислены вопросы, которые решаются на этом этапе.

- Какие бизнес-требования предъявляются к проекту?
- Какие бизнес-задачи он решает?
- Какие инвестиции обеспечат максимальную отдачу?
- Насколько быстро будет выполнен проект?
- Каковы затраты на развертывание приложения?
- Какие платформы оно должно поддерживать?
- Сколько пользователей будут одновременно работать с приложением?
- Насколько важна зашита данных?
- Насколько надежным должно быть приложение?
- Когда потребуется замена или модернизация приложения?
- Как быстро должны учитываться новые бизнес-правила и требования пользователей?

В идеале создание архитектуры производственного приложения начинается с рассмотрения требований бизнес-модели. Если другие подмодели будут реализованы вне контекста бизнес-модели, готовый проект может оказаться не способен решить задачи, поставленные организацией.

Взаимодействие бизнес-модели с другими подмоделями

На рис. 2.3 показано, что бизнес-модель напрямую взаимодействует с пользовательской, логической и технологической моделями. Все эти взаимодействия описаны в табл. 2.4.

Табл. 2.4. Взаимодействие бизнес-модели с другими моделями

Подмодель	Влияние бизнес-модели	Пример
Пользовательская	Описывает пользователей приложения, их квалификацию и конфигурацию их компьютеров	Приложение для неопытных пользователей должно быть снабжено интуитивно понятным интерфейсом и подробной документацией

(продолжение)

Логическая	Устанавливает правила управления ресурсами, что отражается в логических бизнес-правилах	Бизнес-правила показывают, как подсистемы продаж, учета и доставки справятся с новыми заказами
Технологическая	Описывает или ограничивает технологию, необходимую для реализации бизнес-требований	Наличие, например, электронной коммерции может побудить организацию к переходу на интернет-технологии

Примечание Описанные взаимодействия носят двусторонний характер. Например, стоимость и пропускная способность модемов могут ограничить число пользователей корпоративного интернет-приложения. В этом случае технологическая модель заставит пересмотреть цели и требования бизнес-модели.

Влияние Интернета на бизнес-модель

Приведем примеры воздействия Интернет на бизнес-модель:

- появляется возможность создания программного обеспечения, работающего на разных платформах (с некоторыми ограничениями в совместимости и надежности);
- возрастают расходы на тестирование приложения на разных платформах, виртуальных машинах и браузерах;
- появляется возможность без особых затрат публиковать огромный объем информации; благодаря снижению стоимости сопровождения увеличивается доступность продукции и объем продаж;
- появляется возможность создания «слегка интерактивных» приложений, использующих функции Web-серверов (при этом сокращается стоимость развертывания приложения);
- возникает необходимость в защите организации от попыток взлома серверов и проникновения через Интернет вирусов и небезопасных компонентов.

Пользовательская модель

Пользовательская модель предназначена для изучения пользователей приложения. На этом этапе рассматриваются следующие вопросы:

- кто пользуется приложением; какова квалификация пользователей; каковы типичные сценарии работы с приложениями;
- каковы требования пользователей к последовательности выполнения задач, удобству работы с приложением, обучению, производительности системы и взаимодействию с внешними приложениями и данными;

- кто будет пользоваться приложением — служащие предприятия, которые при необходимости будут вынуждены смириться с некоторыми неудобствами работы с приложением, или же покупатели, которые могут не купить программу даже при наличии минимальных недостатков;
- какой объем документации необходим пользователям; будут ли они терпеливо читать все инструкции до конца;
- требуется ли техническая поддержка продукта; будут ли пользователи платить за нее;
- сколько пользователей будет обращаться к приложению одновременно;
- насколько мощны компьютеры пользователей; какова пропускная способность их сетевых соединений;
- какой уровень зашиты необходим и что требуется для его обеспечения.

Примечание В крупных средах возможно разделение пользовательской модели на четыре подмодели: удобства использования, документации, поддержки и защиты.

Взаимодействие пользовательской модели с другими подмоделями

На рис. 2.3 показано, что пользовательская модель напрямую взаимодействует с физической, логической и бизнес-моделью, а также с технологической моделью. Эти взаимосвязи описаны в табл. 2.5.

Табл. 2.5. **Взаимодействие пользовательской модели с другими моделями**

Подмодель	Влияние пользовательской модели	Пример
Бизнес	Количество пользователей, их квалификация и конфигурация их компьютеров определяют затраты на обучение и поддержку	Приложение для заказа товаров должно работать достаточно быстро, чтобы обеспечить эффективность работы служащих. Для реализации этого требования могут понадобиться инвестиции в оборудование или инфраструктуру
Логическая	Понимание пользователем его задач определяет логическую структуру функциональных возможностей	Если при оформлении заказа надо проверять кредитный баланс клиента, соответствующая функция должна быть помещена в отдельном

(продолжение)

		асинхронно выполняющемся компоненте
Технологическая	Требования пользователей определяют технологию, необходимую для их реализации	Элементы пользовательского интерфейса приложения, доступного через Интернет, стоит реализовать с применением кросс-платформенной технологии
Физическая	Число пользователей и их местонахождение влияет на архитектуру компонентов	Если число обращений к сервисам приложения постоянно изменяется, сервисы надо разворачивать с помощью легко масштабируемой архитектуры, например, с применением пулов или очередей

Влияние Интернета на пользовательскую модель

Приведем примеры воздействия Интернета на пользовательскую модель:

- доступ к огромному количеству профессионально подготовленных HTML-документов;
- взрывной рост числа пользователей производственных приложений, причем пользователей малоквалифицированных;
- опасливое отношение пользователей к Web-приложениям, получающим доступ к ресурсам персонального компьютера пользователя (синдром «Большого Брата»);
- утрата некоторых средств организации пользовательского интерфейса;
- замедление отклика приложений из-за медленных сетевых соединений;
- ненадежный доступ к информации и приложениям с помощью относительно неустойчивых интернет-соединений.

Логическая модель

Логическая модель определяет бизнес-объекты и применяемые к ним правила. Изучаемые на этом этапе проблемы зависят от деятельности организации, для которой разрабатывается данный программный продукт. Рассмотрим, на какие вопросы дает ответ логическая модель.

- Когда покупателю положена скидка?
- Когда следует пополнять запасы товаров?
- Какие налоги с продаж должны взиматься с покупателей?

Логическая модель состоит из двух относительно независимых подмоделей.

- **Логическая модель данных** — отвечает за описание бизнес-объектов системы (например, продуктов, заказчиков и заказов), а также устанавливает правила для таких объектов — например, «Все новые заказы должны быть представлены в таблице «Заказы» и в таблице «Отгрузка»».
- **Логическая модель объектов** — отвечает за правила и алгоритмы, работающие с объектами данных, распределяет эти правила по интерфейсам и классам и определяет взаимодействие объектов друг с другом в процессе выполнения требований моделей высокого уровня. Кроме того, эта модель определяет соответствие объектов архитектуре многоуровневых приложений,

Взаимодействие логической модели с другими подмоделями

На рис. 2.3 видно, что логическая модель напрямую взаимодействует с физической, пользовательской и технологической моделями, а также с бизнес-моделью. Все эти взаимодействия описаны в табл. 2.6,

Табл. 2.6. Взаимодействие логической модели с другими моделями

Подмодель	Влияние логической модели	Пример
Бизнес	Определяет методы инкапсуляции бизнес-правил	В приложении часто меняющиеся ставки налогов и правила надо реализовать в отдельных, легко модифицируемых компонентах
Пользовательская	Определяет логические объекты, на которые воздействуют пользователи при решении различных бизнес-задач	Для решения типичных задач, связанных с продажами, в приложении должны быть предусмотрены логические объекты «клиент» и «заказ»
Технологическая	Определяет сервисы и функциональные возможности используемой технологии	Компонент проверки банковского баланса должен использовать технологию, способную взаимодействовать с соответствующими институтами
Физическая	Определяет сервисы, которые потребуются при развертывании	Компонент, отвечающий за перевод денежных средств, должен устанавливаться в

(продолжение)

выбранной физической архитектуры	подготовленной инфраструктуре, поддерживающей надежные протоколы **зашиты** данных

Влияние Интернета на логическую модель

Приведем примеры воздействия Интернета на логическую модель.

- В связи с наметившейся тенденцией переноса бизнес-приложений на мощные интернет-серверы растет спрос на многоуровневые приложения, в которых бизнес-логика отделена от пользовательского интерфейса.
- Корпорации и независимые производители программного обеспечения стремятся реализовывать продукты в форме объектов, которые просты в сопровождении, повышают реактивность систем и могут повторно использоваться.

Интернет оказывает влияние на физическую модель, поэтому он практически не затрагивает логическую модель организаций, уже перешедших на объектно-ориентированные распределенные приложения.

Технологическая модель

Технологическая модель определяет технологии, способные решить поставленные задачи. Она используется для поиска, приобретения или создания необходимых технических ресурсов, удовлетворяющих требованиям проекта. Рассмотрим вопросы, на которые дает ответ технологическая модель.

- Под управлением каких операционных систем работают рабочие станции, серверы и СУБД?
- Какие сетевые протоколы должны поддерживаться?
- Какие технологии защиты данных необходимы?
- Какая технология обеспечения масштабируемости будет применяться?
- Как поддерживать целостность базы данных в условиях многокомпонентных распределенных транзакций?
- Как обрабатывать массовые асинхронные запросы?
- С помощью какой технологии осуществлять доступ к традиционным системам?
- Какой метод реализации пользовательского интерфейса оптимален?
- Какая технология доступа к удаленным базам данных потребуется?
- С помощью какой технологии реплицировать данные?
- Как следует вести работу над продуктом, принимая во внимание потребности бизнеса (графики, ресурсы, квалификация персона-

ла и затраты на проект) и возможности имеющейся технологии (объекты и компоненты, доступ к данным, пользовательский интерфейс, распределенные транзакции, зашита данных, а также средства проектирования, программирования и отладки)?

- Какие средства понадобятся для создания, отладки и развертывания приложения?

Примечание Написанный и скомпилированный код становится частью технологической модели точно так же, как и элементы управления или сервисы операционной системы, привлеченные из внешних источников. Исходный код проекта также является частью технологической модели, а результаты тестов удобства использования относятся к пользовательской модели.

Взаимодействие технологической **модели** с другими подмоделями

На рис. *2.3* показано, что технологическая модель напрямую взаимодействует с физической, логической и бизнес-моделью, а также с пользовательской моделью. Эти взаимодействия описаны в табл. 2.7,

Табл. 2.7. Взаимодействие технологической модели с другими моделями

Подмодель	Влияние технологической модели	Пример
Бизнес	Предлагает физическую реализацию бизнес-задач приложения	Новые интернет-технологии породили новые бизнес-возможности
Логическая	Реализует логическую структуру приложения в виде физических компонентов	Бизнес-правила инкапсулируются в компонентах, образующих инфраструктуру приложения
Пользовательская	Реализует функции приложения с помощью технологии, которая соответствует квалификации пользователя, конфигурации его компьютера и способу подключения к сети	Если интернет-приложение предоставляет пользователям доступ к удаленной базе данных, группа разработчиков должна приобрести средства программирования и отладки многопользовательских асинхронных распределенных приложений
Физическая	Предоставляет технологию для создания в рамках выбранной фи-	Если интернет-приложение должно запускаться под управлением разных опера-

(продолжение)

зической архитектуры приложения, обладающего всеми необходимыми функциональными возможностями	ционных систем на разных серверах, группе разработки, возможно, потребуется использовать несколько языков программирования

Влияние Интернета на технологическую модель

Приведем примеры воздействия Интернета на технологическую модель:

- появление языка гипертекстовой разметки **HTML,** языка Java, протокола HTTP, интернет-серверов, брандмауэров, цифровой подписи компонентов и Web-обозревателей;
- потребность в производительных серверах приложений, где размещаются компоненты, реализующие бизнес-правила;
- значительное расширение списка платформ, на которых необходимо проводить тестирование и обеспечивать поддержку приложения.

Модель разработки

Модель разработки описывает процесс разработки и ресурсы, используемые при создании приложения, и является «клеем», связывающим все подмодели. Рассмотрим, на какие вопросы дает ответ модель разработки.

- Кто и над какой частью проекта должен работать?
- В каком порядке следует решать задачи (логическое проектирование объектов, тестирование удобства использования, проектирование базы данных, программирование и тестирование и т. д.) и как оценить объем проделанной работы?
- Сколько разработчиков занято в проекте и какова их квалификация?
- Что нужно сделать для оптимизации разработки и повышения качества конечного продукта?
- Как управлять проектом и координировать его с другими проектами?
- Насколько важны повторное использование и компонентный подход? (Ответ на этот вопрос влияет на координацию с другими проектными группами, на важность стандартов, на время, затраченное на проектирование, и на стоимость архитектуры обшего назначения.)
- Каким образом информация о ходе проекта будет передаваться остальным разработчикам и тестерам?
- Как развернуть приложение в производственной среде?
- Как осуществлять администрирование?
- Как проводить развертывание модернизированных версий?

Модель разработки производственной архитектуры может использовать еще две модели MSF. Однако их разрешается применять и по отдельности, поскольку они являются равноправными.

- **Модель проектной группы** — связана с организацией, координацией и управлением группами разработчиков; обсуждается в главе 3.
- **Модель процесса разработки приложений** — позволяет решать проблемы, связанные с планированием, составлением графиков и этапами процесса разработки; обсуждается в главе 4.

Взаимодействие модели разработки с другими подмоделями

На рис. 2.3 показано, что модель разработки объединяет все подмодели, координируя их требования и направляя процесс разработки. Поскольку эта модель должна учитывать все проектные решения, для нее нет «типичных» форм взаимодействия.

Влияние Интернета на модель разработки

Приведем примеры воздействия Интернет на модель разработки:

- возросла важность серверных приложений (без пользовательского интерфейса);
- приложения стали сложнее, и, как следствие, увеличились требования к ресурсам, повысилась сложность тестирования и отладки многопользовательских распределенных постоянно эксплуатирующихся приложений;
- основной акцент при разработке сместился в сторону компонентов, не зависящих от местоположения, а также способов их создания, координации и совместного использования;
- расширились обязанности программиста — теперь он не только создает приложения, но и публикует их;
- сильно вырос спрос на универсальную, масштабируемую, основанную на компонентах архитектуру приложений масштаба предприятия.

Физическая модель

Физическая модель описывает физические ресурсы, необходимые для выполнения требований других подмоделей. Помимо этого, она реализует процесс разрешения конфликтов, возникающих при столкновении интересов разных подмоделей. Окончательная физическая модель производственного приложения называется физической архитектурой. Посмотрим, на какие вопросы дает ответ физическая модель.

- Как использовать физические ресурсы компьютеров, пропускную способность сетей, сетевые протоколы, базы данных, компоненты, операционные системы и приложения сторонних производителей для выполнения бизнес-требований (таких как масштабируемость и надежность)?

- Каким образом осуществить миграцию?
- Какой способ доступа к ресурсам наиболее эффективен (на локальном компьютере или через Интернет)?
- Как выполнить требования к удобству использования приложения и его производительности в условиях медленного соединения с ЛВС или ГВС, периодически отключающихся сетевых серверов и ненадежных каналов связи с Интернетом?

Архитектура приложения

Архитектура приложения представляет собой концептуальное описание структуры программного продукта. Как показано на рис. 2.4, каждое приложение разделено на три уровня: пользовательский, прикладной и уровень данных. Кроме того, во всех приложениях имеется презентационный код, код обработки бизнес-правил и данных, а также код, отвечающий за хранение информации. Отличаются же архитектуры приложений только организацией кода, индивидуальной для конкретного приложения.

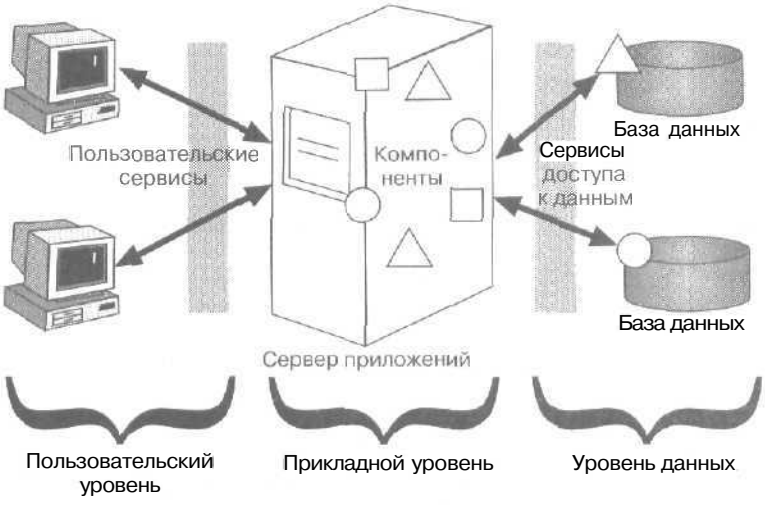


Рис. 2.4. **Физическая модель и ее уровни**

Многие современные приложения построены на базе *двухуровневой архитектуры «клиент/сервер»*. При этом клиентский процесс отвечает за обработку и отображение данных. Сами же данные централизованно хранятся на серверах, к которым по мере необходимости подключаются клиенты. Часто время использования приложения ограничивается длительностью такого соединения.

Клиент-серверные приложения хорошо работают только в контролируемых средах, когда число пользователей предсказуемо (это необходимо для выделения ресурсов). Однако эта архитектура становится неэффективной, если пользователей очень много или их число неизвестно. Помимо этого, масштабируемость таких систем невысока, так как количество непосредственных соединений с сервером ограничено. Невелики и возможности повторного применения, поскольку пользователи привязаны к определенному формату базы данных. А так как клиентское ПО содержит блок обработки информации, обший объем приложения довольно велик. (Такой тип клиентских приложений иногда называют *«толстым» клиентом.)* В этом случае при изменении алгоритмов обработки данных приходится устанавливать новые приложения на каждый клиентский компьютер.

Можно добиться небольшого улучшения путем перемещения прикладных алгоритмов и блоков обработки информации на серверы данных (например, с помощью хранимых процедур Microsoft SQL Server). Такую архитектуру иногда называют *2,5-уровневой.* Масштабируемость подобных приложений немного лучше, но все равно мала для выполнения требований распределенных приложений с большим числом клиентов. Кроме того, возможность повторного использования остается на прежнем невысоком уровне.

Масштабируемость и степень повторного использования можно заметно улучшить, добавив в архитектуру приложения третий уровень. В такой *многоуровневой архитектуре* (ее также называют N-уровневой) все уровни — пользовательский, прикладной и уровень данных — логически разделены (рис. 2.5).

Опишем функции каждого уровня.

- Пользовательский **уровень** — отображает данные и, что также возможно, позволяет пользователю редактировать их. Существуют два основных типа интерфейса: «родной» (реализуемый средствами подсистемы пользовательского интерфейса операционной системы) и на основе Web. Первые используют сервисы операционной системы — например, в Microsoft Windows применяются API Win32 и элементы управления Windows. Web-интерфейсы основаны на HTML или XML (Extensible Markup Language), в результате они могут отображаться любым обозревателем на любой платформе. Пользовательский уровень подробно описан в главе 7.
- Прикладной уровень — здесь реализованы бизнес-правила и ограничения на данные. И хотя его сервисы используются презентационным уровнем, он не привязан к какому-либо клиенту — сервисы прикладного уровня доступны любому клиенту. Бизнес-правила выражаются в форме прикладных алгоритмов, корпоратив-

ных правил и т. д, (например, «Пользователю предоставляется 10%-ная скидка на рекламу, размещенную до вторника» или «Все заказы. поступившие из Москвы, облагаются налогом с продаж, равным 4%»). Ограничения на данные гарантируют точность и целостность хранимой информации (например, «Заголовок заказа должен содержать как минимум одну подробную запись»). Бизнес-правила обычно реализуются отдельным модулем на централизованном сервере, что лает возможность доступа к нему сразу нескольким клиентам. (Такая изоляция кода соответствует принципам управления программным обеспечением, описанным выше в этой главе.) Прикладной уровень подробно обсуждается в главе 8.

Рис. 2.5. **Многоуровневая архитектура приложений**

* **Уровень данных** — прикладной уровень не знает, как и где хранится обрабатываемая им информация, В этом вопросе он полагается на сервисы доступа к данным, выполняющие всю работу по получению и передаче данных. Сервисы доступа к данным также реализуются в виде изолированных модулей, «знающих» о месте хранения информации. Таким образом, если хранилище перемещено или изменен его формат, потребуется обновить только сервисы доступа к данным. Каждый модуль доступа к данным, как прави-

ло, отвечает и за целостность хранилища (например, реляционной базы данных). Более подробно технологии доступа к данным описаны в главе 9. Для многоуровневых приложений в качестве хранилища информации подходят простые системы управления базами данных (СУБД), необходимые для обслуживания данных в таблицах и быстрой выборки информации (например, с помощью индексов). Хранилища данных, в частности, SQL Server, Exchange Server, In Memory Database (IMDB) и Microsoft Access, также описаны в главе 9.

Действительно ли логическая архитектура входит в состав физической модели? Это не опечатка? Дело в том, что логическая архитектура объединяет несколько физических; другими словами, приложения конструируются в виде логических сетей, связывающих потребителей и поставщиков услуг (сервисов). Сервисами называют программные модули, выполняющие определенные операции.

Содержащееся в названии архитектуры слово «многоуровневая» не подразумевает обязательное размещение на разных компьютерах. Главная задача этой архитектуры — обеспечить высокую масштабируемость приложений, а им необходимо совместно использовать различные ресурсы (например, базы данных), Но вместо непосредственного обращения к серверу данных клиентское приложение передает свой запрос прикладным сервисам. Как показано на рис 2.6 и 2.7, один экземпляр бизнес-сервиса способен обслуживать несколько клиентов, таким образом сокращая потребление ресурсов и улучшая масштабируемость системы.

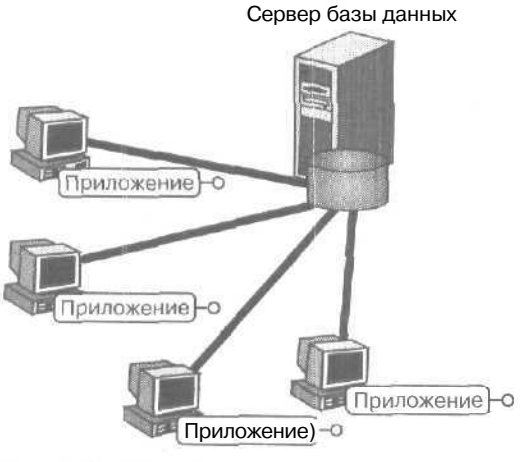


Рис. 2.6. Клиент-серверная архитектура

А так как бизнес-сервисы не осуществляют непосредственное управление данными, их можно реплицировать, дабы увеличить число поддерживаемых клиентов. Довольно часто сервисы реализуются независимо от клиентских приложений, что повышает их гибкость и увеличивает возможности их повторного использования в других программных продуктах. Инкапсулировав прикладную логику с помощью открытых интерфейсов, разработчики создают основу для повторно используемых сервисов, которые легко объединить в другой последовательности при разработке нового приложения. Кроме того, не составит труда обновить необходимые функции при изменении бизнес-требований, причем этот процесс никак не повлияет на использующие их приложения. Очевидно, что такой метод сокращает затраты на управление и реализацию изменяющихся требований.

Рис. 2.7. Многоуровневая архитектура

Многоуровневая архитектура приложений помогает разработчикам работать и с традиционными системами — например, «обернув» дос-

туп к таким системам в оболочку, состоящую из прикладных сервисов, доступа к данным и хранилища данных. При этом клиентские приложения должны получать доступ только к прикладному уровню, а не к самим традиционным системам. Если такая система будет изменена или заменена, понадобится только обновить программную оболочку,

Взаимодействие физической модели с другими подмоделями

На рис. 2.3 показано, что физическая модель напрямую взаимодействует с пользовательской, логической и технологической моделями. Эти взаимодействия описаны в табл. 2.8.

Табл. 2.8. **Взаимодействие физической модели с другими моделями**

Подмодель	Влияние физической модели	Пример
Пользовательская	Делает приложение доступным пользователям	Физическая архитектура должна поддерживать различные конфигурации компьютеров и масштабироваться в зависимости от количества пользователей и их требований
Логическая	Определяет границы, в которых логические конструкции можно адекватно отобразить на физическую архитектуру	В приложении, обрабатывающем онлайновые транзакции, должен быть компонент, взаимодействующий с инфраструктурой управления транзакциями
Технологическая	Ограничивает технологию, которая может поддерживаться физической инфраструктурой	Интернет-приложение, доступ к которому открыт для всех пользователей, должно применять самую общую технологию отображения элементов пользовательского интерфейса, работающую на разных платформах и в разных обозревателях

Влияние Интернета на физическую модель

Приведем примеры воздействия Интернета на физическую модель:

- сильно возрастает важность масштабируемых архитектур, поддерживающих неизвестное или неограниченное число пользователей;
- необходим доступ к функциям приложения с различных клиентских платформ;
- возрастает асинхронное использование распределенных ресурсов:

- становится возможным создание новых типов приложений, как чисто информационных (простой HTML), так и активных (интерактивный HTML, ActiveX и Java);
- появление традиционных приложений, внедренных в Web-страницы;
- появляются нетребующие установки пользовательские интерфейсы (HTML и обозреватели);
- необходима установка брандмауэров, разделяющих внутренние ЛВС и Интернет;
- необходима установка приложений по требованию (ActiveX-компоненты и Java-апплеты);
- необходимо упаковывать распространяемые компоненты и подписывать их цифровой подписью.

Модель приложения MSF

Модель приложения MSF описывает методы проектирования и разработки программного обеспечения на базе многоуровневой архитектуры, основанной на сервисах. Приложение в этой модели рассматривается как сеть совместно работающих, распределенных и повторно используемых сервисов, решающих поставленные бизнес-задачи. Сервисы приложений — это единицы прикладной логики, включающие методы, реализующие определенные операции, функции и преобразования. Доступ к этим сервисам осуществляется с помощью опубликованных интерфейсов, соответствующих спецификациям. Как уже говорилось в главе 1, в этой модели приложение использует три типа сервисов: пользовательский, прикладной и сервис данных. В результате становится возможной параллельная разработка разных элементов, обеспечивается более полное использование возможностей технологии, упрощается сопровождение и поддержка, а также расширяются возможности развертывания и масштабируемость приложения. Эти три сервиса могут находиться в любом месте среды — как на одной рабочей станции, так и на серверах и клиентах, размещенных по всему миру.

Пользовательские сервисы

Пользовательские сервисы — это алгоритмы, отвечающие за пользовательский интерфейс. Интерфейс не всегда отображается на экране монитора, он может быть программным — ведь зачастую с приложением работает не только человек, но и другое приложение. Пользовательские сервисы изолируют представление информации от структуры пользовательского интерфейса приложения.

Прикладные сервисы

Прикладные сервисы контролируют выполнение бизнес-правил. Они обеспечивают транзакционную целостность и преобразуют данные в информацию с помощью бизнес-правил.

Сервисы данных

Сервисы данных — самый нижний уровень, отвечающий за манипулирование данными. Они также отвечают за целостность и согласованность данных. Кроме того, с их помощью достигается независимость реализации приложения от местонахождения и структуры хранилища информации. Большинство сервисов данных способны находить, создавать, считывать, обновлять и удалять информацию.

Резюме

В начале главы мы описали архитектуру приложений масштаба предприятия. Затем мы обсудили различные методы ее создания, привели основные характеристики современных приложений и принципы управления программными продуктами. Затем мы рассказали о модели производственных приложений, состоящей из шести подмоделей; моделей бизнеса и разработки, а также пользовательской, логической, технологической и физической моделей. Мы описали каждую из этих подмоделей. В заключение мы познакомили вас с моделью приложения MSF, которая описывает трехуровневую архитектуру приложений, основанную на пользовательских и прикладных сервисах, а также на сервисах данных.

Закрепление материала

1. Что такое архитектура приложения? Перечислите характеристики удачного архитектурного решения.
2. Какими способами можно описать архитектуру программного продукта?
3. Охарактеризуйте производственные приложения.
4. Что такое шаблоны и антишаблоны?
5. Назовите любые пять из основных принципов управления созданием программного обеспечения.
6. Перечислите шесть подмоделей модели производственного приложения.
7. Что такое модель приложения MSF?

ГЛАВА 3

Проектные группы

В этой главе

Итак, кто же непосредственно выполняет проект? Мы постараемся ответить на этот вопрос, рассказав об обязанностях членов группы и о создании проектных групп в контексте модели проектной группы MSF. Мы начнем с поиска кандидатов на роль руководителей и расскажем о шести основных направлениях работы группы. Затем мы познакомим вас с ролями участников проекта и распределением обязанностей среди членов группы, Далее мы расскажем о способах анализа требований к проекту с точки зрения его участников, методах масштабирования группы в соответствии с его значением и размером и, наконец, перечислим требования к руководителям и проектной группе, которые позволят добиться эффективного управления проектом.

В этой главе использован свой собственный опыт проектирования и разработки архитектуры приложений, а также следующие материалы:

- материалы консультативной службы Microsoft;
- Джим Маккарти (Jim McCarthy) «Dynamics of Software Development» (Microsoft Press, 1995);
- Стив Макконнелл (Steve McConnell) «Rapid Development: Taming Wild Software Schedules» (Microsoft Press, 1996) и «Software Project Survival Guide» (Microsoft Press, 1998);
- Барри Бём (Barry Boehm) «Software Engineering Economics» (Prentice Hall, 1981).

Изучив материал этой главы, вы сможете:

- ✓ разобраться в проблемах иерархической модели ресурсов;
- ✓ назвать причины необходимости использования модели проектной группы MSF;
- ✓ изучить роли и обязанности членов группы в контексте модели проектной группы;
- ✓ разобраться в особенностях масштабирования проектной группы для крупных и мелких проектов;
- ✓ осознать опасности совмещения нескольких ролей;
- ✓ охарактеризовать эффективного руководителя;
- ✓ описать средства повышения эффективности проектной группы;
- ✓ выявить области, в которых группе необходимо дополнительное обучение.

Модель группы и иерархическая модель

Большинство программистов с любовью и нежностью вспоминают свою первую программу, выводящую на экран монитора приветствие «Hello, World», — свою первую удачу в программировании. Но к действительности эта радость не имеет отношения — сложность промышленных приложений и систем такова, что процесс их разработки стал практически неуправляемым. Кроме того, их развертывание на сотнях компьютеров, расположенных в разных местах, значительно раздвигает границы процесса разработки.

Как мы уже упоминали в главе 1, один человек не способен создать приложение масштаба предприятия. Ни один разработчик просто не удержит в голове все требования к сисгеме и варианты проекта. Поэтому сегодня разработкой промышленных систем занимаются проектные группы, и все обязанности распределяются среди членов группы,

Работа в коллективе отличается определенными сложностями. Прежде всего, коллектив хочет знать «кто начальник». Ответ на этот вопрос дает иерархическая модель организации, определяющая начальников и подчиненных. Однако, если в современных производственных средах один менеджер проекта отвечает за все тонкости разработки и принимает все важные решения, возникает множество проблем, ведущих к провалу проекта. Иерархическая модель грешит множеством недостатков:

- нехваткой информации;
- невозможностью учесть все особенности проекта;

- отсутствием полноценной связи между всеми участниками проекта, так как вся информация идет в одном направлении — вверх по иерархии, к главному менеджеру;
- трудностью освоения новых технологий, необходимых при создании кросс-платформенных приложений;
- сложностью расстановки приоритетов.

Кроме того, опыта одного человека чаще всего недостаточно для быстрого решения задачи и для интеграции приложения в существующую инфраструктуру.

В организациях, построенных на основе иерархической модели, затруднен обмен информацией — в этой модели он, по определению, осуществляется через посредников. Вся информация иерархических групп «фильтруется» тремя или четырьмя менеджерами, что значительно повышает вероятность утери самого важного. Часто такое отсеивание идей происходит при прохождении сообщения от разработчика, непосредственно занимающегося проектом, к высшему руководству. Естественно, некоторые участники «выпадают» из процесса, что снижает эффективность их труда и повышает вероятность провала проекта.

Дабы сгладить недостатки иерархической модели, в проектной группе предусматривается распределение обязанностей руководителя между членами коллектива. При этом за проект отвечает не один человек, а все члены группы — каждый за свой участок.

Модель группы не определяет структуру коллектива с точки зрения отдела кадров. Ведь в такую разностороннюю группу привлечены ресурсы из разных отделов организации. Задача модели проектной группы — определить цели проекта и распределить обязанности. Руководители каждого направления с помощью выделенных им ресурсов выполняют возложенную на них часть работы. Обязанности ролей определяются работой над проектом, а не деятельностью «штатной единицы». При этом руководители направлений выполняют свои обычные функции: составляют график выплаты премий, распределяют отпуска и контролируют эффективность работы сотрудников, Начальник может оценить степень участия и эффективность работы сотрудников в проектной группе, но это — прерогатива менеджера конкретного сотрудника, а не проектной группы,

И у коллективного подхода есть недостатки:

- разрозненная связь с внешними источниками информации;
- несогласованное представление о разных сторонах проекта;
- несогласованность личных планов членов группы;
- отсутствие опыта, снижающее эффективность коллективной работы.

В этой главе мы опишем метод, который позволяет избежать подобных проблем.

Обязанности членов группы

MSF — не готовое решение, а каркас, который можно адаптировать для нужд любой организации. Один из элементов этого каркаса - *модель проектной группы*. Она описывает структуру группы и принципы, которым надо следовать для успешного выполнения проекта.

Хотя модель группы разработчиков весьма конкретна, при знакомстве с MSF ее нужно рассматривать в качестве отправной точки. Разные коллективы реализуют этот каркас по-разному, в зависимости от масштаба проекта, размеров группы и уровня подготовки ее членов.

Чтобы проект считался удачным, следует решить определенные задачи;

- **удовлетворить требования заказчика—** проект должен выполнить требования заказчиков и пользователей, иначе ни о каком успехе не может быть и речи, возможна ситуация, когда бюджет и график соблюдены, но проект провалился, так как не выполнены требования заказчика;
- **соблюсти ограничения** — разработчики проекта должны уложиться в финансовые и временные рамки;
- **выполнить спецификации, основанные на требованиях пользователей** — спецификации — это подробное описание продукта, создаваемое группой для заказчика; они представляют собой соглашение между проектной группой и клиентом и регулируют вопросы, касающиеся приложения, в основе этого требования лежит принцип «сделать все, что обещано»;
- **выпустить продукт только после выявления и устранения всех проблем** — не существует программ без дефектов, однако группа должна найти и устранить их до выпуска продукта в свет, причем устранением ошибки считается не только ее исправление, но и, например, занесение в документацию способа ее обхода; даже такой способ устранения проблем предпочтительнее, чем выпуск приложения, содержащего невыявленные ошибки, которые в любой момент могут преподнести неприятный сюрприз пользователям и разработчикам;
- **повысить эффективность труда пользователей —** новый продукт должен упрощать работу пользователей и делать ее более эффективной. Поэтому приложение, обладающее массой возможностей, применять которые сложно или неудобно, считается провальным;
- **гарантировать простоту развертывания и управления —** эффективность развертывания непосредственно влияет на оценку пользо-

вателем качества продукта, например, ошибка в программе установки может создать у пользователей впечатление, что и само приложение небезгрешно, от проектной группы требуется не только подготовить продукт к развертыванию и гладко провести его, но и обеспечить пользователей поддержкой, организовав сопровождение приложения.

Примечание На проект влияет множество факторов, некоторые из которых создают дополнительные ограничения. Когда проектная группа рассматривает на совещаниях цели проекта и график его создания и выпуска, следует убедиться, что проект окупится, то есть удастся выпустить нужный продукт, не превышая запланированных расходов.

Для достижения этих целей в модели проектной группы выполняемые задачи распределяются по шести ролям: менеджмент продукта, менеджмент программы, разработка, тестирование, обучение пользователей и логистика. Люди, выполняющие конкретную роль, должны рассматривать проект со своей «колокольни» и обладать необходимой для этого квалификацией.

Шесть ролей модели проектной группы, подробно описанные в этой главе, связаны с шестью целями проектной группы, что проиллюстрировано в табл. 3.1. Все эти цели важны для успеха проекта в целом, и поэтому все роли равноправны. В этой модели нет руководителя всего проекта — есть группа людей, знающих, что нужно делать, и делающих это.

Табл. 3.1. Цели и роли

Цель	Роль
Удовлетворение требований заказчика	Менеджер продукта
Соблюдение ограничений проекта	Менеджер программы
Соответствие спецификациям	Разработчик
Выпуск только после выявления и устранения проблем	Тестер
Повышение эффективности труда пользователя	Инструктор
Простота развертывания и постоянное сопровождение	Логистик

Как же начать работу над проектом, не зная, сколько времени на это потребуется, сколько проект будет стоить и каких результатов ожи-

дать? Ответить на эти вопросы поможет модель проектной группы MSF, обсуждаемая в следующих разделах этой главы, и модель процесса разработки, описанная в главе 4. Обе эти модели предлагают составлять расписания и графики «снизу — вверх», в проектировании придерживаться подхода «сверху — вниз» и, кроме этого, распределять ответственность за результаты работы между членами группы. На первый взгляд может показаться, что эти рекомендации не отвечают на поставленный вопрос. Однако на самом деле они позволяют сократить риски на ранних стадиях проекта. Приняв на этих стадиях правильное решение, вы избежите серьезных изменений в дальнейшем, что намного сократит время выполнения проекта и затраты на него.

Модель проектной группы

Работа над проектом включает множество разных видов деятельности и изучение требований с разных точек зрения, поэтому распределение основных задач по нескольким ролям повышает шансы на успех проекта. Как видно из рис. 3.1, в MSF определены шесть таких ролей, которые и составляют модель проектной группы. У каждой из ролей свои обязанности, выполнение которых и обуславливает удачу проекта.

Рис. 3.1. Роли в модели проектной группы

Обычно каждую роль исполняют несколько человек, один из которых считается руководителем, именно он на совещаниях с другими членами группы представляет свое направление.

Внимание! Коллективная ответственность не должна выливаться в коллективную безответственность. Это особенно важно в модели проектной группы, поскольку выполнение каждой ролью всех своих обязанностей — обязательное условие успеха проекта.

В проектную группу должны входить:

- опытные руководители;
- инициативные сотрудники, способные принимать решения и нести ответственность за свое направление работы,

Их задача:

- сконцентрироваться на выпуске продукта:
- выработатьобщее представление о проекте.

Для эффективной работы проектной группы требуется соблюдение следующих правил в отношениях между людьми:

- **доверие** — делает действия людей согласованными, при этом все следуют принципу «мы делаем то, что обещали сделать»;
- **уважение** — люди признают способности других, следуя правилу «каждый из нас необходим нашей группе»;
- **согласие** — все должны знать и поддерживать цели проекта и верить в его успех — «мы завершим проект, и точка»;
- **ответственность** — люди должны ясно понимать цели проекта, свои обязанности и чего от них ожидают — *«я* сделаю свою работу, вы — свою, и к четвергу мы построим наш дом».

Подытожив эти характеристики, мы получаем главный принцип — «обязанность каждого — выпустить нужный продукт в нужное время».

Менеджер продукта

Менеджер продукта должен вовремя реагировать на потребности заказчика. Его главная задача — сформировать общее представление о поставленной задаче и о том, как ее решать. Он должен ответить на вопрос «Зачем мы делаем все это?» и убедиться, что все члены группы знают и понимают ответ на него.

Основная цель этой роли — удовлетворение требований заказчика. Для этого менеджер продукта выступает представителем заказчика в группе разработчиков и представителем группы у заказчика. (На этом этапе важно понимать разницу между заказчиком и пользователями: заказчик платит за создание продукта, а пользователи с ним работают.) Кроме того, менеджер продукта вместе с менеджером программы должны прийти к компромиссному решению относительно функциональных возможностей продукта, сроков его разработки и финансирования проекта.

Как представитель заказчика, менеджер продукта отвечает за выполнение требований заказчика, создание бизнес-сценариев, формирование обшего представления о проекте у группы и клиента, а также проверяет, удовлетворяют ли решения группы потребностям заказчика.

Как представитель группы, менеджер продукта отвечает за взаимодействие с заказчиком и управление его ожиданиями. При этом он

проводит брифинги с клиентом и главными менеджерами, устраивает встречи с пользователями и демонстрации продукта.

Менеджер продукта сообщает группе предварительную дату выпуска продукта, устанавливаемую заказчиком. Однако в действительно успешных проектах графики составляются «снизу— вверх». При этом группа сама определяет возможную дату выхода приложения, которую менеджер продукта сообщает клиенту.

Примечание Заказчик (лицо, финансирующее проект) всегда ожидает от проекта чего-то большего. Естественно, ему хочется, чтобы все его пожелания воплотились в жизнь. И здесь основная роль отводится менеджеру продукта, следящему за ожиданиями клиента. Таким образом, работа менеджера часто определяет успех или провал проекта,

---

Менеджер продукта должен постоянно общаться с заказчиком. выясняя: «Мы сделали то, что обещали?» Допустим, в рамках проекта предполагалось реализовать пять операций за две недели и 10 000 долларов. Однако за полторы недели было потрачено 7 000 долларов, а реализованы только три функции. Как вы думаете, покажется ли этот проект заказчику успешным? Однако квалифицированный менеджер продукта мог бы ответить: «Был проведен второй этап переговоров, и мы договорились с заказчиком относительно самых важных операций, которые можно реализовать, не превышая финансирование и не выходя из графика. **То,** что мы обещали, мы выполнили. Сейчас же мы начинаем новый проект, в рамках которого реализуем две оставшиеся функции и еще две дополнительные». Таким образом, менеджер продукта пришел с заказчиком к компромиссу, продемонстрировав процесс принятия решений и проинформировав клиента об изменившихся рисках и проблемах. (Обычно трудности возникают при определении затрат на проект, даты его выполнения и функциональных возможностей, а также при выделении ресурсов.) Если все участники проекта смогут сказать: «Да, мы сделали то, что обещали», между заказчиком и проектной группой воцарится доверие и уважение.

Примечание Основная причина неудачи многих проектов — непонимание ожиданий заказчика. При любых — запланированных или нет - изменениях проекта, его ресурсов или даты выхода продукта ожидания заказчика, пользователей и проектной группы должны быть скорректированы.

Группа менеджмента продукта представляет интересы заказчика и помогает ему определить необходимые функции приложения и их

приоритеты. Отложить работу над какой-либо функцией вправе только заказчик, а переговоры по этим вопросам ведет менеджер продукта. За выполнение и составление графиков отвечает менеджер программы, поэтому в его власти изъять определенные функциональные возможности, чтобы текущая версия появилась в срок. Если менеджер программы решит сократить набор функциональных возможностей, чтобы не выбиться из графика, он обязан сообщить об этом менеджеру продукта и заказчику, которые вправе согласиться или нет. В последнем случае заказчик и менеджер продукта должны согласовать изменение даты выпуска продукта. Кроме того, они могут увеличить финансирование с целью расширения состава группы. Затем менеджер программы определяет, как и на что потратить дополнительные деньги и поможет ли это выполнить план. Окончательное решение о финансировании принимают заказчик и менеджер продукта.

Внимание! Дополнительные ресурсы не всегда ускоряют разработку, так как возрастают издержки, связанные с обменом информацией и управлением.

Руководителя группы менеджмента часто называют «борцом за продукт». Как правило, это один из топ-менеджеров организации. Остальные члены группы менеджмента продукта должны хорошо разбираться в структуре организации, ее стратегии и бизнес-целях.

Примечание Если проектная группа состоит из консультантов или приглашенных из других организаций людей, менеджер продукта должен учитывать интересы как заказчика, финансирующего проект, так и заказчика, нанявшего консультантов и оплачивающего их услуги. Как правило, консультационная группа требуется для ознакомления организации с MSF и обеспечения взаимодействия заказчика и проектной группы.

Менеджер программы

Задача менеджера программы — вести процесс разработки с учетом всех ограничений. Руководитель этого направления должен понимать разницу между понятиями «руководитель» и «начальник». В своей книге «Dynamics of Software Development» бывший директор отдела программного менеджмента Microsoft Джим Маккарти пишет:

> *«Запомните, что ваша цель — повысить авторитет каждого члена группы, а не подавить его своим».*

Главная обязанность менеджера программы — выполнить все стадии разработки так, чтобы нужный продукт был выпущен в нужное время. Он координирует деятельность других членов группы. И хотя иногда ему придется подгонять своих сотрудников, он не должен и помышлять о диктаторском стиле управления.

Главный менеджер программы составляет график проекта на основе информации, полученной от остальных членов группы. Он координирует этот график с руководителями всех подгрупп. Буферным временем проекта также управляет менеджер программы. Если отдельные части работы выполняются раньше графика или, наоборот, задерживаются, именно менеджер программы должен выяснить, как это скажется на проекте, и изменить график.

Для выполнения своих обязанностей менеджер программы должен отлично разбираться в деловой стороне проекта и иметь ясное представление о технологиях, необходимых для его выполнения. Естественно, что от руководителя отдела программного менеджмента требуется коммуникабельность и талант организатора.

Программный менеджер компании, где только начинают применять MSF, должен полностью понимать все модели и процессы, повышающие эффективность труда членов группы. И хотя умение руководить нужно всем ролям, особенно это качество важно для менеджера программы. Его авторитет должен стать непререкаемым еще до начала проекта у всех его участников, включая бизнес-отделы и руководство организации. Как правило, группа менеджмента программы управляет ресурсами, используемыми другими ролями, а также составляет и координирует расписания совещаний.

Так как менеджер программы отвечает за набор функциональных возможностей приложения, одна из его обязанностей — определить их набор, необходимый для выполнения требований заказчика. Критичные для проекта требования выявляет менеджер продукта, но набор реализующих их функций определяет менеджер программы. Кроме того, менеджер программы дает заказчику рекомендации, касающиеся дальнейшего развития продукта. Отдел программного менеджмента отвечает за функциональные возможности, изложенные в спецификациях (здесь описано, что будет создано) и в главном плане проекта (определяет, как это будет сделано). Он также координирует работу над этими документами. Тем не менее каждый участник проекта вносит свой вклад в функциональные спецификации и в план проекта.

Внимание! Важно, чтобы менеджер программы понимал, что каждый член группы разбирается в своих обязанностях намного лучше его,

Менеджер программы должен полностью положиться на опыт остальных сотрудников и только следить за соблюдением всех условий и ограничений.

Менеджер программы отвечает и за бюджет проекта, объединяя требования к ресурсам всех членов группы в единый план расходов. Естественно, его задача — не только разобраться в этих требованиях, но и контролировать реальные затраты, сравнивая их с запланированными. Кроме того, менеджер программы должен регулярно сообщать о состоянии работы всем основным участникам проекта. Ведь одним из симптомов неудачного продукта является отсутствие информации о состоянии проекта, пока деньги на него не кончились. Важно понимать, что решение о дополнительном финансировании может потребовать изменения других сторон проекта, а следовательно, его будет принимать менеджер программы совместно с заказчиком.

Разработчик

Разработчики знакомят остальных членов группы с применяемыми технологиями и собственно создают продукт. В качестве консультантов они предоставляют исходные данные для проектирования, проводят оценку технологий, а также разрабатывают прототипы и тестовые системы, необходимые для проверки решений и сокращения рисков на ранних стадиях процесса разработки. Чтобы создать продукт определенного качества, разработчикам не следует замыкаться на создании кола, они должны участвовать и в решении прикладной задачи. Они творят не ради творчества, а для реализации требований заказчика. Часто, чтобы полностью разобраться в проекте, приходится создавать прототипы, а чтобы протестировать новую технологию, — испытательные системы, помогающие принять окончательное решение относительно архитектуры приложения. Этим также занимаются разработчики.

Как программисты разработчики отвечают за низкоуровневое проектирование и оценку затрат на реализацию продукта. В большинстве организаций несколько основных разработчиков занимаются и архитектурой приложения. Как правило, это требуется на ранних стадиях проекта, когда уточняются детали функциональных спецификаций и описывается взаимодействие продукта с внешними системами.

Разработчики сами оценивают сроки своей работы. Такая концепция MSF — создание графиков ответственными за выполнение конкретного участка членами группы — называется *составлением расписания «снизу — вверх»*. Она позволяет выпустить нужный продукт в нужное время за счет уточнения графиков и повышения ответственности за выполнение работы в запланированные сроки.

Разработчики отвечают и за техническую реализацию проекта — в основном на фазах создания логической и физической модели, обсуждавшейся в главе 2. На этих стадиях их задача — определить методы реализации функциональных возможностей и заданной архитектуры, а также оценить сроки выполнения этой работы. Заметим, что разработчики не выбирают функции — они только решают, как их реализовать.

Кроме того, на стадии «Планирование» разработчики решают, какое влияние окажет на проект добавление или удаление некоторых функций. Разработчики не участвуют в заключительной стадии проекта — развертывании продукта, однако они должны тесно сотрудничать с логистиками на стадии установки приложения.

Внимание! Руководителю группы разработки не рекомендуется совмещать несколько ролей. Будучи программистом, он должен делать то, что у него лучше всего получается — писать качественный код.

Тестер

Задача тестеров — испытание продукта в реальных условиях, дабы определить, что в продукте работает и что не работает, и нарисовать таким образом точный «портрет» приложения. Естественно, для проведения тестов нужно отлично разбираться и в требованиях пользователей, и в том, как их удовлетворить.

Тестеры разрабатывают стратегию, планы, графики и сценарии тестирования, которые позволяют убедиться, что все ошибки выявлены и исправлены до выпуска приложения. Ошибкой называют любую проблему, из-за которой продукт не выполняет свои функции. Ею может оказаться и ошибка в коде, называемая «жучком», и отклонение от спецификаций, заданных менеджером программы, и недоработки в документации, подготовленной группой обучения пользователей.

Примечание Важно различать тестирование и *общую оценку качества* (Total Quality Assurance, TQA). Тестирование касается только проекта — точнее, его технической стороны. Проверку качества организует руководитель, ответственный за качество, — он выясняет соответствие продукта корпоративным, правительственным и другим стандартам.

Нельзя совмещать должности тестера и разработчика. Разделение этих обязанностей:

- гарантирует независимую проверку того, что продукт действительно выполняет все требования;
- повышает качество продукта за счет конкуренции между группами.

Хотя проверяют качество продукта только тестеры, за выпуск хорошего продукта отвечают все члены проектной группы.

Внимание! За качество кода отвечают разработчики. Отделение тестирования от разработки не снимает с них эту обязанность.

Контроль изменений

При работе над проектом необходимо контролировать изменения. Этим должны заниматься все участники группы, но чаще всего в полном объеме этим приходится заниматься именно группе тестирования. Разработчики уже не один год применяют системы, подобные Microsoft Visual SourceSafe, но они осуществляют только контроль версий. Такие системы пригодны для отслеживания версий любых документов, функциональных спецификаций, исходного и скомпилированного кода, но они только частично регистрируют изменения, В качестве примера можно привести сохранение версий документа в Microsoft Word — эта функция не гарантирует, что вы внесете верные правки и представите в качестве результата правильную версию. Таким образом, для управления изменениями необходимо:

- создать эталонный документ;
- определить изменяемые элементы;
- определить влияние изменений на существующие системы, процессы или документы;
- определить метод реализации изменений;
- назначить человека, который внесет изменения;
- определить влияние изменения на условия выполнения проекта, его бюджет, график и политику;
- получить одобрение изменений (скажем, у руководителя проекта);
- внести изменения;
- создать новый документ, в котором изменение учтено.

Внимание! Изменения лучше анализировать и принимать порциями — иначе на это уходит слишком много времени.

Прочие обязанности

Некоторые важные обязанности тестеров часто упускают из виду. К ним относятся:

- **уведомление об ошибках и их отслеживание** — тестовая группа отвечает не только за управление изменениями, но и за систему выявления ошибок и информирования о них;
- **сборка** продукта — в группе должен быть человек, ответственный за сборку (компиляцию) продукта, и часто такой «главный сбор-

шик» является тестером, он может использовать только код, хранящийся в системе управления версиями; эту рутинную работу удается автоматизировать с помощью сценариев, однако необходимо проверять правильность сборки;

- **выявление и контроль рисков** — это обязанность всех членов группы, менеджер программы должен разработать метод контроля — например, с помощью электронных таблиц Microsoft Excel; тестеры отвечают за работу программы в реальных условиях, поэтому их задача — представить группе анализ рисков с этой точки зрения.

Внимание! План контроля риской, созданный или пересмотренный за пять минут до начала совещания, совершенно бесполезен. Процесс контроля рисков должен быть непрерывным и постоянным — только тогда гарантируется качество продукта и соблюдение сроков выпуска.

Инструктор

Цель группы обучения — повысить эффективность труда пользователей. Поэтому инструкторы «принимают сторону» пользователей подобно тому, как менеджеры продукта представляют интересы заказчика. Однако перед пользователями инструкторы выступают в роли представителей проектной группы.

В этом последнем качестве группа обучения отвечает за выпуск удобного, полезного продукта, которому практически не нужна поддержка. Персонал группы тестирует удобство использования продукта, выявляет проблемы в этой области и проверяет проект пользовательского интерфейса.

Активно участвуя в создании пользовательского интерфейса, инструкторы сокращают затраты на сопровождение продукта и поддержку пользователей. Часто же бывает, что изучению этих расходов практически не уделяется внимания, хотя все расчеты очень просты: чем легче работать с приложением, тем меньше затраты на поддержку.

Внимание! Не попадайтесь в ловушку; не завышайте планируемый прирост эффективности труда пользователей — это приводит к переоценке прибыли от инвестиций в проект.

Довольно часто приложение сдается в эксплуатацию без плана обучения, а проектные группы даже не имеют представления о том, как пользователи будут изучать приложение. Задача инструкторов — не допустить этого, заранее спроектировав, составив и протестировав все необходимые материалы, включая краткие памятки, руковод-

ства пользователей, системы онлайновой помощи, Web-страницы и даже — если понадобится — целый учебный курс. Когда материалы подготовлены, труппа координирует обучение пользователей.

Управлять ожиданиями пользователей так же важно, как и ожиданиями заказчика. Однако не следует думать, что пользователи будут разбираться в функциональных спецификациях. Они более благосклонно отнесутся к прототипам системы, демонстрация которых значительно повысит шансы на успех проекта. Кроме того, пользователям полезно показать действующие модули программы. Хотя маркетинг входит в обязанности менеджеров продукта, также важно вовремя информировать и пользователей. Для этого следует периодически рассылать электронные сообщения о новых функциях программы и ее бета-версиях,

Примечание Роль обучения важна и в случае разработки приложения для другой организации. Ведь при этом вам неизвестно, насколько хорошо ее сотрудники информируют пользователей. Так же, как отдел менеджмента продукта управляет ожиданиями заказчика, отдел обучения должен управлять ожиданиями пользователей. Настоящий успех ожидает продукт, если пользователи узнают о его возможностях не сразу, а постепенно. При этом важны обучение и подготовка.

Логистик

Логистик представляет интересы служб поддержки и сопровождения, справочных служб и других служб канала доставки. Он занимается развертыванием продукта и его сопровождением и контролирует продукт с этой точки зрения в процессе проектирования. Кроме того, его задача — составление графиков развертывания приложения. Логистики, менеджеры продукта и менеджеры программы совместно определяют порядок передачи продукта пользователям и организации, после чего логистики готовят их к развертыванию приложения.

Логистик, участвующий в крупном проекте, должен обладать опытом развертывания крупномасштабных приложений на нескольких сотнях компьютеров. Именно поэтому от него требуется коммуникабельность и хорошая техническая подготовка. Логистик руководит всеми сотрудниками, устанавливающими и настраивающими пользовательские системы. Кроме того, он должен уметь координировать установку программного обеспечения и оценивать ее результаты.

Логистик обязан разбираться в инфраструктуре продукта и требованиях к его сопровождению. Его задача — проверить, чтобы все серверы развертывания и рабочие станции пользователей удовлетворяли этим требованиям. Обычно данные вопросы учитываются в планах

выпуска, развертывания и сопровождения продукта. Заметим, что развертывание приложений значительно упрощается при использовании таких средств Microsoft Windows 2000, как Active Directory и System Management Server.

Хотя основная задача логистика заключается в плавном развертывании продукта, обязанность руководителя этого направления — составить план сопровождения и эксплуатации и убедиться, что существующие группы способны с этим справиться. Важно обучить не только пользователей, но и персонал справочной службы. Причем последних надо начать знакомить с продуктом еще на ранних стадиях разработки — скажем, на этапе бета-тестирования. Перед сдачей приложения в эксплуатацию следует составить документацию, определить требования к резервному копированию данных и разработать план восстановления на случай отказа систем. После развертывания логистики в течение некоторого времени консультируют группу сопровождения. Конечно, сложно предсказать все трудности, которые могут возникнуть в процессе эксплуатации приложения, поэтому во всех планах следует предусматривать и порядок действий в экстренном случае.

Помимо перечисленных выше обязанностей, логистик занимается сопровождением промежуточных версий продукта в процессе разработки. Его знания требуются и при изучении поведения приложения в тестовой среде, Кроме того, помощь логистиков необходима при создании тестовых, сертификационных и производственных систем. Они должны также сопровождать приложение в процессе моделирования эксплуатации на этапе бета-тестирования.

Многие организации используют системы мониторинга производительности и стабильности. Поэтому логистики обязательно должны проинформировать разработчиков о наличии таких средств. Например, интеграция средств управления продуктом в среду административной консоли Microsoft Management Console (MMC) упростит интеграцию продукта с другими системами Microsoft.

Размер группы логистики определяется графиком развертывания, уровнем автоматизации установки, наличием средств автоматического развертывания приложений и другими характеристиками этого процесса.

Размеры группы и масштаб проекта

В проектной группе за каждое направление должен отвечать как минимум один человек. При реализации крупного проекта возникает затруднение, связанное с эффективным обменом информацией. В небольших организациях или при работе над мелкими проектами

роли можно совмещать. Однако в этом случае существует другая проблема — как «усидеть на нескольких стульях» одновременно, не упустив из виду ни одной существенной детали проекта с точки зрения каждой роли.

Крупные проекты

В своей книге «Rapid Development: Taming Wild Software Schedules» бывший сотрудник Microsoft Стив Макконнелл пишет:

> *«Для реализации крупного проекта необходимо, чтобы в организации был наработан опыт формализованного и непрерывного обмена информацией. ...А это возможно при наличии иерархической структуры, то есть небольших групп, в каждой из которых есть сотрудник, отвечающий за взаимодействие с другими группами и менеджерами».*

Таким образом, чтобы справиться с крупным проектом, приходится делить проектную группу на тематические и функциональные подгруппы.

Тематические группы

Это небольшие подгруппы из одного или нескольких человек, роли которых различны. Каждой из таких групп выделяется некий набор функциональных возможностей приложения, за все стороны проектирования и разработки которого она и отвечает (включая составление проекта и графика реализации). Например, какой-либо группе можно предоставить решать задачу вывода данных на печать,

По этому поводу Стив Маконнелл замечает:

> *«Достоинства тематических групп — эффективность, сбалансированность и ответственность. Возможности каждой такой группы велики, ведь в ее состав входят представители... всех заинтересованных сторон. Ее члены принимают решения, учитывая все возможные мнения, а, следовательно, нет никаких оснований их изменять.*
>
> *По этой же причине в такой группе высока ответственность. Ее члены могут обратиться ко всем людям, опыт которых необходим в их работе. Поэтому, если они так и не найдут оптимального решения, в этом они смогут винить только себя. Такая группа сбалансирована. Вряд ли вы захотите, чтобы окончательные спецификации создавал только отдел разработки, маркетинга или контроля качества. Только решение, принятое группой представителей каждого из этих отделов, будет по-настоящему сбалансировано».*

Функциональные группы

Функциональные группы формируются в рамках одной роли. Они нужны в очень крупных проектных группах или при работе над крупномасштабными проектами, когда отдельные роли нуждаются в до-

полнительном подразделении. Например, в Microsoft отдел менеджмента продукта обычно состоит из групп планирования и маркетинга. Обе они занимаются менеджментом продукта, но первая отвечает за определение действительно необходимых заказчику функций приложения, а вторая — за информирование потенциальных клиентов о достоинствах продукта.

Приведем еще один пример. Иногда группу разработчиков требуется разделить на три подгруппы, каждая из которых занимается одним уровнем архитектуры (пользовательским, прикладным или уровнем данных). Довольно часто в крупных отделах разработки группу программистов подразделяют на группу решений и компонентную группу. Первые создают собственно приложение, «склеивая» вместе его компоненты, разработанные вторыми. Кстати, это разделение часто оказывается и разделением по языкам программирования: разработчики решений, как правило, работают с языками, позволяющими быстро собирать приложения из готовых компонентов, тогда как выбор языка для разработки повторно используемых компонентов, пригодных для многих проектов, определяется прежде всего соображениями эффективности.

Небольшие проекты

Хотя в модели группы разработчиков предусмотрено шесть направлений деятельности, необязательно включать в проектную группу шесть человек. Другими словами, некоторые должности можно совмещать. Конечно, основной смысл такого разделения в том, чтобы каждую из шести задач решал один из членов группы. Однако не во всех проектах это возможно.

В небольших группах один человек может играть несколько ролей. При этом мы рекомендуем соблюдать следующие принципы разделения должностей.

- **Нельзя совмещать разработку с другими видами деятельности** — создателей приложения не стоит отвлекать от основной задачи. Если «повесить» на разработчиков дополнительные обязанности, то скорее всего график работ будет нарушен, а дату выпуска продукта придется отодвинуть.
- **Конфликт интересов** — нельзя совмещать роли, интересы которых противоположны. Пример — менеджер продукта и менеджер программы. Первый хочет выполнить все требования заказчика, второму же надо уложиться в график и бюджет. Если совместить эти роли, возникает опасность упустить просьбу заказчика о внесении изменений в проект либо, напротив, принять их без должного анализа влияния на график работ. Таким образом, назначение на эти

роли разных людей позволяет соблюсти интересы всех участников проекта.

На рис. 3.2 показаны комбинации ролей, оказывающие позитивное и негативное влияние на проект. Роли, отмеченные буквой «З» — Запрещено — нельзя совмещать из-за конфликтов интересов. Вероятность совмещения ролей, отмеченных буквой «Н» — Нежелательно — мала из-за сильного различия в необходимой квалификации. Например, знания и опыт менеджера продукта и логистика сильно отличаются. Сочетания ролей, помеченные буквой «Д» — Допустимо — возможны, так как их интересы совпадают. Например, как тестеры, так и инструкторы отвечают за выполнение требований пользователей.

Естественно, успех совмещения ролей зависит от конкретных членов группы, их навыков и опыта, В некоторых проектных группах возможно успешное совмещение ролей, комбинация которых согласно нашей таблиде опасна. Главное — помнить цели каждого направления и контролировать совмещение должностей в соответствии с ними, предотвращая конфликты. В противном случае некоторые обязанности какой-то роли будут не выполнены, что увеличивает число неконтролируемых рисков.

	Менеджер продукта	Менеджер программы	Разработка	Тестирование	Инструктор	Логистика
Менеджер продукта		З	З	Д	Д	Н
Менеджер программы	З		З	Н	Н	Д
Разработка	З	З		З	З	З
Тестирование	Д	Н	З		Д	Д
Инструктор	Д	Н	З	Д		Н
Логистика	Н	Д	З	Д	Н	

Д - Допустимо Н - Нежелательно З - Запрещено

Рис. 3.2. Деструктивные и созидательные сочетания ролей

Создание группы

Модель проектной группы описывает структуру группы для работы над проектом создания приложений масштаба предприятия. Однако одной структуры недостаточно — важным фактором успеха является

квалификация членов группы. В этой главе мы опишем методы проверки их квалификации.

Поиск руководителей

Главная задача человека, ответственного за создание проектной группы, — подобрать квалифицированных исполнителей. Эта кажущаяся простой (но на самом деле сложная) задача имеет огромное значение для успеха всего проекта.

Найти лидеров — несложная проблема; в любой организации они всем известны. Важно понимать, что мы говорим именно о лидерах. а не начальниках. Конечно, в любой организации есть менеджеры, директора и так далее, но положение в иерархической структуре далеко не всегда гарантирует наличие качеств лидера. Лидеров определяют действия и качества, а не должности.

Зачем нужны именно лидеры? Потому что исполнителей и так в избытке. Важно понимать, что деление сотрудников на лидеров и исполнителей не умаляет деловых качеств и квалификации последних. Чтобы лидер добился удачи, он должен набрать отличных исполнителей. Однако между лидерами и подчиненными должно быть взаимопонимание. Группа обсуждает, что нужно делать, а затем выполняет принятое решение, поэтому все, и руководители, и подчиненные, одинаково важны для проекта — в отсутствие кого-либо из них успех проекта невозможен.

Руководители должны обладать:

- умением понимать и помогать;
- коммуникабельностью;
- авторитетом внутри организации и за ее пределами;
- чувством ответственности за поставленные цели;
- умением принимать конструктивные решения;
- уверенностью в своих силах;
- достаточной для решения поставленных задач квалификацией;
- способностью помочь другим развить свои таланты и приобрести опыт.

Многие прочтут этот список и скажут: «Я обладаю всеми этими качествами». Еще больше людей подумают: «Я могу обладать этими качествами». Однако настоящего лидера отличают именно эти *способности, а* не *намерения.* Это относится и к качествам руководителя: нужно оценивать оказываемое им влияние, а не его намерения. Ведь давно известно, что «реальность — это то, что видят другие».

Повышение эффективности коллективной работы

Вот какие отношения друг с другом и к работе возникают в такой группе:

- заинтересованность;
- надежда, оптимизм, готовность к работе;
- определение задач и решений;
- проявление взаимопомощи;
- доверительные уважительные отношения;
- единение.

Последнее очень важно: эффективность работы группы при этом выше всего.

Для успеха проекта недостаточно только распределить роли и обязанности. Помимо структуры, следует придерживаться определенных принципов и методов. Ниже обсуждаются «лучшие методы и принципы», которые успешно применялись не только внутри Microsoft, но также партнерами и заказчиками компании.

Общее представление о проекте

Важнейший фактор для успеха проекта — единое понимание целей и задач проекта всеми участниками. Каждый из них изначально имеет свое мнение, касающееся приложения. В процессе формирования общего представления о проекте все эти мнения обсуждаются, что позволяет добиться единства целей всех членов проектной группы и заказчика.

На основе выработанного представления о проекте создается документ «Концепция проекта», который:

- описывает не только то, что делает продукт, но и то, чего он не делает;
- конкретизирует продукт (например, позволяет включать и исключать определенные функциональные возможности из данной версии);
- побуждает группу достичь сформулированной цели;
- содержит описание путей реализации проекта, благодаря чему проектная группа и заказчик могут начать работу.

Примечание Создание концепции не гарантирует согласия с ней всех участников проекта (разработчиков, заказчика и пользователей). Поэтому необходимо проследить, чтобы все они тщательно изучили ее и полностью в ней разобрались.

Выработка согласованного мнения о проекте позволяет избежать разногласий между участниками проекта, мешающих достичь поставленных целей и разъединяющих группу. Оптимальный способ согласования концепции — обсуждение целей и задач проекта. Такая дискуссия дает возможность добиться взаимопонимания всех членов группы. При этом их обязательства будут не простым согласием: «Да, мы будем работать над проектом», в них появится мотивация: «Да-

вайте приступим к работе и сделаем что-то действительно новое». Дискуссию на ранних стадиях проекта, как правило, организуют сотрудники отделов менеджмента продукта и менеджмента программы, имеющие опыт маркетинговых исследований.

Внимание! Не обольщайтесь тем, что действия проектной группы и их цель понятны заказчику. Подробно обсуждайте проект с заказчиком на всех стадиях его выполнения.

Группа **равных**

В группе равных важна каждая роль. Такой подход, и только он, делает возможным неограниченный обмен информацией между членами группы, повышает ответственность за выполнение работы и усиливает понимание того, что все шесть целей проекта одинаково важны. Естественно, в таких группах надо проводить проверку качества продукта; это делает представитель заказчика, разбирающийся в решаемой задаче.

Подчеркнем, что равенство — это не анархия. Равенство существует лишь в отношении ролей; в рамках каждой роли следует придерживаться обычной иерархической модели. В каждой группе необходима должностная иерархия и наличие руководителя, ответственного за управление и координирование работы своего направления. Задача остальных членов группы, исполняющих ту же роль, — выполнять поставленные перед ними задачи.

Ориентация на **продукт**

Ориентация на продукт — это не призыв работать над коммерческим программным обеспечением одним способом, а над приложениями для внутреннего пользования — иначе. Это требование — относиться к результату работы, как к продукту.

Прежде всего нужно выяснить, является проект самостоятельным или частью более крупного проекта. MSF рекомендует идентифицировать проекты — это позволяет людям чувствовать себя членами команды. Скажем, в Microsoft принята практика присвоения проектам кодовых имен (например «Чикаго» для Windows 95 и «Мемфис» для Windows 98). Кодовые имена четко идентифицируют проект и работающую над ним группу, позволяют людям четче ощущать причастность к проекту и ответственность за него. Создать и усилить значимость группы, поднять ее боевой дух можно разными способами — скажем, помещая название проекта на футболки, кружки и прочие сувениры.

Сформированное отношение к продукту позволяет сконцентрироваться на результате проекта, а не на процессе его достижения. Это вовсе не означает, что процесс не важен для группы. Мы просто ина-

че расставляем приоритеты — процесс должен служить достижению цели, а сам по себе он не имеет ценности. Увлеченность процессом не должна стоять на пути к результату. Необходимо прежде всего, чтобы все, чувствовали ответственность за выпуск продукта.

Бывший менеджер программ Microsoft Крис Питере описал отношение к продукту следующим образом:

> *«У нас всех... одна и та же работа — выпуск продукта. Ваша задача — не писать код, не тестировать его и не создавать спецификации. Ваша задача — выпустить продукт. Именно это делает группа разработки продукта.*
>
> *Ваша роль разработчика или тестера — вторична. Я не хочу сказать, что она не имеет значения — конечно, она важна, — но она вторична по отношению к вашей основной задаче — выпуску продукта.*
>
> *Когда вы просыпаетесь утром и приходите на работу, вы спрашиваете себя: «Что мы делаем, пишем код или делаем продукт?» Ответ очевиден — мы делаем продукт. Вы не должны программировать, вы обязаны не программировать, и это не каламбур».*

Ориентация на отсутствие дефектов

Ориентация на отсутствие дефектов — это требование соблюдения качества продукта. Оно вовсе не подразумевает, что продукт должен быть выпущен без дефектов; достаточно, чтобы дефектов было не больше нормы, установленной группой на ранних стадиях разработки. Кроме того, это означает, что группе придется постоянно контролировать качество на всех стадиях: если завтра потребуется выпустить готовый продукт, группа должна быть готова выполнить это обязательство. Таким образом, с точки зрения качества продукт всегда должен быть практически готов,

В успешных группах все отвечают за качество продукта — эту обязанность нельзя переложить на других. В этом смысле все члены группы являются представителями заказчика.

Понимание целей бизнеса

Чтобы выпустить успешный продукт, группе недостаточно разбираться только в технической стороне проекта. Создано множество технически совершенных и безупречно написанных продуктов, не способных выполнять задачи, поставленные заказчиком. Чтобы избежать такой ситуации, члены группы должны ясно представлять себе решаемую задачу.

И здесь необходимо активное участие заказчика в процессе разработки. Его обязательно нужно привлекать к созданию концепции проекта, к принятию решений по продукту и его использованию и к анализу промежуточных версий.

Ответственность в равной мере

Крис Питере однажды не без юмора заметил:

> *«Чрезвычайно важно, чтобы ответственность разделяли сотрудники как можно более низких уровней. Ваша цель не в том, чтобы лишиться сна, переживая из-за проекта, которым вы руководите. Ваша цель не в том, чтобы лишить сна руководителей направлений. Не спать по ночам от переживаний за проект должны абсолютно все. Вот когда вы этого добьетесь, считайте, что распределили ответственность должным образом».*

Чтобы добиться согласованной работы всех членов группы, вы должны сделать их роли взаимозависимыми и перекрывающимися, они все в одинаковой мере должны разделять ответственность за выпуск нужного продукта в запланированные сроки. Таким образом разрушается узкая специализация членов группы, которая ведет к изолированности, а не к совместной работе.

Совместное проектирование

Джим Маккарти так описал концепцию общего участия в проектировании в своей книге «Dynamics of Software Development»:

> *«Цель проектирования любого продукта — собрать лучшие идеи. Поэтому в проектировании должны участвовать все члены группы».*

В создании функциональных спецификаций должны участвовать представители всех направлений, так как у каждого из них свое представление о проекте. Они должны убедиться, что в проекте учтены не только требования всей группы, но и требования каждой роли.

Обучение на опыте других проектов

Чтобы не полагаться на удачу, стоит воспользоваться опытом работы над другими проектами, как успешными, так и провалившимися. Изучение опыта — основа совершенствования. Один из методов структурирования и накопления опыта — подведение итогов каждого этапа, зафиксированное в обзоре. Обзор, основа которого — сопоставление планов с результатами, помогает скорректировать дальнейший ход работы и избежать ошибок в дальнейшем. Кроме того, при этом можно выявить хорошо зарекомендовавшие себя методы, чтобы применять их в будущем.

Примечание Очень важно делиться информацией о методах работы с другими группами разработчиков своей организации.

Подытоживая рекомендации по подбору штата, предложенные Барри Бёмом в книге «Software Engineering Economics», мы рекомендуем следующие правила;

- используйте меньше людей, но лучших в своей области;
- подбирайте задачи в соответствии с квалификацией и мотивацией людей;
- помогайте людям набирать опыт и знания;
- подбирайте людей, дополняющих друг друга;
- не бойтесь отсекать все, что не подходит.

Обучение группы

Работа группы эффективна, если все ее члены понимают, что они делают. Однако во многих случаях это простое правило реализуется недостаточно полно. Выпустить же нужный продукт в срок удастся только при наличии формального и неформального планов обучения проектной группы.

Изучение методологии

Разработка программного обеспечения — это не только создание кода. Поэтому изучение методологии разработки руководителями групп и основными участниками проекта сократит затраты и время выполнения проекта.

Целью любого проекта должно стать улучшение обмена информацией. Для этого в MSF и других моделях разработки применяется общий язык передачи информации о состоянии продукта. Основные элементы этого процесса существуют не один год, и многие разработчики применяют их на практике. Используя универсальный язык MSF, опытные разработчики смогут поделиться своим опытом с другими членами группы.

Изучение технологий

Создать хороший продукт очень сложно. Поэтому группам разработчиков, логистиков и, по возможности, менеджеров программы надо изучать существующие технологии. Каждый день появляются новые формы обучения современнейшим технологиям, однако для успешной реализации таких технологий недостаточно только знания их основ, необходимо понимать способы их применения.

Вот какие технологии необходимы для работы над проектом по созданию *системы управления ресурсами* (Resource Management System, RMS), о которой идет речь в практикумах этой книги:

- **HTML;**
- Dynamic HTML;
- Microsoft Active Server Pages (ASP);
- Microsoft Visual Basic Scripting Edition (VBScript);
- Microsoft Internet Information Server (IIS);
- Microsoft Windows NT 4.0;

- Microsoft Windows 2000;
- Microsoft Systems Management Server 2.x (SMS);
- Microsoft Outlook 2000;
- Microsoft Visual InterDev;
- Microsoft Visual Basic 6.0 (VB);
- Microsoft Visual C++ 6.x (C++), библиотеки ATL и STL;
- Microsoft Transaction Server 2.0 (MTS);
- Microsoft SQL Server 7.x;
- создание COM-объектов с помощью VB и C++;
- ActiveX Data Objects (ADO);
- Collaborative Data Objects (CDO).

Часто при обучении упускают из вида сопровождение приложения в существующей информационной инфраструктуре. Однако успех проекта зависит не только от качества созданного кода. Проектная группа должна исследовать взаимное влияние инфраструктуры и приложения друг на друга. Как скажется на приложении появление новых операционных систем, новых версий существующих операционных систем или новых базовых приложений, используемых продуктом?

Кроме того, проектной группе придется учесть то обстоятельство, что во время работы над проектом могут появиться новые технологии или новые версии приложений. Нужно уметь оценивать влияние новых функциональных возможностей таких продуктов на жизненный цикл проекта. Поэтому время на обучение следует включать в график проекта еше до определения даты выпуска приложения.

Координация работы с внешними группами

Проектная группа, рассчитывающая на успех, должна взаимодействовать с внешними группами — как с заказчиком и пользователями, так и с другими разработчиками, Этим занимаются менеджер программы, менеджер продукта, инструктор и логистик. В обязанности этих ролей входят как внутренние, так и внешние контакты, в то время как разработчики и тестеры изолированы от общения с внешним миром. (Такая изоляция приводит к повышению эффективности труда этих двух групп.)

Важно, чтобы взаимодействие с внешними группами было явным и понятным. На рис. 3.3 проиллюстрирована координация взаимодействий, связанных как с бизнесом, так и с технологиями. Проектным группам, как правило, приходится взаимодействовать с большим количеством внешних организаций, включая финансовые и юридические подразделения и отделы контроля качества продукта.

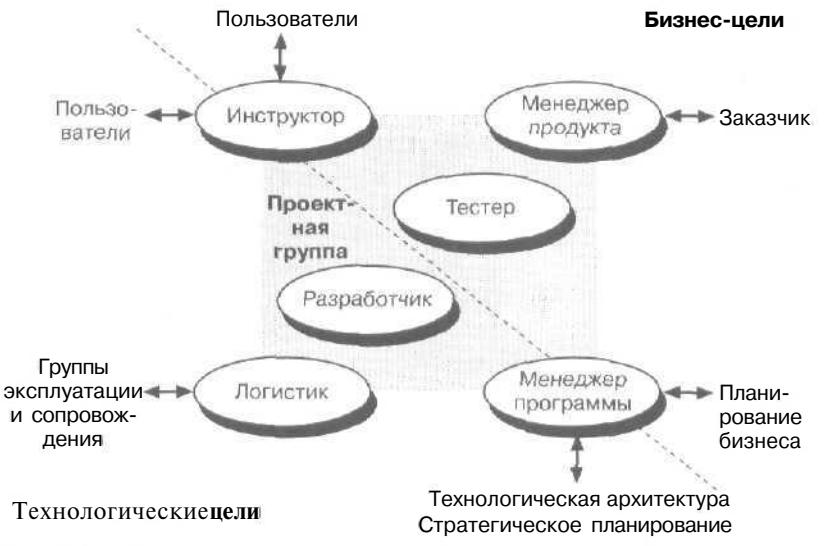


Рис. 3.3. Внутренние и внешние связи проектной группы

Средства управления группой

Для эффективного управления группой ее руководители должны применять методы, способствующие взаимодействию людей при их совместной работе. Приведенный ниже опросник применяется консультационной службой Microsoft для выявления настоящих лидеров. Помните, что ответы человека на эти утверждения говорят о его намерениях, а о его реальном влиянии свидетельствуют ответы на вопросы о нем, данные другими людьми.

- Я просчитываю ход работ надолго вперед,
- Я приветствую нововведения, привлекаю заказчиков к планированию и проектированию.
- Я поощряю сотрудников в поиске оптимальных решений, мне нравится, когда они это делают.
- Я отлично разбираюсь в работе организации.
- Я думаю о «что и зачем» больше, чем о «как и когда».
- Я точно знаю цели работы и выполняю свои обязанности.
- Я ориентирован на действие и могу мобилизовать людей и ресурсы.
- Я культивирую взаимное уважение и доверие, а также совместное принятие решений там, где это нужно.
- Я говорю только правду, и мои принципы всем известны.
- Я с удовольствием говорю людям, что они хорошо выполняют свою работу.

- Я даю людям знать, что я интересуюсь тем, что они делают, я выявляю в людях самое лучшее.
- Я приветствую коллективную работу, общее понимание целей и осознание ответственности.
- Я открыт для общения и всегда сообщаю о планах и целях участникам проекта.
- Я всегда сравниваю ход работ с планом их проведения.
- Я занимаюсь проблемами в соответствии с их приоритетами.
- Я добиваюсь, чтобы люди понимали свои обязанности и свою ответственность; в свою очередь, я честно и своевременно отвечаю на их вопросы.

Если на большую часть вопросов вы ответили утвердительно, то вам удастся превратить разрозненную кучку людей в сплоченную группу, способную добиться гораздо большего, чем каждый из них по отдельности. Важно помнить, что эти характеристики, наряду с изучением процесса разработки и различных технологий, являются отличными средствами создания эффективно работающих групп.

Часто между заказчиками и разработчиками возникает непонимание. «Билль о правах заказчиков» и «Билль о правах разработчиков» помогут вам не забывать о взаимодействии этих групп.

- **Билль о правах заказчиков**

  Менеджеры продукта должны помнить о правах заказчика (цитируется по книге Стива Макконнелла «Software Project Survival Guide». Итак, заказчик имеет право:

  - устанавливать цели проекта и требовать их достижения;
  - получать информацию о длительности проекта и затратах на его выполнение;
  - решать, какие функции должны присутствовать в приложении, а какие нет;
  - вносить разумные изменения в требования к проекту и получать информацию о связанных с ними расходах;
  - регулярно получать информацию о рисках, влияющих на затраты, график или качество продукта, и предложения по их устранению;
  - иметь доступ к основным документам проекта.

- **Билль о правах разработчиков**

  Похожий билль о правах Стив Макконнелл предложил и для разработчиков. Они имеют право:

  - знать о целях проекта и их приоритетах;
  - получать подробную информацию о продукте и уточнять ее в случае неясности;

- иметь доступ к заказчику, менеджеру, специалисту по маркетингу и другим участникам проекта, ответственным за принятие решений;
- применять на каждой фазе проекта нужные технологии, в частности, не начинать кодирование слишком рано;
- участвовать в оценке затрат и графика работ и их утверждении, включая право предоставлять только те оценки, которые возможны на данной стадии проекта, право затрачивать необходимое время на проведение таких оценок и корректировать их при изменении требований к приложению;
- сообщать о состоянии проекта заказчику и руководству;
- работать в комфортной и производительной среде, особенно на критических этапах проекта.

Резюме

Модель проектной группы и принципы создания приложений, описанные в этой главе, еще не гарантия успеха проекта — важны и другие факторы. Тем не менее структура проектной группы очень важна. Она считается основой успеха проекта, а реализация модели проектной группы и использование ее ключевых принципов поможет повысить эффективность работы и достичь успеха.

В своей книге «Rapid Development; Taming Wild Software Schedules» Стив Макконнелл так проиллюстрировал эту точку зрения:

> *«Даже если ваша группа состоит из квалифицированных, целеустремленных и трудолюбивых людей, неправильная структура может свести на нет все их усилия. Неудачная структура группы увеличит время разработки, снизит ее качество, повлияет на боевой дух членов группы и повысит текучесть кадров, а в результате проект будет закрыт».*

В этой главе мы обсудили модель проектной группы, цель которой — помочь вам избежать этих проблем. Исследуя характеристики успешных коллективов, мы дали определения основных черт лидеров группы и самих групп. Затем описали некоторые средства оценки отдельных людей и коллективов.

Закрепление материала

1. Перечислите шесть ролей модели проектной группы.
2. Назовите основные цели и обязанности каждой роли.
3. Каковы особенности проектных групп больших и малых проектов?
4. Перечислите стадии процесса разработки и их связь с развитием группы.
5. Назовите два метода обучения, позволяющие повысить эффективность труда членов группы.

Практикум 2. Знакомьтесь: проектная группа

Билл Парди еще раз взглянул в блокнот. Сегодняшний день начинался с записи: «Проект RMS».

— Просто не верю, что я пришел сюда в 8 часов утра из-за какого-то дурацкого совещания, — проворчал он себе под нос. Он вспомнил, что все началось с того, что Дэн Шелли, новый директор по информационным технологиям, предложил Биллу и другим начальникам отделов посетить в январе учебные курсы. Тогда Билл шутил, говорил, что от такого предложения у него голова закружилась. Дэн улыбнулся ему в ответ, но стало ясно, что он очень серьезно подходит к методологии разработки Microsoft Solution Framework, и рано или поздно всем придется иметь с ней дело.

«Ну все, пора покончить с этим раз и навсегда», — подумал Билл. Он бросил блокнот в сумку и направился в конференц-зал.

В другом офисе Дэн Шелли, сидя за своим столом, прихлебывал уже третью чашку кофе и в который раз перечитывал свои заметки. Он готовился с шести утра — все необходимые документы были разложены по местам, и проектор готов к работе. Дэн полностью осознавал важность этого проекта и именно этого совещания.

«Если «Фергюсон и Барделл» собирается получить максимальную отдачу от информационных технологий, нужно лучше интегрировать в бизнес все информационные процессы, а поможет в этом MSF, — думал он, поднимаясь из-за стола. — Труднее всего будет с Биллом, но если он поймет всю ценность MSF для своего отдела и всей компании, он присоединится к нам».

Сложив заметки в портфель, он взглянул на часы и, громко сказав: «Представление начинается!», вышел из комнаты.

На пути к конференц-залу он увидел спешащего Тима О'Брайана, который, заметив Дэна, сразу же заулыбался и закричал:

— Видишь, Дэн, я пришел раньше десяти часов!

— Да, но ты можешь все испортить, если опоздаешь на совещание, — по-дружески заметил Дэн.

– Если я прибуду впереди тебя, то не опоздаю, — ответил Тим, обгоняя Дэна, чтобы войти *в* комнату первым.

– Вот видишь, я добрался раньше тебя! — победно добавил он,

Войдя в конференц-зал, Дэн заметил главного бухгалтера Джейн Клэйтон, которая сидела за столом и оживленно беседовала с Мэри-Лу Морис, представителем обучающей фирмы, услугами которой они пользовались не один год. Мэри-Лу работала с отделом Джейн, и результат этой работы оказался настолько хорош, что и начальники остальных подразделений компании «Фергюсон и Барделл» начали при-

влекать ее к обучению персонала. Она так много времени проводила в офисах компании, что многие считали ее сотрудником «Фергюсон и Барделл».

Женщины взглянули на Дэна и заулыбались.

— Знаешь, по условиям моего контракта за такие ранние совещания полагается платить в полтора раза больше. — сказала Мэри-Лу.

— А это не компенсируется долгими перерывами в занятиях, когда твои ученики водили тебя обедать в новый ресторан? — спросила Джейн с выражением притворной строгости на лице.

— Будешь так говорить, буду брать по двойной ставке каждый раз, когда ты присылаешь ко мне менеджеров, — улыбаясь, ответила ей Мэри-Лу.

Сидящая с другой стороны стола Марта Вольф-Гелен тоже немного оживилась. Марта была новичком в проектном подразделении фирмы. И хотя она только недавно окончила Массачусетский технологический институт, она уже зарекомендовала себя способным, организованным и вдумчивым работником. Дэн с одобрением отметил про себя, что она уже записала несколько вопросов на обороте повестки дня. «Она, как никто другой, подходит для выбранной мною роли», — подумал он.

В этот момент в комнату вошел Билл Парди и занял среднее кресло.

— Доброе утро, шеф, — поприветствовал его Дэн, но тот только проворчал что-то себе под нос. «Похоже, совещание будет интересным», — подумал Дэн.

— Спасибо всем за то, что согласились работать над этим проектом, — начал Дэн. — Как я уже говорил вам при личных встречах, необходимость в данном проекте очевидна, более того, он очень важен для фирмы. Я выбрал именно вас, потому что вы сможете привнести что-то свое, уникальное, что делает именно вас нужными для решения задач проекта. Сейчас я расскажу вам о них и о вашем участии.

— И тебе спасибо, что пришел вовремя, — просиял Тим, но Дэн прервал его шутку: — У нас много работы, что *видно* по повестке дня. Так что начнем.

Повестка дня

— Во-первых, — продолжил Дэн, — посмотрим, что для вас может оказаться новым. Во всех повестках дня всегда будет три стандартных пункта. Первый — пересмотр повестки. Это позволит нам при необходимости поправить план совещания в самом начале. Я буду отправлять ее вам за несколько дней до нашей встречи, так что вы сможете ее изучить и при желании обсудить. Таким образом, эта часть совещания — ваш шанс добавить или убрать из повестки некоторые вопросы.

В конце совещания я оставил время для вопросов, Это — ваша последняя возможность уточнить детали до заключительной стадии совещания. Вопросы могут и не относиться к повестке дня, но они должны касаться проекта и всей группы. Мы не должны тратить время на обсуждение других проектов или проблем, в решении которых не участвует вся наша группа.

Заключительная часть совещания — подведение итогов. На этом этапе нам надо убедиться, что каждому ясна личная и общая задачи на этом этапе. Мы утвердим задачи и графики их выполнения. Возможно, нам придется анализировать ситуацию, изучать возникшие проблемы и находить ошибки. Но все это нормально: на ошибках учатся. В конце же совещания мы будем определять, куда двигаться дальше и что предпринять для реализации следующего этапа нашего проекта. Так мы и будем завершать наши встречи. Вопросы есть?

Так как никто не ответил, Дэн переспросил;

- Никто не хочет добавить что-нибудь к повестке дня?

Но все только выжидающе смотрели друг на друга. Наконец, Марта подняла руку:

- Извините, мистер Шелли, но в вашем вступительном слове вы упомянули о важности проекта для всей компании. Я ценю приглашение в проект, но не очень понимаю, зачем я здесь нужна. Может быть, стоит объяснить каждому, какова его роль.

- Во-первых, Марта, меня зовут Дэн, а не мистер Шелли. Во-вторых, ваш вопрос просто замечательный. Я планирую познакомить вас с вашими обязанностями позднее, когда вы поймете, почему приглашены в группу. Кстати, это пятый пункт повестки дня. Вы не против?

Марта, улыбнувшись, кивнула, а Дэн, обращаясь ко всем остальным, добавил:

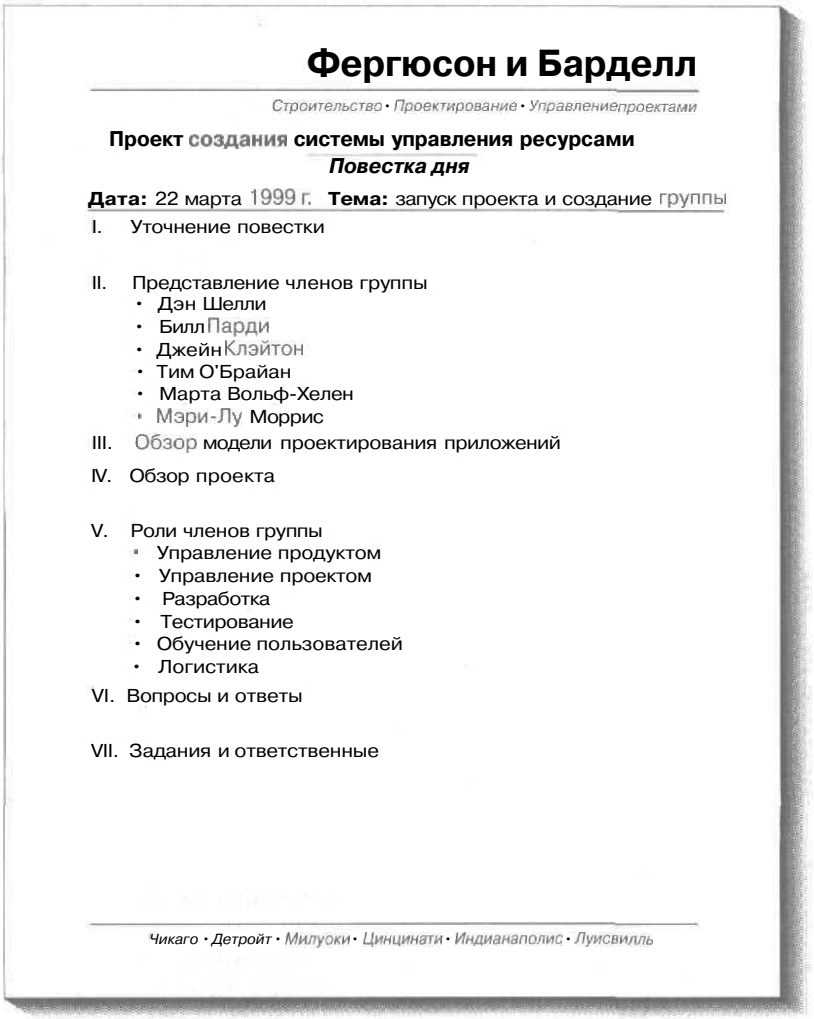
— А вы не против продолжить работу?

Все кивнули в знак согласия. Тим же наклонился к Марте и громко прошептал:

— Спасибо, Марта. Меня тоже интересовал этот вопрос, но когда я в прошлый раз на совещании задавал вопросы, меня назначили руководителем проекта.

Все засмеялись, напряжение заметно спало.

— Джейн, — сказал Дэн, — почему бы тебе не представиться — таким образом мы начнем первый пункт нашей повестки дня. Расскажи о своей должности, сколько времени ты работаешь в компании, чем занимаешься. Мы будем работать вместе не один месяц, поэтому, чем быстрее мы познакомимся, тем лучше.



Фергюсон и Барделл

Строительство • Проектирование • Управление проектами

Проект создания системы управления ресурсами
Повестка дня

Дата: 22 марта 1999 г. **Тема:** запуск проекта и создание группы

I. Уточнение повестки

II. Представление членов группы
- Дэн Шелли
- Билл Парди
- Джейн Клэйтон
- Тим О'Брайан
- Марта Вольф-Хелен
- Мэри-Лу Моррис

III. Обзор модели проектирования приложений

IV. Обзор проекта

V. Роли членов группы
- Управление продуктом
- Управление проектом
- Разработка
- Тестирование
- Обучение пользователей
- Логистика

VI. Вопросы и ответы

VII. Задания и ответственные

Чикаго • Детройт • Милуоки • Цинцинати • Индианаполис • Луисвилль

Мы должны знать как о достоинствах каждого, так и о его слабостях. Чтобы у нас получилась хорошая команда, надо использовать силу друг друга и помогать друг другу в преодолении слабостей. Между нами все должно быть начистоту, так что давайте познакомимся. Начинай, Джейн, расскажи о себе.

Знакомство

– Ты уверен, Дэн. что после такой проповеди я не запою соловьем? — сказала Джейн, улыбаясь. Посерьезнев, она поведала о работе в «Фергюсон и Барделл», о том, как она стала главным бухгалтером, кратко перечислила свои обязанности и в заключение сказала, что в настоящее время отвечает перед советом директоров за бухгалтерский учет фирмы. — Так что если что-то не работает или работает медленно, я получаю нагоняй сверху. Это побуждает меня искать действительно оптимальные решения.

– Хорошо, — кивнул Дэн, — нам это пригодится. Рад, что ты здесь, Джейн. Теперь очередь Мэри-Лу.

— Я уже четыре года занимаюсь обучением персонала различных чикагских компаний, в том числе и «Фергюсон и Барделл». Сотрудников вашей фирмы я обучала в основном работе с офисными приложениями, такими как Office. хотя немного затрагивала и работу операционных систем. Однако никаких специальных технических курсов в «Фергюсон и Барделл» я не вела.

— Она отлично работает, — добавила Джейн. — После обучения мои сотрудники работают со всеми нужными приложениями. Кажется, она понимает повседневные задачи и говорит о них в своих лекциях.

– Так и должен работать хороший преподаватель, — заметил Дэн. — Именно отличное понимание потребностей пользователя сделало тебя кандидатом в нашу группу. Ты нам очень пригодишься.

– Когда же мы узнаем, в чем заключается наша работа? — прервал его Билл.

– Подожди. Билл, это 4 и 5 пункты повестки дня. Так, теперь пусть Тим расскажет нам историю своей жизни.

– Похоже, она будет довольно короткой, — усмехнулся Тим, — ведь в «Фергюсон и Барделл» я работаю всего несколько лет. Но, хотя я здесь почти самый младший, — продолжил он, повернувшись к Биллу, — силы, отданные этой компании, сравнимы с вкладом таких старожилов, как Билл.

Билл что-то пробурчал в ответ, но на его лице появилась легкая улыбка. За время работы Тима в компании Билл сотрудничал с ним не один раз и понял, что, несмотря на мальчишескую ухмылку, Тим — отличный системный инженер и грамотный специалист по сетям. Невероятно, но у них сложились очень хорошие отношения. Тим был одним из немногих, кто мог безнаказанно подшучивать над Биллом Парди.

– Так, — продолжил Тим, снова обращаясь ко всей группе, — я попал сюда сразу же после колледжа в качестве заместителя начальника сетевого отдела. Все, что эта должность дала мне, был пейджер и «привилегия» быть готовым к вызову на работу 24 часа в сутки 7

дней в неделю. Затем что-то произошло и, я даже сам не знаю как, — наверное, я оказался в нужном месте в нужное время — стал начальником сетевого отдела.

— Не прибедняйся, — сказала Джейн Тиму, после чего, уже обращаясь к Дэну, добавила: — Что-то, как он это называет, случилось через шесть месяцев после его поступления к нам. Мы переходили на NT, и не все было гладко. Тим внес несколько предложений, но работающий у нас консультант проигнорировал слова молодого помощника. В конце концов, после постоянных сбоев сети в течение двух недель, твой предшественник отстранил его и спросил Тима: «Ты можешь все исправить?» Тот приступил к работе, и через полтора дня сеть заработала стабильно и до сих пор так работает.

Видя, как Тим покраснел от такой похвалы, Билл тихо проворчал: «Отличнаяработа»,

- Поэтому, когда весной освободилось место менеджера, — улыбаясь, сказал Дэн, — ты отделался от третьей смены, но сел на «горячий» стул?

Тим кивнул, остальные же довольно хихикали,

-- Знаешь, во время работы над проектом этот стул может стать еще «горячее». Но, Тим, я хочу, чтобы ты знал. Я много слышал о твоих достижениях, и, похоже, говорившие о тебе люди не преувеличивали, — Тим снова начал краснеть, но в этот момент Дэн, вызвав всеобщий смех, добавил: — Только будильник купи, ладно?

Затем он повернулся к Марте и сказал:

— Марта, ты здесь новичок, и многие не знакомы с тобой. Я знаю, что ты окончила Массачусетский технологический институт. Что привело тебя к нам, чем ты занимаешься?

- Прошлой весной я получила степень магистра в области электротехники. К тому времени я уже сотрудничала с отделением «Фергюсон и Барделл» в Луисвилле, и мне нравилась работа и люди, меня окружавшие. За месяц до выпуска со мной встретился директор проектного подразделения фирмы в Чикаго и после недолгих переговоров предложил поработать в чикагском отделении, что я, собственно, и делаю с лета.

Дэн уже знал об упомянутых Мартой переговорах, так как именно директор проектного подразделения порекомендовал ему Марту. Однако он подумал, что проектной группе необходимо знать о Марте поподробнее, и спросил ее:

-- Ты не против рассказать, о чем вы договаривались?

-- Нет, — засмеялась Марта, — в этом нет ничего особенного. Я просто сказала, что хочу участвовать в проектах, затрагивающих разные сферы деятельности. Ведь одна из привлекательных возможное-

тей работы в «Фергюсон и Барделл» — сотрудничать не только с архитекторами и менеджерами проектов, но и с инженерами. А я еще не уверена, в какой области хочу специализироваться, поэтому и стремлюсь поучаствовать в разных проектах.

— Думаю, что этот проект как раз для тебя, — сказал Дэн. — Рад, что ты присоединилась к нам.

По комнате пронесся одобрительный шепот.

— Ну что ж, Билл, — сказал Дэн, и все обратили свои взгляды на Билла. — Я оставил тебя напоследок. Как уже заметил Тим, ты работаешь здесь дольше всех и, возможно, уже забыл больше, чем мы знаем. Так что же ты хочешь сообщить о себе, чиф*?

Билл поднял глаза, казалось, он не знает, что сказать. Он не любил говорить о себе, по крайней мере, хвалить себя. Наконец, слегка пожав плечами, он начал:

— Я думаю, то, что я был военным, многое объясняет. Когда я попал во флот, я был никем, болтуном с огромным самомнением. Там меня научили, что слова должны подкрепляться действием, С тех пор я предпочитаю работать, а не разговаривать. Там же я научился программировать. Какие программы мы делали! Мы писали код, строчка за строчкой, в течение нескольких дней. Нельзя было отлучиться даже ненадолго, ни о каких разговорах даже речи не было. Единственное, что ценилось, сколько строк кода ты написал и скомпилировал за день. Мы собрали людей, которые и сейчас работают, продираются сквозь все эти данные, мы часами занимались этим. Сейчас же я не могу найти людей, которые согласились бы поработать хотя бы несколько часов сверх положенного. Кажется, что на создание программ требуется все больше времени, несмотря на все эти «Visual-что-то-там», которые должны облегчать работу. Иногда я хочу вернуться во времена VAX, вчитываться в зеленые буквы терминалов, а не щелкать по экрану этой дурацкой мышью.

Затем, как будто исчерпав силы этой тирадой, Билл опустился в кресло и уставился в стол.

Некоторое время все молчали. Наконец, Дэн нарушил тишину:

— Билл, все мы когда-то так думали. Управлять другими сложнее, чем выполнять свою часть проекта. Но, надеюсь, что процессы и модели, используемые нами, сделают эту работу чуть проще.

После небольшой паузы к Дэну обратилась Джейн:

— Так, мы все уже о себе рассказали. А как насчет тебя, Дэн? Что привело тебя в «Фергюсон и Барделл», почему ты работаешь над этим проектом, который так важен, что никто о нем ничего не знает?

* Чиф — на морском сленге старший помощник. — *Прим, ред.*

Дэн улыбнулся и, немного подумав, ответил:

– Как вы знаете, я перешел сюда из юридической фирмы. Но, скорее всего, вы не знаете, что я начинал в должности, не имеющей никакого отношения к ИТ. Я был учителем истории и не уверен, что сейчас перестал быть педагогом. Мне нравится помогать людям учиться, расти, достигать цели. Именно поэтому я перешел в компьютерную фирму. Ведь технологии — это шаг в будущее, а не барьер на пути к нему. Именно так я вижу свою работу: помогать своим сотрудникам стать экспертами в своей области с помощью разумного применения новых технологий. Запомните, технологии — не цель, они только средство ее достижения. А в нашем случае этим пределом будет лучшая архитектурная и инженерная работа, лучшее управление проектами наших клиентов. Наши бизнес-задачи должны привести нас к технологическим задачам, В этом нам поможет MSF.

MSF

При упоминании этой аббревиатуры все члены группы поняли, что Дэн перешел ко второму пункту повестки дня, и достали свои блокноты и ручки. Дэн выключил свет и вставил в проектор первый слайд, На нем была изображена окружность, разделенная на три части.

— Билл и Тим изучали MSF в январе, поэтому им многое уже знакомо. Мэри-Лу. кажется, ты брала материалы по MSF в библиотеке еще в феврале. Ты их изучила? — спросил Дэн, а когда Мэри-Лу утвердительно кивнула, добавил: — Хорошо. Полумается, что только Джейн и Марта не знакомы с этой технологией. Я дам вам материалы по ней, просмотрите их до следующего совещания. Сейчас же я хочу напомнить вам основы MSF, знание которых необходимо для сегодняшнего обсуждения. Мы будем поступать так и в дальнейшем.

MSF — сокращение от Microsoft Solution Framework. Это набор принципов, моделей и процессов, с помощью которых мы будем создавать собственные методы и процедуры управления нашим проектом. Эта технология не просто набор концепций, она отлично применяется на практике, ведь она создана на основе опыта Microsoft и многих других компаний.

На слайде показано, что работа ИТ-отдела делится на три части: планирование, разработка и управление. Заметьте, что диаграмма имеет форму окружности; это подчеркивает, что данная работа итеративна. Другими словами, процессы планирования, разработки и управления никогда не останавливаются.

Дэн вставил в проектор новый слайд, на котором круг был разделен на четыре части.

— На этом рисунке изображена одна из моделей MSF, модель процесса разработки. Она описывает последовательность нашей работы над проектом. Этот процесс состоит из четырех фаз: «Анализ», «Планирование», «Разработка» и «Стабилизация». Другими словами, сначала мы определим, что представляет собой наша задача и каким может быть ее решение. Затем нам придется составить подробный план реализации этого решения. Подготовив такой план, мы займемся разработкой, а затем развернем наше приложение и добьемся его стабильности и удобства использования. В дальнейшем мы подробно рассмотрим каждую фазу данного процесса, но на данный момент нам достаточно и этого.

- Что это за ромбы на круге? Они похожи на базы на бейсбольном поле, — спросила Джейн.

- Рад, что ты задала такой вопрос. Это этапы, завершающие каждую фазу проекта. По их достижении все члены проекта изучают ход работы, синхронизируют задачи и при необходимости вносят исправления. Кроме того, внутри каждой фазы тоже есть промежуточные этапы, но их мы изучим, когда будем говорить о соответствующих фазах нашего проекта.

Дэн выключил проектор и снова зажег свет. Показывая на коробку, стоящую на шкафу, он сказал:

- Я приготовил для каждого из вас скоросшиватели, в которые вы будете складывать материалы. К следующему совещанию изучите документы, которые я уже положил в ваши папки. В этих бумагах описана часть MSF, относящаяся к разработке новых приложений, таких, как наш проект RMS.

Приложение RMS

- Итак. RMS — сокращение от Resource Management System (система управления ресурсами), — сказал Дэн, прислонившись к шкафу. — Это новое приложение, которое будет использоваться во всех подразделениях «Фергюсон и Барделл». Как я уже говорил, этот проект очень важен для компании. В действительности, RMS будет управлять работой всех сотрудников фирмы, которые и являются се ресурсами. Средствами этого приложения будут создаваться графики работ и контролироваться их выполнение. Иначе говоря, любой человек в «Фергюсон и Барделл» сможет воспользоваться результатами работы этой системы. А так как RMS учитывает объем работ всех сотрудников компании, от его стабильности будет зависеть зарплата каждого.

Все члены группы притихли, проникшись важностью проекта. Дэн же выдержал паузу и продолжил:

— Обычно, такие проекты разрабатывал отдел Билла. Но, размышляя над системой управления ресурсами, я решил, что она отлично подходит для введения MSF в практику работы компании. Думаю, что, когда вы осознаете, какую пользу принесет MSF, вы поймете причины моего решения.

Теперь хочу прояснить одну вещь. То. что я решил привлечь всех вас к работе над проектом, не означает, что я не доверяю чифу. С ним этот вопрос мы уже обсудили.

— Но, Дэн, — прервала его Джейн, — ты сказал, что этот проект — разработка нового приложения. Какое отношение к этому имею я? Я не имею ни малейшего представления о том, как— как ты это назвал, Билл? — «писать код». Я умею только макросы Word записывать. Зачем я нужна в проекте создания системы учета ресурсов?

— Мне тоже интересно, зачем ты пригласил меня сюда, — добавила Марта. — Я действительно немного программировала в колледже, но вы же не собираетесь писать это приложение на Бэйсике или на Фортране. Поэтому мне тоже хотелось бы знать, что я здесь делаю.

— Отличный вопрос, — ответил Дэн. — Позвольте я все объясню с помощью следующих слайдов.

С этими словами он снова включил проектор.

Модель проектной группы MSF

На следующем слайде под заголовком «Цели группы» был напечатан список из шести пунктов.

- Хорошенько запомните эти шесть составляющих успешного проекта, — сказал Дэн. — Сейчас выясним, что вы о них думаете, — добавил он и вслух прочитал: — Удовлетворенный заказчик. Соблю-

денные ограничения. Соответствие спецификациям. Выпуск продукта только после устранения всех проблем. Повышение эффективности работы пользователей. И, наконец, гладкое развертывание и постоянное сопровождение.

Главный постулат MSF — проект нельзя назвать успешным, если не достигнуты *все шесть* целей. Как вы думаете, это правильные цели?

– Конечно, Дэн, это просто замечательные цели, но я еще ни разу не встречала проект, в котором все это было бы выполнено, — воскликнула Джейн. — Планы не слишком оптимистичны?

— Кроме того, некоторые из них противоречат друг другу, — добавил Тим. — Каким образом можно удовлетворить требования заказчиков и учесть ограничения и спецификации, предложенные пользователями? Выполнить все это в рамках одного проекта просто невозможно.

– В этом и заключается прелесть MSF, — ответил Дэн, отступив на шаг от проектора. — Я видел, как работает эта модель. Я видел, как проектная группа успешно завершила проект благодаря MSF. В юридической фирме, где я работал раньше, мы разрабатывали систему отслеживания информации о клиентах, применяя MSF. Если вы спросите любого участника того проекта, он вам скажет, что все шесть целей были достигнуты. Поверьте мне, это *можно* сделать.

Он, вставив в проектор слайд, на котором были изображены шесть овалов, расположенные по окружности.

– Здесь изображены шесть ролей членов проектной группы. Каждый из вас будет выполнять одну из них. Перед тем как я распределю эти роли среди вас, я хочу подробнее остановиться на этом рисунке.

Существуют два типа проектных групп. С одним из них, *иерархическим*, вы, возможно, знакомы: все члены группы подчинены одному человеку, менеджеру проекта, которому и сообщают о ходе своей работы. Основной недостаток этой модели заключается в том, что только этот менеджер отвечает за успех проекта, а следовательно, заинтересован в нем.

Мы будем придерживаться другой модели проектной группы, той, что изображена на рисунке. Заметьте, что все овалы — одинакового размера и связаны друг с другом. Другими словами, они равны, то есть все члены группы *равноправны*. Каждый участник проекта одинаково важен и одинаково отвечает за успех или провал проекта. Обратите внимание, что овалы закрашены разными цветами. Это показывает, что каждый член проекта исполняет свою роль, несет свою ответственность. По мере работы над проектом вы разберетесь в своих обязанностях, но сейчас важно понять, что у каждого будет роль,

которую может исполнять *только* он. А теперь перейдем к изучению этих ролей.

Роли членов группы

Дэн вставил в проектор слайд, похожий на предыдущий, но здесь каждый овал был подписан. Он указал на самый верхний из них, обозначенный «Менеджер продукта».

— Джейн, ты будешь менеджером продукта. Ты будешь представлять заказчика перед нами и нашу группу перед заказчиком. Твоя главная задача — гарантировать, что мы выполняем его требования.

Джейн оторвалась от блокнота и спросила:

— А кто будет заказчиком, которого я представляю?

— Часто заказчик и пользователь — одно лицо, но далеко не всегда. Лучший способ выявить заказчика — найти человека или группу людей, чьи запросы реализует проект. Кто в «Фергюсон и Барделл» заинтересован в эффективном распределении времени и средств?

— Я хотела сначала сказать «я», — немного подумав, ответила Джейн, — потому что я действительно заинтересована в этом. Но отделы менеджмента, отвечающие за распределение времени и средств, подотчетны финансовому директору, поэтому можно назвать заказчиком его.

— Точно, — кивнул Дэн. — Таким образом, в качестве менеджера продукта ты будешь представлять свой отдел — бухгалтерию — и нашего конечного заказчика, финансового директора. Твоя цель, во-первых, полностью разобраться в потребностях заказчика, а затем представить нашей группе отчет о его требованиях и возможных способах их выполнения. Теперь ясно, почему тебе не нужно писать код? Ты здесь потому, что способна понять потребности заказчика, донести их до нас и не дать нам отвлечься от их выполнения.

Джейн кивнула в знак согласия, после чего Дэн перешел к следующему овалу.

— Следующая роль — менеджер программы. Ее исполнять буду я. Я отвечаю за выпуск нужного продукта в нужное время. Это значит, что в мои обязанности входит составление графиков, распределение обязанностей и ресурсов проекта.

— Кажется, раньше это называлось «менеджер проекта», — сказал Тим.

— Эти должности похожи, но между ними несколько серьезных отличий. Помните, что все члены группы равноправны. В успехе проекта заинтересованы все, а не только я. Кроме того, так как важна точка зрения каждого, я не могу заставлять вас делать то, что я хочу. Я — руководитель и координатор проекта, но не начальник.

— Судя по этому, ты не сможешь здесь пользоваться своим положением директора по ИТ. — улыбаясь, заметил Тим.

– Точно. Я начальник только для задач моей роли, — ответил Дэн, после чего показал на следующий овал и добавил: — Эта роль — разработчик. Как вы все догадываетесь, она принадлежит Биллу. Его задача на фазе проектирования — технические консультации, он будет помогать нам понять, что же все-таки можно реализовать, а что — нет. Затем на основании подготовленного нами проекта ему придется разработать приложение.

— Другими словами, — сказал Билл, — я получаю от Джейн требования, после чего мои люди пишут приложение и отдают его ей, правильно? Практически то же самое, что мы делаем и сейчас.

– Это только часть полной картины, — ответил Дэн. — Есть и другие ее части, другие роли, с которыми тебе тоже придется взаимодействовать и которые отличают этот процесс от того, который используется сейчас. Одна из таких ролей, — продолжил он, указывая на следующий овал, — тестирование. Марта, это — твоя работа.

Марта стала записывать свои задачи, перечисляемые Дэном,

– Тебе надо спланировать и провести тестирование, которое определит качество нашего продукта. Ты всегда должна знать, какие проблемы приложения еще не решены. В итоге именно ты будешь определять статус приложения, четко сформулировав, что в приложении неправильно и что правильно.

Прежде чем Марта вставила слово, взорвался Билл;

– Ты хочешь сказать, что Марта, новичок в «Фергюсон и Барделл», не умеющая программировать, будет тестировать написанную моими людьми программу?

– Именно это я имел в виду, — ответил Дэн. — И если ты немного поразмыслишь, ты поймешь, что стоит поступить именно так. Нельзя одновременно писать код и его тестировать, хотя бы из-за непременного в таких случаях отсутствия конфликта интересов. Ведь нам нужна независимая проверка, нам нужен кто-то, кто скажет нам всю правду. Этот человек не должен участвовать в создании кода, чтобы глаз не «замылился». А то, что она не умеет программировать, не имеет никакого значения. Код создают *твои* люди, Марта же должна найти способы его протестировать. Именно здесь нам пригодится ее техническое образование и внимание к деталям.

Повернувшись к Мэри-Лу, Дэн указал на следующий овал с надписью «Инструктор»:

– Мэри-Лу, это твоя область — обучение пользователей. Как и Джейн, связывающая нашу группу с заказчиком, ты будешь связывать нас с пользователями приложения. На стадии проектирования

ты поможешь нам разобраться в потребностях пользователей, необходимых им функциях. Основные же твои задачи — создание системы поддержки продукта, в том числе и справочной системы, и планирование и проведение обучения.

Сделав несколько пометок в блокноте, Мэри-Лу спросила:

– По-моему, я кое-что уже придумала, но скажи мне, в чем разница между заказчиком и пользователем?

– Это просто, — вступила в разговор Джейн. — Заказчик платит за продукт, а пользователь с ним работает.

Все, в том числе и Дэн, рассмеялись.

– В общем-то, это правильно, — сказал он. — Заказчик, как правило, хочет, чтобы мы своим приложением избавили его от некоторой проблемы. Он готов предоставить нам ресурсы — деньги, время. иногда рабочую силу. Пользователям же это приложение, возможно, и не нужно, по крайней мере, так же как заказчику, но именно они будут с ним работать. Поэтому требования заказчика и пользователей могут отличаться, а иногда даже противоречить друг другу. Вот почему эти роли разделены.

Все это время Тим нетерпеливо ерзал на стуле и, не выдержав, спросил:

– Хорошо, Дэн, все уже получили свою роль. А как же я?

— А вот и ты, — Дэн указал на последний овал, — логистик. Ты связываешь нас со службой сопровождения и эксплуатации продукта. В твои задачи входит планирование и управление развертыванием системы. Поэтому во время разработки ты должен обращать внимание на вопросы поддержки и менеджмента. Ты будешь сопровождать продукт на стадии бета-версии и обучать сотрудников справочной службы, осуществляющих сопровождение приложения. Тим, для тебя такая должность — возможность роста. Она подразумевает, скорее, организаторскую работу, а не личный труд. Я выбрал на эту должность тебя, потому что считаю, что когда-нибудь ты станешь отличным менеджером, и в этом тебе, несомненно, поможет знание MSF.

– Помните шесть целей проекта? — спросил Дэн, вставив новый слайд. — Уверен, многие из вас уже поняли, что они связаны с нашими шестью ролями. Посмотрите, как они взаимодействуют.

— Джейн в роли менеджера продукта отвечает за выполнение требований заказчиков. Я выполняю обязанности менеджера программы и отвечаю за создание продукта с учетом всех требований и ограничений. В обязанности Билла входит выполнение функциональных спецификаций. Марта не даст нам выпустить приложение, пока не будут устранены все известные проблемы. Мэри-Лу поможет повы-

сить эффективность труда пользователей. И, наконец, Тим проследит за развертыванием продукта, — сказал он. — Вопросы есть?

Билл перекладывал свои бумаги, становясь с каждым словом Дэна все более взволнованным. Наконец он выпалил:

— Я не понимаю, Дэн. Мне кажется, что ты забрал у разработчиков всю работу и раздал ее не по назначению. Анализировать продукт будет бухгалтер, проводить тестирование — новичок-инженер, собирать требования пользователей — инструктор, обучающая пользователей шелкать по кнопкам! Скажи, ради бога, что мне-то остается?

— Скажу одним словом — *разработка,* — ответил Дэн, не повышая голоса. Он ожидал, что появятся недовольные, и понимал, что лучше бороться с ними быстро и тихо, чем медленно и громко. — В материалах по MSF сказано, что один человек может исполнять несколько ролей. Но одну из них нельзя совмещать ни с какой другой — это разработка. «Не трогайте программистов, пусть программируют». Вот ваша цель.

Билл, переложив все эти обязанности на других членов группы, вы сможете делать то, что у вас получается лучше всего: писать высококлассный код. Твоя работа — создать приложение, удовлетворяющее требованиям заказчика и пользователей, требованиям к развертыванию и сопровождению, и сделать это так, что мы даже не заметим, что ты что-то делал. Это как хороший инструмент — мы просто берем его и работаем, а если и взглянем на него, то только для того, чтобы отметить мастерство человека, изготовившего его. Поверь, Билл, MSF отлично подходит для разработчиков.

Все еще недовольный Билл ответил ему:

— Надеюсь, вы учли, что вся эта «коллективная» работа, сбор всех этих требований и планирование займет как минимум полгода?

— Мы выполним проект за треть твоего срока, Билл, — с улыбкой сказал Дэн. — Всего за два месяца. Именно через два месяца у нас будет готовое приложение. И при этом мы не отступим от процесса MSF ни на шаг.

От этих слов у Билла отвисла челюсть.

— Это невозможно! — возмутился он. — Просто невозможно выполнить все требования, перечисленные тобой, всего за два месяца.

Дэн немного помолчал, вспоминая точно такой же разговор с группой, разрабатывающей промышленную архитектуру.

— Билл, практически все так реагируют на MSF при знакомстве с этой моделью. Скажу начистоту, ты прав. Это невозможно. Поэтому частью нашего процесса разработки будет поиск решения этой про-

блемы. Но это будет потом, — он снова вернулся к повестке дня. — А сейчас у нас остались еще два пункта. Во-первых, ваши вопросы.

Все выглядели несколько ошалевшими, поэтому Дэн даже не ожидал, что кто-нибудь задаст вопросы. Они будут, но позже, когда проект пойдет полным ходом.

— Отлично. Переходим к следующему пункту: задание к следующему совещанию. Сегодня оно простое — вы возьмете папки и просмотрите лежащие в них материалы по модели, которой мы коснулись сегодня, а также изучите модель процесса разработки. Надеюсь, вы разберетесь в своих обязанностях в проекте. Кроме того, каждому из вас я могу дать персональные задания, так что проверяйте свою электронную почту. Встретимся здесь же в четверг, в это же время. До свидания.

Когда члены новоиспеченной группы потянулись к шкафу за папками, Дэн отвел в сторону Билла.

— Билл, ты помнишь, что сказал о Мэри-Лу — как же это прозвучало? А, «щелкать по кнопкам»? — спросил он, в ответ Билл смущенно кивнул. — Знаешь, Мэри-Лу не только щелкает по кнопкам — она, между прочим, сертифицированный разработчик. А инструктором стала, потому что ей нравиться учить людей. Каждый из вас занимается своим делом. Надеюсь, ты понял.

Билл опустил глаза и, увидев, что Мэри-Лу вышла, пробормотал:

– Спасибо, что ты сказал мне это с глазу на глаз. Мне было бы очень стыдно, если бы ты или она ответили мне на совещании.

– Нет проблем, чиф. Я не хочу отчитывать своих сотрудников на людях. Я только хочу, чтобы ты понял положение дел, — Дэн собрал свои бумаги и повернулся к Биллу. — Вообще-то моя цель в том, чтобы ты и твои ребята оказались героями. И если мы будем работать вместе, думаю, ты будешь приятно удивлен оценкой вашей работы по завершении проекта.

– Если это действительно будет так, — ответил Билл, — то даже такой старый морской волк, как я, наверняка сможет выучить несколько новых трюков.

Мужчины улыбнулись друг другу, взяли папки и двинулись к выходу.

ГЛАВА 4

Процесс разработки

В этой главе

Для успешного управления проектами разработки программного обеспечения нужны два важных качества. Первое — *строгость* — необходимо для постоянного контроля состояния проекта. Второе — *гибкость* — требуется для успешной адаптации к неизбежным неожиданностям. В этой главе мы сначала рассмотрим два традиционных подхода к разработке приложений: модель водопада и спиральную модель, а затем обсудим еще один способ — универсальный процесс разработки, ставший популярным в последние годы.

Большая часть этой главы посвящена *модели процесса разработки приложений MSF* (MSF Process Model for Application Development), или, короче, *модели процесса разработки*. Модель процесса MSF — это не детальная инструкция, а структурный каркас, на базе которого организации могут создавать конкретные способы решения своих задач. Модель процесса разработки — составная часть этой структуры, описывающая жизненный цикл проекта разработки программного обеспечения. Она позволяет проектной группе создавать продукт в постоянном контакте с заказчиком и адаптировать процесс в соответствии с его пожеланиями. Кроме того, этот метод способен обеспечивать самую быструю реализацию ключевых составляющих проекта. Модель процесса разработки приложений MSF — гибкий компонент общей модели процесса MSF, с успехом применяемый в индустрии разработки программного обеспечения для повышения управляемости проектов, минимизации рисков, повышения качества продукции и ускорения разработки.

Кроме того, в этой главе вы познакомитесь с практическими рекомендациями по проведению формального рецензирования, с критериями отбора участников этой процедуры и методами ее организации. Мы покажем, как организовывать и проводить соответствующие совещания, как собирать и документировать информацию. В завершение мы обсудим создание и роль специальной обзорной группы Е больших проектах.

В этой главе мы использовали собственный опыт проектирования архитектуры приложений и их реализации, кроме того, мы воспользовались следующими материалами:

- Microsoft Solutions Framework;
- Уокер Ройс (Walker Royce) «Software Project Management: A Unified Framework» (Addison-Wesley, 1999);
- Грэйди Буч (GradyBooch), Джеймс Рамбо (JamesRumbaugh), Ивар Джекобсон (Ivar Jacobson) «The Unified Modeling Language User Guide» (Addison-Wesley. 1998);
- Ивар Джекобсон (Ivar Jacobson), Грэйди Буч (Grady Booch), Джеймс Рамбо (James Rumbaugh) «The Unified Software Development Process» (Addison-Wesley, 1999).

Изучив материал **этой главы, вы сможете:**

- ✓ охарактеризовать модель водопада и спиральную модель и перечислить их недостатки;
- ✓ перечислить основные потоки универсального процесса;
- ✓ описать основные модели универсального процесса;
- ✓ охарактеризовать четыре фазы модели процесса разработки MSF;
- ✓ описать преимущества выпуска версий и роль итерационного похода в процессе разработки;
- ✓ описать обязанности различных членов проектной группы в рамках модели процесса разработки MSF;
- ✓ описать взаимосвязь переменных факторов и проектных ограничений и концепцию достижения компромисса;
- ✓ анализировать проект для выявления основных этапов:
- ✓ анализировать проекты разработки ПО для выявления основных целей последовательных итераций проекта.

Модели разработки приложений

Любой проект разработки программного обеспечения в своем развитии проходит определенный жизненный цикл — последовательность

этапов и совокупность действий, в результате которых создается первая версия продукта. Можно построить модель жизненного цикла, описывающую на некотором уровне абстракции его составляющие и порядок их определения, реализации, тестирования и выполнения. Реалистичная модель жизненного цикла упрощает выполнение проекта и гарантирует, что в проекте с каждым следующим этапом реализуется все больше запланированных задач.

Современные методы разработки приложений представляют собой квинтэссенцию опыта и практики таких традиционных моделей, как модель водопада и спиральная модель. Сначала мы познакомим вас с этими моделями и лишь затем перейдем к универсальному процессу и модели процесса разработки MSF.

Модель водопада

Для описания традиционного способа разработки приложений часто применяется метафора постройки дома. Сначала строитель беседует с заказчиком, выясняет, что нужно построить, проектирует здание и затем строит его по согласованному плану. По окончании строительства подрядчик должен удостовериться, что здание устойчиво, а псе его компоненты, от электропроводки и водопровода до дверного звонка, исправны и работоспособны. Затем заказчик проверяет, что все сделано в соответствии с его требованиями. Часто уже после того, как заказчик вселился н здание, строителю приходится устранять возникающиепроблемы.

Этапы постройки дома совпадают с основными стадиями модели водопада. Как показано на рис. 4.1, модель водопада представляет процесс разработки в виде строго упорядоченной последовательности этапов. К ним относятся:

- сбор требований к системе и программному обеспечению;
- анализ;
- проектирование;
- кодирование итестирование модулей;
- интеграция системы;
- тестирование приложения в целом;
- передача в эксплуатацию.

Для перехода к следующему этапу необходимо полностью закончить работу над текущим и тщательно описать его результаты.

Модель водопада в настоящее время используется сравнительно редко в связи с тем, что при ее применении на каждой стадии выполнения проекта возникают проблемы.

- **Излишние затраты времени** — как правило, интеграция всех подсистем в работающее приложение занимает гораздо больше времени, чем запланировано.

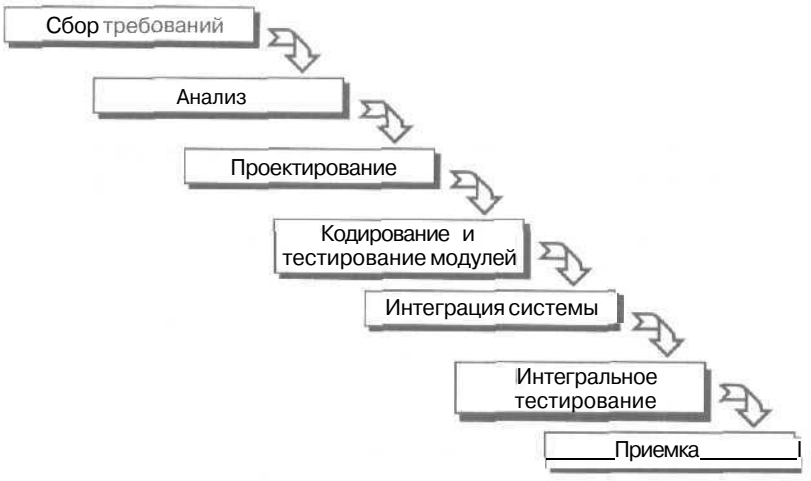


Рис. 4.1. Модель водопада

- **Выявление причин, по которым необходимо изменять проект на поздних стадиях** — при использовании модели водопада это обычное дело. Проверка работоспособности и реализуемости проекта приложения на ранних стадиях, как правило, не выполняется.
- **Неадекватное управление рисками** — риски, связанные с проектом, выявляются на поздних стадиях выполнения проекта.
- **Отсутствие процедуры пересмотра требований** — в этой модели требования должны быть сформулированы и зафиксированы на первых стадиях разработки. Как правило, цели и задачи в начале проекта осознаются не полностью, и поэтому их приходится пересматривать на поздних стадиях, что приводит к значительному росту затрат на разработку и задержке выпуска продукта. Если же изменившиеся требования не учтены в проекте, заказчик не будет считать приложение отвечающим поставленным задачам.
- **Ограниченные возможности обмена информацией** — традиционная практика предусматривает однократное рецензирование по окончании каждой стадии проекта. Отсутствие постоянного обмена информацией ведет к излишнему вниманию к деталям и может вылиться в непонимание между участниками проекта.
- **Отсутствие рецензирования** — первые четыре этапа модели водопада, по сути, — бумажная работа, и чтобы демонстрировать прогресс проекта, по окончании каждой стадии необходимо готовить массу документов. Лавинообразное нарастание объема проектной документации ведет к тому, что понятыми оказываются лишь вопросы, обсуждавшиеся последними; наиболее сложные стороны

проекта не проверяются, а правильность их решения принимается за аксиому.

Спиральная модель

Применение более современных методов привело к появлению итерационного подхода к управлению процессом разработки — так называемой спиральной модели, проиллюстрированной на рис. 4.2. Эта модель подразделяется на следующие стадии:

- **исследование** — анализ требований и предварительное планирование;
- **проработка** — проектирование приложений;
- **переходный период** — оценка и стабилизация приложения;
- **создание** — реализация приложения.

Уокер Ройс в своей книге «Software Project Management: A Unified Framework» отмечает, что каждая из этих стадий обычно подразделяется на пять фаз: сбор требований, проектирование, реализация, развертывание и управление. В рамках спиральной модели процесс разработки состоит из упомянутых выше четырех стадий, на каждой из которых эти пять фаз выполняются многократно. Например, на стадии исследования может понадобиться четыре итерации, то есть четырехкратное выполнение цикла. Цель итерационного подхода — добиться последовательного уточнения характеристик и архитектуры продукта, которое обеспечивает его постоянное приближение к окончательному виду.

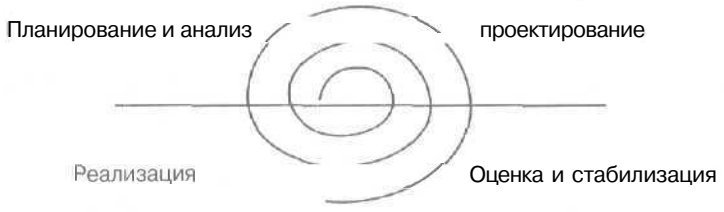


Рис. 4.2. Спиральная модель

Как и модель водопада, спиральная модель порождает множество проблем. Некоторые из них перечислены ниже.

- **Задержка реализации функциональных возможностей** — реализация сложных компонентов, представляющих особую важность для заказчиков и пользователей, откладывается на завершающие итерации. Это затягивает выполнение проекта и повышает стоимость работ.
- **Никогда не заканчивающиеся проекты** — проекты, реализуемые по спиральной модели, часто начинают жить собственной жизнью,

когда работа над проектом фактически превращается в работу для проекта; такие проекты или никогда не кончаются, или кончаются ничем. При изменении целей проекта итерационный подход вынуждает группу продолжать работу над реализацией прежних целей, игнорируя новые задачи. Это ведет к утрате ориентации и резкому снижению полезности проекта.

- **Неизвестная стоимость** — постоянное повторение стадий и затягивание реализации сложных требований затрудняет оценку стоимости и доходности проекта. Это, в свою очередь, усложняет оценку затрат и эффективности при обосновании следующих проектов.
- **Отсутствие стабильности** — постоянное изменение системы формирует у заказчиков и пользователей мнение о нестабильности продукта. Постоянное обновление всех составляющих проекта на каждой итерации резко увеличивает затраты и значительно повышает стоимость развертывания продукта.
- **Отсутствие автоматизации** — необходимость инвестиций в автоматизацию процесса разработки программного обеспечения часто упускается из вида. Значительные первоначальные расходы на средства автоматизации и повышения производительности при этом рассматриваются как издержки, а не как капиталовложения. Однако в отсутствие средств автоматизации итерационный подход к разработке приводит к значительным задержкам выпуска продукта. Даже по окончании разработки отсутствие средств автоматизации развертывания значительно затрудняет установку новых версий продукта на компьютеры пользователей.

Универсальный процесс

Этот раздел основан на материале книг Уокера Ройса «Software Project Management: A Unified Framework» и Ивара Джекобсона, Грэйди Буча и Джеймса Рамбо «The Unified Software Development Process», но с учетом нашего собственного опыта разработки. Если вас интересует точное определение и детальное обсуждение универсального процесса, обратитесь к перечисленным книгам.

Универсальный процесс (Unified Process, UP) — широко распространенный метод анализа, проектирования и разработки приложений масштаба предприятия. Основные характеристики этого процесса, базирующегося на универсальном языке моделирования, таковы:

- сценарии использования как основа проекта;
- приоритет архитектуры;
- итерационный и инкрементальный процесс разработки;
- прогнозирование рисков;

- объектно- и сервисно-ориентированное проектирование;
- активное накопление и повторное использование объектно-ориентированных шаблонов, объектов и кода.

Этапы

В основе универсального процесса лежат пять основных этапов, которые постоянно выполняются на всех четырех фазах процесса разработки вплоть до создания приложения. Завершение выполнения этих этапов называется итерацией, и каждая итерация завершается промежуточным выпуском продукта. Названия этапов несколько условны. На каждой итерации цикл начинается со сбора требований.

- **Требования** — сбор бизнес-, технических и прикладных требований к проекту.
- **Анализ** — бизнес- и прикладное моделирование на основе собранных требований.
- **Проектирование** — создание архитектуры на основе объектно-ориентированного подхода.
- **Реализация** — разработка спроектированного приложения (на ранних стадиях — разработка прототипов).
- **Тестирование** — проверка сделанной работы.

Ниже мы опишем основные этапы универсального процесса несколько подробнее.

Требования

Основная задача этапа «Требования» — направить итерацию на разработку приложения в соответствии с требованиями заказчика и пользователей. Для этого необходимо описать приложение с той степенью детализации, которая достаточна для достижения согласия между заказчиком, пользователями и проектной группой относительно того, что приложение должно делать и чего оно делать не должно. Информация на этой стадии собирается из множества источников: от участников проекта, из существующих систем и документов, подготовленных заказчиком.

По мере сбора информации проектная группа создает предварительный список требований. Требования удобно структурировать, снабдив каждое из них кратким названием и описанием, статусом (предложение, принято, включено в окончательный список и т. п.), оценкой стоимости реализации и описанием рисков, сопряженных с выполнением этого требования.

К компетенции этого этапа относится и контекст, в котором будет работать приложение. Авторы универсального процесса рекомендуют описывать контекст с помощью бизнес-моделей или моделирования предметной области. Функциональные требования, описываю-

шие взаимодействие приложения с субъектами и объектами контекста, описываются схемами использования. В процессе сбора требований необходимо проанализировать и нефункциональные требования, например, производительность, расширяемость и надежность. Они могут быть связаны с конкретными схемами использования или добавлены к модели схем использования в качестве нефункциональных требований к системе.

Примечание Схемы использования должны быть понятны заказчику и пользователю.

Помимо списка требований, проектная группа может создать набор проектов пользовательского интерфейса или прототипов, описывающих взаимодействие различных типов пользователей приложения. Ивар Джекобсон и его соавторы в своей книге описывают результаты работы этапа «Требования» следующим образом:

- бизнес-модель (или модель предметной области), задающая контекст проектируемой системы;
- модель схем использования, описывающая функциональные и общие требования, вытекающие из конкретных схем использования. Модель представляется в форме результатов опроса, набора диаграмм и детального описания каждой схемы использования;
- набор набросков пользовательского интерфейса и прототипов для каждого актера, представляющий проект пользовательского интерфейса приложения;
- список дополнительных требований, не относящихся к конкретным схемам использования.

Анализ

На этом этапе требования к приложению анализируются и формулируются на языке разработчиков. В результате функциональные требования, выведенные из схем использования на этапе сбора требований, уточняются и структурируются. Этап «Анализ» — промежуточный шаг между сбором требований и проектированием приложения. Уточнение и абстрагирование требований на этом этапе служит основой проектирования. Ивар Джекобсон и его соавторы характеризуют этап «Анализ» и его результаты следующим образом:

- уточнение требований, сформулированных на предыдущем этапе, включая уточнение модели сценариев использования;
- аналитическая модель формулируется на языке разработчиков, и поэтому на этапе «Анализ» появляется формализм, позволяющий судить о внутренней структуре системы:

- аналитическая модель структурирует требования таким образом, что они становятся более понятными, тем самым подготавливая их к превращению в проектные концепции;
- модель, построенную на этапе анализа, можно рассматривать как первый набросок проектной модели; следовательно, это— важнейшая исходная информация для проектирования и разработки системы.

Основной результат этапа «Анализ» — архитектурное представление аналитической модели. Это представление включает следующее:

- **Анализ классов**, в **том** числе пограничные, элементные и управляющие классы. Пограничные классы располагаются между пользовательскими ролями *(актерами* в терминологии универсального языка моделирования) и внутренними структурами приложения; они, как правило, являются идеальными кандидатами на роль сервисов представления или пользовательских сервисов. Элементные классы описывают долгоживущую или постоянно хранящуюся информацию, Управляющие классы реализуют различные типы поведения приложения, связанные с порядком работы, транзакциями и т. д.
- **Анализ реализации схем использования —** Джекобсон и др. определяют это как «*...взаимодействие различных составляющих аналитической модели, описывающее реализацию конкретной схемы использования в терминах классов и объектов, выделенных на стадии анализа*».

  Такое сочетание диаграмм использования и диаграмм классов описывает их взаимодействие. На основе анализа реализации проектная группа вырабатывает методы объединения схем использования по классам, объектам и итерациям.
- **Пакетирование** — организация классов и реализаций схем использования в функциональные требования, вытекающие из схем использования. Пакет основывается на схеме использования, поддерживающей определенный бизнес-процесс, и конкретной пользовательской роли или на группе схем использования, допускающих обобщение или связанных какими-либо отношениями.

Проектирование

Завершив анализ, можно приступать к низкоуровневому проектированию. На этом этапе выявляются классы и описывается их поведение, после чего классы распределяются по четырем уровням: стандартный пользовательский интерфейс (презентационный уровень). бизнес-уровень, уровень доступа и уровень данных.

На этапе проектирования выполняются следующие работы:

- определяются структуры подсистем (проектная модель);
- подсистемы распределяются между уровнями (проектная модель);
- определяются интерфейсы классов и объектов (проектная модель);
- активные классы связываются с узлами развертывания (модель развертывания).

По окончании этого этапа архитектура приложения считается полностью созданной. На этом этапе завершается и создание проектной модели, являющейся логическим представлением физической модели приложения. В отличие от этапа «Анализ», результат этапа «Проектирование» — проектная модель, описывающая все классы с их поведением, свойствами и методами, и их взаимосвязи — в течение жизненного цикла проекта остается практически неизменным.

Реализация

Универсальный процесс создавался на основе принципов спиральной модели, поэтому предполагает инкрементальный подход к созданию прототипов и разработке. Этот процесс циклически продолжается до тех пор, пока все необходимые функции не реализованы, а проектная группа полностью не удовлетворена реализацией. Этап «Реализация» представляет собой практическое воплощение этапа «Проектирование»; в частности, должно существовать однозначное соответствие между проектными классами и кодом, созданным на этапе разработки. Каждый из классов может представлять собой отдельный исполняемый модуль, или, напротив, один модуль способен объединять несколько классов — способ определяется выбранным языком программирования и проектом пакетирования приложения.

Этап «Реализация» состоит из:

- реализации прототипа архитектуры;
- реализации компонентов (классов и объектов);
- тестирования компонентов;
- интеграции компонентов;
- сборки приложения:
- создания тестов на основе схем использования;
- проверки архитектуры;
- планирования следующей сборки;
- переходу к следующей итерации.

Тестирование

Цель этого этапа — сверка достигнутых результатов с ожидаемыми. Тестирование начинается по окончании этапа «Реализация» независимо от того, выпуском какого типа — внутренним, промежуточным или внешним — завершается этот этап. На каждой итерации тестовая модель уточняется: из нее изымаются тесты, утратившие актуаль-

ность, создаются схемы регрессионного тестирования и добавляются тесты для будущих сборок. Каждый тест реализует конкретный метод проверки определенных функций приложения и включает условия тестирования, необходимые входные данные и ожидаемые результаты. Тесты создаются на основе схем использования. Помимо тестирования приложения, необходимо предусмотреть тесты процедуры установки и корректности конфигурации приложения на каждой используемой платформе. Кроме того, часто оказывается полезно создать компоненты, автоматизирующие процесс тестирования.

Фазы

Поскольку универсальный процесс базируется на спиральной модели, его фазы — исследование, проработка, создание и переходный период — совпадают с фазами спиральной модели. Каждая из них преследует свои цели:

- фаза «Исследование» предназначена для создания модели предметной области;
- итерации фазы «Проработка» ставят своей целью создание базовой архитектуры;
- итерации фазы «Создание» преследуют цель создания продукта путем последовательного выпуска версий с постепенно расширяющимися функциональными возможностями;
- переходный период необходим для проверки готовности продукта к эксплуатации.

На фазах «Исследование» и «Проработка» основное внимание уделяется сбору требований наряду с анализом, проектированием и реализацией архитектуры. По завершении каждой из фаз приложение со все возрастающей степенью детализации описывается совокупностью моделей универсального процесса. Для перехода к каждой следующей фазе необходимо выполнить задачи текущей фазы. На завершающем этапе каждой фазы результаты ее выполнения анализируются, и на основе анализа принимается решение о продолжении (или прекращении) работ и соответственно об одобрении бюджета и графика следующей фазы. Завершающие этапы каждой фазы служат, таким образом, точками синхронизации технической и деловой сторон проекта.

Исследование

Число итераций этой фазы трудно предсказать заранее, однако обычно оно не превосходит двух, а основное внимание уделяется этапу «Сбор требований». Задачи фазы «Исследование» четко определены ее целями: описание основных характеристик приложения, снижение вероятности реализации основных рисков и подготовка обосно-

вания проекта с точки зрения его связи с основными задачами бизнеса. На этой стадии:

- **выделяется предметная область действия системы,** то есть определяются границы применения приложения и его взаимоотношения с другими системами;
- **предлагается предварительная архитектура,** в частности, уточняются новые, сложные и рискованные элементы приложения; этот этап необходим, чтобы продемонстрировать возможность создания стабильной архитектуры;
- **выявляются основные риски,** а также составляется план управления рисками, направленный на снижение вероятности их возникновения и адекватную реакцию в случае реализации риска;
- **демонстрация способности предлагаемой системы решить поставленные бизнесом задачи** (обычно на примере прототипа приложения), позволяющая достичь согласий по этому вопросу с заказчиком и пользователями.

Фаза «Исследование» завершается формулировкой целей проекта.

Проработка

Как и фаза «Исследование», фаза «Проработка» обычно ограничивается двумя-тремя итерациями. На этой фазе основное внимание уделяется созданию базовой архитектуры приложения, более или менее детальной оценке стоимости и выработке предварительного графика. Кроме того, на этой стадии планируются работы фазы «Создание». Перечислим основные работы, выполняемые на фазе «Проработка»:

- **создание базовой архитектуры приложения,** включающей все функциональные возможности, признанные важными всеми участниками проекта;
- **выявление основных рисков,** включая план, стоимость и график управления ими на следующих стадиях;
- **разработка методов оценки (метрик) качества,** включая надежность, число ошибок и производительность (скажем, время отклика);
- **сбор схем использования,** покрывающих минимум 80% функциональных возможностей приложения;
- **подготовка заявки на выполнение проекта,** включая график, состав группы и стоимость проекта.

Фаза «Проработка» завершается созданием архитектуры приложения.

Создание

Фаза «Создание» — основная по времени и потреблению ресурсов. Она же требует наибольшего числа итераций. Цель этой фазы — создание приложения, а основная задача — завершить разработку при-

ложения и убедиться, что оно готово к переходному периоду. К началу переходного периода надо убедиться, что приложение достигло первоначальной стабильности и готово к бета-тестированию. Инкрементальный подход к разработке рекомендует последовательно наращивать функциональные возможности продукта.

На стадии «Создание»:

- **расширяется список схем использования**, включая уточнение, детализацию и описание всех схем;
- **завершается выполнение первых трех этапов;**
- **начинается тестирование** (обычно на этой фазе выполняется примерно 15% этапа «Тестирование»);
- **поддерживается целостность приложения** — все вносимые изменения не должны выходить за рамки утвержденной архитектуры;
- **ведется управление рисками,** выявленными на предыдущих стадиях.

Фаза «Создание» завершается этапом «Готовность к опытной эксплуатации».

Переходный период

Переходный период начинается с представления первой бета-версии приложения заказчику и ограниченному кругу пользователей. Основные задачи переходного периода — подготовить продукт к выпуску и пользователей к его эксплуатации. Для завершения этой фазы — этапа «Выпуск продукта» — необходимо обстоятельное тестирование продукта пользователями именно в той среде, где он будет эксплуатироваться.

В переходный период проводится:

- **подготовка развертывания,** включая подготовку инфраструктуры организации и узлов развертывания, а также информирование заказчика о необходимых изменениях;
- **подготовка документации,** включая документацию по эксплуатации, материалы для пользователей и другие материалы, которые должны сопровождать продукт;
- **оптимизация приложения** для работы в эксплуатационных условиях;
- **устранение проблем,** выявленных при бета-тестировании;
- **модификация приложения,** которая может понадобиться, если будут выявлены проблемы, не обнаруженные на предыдущих стадиях.

Итерации

Итерации каждой фазы продолжаются до тех пор, пока не достигнута цель этой фазы. По мере продвижения проекта от фазы к фазе степень важности и затраты на отдельные этапы изменяются; этот процесс проиллюстрирован на рис. 4.3 (он основан на диаграмме из цитированной выше книги Ивара Джекобсона и др.).

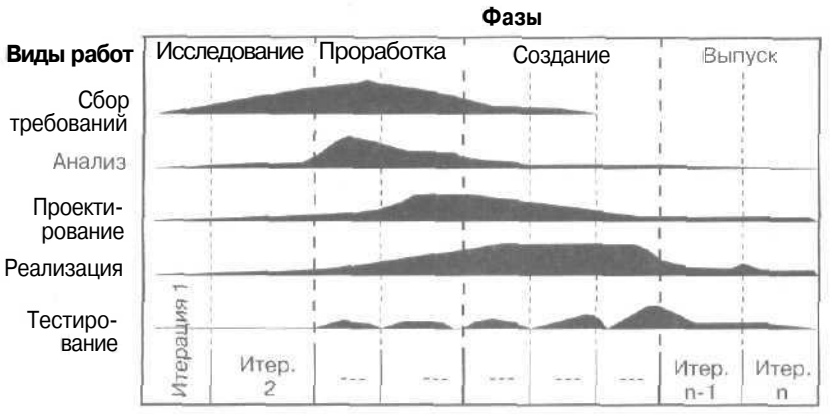


Рис. 4.3. Распределение приоритетов этапов на разных стадиях проекта

Кроме того, фазы «Исследование» и «Проработка» требуют повышенного внимания к управлению проектом и к организации среды разработки.

С течением времени степень детализации различных моделей универсального процесса растет, и модели постепенно приобретают завершенный вид. Как показано на рис. 4.4, также заимствованном из цитированной выше книги Джекобсона с соавторами, по окончании фазы «Создание» шесть основных моделей практически завершены, хотя обычно они еще требуют окончательной доводки (это делается в переходный период).

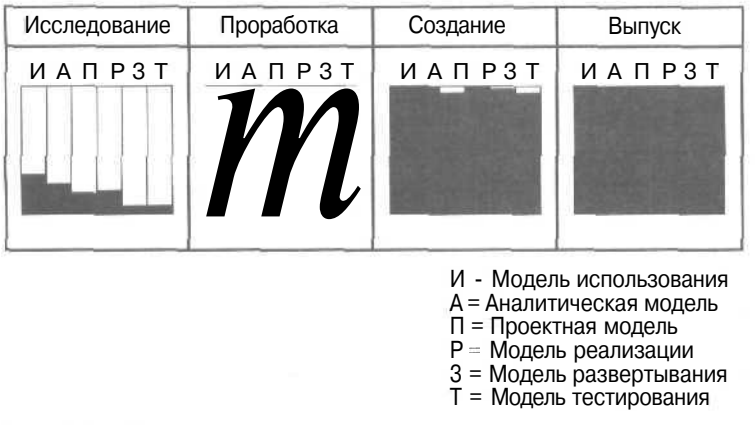


Рис. 4.4. Степень завершенности моделей на разных стадиях проекта

Джекобсон также приводит несколько важных последствий итерационной и инкрементальной природы универсального процесса.

- Одобрение проекта на стадии «Исследование» требует демонстрации методов снижения основных рисков и прототипа, доказывающего корректность концептуальной модели.
- Чтобы сделать обоснованную заявку на проект в конце стадии «Проработка», необходимо понимать, что заказывает организация (основой для понимания служат базовая архитектура и требования), и быть уверенным в отсутствии скрытых рисков (то есть недостаточно изученных возможностей увеличения стоимости и расползания графика).
- Снижение затрат, минимизация числа дефектов и сокращения времени на выпуск продукта требуют применения повторно используемых компонентов (естественное следствие ранней разработки архитектуры на основе исследования предметной области).
- Чтобы избежать задержек выпуска продукта, превышения бюджета и низкого качества, необходимо начинать с решения самых сложных вопросов.
- Единственная возможность избежать выпуска продуктов, которые устарели еще до выпуска, — перестать игнорировать изменения. Итерационный подход, включающий разные фазы, позволяет учитывать изменения по мере выполнения проекта.

Модель процесса разработки MSF

Как мы уже указывали, традиционные модели разработки программного обеспечения, например, модель водопада или спиральная модель, не соответствуют сложности проектов создания современных приложений масштаба предприятия.

В модели водопада проект выполняется последовательно, от первоначальной концепции до тестирования системы. Этапы этой модели служат для оценки результатов каждой фазы и являются точками перехода от одной фазы к другой. Этот метод подходит для проектов, где все требования можно сформулировать в самом начале, однако редко успешен в сложных проектах, для которых характерно изменение требований в ходе проекта. Кроме того, этот метод отличается наличием гор документации и однократным обзором на каждой из стадий проекта. Эти особенности модели водопада ведут к своеобразному «аналитическому параличу» и отсутствию хотя бы минимального взаимопонимания между разработчиками с одной стороны и заказчиком и пользователями — с другой.

В спиральной модели приложение разрабатывается в несколько итераций. Первые итерации жизненного цикла спиральной модели

позволяют уточнить модель приложения, а на последующих итерациях реализуется набор его функциональных возможностей. Спиральная модель позволяет выявить основные риски на возможно более ранних стадиях проекта и справиться с ними в первых версиях продукта. Благодаря своей итерационной природе спиральная модель легко адаптируется к изменениям требований, что позволяет повысить качество продукта. Однако для эффективности этого метода необходима высокая степень автоматизации процессов и документооборота. На практике этот метод часто вызывает у заказчика чувство нестабильности, поскольку продукт изменяется слишком быстро. И наконец, многие проекты, разрабатывавшиеся по спиральной модели, так и не заканчивались — итерационный цикл продолжался без конца.

Как показано на рис. 4.5, модель процесса разработки MSF сочетает сильные стороны обоих методов, позволяя воспользоваться преимуществами поэтапного подхода модели водопада и достоинствами итерационной разработки, присущей спиральной модели.

Рис. 4.5. Модель процесса разработки

Модель процесса разработки MSF имеет три отличительные особенности:

- разбиение на фазы (рис. 4.5);
- контроль выполнения работ на каждой фазе;
- итеративность (стрелка на рис. 4.5 возвращает процесс к первой фазе).

Хотя на рис. 4.5 все четыре фазы занимают по четверти времени, отведенного на проект, в действительности дело не всегда обстоит именно так. Распределение времени и ресурсов между фазами диктуется особенностями бизнес-проблемы и технологической инфраструктуры.

Фазы

Модель процесса разработки MSF состоит из четырех взаимосвязанных фаз. Каждая из них характеризуется своими задачами и результатами, которые должны быть достигнуты прежде, чем проектная группа сможет перейти к следующей фазе. Ниже перечислены четыре фазы модели процесса разработки и их основные результаты,

- Анализ, цель которого — выработать единую концепцию проекта.
- Проектирование, результаты которого — подробный план проекта и архитектура приложения.
- Разработка, цель которой — создание полнофункционального продукта.
- Стабилизация, задача которой — создание стабильного продукта, готового к развертыванию.

Этапы

Модель процесса разработки MSF базируется на этапах, представляющих собой точки синхронизации и обзора выполненной работы в противоположность традиционным методам, где они выступают в качестве моментов фиксации приложения или его спецификаций. Этапы позволяют проектной группе контролировать ход выполнения проекта и корректировать его — например, в связи с изменением требований заказчика или в ответ на реализацию какого-либо риска.

Этапы модели процесса разработки MSF подразделяются на *основные* и *промежуточные*. И те, и другие характеризуются необходимостью добиться некоторых *результатов* — только в этом случае проектная группа может считать этап пройденным.

Основные этапы

Каждая фаза процесса разработки завершается основным этапом, результаты которого представляются заказчику. Каждый основной этап — момент, когда все участники проектной группы могут и должны согласовать результаты своей работы. Кроме того, именно тогда внешние (по отношению к проектной группе) участники (заказчик, пользователи, группы эксплуатации и сопровождения и т. д.) знакомятся с состоянием проекта.

Важность основных этапов заключается в возможности постоянного контроля за ходом выполнения проекта. На каждом таком этапе проектная группа и заказчик, изучив результаты этапа, принимают совместное решение о необходимости перехода к следующей фазе. Таким образом, основные этапы играют роль точек перехода проекта из одной фазы в другую.

Промежуточные этапы

Каждая фаза процесса разработки в процессе выполнения проходит через промежуточные этапы, которые, как и основные, предназначены для согласования и синхронизации работ отдельных подгрупп проектной группы. Отличие промежуточных этапов от основных заключается в том, что их результаты не представляются внешним участникам проекта — они предназначены только для проектной группы.

Промежуточные этапы, как и основные, служат для отслеживания хода выполнения проекта, Кроме того, они позволяют разбить большой проект на управляемые и выполнимые части.

Итеративность

Одна из важнейших особенностей модели процесса разработки MSF — повторяемость. За время жизни продукта процесс повторяется несколько раз, что позволяет пересматривать и расширять функциональные возможности продукта в соответствии с изменением бизнес-требований.

Фазы процесса разработки MSF и их основные результаты

Каждая фаза процесса разработки MSF завершается основным этапом. В этом разделе мы вкратце опишем каждую из фаз с тем, чтобы дать читателю представление об основных задачах каждой фазы и их взаимосвязи. В следующих главах мы обсудим эти фазы во всех подробностях и опишем промежуточные этапы для каждой из них.

Фаза «Анализ»

Цель фазы «Анализ» — выработать единую концепцию проекта для всех его участников. Единая концепция предполагает наличие следующих составляющих.

- **Согласованное всеми сторонами понимание бизнес-проблемы, на решение которой направлен проект** — не раз случалось, что, только закончив разработку приложения, группа понимала, что приложение решает не ту проблему, которую ставил заказчик. У подобного финала могут быть разные причины, однако чаще всего это недостаточно тесное взаимодействие с заказчиком, особенно на ранних стадиях проекта. Избежать этого можно только одним способом — убедиться, что проектная группа правильно представляет себе бизнес-проблему, *прежде чем* она приступит к созданию продукта.
- **Решение, отвечающее ожиданиям заказчика** — ничуть не реже разработчики, закончив создание приложения, выясняют, что заказчик ожидал чего-то совсем другого. Мир становится все более компьютеризованным, и нынешние заказчики много лучше представ-

ляют себе технологии, чем прежде. Вполне возможно, что у заказчика есть свое мнение о будущем продукте и связанные с этим ожидания. Важная часть фазы «Анализ» — выяснение мнения заказчика и его ожиданий, связанных с продуктом, на ранней стадии проекта. Мы уже говорили в главе 3, что взаимодействие с заказчиком и согласование точек зрения на продукт — обязанность менеджера продукта.

- **Обоснованная оценка проектных ограничении — на** стадии «Анализ» формируется понимание важнейших ограничений — сроков, ресурсов и характеристик продукта. На этой фазе, как правило, возможны лишь очень предварительные оценки этих параметров.

Уточнение проектных ограничений и достижение компромисса между тремя важнейшими факторами достигается их последовательной оценкой. По мере анализа, создания прототипов и планирования группа может прийти к решению о необходимости частичного (или даже полного) пересмотра задач проекта в связи с:

- более полным осознанием требований заказчика;
- изменением бизнес-требований;
- выявлением технических проблем или рисков;
- необходимостью привести в соответствие сроки выполнения проекта, нужные ресурсы и характеристики продукта.

Концептуальное видение проекта, создаваемое на стадии «Анализ», позволяет очертить его рамки и готовит почву для более детального и формального планирования, которое выполняется на следующей стадии.

Этап «Одобрение концепции»

Стадия «Анализ» завершается этапом «Одобрение концепции». На этом, первом из основных этапов заказчик и проектная группа согласуют цели проекта, включая общие характеристики и функциональные возможности продукта.

Достижение этого этапа гарантирует объединение проектной группы, что является одним из важнейших условий успеха проекта. Группа должна ясно понимать, что она собирается сделать, и уметь формулировать свое видение так, чтобы добиться необходимой мотивации и заказчика, и проектной группы.

Перечислим результаты, которые необходимы для достижения этапа «Одобрение концепции» (все они представляются в виде документов):

- концепция;
- оценка основных рисков;
- структура проекта.

Кроме того, рекомендуется добавить к этому списку прототип приложения, демонстрирующий корректность и реализуемость концепции.

Как мы уже отмечали выше, основной этап — момент, когда проектная группа и заказчик согласуют результаты работы. На каждом основном этапе проектная группа и заказчик, изучив результаты этапа, принимают совместное решение о необходимости перехода к следующей фазе. Как отмечалось в главе 3, в крупных проектах основные этапы предполагают согласование результатов работы различных подгрупп проектной группы. Этап «Одобрение концепции» — первая возможность для заказчика и проектной группы определиться, стоит ли выполнять проект. Чтобы избежать недопонимания на следующих стадиях проекта, решение о переходе к следующей фазе должно быть принято всеми участниками проекта. Создание прототипа приложения на стадии «Анализ» помогает группе добиться ясного представления о создаваемом продукте.

Одобрение концепции означает, что проектная группа, заказчик и другие основные участники проекта согласовали:

- бизнес-задачи, на решение которых направлен проект;
- концепцию продукта;
- цели проектирования;
- риски, связанные с запуском проекта;
- базовую концепцию бизнес-решения;
- принципы управления проектом и состав проектной группы.

Подводя итог, еще раз напомним, что основная цель фазы «Анализ» — добиться консенсуса между всеми участниками проекта. Создание концепции позволяет всем участниками проекта оценить ее и принять решение о необходимости разработки проекта. Понимание, согласие и приверженность концепции создают идеальную почву для перехода к фазе «Планирование».

Фаза «Планирование»

Мы все знаем о правиле «Сначала работай над планом, потом работай по плану», однако каждодневная текучка и сроки часто заставляют нас пренебречь планированием. Одна из причин популярности модели процесса разработки MSF — огромное внимание, которое уделяется в этой модели планированию и проектированию.

Почему планирование столь важно? Причина проста: чем раньше обнаружены недостатки проекта, тем дешевле обходится их устранение. Относительная стоимость проектных ошибок, выявленных на разных стадиях выполнения проекта, проиллюстрирована на рис. 4.6. Диаграмма ясно показывает, что тщательное планирование приносит плоды, снижая расходы времени и ресурсов на устранение проблем на заключительных стадиях проекта.

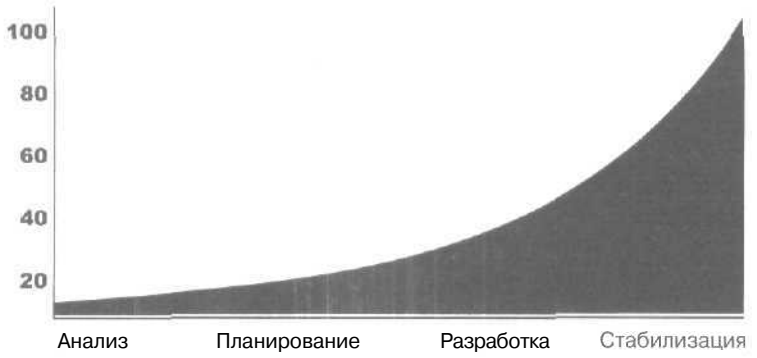


Рис. 4.6. Стоимость исправления неверных проектных решений на разных стадиях

Недостаточное планирование — распространенная ошибка, однако встречается и другая крайность — избыточное планирование. Не стоит впадать в «аналитический паралич»: планирование важно, но не менее важно знать, когда пора перестать детализировать планы и перейти к делу. Как правило, объем необходимого планирования зависит от масштаба проекта (оцениваемого ожидаемым числом строк кода или числом компонентов) и его важности.

На стадии «Планирование» определяется архитектура продукта. Как мы отмечали в главе 2. архитектура приложения основана на концептуальной, логической и физической проектных моделях, которые подробно обсуждаются в главе 6. Важная составляющая разработки надежного и хорошо масштабируемого приложения — применение современной многоуровневой архитектуры, которую можно реализовать в виде монолитного, клиент-серверного или многоуровневого приложения.

Компромиссный треугольник

При обсуждении фазы «Анализ» мы отмечали, что любой проект характеризуется тремя важнейшими факторами: графиком, ресурсами и характеристиками продукта. Эти факторы проиллюстрированы на рис. 4.7 в виде так называемого *компромиссного треугольника.* На стадии «Планирование» оценки этих факторов постепенно уточняются, и к концу этой фазы группа определяет график, в который она собирается уложиться, ресурсы, которые ей понадобятся, и характеристики продукта, которые будут реализованы.

На стадии «Планирование» группа добивается первоначального баланса трех факторов, или, иначе говоря, равновесия сторон компромиссного треугольника. Одна из главных проблем следующих фаз

— сохранить это равновесие. Мы подробнее обсудим компромиссный треугольник в следующих разделах.

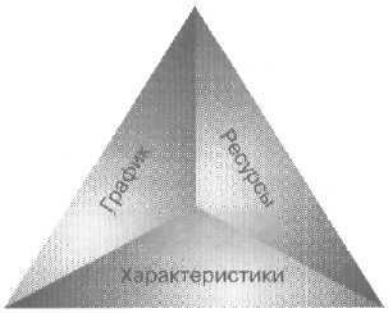


Рис. 4,7. **Компромиссный треугольник**

Этап «Одобрение **плана проекта**»

Фаза «Планирование» завершается этапом «Одобрение плана проекта», который знаменует достижение согласия по основным вопросам проекта между проектной группой, заказчиком и другими участниками проекта. Этот этап позволяет установить приоритеты и оценить перспективы. По сути, именно на этом этапе проектная группа и заказчик заключают контракт (хотя юридически это может случиться как раньше, так и позже). По завершении этого этапа группа может переходить к созданию продукта.

Для достижения этого этапа необходимы:

- функциональные спецификации;
- основной план проекта;
- основной график проекта;
- пересмотренный документ оценки рисков.

Кроме того, мы рекомендуем включить в этот список прототип системы, демонстрирующий корректность проектных решений и позволяющий основным участникам проекта оценить архитектуру приложения. Кроме того, прототип позволяет проверить реализуемость проектных решений до достижения этапа «Одобрение плана проекта».

Достижение этого этапа означает, что проектная группа, заказчик и другие основные участники проекта достигли взаимопонимания по следующим вопросам:

- что нужно сделать, чтобы продукт соответствовал потребностям бизнеса;
- приоритетные задачи;
- сколько времени потребуется для завершения проекта;
- как создавать продукт и кто будет этим заниматься;

- риски, связанные с созданием продукта;
- промежуточные этапы и их результаты.

Фаза «Разработка»

Теперь разработчики уже готовы писать код. Сейчас мы покажем, что именно к этому проектная группа готовилась на двух предыдущих стадиях. Теперь, на стадии «Разработка», эта задача — создание приложения — становится главной.

Третья часть этой книги целиком посвящена фазе «Разработка», в частности, средствам и методам ее выполнения. Пока же скажем лишь, что оно заметно упрощается благодаря работе, проделанной на фазах «Анализ» и «Планирование». Совершенно ясно, что гораздо легче разрабатывать приложение с четким пониманием его задач и четко описанной и протестированной архитектурой.

Итеративный подход, применявшийся на предыдущих стадиях, на этой стадии приобретает особое значение. На стадии «Разработка» группа, как правило, последовательно выпускает несколько версий приложения. Эти версии, для которых используются названия альфа-, бета- и окончательная версия, будут подробно обсуждаться в этой и последующих главах.

Кроме того, на этой стадии группа занимается всеми выявленными проблемами. Не следует рассчитывать на полное устранение всех ошибок и проблем на стадии «Разработка» — достаточно, если они будут исследованы. Помните, что цель этой стадии — создать приложение, отвечающее заявленным требованиям и готовое к тестированию.

Этап «Завершение разработки»

Основная цель стадии «Разработка» — достижение этапа «Завершение разработки», знаменующего полную реализацию всех функциональных возможностей и готовность продукта ко внешнему тестированию и стабилизации. На этом этапе заказчик, пользователи, группы эксплуатации и сопровождения могут оценить законченный продукт и выявить проблемы, которые нужно устранить до выпуска продукта.

Достижение этапа «Завершение разработки» характеризуется следующими результатами:

- законченной версией пересмотренных функциональных спецификаций;
- пересмотренным планом и графиком проекта;
- пересмотренным сводным документом оценки рисков;
- исходными текстами приложения и исполняемыми модулями;
- средствами повышения эффективности работы пользователей и сопроводительными материалами;
- тестовыми спецификациями и схемами тестирования.

На этом этапе группа должна завершить разработку и тестирование всех функциональных возможностей продукта. Работы по дополнительной оптимизации кода, а также выявление и устранение ошибок, могут продолжаться на стадии «Стабилизация».

Достижение этапа «Завершение разработки» означает, что проектная группа не собирается реализовывать никакие дополнительные функциональные возможности продукта, а все участники проекта согласны с:

- полнотой реализации запланированных функциональных возможностей продукта;
- производительностью продукта;
- готовностью продукта:
- стратегией тестирования и развертывания продукта (включая бета-версии).

Фаза «Стабилизация»

Профессионалы хорошо знают, что тестирование — важнейшая часть любого проекта. К сожалению, это понимают не все, и многие разработчики не уделяют должного внимания тестированию приложений. Трудно сказать, почему они решают проигнорировать тестирование, — возможно, считают свой код безошибочным, а может быть, наоборот, догадываются, что их код так плох, что лучше его вовсе не тестировать.

Опытные разработчики отлично знают, что безошибочного кода не бывает, и лишь тщательное тестирование способно снизить число ошибок до приемлемого уровня. Опытный разработчик отличается от новичка тем, что не просто ожидает тестирования, а требует его,

---

Внимание! Очень важно подготовить пользователей к началу стадии «Стабилизация». Готовя пилотное развертывание, помните, что его главная цель — выявить проблемы производительности и совместимости. Выявленные проблемы надо устранить в следующих выпусках, после чего можно переходить к развертыванию.

Функциональные возможности приложения тестируются на стадии «Разработка», а производительность и совместимость — на стадии «Стабилизация». На этой стадии устраняются все найденные проблемы, а также завершается создание всех материалов, необходимых группам эксплуатации и сопровождения. Вообще одна из задач стадии «Стабилизация» — доделать все, что не закончено. Документация, инструкции по установке, окончательный список проблем с рекомендациями для будущих версий, руководство по развертыванию —

все эти материалы на стадии «Стабилизация» приобретают окончательный вид.

Фаза «Стабилизация» начинается, когда проектная группа переносит основное внимание с разработки на стабилизацию и выпуск продукта, и заканчивается, когда заказчик принимает продукт. Важная особенность этой фазы — участие пользователей в тестировании продукта. Кроме того, она является периодом интенсивного обучения групп эксплуатации и сопровождения. На этой стадии группа логистики передает обязанности по сопровождению продукта группам, ответственным за эту работу в организации. Процесс завершается выпуском продукта.

Этап **«Выпуск продукта»**

Достижение этапа «Выпуск продукта» — главная задача проектной группы. Он свидетельствует о завершении работы над продуктом и готовности к развертыванию. Кроме того, на этом этапе происходит перераспределение ответственности за продукт — от группы разработки к группе логистики и сопровождения.

Для достижения этапа «Выпуск продукта» необходимы следующие результаты:

- окончательная версия продукта;
- документация к окончательной версии;
- материалы для сопровождения приложения и поддержки пользователей;
- результаты и средства тестирования;
- архивы проекта;
- обзор всех основных этапов проекта.

На этом этапе продукт готов к выпуску и эксплуатации. Этап «Выпуск продукта» свидетельствует о том, что все участники согласны с тем, что:

- продукт стабилен и все известные ошибки устранены;
- продукт принят заказчиком;
- ответственность за сопровождение передается группе логистики и сопровождения;
- группа начинает работу над следующей версией продукта.

Важность всех фаз

Начинающие часто намеренно сокращают первую и последнюю стадии модели процесса разработки MSF. Мы уже видели, что у всех проектов разработки программного обеспечения есть две общие составляющие: проектирование и кодирование. Эти составляющие практически точно проецируются на две фазы модели процесса разработки: «Планирование» и «Разработка».

Зрелость организации или проектной группы определяется уровнем внимания к стадиям «Анализ» и «Стабилизация». Обе эти фазы характеризуются высокой степенью вовлеченности заказчика и других участников проекта, не входящих в проектную группу. Помните, что проектная группа, не желающая или не умеющая вовлекать всех участников в выполнение проекта, никогда не сможет полностью реализовать свой потенциал.

Одно из важнейших достоинств модели проектной группы, которую мы обсуждали в главе 3, — наличие ролей, ответственных за взаимодействие с заказчиком и пользователями. Применение этой модели гарантирует, что проектная группа не проигнорирует стадии «Анализ» и «Стабилизация», а взаимодействие с заказчиком и пользователями будет профессионально организовано. Группа должно ясно понимать, что успех проекта зависит не только от проектирования и кодирования, но и от двух других фаз.

Принципы модели процесса разработки

Модель процесса разработки MSF играет ключевую роль в организации процесса разработки, указывая, какие действия и когда должны выполняться. У этой модели есть еще две важные особенности. Первая — это тесная связь с моделью проектной группы, сочетание которой с моделью процесса разработки значительно повышает эффективность процесса. Вторая особенность — это принципы и практика модели процесса разработки:

- выпуск версий продукта;
- постоянно «живущие» документы;
- планирование процесса с учетом неопределенностей;
- поиск компромиссов;
- управление рисками;
- ориентация на выпуск продукта в срок;
- разбиение больших проектов на управляемые части;
- ежедневная сборка продукта;
- контроль «снизу — вверх».

Выпуск версий

Мы рекомендуем придерживаться стратегии, разбивающей большой проект на несколько последовательных выпусков версий продукта без промежуточной фазы сопровождения. Найдя компромисс между ограничениями и характеристиками продукта и приняв план выпуска продукта, группа должна как можно скорее начать цикл выпуска версий. Выпуск версий продукта позволяет проектной группе своевременно реагировать на изменение требований, графика и рисков. Регулярное

обновление продукта позволяет поддерживать постоянный контакт с заказчиком и учитывать его пожелания в следующих выпусках.

Первая версия продукта должна включать базовый набор функциональных возможностей, который последовательно расширяется в следующих выпусках. В случае изменения требований или представления о продукте следующие версии могут адекватно отразить эти изменения.

Подводя итог, перечислим преимущества стратегии последовательного выпуска версий продукта.

- **Контакт с заказчиком** — выпуск версий позволяет постоянно поддерживать контакт группы с заказчиком, информируя его о состоянии продукта, и учитывать в следующих выпусках лучшие идеи обеих сторон.
- **Ранняя версия** — группа может быстро выпустить базовый набор функциональных возможностей и собрать отклики от заказчика и пользователей. Если заказчик видит, что работа над продуктом идет в соответствии с графиком, он гораздо спокойнее относится к переносу некоторых функциональных возможностей в следующий выпуск.
- **Ограниченный круг решаемых вопросов** — выпуск версий позволяет всегда иметь дело с относительно компактным кругом вопросов и решать эти проблемы на стадии выпуска очередной версии.
- **Ясность целей** — выпуск версий ставит перед группой четкие и ясные задачи, что позволяет сосредоточиться на решении вопросов, связанных с текущей версией, быстрее добиваться результата и постоянно прогрессировать. Роль версий велика и в уточнении графика — они позволяют разбить работу на небольшие части, которые хорошо управляемы, имеют ясную цель и дают конкретный результат.
- **Свобода и гибкость** — выпуск версий дает группе больше свободы в выборе приоритетов и придает дополнительную гибкость процессу проектирования, позволяя своевременно реагировать на изменение бизнес-требований. Гибкость подхода снижает общую неопределенность за счет распределения конкретных составляющих и функциональных возможностей по версиям; например, при изменении требований достаточно просто изменить приоритеты для очередного выпуска. Одновременно группа всегда располагает стабильной версией продукта.
- **Последовательное и постоянное расширение** функциональных **возможностей продукта** — выпуск версий позволяет группе планомерно расширять функциональные возможности продукта. Последо-

вательность и планомерность производят чрезвычайно выигрышное впечатление на заказчика.

В своей книге «Rapid Development: Taming Wild Software Schedules» Стив Макконнелл замечает:

> *«Одно из важнейших следствий согласия заказчика и пользователей на перенос части функциональных возможностей в следующую версию — их уверенность в том, что следующая версия действительно появится. Если такой уверенности нет, они будут требовать реализации всех (по их мнению) необходимых характеристик в текущей версии, что вряд ли упростит жизнь разработчикам. Частый выпуск версий повышает уверенность заказчика в том, что все требования будут реализованы».*

Еще один способ убедить заказчика и пользователей — предусмотреть выпуск нескольких версий продукта в плане с самого начала. Такой план с распределением функциональных возможностей по версиям также позволяет повысить уверенность заказчика в том, что проект будет реализован.

«Живые» документы

Хотя адекватное планирование жизненно важно для успеха проекта, нужно соблюдать меру — избыточное планирование деструктивно, Как мы уже отмечали, чрезмерно детальное планирование способно вызвать «аналитический паралич». Чтобы избежать бесконечного топтания на фазе «Планирование», проектная группа должна как можно раньше определить степень детализации при планировании, что позволит перейти к разработке, даже если какие-то вопросы остаются неясными. С другой стороны, в любом проекте присутствует доля неопределенности, связанная с возможными изменениями требований, графика и ресурсов, поэтому фиксировать результаты фазы «Планирование» стоит лишь в случае, когда обратное сопряжено со значительным риском.

Концепция «ранняя определенность, поздняя фиксация» лежит в основе концепции «живых» документов — они названы так, потому что постоянно растут и видоизменяются в ходе проекта. Один из признаков зрелости проекта и проектной группы — признание факта необходимости регулярного пересмотра основной проектной документации. С другой стороны, зрелые проекты имеют, как правило, и хорошо управляемый процесс внесения изменений, гарантирующий изменение основной документации, только когда это действительно необходимо.

Планирование процесса

Неопределенность и непредсказуемость неизбежны. Раз уж жизнь полна неожиданностей, в плане проекта должна учитываться возможность появления непредсказуемых факторов. Мы рекомендуем два подхода к этой нелегкой задаче: резерв времени и планирование на основе рисков.

Резерв времени

Обычно срок выпуска продукта определяется простым суммированием оценок времени на выполнение основных этапов проекта и прибавлением полученной суммы к дате начала проекта. Часто эти оценки заранее увеличивают на какой-то процент, чтобы учесть возможные задержки. Отметим, что учет возможных задержек и резервное время — не одно и то же.

Воспользуемся аналогией: резерв времени аналогичен войсковому резерву, который военачальник держит на случай непредвиденных изменений плана сражения. При разработке программного обеспечения резервное время — это предусмотренный менеджером программы период времени в конце графика проекта. Резервом времени распоряжается менеджер программы, причем по своему усмотрению в соответствии с особенностями выполнения различных фаз проекта. Как показано на рис. 4.8, резервное время не распределяется между отдельными этапами — они по-прежнему должны завершаться в срок по графику. Помните, резервное время — не панацея от неудачного планирования, а попытка снизить риск влияния непредсказуемых факторов. За использование резервного времени всегда приходится платить, даже если причина для использования резерва достаточно веская.

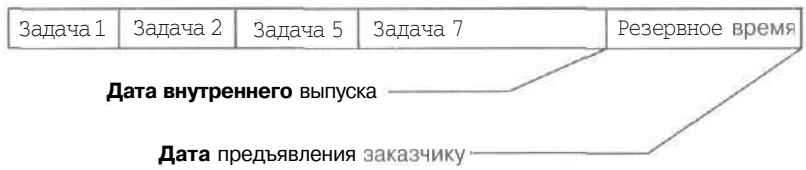


Рис. 4.8. График проекта должен предусматривать резерв времени

Добавление резервного времени к графику приводит к появлению двух дат выпуска. При составлении графика «снизу — вверх» определяется внутренняя дата выпуска. Добавив к ней резервное время, вы получите внешнюю дату выпуска для заказчика.

Планирование с учетом рисков

Это способ, при котором задачи с высокой степенью риска получают высокий приоритет, а задачи, сопряженные с малым риском, — низкий. Если задача, связанная с высокой степенью риска, потребует

больше времени, такой план позволит увеличить выделенное время. У этого метода есть несколько серьезных достоинств:

- он стимулирует раннее создание прототипов, проверяющих корректность концепций;
- позволяет быстро решить, какой набор функциональных возможностей когда выпускать;
- позволяет расставить приоритеты на основе технических и бизнес-рисков;
- стимулирует разработчиков стремиться к раннему выпуску продукта;
- в случае несоблюдения даты выпуска позволяет быстро выяснить причины и найти необходимые компромиссы;
- выявление рисков, наиболее опасных для проекта, позволяет достичь понимания с заказчиком.

Поиск компромиссов

В своей книге «Dynamics of Software Development» Джим Маккарти указывает, что успех любого проекта зависит от равновесия трех важнейших элементов:

> *«...в конце концов, в вашем распоряжении только три вещи: ресурсы (люди и деньги), характеристики (продукт и его качество) и график. Это и есть треугольник ваших возможностей — больше у вас ничего нет. И помните — изменение любого элемента обязательно влечет за собой изменение как минимум еще одного, а чаще всего двух элементов».*

В начале проекта взаимосвязь этих элементов довольно туманна. На этом этапе группа располагает лишь приблизительными оценками того, что предстоит сделать, какие ресурсы потребуются и когда продукт будет готов. На стадии «Планирование» стороны компромиссного треугольника постепенно приобретают большую определенность. По окончании этой фазы группа должна отчетливо представлять себе доступные ресурсы, характеристики продукта и дату выпуска.

Важно помнить, что три стороны компромиссного треугольника взаимосвязаны. Изменение одной из них влияет на остальные. Понимание и применение этой концепции дает проектной группе мощный инструмент адекватной реакции на изменение требований или условий в ходе проекта.

Рассмотрим пример: пусть компромиссный треугольник показывает, что, используя 10% ресурсов, группа сможет реализовать 20% функциональных возможностей к 1 июня. Допустим, в процессе разработки были обнаружены непредусмотренные характеристики про-

дукта, которые необходимо реализовать. Это обстоятельство нарушает равновесие треугольника. Как показывает рис. 4.9, группа восстанавливает равновесие посредством сокращения объема запланированных к реализации функциональных возможностей продукта, привлечения новых ресурсов, изменения даты выпуска или сочетания этих мер.

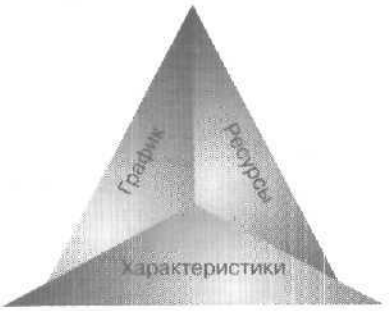


Рис. 4.9. Несбалансированный компромиссный треугольник

Преимущество этого подхода заключается в его простоте. Идея настолько проста, что ее можно объяснить заказчику за обедом, нарисовав треугольник на салфетке и объяснив, какие компромиссы необходимы для успеха проекта.

Хотя компромиссный треугольник — простой и эффективный инструмент, он не отражает приоритетов составляющих его элементов. Один из методов описания приоритетов и уточнения ожиданий проектной группы и заказчика — создание матрицы альтернатив, пример которой приведен в таблице 4.1. Такая матрица позволяет заказчику и группе согласовать важность различных сторон компромиссного треугольника и их приоритеты: последнее важно, когда приходится решать, чем поступиться при поиске компромисса.

Табл. 4.1. Пример матрицы альтернатив

Проект 1				Проект 2			
	Оптимизация	**Ограничение**	**Как получится**		**Оптимизация**	**Ограниченне**	**Как получится**
Ресурсы		✓		**Ресурсы**	✓		
Дата выпуска	✓			**Дата выпуска**			✓
Характеристики			✓	Характеристики		✓	

Проектная группа и заказчик вместе решают, что можно делать с каждым из переменных элементов. Обратите внимание, что в каждой строке и каждом столбце матрицы альтернатив может стоять только одна галочка — смешанные варианты несут с собой серьезный риск для проекта. Столбцы матрицы имеют следующий смысл.

- **Оптимизация** — требует наилучшего значения элемента в конце проекта. Например, оптимизация ресурсов — это их минимальное использование, оптимизация даты выпуска — выпуск продукта как можно раньше (стратегия скорейшего выхода на рынок), а оптимизация характеристик — выпуск продукта с максимальным набором функциональных возможностей.
- **Ограничение** — ограниченной переменной просто присвоено фиксированное значение; смысл ограничения ресурсов ясен, ограничение даты выпуска означает выпуск продукта в заданном временном интервале, а ограничение характеристик — это стратегия выпуска продукта с базовым набором функциональных возможностей.
- **Как получится** — когда одна переменная ограничена, а вторая подвергается оптимизации, значение третьей переменной определяется значениями первых двух. Скажем, если первые две переменные — дата выпуска и характеристики, то ресурсы проекта не ограничиваются. Другая стратегия — создание приложения с фиксированным набором характеристик с использованием минимума ресурсов; в этом случае продукт будет выпущен тогда, когда будет готов (дата выпуска не фиксируется). Галочка в этом столбце означает, что соответствующим элементом распоряжается проектная группа. Например, для выпуска приложения с максимальным набором характеристик к заданному сроку заказчик должен согласиться на использование ресурсов в том объеме, в котором их потребует проектная группа.

Матрица альтернатив — удобный метод принятия решений. Она не является истиной в последней инстанции, однако помогает достичь взаимопонимания с заказчиком. Полезность матрицы альтернатив для проектной группы определяется тем, что она указывает области, которыми заказчик готов поступиться.

Управление рисками

Для большинства проектов управление рисками — важнейший фактор успеха. Чтобы справиться с проектом, проектная группа должна:

- обучаться;
- быстро адаптироваться к переменам;
- предусматривать изменения;
- действовать эффективно.

Способность понимать происходящее вокруг проектной группы и умение предпринимать действия, снижающие неопределенность и повышающие стабильность и предсказуемость, позволит группе работать одинаково успешно и в стабильной обстановке, и в условиях полной неразберихи. Именно такая готовность к неожиданностям и является целью активного анализа рисков и управления ими.

Существует два принципиально разных подхода к управлению рисками. Большинство групп в качестве метода управления рисками практикуют *реагирование* — они так или иначе реагируют на уже возникшую проблему. Мы — сторонники *превентивного управления рисками.* Этот подход, который подробно обсуждается в главе 5, предполагает наличие продуманного плана и процесса управления рисками до их реализации, то есть до возникновения проблемы. Процесс управления рисками должен быть не только продуманным, но и постоянным. При превентивном управлении риски постоянно контролируются, а информация о состоянии важнейших рисков служит основой для принятия решений на всех стадиях проекта. Управление рисками продолжается до их исчезновения или, если проблема все же возникла, до ее устранения.

Ориентация на выпуск в срок

Ориентация на выпуск в срок означает отношение к дате выпуска продукта как к императиву, Естественно, сначала группа планирует выполнение проекта и согласует с заказчиком все элементы компромиссного треугольника. Когда же дата выпуска согласована, этот элемент компромиссного треугольника больше не используется при принятии каких-либо решений корректирующего характера (конечно же, за исключением безвыходных ситуаций).

Ориентация на выпуск продукта в срок имеет массу достоинств.

- **Мобилизует проектную группу** — для выпуска продукта в срок проектной группе придется мобилизовать все свои способности.
- **Расставляет приоритеты по степени важности задач** — функциональные возможности продукта упорядочиваются по приоритетам, чтобы при необходимости можно было отказаться от реализации характеристик с низким приоритетом. Даже если для соблюдения даты выпуска придется отказаться от части функциональных возможностей, все важнейшие возможности продукта будут реализованы благодаря своей приоритетности.
- **Предоставляет в распоряжение группы мощный инструмент принятия решений** — все решения принимаются на основе их связи с выпуском продукта в срок.

- **Обеспечивает мотивацию проектной группы** — отсутствие фиксированной даты выпуска значительно ухудшает моральный климат в коллективе, создавая ощущение бесцельности.

Выпуск продукта к фиксированной дате обеспечивается планированием «снизу — вверх» и наличием резерва времени. Очевидно, тщательное планирование является залогом выпуска продукта в срок,

Разбиение больших проектов на управляемые части

Задачи большого проекта лучше разделить на несколько компактных и, желательно, независимых частей, которые следует трактовать как отдельные проекты или внутренние выпуски продукта. Такой метод можно рассматривать как выпуск нескольких версий в рамках одного проекта, когда финальная версия продукта выпускается только в конце проекта.

Каждый внутренний выпуск требует от двух до четырех месяцев; для каждого из них следует предусмотреть время и ресурсы на разработку, оптимизацию, тестирование и стабилизацию. Кроме того, надо предусмотреть примерно 25-процентный резерв времени на случай непредвиденных обстоятельств.

Для каждого внутреннего выпуска группа разработки реализует конкретный набор функциональных возможностей. Если он допускает тестирование, группа выполняет полный цикл тестирования, отладки и повторного тестирования, как если бы продукт готовился к сдаче заказчику. Когда качество кода станет удовлетворительным, группа переходит к разработке следующего набора функциональных возможностей. Этот метод также позволяет избежать проблем, которые характерны для интеграции всех компонентов большого проекта лишь на последней стадии.

Подразделение больших проектов на компактные составляющие:

- позволяет проектной группе сосредоточиться на конкретной работе по выполнению сравнительно небольших и потому лучше управляемых независимых проектов;
- приносит удовлетворение от завершения промежуточных проектов, избавляя от ощущения безысходности, которое появляется у разработчиков больших проектов;
- вовремя сигнализирует о проблемах, поскольку завершение отдельных проектов служит промежуточными этапами большого проекта; если сдача какого-то проекта задерживается, группа имеет возможность скорректировать ход проекта в целом;
- повышает качество конечного продукта, поскольку каждый внутренний проект проходит отдельный контроль качества;
- позволяет группе приобрести опыт сдачи продукта на промежуточных этапах, тем самым повышая предсказуемость процедуры сдачи продукта в конце проекта.

В своей книге «Debugging the Development Process» (Microsoft Press, 1994) Стив Магуайр (Steve Maguire) пишет:

«Энтузиазм порождает не двухмесячный график; энтузиазм возникает как следствие регулярной сдачи маленьких проектов и ощущения, что сделана пусть небольшая, но интересная и важная работа.

Конечно, можно работать иначе — например, «сначала выполнить все важнейшие задачи». Однако «важнейшие задачи» не составляют отдельный проект — это просто список всего, что заказчик считает главным, поэтому такой метод не рождает дополнительной мотивации.

Например, «подсистема создания графиков» — это самостоятельный проект, со своими целями и задачами. Вы можете составить список «самого главного» по этой теме, но воодушевлять людей будет не он, а ощущение работы над конкретным проектом. У группы появляется реальная цель — не выполнить 352 отдельных требования заказчика, а полностью создать — и выпустить — подсистему, решающую конкретную задачу. В результате эта работа станет самостоятельной и закончится как отдельный проект, с тестированием, стабилизацией и выпуском качественного продукта».

Ежедневная сборка

Обычно в ходе проекта приходится «собирать» исполняемый модуль (или модули) из сотен или даже тысяч исходных файлов. Некоторые проектные группы практикуют ежедневную сборку приложения с последующей проверкой работоспособности исполняемого модуля. Такая проверка очень важна — без нее ежедневная сборка приложения не имеет смысла. Тестирование сборки подробнее обсуждается в главе 12.

Ежедневная сборка имеет несколько важных достоинств:

- минимизирует риски, связанные с интеграцией кода — при ежедневной сборке проблемы этого сорта выявляются на ранней стадии, что позволяет вовремя отладить модуль, вызвавший проблему, или изменить соответствующее проектное решение;
- упрощает поиск причин проблемы — при таком подходе не только проще выявить некорректный код, но и выяснить, что и когда случилось с продуктом, прежде чем он перестал работать;
- снижает риск падения качества.

Чтобы эти достоинства реализовались, сборку и тестирование надо выполнять ежедневно, а не раз в неделю или раз в месяц. Собранный модуль должен работать — в противном случае сборка считается неудачной, и необходимо сразу же выяснить причины неудачи и устранить их. Ежедневная сборка и тестирование — своего рода постоянная имитация сдачи продукта, которая укрепляет дисциплину.

Стандарты ежедневной сборки и тестирования в разных проектах различаются, однако можно порекомендовать следующие минимальные требования:

- удачная компиляция всех файлов и компонентов;
- удачная сборка всех модулей и компонентов;
- отсутствие ошибок, препятствующих запуску приложения;
- прохождение теста на работоспособность.

Планирование «снизу — вверх»

Попросту говоря, работу должны планировать те, кто ее делает. Такой подход:

- **повышает точность оценок,** поскольку в этом случае они основаны на опыте конкретного разработчика, уже решавшего подобные задачи, а не взяты *«с* потолка» руководителем;
- **повышает ответственность исполнителей —** при таком подходе план одновременно является обязательством сотрудника, давшего оценку; конечно же, предварительные оценки могут оказаться (и всегда оказываются) завышенными, но анализ результатов промежуточных этапов, знание проекта и постоянная практика оценивания быстро делают оценки реалистичными и мотивирующими соблюдение графика.

Как показывает рис. 4.10, планирование — задача всей проектной группы. После наполнения общего плана проекта конкретными оценками, полученными от исполнителей, руководство должно добавить резерв времени, чтобы гарантировать соблюдение графика даже при возникновении непредвиденных обстоятельств. Таким образом становится возможной ориентация на выпуск продукта в срок.

Рис. 4.10. Составление графика «снизу — вверх»

Версии

Мы постоянно подчеркивали и будем подчеркивать важность концепции выпуска версий. Как показано на рис. 4.11, выпуск версий предполагает многократное прохождение цикла модели процесса разработки с постепенным расширением функциональных возможностей продукта. В этом разделе мы обсудим концепцию несколько подробнее, начав с рекомендаций по организации выпуска версий продукта. Затем мы поясним, как на собственном опыте убедились в возможности расширения этой концепции — вплоть до многократных итераций модели процесса разработки в рамках создания каждой промежуточной версии,

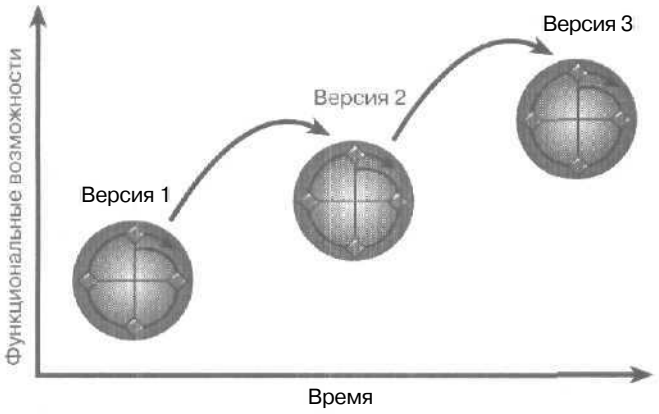


Рис. 4.11. Выпуск версий продукта на стадии «Разработка»

Рекомендации по организации выпуска версий продукта

Приступив к работе над первой версией нового приложения, проектная группа должна сразу же начать планировать выпуск второй версии. Этот итерационный подход очень полезен на всех четырех стадиях модели процесса разработки. Он, как минимум, позволяет распределять функциональные возможности продукта по нескольким версиям, что гарантирует реализацию всех базовых характеристик продукта в первой версии и придает заказчику уверенности в будущем проекта. Кроме того, сам факт последующего итерационного уточнения комплектации версий заставляет всех членов группы подходить к принятию решений взвешенно и серьезно.

Приведем рекомендации общего характера.

- **Ориентируйтесь на продукт** — всегда помните, что ваша главная задача — выпустить версию, реализующую все затребованные заказчиком функциональные возможности. Этот подход позволяет

группе постоянно смотреть чуть-чуть вперед, не ограничиваясь текущей версией, и всегда помнить о необходимости выполнить требования заказчика в полном объеме.

- **Создайте план выпуска версий** — помните, что заказчик и пользователи всегда спокойнее относятся к отсутствию каких-то функциональных возможностей, если знают, что их реализация запланирована в следующих версиях.
- **Проанализируйте все характеристики продукта с точки зрения важности, реализуемости и приоритетности** — всегда помните о конечной цели проекта — это позволит вам принимать разумные решения о том, какие функциональные возможности продукта необходимо реализовать в первой версии, а какие можно отложить до следующей.
- **Реализуйте базовые возможности в первой версии** — именно в ней надо решать основные задачи проекта. Демонстрация понимания этих задач и создание работоспособного программного обеспечения, решающего эти задачи, значительно укрепит доверие заказчика к проектной группе.
- **Если версия не решает задач бизнеса, прекратите работу над ней** - всегда знайте, когда нужно остановиться. Если нет причин выпускать следующую версию, не выпускайте ее.

Фазы и документы

В начале этой главы мы уже рассказывали о «живых» документах — документах, которые пересматриваются и уточняются все время создания проекта. Их необходимость вызвана обязательной гибкостью процесса разработки.

Например, один из результатов фазы «Анализ» — концепция проекта, описанная в одноименном документе. При традиционном подходе к разработке — скажем, в модели водопада — этот документ должен быть полностью завершен до начала стадии «Планирование», Модель процесса разработки MSF, напротив, не только предполагает, что это документ не будет закончен до начала стадии «Планирование», но и допускает его незавершенность до окончания проекта. На стадии «Анализ» создается базовое описание концепции и, соответственно, базовая версия документа, однако этот документ *не будет закончен* до конца проекта.

Разработка на других стадиях

По нашему мнению, часто оказывается полезно выполнять часть работ стадии «Разработка» во время выполнения других фаз проекта. Таким образом вы получаете два важных преимущества.

- **Движение вперед** — неопытные группы уделяют слишком мало внимания анализу и планированию; их цель — поскорее добраться до кодирования. Набив шишек на этом пути (и часто провалив не один проект), такая группа впадает в другую крайность — не приступает к кодированию, пока не проработаны мельчайшие детали плана. Возможность организовать естественный цикл «планирование — разработка» для решения отдельных задач позволяет группе избежать такого шараханья. Кроме того, работа нал частями кода приложения дает группе ощущение видимого прогресса. Ни одна из задач не тянется бесконечно долго, группа укладывается в график, выполняет план и все более убеждается в успехе проекта. Это ощущение удачи само по себе служит отличным стимулом для роста уверенности, что положительно сказывается и на моральном климате, и на квалификации сотрудников.
- **Раннее предупреждение** — один из важнейших недостатков традиционных методов разработки — невозможность выявить неудачное проектное решение до самого конца проекта. Разработка на ранних стадиях проекта служит своеобразной «системой раннего предупреждения», которая позволяет выявить неудачные решения и исправить их, пока затраты на исправление еще не стали неприемлемыми.

Допустим, группа проектирует распределенное приложение. Вы на бумаге оценили необходимую пропускную способность сети, и получившиеся требования к сетевой инфраструктуре выглядят вполне разумными. Тем не менее на стадии «Планирование» группа решила создать прототип для проверки работоспособности архитектуры (обратите внимание, что фаза «Разработка» еще не началась— задача, относящаяся к этой фазе, решается на стадии планирования). Испытания прототипа показали, что приложение нуждается в гораздо большей пропускной способности, чем предсказывали оценки. Более того, стало ясно, что, если разработать приложение в соответствии с исходным проектом, сеть организации просто не выдержит нагрузки, Группа пересматривает проект и предлагает другой подход. Новая версия прототипа оказывается гораздо более скромной в отношении сетевых ресурсов. Вряд ли стоит задавать вопрос, когда лучше обнаружить подобную проблему — на стадии планирования, при разработке или при развертывании готового приложения. Помните, что единственное средство «раннего обнаружения» проблем — разработка небольших, но жизненно важных составляющих проекта на каждой стадии.

Цели разработки

Мы уже отмечали, что группа может избежать проблем, присущих спиральной модели, выделяя на каждой стадии всего жизненного цикла проекта задачи, требующие внимания группы разработки. Задачи группы разработки и ее цели для каждой фазы различаются.

Цели каждого конкретного задания группе разработки должны вытекать из целей фазы в целом. Например, цели фазы «Анализ» — выработка единой концепции, согласие по основным бизнес-задачам, совокупность требований к проекту и т, д. — определяют цели любого задания на разработку на этой стадии. Задания, не отвечающие этому требованию, должны быть отложены или отменены.

Требование согласования целей разработки с целями текущей фазы вызвано необходимостью избежать преждевременной разработки. Обычная ошибка — интенсивная разработка приложения до окончательного оформления архитектуры. Как вы помните, архитектура приложения разрабатывается на стадии «Анализ» и окончательно оформляется на стадии «Планирование»; именно тогда начинается интенсивная разработка. Преждевременное начало этих работ чаще всего оборачивается бессмысленной тратой времени.

В большинстве проектов цели разработки попадают в одну из следующих категорий.

- **Прототип** — на стадии «Анализ» прототип служит удобным инструментом обмена мнениями и уточнения концепции. Прототип — это нефункциональная демонстрационная версия продукта, чаще всего лишь набор снимков с экрана или Web-страниц. Помните, что прототип лишь помогает проектной группе и заказчику достичь соглашения относительно концепции приложения и его пользовательского интерфейса.
- **Доказательство корректности концепции** — в отличие от демонстрационного прототипа, концептуальные системы, служащие для доказательства концепции, являются полнофункциональными приложениями, цель которых — доказать реализуемость концепции. Прототип нацелен на демонстрацию концепции и пользовательского интерфейса, а концепт-системы — архитектуры и технологии. Они отвечают на вопрос: «Можно ли создать стабильно работающее приложение в рамках предложенного проекта?» Концепт-система обычно создается на стадии «Планирование».
- **Альфа-, бета- и промежуточные выпуски** — на стадии разработки проектная группа чаше всего создает две основных промежуточных версии — альфа и бета, Они демонстрируют постепенное слияние пользовательского интерфейса, продемонстрированного на

прототипе, и технологий, проверенных концепт-системой. В больших проектах таких выпусков может быть несколько; мы рекомендуем выпускать по крайней мере альфа- и бета-версию во всех проектах, кроме самых незначительных. Альфа-версия отвечает на вопрос: «Вот так технология и интерфейс выглядят вместе. Есть ли какие-то серьезные проблемы?», а бета-версия — «Мы решили все основные проблемы и большинство мелких. Мы ничего не упустили?». Эта фаза разработки завершается созданием «золотой» версии.

- **«Золотая» версия** — на стадии «Стабилизация» именно эта версия передается ограниченному кругу пользователей для тестирования. По мере выявления и устранения ошибок и проблем, связанных с производительностью, следом за ней развертываются следующие выпуски. Окончательная «золотая» версия, которая появляется в конце стадии «Стабилизация», сигнализирует о завершении проекта и передаче результатов заказчику.

Некоторые разработчики игнорируют прототипы и концепт-системы, считая это пустой тратой времени. На деле справедливой оказывается обратная точка зрения. Даже в небольших проектах прототип и концепт-система способны сэкономить разработчикам немало времени и усилий за счет уточнения требований и проверки реалистичности архитектуры. Лучше потратить немного времени на первую версию и потом выбросить ее в мусорное ведро, чем отправить туда последнюю, на которую затрачены колоссальные усилия.

Роли членов группы в модели процесса разработки

Хотя за общий ход проекта отвечает менеджер программы, в достижении каждого этапа, как показано в таблице 4.2, участвуют все роли. Сопоставляя каждую из ролей (или групп ролей) с основными этапами, мы получаем ясное представление о том, кто отвечает за каждый этап. Такой подход позволяет четко распределить обязанности в группе.

Табл. 4.2. Ответственность ролей модели проектной труппы за основные этапы

Фаза	Этап	Ответственное лицо
Анализ	Одобрение концепции	Менеджер продукта
Планирование	Одобрение плана проекта	Менеджер программы
Разработка	Завершение разработки	Разработчик, инструктор
Стабилизация	Выпуск продукта	Тестер, логистик

На каждом из основных этапов ответственность за следующий этап передается соответствующей роли или группе ролей. Принима-

юшая сторона должна подтвердить передачу ответственности, чтобы каждый член группы осознал переход проекта на следующую стадию*. Организованный переход ответственности от роли к роли — признак здорового проекта.

Артефакты и результаты

В течение проекта группа проходит от этапа к этапу, добиваясь запланированных результатов. Мы делим эти результаты на две категории: *артефакты* (любые документы, имеющие отношение к проекту) и собственно *результаты,* необходимые для достижения основных и промежуточных этапов.

Часть проектной информации никогда не выходит за рамки проектной группы. Ни один из артефактов не попадает к заказчику или к пользователям; ясно, например, что повестка заседания не относится к результатам. Тем не менее все они важны. По окончании проекта все эти документы следует сохранить в архиве проекта в качестве проектной документации.

Результаты, напротив, создаются для передачи заказчику или другим внешним участникам проекта. В качестве результата может выступать документ, исходный код приложения или само приложение. Все они представляют ценность и должны быть сохранены и по окончании проекта.

Обмен документами и результатами должен быть эффективным. Далеко не все документы имеет смысл печатать — современные методы передачи информации, такие как электронная почта или Web-страница, чаще всего гораздо эффективнее. Если прототип носит исключительно демонстрационный характер, стоит организовать его в виде совокупности Web-страниц, сделав тем самым доступным всем заинтересованным сторонам.

Все участники проекта должны помнить, что их цель — создать законченное полнофункциональное приложение в установленные сроки, а не произвести на свет гору бумажной или иной документации. Эффективный обмен информацией — область, где изобретательность и инициативность должны всячески приветствоваться.

Приведем пример. Пусть, скажем, всем участникам онлайновой встречи, которые находятся в пяти разных странах, заранее отправлена презентация в формате PowerPoint, которую в каждом офисе показывают на проекторе, Начинается встреча, которая демонстрируется на другом проекторе. Кроме того, всем участникам встречи доступен прототип приложения, оформленный в виде совокупности

* И чтобы ответственность при передаче не потерялась. — *Прим. ред.*

Web-страниц. По ходу встречи участники высказывают свои пожелания, а группа разработки фиксирует их. Затем разработчики демонстрируют все предложения и замечания (уже реализованные в прототипе) участникам встречи, и те решают, что принять, а что — нет. Если следовать этому сценарию, который казался совершенно фантастическим еше несколько лет назад, слова «быстрая разработка приложений» приобретают совершенно новый смысл.

Конечно, не всякая группа готова к столь активному использованию суперсовременных технологий и не всякому проекту это нужно, однако каждая группа должна уделять максимум внимания эффективности обмена информацией. Эффективность — залог быстрой реализации в проекте главной цели, выпуска стабильного полнофункционального продукта в срок.

Связь моделей

Большая часть этой книги посвящена различным моделям. Напомним, что модель — это абстракция, которая служит для упрощения или представления какой-либо концепции или процесса. Важно помнить, что большинство моделей, описанных в этой книге, связаны между собой, Завершая первую часть книги, мы считаем необходимым продемонстрировать связь между моделями на трех уровнях абстракции.

Модель производственной архитектуры, которую мы обсуждали в первой главе, существует на высшем уровне абстракции. Другими словами, это модель верхнего уровня. Она непосредственно связана со стратегическим планом организации и фактически вызвана к жизни именно им, В свою очередь эта модель является основой для разработки моделей приложений масштаба предприятия.

Информация, собранная в модели или документе верхнего уровня, должна отражаться в моделях следующих уровней. Например, планы и процессы, входящие в стратегический план организации, являются основой бизнес-перспективы модели приложения масштаба предприятия. С другой стороны, бизнес-перспектива модели приложения образует бизнес-модель конкретного приложения. Другими словами, архитектура каждого конкретного приложения должна соответствовать производственной архитектуре организации.

В этой главе мы обсудили модель процесса разработки, которая описывает процесс разработки приложения. Составная часть этой модели — разработка проектной модели для концептуальной, логической и физической архитектур приложения.

Хотя этот процесс не является последовательным, общие рекомендации модели процесса разработки помогают проектной группе пройти этот путь.

Помните, что четкая формализация информационных потоков между разными моделями не всегда возможна. Нужно также помнить о различии одноименных терминов, относящихся к разным моделям. Таблица 4.3 поможет вам не забыть «основной» смысл этих терминов.

Табл. 4.3. Связанные термины разных моделей

Модель производственной архитектуры MSF	Модель приложения масштаба предприятия	Модель проектирования
Бизнес-перспектива	Бизнес-модель	Концептуальный проект
Прикладная перспектива		
Информационная перспектива		
Технологическая перспектива	Технологическая модель Модель пользователя Логическая модель Физическая модель Модель разработки	Логический проект Физический проект

Резюме

Разработка программного обеспечения — непростая работа. Как пишет Джим Маккарти, *«закончить несложную программу вовремя непросто. Создать сложное приложение — колоссальное достижение. Выпустить сложное приложение, уложившись в график — невероятная редкость»*.

Разрабатывать важное и нужное программное обеспечение становится все сложнее из-за лавинообразной популярности Интернета и всеобщего распространения масштабных многоуровневых приложений. В этой главе мы обсудили модели разработки программного обеспечения — модель водопада, спиральную модель и универсальный процесс, уделив особое внимание модели процесса разработки MSF.

Модель процесса разработки MSF обеспечивает строгость и гибкость — два качества, обязательных для успешной разработки таких приложений — за счет поэтапного планирования и итерационного подхода. В этой главе мы рассмотрели четыре фазы этой модели: концептуальное проектирование, планирование, разработку и стабилизацию. Мы обсудили концепцию последовательного выпуска версий продукта и подчеркнули важность итерационного похода к выполнению всех четырех фаз процесса разработки. Кроме того, мы рассмот-

рели документы этой стадии и проанализировали концепцию «ранняя определенность, поздняя фиксация». В завершение мы обсудили выбор целей для каждой итерации.

Закрепление материала

1. Охарактеризуйте модель водопада и спиральную модель.
2. Перечислите основные этапы универсального процесса.
3. Перечислите фазы и промежуточные этапы универсального процесса.
4. Опишите цели и задачи каждой из фаз модели процесса разработки MSF.
5. Перечислите результаты каждой из фаз модели процесса разработки MSF.
6. Опишите преимущества последовательного выпуска версий.
7. Опишите концепцию компромиссного треугольника и методы достижения компромисса.

Практикум 3. Знакомство с проектом RMS

— Доброе утро!

Четверг, 8 часов утра. Со времени первого совещания прошло всего три дня, но настроение членов группы заметно изменилось. Все они уже сидели на своих местах, открыв папки и приготовив ручки. Они с нетерпением ждали начала работы над проектом, жаждали применить свои новые знания. Все, кроме Тима О'Брайана.

- Так, где Тим? — спросил Дэн, с нотками раздражения и удивления в голосе. — Надеюсь, он не проспал.

- Нет, я не проспал! — послышался голос из коридора. — Я вообще не спал.

- Тим, ты выглядишь ужасно! — воскликнула Джейн Клэйтон, когда Тим вошел в комнату, — Что случилось?

— Ну, — сказал Тим, опустившись в кресло рядом с Дэном, — знаешь новый сервер на 24 этаже, тот, что для хранения CAD-чертежей? Его купили пару дней назад, и я попросил одного своего сотрудника настроить его, подключить к сети, назначить права — в общем, подготовить его к работе. Он обещали закончить к сегодняшнему дню. Вчера часов в десять вечера он позвонил мне и сказал, что работает уже два дня, но никак не может подключить сервер к сети. Я пытался помочь ему по телефону, но, кажется, он меня не понял. Поэтому я в 11 вечера сел в машину и приехал сюда. Только полчаса назад нам удалось подключить его.

- Спасибо, что выручил нас, — искренне поблагодарил Тима Дэн. — Я знаю, что проектировщики волновались по поводу этого

сервера, но особенно важно то, что ты делаешь все возможное, чтобы нести информационные технологии в массы.

Все зааплодировали, вызвав у покрасневшего Тима улыбку.

- Прежде чем мы перейдем к делу, — продолжил Дэн, — я бы хотел представить вам нового человека. Большинство из вас его уже знают — это Джим Стюарт, наш финансовый директор. Я попросил его прийти на нашу встречу. Добро пожаловать, Джим.

- Рад здесь присутствовать. Дэн, — ответил Джим, в голосе которого Дэн услышал осторожные нотки. — Я с радостью выслушаю информацию, собранную всеми вами.

— Да, информации сегодня будет много, — Дэн заглянул в повестку и обратился ко всей группе: — Как всегда, начнем с дополнений к повестке дня. У кого-нибудь есть поправки или дополнения?

- Я думала, мы обсудим материалы, которые ты дал нам в прошлый раз, — сказала Мэри-Лу, — но такого пункта в повестке дня нет.

- Ты права, Мэри-Лу, такого пункта нет. Но прежде чем я отвечу тебе, я хочу спросить; кто-нибудь еще хочет изменить повестку? — никто не ответил, поэтому Дэн продолжил: — Как уже заметила Мэри-Лу, я дал вам материалы по модели проектной группы и модели процесса разработки MSF и попросил вас их изучить. Однако в прошлый раз вы задавали такие вопросы, что я решил провести дополнительное совещание по текущей ситуации.

Текущее состояние

- Как многие из вас уже знают, этот проект связан с созданием приложения под названием «Система управления ресурсами» (Resource Management System, RMS). Оно будет отвечать за распределение ресурсов, фиксировать рабочее время и оформлять ведомости и счета. Я попросил некоторых из вас подготовить информацию о том, как это делается сейчас. Понятно, что двух дней на такую работу мало, поэтому задание было дано тем, кто хорошо знаком с этим делом.

— Хорошо, что я не занял тебя, — обратился Дэн к Тиму. — особенно учитывая твои ночные бдения,

— Я просто счастлив, что сегодня я только зритель, — ответил Тим.

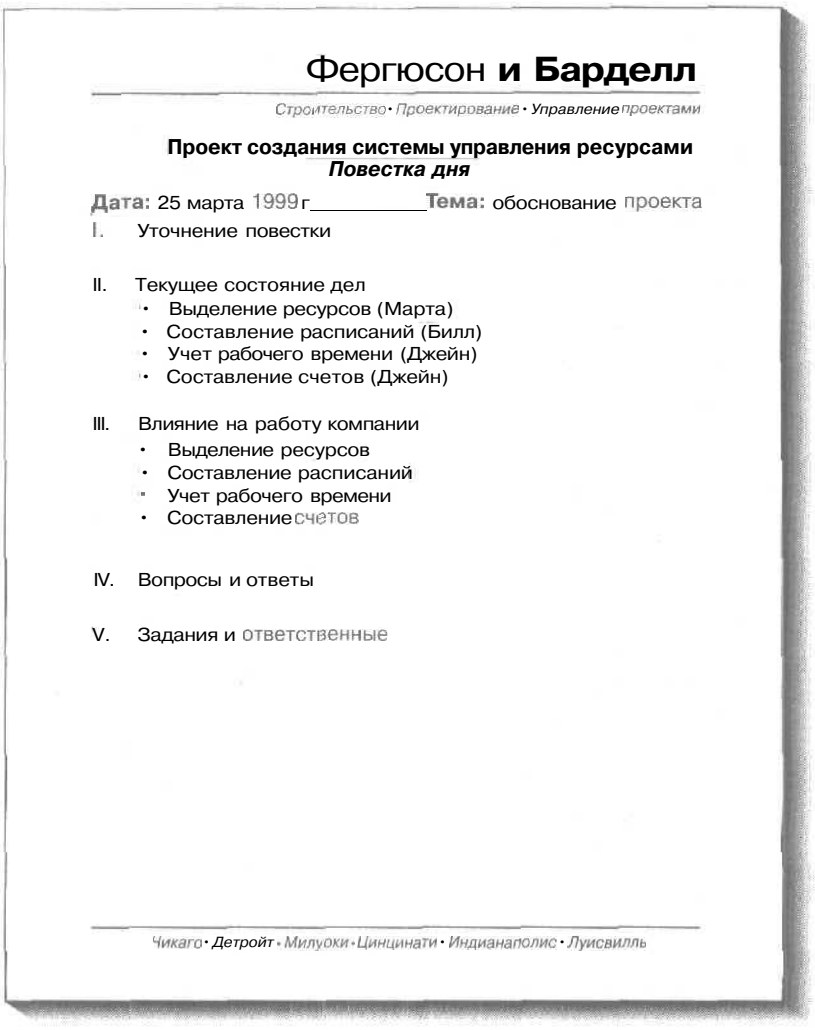
Распределение ресурсов

Дэн повернулся к Марте Вольф-Хелен, сидящей в конце стола и сказал:

- Марта, несмотря на то, что ты недавно начала работать в «Фергюсон и Барделл», ты уже имеешь опыт распределения ресурсов. Почему бы тебе не начать?

— В первую неделю моей работы в «Фергюсон и Барделл», — заговорила Марта, заглянув в записи в своей папке, — меня вызвал к себе

заместитель директора по строительству. Мы говорили о многом, в основном о том, что и как я буду делать в этой компании. При этом он показал мне главное расписание.



Фергюсон **и Барделл**

Строительство • Проектирование • Управление проектами

Проект создания системы управления ресурсами
Повестка дня

Дата: 25 марта 1999 г **Тема:** обоснование проекта

I. Уточнение повестки

II. Текущее состояние дел
- Выделение ресурсов (Марта)
- Составление расписаний (Билл)
- Учет рабочего времени (Джейн)
- Составление счетов (Джейн)

III. Влияние на работу компании
- Выделение ресурсов
- Составление расписаний
- Учет рабочего времени
- Составление счетов

IV. Вопросы и ответы

V. Задания и ответственные

Чикаго • Детройт • Милуоки • Цинцинати • Индианаполис • Луисвилль

Им оказалась простая таблица Excel, верхнюю строку которой занимали ячейки с датами, а левый столбец — имена его подчиненных: инженеров, конструкторов, секретарей и административных служащих. Отсортированы они были по алфавиту, сначала по фамилии, потом по имени.

Директор объяснил мне, что, когда ему нужны свободные сотрудники, он открывает эту таблицу и отыскивает в ней человека, не занятого другой работой. Если такой находится, он помечает его, записывая в ячейку с соответствующей датой название проекта. В противном случае ему приходится либо перераспределять ресурсы, либо договариваться с заказчиком об изменении срока работ.

— Как видите, система несложная. Она применяется во многих мелких фирмах, — прокомментировал слова Марты Дэн. — Однако «Фергюсон и Барделл» — не мелкая фирма, и это наводит меня на мысль, что у заместителя директора часто возникают проблемы. Я прав, Марта?

— Он мне рассказал, — улыбнулась Марта, — что говорил тебе об этом на прошлой неделе во время гольфа и что не будет с тобой играть в гольф до тех пор, пока ты все не исправишь. Так что не притворяйся, что ничего не знаешь.

За столом послышались смешки, а Марта продолжила:

— Я составила список проблем, которые мы с ним обсуждали. Записать их на доске?

- Отличная идея, — ответил Дэн. — Назовем список «Распределение ресурсов», а по ходу совещания дополним его.

Марта подошла к доске, написала сверху «Список проблем» и чуть ниже — «Распределение ресурсов».

Проблемы

— Во-первых, — начала Марта, — в электронной таблице отсутствует информация о квалификации сотрудников.

Написав это на доске, она продолжила:

- Из-за того, что таблица мала, нельзя имя каждого сотрудника сопроводить описанием его квалификации — эта информация переменной длины. А значит, нельзя сортировать или искать сотрудников с необходимой квалификацией. Об этой проблеме я знаю, потому что так уже случилось и со мной.

- Как «так»? — спросил Тим,

— В институте я специализировалась на программируемых логических контроллерах. К сожалению, директор об этом не знал и не назначал меня на такие заказы. А однажды мы потеряли заказ из-за того, что, как он считал, у него нет людей с необходимой квалификацией.

Джим нервно заерзал на стуле, но ничего не сказал.

- Вторая проблема, — продолжила Марта, добавляя ее к списку на доске, — состоит в том, что расписание отделено от календарной системы, в качестве которой в «Фергюсон и Барделл» применяется Outlook и Exchange. Кроме того, многие служащие просто не считают нужным постоянно обновлять свои календари в Outlook. В результате менеджеры не могут искать доступные ресурсы по календарю. По-

этому, когда они подключают людей к работе, они посылают им сообщение по электронной почте или звонят. Часто в ответ менеджеры получают сообщение, что человек уже занят. Так как не существует связи между таблицей ресурсов и календарями, выясняется, что человек находится в отпуске или уже работает в другом проекте. Тогда менеджеру приходится начинать поиск сначала.

И последнее, такая система совершенно неуправляема. Она отлично работает, если менеджер отвечает, скажем, за 30 человек. Однако в «Фергюсон и Барделл» у менеджера в подчинении часто бывает 100 человек или даже больше. Они просто не в состоянии отслеживать квалификацию своих подопечных, а тем более их занятость. Попросту говоря, мы переросли такой метод распределения ресурсов.

Марта села на свое место, наблюдая за реакцией Дэна.

— Отличная работа, Марта, — сказал он. — Коротко и ясно.

Табели

— Чиф, — сказал Дэн, обращаясь к Биллу Парди, — я просил тебя рассказать о табелях по двум причинам: ты имеешь с ними дело уже много лет и, насколько мне известно, проблем с ними у тебя хватает. Так что ты скажешь?

— Надеюсь, мне вставать не надо, — проворчал Билл. — Я так устал, заполняя их.

Остальные члены группы согласно кивнули, после чего Билл начал:

— Заполнение табелей — в основном не автоматизированный процесс. Мы готовим их на основе шаблона Word. В его верхней части указываются имя служащего, его идентификационный номер, отдел, должность, статус и руководитель. Ниже помещена таблица, в которой записаны дата, номер задания, категория, фаза, полное время, оплачиваемое время, затраты на переезды, питание и жилье, а также описание задания.

С этими словами он раздал образец табеля членам группы, хотя многие из них уже были знакомы с этим документом.

— Стоит обратить внимание на некоторые детали, — продолжил он. — Служащие заполняют табель строчка за строчкой по мере выполнения работы. В каждой такой строке надо указать идентификатор задания, который обозначает проект и назначается финансовым отделом.

— А откуда они узнают этот номер? — поинтересовалась Мэри-Лу. Она работала по контракту и поэтому ничего не знала о системе заполнения табелей в «Фергюсон и Барделл».

— Люди узнают его при получении задания, — ответил Билл. — К сожалению, иногда им сообщают неправильный идентификатор, а

иногда и вообще не сообщают, так как проект — новый. Это и есть одна из проблем, связанных с заполнением табелей.

— К проблемам этого процесса мы перейдем позже, — прервал его Дэн. — Сейчас же пусть Билл закончит описание существующей в компании системы табелей. Для чего нужны колонки «Категория» и «Фаза»? — добавил он, обернувшись к Биллу.

— Они используются не всегда — только в крупных проектах. Фаза обозначает часть проекта, над которой ведется работа, а категория определяет вид деятельности, — ответил Билл, после чего взглянул на Джейн и добавил: — Для маленьких заданий колонка «Фаза» не заполняется. А если тип работы очевиден, мы и колонку «категория» оставляем пустой, предполагая, что в финансовом отделе ее заполнят. Правильно, Джейн?

— Все абсолютно точно, Билл, — ответила Джейн. — А ты помнишь старую поговорку о *предположениях*?

После того, как смех утих, Джейн продолжила:

— Особенно же мне нравится, что люди из твоего отдела во всех ячейках колонки «Категория» пишут «Разработка». Что вы разрабатываете за обедом?

Прежде чем Билл смог ответить, заговорил Тим:

— Все очень просто! Обедая там, где и Билл, ты *зарабатываешь* изжогу!

В этот раз засмеялись все, даже Джим. Билл же проворчал:

— Ничем не могу помочь, если у тебя такой нежный желудок.

Несмотря на тон высказывания, всем было ясно, что Билл пошутил.

Хотя работы оставалось еще много, Дэн не торопился вернуться к делам. Ведь именно в такие моменты сплачивается коллектив, и ему было радостно это видеть. Дэн подождал, пока смех уляжется, и сказал:

— Возможно, нам стоит пойти пообедать с Биллом и все проверить самим, — затем он повернулся к Биллу и добавил: — Я вижу, что в табеле есть колонки «Полное время» и «Оплачиваемое время». Они зачем?

— Дело в том, что не вся работа оплачивается, — ответил Билл, — Например, при обучении. Кроме того, иногда человеку кажется, что работа займет больше времени, чем следует. В таком случае он указывает его, как неоплачиваемое. Мы стараемся не допускать этого, но иногда разрешаем.

Следующие колонки описывают затраты на переезды, питание и жилье. Естественно, все служащие должны представить соответствующие квитанции.

И наконец, последняя, самая важная, колонка «Описание», информация из которой указывается в счете заказчика.

— А что если возникнут траты, не учтенные в этом документе, например, покупка книги? — прервала его Мэри-Лу.

— Это зависит от обстоятельств, — ответил Билл. — Если такие расходы оплачиваются заказчиком вместе со всем проектом, они включаются в одну из этих трех колонок, а в колонке «Описание», объясняется их назначение. В противном случае они считаются внутренними расходами и обрабатываются внутренней системой отслеживания расходов.

Мэри-Лу кивнула, и Билл продолжил:

— Закончив заполнение таблицы, служащий отправляет ее по электронной почте в центральный офис. Мы создали для этих целей почтовый ящик Exchange с псевдонимом Time, с помощью которого вы отправляете заполненный табель в офис средствами Outlook или через Интернет.

— После чего все это становится проблемой Джейн, — добавил Дэн. — Хорошо, Билл, перечисли проблемы этой системы. А так как ты очень устал от заполнения табелей, на доске писать буду я.

Проблемы

— Я, наверное, получу приз за наибольшее количество проблем, — сказал Билл. — Основная проблема заключается в возможности ошибок, так как люди сами заполняют табель. Данные, введенные в таблицу Word, не проверяются, поэтому можно ошибиться при вводе или расчете времени, особенно если работа так раздроблена, что для ее описания нужны две или три страницы. Кроме того, мы постоянно получаем неправильные номера проектов, а иногда не получаем их вообще. В этом случае оплачивается время другого проекта, а если мы не заметим такой ошибки, заказчик получит счет, не соответствующий действительности.

Вторая проблема заключается в том, что табель статичен. Другими словами, по мере ввода данных сотруднику не сообщается, все ли он правильно ввел. Конечно, люди могут все проверить вручную, но часто их поджимают сроки, они спешат заполнить и отправить расписание, и поэтому они этого не делают. В результате они пропускают целый день или, наоборот, вводят что-нибудь дважды. Если бы табель обновлялся после ввода каждого значения, ошибок бы стало гораздо меньше.

Третья проблема, о которой уже говорила Марта, заключается в разделении табелей и календарей. Некоторые применяют календарь Outlook, некоторые — другие программы или даже карманные компьютеры, однако независимо от применяемого метода, всем приходится открывать календарь, изучать его данные и только потом заполнять табель. Быстрого способа сделать все это нет; нельзя даже перенести данные календаря в табель для редактирования.

– К тому же эта проблема приводит нас еще к одной, — продолжал Билл, наблюдая за Дэном, записывающим его слова на доске. — Возможно, вы и не знаете, но в нашей системе поощряется дублирование информации. Однако один из главных принципов управления данными гласит: сохранять информацию только в одном экземпляре и, предпочтительно, централизовано, в легко доступном месте. Сейчас же происходит следующее — данные пользователей с различных устройств преобразуются в табели, которые набираются на другом компьютере. Мы обязательно должны сократить число мест хранения данных.

– Мы полностью с тобой согласны, — сказал Дэн, записав все на доске. — Нам не удастся решить эти проблемы в первых версиях RMS, но мы можем поставить себя такую цель,

Затем он снова повернулся к доске и спросил:

– Что еще?

Билл посмотрел в свои записи и обратился к группе:

– Как видите, такую систему почти невозможно использовать. Именно поэтому большинство наших служащих, ненавидящих такую утомительную работу, заполняют свои табели в последнюю минуту. В результате табели опаздывают, что задерживает весь бухгалтерский процесс.

Затем он повернулся к Джиму и добавил:

— Я полагаю, что из-за запаздывания табелей возникают и другие проблемы. Правильно?

Джим энергично кивнул и хотел что-то сказать. Однако Дэн опередил его:

— Джим, как ты смотришь на то, чтобы высказаться позже, когда мы перейдем к третьему пункту повестки дня? Я предоставлю тебе достаточно времени, ведь, как я думаю, то, что ты скажешь, будет для нас очень ценно, а я не хочу снижать всю важность твоего выступления, разделяя его на части.

Джим немного подумал и ответил:

— Отличный план, Дэн. Я пока сделаю несколько заметок, а обсудим их мы потом.

Учет

Теперь Дэн обратился к Джейн:

– Ну вот, Джейн, мы распределили ресурсы, составили табель и отправили его по почте. Что дальше?

Джейн открыла свою толстую светло-коричневую папку, вынула из нее рисунки и раздала их всем присутствующим.

— Я подумала, что мой рассказ станет понятнее, если его изобразить в виде схемы, — под одобрительный шепот членов группы сказала она.

- Как видите, начало работы — в верхнем левом углу схемы: когда мы получаем табель из почтового ящика Time. Всего ее делают восемь человек. Причем их задания постоянно меняются, ведь если бы они работали с табелями весь день, то давно бы уже уволились,

Но независимо оттого, кто выполняет работу, процедура не изменяется. Мы открываем каждое сообщение и сохраняем прикрепленный к нему табель в папке, названной по фамилии служащего. Файл называется датой его получения. После этого мы открываем табель и распечатываем его.

Имея на руках бумажную копию табеля, мы вводим данные из него в наш бухгалтерский пакет, Затем мы распечатываем всю информацию за неделю для каждой группы и пересылаем ее по факсу или вручаем лично соответствующему менеджеру для утверждения. Он изучает табели всех сотрудников и утверждает — или не утверждает — их. Затем он пересылает подписанные документы со своими пометками обратно нам.

Воспользовавшись паузой, Марта опросила:

— Сколько табелей обычно не утверждается?

— Около пяти процентов, — ответила Джейн. — Обычно это происходит, если человек не ввел какие-либо данные или ошибся в расчетах.

Марта кивнула, и, видя, что больше вопросов нет, Джейн продолжила:

— Если табель не утвержден, мы отсылаем его автору с пометками менеджера и с просьбой внести исправления. Кроме того, мы удаляем старый файл, чтобы у нас не было двух табелей для одного и того же времени. В заключение все утвержденные документы пересылаются в систему учета рабочего времени и оплаты труда,

— Давай на этом остановимся, Джейн, и перечислим все проблемы этого этапа, — сказал Дэн, подходя к доске. — Какие проблемы возникают при регистрации табелей?

Проблемы

- Проблем две, но они вызваны третьей. С какой мне начать? — спросила Джейн.

- Начни с главной, а затем перейди к ее следствиям, — ответил Дэн. — Какая из них главная?

— Главное — то, что эта огромная, утомительная работа проводится вручную. Она выполняется в несколько этапов, большинство которых состоит из ввода данных и печати документов. В этом процессе занято много людей, а компьютеров у меня мало.

Дэн написал на доске «Работа выполняется вручную».

— Думаю, все согласятся, что обработка 800 табелей — процесс продолжительный и утомительный, — сказал он и нарисовал стрелку, направленную вниз от надписи. — А какие две проблемы вызваны этим?

- Первая — большое количество ошибок. Вторая — продолжительность процесса.

- Объясни, пожалуйста, поподробнее, — попросил Дэн, записывая ее слова на доске.

- Иногда нам приходится вводить более 10 000 записей в неделю, Естественно, что даже лучшие сотрудники допускают ошибки. Большинство мы находим при повторном изучении документов, но некоторые ошибки все равно остаются и попадают к заказчику. Я очень долго пыталась снизить количество ошибок и неплохо в этом преуспела, но все-таки мы люди, а не автоматы.

Кроме того, этот процесс отнимает у меня почти все ресурсы. Я бы могла выполнять гораздо больше работы, если бы не эти проклятые табели. Причем они никогда не кончаются. Каждый понедельник ты идешь на работу, думая о том, что целый день провозишься с вводом данных. Это ужасно! — поняв, что слишком разволновалась, Джейн сделала паузу, чтобы перевести дыхание, и добавила: — Вот все три проблемы.

— А мне кажется, что есть и четвертая, — вставила Марта.

- И пятая, — добавила Мэри-Лу.

- Ну что ж, выслушаем вас, — сказал Дэн, снова поворачиваясь к доске. — Начинай, Марта.

— Джейн сказала, что она постоянно меняет сотрудников, занимающихся расписаниям, так как в противном случае они давно бы уволились. Кроме того, она упомянула, что ее люди очень не любят эту работу. Поэтому мне показалось, что существует еще одна проблема — воздействие на служащих, их дух и отношение к компании,

— Знаешь, — сказала Джейн, немного помедлив, — я не думала над этим вопросом, так как сконцентрировалась на технической стороне дела. Но, конечно, ты права, самый большой подарок, который я могу преподнести своим сотрудникам — это освободить их от обработки табелей.

Затем она повернулась к Дэну и спросила:

— А мы должны учитывать такие проблемы, Дэн?

- Безусловно, — ответил Дэн. — Иногда их трудно оценить, мы увидим это позднее, когда будем изучать возврат инвестиций. Но Марта попала в точку. Ведь очень важно повысить привлекательность «Фергюсон и Барделл» для служащих, особенно если учитывать за-

траты на их найм и обучение, — он повернулся к доске и добавил: — Так и запишем: «Влияние на моральное состояние служащих и их отношение к компании». Теперь выслушаем Мэри-Лу.

— Я заметила проблему, о которой уже говорил Билл, — дублирование данных, — сказала Мэри-Лу. — По моим подсчетам, расписания хранятся сразу в трех местах: на компьютере служащего, в главном каталоге табелей и в системе учета времени и оплаты труда. — Обращаясь к Биллу, она добавила: — При этом сложно гарантировать целостность данных, а о путанице разных версий я даже и не говорю. Это так?

— Еще бы! — ответил Билл, внезапно поняв, что соглашается с человеком, которого совсем недавно называл «инструктором по нажиманию кнопок». Он взглянул на улыбающегося Дэна, уже говорившего ему о квалификации Мэри-Лу, и добавил: — Если бы ты проектировала этот процесс заново, с чего бы ты начала?

— Я бы начала с создания централизованного хранилища данных, — решительно ответила Мэри-Лу, — сократила бы количество мест, в которых анализируется и хранится информация. Возможно, все сразу бы и не получилось, но, если хорошенько подумать, можно разработать почти идеальную систему.

Билл согласно кивнул.

— Ты мыслишь в правильном направлении, Мэри-Лу, — вставил Дэн, — но это — тема следующих совещаний, поэтому мы рассмотрим твое предложение позже.

Выписка счетов

— На твоей схеме изображено больше, чем ты нам рассказала, — обратился Дэн к Джейн. — Поведай нам о последней части.

— Все довольно просто, — ответила Джейн, заглядывая в рисунок. — Как видите, после пересылки табелей в систему учета времени и оплаты труда мы выписываем предварительный счет, и передаем его в отделы менеджмента и продаж. Если они этот счет утверждают, мы переводим его в главную систему учета, которая выписывает окончательные счета.

— В этой части есть какие-нибудь проблемы? — спросил Дэн, приготовившись записывать.

Проблемы

— Я долго размышляла, и пришла к выводу, что единственная проблема, возникающая на этом этапе, связана не столько с процессом, сколько со структурой «Фергюсон и Барделл».

— И в чем она заключается?

— Дело в том, — ответила Джейн, — что у нас два способа оплаты: почасовая и за проект. С почасовой оплатой все в порядке. Однако

работа над проектом оплачивается независимо от времени, потраченного на нее. Изучая данные, я пришла к выводу, что мы теряем деньги. Это происходит из-за способа учета отработанного времени — ведь мы часто не знаем, что вышли из графика, пока не выпишем окончательный счет.

– То есть проблема заключается в том, что мы не можем определить время, затраченное на проект, и сравнить действительно затраченное время с нашими обязательствами? — спросил Тим.

– Точно, — ответила Джейн. — Это происходит не часто, так как наши менеджеры и представители отдела продаж тесно сотрудничают и дают точные оценки затрат времени на крупные проекты. Однако в мелких проектах, занимающих всего несколько недель, такое случается. И к тому моменту, когда мы понимаем, что проект вышел из графика, уже ничего нельзя предпринять.

– Хорошо, — сказал Дэн, — занесем эту проблему в наш список. Возможно, по мере работы над проектом мы что-нибудь и придумаем.

С этими словами он отошел от доски и направился к своему месту.

Проблемы бизнеса

– Как я уже сказал, — Дэн откашлялся, — я пригласил сюда Джима Стюарта, финансового директора, поделиться с нами мнением о системе учета времени и оплаты. Сегодня он уже сделал несколько заметок в своем блокноте и, я думаю, захочет поведать нам, что именно его заинтересовало. Джим, одна из причин, по которой я тебя пригласил, — то, что ты работаешь здесь дольше меня. Твое мнение очень пригодится нам, ведь мы хотим сделать действительно полезный продукт. Так что расскажи нам, как выглядят перечисленные нами проблемы с точки зрения бизнеса.

Дэн сел, все внимание теперь было обращено на Джима.

Когда Дэн пригласил его на совещание, Джим даже не знал, что и думать. Вначале его задевало, что Дэн не подчиняется ему. Он беспокоился, что новый директор по ИТ будет слишком много тратить, не заботясь о компании. Однако за шесть месяцев работы с ним он понял, что Дэна интересуют не только компьютеры, но и те же самые вопросы, что и самого Джима. Оба они хотят, чтобы фирма более эффективно работала с заказчиками, служащими и инвесторами.

Приглашение на совещание, касающееся в основном технических вопросов, было для него в новинку. Однако ему казалось, что Дэну и остальным членам группы действительно интересно его мнение. Поэтому, собравшись с мыслями, Джим начал:

— Я очень ценю возможность присоединиться к вам сегодня. Мне очень понравилось, как вы обсуждали наши общие проблемы, осо-

бенно выступление Джейн и ее самоконтроль. Мне приходилось слышать, как она описывала нашу систему учета времени и оплаты гораздо красочнее.

Джейн покраснела, остальные же понимающе засмеялись. Напряжение спало, и все, включая Джима, немного расслабились.

- По ходу нашей встречи я сделал несколько заметок, — продолжил Джим, — и теперь я хотел бы высказать свое мнение относительно основных проблем бизнеса. Я не собираюсь нарушать повестку дня, поэтому я просто перечислю все эти вопросы, охватывающие не одну область деятельности. Так подойдет?

Все согласно кивнули, и Джим направился к доске.

- На прибыль компании можно повлиять только двумя способами — либо увеличить доходы, либо снизить затраты. Для этого существует огромное число методов. Я намеренно упрощаю, но это хороший способ знакомства с проблемой. Итак, на что же влияет нынешняя система учета?

Мне кажется, что у нашей системы учета времени есть четыре проблемы. Две влияют на расходы, и еще две — на доходы.

Первая — неэффективное использование ресурсов. Отличный пример этому привела Марта: у нас есть сотрудники с определенной квалификацией, но мы о ней не знаем. В результате мы привлекаем малоквалифицированного специалиста, попросту не подозревая о существованиивысококвалифицированного,

Естественно, что такое неэффективное использование ресурсов приводит к снижению дохода. Заказчикам нужны компетентные исполнители, но не более того, поэтому для нас главное — выбрать того сотрудника, который бы гарантированно справился с решением задачи. Если мы пошлем работника с зарплатой в 200 долларов выполнять 50-долларовую работу, то мы либо выставим заказчику счет в 50 долларов и потеряем деньги, либо выставим счет в 200 долларов и потеряем заказчика.

Конечно, возможна и другая описанная Мартой ситуация, когда мы совсем теряем заказ из-за того, что не знаем о наличии у нас необходимых ресурсов. Это хуже всего! Однажды из-за такого инцидента мы потеряли отличного сотрудника торгового отдела. Она шесть месяцев уговаривала одну компанию дать нам заказ, а когда добилась своего, оказалось, что мы не готовы к этой работе, хотя, как в последствии выяснилось, могли бы ее выполнить.

Вот две проблемы, влияющих на доход фирмы. Теперь перейдем к проблемам, оказывающим влияние на расходы. Их тоже две. Первая — в затратах на заполнение и проверку табелей и последующую

подготовку, проверку и пересылку счетов. Хотя затраты времени на этот процесс неизбежны, но мы все же должны найти способы их сокращения. Если нам удастся освободить 15 минут в неделю, мы получим 10 часов в год. Умножьте это число на 800 служащих и получите 800 дополнительных человеко-часов в год.

Мне кажется, что особенно важно оптимизировать ввод данных. Джейн говорила, что если бы этот процесс был хотя бы частично автоматизирован, то у нее освободился бы один, а может быть и два сотрудника. Изучая расходы, связанные с одной такой должностью, можно прийти к выводу, что только ради этого стоит написать новое приложение.

Но даже если мы не сократим две должности, а просто высвободим людей для другой, более продуктивной работы, это значит, что по мере роста компании нам не придется нанимать новых сотрудников — нам удастся воспользоваться опытом людей, уже знакомых с «Фергюсон и Барделл». Так мы тоже увеличим эффективность труда.

Джим сделал паузу. Оглядывая комнату, он заметил, что многое из сказанного было новым для большинства группы. Казалось, что только Дэн был невозмутим, да и Джейн уже слышала нечто подобное раньше, остальные же никогда не сталкивались с влиянием технических проблем на бизнес.

- Есть и еще одна проблема, — продолжил Джим. — Она довольно специальная, однако с ней я сталкиваюсь каждый день, и она способна очень сильно влиять на наши доходы. Связана она со стоимостью средств.

Как и у всех компаний, у «Фергюсон и Барделл» есть финансовые потоки. Мы получаем деньги по счетам и тратим их на зарплату сотрудников и другие цели. Если мы получаем больше, чем тратим, у нас есть прибыль. Если же уходит больше, чем приходит — у нас убытки,

Если мы хотим быть прибыльной компанией, у нас всегда должен быть положительный баланс денежных потоков. Но большинство наших проектов выполняются для крупных фирм, и большинство счетов оплачиваются по достижении определенных стадий. Поэтому в какие-то моменты у нас бывает очень маленькая прибыль или даже убытки.

Важный фактор этого процесса — цикл обращения счетов. Джейн, как часто мы выставляем счета?

— Дважды в месяц, — ответила Джейн.

— А каковы сроки их оплаты?

- Мы выставляем их для оплаты в течение 30 дней, но большинство из них оплачиваются только через 45.

— А что мы делаем, чтобы расплатиться по нашим счетам — скажем, по зарплате, — если в данный месяц мы ждем оплаты счета и у нас не хватает денег?

– Мы берем кредит.

– Точно! — воскликнул Джим. — Мы берем кредит. Вот тут-то и возникает проблема стоимости средств.

Джим казался очень довольным тем, что ему удалось объяснить такое сложное понятие, как стоимость средств, группе технарей. Однако Дэн, поняв, что не все еще разобрались в этом вопросе, решил прикинуться, как будто бы и он тоже не совсем понял суть дела:

— Джим, если я правильно понял, нам приходится брать кредит, если оплата наших счетов задерживается. Но почему ты называешь это стоимостью средств и как она связана с нашей системой учета времени и оплаты?

Все с облегчением посмотрели на Дэна, который не постеснялся показать свое незнание. Джим улыбнулся, поняв ситуацию, и ответил:

— Дело в том, что при возвращении кредита мы должны выплачивать проценты. Они не высоки, но иногда составляют значительную сумму. Вот эти проценты и называются стоимостью средств.

Теперь о том, как она связана с системой учета отработанного времени и оплаты. Сейчас мы выставляем счета дважды в месяц из-за того, что нам требуется две недели для их обработки. Если бы этот срок был уменьшен, и подготовка счета занимала одну неделю, мы получали бы деньги на неделю раньше, отдавали кредит на неделю раньше и, соответственно, платили бы меньше по процентам. За год нам удалось бы сэкономить довольно много.

— Сколько? — потребовали ответа сразу все члены группы.

Джим был несколько ошеломлен, поэтому Дэн повторил вопрос уже спокойно:

– Сколько денег можно сэкономить за год, если перейти на недельный цикл выставления счетов?

Джим минуту подумал, проводя в своем блокноте вычисления и наконец сказал:

– Если я не ошибся в расчетах, такой переход сэкономит компании 500 000 долларов,

Дэн присвистнул, а Тим пробормотал:

– Ну и ну!

Джим же быстро встал ил:

– Это только предварительные цифры, основанные на текущих кредитах и на данных прошлого года. Они могут оказаться меньше, если мы будем брать кредиты реже.

– Однако они увеличатся, если правительство повысит учетную ставку? — спросила Джейн, в ответ на что Джим согласно кивнул.

Задания и ответственные за их выполнение

Дэн поднялся и сказал:

— Спасибо. Джим, ты открыл нам глаза. Я думаю, теперь нам всем ясно, почему мы рассматриваем возможность замены системы учета. — Затем он обратился к Марте, Биллу и Джейн: — Спасибо и вам за ваши обстоятельные описания процесса и его проблем. Теперь все отлично представляют систему учета. Вопросы есть? Тим?

– Ты только что сказал, что мы «рассматриваем» возможность замены системы учета, — ответил Тим, постукивая карандашом по столу. — Мне кажется, мы не должны ничего «рассматривать», мы должны уже *работать* над ней.

— Я понимаю, что многие так думают, но поймите, что мы только изучили ситуацию. Мы еще не знаем, каких расходов потребует создание новой системы, кроме того, мы не можем начать работу, пока не определим основные параметры системы. Сначала надо проанализировать решение, затем определить его стоимость. Только после этого мы сможем решить, стоит ли вообще что-нибудь создавать. В этом и заключается прелесть методического подхода. Если мы будем следовать этой модели, мы никогда не начнем работу над проектами, которые не стоит затевать. Мы просто не будем тратить ресурсы, не убедившись в необходимости этого. Еше вопросы есть? Нет? Тогда распределим задания.

– Я напечатаю записанные на доске проблемы и помещу их в каталог проекта. Вы найдете их в файле «Исходные проблемы» в каталоге «Анализ». Джейн, поработай, пожалуйста, с Джимом и определи точную стоимость средств. Если мы решим, что можем разработать решение, сокращающее цикл выставления счетов до одной недели, нам понадобятся точные данные. Джим, ты не против? У тебя есть время?

– Все зависит от Джейн. У тебя найдется время поработать со мной? — спросил Джим.

– Завтра у нас отсутствуют несколько сотрудников, поэтому будет трудно. А если сегодня днем?

– В два часа подойдет? — предложил Джим, и Джейн согласно кивнула.

– Хорошо, — сказал Дэн, раздавая всем членам группы новые материалы. — Здесь материалы для подготовки к следующему совещанию. Не забудьте перед ними прочитать предыдущие. В новых материалах описаны постановка целей и итерационное проектирование.

Они нам понадобятся уже в понедельник. Встретимся здесь в то же время. Мы уже почти готовы к созданию новой системы учета времени и оплаты.

Выходя из комнаты, Марта спросила Тима:

— Я уже слышала термин «итерация» раньше, но не совсем уверена, что он значит в технике.

— Самое лучшее определение я прочитал на стене компьютерного класса в школе, — ответил Тим. — Там было написано «Итерация, существительное — см. итерация».

Они рассмеялись и отправились по своим офисам.

Практикум 4. Определение целей

Войдя в дубовый зал, предназначенный для заседаний совета директоров «Фергюсон и Барделл», Тим О'Брайан огляделся и воскликнул:

— Посмотрите на картину на стене! Она больше, чем мой письменный стол!

— Если ты хочешь увидеть действительно большой стол, посмотри на стол заседаний, — сказала Джейн Клэйтон, появившаяся в комнате вслед за Тимом.

Зал постепенно заполнялся разработчиками RMS. И у всех, кроме Дэна Шелли и Билла Парди, уже хорошо знакомых с этим залом, он вызывал восхищение. Дэн работал в этом зале уже почти два дня, готовя материалы к совещанию группы в понедельник. Билл несколько лет назад помогал устанавливать здесь оборудование, поэтому уже не удивлялся картине на стене.

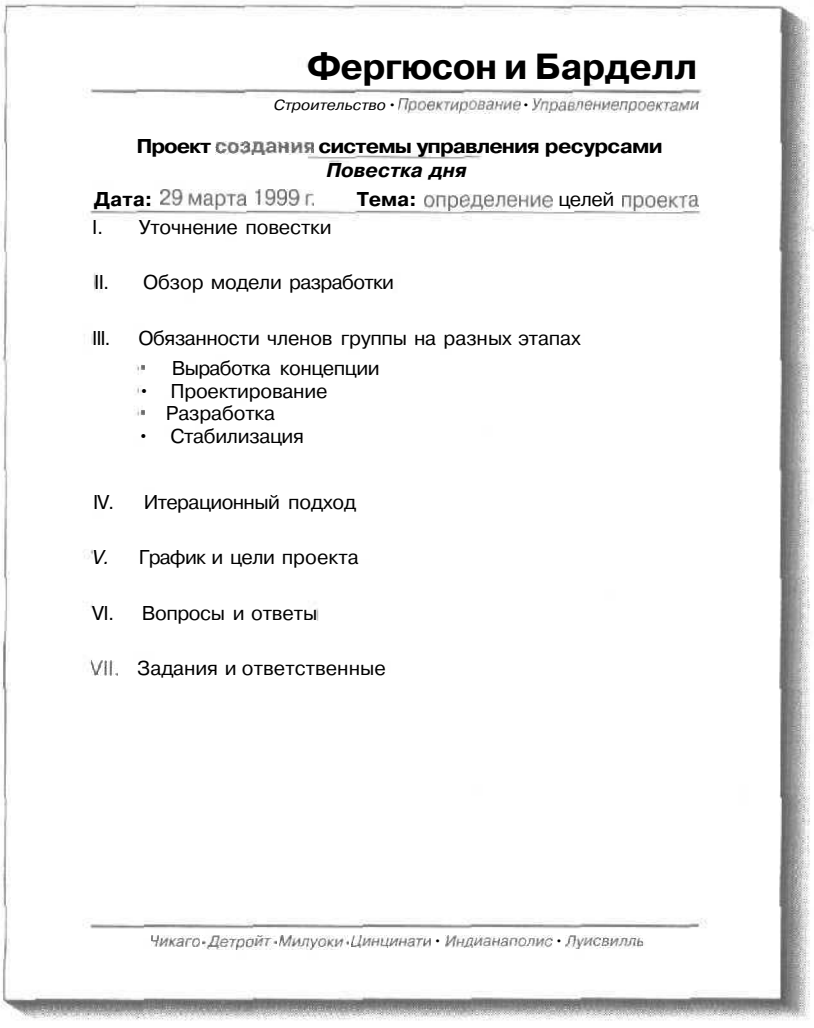
— Ладно, ребята, это всего лишь большой зал заседаний. Давайте сядем за круглый стол и начнем, — Дэн уже понимал, что время поджимает, и торопился начать. За выходные он пересмотрел график проекта, так как стало понятно, что времени не хватает. Теперь же он понял, что времени еще меньше, чем он и его группа могли себе представить.

— Почему ты вызвал нас, Дэн? — спросила Мэри-Лу Моррис, занимая кресло рядом с Джейн.

— Послушай, а у тебя есть план, как нам надо рассаживаться, или какие-нибудь предпочтения? — добавила Марта Вольф-Хеллен, размышляя о том, сесть ли рядом с Биллом, который, казалось, не любил ее, или с Мэри-Лу, которая обычно разговаривала исключительно с Джейн.

— У меня есть план, но только для одного из вас. Впрочем, об этом потом. Сейчас просто сядь где-нибудь, — он занял место во главе стола и пригласил остальных сесть.

Джейн сразу заметила диаграммы на стенах и стенды с плакатами, расставленные по всему залу. Она повернулась к Дэну,



Фергюсон и Барделл

Строительство · Проектирование · Управлениепроектами

Проект создания системы управления ресурсами
Повестка дня

Дата: 29 марта 1999 г. **Тема:** определение целей проекта

I. Уточнение повестки

II. Обзор модели разработки

III. Обязанности членов группы на разных этапах
- Выработка концепции
- Проектирование
- Разработка
- Стабилизация

IV. Итерационный подход

V. График и цели проекта

VI. Вопросы и ответы

VII. Задания и ответственные

Чикаго · Детройт · Милуоки · Цинцинати · Индианаполис · Луисвилль

– Боже мой, Дэн, ты работал все выходные, как бобер! — она показала рукой на зал: — Зачем все это?

Прежде чем обратиться к собравшимся, Дэн разложил перед собой какие-то бумаги.

– Ну что ж, ребята, сейчас я все объясню. После нашей встречи на прошлой неделе я отправился к Майку Адэмли, исполнительному

директору нашей компании, и объяснил ему, что мы собираемся делать. Я сказал ему, что нам необходимо много места для работы, точнее, что нам необходима своя «нора». Он разрешил мне воспользоваться этим залом. Следующие два месяца он будет нашим домом.

– Мы два месяца будем пользоваться дубовым залом? Ух ты, мы становимся важными персонами! — сказал Тим с самодовольной улыбкой, глубоко откидываясь в кресле.

– Не очень-то важничай, — сказала Джейн. — Важны не мы, а наша работа.

— Совершенно верно, Джейн, — сказал Дэн. Он включил проектор и вложил в него слайд: — И настало время сдвинуть эту важную работу с места. Поэтому, если вы не против, вернемся к повестке дня, которую я разослал вам на выходных. Давайте приступим.

Ощутив перемены в голосе Дэна, все быстро достали свои папки.

– Хорошо, — продолжил Дэн, — сначала, как всегда, повестка дня, У кого-нибудь есть какие-нибудь дополнения?

– У меня есть, — сказал Тим. Все удивленно уставились на него, Тим настолько восхищался Дэном, что никогда не «поднимал голос».

– Хорошо, Тим, для этого мы и собираемся. Что ты хотел бы изменить?

Тим вдруг притворился сонным.

— Ну, я думаю, вы знаете, я обычно очень спешу, чтобы успеть на встречу, — сказал он так, будто бы это было откровением для остальных членов группы. — Я знаю, что во время встреч нам подают кофе и сок, но я надеялся, что это будет что-то более серьезное.

— Например, завтрак? — засмеялась Джейн. — Чего ты от нас хочешь? Устроить для тебя поздний завтрак?

Тим залился краской и сказал:

– Нет, не совсем так. Я лишь хотел чего-нибудь «поклевать». Ну, скажем... пончики!

Все расхохотались. Любовь Тима к сладкому была общеизвестна.

– Ну и кто же, — сказал Дэн — должен обеспечивать тебя этими деликатесами?

– Я подумал, что все по очереди — сказал Тим и, порывшись в своем рюкзаке, достал полную коробку пончиков. — И решил сделать первый шаг.

Дэн рассмеялся и ответил:

– Тим, я всегда знал, что ты не забудешь ничего важного. — Напряжение в его голосе исчезло: — Ну ладно, все взяли из шкафа по салфетке и принялись за пончики.

Пока все поглощали пончики, Дэн язвительно заметил:

— Вообще-то все поправки к повестке должны приниматься голосованием, но предложение Тима, кажется, и так прошло единогласно. Отличная мысль, Тим. Будем заказывать пончики на каждую встречу. А теперь давайте займемся делом.

Особенности модели процесса разработки MSF

Дэн взглянул в повестку:

— Первое, что мы должны сделать — это проверить, как вы справились с домашним заданием. Кто может мне сказать, какова первая фаза модели процесса разработки MSF и зачем она нужна?

Дэн, ожидая ответа, держал слайд в руке:

— Мэри-Лу?

— Первая фаза называется «Анализ». Ее цель — во-первых, понять проблему, которую мы решаем и, во-вторых, построить концепцию, с которой были бы согласны все участники проекта.

Дэн одобрительно кивнул и вставил слайд в проектор. Это была диаграмма модели процесса разработки, которую они уже видели на первом собрании.

— Процесс разработки MSF состоит из четырех фаз: анализ, планирование, разработка и стабилизация. У каждого из них свои результаты, и каждый завершается основным этапом. У каждой фазы есть промежуточные этапы. Кто может объяснить мне разницу между итоговым и промежуточным этапом?

— Это просто, — сказал Тим, откусывая пончик. — Результаты промежуточных этапов готовятся и изучаются исключительно проектной группой; заказчик о них ничего не знает. Результаты основного этапа, напротив, согласуются с заказчиком. Кроме того, по результатам основного этапа заказчик и группа вместе решают, нужно ли продолжать проект дальше. — Он откусил кусок пончика и продолжил: — Каждый этап — это момент, когда все участники проекта могут оценить ход проекта и синхронизировать свои действия.

— Хорошая работа, Тим, — сказал Дэн.

Когда Тим потянулся в коробку за следующим пончиком, Дэн вытащил ее у него из под носа и сказал:

— А теперь, чтобы получить призовой пончик, назови мне по порядку 4 основных этапа.

Все засмеялись над шуткой Дэна, а Тим подошел к доске, взял маркер и нарисовал схему четырех фаз модели процесса разработки. Он надписал названия всех фаз, потом названия основных этапов, и наконец голосом первоклассника произнес:

— Фаза «Анализ» завершается этапом «Одобрение концепции», фаза «Планирование» — этапом «Одобрение плана проекта», фаза

«Разработка» — этапом «Завершение разработки», а фаза «Стабилизация» — этапом «Выпуск».

Он изысканно поклонился в ответ на общие аплодисменты и вернулся к своему креслу.

— Великолепно! — улыбаясь, воскликнул Дэн, довольный тем, как молодой менеджер принял вызов, и вернул ему коробку с пончиками. — Будем надеяться, что все остальные знают это так же хорошо, как ты. — Он повернулся к остальным членам группы: — Итак, кто может описать принципы модели процесса разработки MSF?

Марта застенчиво подняла руку.

- Марта?

Марта изложила все принципы процесса разработки MSF, загибая пальцы:

- Выпуск версий, постоянно обновляемые документы, планирование с учетом неопределенности, поиск компромиссов, анализ и контроль рисков, ориентация на выпуск продукта в срок, разбиение больших проектов на управляемые части, ежедневная сборка приложения и планирование снизу-вверх.

Она села и улыбнулась Дэну.

Остальные ошеломленно уставились на нее. Тим присвистнул — низко и протяжно.

- Ну и Марта, ничего себе! — Он переправил коробку с пончиками через стол Марте. — Бери, ты заслужила все.

Марта, посмеиваясь, отправила коробку обратно:

- Нет, спасибо, Тим. Я, в отличие от тебя, не очень люблю пончики. Оставь их на потом.

Помолчав, Дэн сказал:

- Впечатляет, Марта. Я не уверен, что смогу перечислить их по порядку, как ты, без бумажки.

Роли и обязанности

Дэн вложил в проектор следующий слайд. На экране появилась таблица, изображавшая четыре фазы модели процесса разработки с оставленными пустыми местами рядом с названием каждой из них.

- Мы уже потратили некоторое время на освоение модели проектной группы, и каждый из вас уже знаком со своей ролью и обязанностями в проекте. Теперь я хочу увязать эти роли с моделью процесса разработки MSF, — он повернулся так, что смог писать на слайде и продолжил: — Как менеджер программы, я отвечаю за общий ход проекта. Но каждый из вас должен довести какую-то часть проекта до логического завершения. Скажите мне, основываясь на ваших зна-

ниях о ролях в проекте, кто из членов группы отвечает за этап «Анализ»?

– Это я, не так ли? — ответила Джейн, перерисовывая схему в свой ежедневник. — Во-первых, на стадии «Анализ» проектная группа и заказчик согласовывают общую точку зрения на проект и процесс разработки, во-вторых, я одна пока что общалась с заказчиком. Поэтому я, как менеджер продукта, должна отвечать за стадию «Анализ» и за этап «Одобрение концепции»,

Остальные кивнули головами, когда Дэн сказал:

— Совершенно точно, Джейн. Именно менеджер продукта ведет группу к первому этапу. — Он написал *«менеджер продукта»* рядом с первой фазой и продолжил: — Кстати, вы помните, что я говорил относительно плана рассаживания: у меня есть такой план для одного из вас? Джейн, со следующего раза ты будешь сидеть во главе стола и начинать наши собрания.

Мэри-Лу воскликнула:

— Эй, Джейн, ты поднимаешься вверх по служебной лестнице!

Сомнение появилось на лице Джейн.

– Я не знаю, что и думать, Дэн, — произнесла она, глядя на остальных. — Я не уверена, что смогу это делать. О некоторых деталях этого проекта я почти ничего не знаю.

— Да, это так. Но ты знаешь заказчика лучше, чем любой из нас, поэтому ты лучший кандидат на руководство созданием концепции проекта. Не беспокойся, Джейн; по мере продвижения проекта я буду помогать тебе делом и советом. И не забудь — это хорошая группа.

Рядом с названием следующей фазы — планирование — Дэн написал свое имя,

– Менеджер программы отвечает за общий ход проекта, но он прежде всего ответственен за фазу «Планирование». Хороший менеджер программы начинает планировать проект в тот день, когда его взяли на работу.

Дэн повернулся к слайду:

— Давайте сразу перейдем к третьей фазе, фазе разработки. Я думаю, что все мы можем назвать имя человека, который будет за это отвечать.

Билл Парди заерзал в кресле:

— Резонно предположить, что это буду я. не так ли? Поскольку это проект *разработки,* пора бы уже привлечь к нему отдел разработки.

Всем стало ясно, что Билл не был активным сторонником использования модели процесса разработки MSF. Остальные уставились на Дэна, ядовито улыбнувшегося Биллу.

- На самом деле, Билл, ты увидишь, что ты и твои парни окажетесь вовлеченными в проект гораздо раньше, чем ты думаешь. Но все же сначала мы должны закончить распределение ролей по этапам. — Билл пожал плечами, и Дэн продолжил: — В нашей группе шесть ролей, а этапов — всего четыре, поэтому за выполнение некоторых этапов будут отвечать несколько человек. Как ты думаешь, Билл, кто из сотрудников будет вместе с тобой отвечать за фазу «Разработка»?

Билл и вместе с ним все остальные начали удивленно оглядывать аудиторию, пытаясь представить себе этого несчастного.

— Я дам тебе подсказку, Билл, — сказал Дэн. — Это тот, кого больше всего ценит опытный разработчик.

Билл нахмурился, перебирая возможные кандидатуры. В конце концов, он сказал:

— По-видимому, наши программисты должны прислушиваться ко всем остальным членам группы. Заказчики, операторы, тестеры — все эти люди чрезвычайно важны для проекта.

Дэн продолжил, одобрительно кивнув:

— Совершенно точно, Билл, и поэтому структура проектной группы MSF именно такова. Но когда ты разрабатываешь приложение, одна группа занимает главенствующее положение.

На лице Билла опять отразился напряженный мыслительный процесс.

- Пользователи! Поскольку мы проектируем приложение для них, мы должны работать с ними! — тут он понял, что в этой группе с пользователями общается исключительно Мэри-Лу, и обернулся к ней: — Это означает, что ты будешь работать вместе со мной.

Он вспомнил, как назвал ее «инструктор по нажиманию кнопок», и подумал, помнит она это или нет.

Мэри-Лу засмеялась и положила руку ему на рукав:

- Не беспокойся, Билл. Даже от нас, «специалистов по нажатию кнопок», есть толк, когда дело доходит до пользователей. Я обещаю не писать ни одной строчки кода, пока ты меня об этом не попросишь.

Все громко засмеялись, а Тим сказал Биллу театральным шепотом:

- Кажется, она тебя уела, старик.

Билл выглядел ошарашенным.

— На самом деле роли «Разработчик» и «Инструктор» действительно взаимодействуют на фазе «Разработка», — сказал Дэн, записывая названия ролей на слайде. — Мэри-Лу будет работать с пользователями первых версий и передавать их отзывы мне и Биллу, чтобы группа разработчиков могла учитывать их в своей работе. — Он повернулся к

Биллу; — Если люди будут отзываться о работе не после выхода продукта, а во время его разработки, твоя работа пойдет лучше и быстрее.

Тим шепнул Биллу:

— Я думаю, это очень хорошее предложение. Теперь у тебя есть кто-то, кто будет общаться с пользователями вместо тебя. Они любят ее и с удовольствием будут разговаривать с ней. Посмотри: она встречается с ними, приносит тебе информацию и их идеи, а ты знай себе вставляешь их в продукт. Ты будешь героем, и тебе даже не придется для этого улучшать свои манеры.

Все добродушно засмеялись, и даже мрачноватый Билл улыбнулся.

Тим сказал Дэну:

— Судя по тому, что осталось, за последний этап отвечают «Тестер» и «Логистик», то есть Марта и я, не так ли?

Дэн написал названия этих ролей рядом с последней фазой и сказал:

– Все правильно, Тим. Сначала мы тестируем приложение, а затем развертываем его. Дело Марты — убедиться, что приложение готово к передаче пользователям, а твоя работа — решить, как это сделать. Как только вы оба закончите, можно считать, что первая версия приложения завершена.

– Здорово, — сказал Тим, снова откидываясь в кресле. — А до тех пор мы с Мартой можем расслабиться.

— Ну нет, — ответил Дэн. заправляя в проектор следующий слайд. — Ты забыл об итерации в рамках одного выпуска.

Тим подскочил в кресле и изумленно уставился на Дэна, как и все остальные.

Итерации

– Посмотрите сюда, — сказал Дэн, показывая на слайд. — Это модель процесса разработки MSF, только я распрямил ее цикл. Что это вам напоминает?

— Хм... найдите необходимые средства, придумайте план, разработайте приложение, протестируйте его и приступайте к развертыванию, — прочитала Мэри-Лу, — Да это же...

– Модель водопада под другим названием, — категорично произнес Билл, — Я давно говорил, что ваша MSF — это тот же старый велосипед, только называется по-новому.

— Подожди, Билл, — сказал Дэн, делая какие-то пометки на слайде. Он нарисовал маленькие циклы модели процесса разработки MSF рядом с каждой фазой на прямой линии. — Билл, разве это похоже на достопамятную модель водопада?

— Дэн, я тоже не поняла этого, когда читала переданные тобой материалы, не понимаю этого и сейчас, — произнесла очевидно озадаченная Марта. — Ты хочешь сказать, что, выполняя каждую стадию процесса разработки, мы повторяем общий цикл проекта?!

– Марта, ты угадала. И здесь мы слегка отклоняемся от строгого следования модели MSF, — пояснил Дэн. — На каждом из этапов нам придется выполнить какую-то часть фазы «Анализ», часть планирования, часть разработки и часть стабилизации. На некоторых этапах мы будем совершать этот цикл не один раз, а несколько.

— Скажи, а промежуточные этапы на этих мини-циклах тоже будут? — спросила Джейн.

– Нет, эта система не похожа на фотографию двух зеркал, где изображения повторяются, все уменьшаясь, пока не перестанут быть различимыми. Здесь меньшие структурные единицы лишь подобны, а не в точности повторяют ход всего процесса разработки MSF с промежуточными результатами внутри промежуточных результатов и т. д.

Итерации подразумевают, что мы занимаемся и планированием, и программированием в течение всего проекта. Мы даже немного позанимаемся тестированием и развертыванием на этапе «Анализ», не дожидаясь конца проекта.

— Следовательно, — медленно сказала Мэри-Лу, обдумывая сказанное, — это означает, что даже концепция проекта не будет завершена до конца проекта, хотя она и одобрена еще на первой стадии.

– Именно, — сказал Дэн. — И именно поэтому возникает концепция живых, постоянно обновляемых документов. Пока мы занимаемся проектом, все материалы открыты для уточнения на основе новой или изменившейся информации, которую мы получаем по мере продвижения к цели.

– Что удерживает процесс от зацикливания? — спросил Билл с негодованием в голосе. — Мне кажется, что ты будешь выполнять итерацию за итерацией, пока не выйдут время и деньги,

– Число итераций ограничено, — ответил Дэн. — И, кроме того, ты в начале каждой итерации заявляешь ее цели, чтобы любой член группы знал, когда и чем она должна закончиться.

— Сколько итераций будет в проекте разработки RMS? — спросила Джейн.

– Это во многом зависит от графика проекта, — ответил Дэн. — Это наша следующая тема, давайте перейдем непосредственно к графику. — С этими словами он вставил в проектор новый слайд: — Что вы по этому поводу думаете?

Увидев график, все разинули рты.

График проекта RMS и первоначальные цели

— Слушай, а ты не шутишь? Два месяца?! — Билл выглядел изумленным. — Ты действительно думаешь, что мы успеем сделать все к намеченному сроку?

- Билл, у нас нет и не будет однозначного ответа на этот вопрос, пока мы не займемся планированием. Припомни, ведь составление графика — одна из задач фазы «Планирование». Но сейчас мы можем построить приблизительный график проекта, который даст нам каркас для составления подробного графика. Вот этим-то я тут и занимался.

- Давайте посмотрим, — сказал Джейн, переписывая график в записную книжку. — Похоже, у нас есть полторы недели на анализ, еще полторы — на планирование, три недели на разработку и две недели на стабилизацию. Даю голову на отсечение, что в анализе будет одна итерация, в планировании — тоже, в разработке — две, а в стабилизации — тоже одна. Я права?

Дэн, улыбаясь, произнес:

— Абсолютно, Джейн. Твое бухгалтерское образование тебе отлично помогает.

— Дэн, а как насчет целей итераций? Ты говорил, что мы должны обговорить их, — спросила Марта.

- Да, я набросал первоначальный список целей проекта — он здесь, — ответил Дэн, демонстрируя диаграмму на бумаге. Поглядывая на копию, лежащую перед ним, Дэн продолжил: — Прежде всего, я сделал список целей разработок для каждой итерации — их проще всего планировать.

- На первой итерации — единственной на стадии «Анализ», цель разработки — прототип. Прототип — это визуализация нашего коллективного мнения о том, как должно выглядеть приложение. Это инструмент для общения с заказчиком, и это лишь набор интерфейсов. Билл, скажи, твои ребята могут сделать прототип за неделю?

- Без проблем. С новыми средствами визуального проектирования можно и быстрее. Уж интерфейсы-то мы сконструируем, — отозвался Билл.

- Замечательно. Итак, главная цель на первой итерации — построить прототип и использовать его как инструмент для создания концепции продукта. Теперь относительно итерации фазы «Планирование». Здесь наша цель — концептуальная версия приложения. В этой версии мы будем работать над основными вопросами проектирования и архитектуры и попробуем убедиться, что наш проект выполним. В отличие от прототипа, концептуальная версия функциональна и более или менее способна работать. Прототип — это исклю-

чителъно интерфейсы, а концептуальная версия — напротив, вся «начинка» приложения: доступ к данным, серверные компоненты и тому подобное.

-- Я понял! — сказал Билл, смягчаясь. — Насколько я понимаю, на фазе «Разработка» мы создаем альфа- и бета-версии приложения. Все, как и раньше, но... — Билл явно увлекся, — ...теперь альфа-версия уже почти готова, ведь к этому времени мы уже почти все для нее сделаем! Мы просто возьмем интерфейсы из прототипа, «начинку» — из концептуальной версии, соединим их с помощью связующих объектов — и готово! Похоже, модель процесса разработки MSF сделает нашу альфа-версию больше похожей на первые бета-версии моих прежних проектов.

-- Совершенно верно, Билл. Ты понял главную идею, — сказал Дэн. — Теперь, на последней итерации, мы сводим воедино все результаты групп тестирования и логистики и выпускаем «золотую» — итоговую — версию. И на этом мы отстрелялись.

Все замолчали. Наконец, заговорила Джейн:

-- Ну, если я сегодня старшая, у меня вопрос — что дальше?

— Хороший вопрос, Джейн, — ответил Дэн, — Согласно повестке у нас следующий вопрос — распределение обязанностей. Однако сначала давайте посмотрим, нет ли у кого-нибудь вопросов? — Он сделал паузу. После минутного замешательства Тим поднял руку. — Что, Тим?

— Дэн, все выглядит превосходно. Я имею в виду, что быстро сделать проект — это здорово, мне это очень нравится, Но это радикально отличается от того, что мы делали раньше, и я чувствую неуверенность. Думаю, все остальные тоже. — Все закивали, и Тим продолжил: — Может, ты покажешь нам какой-нибудь пример, как какая-нибудь группа все это делала? Это бы нам очень помогло.

Дэн кивнул и ответил:

-- Тим, самый лучший способ научиться — посмотреть, как это делают другие. Поэтому я и приготовил для вас документы другого проекта, менеджера которого я знаю. Он разрешил показать вам их.

Пока Дэн говорил, на его лице появилась улыбка, и Тим воскликнул:

— Да ты сам был тем менеджером! Это документы из юридической компании, где ты работал.

-- Это правда, и я даже не очень их чистил, так что вы сразу увидите, каким неопытным я тогда был, — ответил Дэн. — Нам всем это было в новинку, однако мы смогли сделать за шесть месяцев проект, который все оценивали как трехлетний. Конечно же, нам пришлось выпустить еше две версии в течение года, где мы реализовали все

функции, однако все базовые возможности вошли в первую версию. Руководство моей старой компании разрешило поделиться с вами этими материалами, и я надеюсь, что это поможет вам разобраться в концепции.

Он заметил, что Марта выглядит подавленной,

— Марта, ты выглядишь так, как будто что-то тебя очень беспокоит. Я могу чем-то помочь?

- Нет, не сейчас. Я сначала посмотрю эти документы, а потом зайду к тебе поговорить.

Все обратили внимание на тон ее голоса, но Дэн решил сделать вид, будто ничего не заметил:

— Отлично. Моя дверь всегда открыта для всех вас. Заходи, и мы что-нибудьпридумаем.

Дэн повернулся к группе:

- Еще есть какие-нибудь вопросы? Нет? Тогда рассмотрим обязанности и определим ответственных. Каждый должен взять стопку документов из коробки на полке и прочесть их к нашей следующей встрече утром в среду. На основе этих материалов и ваших знаниях о приложении каждый из вас должен выявить хотя бы одну цель, относящуюся к области, за которую вы отвечаете, для каждой итерации. Я хотел бы получить ваши предложения по электронной почте завтра утром, тогда я успею прокомментировать его до нашей следующей встречи. Это еще не будет окончательный список, но с этих материалов мы начнем.

Билл, ты должен помочь своим людям сделать прототип. Я надеюсь, что мы начнем работать с ним после нашей встречи в среду. Джейн, не могла бы ты остаться на минутку? Я хочу рассказать тебе о сценариях использования, чтобы ты смогла работать с ними на следующей встрече. Все понятно? Следующая встреча в среду утром, время и место — то же. Мэри-Лу, твоя очередь обеспечить нас пончиками, хорошо?

Когда встреча закончилась, Дэн обратил внимание на контраст в поведении Марты и всех остальных. Все разговаривали об RMS, обсуждали возможные цели и проблемы — в общем, получали удовольствие от работы. Все, кроме Марты. Взяв пакет с бумагами, она посмотрела на остальных и тихо вышла из комнаты. Дэн подумал: «Когда я наконец убедил Билла, на горизонте появилась новая проблема, Я надеюсь, Марта поговорит со мной перед завтрашней встречей, и мы сможем найти обший язык».

К сожалению, она этого не сделала. В результате следующая встреча началась скандалом, которого никто не ожидал.

Часть II. Проектирование

ГЛАВА 5

Предпроект

В этой главе

Без хорошо продуманного, ясно сформулированного, ориентированного на цели бизнеса представления о будущем продукте (назовем это *видением,* или *концепцией продукта)* никакая разработка не может быть успешной. В этой главе мы опишем динамику фазы «Анализ» модели процесса разработки MSF. Мы обсудим, какую информацию необходимо получить от участников проекта и как создавать концепцию продукта. Мы расскажем о партнерстве различных ролей в проектной группе, созданной в соответствии с моделью группы разработчиков MSF, и выясним, какова область ответственности каждой роли на фазе «Анализ». Мы также посмотрим, как процесс исследования развивается во времени. Наконец, мы детально обсудим процесс управления рисками, базирующийся на модели управления рисками MSF. Управление рисками выполняется в течение всего процесса разработки — от начала и до конца,

В этой главе мы использовали наш собственный опыт проектирования и реализации архитектуры приложений, и следующие материалы;

- учебный курс 1516 «Principles of Application Development»;
- доклад Лауры Литвак (Laurie Litwack), старшего программного менеджера подразделения «Решения для малого бизнеса» компании Microsoft, «Creating a Vision for Your Product».

Изучив материал ЭТОЙ главы, вы сможете:

- описать роли членов проектной группы на фазе «Анализ» процесса разработки;
- описать ход фазы «Анализ» модели процесса разработки MSF;
- разобраться в концепции работы с рисками и оценить важность постоянного процесса управления рисками;
- описать процедуру оценки риска;
- понять разницу между снижением риска и планом действий на случай реализации риска;
- описать результаты, необходимые для достижения этапа «Одобрение концепции»;
- описать назначение и содержание документа «Концепция продукта»;
- описать оптимальные методы согласования концепции.

Предпроектное исследование

Успех проекта зависит от способности проектной группы и заказчиков добиться *общего видения* целей и задач проекта. Обычно эти цели и задачи являются производными от целей организации. Концепция применяется не только при реализации технологии, поэтому соотнесение проекта со стратегическими целями организации требует тщательного анализа.

Прежде чем группа перейдет к следующим фазам проекта, надо разработать концепцию, которая описывает основные цели и направление проекта. Для выявления всех элементов идеального проекта и его соогношения с бизнес-целями организации хорошо себя зарекомендовали мозговые штурмы. При таком способе решения проблемы члены группы, хорошо знающие цели бизнеса, получают возможность применить свои знания и способности для выработки оптимальной концепции.

Хотя разработка концепции проекта состоит из нескольких итерационных циклов, необходимо, чтобы к концу процесса исследования все участники проекта, прямо и косвенно отвечающие за него, получили полное представление о нем. Очень важен процесс согласования: он помогает проектной группе и заказчику прояснить цели проекта и укрепляет их заинтересованность в конечном результате, которую они должны сохранить до завершения проекта. Без четко определенной концепции очень легко увязнуть в мелочах.

Ясная концепция обеспечивает успех проекта, указывая группе направление работы, однако для этого концепция должна непосредственно решать практические задачи проекта. Это достигается посредством включения в модель процесса разработки MSF *фазы «Анализ»* (рис. 5,1); кроме того, концепция, созданная на этой фазе, предназначается прежде всего для текущей версии продукта. Можно разработать и долгосрочную концепцию — для всех версий продукта; она пригодится для стратегического планирования следующих выпусков продукта. Каждый последующий выпуск продукта будет в этом случае реализацией отдельных пунктов долгосрочной концепции, что, в свою очередь, позволит оценивать успех каждого выпуска, сопоставляя его результаты со стратегическими целями.

Результат фазы «Анализ» — предпроект — представляет собой документ, описывающий концепцию проекта. Он детализирует:

- концепцию проекта;
- рамки проекта;
- исходную информацию;
- бизнес-требования;
- проектные требования;
- результаты;
- риски.

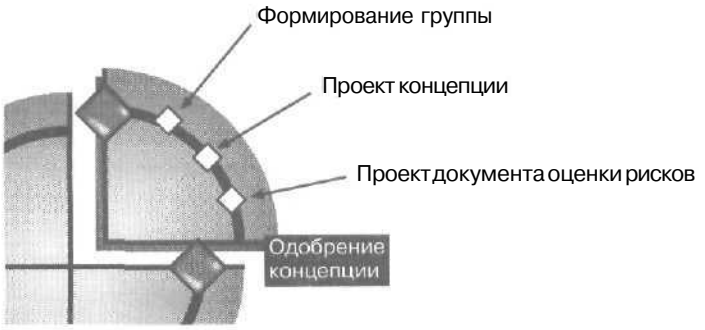


Рис. 5.1. Фаза «Анализ» модели процесса разработки

В этой главе мы опишем фазу «Анализ» и предпроект, а также связи его элементов. Имейте в виду, что на фазе «Анализ» группа использует для обмена информацией разные документы. Однако эти документы не являются конечным результатом фазы — ее целью является как раз налаживание обмена информацией. Документы просто служат инструментом для достижения этого результата.

Фаза «Анализ» завершается, когда группа достигает этапа «Одобрение концепции». Это тот момент, когда проектная группа и заин-

тересованные в проекте стороны принимают совместное решение об одобрении концепции конечного продукта и решают, что можно переходить к фазе «Планирование».

Назначение концепции

Разработка продукта без создания концепции подобна плаванию корабля без компаса: корабль куда-то плывет, но куда? Знает ли экипаж маршрут движения? Прибудут ли пассажиры именно туда, куда купили билет?

Хотя разработка концепции продукта безусловно необходима, именно этот этап разработки часто пропускается или недооценивается. Большинство людей, вовлеченных в процесс разработки приложения, имеют гораздо больше навыков в решении задач, необходимых для выпуска продукта, чем в создании его концепции. Многие находят саму идею концепции продукта расплывчатой и неясной, особенно это касается новичков в разработке продуктов. Однако очень важно понимать что анализ:

- служит первой фазой планирования и проектирования;
- позволяет добиться понимания и согласия между всеми участниками с самого начала проекта;
- помогает группе объединить разные точки зрения в общую концепцию;
- обеспечивает основу для будущего планирования;
- выявляет факторы, которые заказчик и основные участники считают важнейшими для проекта.

Сейчас, когда все больше и больше организаций считают необходимым «удовлетворить клиента», разработка концепции становится еще более актуальной, чем прежде. Это первый и самый ответственный шаг разработки, потому что именно он определяет направление, в котором будет двигаться проектная группа, и цель «путешествия». В своем интервью Лауре Литвак Стив Синовски (Steve Sinofsky), в то время вице-президент Microsoft по выпуску продукта Microsoft Office, отмечал:

> *«Концепция важна потому, что она помогает команде принимать решения, наиболее походящие для достижения поставленной цели. В этом смысле хорошая концепция является инструментом, который объединяет всех сотрудников в единую команду, создающую отличный продукт. Если над проектом работают 10 человек, они всегда могут поговорить друг с другом, достигнув общего понимания целей и действий. Когда же над проектом работает 1000 человек, нужен способ, обеспечивающий принятие решений без необходимости каждому участнику разговаривать с остальными 999».*

Ясная концепция создает доверие и единство членов группы, указывает перспективу, уточняет цели и упрощает принятие решений. Общее видение дает группе энергию для достижений наилучших результатов и обеспечивает:

- **ясность** — члены группы должны осознать, что же они пытаются создать, прежде чем приступать к разработке. Кроме того, они должны понимать, почему менеджер программы, менеджер продукта и заказчики включают тот или иной пункт в список требований и не включают другие. Концепция должна давать ясное представление не только о том, что будет делаться, но и о том, что *не* будет делаться;
- **расстановку приоритетов** — на все, что приходит в голову, никогда не хватает времени. На всем протяжении жизненного цикла продукта проектной группе нужны критерии, которыми можно руководствоваться при выборе решений из массы вариантов. Группа разработки решает, как кодировать все, что описано в функциональных спецификациях и технических требованиях, и как заложить основу для будущих версий продукта. Группа тестирования выявляет ошибки. Группа обучения пользователей решает, какие возможности следует выделить особо и как объяснить их пользователям. Менеджер программы взаимодействует с разработчиками, чтобы решить, как реализовать эти возможности и как пользователь будет работать с каждой из них. Для принятия решений команде нужно понимать не только то, что и почему должно делаться, но и в каком порядке. Удачная концепция образует иерархию приоритетов для принятия решений;
- **интеграцию** — концепция продукта должна поддерживать и дополнять концепции и функциональные возможности других продуктов, использующихся в организации. Многие другие продукты, возможно, обслуживают тех же клиентов и пользователей, что и разрабатываемый вами продукт, и, возможно, некоторые функции перекрываются. Любое их дублирование в различных продуктах надо изучить, чтобы выяснить, оправдано ли оно с точки зрения бизнеса;
- **будущие инвестиции** — назначение концепции — не только направлять разработку сегодняшних продуктов, но и заложить основу на будущее. Например, если отдел разработки предполагает включить в следующую версию продукта средства поддержки электронной коммерции, уже в текущей версии необходимо принять во внимание низкую пропускную способность каналов связи. Такая предусмотрительность поможет сберечь время в будущем.

Трудности фазы «Анализ»

Применяя MSF и модель процесса разработки MSF на практике, мы пришли к выводу, что ее использование заметно повышает как эф-

фективность работы проектной группы, так и степень удовлетворенности заказчика. Однако, как всегда, у этих моделей есть свои опасности. В этом разделе мы опишем некоторые ловушки, в которые нам случалось попадать.

1. **Игнорирование динамической природы модели процесса разработки** -- поначалу мы воспринимали процесс исследования как догму, не осмеливаясь ни на йоту отклониться от документации MSF. Однако, когда мы накопили кое-какой опыт, мы научились перекраивать процесс и его составные части применительно к нуждам и размерам организации. Фаза «Анализ» — это инструмент для обмена информацией и выяснения общей точки зрения, а не закон, нарушение которого карается.
2. **Неполное выполнение стадии «Анализ»** — сначала нам не всегда хватало дисциплины, чтобы полностью провести исследование до начала выполнения проекта. Из-за недостатка времени и средств есть соблазн предпочесть традиционные методы, а исследование отложить на случай, когда оно действительно необходимо. Нужно найти в себе силы противодействовать этому побуждению и продолжать придерживаться логики модели, потому что для успеха проекта жизненно важны все ее детали .
3. **Непонимание того, что фаза** «Анализ* **закладывает основу для разработки сложных концепций** — исследование побуждает проектную группу глубоко проанализировать сложные стороны проекта. Это помогает создать легко реализуемую концепцию, ускоряющую и рационализирующую процесс разработки.

Процесс исследования

Принципы и рекомендации, которые мы приводим в этом разделе, с небольшими изменениями заимствованы из доклада Лауры Литвак — опытного руководителя группы разработки операционных систем, серверных приложений и сетевых подсистем,

Распределение обязанностей

Фаза «Анализ» начинается с создания проектной группы и распределения ее участников по шести основным ролям. (Подбор группы и методы управления проектными группами подробно обсуждаются в главе 3.) Все роли принимают участие в исследовании.

В табл. 5.1 перечислены конкретные обязанности ролей на фазе «Анализ». Руководитель каждой роли отвечает за решение соответствующих задач и обмен информацией с остальными членами проектной группы.

Табл. 5.1. Обязанности ролей на стадии «Анализ»

Роль	Обязанности
Менеджер продукта	Подготовка документа «Концепция проекта» Работа с заказчиком Вовлечение заказчика в разработку прототипа Управление рисками
Менеджер программы	Формулировка целей проектирования Описание концепции решения Выявление структуры проекта Управлениерисками
Разработчик	Разработка прототипов Выбор основных направлений разработки Выявление требований и взаимосвязей Управление рисками
Инструктор	Сбор требований, касающихся удобства использования Работа с пользователями Вовлечение пользователей в разработку прототипов Управление рисками
Тестер	Разработка стратегии тестирования Формулировка критериев приемки продукта Разработка системы выявления ошибок Разработка системы управления рисками Управление рисками
Логистик	Вопросы развертывания и их влияние на проект Вопросы сопровождения и их влияние на проект Управление рисками

Как видно из таблицы, все роли вовлечены в работу по оценке рисков. Особенности выявления рисков и управления ими обсуждаются далее в этой главе.

Укажем несколько этапов, которые могут быть полезны проектной группе на стадии исследования. Они подобны тем, что используются в универсальной модели процесса, которую мы обсуждали в главе 4. Мы не считаем, что следование этим этапам — единственный путь выполнения фазы «Анализ», но они помогут тем, у кого нет опыта в использовании модели процесса разработки MSF. Нумерация этапов приведена только для наглядности: речь не идет о строгой последовательности действий или о формальных отчетах. И еще: эти этапы логически последовательны, но не воспринимайте их последовательность как догму. Например, если на шаге 4 (реализация) обнару-

жились непредусмотренные схемы использования продукта, можно вернуться к шагу 2 и включить их в перечень.

Шаг 1: изучение

Чтобы концепция оказалась полной, группа должна заняться исследованием, сосредоточив свои усилия на изучении нужд заказчика и аналогичных продуктов, разрабатываемых или используемых в организации. Задача на этом шаге — сбор и обобщение как можно большего объема информации. Источники этой информации — заказчики проекта, пользователи, существующие приложения и документация к ним, а также эксперты в области разработки программного обеспечения или в соответствующих предметных областях (таких как дистрибуция и производство). При назначении приоритетов группе стоит ограничиться этими источниками; назначение дополнительных приоритетов выполняется позже. На основе собранной информации группа формирует концепцию, в рамках которой в общих чертах определяются приоритеты, выявляются зависимости и формируется предварительныйграфик.

Изучение других приложений

Если у продукта есть предыдущая версия, группа должна начать с изучения концепции этой версии. Чаще всего следующая версия приложения базируется на концепции предыдущих версий. Точно так же группе следует учитывать концепции других приложений, находящихся в разработке.

Перечислим вопросы, которые стоит задать при создании концепции нового продукта.

- Использование нового продукта вызовет качественный скачок в работе организации или это лишь небольшой шаг вперед?
- Являются ли концепция и архитектура новыми для организации?
- Какие новые возможности принесет продукт организации?

Изучение заказчика и пользователей

Мы постоянно подчеркиваем, как важно знать и понимать требования заказчика. Это необходимое условие успеха любого вида разработки. Цель фазы «Анализ» — выявить проблемы заказчика и особенности работы пользователей с продуктом, Здесь важно ответить на следующие вопросы.

- **Инициативы** — какие инициативы предпринимает организация? Как они повлияют на продукт?
- **Существующая информация** — что еще (кроме того, что смогла выяснить группа) известно о заказчике? В какой части проекта группа может использовать опыт предыдущих работ? Какую информацию о заказчике надо прояснить или подтвердить?

- **Новое аппаратное обеспечение** — как повлияет на требования заказчика применение нового аппаратного обеспечения? Как использовать новое аппаратное обеспечение (например, сенсорные экраны) для целей проекта?
- **Новые технологии** — как повлияют на требования заказчика новые технологии? Как использовать новые технологии, чтобы помочь заказчику? Например, стоит ли применять мощные средства поиска или распознавания голоса?
- **Вопросы поддержки** — какие проблемы возникали при использовании аналогичных продуктов? Каковы 10 основных проблем сопровождения программных продуктов в организации? Какие возможности новых приложений наиболее востребованы? Как оценивают концепцию группы сопровождения и эксплуатации?
- **Планы обучения пользователей** — как, по мнению группы обучения пользователей, необходимо документировать продукт? Какие возможности и концепции предшествующих приложений трудны для понимания? При описании каких возможностей предыдущих приложений возникли трудности? (Обычно средства, которые трудно документировать, трудно и использовать.)
- **Глобальные проблемы** — пользователи каких стран заинтересованы в англоязычной версии продукта? Смогут ли зарубежные пользователи работать с этой версией? Какие исследования необходимо провести, чтобы убедиться, что продукт удастся использовать в разных странах мира? Понадобятся ли локализованные версии продукта?

Изучение аналогичных продуктов

Концепция, которую группа разрабатывает для продукта, должна соответствовать концепциям других продуктов, используемых в организации. Надо сказать, что это самое простое исследование, поскольку речь идет не столько о сборе и анализе информации, сколько об использовании в проекте уже накопленных другими знаний. Вот какими вопросами должна задаваться группа на это этапе.

- **Точки соприкосновения** — каковы концепции других продуктов, разработанных для того же заказчика? Можно ли воспользоваться результатами разработки этих продуктов?
- **Операционная система** — как новый продукт будет взаимодействовать с существующей операционной системой? Что произойдет при появлении следующих версий операционной системы? Как изменения в операционной системе отразятся на архитектуре продукта? Как можно использовать и поддержать возможности новой операционной системы?
- **График** — какие еще продукты, системы или приложения создаются в организации в то же время?

Шаг 2: анализ

Основные требования, собранные на первом этапе, ни в коем случае не являются точными. Один из способов структурирования информации — сгруппировать ее в упрощенные утверждения или вопросы с ответами. Например, «Консультационной группе приходится вводить почасовые сведения в трех местах» и «Консультанты хотели бы делать это только один раз». Часто один параметр этого «уравнения» уже есть, и группе нужно выявить второй. Концепции, выписанные с одной стороны листа, отражают текущее состояние, а перечисленные с другой — будущее состояние. Учет стратегических целей организации также помогает сформулировать требования к проекту.

В главе 6 мы обсудим применение концепции «существительное — глагол» к логическому и физическому проектированию. Взгляд на ключевые существительные и глаголы также может быть полезен и при создании концепции проекта. Если организаторы проекта часто используют одни и те же существительные и глаголы, когда говорят о проекте, эти слова, скорее всего, важны для проекта.

Информация делового характера и информация, поступающая от пользователей, поможет создать перечень характеристик продукта. Как и в случае с требованиями, полученные таким образом характеристики на этом этапе не следует распределять по версиям продукта — сейчас важно собрать по возможности полный список. Выделение из этого перечня характеристик, которые актуальны для текущего и будущих выпусков продукта — задача последующих этапов.

При документировании требований иногда очень полезны диаграммы схем использования и процессов универсального языка моделирования — они наглядно представляют информацию. Модель схем использования позволит описать требования на языке пользователей. Кроме того, полезно и простое словесное описание. Завершение этого шага — создание исчерпывающего перечня схем использования, которые будут положены в основу всего процесса разработки.

Шаг 3: рационализация

Когда составлен перечень схем использования и соответствующих характеристик продукта, надо сгруппировать подобные по своей природе характеристики в сегменты, или *наборы характеристик*. Основой для группирования могут служить действия пользователей, работа приложения, использование данных и т. д. В этот момент группа еще работает с теми характеристиками, которые описывают как текущее, так и будущее состояние. Для описания текущего состояния процессов группа может применять и другие методы (например, образцы проектов).

Шаг 4: реализация

Когда изучение, анализ и рационализация выполнены, соответствующие прототипы созданы, а вся собранная информация структурирована, накапливается пакет документов для создания концепции. На этапе изучения задача группы заключается не в анализе информации, а в ее сборе — это лишь первое знакомство. На этапе анализа собранная информация уточняется и структурируется в виде требований и схем использования, что служит основой для создания предварительного списка характеристик продукта. На этапе рационализации характеристики группируются в пакеты. На заключительном этапе решается, какие пакеты характеристик будут реализованы в текущей версии продукта.

В этот момент группа определяет приоритеты пакетов функциональных возможностей продукта. Схемы использования, диаграммы методов использования, диаграммы видов деятельности и сценарии использования, описывающие текущий выпуск продукта, объединяются, после чего определяется предварительная концепция проекта, Она содержит долгосрочные цели и служит подтверждением того, что текущая версия продукта — это движение в верном направлении. На этом этапе собрано достаточно информации, чтобы все стороны компромиссного треугольника — ресурсы, характеристики и затраты времени — были оценены, оптимизированы и утверждены. На этом этапе создается и предварительный график проекта, который будет направлять работу на фазе планирования.

Шаг 5: утверждение

Как мы уже говорили, предварительный вариант концепции создается на этапе реализации. Сейчас группа еще раз рассматривает и корректирует этот документ. Этап утверждения фазы «Анализ» — это момент, когда члены группы должны взглянуть назад и оценить, каков прогресс и каковы результаты их работы, чтобы убедиться, что они готовы к этапу «Одобрение концепции».

Обмен информацией

Цель исследования заключается не в создании документов, а в обсуждении разных точек зрения проектной группой, а также группой и другими участниками проекта. Прототипы могут стать отличным способом обмена информацией. В основе концепции создания прототипов — известная пословица «Лучше один раз увидеть, чем сто раз услышать». Уровень детализации в прототипе обычно очень невелик. Это может быть просто набор экранов или примеры типичных страниц. Однако с помощью прототипа гораздо легче объяснить пользователям, как будет работать приложение и какие функции в нем будут реализованы.

Внимание! Не следует ожидать использования значительной части кода прототипа на следующих стадиях проекта. Такая постановка задачи часто ухудшает качество кода и тормозит процесс разработки приложения.

Управление рисками

Ключевой фактор успеха разработки любого приложения — это идентификация рисков, возможных в проекте, и разработка системы управления этими рисками, которая должна работать в течение всего жизненного цикла проекта. Мы начинаем обсуждение управления рисками в этой главе по той причине, что эта работа начинается на стадии «Анализ», однако процесс управления рисками продолжается все время выполнения проекта.

Многие профессионалы в области информационных технологий имеют неверное представление об управлении рисками. В лучшем случае они считают, что это необходимая, но скучная работа, которую надо выполнить в начале проекта, прежде чем начнется настоящая работа — написание кода. В худшем случае управление рисками считается одной из форм бюрократического издевательства, которая препятствует организации в достижении ее целей. Каждый проект связан с риском, и на этот риск приходится идти. Однако это не значит, что попытка выявления рисков и управления ими бесполезна или что она будет «душить» творчество.

Риск— это *возможность потерь.* Для конкретного проекта это может быть:

- снижение качества продукта;
- увеличение расходов;
- срыв сроков;
- невозможность достижения целей проекта.

Поскольку риск — это потенциальная проблема, еще не возникшая в действительности, эффективное управление рисками — динамический процесс.

Члены группы обычно знают, какие риски связаны с их проектом, но часто забывают обмениваться этой информацией. Обычно продвижение информации о рисках вниз по служебной иерархии происходит без труда; проблема заключается в том, чтобы обеспечить движение этой информации в обратном направлении. На каждом уровне иерархии хотят знать риски, выявленные на нижних уровнях, но никто не торопится сообщать о выявленных рисках на вышележащие уровни. Для успеха управления рисками организация должна создать такую обстановку, в которой люди, выявившие риски проекта и со-

общившие о них, будут защищены от неприятностей. Только в обстановке, где «за дурные вести не убивают», члены группы чувствуют себя достаточно свободно, чтобы высказывать самые смелые предположения и взгляды, которые значительно расширяют рамки и качество управления рисками.

Иногда сообщения о новых рисках рассматриваются как форма жалобы или источник неприятностей. Часто сообщение о риске с большей готовностью воспринимается начальством как свидетельство проблем у сообщившего о нем сотрудника, чем как проблема продукта. Если в организации действует неписаное правило «замалчивать риски», то жизненно важная информация, которая могла бы способствовать снижению риска, будет утрачена, а соответствующий план действий никогда не будет выработан.

Внимание! Помните, что риск есть возможность потерь, а не их гарантия. Члены группы могут с незаслуженным скепсисом воспринимать проект с перечнем из 15-20 рисков, забывая о том, что общая оценка вероятности этих рисков невелика.

Источники риска

Для эффективного управления рисками необходимо принимать во внимание бизнес-среду, в которой будет выполняться проект. Многие проекты в области информационных технологий проваливались не из-за неудачной технологии или плохого управления, а потому что игнорировались важные деловые и организационные факторы: конкуренция, финансовое состояние и корпоративная культура. Потенциальными источниками риска считаются:

- задачи и цели;
- лица, принимающие решения;
- методы управления организацией;
- заказчики и пользователи;
- бюджет и расходы;
- график;
- характеристики проекта;
- процесс разработки;
- среда разработки;
- персонал;
- эксплуатационная среда;
- новые технологии.

К возможным последствиям проекта относят:

- превышениезапланированных расходов;
- несоблюдение графика;

- неадекватные функциональные возможности продукта;
- аннулирование проекта;
- неожиданная смена персонала;
- недовольство заказчика;
- ущерб престижу организации;
- деморализация персонала;
- низкая производительность продукта;
- судебные иски.

Важно помнить, что разным типам проектов присущи разные риски и что каждый их них необходимо рассматривать отдельно.

Способы управления рисками

Возможны три способа управления рисками.

- **Размахивание «волшебной палочкой»** — проектная группа оценивает риски лишь однажды, на начальном этапе. Основные риски идентифицируются, но больше никогда всерьез не рассматриваются, а соответствующие действия не предпринимаются,
- **Быстрое реагирование** — проектная группа реагирует на последствия риска, когда проблема уже возникла.
- **Превентивный** — проектная группа управляет рисками, постоянно отслеживая условия их реализации.

Ясно, что «волшебная палочка» — не пример удачного управления рисками. Быстрое реагирование тоже не приведет к успеху, потому что оно не предполагает мер предотвращения рисков. Идентификация рисков и их предупреждение — признак хорошо организованного процесса управления рисками. Превентивные меры начинаются на стадии «Анализ» и продолжаются до выпуска продукта, то есть до конца фазы «Стабилизация». Применительно к большинству рисков цель проектной группы состоит в том, чтобы предупредить их реализацию, а также сформулировать перечень действий, которые следует предпринять, если проблема все-таки возникла. Чтобы достичь высокого уровня превентивного управления рисками, проектная группа должна уметь беспристрастно оценивать риски и предпринимать действия по устранению их причин, а не симптомов.

Важно отметить, что не имеет решающего значения, насколько хорошо группа работает над оценкой рисков, — успех проекта определяется управлением рисками.

Процесс управления рисками состоит из превентивных решений и действий:

- выявления рисков;
- выявления рисков, требующих вмешательства;
- разработки стратегии снижения рисков.

Эффективная проектная группа оценивает риски в течение всего жизненного цикла проекта и использует эти оценки в процессе принятия решений на всех фазах проекта. Как показано на рис. 5.2, группа отслеживает риск до тех пор, пока он не исчезнет или не превратится в проблему, которую придется устранять.

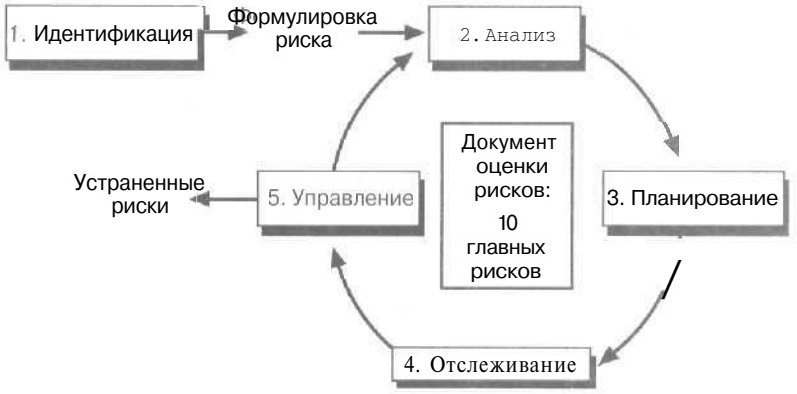


Рис. 5.2. Превентивное управление рисками

Шаг 1: идентификация риска

Идентификация риска — первый этап превентивного управления рисками. Риски сначала должны быть идентифицированы, и только после этого ими можно будет управлять. Идентификация рисков предоставляет в распоряжение проектной группы информацию, которая позволяет выявить основные риски, способные негативно повлиять на проект.

Чтобы выявить риски и проранжировать их, члены группы и другие участники проекта проводят серию мозговых штурмов и последующее обсуждение. Чтобы упростить этот процесс, факторы риска стоит сгруппировать по областям возникновения, таким как разработка специализированного программного обеспечения. развертывание инфраструктуры, развертывание программного обеспечения, планирование производственной архитектуры и компонентная разработка. В рамках каждой из этих областей факторы риска могут быть также сгруппированы по таким категориям, как цели и задачи, руководство, менеджмент, бюджет и т. д.

Например, в табл. 5.2 представлены три фактора риска, относящиеся к целям проекта. Для каждого фактора рассмотрены высокая, средняя и низкая вероятность возникновения риска. Если риск имеет низкую вероятность возникновения, проект сможет выполнить

поставленные перед ним задачи; в противном случае — если фактор имеет высокую вероятность возникновения — проект не решит поставленные задачи.

Табл. 5.2. **Образец оценки** факторов **риска**

Фактор риска	Низкая вероятность возникновения	Средняя вероятность возникновения	Высокая вероятность возникновения
Соответствие проекта поставленным целям	Полностью соответствует	Риск угрожает достижению одной или нескольких целей проекта	Проект не выполняет поставленных задач
Мнение заказчика	Ожидает, что группа создаст заказанный продукт	Считает, что группа не работает над нужным продуктом	Считает, что группа не может создать нужный продукт
Руководство	Не влияет на проект	Изменяет отдельные особенности управления проектом	Значительно изменяет управление проектом или организацию работ

Для всех рисков, обнаруженных в ходе рассмотрения списка факторов риска, необходимо занести в сводный перечень *формулировку риска* (постановку проблемы). Риск должен быть ясно сформулирован, прежде чем им можно будет управлять. При описании риска группа должна рассматривать не только симптомы, но и причины и последствия. Как показано на рис. 5.3, каждая формулировка риска должна включать проблему (условие), причину проблемы и возможные последствия — как для проблемы, так и для проекта.

Рис. 5.3. Пример формулировки риска

Процесс формулировки рисков — мощный метод выявления различий во взглядах членов группы и других участников проекта. Маловероятно, что все будут согласны с принятым ранжированием факторов риска. В зависимости от своего опыта и области интересов разные члены группы по-разному видят проект. Если согласие не достигается, лучше всего решить вопрос голосованием. Если голоса разде-

лятся поровну, из соображений осторожности следует выбрать пессимистический вариант оценки риска.

Шаг 2: анализ риска

Анализ риска — это процесс, в ходе которого данные о риске превращаются в информацию для принятия решений. Только всесторонний анализ гарантирует, что группа будет работать с правильно описанными рисками.

Риск характеризуется главным образом двумя факторами.

- **Вероятность риска** — это вероятность того, что событие действительно произойдет. Для оценки ее величины можно использовать простую процентную шкалу (0-100). Только риск с оценкой вероятности больше 0% представляет опасность для проекта. Риски с вероятностью 100% уже реализовались; другими словами, это известные проблемы. Группа может найти более эффективным такой подход, при котором используются от 1 до 3 точек на этой шкале, соответствующие 25, 50 и 75%, потому что иногда несущественное различие в оценках вероятности — например, 60% или 70% — вызывает лишние споры.
- **Влияние риска** — влияние риска отражает степень важности неблагоприятных последствий реализации риска. Измерить эти потери непросто. Если риск имеет финансовый характер, группа может оценивать потери в денежном выражении: это долгосрочные затраты на эксплуатацию и сопровождение, потеря части рынка, краткосрочные затраты на дополнительную работу или упущенные возможности. Когда финансовые последствия оценены, их величину можно классифицировать по пятибалльной шкале, где 5 будет означать самый большие потери. Иногда риски таковы, что их последствия удается оценить только субъективно (скажем, от I до 5). Эта шкала, по существу, оценивает шансы на успех проекта: высокое значение указывает на серьезные потери для проекта, а небольшое соответствует минимальному влиянию на проект.

Чтобы оценить перечень рисков, группа должна полностью осознавать ту опасность, которую несет проекту каждый риск. Иногда у риска с высокой вероятностью столь незначительные последствия, что его можно спокойно проигнорировать. С другой стороны, риски, влияние которых существенно, могут считаться маловероятными, и о них тоже можно забыть. Управлять стоит только рисками, имеющими высокую вероятность и значительные последствия. Снижение риска достигается как за счет уменьшения вероятности его возникновения, так и за счет снижения важности его последствий,

Чтобы выявить риски, рекомендуется распространить среди участников проекта список рисков и предложить его заполнить. В список можно включить следующие пункты;

- **идентификатор** — название, уникально идентифицирующее формулировку риска (идентификаторы понадобятся для отслеживания и отчетности);
- **источник** — источник может быть идентифицирован по предметной области (разработка программного обеспечения, инфраструктура развертывания и т. п.), по категории (задачи и цели, руководство, менеджмент и т. д.) или по фактору (соответствие проекта требованиям заказчика. стабильность организации и т. п.);
- **условия возникновения** — описание условий, в которых риск может реализоваться и поставить проект под удар;
- **вероятность** — оценка вероятности реализации риска. Вероятность обычно выражается значениями от 1 до 3, представляющими величины в процентах (25. 50 и 75% соответственно);
- **последствия** — оценка последствий реализации риска для проекта; может быть выражена в денежном эквиваленте или числом по 5-балльной шкале, которое показывает относительную важность этой проблемы;
- **влияние** — сводная количественная оценка риска, равная произведению его вероятности на количественную оценку его последствий. Заметим, что для ранжирования все оценки влияния рисков надо выражать в одних и тех же единицах;
- **контекст** — дополнительная информация, проясняющая ситуацию, в которой риск может реализоваться;
- **связанные риски** — перечень идентификаторов рисков, связанных с данным; этот список полезен для выявления связанных и независимых рисков.

Проранжировав риски по степени влияния, группа может заняться стратегией управления рисками и включением планов действий на случай реализации рисков в концепцию проекта. Поскольку управление рисками требует времени и усилий, ключевым моментом является идентификация ограниченного числа основных рисков, которые надо постоянно отслеживать. Простой, но эффективный прием — выявить десять основных рисков проекта. Эту «горячую десятку» должны одобрить все участники проекта. Дополнительный перечень важных рисков нужно включить в концепцию проекта (документ, созданный на стадии «Анализ») и в основной план проекта, который создается на стадии планирования.

Шаг 3: план действий

На этом шаге информация о риске превращается в план действий. На стадии планирования вырабатываются действия, которые необходимо предпринять в отношении отдельных рисков, определяются приоритеты этих действий и создается обший план управления рисками, который образует основу сводного документа оценки рисков, являющегося обязательным результатом этапа «Одобрение концепции».

Планируя действия при возникновении риска, группа должна рассматривать каждый идентифицированный риск с четырех сторон.

- **Исследование** — достаточно ли информации об этом риске? Требуется ли дальнейшее изучение ситуации, чтобы собрать больше информации и прояснить все характеристики риска прежде, чем вырабатывать план действий?
- **Приемлемость** — может ли группа игнорировать последствия риска и не предпринимать никаких действий?
- **Управление** — может ли группа сделать что-нибудь, чтобы снизить воздействие риска, если он реализуется?
- **Возможность избежать проблемы** — можно ли избежать риска, изменив продукт?

После выявления рисков, нуждающихся в реагировании, группа получает три возможности:

- снизить вероятность возникновения риска;
- уменьшить размеры потерь;
- изменить последствия риска.

Чтобы снизить риски, управление которыми находится в компетенции группы, надо применить ресурсы, необходимые для этого. Если же риск находится вне компетенции группы, стоит поискать обходные пути. Эффективными могут оказаться:

- переход на другое аппаратное обеспечение;
- перенос части функциональных возможностей продукта в другую систему, которая лучше приспособлена для решения этих задач;
- передача контракта более квалифицированному подрядчику.

Чтобы гарантировать надежность выполнения проекта, группа должна предусмотреть *стратегию на случай чрезвычайных обстоятельств* — план ликвидации последствий, который должен быть задействован в случае, если группа не справится с риском. Например, предположим, что вам необходим специализированный программный продукт, который позволит развернуть программное обеспечение на настольных системах, но дата его выпуска не определена. Стратегия группы на этот случай может предусматривать использование альтернативного продукта. В этом случае на стадии планирования группе следу-

ет подобрать альтернативный продукт, чтобы использовать его, если оригинальный продукт не будет готов к моменту выпуску продукта.

Решение о том, в какой момент обращаться к чрезвычайной стратегии, — это вопрос определения «порогового значения». Часто группа может установить «пороговые значения» для каждого чрезвычайного плана, основываясь на типе риска или виде последствий. В табл. 5.3 приведены примеры рисков, их последствия и механизмы для запуска плана действий в чрезвычайной ситуации. Когда «пороговое значение» превышено, приводится в действие соответствующая стратегия, уменьшающая последствия риска.

Табл. 5.3. Примеры рисков и «пороговых значений»

Тип риска	Пороговое значение
Нарушение графика	Последняя дата обращения к чрезвычайной стратегии Последняя возможная дата выбора другого поставщика
Необходимость дополнительных ресурсов	Последняя дата, когда остается время на поиск ресурсов Самая крупная сумма штрафа или пени Самая большая величина возможного перерасхода
Дополнительные расходы для заказчика	Лимит средств
Время на обучение	Лимит времени

Для создания сводного документа оценки рисков необходимо идентифицировать несколько ключевых пунктов для каждого риска. Они обычно вводятся в автоматизированную форму, описывающую действия в случае риска, чтобы облегчить обмен информацией между членами группы и руководством:

- **идентификатор риска** — уникальное имя, которое идентифицирует формулировку риска и используется для отчетов и отслеживания;
- **формулировка риска** — описание условий реализации риска, а также потерь, которые возможны в этом случае;
- **стратегия управления риском** — описание стратегии управления риском, включая все допущения;
- **метрики стратегии управления рисками** — параметры, позволяющие оценить, работает ли стратегия:
  - **вероятность** — оценка вероятности реализации риска; обычно выражается значениями от 1 до 3, представляющими величины в процентах (25, 50 и 75% соответственно):

- **последствия** — оценка последствий реализации риска для проекта; может выражаться в денежном эквиваленте или числом по 5-балльной шкале, которое показывает относительную важность этой проблемы;
- **влияние** — сводная количественная оценка риска, равная произведению его вероятности на количественную оценку его последствий;
- **действия — действия** по управлению риском; все предпринятые действия должны регистрироваться системой отслеживания рисков;
- **сроки** — даты, к которым группа должны закончить выполнение этапов;
- **ответственные** — лица, ответственные за перечисленные действия;
- **чрезвычайная стратегия** — краткое описание плана действий на случай, если намеченный план не справится с риском;
- **«пороговые значения» и параметры чрезвычайных стратегий** — служат для принятия решения о запуске чрезвычайной стратегии и оценки ее эффективности.

Шаг 4: отслеживание риска

Отслеживание риска — важная составляющая эффективного управления рисками. Это процесс, в ходе которого проектная группа контролирует статус рисков и последствия действий, предпринятых для их снижения. Этот процесс включает в себя определение параметров и разработку пороговых значений, которые нужны, чтобы убедиться, что запланированные на случай риска действия работают. Отслеживание — это «сторожевой пес» управления рисками.

Замечание Рекомендуется включать перечень рисков в регулярные обзоры проекта. Этот документ должен содержать анализ влияния десяти главных рисков проекта.

При каждом обзоре проекта группа анализирует главные риски проекта и статус всех действий по управлению ими. Полезны также регулярное ранжирование рисков и сведения о числе появлений риска в десятке главных рисков.

Отчет о статусе риска может идентифицировать четыре возможных ситуации в управлении риском:

- риск устранен, план действий выполнен;
- действия происходят по плану, реализация плана продолжается;
- действия идут не по плану, следует принять корректирующие меры или ввести в действие чрезвычайный план;
- ситуация существенно изменилась, необходим пересмотр плана действий и стратегии.

По мере управления рисками общее влияние рисков на проект должно постоянно снижаться.

Шаг 5: управление рисками

После того как группа подобрала спусковые механизмы и параметры, самая трудная часть управления рисками завершена. Теперь управление рисками становится одной из составляющих управления проектом и включает:

- контроль планов действий на случай реализации риска;
- корректировку отклонения от плана;
- реакдию на достижение «пороговых значений»;
- совершенствование процесса управления рисками,

Самое главное — никогда не забывать об управлении рисками. Иначе, как отметили Рон Игуэйра (Ron Higuera) и Яков Хаймс (Yacov Haimes) в докладе, посвященном управлению рисками в проектах разработки программного обеспечения, *«...если процесс управления рисками не интегрирован в ежедневную практику управления проектом, он будет вскоре низведен до уровня второстепенной деятельности»*.

Этап «Одобрение концепции» и его результаты

Цель фазы «Анализ» — достижение этапа «Одобрение концепции», который является кульминацией работы группы в течение этой фазы. На этом этапе заказчик и проектная группа согласуют основные вопросы проекта.

Для достижения этого этапа необходимы четыре документа:

- «Концепция», где описан разрабатываемый продукт, решаемые им задачи, его характеристики и предварительный график;
- «Структура проекта», где описаны распределение ролей в проектной группе и области ответственности каждой роли;
- «Основной документ оценки рисков», где перечислены риски проекта и описаны методы управления ими.

Мы также рекомендуем включить в список обязательных результатов прототип приложения, который послужит иллюстрацией плана разработки приложения и визуальной демонстрацией концепции.

Замечание Цель этих документов — обеспечить эффективный обмен информацией. В этом контексте результат не обязательно должен быть зафиксирован на бумаге — он может иметь любую форму, обеспечивающую эффективный обмен информацией (документ, диаграмма, экран, электронная почта и т. д.).

Все основные этапы модели процесса разработки MSF означают достижение согласия между заказчиком, проектной группой и остальными ключевыми участниками проекта. Достижение этапа «Одобрение концепции» означает достижение согласия по вопросам, перечисленным в табл. 5.4.

Табл. 5.4. Пункты, которые необходимо согласовать на этапе «Одобрение концепции»

Пункты соглашения	Где описаны
Бизнес-задачи, которые решает проект	Концепция
Концепция проекта	Концепция
Цели разработки	Концепция
Проектная группа	Структура проекта
Первоначальная концепция бизнес-решения	Концепция и прототип
Риски, связанные с проектом	Пересмотренный документ оценки рисков

Этап «Одобрение концепции» предоставляет заказчикам и проектной команде достаточно информации для принятия важного решения — следует ли продолжать выполнение проекта. В принципе, после рассмотрения результатов фазы «Анализ» и концепции продукта заказчик и проектная группа могут решить, что он не оправдает расходов. Это очень важный момент в жизни проекта. Хорошо проделанная работа на стадии «Анализ» позволит проектной группе продолжить движение вперед с уверенностью в успехе.

Концепция

Документ, описывающий концепцию, должен включать, по крайней мере:

- формулировку концепции;
- результаты исследования требований пользователей;
- информацию о конкурентоспособности решения;
- описание функциональных возможностей;
- приблизительный график.

Формулировка концепции

В 60-е годы, когда США пытались отправить человека на Луну, президент Джон Ф. Кеннеди сформулировал постулат, который сплотил всю страну;

«б *следующем десятилетии мы пошлем человека на Луну и благополучно вернем его на Землю*».

Это высказывание иллюстрирует принципы формулирования концепции:

- конкретность;
- измеримость;
- достижимость;
- важность;
- наличие четко указанных сроков.

Чтобы вернуть обсуждение на Землю, приведем другой пример основательной формулировки концепции:

«Быстро создать самую быстродействующую электронную таблицу на планете».

Познакомившись с такой формулировкой, каждый член проектной группы немедленно понял бы, что достижение результата важнее, чем любой другой фактор. Краткая формулировка концепции должна отвечать следующему правилу: она должна быть настолько ясной, чтобы новый сотрудник, руководствуясь ей, смог бы работать над проектом не хуже ветеранов.

Результаты исследования требований **пользователей**

Информация, полученная от заказчика и пользователей, является обоснованием всего, что делает проектная группа. Просмотры Web-узлов, контекстные исследования, анализ рынка, фокус-группы, международные визиты помогают собрать информацию, необходимую для того, чтобы описать нужды заказчика и пользователей. Кроме того, эти исследования являются основой для создания сценариев, описывающих, как пользователи работают сейчас и как они будут работать в будущем, используя новый продукт.

Конкурентоспособность

Обеспечит ли продукт окупаемость инвестиций в существующей конкурентной среде как сегодня, так и завтра? Какие еще решения возможны для удовлетворения потребностей заказчиков? Группа должна проанализировать ситуацию и убедиться в том, что разрабатываемый продукт решает проблемы заказчика лучше, чем другие доступные приложения, и что это именно то направление, которое оправдывает инвестиции. Лучшее решение проблемы — не обязательно программный продукт. Например, иногда проще и эффективнее с точки зрения затрат нанять сотрудника, например, бухгалтера, взамен использования финансового программного обеспечения. Даже если программное решение соответствует потребностям, нужны веские основания, чтобы начать разработку «с нуля», а не воспользоваться

существующими программами. Для обоснования необходимости разработки следует провести анализ других систем, реализующих сходные функции. Даже если принято решение не покупать имеющийся на рынке программный продукт, а разрабатывать новый, полученная в ходе такого анализа информация расширит кругозор группы и позволит ей создать более мощный продукт.

Иногда проект конкурирует с потребностями других ресурсов организации. Например, в конце 90-х годов многие компании направили значительную часть своего бюджета на решение проблемы 2000 года, что серьезно сказалось на других проектах.

Наборы функциональных возможностей

Наборы — это категории, по которым распределяются все характеристики продукта. Менеджер программы использует эти категории, чтобы определить соответствие каждой характеристики целям продукта. Если характеристика не вписывается ни в один из классов, она, вероятно, не так важна для текущего выпуска.

Распределение функциональных возможностей по наборам должно учитывать фактор выпуска версий. Считается, что в каждом выпуске продукта может быть реализовано не более трех классов характеристик. Даже если это и так, совсем неплохо создать большее число наборов с тем, чтобы предусмотреть и текущие, и будущие нужды проекта. Такая практика создает фундамент для определения функциональных возможностей, которые предстоит реализовать в следующих версиях продукта.

Приоритеты и график

По завершении этапов изучения, анализа и рационализации фазы «Анализ» группа примерно представляет себе, как будет выглядеть график и какие компромиссы понадобятся, чтобы соблюсти его. Кроме того, группа подобрала подходящий компромиссный треугольник для проекта и будет использовать его для оптимизации ресурсов, функциональных возможностей и сроков сдачи продукта. Включение этой информации в описание концепции показывает, что группа думает о реальных условиях разработки; кроме того, на ее основе начинается структурирование других элементов концепции.

Прототип

Организовать четкий и ясный обмен информацией с заказчиком и пользователями — дело непростое. Проектную группу необходимо проинформировать о бизнес-проблемах, а информацию о том, как продукт решит эти проблемы, надо довести до заказчика и пользователей.

Существенно облегчит такую связь прототип приложения, который полностью или частично демонстрирует концепцию продукта.

Его разрабатывают в ходе фазы «Анализ» и включают в результаты этапа «Одобрение концепции». Прототип может представлять собой как работающее приложение, так и просто набор экранов. Прототип помогает разъяснить концепцию и выявить дополнительные вопросы группы, заказчика и пользователей продукта.

Структура проекта

Концепция описывает, что именно будет сделано, а структура проекта — кто будет это делать. Этот документ определяет:

- каждую роль в группе;
- кто отвечает за каждую роль;
- состав групп и контактную информацию;
- организаторов проекта:
- методы управления проектом — например, системы контроля и виды отчетности;
- связанные с проектом графики, адреса электронной почты и Web-узлы.

Также полезно включить в описание структуры краткую информацию о лидерах группы, их опыте и контактную информацию для заинтересованных лиц, таких как заказчик и другие ключевые участники проекта.

Сводный документ оценки рисков

Важным результатом этапа «Одобрение концепции» является сводный документ оценки рисков, который содержит перечень основных рисков, их оценки и планы управления ими. В отчете классифицированы основные риски и приведены их описания. Эта информация собирается на этапе планирования процесса управления рисками MSF, который мы описали в этой главе. Обычно в этот документ включают список десяти основных рисков — это полезно как для информирования управленческого персонала, так и для повышения внимания группы к управлению рисками. Список содержит основные риски, планы управления ими и ответственных за осуществление этих планов. Наконец, простая схема рисков с их управленческими метриками помогает быстро разобраться в том, какие риски имеют самую высокую вероятность и самое сильное влияние на проект,

Согласование концепции

Когда группа создала первый вариант концепции, наступает время ее согласования с основными участниками проекта. Этот процесс состоит из двух этапов.

- **Проверка концепции** — документы следует обсудить со всеми участниками проекта. Это время сбора мнений, внесения поправок и дополнений по вопросам, не отраженным в концепции.
- **Реклама концепции** — концепцию изучают все ключевые участники проекта, чтобы понять, почему продукт важен и чем он может быть им полезен. Кроме того, они привыкают к мысли, что в продукт войдут только функциональные возможности, отраженные в концепции. Это хорошая возможность потренироваться в принятии компромиссных решений, ранжировании наборов функциональных возможностей и планировании версий продукта. Обсуждение следует начинать с участников, обладающих максимумом компетенции в области проекта, и лишь потом привлекать всех остальных.

Внимание! Обсуждение — это улица с двусторонним движением. Группа должна учитывать мнение всех заинтересованных сторон и адекватно отразить его в следующих версиях концепции, Члены группы должны не только сообщить участникам проекта, что именно будет делать продукт, но и постоянно обосновывать соответствие продукта ожиданиям заказчика.

Первый вариант концепции пересматривается и уточняется. После того как концепция продукта согласована с заказчиком и одобрена, группа может переходить к фазе «Планирование». На этом этапе фиксируется базовая концепция, которая будет использоваться при разработке этого выпуска продукта. Хотя базовая концепция может изменяться в ходе следующих стадий проекта, она, тем не менее, составляет основу этого выпуска продукта.

На этой стадии важно помнить, что концепция не является чьей-то личной собственностью — это общее мнение проектной группы. Только так можно сформировать жизнеспособную концепцию, и вся группа должна хорошо это понимать и поддерживать этот подход.

Другие области применения аналитического подхода

Фаза «Анализ» приносит ощутимые результаты, поэтому стоит попытаться применить этот подход ко всей организации в целом. Однако для большинства организаций масштабное использование аналитического подхода требует существенных изменений в культуре и методах работы. Мы рекомендуем делать это постепенно. Через какое-то время, используя не только аналитический подход, но и всю модель процесса разработки MSF, удастся значительно повысить шансы на успех любого проекта, будь то разработка приложений или что-то другое. Приведем примеры соответствующих действий.

- Попробуйте применить принципы фазы «Анализ» в одном проекте, выбранном для внедрения MSF и модели процесса разработки MSF в организации.
- Создайте концептуальный документ и попробуйте использовать его как механизм обмена информацией и согласования концепции.
- Оказывайте всяческое содействие встречам и совещаниям, на которых обсуждается модель процесса разработки MSF в общем и фаза «Анализ» в частности.
- Сделайте видеосъемку обсуждения концепции проекта и используйте ее как учебный материал.
- Накапливайте удачные методы анализа, применяющиеся в проектах, и поощряйте их применение в будущих проектах.

Напомним еще раз, что главная цель фазы «Анализ» — добиться единого понимания того, какой продукт нужен и как его сделать. Понимание этого факта и последовательное применение аналитического подхода повысит эффективность всех проектов, выполняющихся в вашей организации, и увеличит шансы на успех каждого конкретного проекта.

Резюме

Цель стадии «Анализ» — обретение ясного понимания бизнес-задач, стоящих перед проектом разработки программного обеспечения, и того, как создать приложение, решающее эти задачи.

Создав концепцию, обосновывающую проект и описывающую его цели, ограничения и риски, проектная группа достигает этапа «Одобрение концепции», которым завершается фаза «Анализ» модели процесса разработки MSF. Концепция позволяет:

- объединить усилия всех членов;
- упростить процесс принятия решений;
- гарантировать согласованность решений;
- добиться мотивированности группы;
- гарантировать качество продукта;
- оценить эффективность выполнения проекта.

Помимо концепции, на этом этапе группа может создать прототип приложения, который помогает:

- добиться понимания основных характеристик продукта далекими от технологии участниками проекта;
- визуально продемонстрировать концепцию продукта.

Четкое распределение обязанностей позволяет гарантировать учет всех факторов, влияющих на успех проекта. Чтобы достичь этой цели, документ «Структура проекта» должен описывать;

- руководителей шести ролей модели проектной группы MSF;
- кадровые требования для всех шести ролей.

На этой стадии начинается процесс управления рисками, основные характеристики которого описываются в сводном документе оценки рисков. Этот документ:

- идентифицирует наиболее серьезные риски;
- инициирует процесс управления рисками.

Кульминация фазы «Анализ» — одобрение концепции, знаменующее ее понимание всеми участниками проекта и достижение согласия между заказчиком, проектной группой и остальными ключевыми участниками проекта.

Закрепление материала

1. Каковы основные цели фазы «Анализ»?
2. Какие роли отвечают за достижение целей фазы «Анализ»?
3. Каковы основные компоненты концепции?
4. В чем преимущество использования прототипов?
5. Что такое риск?
6. Что такое управление рисками?

Практикум 5, Концепция RMS

Все начиналось хорошо. Первая встреча группы RMS, которая была посвящена постановке задачи, прошла успешно. Это было в понедельник — они встречались тогда в дубовом зале и действительно продвинулись в реализации модели процесса проектирования MSF. Даже пончики Тима не смогли сбить их с пути. Поскольку разработку проекта предполагалось провести в четыре фазы, решили, кто будет отвечать за каждую фазу, в деталях обсудили предварительный график работ, составленный Дэном. В общем, сделали шаг вперед. Когда встреча близилась к концу, Дэн завел разговор об итерационном подходе и выпуске версий. Он и предположить не мог, что такая простая вещь вызовет столько проблем.

Первый раунд

В среду встреча и началась-то плохо, а продолжилась еще хуже. Во-первых, Тим опоздал. Собственно говоря, ничего особенного в этом не было — Тим и прежде регулярно опаздывал, однако теперь Дэн «работал над этой проблемой» и, казалось, немало преуспел. Тим стал дисциплинированнее, и Дэн имел основание собой гордиться. Но в среду Тим не появился в 8 утра, Дэн решил немного подождать, надеясь, что он вот-вот придет. Через некоторое время стало ясно, что

Тим не появится, и Дэн послал Джейн позвонить ему. Она вернулась через пару минут и сообщила: «Я застала его дома. Он просто проспал, ничего особенного. Сказал, что сейчас прибежит».

Фергюсон и Барделл

Строительство • *Проектирование* • *Управлениепроектами*

Проект создания системы управления ресурсами

Повестка дня

Дата: 31 марта 1999 г._____Тема: исследование 1

I. Уточнение повестки

II. Цели итерации

III. Результаты исследования

IV. Компромиссный треугольник

V. Концепция проекта — предварительный вариант

VI. Вопросы и ответы

VII. Заданий и ответственные

Чикаго• Детройт• Милуоки• Цинцинати • Индианаполис- Луисвилль

– Тогда давайте начинать. Он нас догонит, — Дэн был огорчен и немного сердит, но старался этого не показывать. По крайней мере, Мэри-Лу не забыла про пончики. Впрочем, наслаждаться ими оказалось некому, так что Мэри-Лу тоже казалась слегка обиженной.

Дэн заглянул в повестку дня:

– Хорошо, друзья, как обычно для начала обсудим повестку дня, Кто-нибудь хочет добавить или изменить что-нибудь в повестке дня, сделать уточнения?

Марта подняла руку.

— Да, Марта?

Она встала, и Дэн заметил, что у нее немного дрожали руки, когда она оперлась о стол. «Ой-ей-ей», — подумал он. Она казалась расстроенной еще в понедельник в конце встречи и он надеялся, что она придет к нему поговорить, но она не пришла. «Интересно, — подумал он, — поделится ли она сейчас тем, что ее беспокоит».

— Я прочитала материал, который ты дал нам. Дэн, и выслушала, что ты сказал по этому поводу. Мне нравится модель проектной группы и мне приятно работать над таким проектом. Когда мы говорили о подоплеке RMS, я поняла, что это будет не только очень интересная, но и важная работа.

Но на последней встрече, когда мы прорабатывали модель процесса, у меня появились сомнения. Концепция итерационного подхода к планированию мне просто непонятна. Как что-нибудь создавать, когда планы не до конца определены? Ты описал модель водопада и сказал, что MSF — более эффективный метод, но я подумала, что в течение столетий все здания и мосты сооружались именно по модели водопада. Я инженер по образованию, у меня уже есть сложившиеся представления; я обычно сначала делаю общие планы, затем более подробные, детальные чертежи и так далее — до сборочных чертежей. Только когда *все* чертежи подготовлены и одобрены, можно начинать строительство моста или какого-то другого объекта. Как могут разработчики программного обеспечения действовать иначе?

Должна признаться, что теперь я не доверяю этому процессу. Я прочитала документы, относящиеся к прошлому проекту, и я думаю, вам просто повезло, что все получилось так удачно.

Марта сделала паузу, затем глубоко вздохнула, словно пыталась собраться перед заключительным рывком в гонке.

— Дэн, я только начинаю свою карьеру, и я не могу сейчас полагаться на удачу. Я нахожу процесс MSF бесперспективным и думаю, что проект провалится. Я не могу позволить себе быть связанной с таким большим провалом на заре моей карьеры. Найдите кого-нибудь другого на роль тестера в вашу группу.

Она собрала веши и вышла из зала заседаний.

В зале воцарилось молчание. Все следили, что же будет делать Дэн, но он просто стоял, глядя на дверной проем, через который вышла Марта. В конце концов, Билл откашлялся и спросил:

— Мне сходить за ней?

Дэн вздрогнул, словно очнулся ото сна, затем взглянул вниз на стол. Наконец он ответил, не глядя на Билла:

– Нет, чиф, это не твоя проблема. Я займусь этим сам.

«Правда, пока не знаю как», — подумал он про себя. Он снова словно отряхнулся, но теперь уже смог взглянуть на остальных членов группы.

— Ладно, Марта подняла довольно интересные вопросы, Хотя ее сейчас нет и она не может участвовать в совещании, давайте обсудим ее возражения, прежде чем двинемся вперед. Может быть, кто-то испытывает те же сомнения или, наоборот, готов ответить Марте?

Стало ясно, что Дэн уже может говорить — шок после ухода Марты прошел, и началось обсуждение. Большинство признало ее аргументы правильными, но хотело знать, не упустила ли она что-нибудь из виду в своем анализе. Джейн упомянула космическую программу: «Они наверняка использовали модель водопада при организации полета на Луну, не так ли? Возможно, огромный размер проекта вызвал необходимость такого подхода».

– Это только часть проблемы, Джейн, — подала голос Мэри-Лу. — Я думаю, что тут был еще один фактор — ясность конечной цели. Когда строится мост иди проектируется полет на Луну, конечная цель не вызывает сомнений. Ты можешь заниматься планированием и при этом быть уверена, что это не окажется потерей времени, потому что стремишься к цели, которая наверняка не изменится. Чем крупнее и конкретнее цель, тем более оправдано использование модели водопада. Наш проект из другой оперы: он не так велик, и мы пока не знаем точно, что же мы хотим построить.

Дэн начал было отвечать Мэри-Лу, но внезапно его перебил Билл.

— Нет, вы оба ошибаетесь, — он вскочил и побежал к доске. Схватив маркер, он повернулся к группе и уже собирался что-то сказать, когда заметил выражение раздражения и обиды на лицах Мэри-Лу и Джейн. Осознав, что поступил не очень вежливо, он остановился.

— Простите ради Бога — я не хотел быть грубым. Меня только что осенила мысль, которая могла бы пролить свет на обсуждаемую тему.

Он повернулся к Дэну.

– Возможно, ты не знаешь, что я участвовал в космической программе, когда служил на флоте. Ничего особенного: я был в группе обработки данных. Но я вернулся мыслями к тем временам и в свете того, что сказала Марта, я вдруг понял то, чего не понимал прежде.

Билл подошел к доске и нарисовал примерную диаграмму модели водопада.

– По словам Марты, все инженеры знают, что крупные проекты типа космических программ используют эту модель. Мы просто со-

гласились с ней в этом. Так вот, она не права, — в конце диаграммы он надписал *«Посадка на Луну»*. — Если вы просто думаете о полете на Луну, это выглядит как классическая модель водопада.

Над диаграммой он нарисовал три цикла модели процесса проектирования MSF и затем взволнованно сказал:

— Проблема состоит в том, что мы забыли о других попытках. А ведь были программы «Меркурий», «Джемини», и только потом появился «Аполлон». Каждый был «очередной версией», если хотите, И знаете еще что? Внутри каждой из этих версий мы делали опытные образцы, демонстрирующие концепцию, — модели для испытания в аэродинамической трубе. И после каждого испытания мы возвращались назад к проектированию, к программе работ, к подсистемам и к отдельным блокам и переделывали то, что нужно.

Все, что у нас было в начале, — концепция, сформулированная Кеннеди. Мы не представляли себе, как в реальности может выглядеть высадка человека на Луну. Мы брали то, чему научились по пути и использовали это в следующей версии. Это был на самом деле абсолютно итерационный процесс, и мы применили именно его, чтобы выполнить один из самых крупных и сложных проектов, когда-либо предпринимавшихся человеком.

Он положил маркер, повернулся и посмотрел прямо на Дэна:

— Знаешь, я был настроен скептически по отношению к этому проекту. После нашего заседания в понедельник я вернулся к себе и просмотрел документацию проекта, разработанного твоей юридической фирмой. Признаю, это далось мне непросто. Когда я прочитал, что вы выпустили первую версию, урезав половину возможностей и отключив некоторые кнопки, я просто съежился. Но затем я прочитал отзывы ваших пользователей.

Билл вернулся на свое место:

— Я был поражен. Пользователи остались довольны! Я продолжал читать, силясь понять, в чем дело, и меня осенило. Поскольку вы зафиксировали дату выпуска и затем заразили всех своей идеей выпуска версий, вы создали продукт, который был одновременно своевременен и функционален. Эта первая победа показала пользователям, что они могут доверять вам, и они были готовы ждать второй и третьей версий, в которых пропущенные возможности уже будут работать.

Решающим аргументом для меня оказался вчерашний ленч. Ты попросил меня подыскать группу для прототипа, поэтому я пригласил на ленч Сэма и Бэт. Когда я объяснил им, какой подход мы собираемся использовать в этом проекте, они не могли дождаться начала работы. Я спросил их, откуда такой энтузиазм, и они ответили, что это будет просто классно — передать продукт пользователям еще до

того, как они забудут, зачем он им нужен, — Билл улыбнулся и продолжил: — Мои ребята, конечно, умеют выражаться прямо.

— Они научились у своего руководителя, Билл, — сказала Джейн, улыбаясь. Дэн и Мэри-Лу тоже не могли удержаться от смеха, а Билл просто покачал головой. Наконец он продолжил:

— Таким образом, я полагаю, что я принял твой образ мыслей, Дэн. Я, конечно, хочу увидеть результат работы, но на самом деле меня не меньше интересует сам процесс — как мы это будем делать.

«Хорошо, — подумал Дэн. — где-то найдешь, где-то потеряешь».

Тут появился Тим. Оглядев комнату, он спросил:

— А где Марта? И где пончики?

«Дождем смыло», — подумал Дэн, ухмыляясь.

— Пончики здесь, соня, — Дэн подтолкнул коробку с пончиками через стол. — Подбрось сахару в свою систему и сможешь догнать остальных.

После этого настрой изменился, и дело пошло гладко. Все занялись пересчетом своих целей на пять итераций, которые обсуждались на последней встрече, когда заполняли диаграмму. Несколько раз Дэну пришлось разъяснять кое-что, но, в общем, он был доволен: все прочитали материалы и осознали метод выпуска версий.

		Исследование	Планирование	Разработка 1	Разработка 2	Стабилизация
Менеджер продукта	Исследование	Подготовка концепции	Концептуальный проект уточнение концепции	Функциональные возможности альфа-версии	Функциональные возможности бета-версии	Оценка реакции пользователей
Менеджер программы	Планирование	Контроль реалистичности графика	Завершение всех планов	Проверка планов	Планирование завершающей стадии	Контроль окончательного тестирования и готовности к развертыванию
Разработчик	Разработка 1	Прототип	Концептуальная система (КС)	Альфа-версия	Бета-версия	Окончательная версия
Инструктор	Разработка 2	Отклики пользователей на прототип	Помощь в концептуальном проектировании	Документирование взаимодействия пользователей с приложением	Предварительный план обучения пользователей	Окончательный план обучения пользователей
Тестер	Стабилизация	План тестирования прототипа	План тестирования КС	Управление ошибками	Тестирование производительности	Окончательное тестирование
Логистик	Стабилизация	Установка прототипа	Установка КС. оценка результатов	Пилотное развертывание 1	Пилотное развертывание 2	Выпуск

Затем он описал концепцию компромиссного треугольника и рассказал, как использовать ее, чтобы сбалансировать ресурсы, графики и возможности. Все прониклись этой идеей моментально и постарались привести примеры для каждой стороны треугольника. Дэн показал матрицу проекта, попросив перечислить ограничения, имеющие значение для проекта. Через минуту Тим уже сформулировал все три: оптимизировать график, ограничить ресурсы и реализовать только необходимые функциональные возможности.

Наконец Дэн положил слайд в проектор и объяснил, из каких промежуточных этапов состоит фаза исследования. Первым этапом он назвал формирование проектной группы. Услышав хихиканье, Дэн отреагировал так:

— Да, примерно 30 минут назад я думал, что по крайней мере этот вопрос мы закрыли.

Он вернулся к описанию двух оставшихся этапов.

— Мы подготовим первый вариант документа оценки рисков в пятницу. Постарайтесь прочитать материал, который вы получили по этому поводу, — он помолчал. — Я запланировал это на сегодня, но думаю, лучше отложить до пятницы.

Все закивали и Дэн понял, что они надеются, что он убедит Марту присоединиться к группе. «Похоже, у меня будет еще один промежуточный результат», — думал он, когда встреча закончилась.

Когда он вернулся в свой офис, секретарь сообщила, что заместитель директора по строительству уже два раза звонил и, похоже, он чем-то огорчен. Дэн перезвонил ему.

— Что ты сделал с Мартой, Дэн? — замдиректора почти кричал. — Она пришла ко мне и сказала, что ушла из вашей группы, потом стала что-то говорить насчет MSF или FMS и итераций. Сейчас она у себя и доносятся такие звуки, словно она собирает свои вещи! Ты не хочешь рассказать мне, что, собственно, происходит?

Дэн немедленно отправился в технический отдел и коротко объяснился, из-за чего расстроилась Марта. Они пошли вместе к Марте в офис, чтобы убедить ее остаться и в компании, и в группе. Убедить Марту остаться в компании удалось довольно быстро, а с проектом были сложности до тех пор, пока Дэн не привел пример Билла с космической программой. Марта думала долго и, в конце концов, спросила своего босса:

— А как вы думаете?

— Марта, — ответил тот, протирая свои очки, — я не очень хорошо представляю себе все стороны разработки программного обеспечения. Но зато я хорошо знаю как пользователь, что, если этот метод быстрых итераций даст мне необходимый продукт в короткие сроки, — я обеими руками «за». И вот еще что, — он наклонился вперед, словно собирался поделиться чем-то очень личным, — откровенно говоря, я завидую тебе, потому что ты участвуешь в этом проекте. У нас в техническом отделе мы действительно сначала проектируем все в деталях, но мы *вынуждены* это делать. Когда-то было решено, что *эти* заклепки делаются именно из *такой-то* стали, и чтобы отказаться от такого решения, нужно провести немыслимую работу. С Дэном все по-другому. Вы можете вносить изменения значительно быстрее

и значительно дешевле. В течение нескольких часов программисту удается совершенно изменить вид страницы на экране или даже разработать новую, — он вздохнул. — Будь я помоложе, я бы использовал шанс поучаствовать в этом, хотя бы просто для расширения кругозора. — Он надел очки и внимательно посмотрел на Марту: — Марта, ты молода. Тебе следует использовать этот шанс,

В конце концов, Марта согласилась вернуться в проект. Дэн отлично понимал, что убедили ее слова директора.

Второй раунд

Когда Дэн пришел в пятницу в дубовый зал, Тим уже был там, просматривая материалы. Дэн обратил внимание, что кофе готов, и налил себе немного. Он знал, что Тим не пьет кофе, поэтому был приятно удивлен тем, что кофе оказался хорошим.

— Спасибо за кофе, Тим.

— Понимаешь, после опоздания в среду я решил, что сегодня, пожалуй, лучше приду пораньше, чтобы кое в чем разобраться, — Тим говорил с ухмылкой. — Конечно, мне помогли.

Дэн удивленно увидел, как в комнату вошла Марта с очередным кофейником в руке. Марта посмотрела на него и улыбнулась:

— Итак, после всего что было, у нас двоих есть шансы отделаться выговором или теперь нам придется целый месяц готовить кофе, Дэн?

— Мне еще нужно все проверить, но. думаю, вы оба легко отделаетесь, — Дэн отхлебнул еще кофе. — Но должен сказать, что еше одно такое же собрание, и я заканчиваю с кофе и перехожу на жидкость для бальзамирования.

Тим и Марта засмеялись, Марта села напротив Дэна.

— Извини, Дэн. Я чувствовала себя так, словно меня поймали в ловушку, и была не уверена в себе; мне казалось, что я втянута в проект, который невозможно реализовать, и я не знала, смогу ли быть на должном уровне.

— Извинения приняты, Марта, — Дэн помолчал и сказал: — Что мы можем сделать, чтобы ты почувствовала себя увереннее?

Как раз в тот момент, когда она собиралась ответить, остальные три члена группы вошли в конференц-зал. Мэри-Лу и Билл спорили. Дэн посмотрел на Джейн, которая загадочно улыбалась. «Похоже, разработчик и инструктор уже взаимодействуют на благо пользователей. Приятно видеть такую высокопрофессиональную дискуссию». В этот момент Мэри-Лу крикнула: «Только Web!», на что Билл ответил (тоже на повышенных тонах): «Под Windows!». Джейн, улыбаясь, просто пожала плечами.

– Так, берите пончики и садитесь, — обратился к ним Дэн, подталкивая коробку с пончиками через стол, Мэри-Лу, Билл и Джейн уселись, и Дэн продолжил: — Прежде чем мы перейдем к повестке дня, вы, может быть, поделитесь с нами, о чем идет спор? Мэри-Лу?

Мэри-Лу опустила свой пончик.

– Когда л поняла, что несу ответственность за то, чтобы интересы пользователей **RMS** не были забыты в процессе работы, я решила поговорить с некоторыми из них, чтобы понять, что они думают. Ты знаешь, многие ходят на мои занятия. Я поговорила с ними, и оказалось, что все они хотят иметь возможность работать через Интернет. Поэтому я и говорю Биллу, что нам нужно, чтобы у приложения был Web-интерфейс.

— Ая говорю Мэри-Лу. что это чушь, — Билл обращался сразу и к Мэри-Лу, и к Дэну. — Это будет означать, что нам придется делать два приложения: одно для Интернета, другое для локальной сети. Мы упремся в проблемы безопасности, аутентификации, доступа — это какой-то кошмар для разработки. Я повторяю — только под Windows, и пусть они приходят в офис по субботам, чтобы заполнить свои табели. — Он повернулся к Дэну. — Итак, кто из нас ошибается? – спросил он с самодовольным выражением на лице.

— Оба!

Мэри-Лу и Билл были шокированны ответом Дэна, а Джейн рассмеялась.

– Вот видите? —сказала она, показывая на обоих. — Мы не принимаем окончательных технологических решений до начала стадии планирования, не так ли?

— Я вижу, ты читаешь материал загодя, Джейн, — сказал Дэн. Он повернулся к Биллу и Мэри-Лу: — Я рад, что вы обращаетесь к этой проблеме, но Джейн права — сейчас слишком рано решать вопросы интерфейса. Если мы это сделаем, это ограничит наши возможности.

Обращаясь ко всей группе, он продолжил:

– Это и есть наш сегодняшний вопрос. Перейдем к повестке дня. Как видите, у нас сегодня полно дел. Поскольку мы сократили нашу встречу в среду, мы не обсудили концепцию, как я планировал. Поэтому, прежде чем заняться сегодняшней повесткой, я бы хотел добавить это первым пунктом сегодня. Нет возражений?

Все кивнули в знак согласия и начали записывать новый пункт.

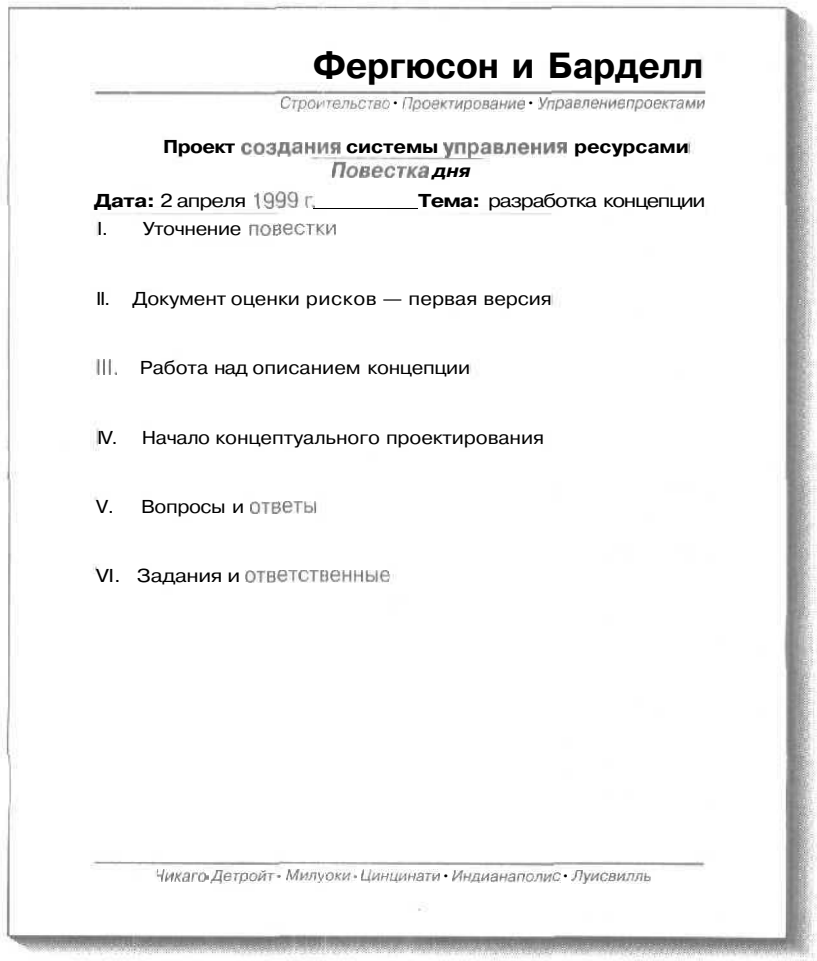
– Может быть, кто-то хочет внести изменения в повестку дня? — спросил Дэн.

Марта подняла руку.

– Этого нет в повестке дня, но я хотела бы сейчас принести извинения за свой демарш в среду. Это было несолидно и непрофессионально, и мне жаль, что я так поступила.

Ответом были улыбки и поклоны членов группы, а Дэн сказал;

- Мы с Мартой потом поговорили, и я думаю, что мы устранили ее беспокойство по поводу итерационных процессов. Фактически спор Билла и Мэри-Лу — еще один пример, показывающий, почему мы предпочитаем итерационный способ разработки. При постепенной работе через выпуск последовательных версий мы останавливаемся на лучшем решении, проверяя приложение в реальной среде и используя полученные результаты. По существу, мы фильтруем информацию, в конце получая согласованное решение, соответствующее нашим потребностям. Это понятно?



Фергюсон и Барделл

Строительство • Проектирование • Управлениепроектами

Проект создания системы управления ресурсами
Повестка дня

Дата: 2 апреля 1999 г. **Тема:** разработка концепции

I. Уточнение повестки

II. Документ оценки рисков — первая версия

III. Работа над описанием концепции

IV. Начало концептуального проектирования

V. Вопросы и ответы

VI. Задания и ответственные

Чикаго • Детройт • Милуоки • Цинцинати • Индианаполис • Луисвилль

Марта кивнула и улыбнулась:

– И именно моя работа будет гарантировать, что мы не провалим работу, так?

– Именно. Поэтому мы так рады, что ты с нами, — он подождал секунду, чтобы увидеть, нет ли других комментариев, и затем продолжил: — Хорошо, давайте начнем работу с документами. — Он положил слайд в проектор.

– Здесь шесть частей документа, который вы прочитали. Кто хочет высказаться прямо сейчас?

— Джейн хочет, — сказал Тим. — Менеджер продукта — главный человек на фазе «Анализ», и он руководит подготовкой концепции.

– Верно, Тим, — сказал Дэн, поднимаясь со стула. — Итак, госпожа менеджер продукта, занимайте место во главе стола и управляйте собранием.

– Джейн Клейтон, начинайте! — сказала Мэри-Лу, а остальные зааплодировали и засмеялись. Джейн тоже смеялась, но в ее глазах была решимость. Она положила записи напротив себя и взглянула на товарищей: — Хорошо, друзья, давайте рассмотрим концепцию. Прежде всего, какую проблему мы пытаемся решить?

Дэн кивнул одобрительно — ему понравилось, что Джейн подключает группу к дискуссии. «Возможно, она не очень хорошо представляет себе процесс разработки программного обеспечения, — подумал он, — но зато умеет вести собрание».

Все быстро согласились с такой постановкой проблемы; замена системы учета отработанного времени и совершенствование процесса распределения ресурсов между проектами. С формулировкой концепции было тяжелее, потому что Билл, Тим и Мэри-Лу хотели описать продукт, используя специфические термины, применяемые при описании технологии и продуктов. Наконец, Джейн повернулась к Марте.

– Можешь быть, ты расскажешь об основной цели продукта?

– Конечно. Как насчет такой формулировки: «RMS позволяет руководителям распределять ресурсы между проектами, базируясь на доступности и характеристиках ресурсов, а всем пользователям – регистрировать отработанное время в едином хранилище данных. Таким образом, они смогут вести подготовку отчетов и выписку счетов из единого источника»,

– Но это так *общо,* так *широко!* ~ воскликнул Билл.

– Зато схвачена суть, — ответила Джейн. — Концепция и все ее части обязательно должны быть высокоуровневыми. Мы не обращаемся к частностям до фазы планирования. Если мы соглашаемся с формулировкой Марты, то ее вполне можно пока использовать. Но все ли согласны с тем, как ее формулировка описывает то, чего мы ждем от продукта?

После некоторой дискуссии группа одобрила предложение Марты как основу формулировки целей проекта. После этого Джейн поведала об исследовании запросов пользователей.

- Как насчет концепции? — спросил Билл.

Джейн ответила:

— Давайте сначала пройдемся по использованию и бизнес-целям. Я думаю, подход станет яснее.

Билл нахмурился, но кивнул, и Джейн продолжала;

- Я подготовила несколько профилей потенциальных пользователей RMS, на которые попрошу вас взглянуть. Мэри-Лу помогла мне, так как она уже обучала многих сотрудников. Я хочу, чтобы вы выразили свое отношение к этим профилям.

Профили оказались краткими, но емкими. В шапке таблицы она наметила четыре типа пользователей: сотрудники, руководители, клерки и администрация. В левой части таблицы она выделила два подтипа: мобильные и офисные. В ячейках таблицы она перечислила стандартные компьютеры, имеющиеся в распоряжении каждого типа пользователей, средний уровень квалификации и базовые потребности в приложении.

— Джейн, почему у тебя нет информации относительно мобильных сотрудников? — спросил Дэн.

- Потому что у нас их нет, — ответила Джейн. — Нашим клеркам не разрешен доступ в корпоративную сеть извне.

— Хорошая работа, Джейн, — сказал Дэн. — Нам следует иметь в виду запросы пользователей, когда мы будем проектировать приложение. — Джейн кивнула и перешла к рассказу о бизнес-целях.

Она начала с того, что написала на доске «*$412 000*» и спросила:

- Кто-нибудь знает, что означает эта цифра?

Первым отреагировал Тим:

— Сумма моих кредитов на учебу?

Когда все закончили смеяться, Джейн сказала:

- Может быть и так, но я имела в виду другое — чем может быть полезен RMS. Это цифра, к которой мы с Джимом Стюартом пришли при оценке возможной экономии при переходе к еженедельной выписке счетов.

- Это похоже на очень важную бизнес-цель, — сказал Билл.

— Да, но что будет, если мы не достигнем ее? Есть еще какие-то бизнес-цели и потребности, в достижении которых поможет RMS, даже если мы оставим двухнедельный цикл прохождения счетов без изменений. Давайте-ка потратим немного времени на мозговой штурм, — Джейн, взяв маркер, записывала на доске идеи, которые высказывали члены группы. Через некоторое время перечень бизнес-целей был составлен.

— А сейчас, — сказала она, — с учетом того, что мы уже сделали — а мы поставили проблему, сформулировали концепции, определили профили пользователей и бизнес-цели — сформулируйте мне в двух-трех предложениях, какова ваша концепция решения.

Снова она работала у доски, а члены группы предлагали формулировки в терминах продуктов и технологий. Наконец, вмешался Дэн:

– Так, друзья, уходите от сложных формулировок и специальных терминов. Это может понять только другой разработчик. Описывайте все так, чтобы было понятно самому неквалифицированному пользователю.

– Итак, — сказала Джейн, — кто может сформулировать цели проектирования продукта? Что он должен делать и каковы ограничения? Что мы можем сейчас об этом сказать?

– Ну, для начала мы не должны выходить за пределы существующей пропускной способности сети, — решительно сказал Тим.

– Почему нет? — спросила Мэри-Лу.

– Потому что в прошлом году потратили кучу денег на то, чтобы подключить всех к 100-мегабитному Ethernet и предоставить всем доступ в Интернет, и я не хочу затевать все это еще раз. У меня на этот год совсем другие планы, — ответил Тим.

– Кроме того. — добавил Дэн, — я бы сказал, что RMS должен приспосабливаться к нашим текущим технологиям. Мы, например, не можем изменить организацию сети или систему передачи сообщений. Я не думаю, что существуют цели, которыми можно было бы обосновать такие затраты.

– Хорошо, а у меня есть эгоистическая цель, — довольно резко сказал Билл. — Видите ли, у меня появились молодцы, у которых просто зуд — они хотят попробовать все, чему их научили. Кое-что я смог понять и, похоже, это действительно способно работать. По-моему, стоит предоставить им свободу, чтобы посмотреть, соответствуют ли эти новые подходы нашим задачам или им это только кажется.

– Это никакой не эгоизм. Билл, — сказал Дэн, — это просто забота о подчиненных. Каждый хороший руководитель хочет, чтобы его люди могли вырасти и проявить себя. Но, конечно, остается вопрос: действительно ли с помощью этих технологий можно сделать то, что нам нужно? Я думаю, мы оставим решение на твой суд, потому что мы знаем, что в конечном счете ты сделаешь то, что нужно.

Все согласились, и Билл кивнул в знак благодарности.

– Хорошо, идем дальше, — сказала Джейн. — Полагаю, у нас есть предварительный вариант концепции. Я наберу его и сегодня же разошлю вам по электронной почте. Верните его мне со всеми замечаниями, которые придут вам в голову, и мы обсудим все это в понедельник, – она повернулась к Дэну: — Я должна еще что-нибудь сделать, босс?

— Еще одна вещь. Джейн, Отправь копию Джиму Стюарту, Он придет на нашу встречу к понедельник, и я хочу, чтобы он мог предварительно познакомиться с документом. Не забывайте, что он заказчик этого проекта и должен одобрить концепцию. — Джейн кивнула и собралась было уходить с председательского места, но Дэн жестом попросил ее оставаться на месте. — Просто побудь здесь, пока мы будем намечать документ оценки рисков, Джейн. Ты отлично провела нас через процесс разработки концепции, и я бы хотел, чтобы ты помогла нам в начале работы над рисками, — он передал ей через стол толстую папку: — Здесь слайды, которые тебе понадобятся.

Джейн перебрала слайды и нашла тот, который иллюстрировал процесс оценки рисков. Она положила его в проектор.

— Похоже, что первая вещь, которую мы должны сделать — идентифицировать риски, — она стерла с доски, взяла маркер и сказала: - У кого есть идея, с чего начать?

Некоторое время спустя перечень рисков был составлен. Наконец, Дэн сказал:

— Достаточно на сегодня. Я оформлю это и разошлю вам. Вы оцените вероятность и влияние каждого риска на проект, и мы обсудим это в понедельник.

Он встал и посмотрел на Джейн,

- Поскольку мы в первом приближении описали концепцию, пора начинать концептуальное проектирование. Как дела со схемами использования?

Джейн вытащила стопку бумаг из своего портфеля.

- Вроде бы неплохо. — сказала она. — Это оказалось несложно после того, как ты один раз показал мне, что нам нужно.

— Отлично. — Дэн пересел во главу стола и стал делать пометки в своем экземпляре повестки дня. — У кого-нибудь есть вопросы? Нет? Давайте распределять задания. Билл, ты и твои люди должны быть в понедельник. Сможете что-нибудь показать нам?

- Наверняка. Сэм и Бэт уже работают над этим, и у них есть несколько хороших идей. Я передам им вариант концепции, чтобы они могли убедиться, что находятся на верном пути. У них есть работа до выходных, но они знают наш график и сказали, что поработают сверхурочно.

- Помни, Билл, первый прототип не обязан быть идеальным, — предостерег Дэн. — Блок-схема, может быть, наметки интерфейса — ничегоособенного.

Билл кивнул:

- Я знаю, мы говорили об этом, но Сэм так быстро работает в Visual Basic, что, я думаю, у него уже готовы некоторые экраны. Бэт

работает над первым приближением блок-схемы, так что она, возможно, тоже будет готова.

- Ну и ну, это грандиозно, Билл. Это много больше, чем я мог надеяться, ведь времени было совсем мало, — Дэн заглянул в свои записи и добавил: — Джейн, ты отвечаешь за то, чтобы вариант концепции оказался у Джима Стюарта до понедельника. Я отправлю всем матрицу рисков, и каждый из вас внесет туда собсгвенное мнение о вероятности и опасности каждого риска. Джейн пошлет каждому из вас наш вариант концепции, чтобы вы его просмотрели и отредактировали до конца недели. Отправьте мне по почте и вашу работу по рискам, и ваши поправки к концепции до шести вечера в воскресенье.

В понедельник утром мы посмотрим первые прототипы, попытаемся закончить с рисками, доведем до конца концепцию и встретимся с нашим заказчиком, Джимом Стюартом, чтобы узнать, как он оценивает наше продвижение. Все ясно?

- Кто принесет кофе и пожевать? — спросил Тим, прикончив последний пончик.

- По-моему, моя очередь, — прорычал Билл. — Уж я-то точно не хочу, чтобы ты умер от истощения, Тим. — Он слегка ударил Тима в живот, и Тим повалился на стол, как смертельно раненый.

Пока все смеялись, Джейн сказала:

- Забудь об этом, Билл — у тебя достаточно дел на выходные. Я принесу все в понедельник. К тому же, когда ты готовил кофе, у него был такой вкус, словно ты нашел его где-то на корабле,

- Вот-вот, в трюме! — закричал Тим и выбежал из комнаты, преследуемый Биллом.

Мэри-Лу повернулась к Джейн и Марте.

- Пойдемте, девочки, пусть мальчики развлекаются, — сказала она.

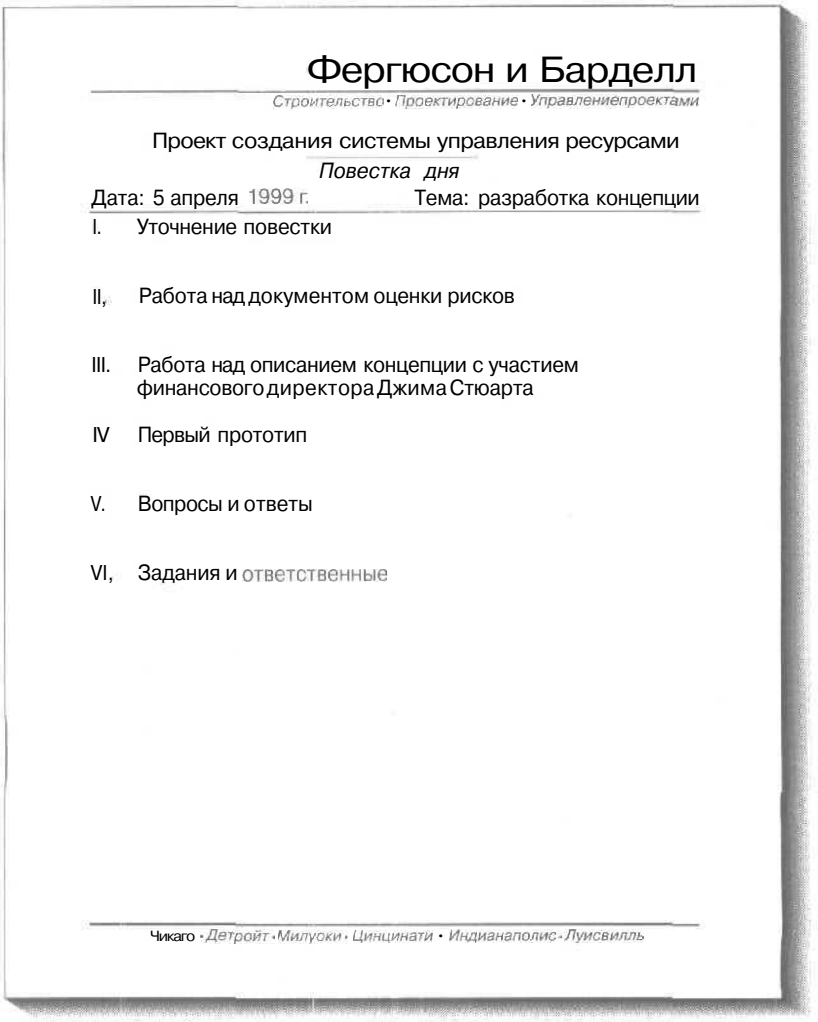
Дэн улыбнулся. «Это здорово, когда встречи получаются одновременно и полезными, и приятными», — думал он, собирая свои вещи. — Правда, иногда самая лучшая работа следует за конфликтом». Он выключил свет и направился в свой офис, уже обдумывая встречу в понедельник.

Взгляд клиента

В понедельник утром рассвет был ранним и ясным — слишком ранним для Дэна, который, казалось, только и заснул, когда зазвенел будильник, «По крайней мере, у меня все готово, так что я могу просто все взять и пойти», — размышлял он, ковыляя в душ.

Через сорок пять минут он уже подъехал к парковке, схватил портфель и направился в свой офис. Внезапно урчание в животе напом-

нило ему о том, что он еще ничего не ел. «Я очень надеюсь, что Джейн не забыла принести поесть», — подумал он, собрав папки с материалами, которые он напечатал прошлой ночью, и направляясь в конференц-зал.



Фергюсон и Барделл

Строительство • Проектирование • Управлениепроектами

Проект создания системы управления ресурсами

Повестка дня

Дата: 5 апреля 1999 г. Тема: разработка концепции

I. Уточнение повестки

II, Работа над документом оценки рисков

III. Работа над описанием концепции с участием финансового директора Джима Стюарта

IV Первый прототип

V. Вопросы и ответы

VI, Задания и ответственные

Чикаго • *Детройт • Милуоки • Цинцинати • Индианаполис • Луисвилль*

Перед собой он увидел Билла и двоих молодых людей, которые шли в ту же сторону. В руках у одного из них был ноутбук. Все трое оживленно спорили. Дэн услышал слова «COM» и «Visual Modeler» и

понял, что Билла сопровождали Сэм и Бэт, те два программиста, которые работали над прототипом.

Дэн прибавил шагу и, догнав их, спросил:

- Ну как, есть хорошие новости?

Сэм и Бэт кивнули и улыбнулись, и затем посмотрели на Билла.

— Точно! Дэн, эти двое просто творят чудеса. Я поражен тем, что они умудрились сделать с прошлой среды, — Билл печально покачал головой. — Я посрамлен, Дэн. Похоже, что старые ковбои Кобола безнадежно отстали,

- Не давайте ему обмануть себя, мистер Шелли, — сказал Сэм, когда они зашли в конференц-зал. — Он был с нами, делал экраны и все чистил, пока мы двигались вперед. Когда Бэт показала ему блок-схему, он внес очень полезные предложения. В базах данных он разбирается, это точно.

— Я уже говорил тебе, Сэм, что в этом году денег на повышение зарплаты нет, так что перестань подлизываться, — прорычал Билл, однако Дэн заметил, что комплименты были ему приятны.

Схватив пончик и кофе, Дэн пошел к центру комнаты, испытывая благодарность к Джейн. Он убрал папки с новыми материалами в шкаф, заправил слайд с повесткой дня в проектор и повернулся к группе:

- Всем доброе утро.

Когда стихли ответные приветствия, Мэри-Лу добавила:

— О, этим утром мы все выглядим поживее!

Дэн улыбнулся.

- Все в порядке, Мэри-Лу. Правда, я жив только за счет энтузиазма, — он заглянул в повестку дня. — Все готовы начать?

- Эй, а где мистер Стюарт? — спросил Тим. — Я думал, что он присоединится к нам.

— Он подойдет попозже, Тим, — Дэн раздавал папки, которые он подготовил прошлой ночью. — Я думаю, что мы можем справиться с идентификацией рисков довольно быстро, так что я попросил его присоединиться к нам примерно в 8:45.

Дэн поменял повестку дня на другой слайд.

— Я высоко ценю работу, проделанную вами с матрицей рисков, которую я отправил. Оценки вероятностей и влияний оказались на редкость схожими. Я привел их к общему знаменателю, посчитал общее влияние и проранжировал их. Давайте посмотрим и решим, согласны ли мы с коллективным разумом нашей группы.

Как Дэн и ожидал, все признали главным риском расползание границ проекта. Из-за сжатого графика срыв сроков сдачи оказался следующим. Тим убедительно доказал, что группа должна рассматривать влияние проекта на работу сети и вероятные в связи с этим про-

блемы с сетью как третий важнейший риск. За остальные риски проголосовало всего по несколько членов группы.

Группа поработала с каждым риском, рассматривая планы снижения риска, планы действий в непредвиденных ситуациях и их запуск. В большинстве случаев оценка рисков не требовала никаких дальнейших исследований. Однако когда Тим и Билл начали спорить о влиянии, которое приложение может оказать на сеть, все решили, что Тиму придется проделать дополнительную работу по оценке риска избыточной нагрузки на сеть.

Они как раз завершали работу по оценке рисков, когда в зал вошел Джим Стюарт.

— Добро пожаловать, господин директор! — сказал Дэн. — Берите пончик и садитесь в кресло — то и другое имеется в достаточном количестве.

Джим сел напротив Дэна, проигнорировав его рекомендации.

— Надеюсь, я вовремя, — скачал он отрывисто. Что-то в его тоне насторожило Дэна, хотя лицо Джима оставалось беспристрастным, Дэн подумал: «Возьми себя в руки! Даже если ты устал, не злись на других». Он взял коробку с пончиками, убрал ее в шкаф и посмотрел прямо на Джима.

— Ты вовремя, как всегда. Мы как раз заканчиваем подготовку документа по оценке рисков и готовы рассмотреть концепцию вместе с тобой. Надеюсь, у тебя есть экземпляр? — Он посмотрел на Джейн, и она кивнула.

— Да, я его получил, — ответил Джим. Он снял очки и начал медленно их протирать. Дэн уже наблюдал его на совещаниях высшего звена и знал, что, когда Джим занимается своими очками, — это дурное предзнаменование.

— Да, я получил экземпляр, — повторил Джим. Теперь уже все члены группы уловили его тон, и смотрели на него, силясь понять, что это означает. — Я даже принес свой экземпляр сюда. Должен сказать, что он показался мне неполным.

«Началось», — подумал Дэн.

— Неполный в каком смысле, Джим? — спросил он.

Джим начал перелистывать свой экземпляр.

— Я вижу формулировку концепции, пользовательские профили и прочее. Все очень интересно. Особенно предлагаемый подход. — Он положил документ обратно на стол и посмотрел прямо на Дэна. — Но я не увидел ни слова о том, как решение поможет в руководстве проектами. Ни одного слова. Зачем мы затеваем все это, изобретая заново всю систему учета времени и оформления счетов, если мы никак

не помогаем менеджерам в руководстве проектами и в контроле расходов? — И он указал очками на Дэна.

Стараясь сдерживаться, Дэн ответил:

– Что ты имеешь в виду, Джим? Это черновой документ, в конце концов. Мы просили тебя прийти, чтобы получить информацию и услышать твое мнение. Расскажи нам, как ты видишь средства управления проектами в RMS.

– Это же ясно как дважды два, — возмущенно ответил Джим. Он встал и начал ходить туда-сюда. — У нас есть проекты, которые постоянно превышают бюджет. Мы вынуждены тратить тысячи долларов на некоторые из них, потому что они уже запущены. Из-за плохой организации контроля рабочего времени большинство менеджеров проектов жалуются, что не могут получить данные о том. сколько времени затрачено и сколько остается, Прежде чем они разберутся в этом. выясняется, что все лимиты уже превышены.

Теперь появляетесь вы и предлагаете хранить все данные централизованно. Вы разработали эту чудную концепцию, в которой нет ни слова о том, чтобы дать менеджерам проекта инструмент контроля за выполнением проектов. Это что, очень трудно сделать? Как человек, отвечающий за бюджет для этой работы, я требую добавить это в проект!

Тут Билл вскочил и закричал:

– Мы тут что, вьючные животные?! Это — ноша для быка, Джим! Зачем ты сюда явился, размахивать своим авторитетом? Мы тут собрались, поломав все свои планы, чтобы сделать это приложение за два месяца — два месяца! — а у тебя хватает духа просить нас о большем! Я не знаю, почему ваши менеджеры проектов не могут вручную отслеживать ход проектов, как это делаем мы. Мы в информационном отделе отлично контролируем свои проекты, слава Богу, и без специальных инструментов,

Джим пристально посмотрел на Билла,

— Да, я видел. как ты руководишь проектами, Билл. Насколько ты задержал свой последний проект— на шесть месяцев, если я не ошибаюсь?

Билл покраснел, однако он не успел ничего ответить — Дэн стукнул кулаком по столу:

— Достаточно! Сядь, Билл, и помолчи, пока ты не сказал еще что-нибудь такое, о чем всем придется жалеть. Джим, здесь не базар, чтобы бросать обвинения в лицо Биллу. Мы знаем причины, по которым пострадал тот проект. Именно их мы и пытаемся избежать в этом проекте.

Дэн прошел на председательское место и снова положил в проектор матрицу рисков.

– Джим, мы работали над этим до твоего прихода. Пожалуйста, посмотри на нее, пока я буду объяснять, что произошло со средствами контроля хода проектов.

Дэн показал на первую строку матрицы.

– Это те риски, которые мы идентифицировали для этого проекта. Какой риск мьт признали самым главным?

Джим, сощурившись, смотрел на экран, пока не вспомнил, что следует надеть очки. Первый пункт он прочел вслух;

– Раздувание рамок проекта. Звучит как название какой-нибудь ужасной болезни.

– Так и есть, Джим, так и есть, — сказал Дэн. — Раздувание рамок разрушает план проекта едва ли не чаще всех остальных рисков. Это происходит, когда заказчик, движимый самыми лучшими намерениями, настаивают на добавлении новых возможностей к проекту, не учитывая все связи и ограничения, — Дэн повернулся к Джейн: — Джейн, мы учитывали отслеживание хода проектов как потенциальную цель RMS?

– Конечно, Дэн, — ответила Джейн. — Многие пользователи говорили об этом, когда я беседовала с ними. Мы обсуждали это на последней встрече.

– И что же случилось с этим требованием? — спросил Дэн, косясь на Джима, чтобы видеть его реакцию на ее слова,

– Мы решили отложить это на вторую версию,

– Но почему? — Джим повысил голос. — И что еще за вторая версия? Почему не заложить это в приложение уже сейчас?

– Из-за компромиссного треугольника, — ответила Мэри-Лу, направляясь к доске. Она нарисовала треугольник и объяснила Джиму связь ресурсов, характеристики продукта и графика работ. — Видишь ли, мы решили, что, если мы включим средства контроля проектов в первую версию, мы сорвем график, а мы этого не хотим. Мы хотим выпустить эту версию RMS вовремя, чтобы наши пользователи начали верить нашим словам. Даже понимая важность средств, о которых ты говоришь, мы не рассматриваем их как основную цель проекта и поэтому отложили ее на вторую версию.

– И я могу быть уверен в том, что вторая версия действительно появится? — скептически спросил Джим. — Обещать-то легко, а вот выполнить иногда трудно.

– Ты начнешь верить в появление второй версии, когда мы вовремя выпустим первую, — сказал Дэн, — когда ты увидишь, что мы держим слово. Мы начнем работу над второй версией сразу после выпуска первой и будем работать над ней так же, как над первой,

Джим обдумал все услышанное и сказал:

— Дэн, ты пришел к нам из юридической фирмы с великолепными рекомендациями. Одна вещь, которую они отметили — это то, что практически все твои проекты сдавались вовремя и всегда соответствовали спецификации, — он помолчал, глядя на Билла, и затем продолжил: — Без обид, Билл, но я еще никогда не видел, чтобы наши разработчики выпускали что-нибудь вовремя или по спецификации, а уж чтобы и то, и другое.... Я знаю, что я скептик. Если ты, Дэн, и вы все сможете сделать то, что написано в этом документе в указанные сроки, я подожду вторую версию с реализацией нужных мне средств управления проектами.

Воцарилось молчание, которое нарушил Тим. Глядя на Дэна, он сказал:

— Ну и что же, мы снизили этот риск или полностью исключили его? — все замерли и посмотрели на Джима, ожидая его реакцию на комментарий. Джим пристально взглянул на Тима, потом засмеялся и все расслабились.

Джим повернулся к Биллу:

— Мне жаль, что я так на тебя набросился. Наверное, мне на минуту показалось, что могу своим приказом создать все программное обеспечение. — Он протянул руку и после паузы Билл медленно пожал ее.

— Да, что там, я тоже потерял самообладание. Извини.

— Я надеюсь. — сказал Дэн, — что примерно через семь недель у вас снова будет повод пожать друг другу руки. Это будет означать, что мы закончили разработку первой версии вовремя и начали разработку второй версии с теми возможностями, о которых говорил Джим. Чтобы достичь этого, нам надо обсудить остальные пункты сегодняшней повестки дня, так что — вперед.

Группа работала над концепцией, внося незначительные поправки, но принципиально ничего уже ни изменяя. Когда они закончили, Дэн сказал:

— Итак, Билл, время пришло! Твои ребята действительно готовы пролить свет на то, как будет выглядеть RMS?

— Конечно, — сказал Билл, а Сэм и Бэт двинулись к своим местам. Бэт положила свою блок-схему на стойку во главе стола, а Сэм подготавливал монитор на другом конце. — Сначала посмотрим, как у Бэт организованы потоки данных и программные потоки приложения. Потом посмотрим, как Сэм сделал основные экраны для RMS, и выслушаем ваши мнения.

Работа над блок-схемой не заняла много времени — приложение было несложным. Когда Сэм развернул свой монитор и прошелся по нескольким хорошо подготовленным экранам, Мэри-Лу присвистнула:

— Ого, Сэм, неужели ты сделал все это с прошлой среды?

— На самом деле когда Билл принес нам черновик концепции, я понял — то. что мы уже сделали, совершенно не годится, так что за выходные я почти все переделал, — ответил Сэм. Он минимизировал приложение, демонстрирующее экраны, и сказал: — Посмотрите на это.

Сэм запустил на своем компьютере программу-обозреватель, загрузил файл с жесткого диска, и все увидели таблицу для ввода данных об отработанном времени.

— Я думаю, вы хотите увидеть, на что может быть похож Web-клиент, — сказал он, улыбаясь.

— Как ты сумел сделать все так быстро, Сэм? Отличная работа — Дэн потрясенно смотрел на таблицы и ссылки. — Это даже похоже на таблицы, которые, как я видел, некоторые сотрудники уже применяют для подсчета отработанного времени.

— Это потому, что я шел от электронных таблиц. Office отлично работает с HTML, так что я просто подготовил таблицу, которая была мне нужна, сохранил ее в формате HTML, а затем немного подработал. Это не заняло много времени.

— Да, но чтобы она по-настоящему заработала, времени уйдет побольше, — сказал Билл, нахмурившись. — Не думай, что это поддержка твоих идей о Web-клиенте, Мэри-Лу.

Мэри-Лу только усмехнулась:

— Я просто сохраню эту маленькую демонстрацию в памяти до фазы планирования, когда мы доберемся до разговора о том, какой клиент делать.

Некоторое время группа изучала все экраны в деталях, одобряя одни и предлагая изменить другие. Сэм и Бэт записывали замечания, и Сэм пообещал, что переработанный вариант будет готов к среде.

Наконец, Дэн сказал:

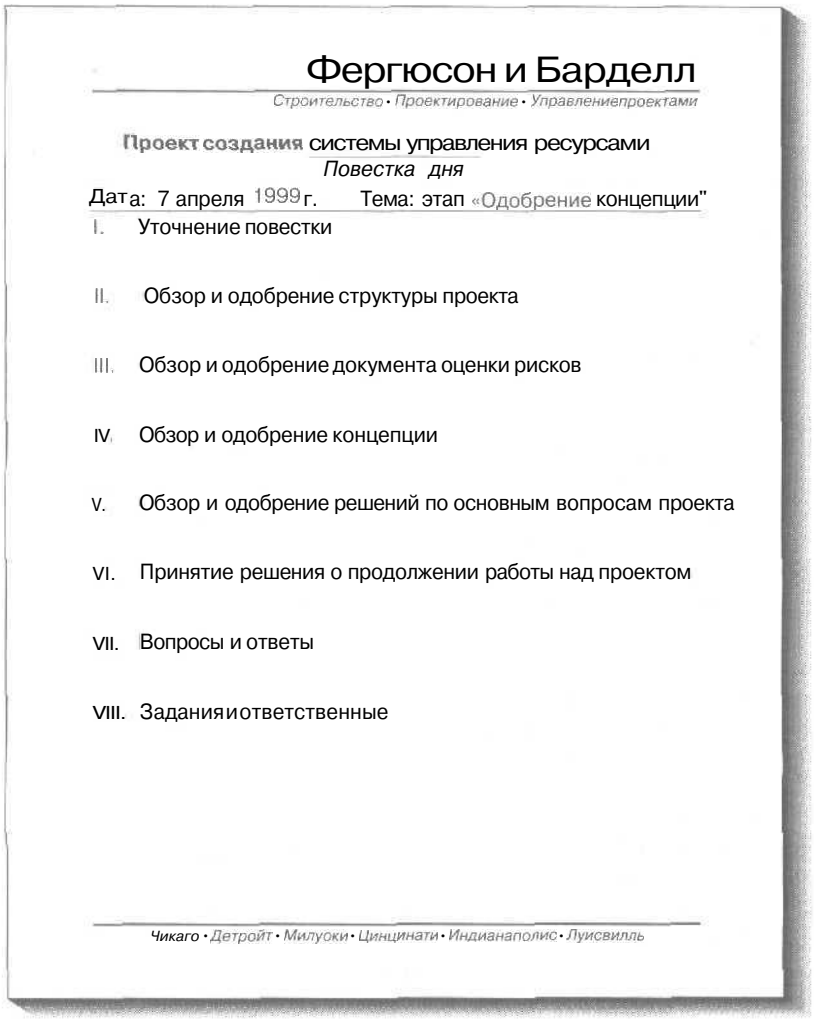
— Я думаю, на этом этапе достаточно. Хорошая работа, Сэм и Бэт. Полагаю, у всех прибавилось уверенности в том, что RMS все-таки состоится.

Он посмотрел на часы:

— Почти 10. Пора заканчивать. У кого-нибудь есть вопросы? Нет? Хорошо, тогда вот план на среду.

— В среду мы встречаемся в последний раз по поводу фазы «Анализ». Мы закончим концепцию и документ оценки рисков и посмотрим исправленный вариант. К тому времени я закончу структуру проекта. Наконец, мы примем самое главное решение — двигаться дальше или нет, — он поднял один из толстых пакетов которые он подготовил заранее: — Здесь материал, который вам придется прочитать к

среде. Тут основные положения к фазе планирования, включая информацию о процессе проектирования MSF. Это немало, я знаю, но вам следует прочесть это до утра среды. Как только мы завершим фазу «Анализ», мы должны быть готовы перейти к планированию и начать разработку проекта. Все поняли?



Фергюсон и Барделл

Строительство • Проектирование • Управлениепроектами

Проект создания системы управления ресурсами

Повестка дня

Дата: 7 апреля 1999 г. Тема: этап «Одобрение концепции"

I. Уточнение повестки

II. Обзор и одобрение структуры проекта

III. Обзор и одобрение документа оценки рисков

IV. Обзор и одобрение концепции

V. Обзор и одобрение решений по основным вопросам проекта

VI. Принятие решения о продолжении работы над проектом

VII. Вопросы и ответы

VIII. Заданияиответственные

Чикаго • Детройт • Милуоки • Цинцинати • Индианаполис • Луисвилль

Члены группы кивнули, все поднялись и начали собираться. В это время Тим повернулся к Джиму с коробкой пончиков:

– Еще не дозрел до пончиков?

Джим посмотрел сначала на коробку, а затем на Тима. Смеясь, он взял один пончик, и они вышли вместе.

Итоги

После бурных предыдущих встреч последняя встреча фазы «Анализ» оказалось безоблачной.

Для начала Дэн представил несколько документов, описывающих базовую структуру проекта. Большинство из них носили информационный характер: кто какую роль исполняет, координаты членов группы, электронные адреса и т. п. Там был раздел, посвященный контролю изменений, поясняющий процесс, который группа должна будет использовать для внесения изменений в документы. Был также и график работ, показывающий, как хотел Джим Стюарт, когда надо готовить отчеты и для кого.

Затем группа рассмотрела документ оценки рисков.

– Есть еще информация по вопросу о влиянии RMS на работу сети? — спросил Дэн у Тима.

Тим вставил слайд в проектор. На экране появился график сетевого трафика за две последние недели.

— Я собрал протоколы за две недели, закачал это в Excel и сделал график, — начал Тим. — Как видите, большую часть недели у нас два пика нагрузки: около 10 часов утра и около 13:30. Обычно мы работаем примерно с 55-процентной загрузкой и пиками до 85%. Если все 800 наших работников обратятся к этому приложению одновременно, у нас могут возникнуть проблемы, характер которых зависит от того, где будет находиться приложение и что именно будут делать пользователи. С другой стороны, вероятность того, что 800 человек одновременно станут заполнять свои таблицы настолько мала, что, я думаю, по этому поводу можно не беспокоиться.

– Но, Тим, — заметила Мэри-Лу, — разве это не зависит также от того, как приложение обращается к базе данных? Если приложение будет заново подключаться к базе данных при обращении к каждой записи, для возникновения неразберихи хватит и 250 человек, одновременно запросивших по 50 записей.

По озабоченному виду Тима все поняли, что он не предусмотрел такую возможность.

— Билл? — сказал он, обернувшись к заведующему отделом разработки. — Что ты думаешь об этом?

Билл переадресовал вопрос Сэму и Бэт:

– Что вы думаете по этому поводу?

Сэм отвечал с усмешкой:

– Не имеет никакого значения, сколько пользователей одновременно используют систему — для базы данных они выглядят как один пользователь. Это решит ваши проблем с трафиком?

Все за исключением Билла были удивлены, он же, казалось, задумался о чем-то важном. Он подмигнул Сэму и затем повернулся к Тиму:

– Не волнуйтесь, мистер администратор сети, я думаю, что Сэм спасет вашу сеть от обвала.

– Каким это образом множество пользователей воспринимаются базой данных как один? Это невозможно. — сказала Мэри-Лу.

Билл повернулся к Дэну:

– Мы, кажется, говорили о прототипах, доказывающих корректность концепции? — Дэн кивнул. — Хорошо, тогда просто отложим этот вопрос до этапа проектирования.

Дэн огляделся вокруг. Все, кажется, были согласны. Правда, Мэри-Лу пожала плечами и сказала:

— Я поверю только тогда, когда увижу это, но я готова подождать.

После этого группа последний раз обсудила концепцию. Дэн разослал всем проект изменений по результатам последней встречи и учел в окончательной версии все пожелания. После нескольких незначительных поправок группа одобрила документ. Дэн повернулся к Джиму,

– Ты тоже согласен? — спросил он.

Джим ответил не сразу.

— Да. Я чувствую себя несколько непривычно — мне всегда приходилось прилагать усилия, чтобы добиваться наличия нужных мне функций в программном обеспечении, а вы просите меня просто поверить вам. Но я заинтригован вашими планами и хочу дать вам возможность или одержать грандиозную победу, или потерпеть грандиозную неудачу. Я с вами.

– Хорошо, — сказал Дэн, пока члены группы обменивались довольными усмешками. — Я думаю, что выражу мнение всей группы, если скажу, что мы хотим, чтобы ты, как заказчик, спрашивал с нас, как мы выполняем свои обязательства. Но с другой стороны, мы хотим, чтобы ты был готов честно признать нашу правоту, когда мы выполним или перевыполним свои обещания. Идет?

– Конечно, — твердо сказал Джим.

– Таким образом, концепция одобрена, — сказал Дэн, заправляя в проектор последний слайд. — Одобрение концепции — первый основной этап модели процесса разработки, Мы достигли необходимых результатов, и все, что нам осталось, — убедиться, что мы сходимся во мнениях по следующим шести пунктам.

Он зачитал все пункты по очереди: обоснование проекта, ожидаемый результат, его достижимость, цели и ограничения, характеристики и риски, структура — и спросил у каждого, согласен ли тот. Дэн старался, чтобы каждый был услышан и чтобы каждый одобрил каждый пункт.

Когда все члены группы выразили свое согласие по всем пунктам, Дэн выключил проектор и вновь заглянул в повестку дня.

— Хорошо, последний вопрос. И такой вопрос будет заключать все наши этапные встречи. Мы переходим к следующему этапу или нет?

Группа не ожидала такого вопроса. Джейн, похоже, выразила общее мнение, когда сказала:

- Мне кажется, все было решено, когда мы одобрили основные позиции.

Дэн покачал головой:

- Нет, Джейн, даже если все согласны, через некоторое время может выясниться, что проект не оправдывает затраты или вдруг обнаружится какое-то серьезное препятствие. Кроме того, нам нужно не просто общее согласие, а общая приверженность проекту. Итак, я снова спрашиваю — мы идем дальше?

Один за другим члены группы и Джим Стюарт сказали «Да». Когда они закончили, Дэн резюмировал:

- Вы все согласились, и мы можем перейти к фазе «Планирование», — он сделал пометку в блокноте: — Я зафиксировал решение группы переходить к фазе планирования. Мы начинаем.

Пока Дэн собирал свои бумаги, члены проектной группы и заказчик обменивались улыбками и рукопожатиями. «Наслаждайтесь пока, — подумал он. — Все еще только начинается».

ГЛАВА 6

План проекта

В этой главе

В процессе планирования вы продумываете способы и сроки реализации идей и оформляете идеи в виде планов, задач и графиков. В этой главе в общих чертах описан переход от «теории» к «практике» и рассказано о задачах проектной группы на стадии «Планирование». Мы подробно рассмотрим модель процесса разработки MSF наряду с концептуальной, логической и физической архитектурой приложения. Не забудем рассказать и о том, как различные уровни модели приложения MSF (пользовательский, прикладной и уровень данных) интегрируются в физическую архитектуру приложения. Мы остановимся на основных результатах рассматриваемого этапа: функциональных спецификациях, плане и графике проекта. И наконец, обсудим принципы разработки графика работ и управления рисками, входящие в обязанности руководителя проекта.

Эта глава основана на нашем опыте проектирования приложений и реализации корпоративных проектов и следующих материалах:

- Microsoft Solutions Framework;
- Уокер Ройс (WalkerRoyce) «Software Project Management: AUnified Framework» (Addison-Wesley, 1998);
- МэриКиртлэнд (MaryKirtland) «DesigningComponent-Based Applications» (Microsoft Press, 1998).

Изучив материал этой главы, вы сможете:

- ✓ описать промежуточные этапы и результаты, на основе которых одобряется план проекта;
- ✓ перечислить основные документы фазы «Планирование» и описать их назначение;
- ✓ охарактеризовать роль каждого сотрудника на фазе «Планирование»;
- ✓ разобраться в этапах проектирования приложения;
- ✓ проанализировать бизнес-требования и соответствующие им разделы проекта;
- ✓ обосновать важность функциональных спецификаций;
- ✓ описать основные принципы составления графика работ.

Разработка плана проекта

Вторая из четырех фаз модели MSF, «Планирование» (рис. 6.1), следует за фазой «Анализ», которая, как вы помните, завершилась этапом «Одобрение концепции».

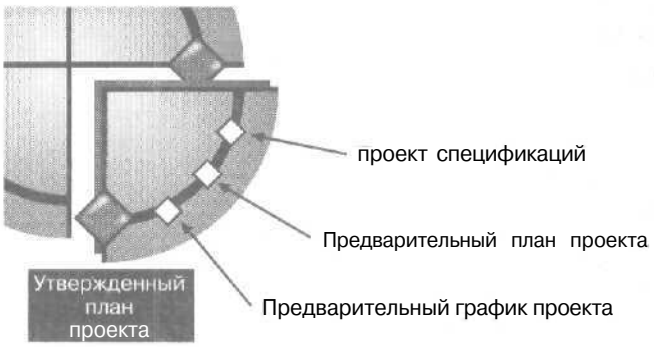


Рис. 6.1. Фаза «Планирование» модели процесса разработки MSF

Вопрос, на который мы отвечали на фазе «Анализ»: «Можно ли разработать технологию решения конкретной бизнес-проблемы, и если да, то как?» В результате мы в общих чертах разобрались в поставленной задаче и наметили ее решение. На этапе «Планирование» ставится другой вопрос: «Сколько времени займет реализация предложенного решения?» Ответом на него должен стать подробный план, согласованный проектной группой и заказчиком.

Из всех фаз разработки MSF труднее всего довести до конца именно планирование. Фаза «Анализ» — это свободный полет мысли, исследование всех возможных решений, волнительный, но чрезвычай-

но интересный период. Фаза «Разработка» — время поработать руками, реализовать все задумки: «Наконец-то мы что-то делаем!» А вот фаза «Планирование» очень изматывает, ведь нужно отвечать на сложные вопросы: «Какую функцию отложить до следующей версии? Какая технология еще не созрела для реализации? Не слишком ли высоки затраты?» Фаза «Планирование» — время сложной, кропотливой работы, поэтому часто возникает искушение пропустить ее.

Но именно на этом этапе решается судьба проекта — ждет его успех или крах. И лучше выявить проблемы на бумаге во время проектирования, чем столкнуться с ними при реализации. Намеченные на этапе «Анализ» идеи должны пройти через горнило фазы «Планирование», и только после этого вы будете уверены, что эти идеи удастся реализовать на стадии разработки и стабилизации.

Фаза «Планирование» и процесс проектирования

Планирование — составная часть процесса проектирования MSF, о котором мы расскажем чуть дальше в этой главе. Хотя частично процесс проектирования MSF выходит за рамки фазы «Планирование», основные решения вырабатываются именно на этом этапе, и именно на них основаны функциональные спецификации — важнейший результат этого этапа.

Распределение ролей при планировании

Для достижения этапа «Одобрение плана проекта», который завершает фазу «Планирование», необходимы усилия всей проектной группы, тем не менее каждый член коллектива должен сконцентрироваться на своих задачах, решение которых позволит успешно завершить процесс проектирования.

Основные направления и обязанности участников группы перечислены в табл. 6.1. Руководитель каждого направления ставит задачи членам своей группы и следит за их выполнением.

Табл. 6.1. Распределение обязанностей и ответственности на этапе «Планирование»

Роль	Обязанности
Менеджер продукта	Руководит сбором требований и концептуальным проектированием. Работает над планом и графиком
Менеджер программы	Управляет процессом проектирования, в частности, разработкой логической архитектуры. Создает эскиз функциональных спецификаций. Руководит всеми работами и определяет соответствие текущих результатов планам и графикам

(продолжение)

Роль	Обязанности
Разработчик	Отвечает за функциональные спецификации, относящиеся к физическому проектированию. Определяет время и трудозатраты, необходимые для разработки и стабилизации приложения, составляет план и график разработки. При необходимости проектирует концепт-систему
Инструктор	Анализирует потребности пользователей. Разрабатывает стратегию сопровождения. Оценивает проект с точки зрения удобства применения продукта. Определяет время и трудозатраты на создание систем поддержки пользователей. Выясняет, насколько удобны пользовательские интерфейсы всех компонентов приложения
Тестер	Оценивает архитектуру с точки зрения тестирования приложения. Разрабатывает план и график тестирования. Подбирает методы и метрики для выявления ошибок, вырабатывает стратегию тестирования
Логистик	Оценивает проект с точки зрения развертывания, управления и сопровождения продукта, а также его совокупную стоимость владения. Разрабатывает план развертывания и сопровождения

Таким образом, ответственный за каждое направление разрабатывает свой план и график, которые вместе составляют общий план и график проекта.

Процесс проектирования

Цель фаз «Анализ» и «Планирование» — наметить действия, необходимые для создания эффективного и полезного приложения. Частью такой работы является проектирование архитектуры приложения. Хотя модель MSF можно использовать при любом методе проектирования, наиболее полной и эффективной технологией, гарантирующей соответствие архитектуры приложения потребностям пользователей и бизнеса, считается процесс проектирования MSF. MSF позволяет рассматривать проект приложения в контексте существующей производственной архитектуры и информационной инфраструктуры предприятия, что особенно эффективно при использовании многоуровневой модели, для которой фактически и создан процесс проектирования MSF.

Так как проектирование проводится в основном на стадии «Планирование», мы немного расскажем в этой главе о процессе проектирования MSF. Мы опишем три составляющие этого процесса, уделим внимание стадии «Планирование» и покажем, как ее промежуточные результаты зависят от результатов процесса проектирования. Наконец, мы познакомим вас с тремя составляющими процесса проектирования MSF (концептуальным, логическим и физическим) и их результатами.

Стадии проектирования

Процесс проектирования MSF состоит из трех частей: концептуального, логического и физического проектирования, каждая из которых служит основой для одноименной модели.

На каждой стадии проектирования решаются конкретные задачи, а результат описывается в терминах этой задачи (табл. 6.2).

Табл. 6.2. Задачи стадий процесса проектирования MSF

Тип проектирования	Цель	Результат
Концептуальное	Учет требований пользователей и бизнеса	Описание задачи и ее решение в терминах сценариев
Логическое	Учет требований проектной группы	Описание решения в виде набора взаимодействующих сервисов
Физическое	Учет требований разработчиков	Описание сервисов и технологий, необходимых для реализации решения

Результаты, полученные в конце каждой стадии, служат исходными данными для следующего этапа процесса (рис. 6.2).

Задачи трех стадий процесса проектирования таковы.

- **Концептуальное проектирование** — исследование потребностей бизнеса и пользователей. Создание с помощью пользователей и других заинтересованных сторон сценариев, полностью отражающих требования к решению.
- **Логическое проектирование** — определение решения и организация взаимодействия его элементов. На основе сценариев, выработанных при концептуальном проектировании, формулируется абстрактная модель решения. Согласно рис. 6.2, сценарии концептуального проекта используются для создания объектов и сервисов, прототипов пользовательского интерфейса и логического проекта базы данных.

- **Физическое проектирование** — уточнение решения с учетом доступных технологий, возможностей реализации и необходимой производительности. На основе результатов предыдущего этапа — логического проектирования — на этой стадии создаются компоненты, спецификации пользовательского интерфейса и физический проект базы данных.

Рис. 6.2. Результаты трех взаимозависимых моделей процесса проектирования

Процесс проектирования MSF состоит из нескольких стадий: от конкретных требований пользователей к абстрактному (концептуальному) проекту решения, затем к логически обоснованному проекту, далее к реальному (физическому) проекту и, наконец, к готовому продукту. Концептуальное проектирование — это создание проекта приложения, то есть перевод требований пользователей на язык разработчиков. Логическим проектированием занимаются уже разработчики, преобразуя высокоуровневые концепции приложения (бизнес-требования) в детализированный проект.

При концептуальном проектировании не берутся в расчет методы и технологии, необходимые для реализации решения. Полученный таким образом проект можно сравнить с эскизом дома. Его авторы — заказчик (пользователь) и архитектор.

Логический проект более точен, в нем приложение описано подробнее. В терминах архитектуры — это планы этажей и фасада дома, На этой стадии архитекторская группа разрабатывает взаимосвязи отдельных элементов проекта и линии коммуникаций.

На стадии физического проектирования выбирается технология, которая будет использоваться для реализации приложения. Физичес-

кий проект уточняет некоторые детали архитектуры и учитывает ограничения, накладываемые существующими методами реализации.

Разные этапы модели MSF учитывают требования разных потребителей, причем функциональная целостность проекта не нарушается. Кроме того, в рамках такой модели можно применять как формальные, так и неформальные методы разработки,

Проект постепенно изменяется, с каждой стадией процесса функциональные спецификации становятся детальнее. Причем эволюцию каждого этапа проекта можно отследить, вернувшись назад вплоть до начальной постановки задачи, что иногда необходимо для проверки или тестирования

Проектирование ведется в основном на стадии «Планирование». но не ограничивается этой фазой. Проектирование начинается еще до официального старта планирования и продолжается до получения готового кода (почти до конца фазы «Разработка»). Основные положения всех трех частей процесса вырабатываются на стадии «Планирование». Как показано на рис. 6.3, многие действия при разработке выполняются параллельно, так как после получения первых результатов одной стадии сразу же начинается другая.

Рис. 6.3. В процессе проектирования различные стадии перекрываются, реализуя итерационный подход и спиральное развитие

Концептуальное проектирование начинается после выработки предварительного проекта, то есть иногда еще до достижения этапа «Одобрение концепции». Логическое и физическое проектирование выполняется параллельно с концептуальным, причем все три эти процесса относятся к стадии «Планирование» модели MSF (хотя и не ограничены ее рамками).

Из-за такой параллельности стадии проектирования — концептуальная, логическая и физическая:

- **перекрываются** — вся работа выполняется одновременно, при этом первым начинается концептуальное проектирование, затем логи-

ческое и физическое. Они выполняются не по порядку, то есть завершение одной стадии не обязательно для начала другой. В действительности логическое и физическое проектирование иногда начинают даже до выработки основных положений концептуального проекта;

- **итеративны** — каждый этап проходиг собственные циклы проверки, тестирования (не путайте с тестированием кода) и корректировки;
- **развертываются по спирали** — каждая итерация всех трех стадий процесса приближает проект к этапу «Одобрение плана проекта» фазы «Планирование».

Очевидно, что некоторое упорядочение работ все же необходимо. Концептуальное проектирование должно начинаться до логического, а то, в свою очередь, — до физического. Это касается и завершения стадий. И, наконец, результаты процесса в целом необходимы для создания функциональных спецификаций.

Концептуальное проектирование

Первая стадия процесса проектирования MSF — концептуальное проектирование — это сбор, документирование и проверка требований пользователей и выработка способов их реализации. Цель этой стадии — изучить требования бизнеса и пользователей в соответствующем контексте, а ее результат — набор информационных моделей и сценариев, документирующих текущее и будущее состояния системы.

Как мы уже говорили, концептуальное проектирование можно сравнить с первым этапом проектирования дома, когда заказчик и архитектор вместе создают эскизы. Модель концептуального проектирования облегчает сбор требований заказчика, пользователей и других заинтересованных лиц. Не имея хорошего концептуального проекта, можно разработать отличное решение, но... не той задачи.

При проектировании и разработке приложений важно, чтобы продукт «был ориентирован на пользователей», иначе говоря, полностью отвечал их требованиям. Для этого необходимо тщательно изучить их самих. Хотя обычно под сбором требований понимают составление списка пожеланий, цель концептуального проектирования — понять и документировать взаимосвязи между пользователями, приложением и бизнесом. Это жизненно необходимо, чтобы понять, что в действительности нужно пользователям,

Цели концептуального проектирования:

- **проект, учитывающий реальные требования пользователей** — для решения задачи необходимо знать о требованиях бизнеса и пользователей;

- **ясное, целостное представление о продукте** — проектная группа должна точно знать о влиянии приложения на бизнес и на пользователей;
- достаточный **уровень абстракции и классификации** — понимание потребностей пользователей в контексте их деятельности помогает исключить ненужные и лучше исследовать необходимые требования;
- **согласие заказчика, пользователей и проектной группы** — важно придти к согласию на ранней стадии реализации проекта, причем не только из-за опасности расхождения во взглядах, но и для согласованности работы всех трех сторон над созданием качественного продукта;
- согласованное **мнение группы о проекте** — это гарантирует успешную реализацию проекта;
- **проверка архитектуры приложения** — на стадии концептуального проектирования представляется первая возможность такой проверки на основе бизнес-модели информационной архитектуры предприятия;
- открытое **общение группы** — работа с заказчиком и пользователями как участниками проекта уменьшает вероятность изменений на следующих стадиях проекта.

Концептуальное проектирование состоит из трех этапов: исследования, анализа и реализации.

На стадии исследования происходит:

- поиск ответов на основные вопросы;
- выявление основных бизнес-процессов и видов деятельности;
- определение приоритетов процессов и видов деятельности;
- изучение пользователей и создание профилей.

На стадии анализа выполняется:

- более глубокое исследование пользователей и бизнеса;
- создание сценариев, описывающих процесс работы, последовательность решения задач и взаимосвязи.

На стадии реализации:

- оптимизируются технологические процессы и поддерживающие их решения;
- проверяется и тестируется пересмотренный проект.

Концептуальный проект включает основные требования к реализации и представляет собой результат работ, выполненных на стадиях исследования, анализа и реализации.

Первый этап: исследование

Главную цель исследования нужно определить до начала концептуального проектирования. Например, можно начать с описания основ-

ных бизнес-процессов соответствующей предметной области, а не с потребностей организации. Для этого надо составить:

- детальное описание основных бизнес-процессов, их границ и функциональных элементов;
- подробный перечень бизнес-операций в рамках этих процессов;
- описание заказчиков и пользователей.

Кроме того, можно добавить карты бизнес-процессов и их взаимозависимостей, информационную структуру и способы ее использования, измерения и расчеты (например, доход, приходящийся на единицу продукции, расходы, расположение производственных подразделений, прогнозы и проекты). Эта информация содержится в описании архитектуры информационных систем предприятия. При применении универсального языка моделирования или универсальной модели процесса эти данные можно увязать с моделями использования, деятельности и сотрудничества.

На этапе исследования проектная группа оценивает существующие бизнес-процессы в контексте стратегии, целей и задач предприятия. Сначала выявляются основные корпоративные процессы, определяющие цели и задачи данного бизнеса. Эти процессы могут быть связаны с производством, обслуживанием или управлением, но в большинстве случаев они пересекаются друг с другом и распределены в организации «горизонтально», а не «вертикально».

Основной процесс:

- составляет суть бизнеса;
- определяет стратегические направления развития и конкурентоспособность компании;
- его владельцы и клиенты известны;
- абсолютно понятен как заказчикам и поставщикам, так и сотрудникам данной организации;
- практически не зависит от остальных основных процессов.

После выявления бизнес-процессов и их приоритетов следует изучить корпоративную культуру данной организации, чтобы получить полное представление о ней и ее работе. При этом важно не количество информации, а ее качество.

Далее исследуют пользователей и их группы. Первым делом нужно выделить как можно больше групп — владельцы фирмы, сотрудники, клиенты и поставщики и т.п. Затем для каждой группы следует создать профиль, описывающий ее роль в организации, отдел, местонахождение, степень участия в различных видах деятельности и любую другую информацию, имеющую отношение к вашему проекту. После этого определяют связи пользователей с процессами и видами деятельности, описанными вами ранее,

Исследование завершено, если решены следующие задачи:

- определены начальные данные, необходимые для концептуального проектирования — информация о промышленной архитектуре, бизнес-процессах и пользователях, а также профили пользователей;
- собраны необходимые данные — бизнес-требования и пожелания пользователей.

Второй этап: анализ

Первая задача анализа — проверить результаты исследования, что обычно происходит на совещаниях. Все собранные данные (это могут быть как диаграммы, так и простые записи) предоставляются проектной группе, сотрудники которой интерпретируют результаты, что позволяет лучше понять требования пользователей.

Очень важно на этом этапе собрать мнение всех членов группы, особенно разработчиков, тестеров, инструкторов и логистиков. Все они в какой-то мере участвуют в проектировании и должны понять и принять мнение пользователей, внести собственные предложения и получить информацию по своим направлениям работы.

Когда результаты исследования проверены, можно приступать к построению моделей контекста, рабочих процессов и последовательностей операций. Такие модели бывают двух типов: схемы использования и сценарии.

Типичные схемы использования

Схемой использования называют зависящую от обстоятельств последовательность действий, выполненную оператором при работе с системой для получения некоторых результатов.

Пол оператором понимают человека, группу людей, другую систему или даже ее часть. Определяющей характеристикой оператора является его роль (или набор ролей) по отношению к бизнесу или системе. Схемы использования позволяют:

- выявлять бизнес-процессы и виды деятельности от начала и до конца;
- документировать контекст;
- отслеживать связи между условиями бизнеса и требованиями пользователей;
- описывать требования в соответствующем контексте;
- уточнить выполняемую задачу.

Проанализировав схему использования, вы сможете:

- связать требования бизнеса и пользователей;
- понять приложение «в общем и целом»;
- определить основу для создания сценариев «пользователь — процесс»;

- объективно и логически оценить предложения пользователей;
- сформировать функциональные спецификации;
- проследить связи между потребностями пользователей и логическим проектом.

Сценарии

Сценарий — это последовательность действий объекта и оператора.

Сценарий поясняет определенную схему использования. Он может отражать текущее состояние процесса или его тенденции. Сценарии включают четыре типа информации.

- **Контекст** — нормы корпоративной этики, правила повеления, методики, инструкции и стандарты, регламентирующие бизнес и поведение пользователя.
- **Технологический процесс** — информация о процессе описывает поток продукции и информации в рамках бизнес-процесса, между отделами организации и потребителями. Имейте в виду, что бизнес-процессы могут выходить за рамки организации,
- **Последовательность задач** — документ, в котором определены виды деятельности и задачи в рамках отдельной части процесса. Это задачи, запускающие данную последовательность, сами задачи последовательности, все ее циклы и решения, а также стандартные и нестандартные пути ее реализации.
- **Физическая среда** — данные о физических и эргономических условиях и о состоянии среды, которые могут как ограничить работу, так и способствовать ее проведению. Информация о физической среде подразумевает географические карты, списки персонала и различных ресурсов, схемы рабочей области и планы этажей, фотографии оборудования, компьютеров, мебели и осветительных приборов.

В сценариях описывается последовательность задач для конкретной роли. Но, чтобы описать все задачи, входящие в рабочий процесс конкретной схемы использования, нужно составить несколько сценариев, поскольку каждый сценарий описывает только один вариант.

Сценарии легко документировать в виде диаграмм, иллюстрирующих последовательность операций с помощью псевдокода или в описательном виде. Столь же просто создаются и прототипы для проверки.

У сценариев есть свои достоинства и недостатки. Начнем с достоинств:

- задают ориентиры для разработки;
- отображают текущее состояние среды, что необходимо пользователям и проектной группе;
- позволяют выявить дополнительные причины необходимости создания новой системы;

- помогают уяснить межсистемные зависимости.

Недостатки сценариев:

- для их разработки требуется много времени, ресурсов и средств;
- невыгодны, если решение — небольшое, или всем хорошопонятное, или не является критическим;
- иногда имеют весьма далекое отношение к проекту.

Поэтому, приступая к проекту, нужно сначала оценить как положительные, так и отрицательные стороны сценариев, исходя из затрат ресурсов на их создание. Чтобы ваши выводы были объективнее, воспользуйтесь оценкой рисков, задав себе вопросы «Насколько велик риск потерять время, создавая сценарии?» и «Какова вероятность непонимания из-за того, что сценарии не созданы?».

Стадия анализа завершена, если решены следующие задачи.

- Собраны пользовательские и бизнес-данные, необходимые для формирования сценариев, включая информацию о контексте, процессах, последовательности задач и физической среде.
- Созданы сценарии, которые проектная группа считает приемлемыми.

На этом этапе сценарии отражают текущее положение вещей. О том, как их изменить, чтобы описать будущее состояние, мы расскажем в следующем разделе.

Третий этап: рационализация

Цель данной стадии — сделать бизнес-процессы частью проекта и внести улучшения там, где это возможно. Ведь концептуальное проектирование охватывает не только изучаемую систему, но и бизнес-процессы, информацию и задачи новой системы.

Помимо проектной группы, в работе по созданию и уточнению проекта могут участвовать уже вовлеченные в работу над ним сторонние специалисты и специально приглашенные консультанты. Однако в зависимости от масштаба проекта и степени понимания особенностей бизнеса группа оптимизирует некоторые процессы без посторонней помощи.

Что же стоит оптимизировать? Первым делом попытайтесь отказаться от:

- непроизводительных операций;
- «узких» мест и ненужных работ;
- избыточных и неэффективных методов и процессов;
- ненужной бумажной работы;
- неконструктивных правил;
- потерь времени.

Работу нужно начинать с вопросов «Каким образом улучшить производительность?», «Что надо оптимизировать?», «Можно ли интегрировать процессы?». Однако недостаточно просто выявить слабые области, необходимо продумать и описать то, какой вы хотите видеть систему. А уж потом можно создать новые сценарии.

Ниже приведены несколько правил, которые помогут вам оптимизировать проект.

- **Разрушайте традиции** — подвергайте все утверждения сомнению. Спросите себя, зачем нужно, например, чтобы заявка обязательно утверждалась на уровне группы, отдела и подразделения?
- **Ориентируйтесь на поставленные цели** — убедитесь, что задачи. поставленные в начале проекта, соответствуют его целям. Считайте ориентиром выполнение требований заказчика в целом, а не только его краткосрочных потребностей.
- **Планируйте работу над продуктами и сервисами** — направляйте действия людей на решение всех поставленных задач, а не одной-единственной.
- **Устраняйте бюрократические и другие препятствия** — замените стандартную иерархическую систему независимыми группами, работающими параллельно.
- **Повышайте производительность** — избегайте разделения работы на части по принципу узкой специализации — наоборот, обобщайте и укрупняйте все ее задачи.
- **Выясните, где можно применить технику** — выявите те работы и направления, которые проще выполнить с использованием техники.
- **Упрощайте анализируемые процессы с самого начала** — разбейте всю задачу на несколько этапов меньшего масштаба, которые можно изучать последовательно.

После того как новые сценарии подготовлены, наступает время их проверки — нужно выяснить, решают ли они поставленные бизнес-задачи. Для этого проектная группа должна:

- создать прототип системы:
- представить проект пользовательского интерфейса;
- получить от пользователей предложения по усовершенствованию системы;
- повторить все сначала, пока пользователи и заказчики не останутся довольны.

На ранней стадии процесса проектирования пробные системы должны обладать только основными функциями — например, пользовательским интерфейсом. На этом этапе группа разработчиков может быстро изменить проект, так как для этого не нужно много усилий. Для проверки концепций проекта стоит применять:

- приложения, отражающие основные функции системы;
- различные документы (как на бумаге, так и в электронном виде);
- изложенные на бумаге прототипы структуры системы и ее взаимодействия с пользователем;
- слайды (например, сделанные с помощью Microsoft PowerPoint), иллюстрирующие основные элементы системы и демонстрирующие работу системы при выполнении каких-либо задач.

По мере проектирования прототипы уточняются, и появляется представление о внешнем виде приложения, в частности о пользовательском интерфейсе.

На стадии проверки вместо документов, описывающих требования, лучше применить сценарии, так как они более информативны, К тому же сценарий очень просто проверить методом пошагового выполнения, ролевой игры или проверки корректности концепции. Главная задача данного этапа — выявить пропущенные или неправильно интерпретированные цели проекта до окончания работы с пользователями. Кроме того, иногда сценарий позволяет выявить разные точки зрения пользователей на какое-либо решение и принять соответствующие меры,

Главное — не пытаться переделать весь бизнес-процесс еще в первом выпуске приложения. В следующих версиях вы сможете добавить новые сценарии, обеспечивающие дальнейшую модернизацию бизнес-процесса. Пользователи часто не приемлют коренного изменения привычных методов работы, поэтому постепенная реализация новых сценариев иногда гораздо удачнее.

Рационализация завершена, если решены следующие задачи:

- созданы сценарии, позволяющие улучшить работу и описывающие тенденции развития;
- сценарии проверены и уточнена информационная архитектура.

Логическое проектирование

Логическое проектирование — это процесс описания решения в терминах организации, структуры, синтаксиса и взаимодействие его частей с точки зрения проектной группы.

Его цель — применить принципы модели MSF, описанные в главе 2, и изучить структуру приложения и взаимодействие его частей. Результат данной стадии — набор бизнес-объектов с соответствующими сервисами, атрибутами и взаимосвязями; детальный проект пользовательского интерфейса и логический проект базы данных.

Логическое проектирование можно сравнить со вторым этапом проектирования дома: архитектор создает план этажей и фасада, определяет разные архитектурные элементы — двери, окна, крыши.

форму стен и пространственные взаимосвязи между ними; в результате получается гармоничное целое.

Логический проект — это решение с точки зрения проектной группы, представляющее поведение решения и его логическую организацию; они понадобятся на стадии физического проектирования. Логический проект помогает уточнить требования концептуального проекта и несколько упростить решение.

На стадии логического проектирования:

- приложение упрощается благодаря определению его структуры, описанию частей системы и их взаимодействия;
- устанавливаются границы и описываются интерфейсы, обеспечивающие организационную структуру взаимодействия между компонентами;
- выявляются все ошибки и несообразности концептуального проекта;
- устраняется избыточность и определяются части проекта, которые можно использовать повторно;
- закладывается фундамент физического проекта;
- улучшается работа различных частей приложения и приложения в целом;
- все члены группы приходят к единому мнению относительно проекта приложения.

Обратите внимание, что логический проект не зависит от его физической реализации. Он описывает, как *должна* работать система. Очень важно полностью понять решение еще до выбора реализующей его технологии.

Логическое проектирование состоит из двух этапов: анализа и рационализации. (Нет стадии исследования, так как начальными данными для логического проектирования служат результаты концептуального.)

На стадии анализа:

- выявляются бизнес-объекты и сервисы;
- определяются их атрибуты и взаимосвязи.

На стадии рационализации:

- бизнес-объекты проверяются;
- выявляются неявно использованные или дополнительные бизнес-объекты и сценарии.

Первый этап: анализ

Назначение этого этапа — преобразовать сценарии концептуального проекта в модули, применяемые в логическом проектировании. *Модули* — это основные схемы использования системы, работающие в их рамках сервисы и виды деятельности, а также связи между ними. Они

являются логическими блоками, абстрагирующими схемы использования и сценарии концептуального проекта. Для каждого модуля проектная группа выявляет сервисы, объекты, атрибуты и взаимосвязи.

Для выявления сервисов, бизнес-объектов, атрибутов и взаимосвязей проектная группа изучает описанные в сценариях процессы и последовательности задач, обращая особое внимание на:

- действия (выражаются глаголами) — это сервисы:
- субъекты и объекты (выражаются существительными) — это бизнес-объекты;
- атрибуты — то, как они связаны со свойствами:
- взаимосвязи, которые определяются свойствами.

Кроме того, важный источник информации — текущая ситуация и физическая среда.

Сервисы

Сервис — это элементарная единица приложения, реализующая операцию, функцию или преобразование. Сервисы состоят как из алгоритмически простых функций, например «Создать» или «Удалить», так и из выполняющих сложные вычисления и преобразования, например, «Оплата», «Проверка» или «Бронирование».

Возможности сервиса необходимо сформулировать как можно шире, причем только глаголами, описывающими действия. Его следует назвать ясно и недвусмысленно. Если вам трудно подобрать название сервиса, значит, его назначение недостаточно понятно и нужно провести дополнительное исследование в рамках концептуального проектирования.

В табл. 6.3 показана последовательность задач и реализующие ее сервисы.

Табл. 6.3. Пример последовательности задач и реализующих ее сервисов

Последовательность задач	Сервис
Клерк гостиницы **проверяет** забронированные комнаты	Просмотреть забронированные комнаты
Система **находит** комнату, **предназначенную** для нового жильца	Найти комнату Назначить комнату
Клерк гостиницы **выдает** ключи	Выдать ключи

Бизнес-объекты

Бизнес-объект инкапсулирует сервисы и данные, используемые при составлении и упрощении решения. Такие объекты — люди или вещи,

описанные в сценариях. С ними связаны атрибуты и взаимосвязи. (Обратите внимание, что это определение отличается от определения COM-объектов, которые являются экземплярами соответствующих классов.)

Некоторые бизнес-объекты не могут быть явно упомянуты в сценарии, хотя их наличие необходимо для выполнения описанных в нем действий. Такие объекты часто называют нефункциональными требованиями. Чтобы выявить бизнес-объекты, нужно изучать структуры, другие системы, устройства, существующие вещи или события, не представленные в сценарии физически, исполняемые роли, местонахождение и отделы организации. Чтобы найти недостающие объекты, проектная группа должна изучить процессы, с которыми не связан ни один объект.

В табл. 6.4 показаны последовательность задач из табл. 6.3 и связанные с ней бизнес-объекты.

Табл. 6.4. Пример последовательности задач и связанных с ней бизнес -объектов

Последовательность задач	Бизнес-объект
Клерк **гостиницы** проверяет **список забронированных комнат**	Клерк гостиницы Список забронированных комнат
Система находит комнату, предназначенную для нового жильца	Система Комната Жилец
Клерк гостиницы выдает ключи	Ключи

Атрибуты

Атрибуты (в COM называемые свойствами) — это элементы объекта, о которых бизнес-процесс должен знать и отслеживать их; иначе говоря, это определение объекта. Каждый экземпляр объекта содержит свой набор значений, основанных на соответствующих определениях. Например, атрибут «Имя» в некотором экземпляре объекта может принимать значение «Джон». Атрибуты также иногда показывают, что один объект владеет другим. Набор значений атрибутов объекта называют его *состоянием.*

При изучении атрибутов важно найти те из них, которые можно получить — например, вычислить — из других атрибутов. В таком случае получение атрибута становится сервисом, а соответствующие вычисления — частью его интерфейсного контракта. Например, если атрибут «Количество забронированных комнат» объекта «Список забронированных комнат» является производным от объекта «Система

бронирования», то в объекте «Система бронирования» должен быть определен сервис «Подсчитать количество забронированных комнат».

Проектная группа должна изучить полный набор атрибутов объекта и, если один из атрибутов сильно отличается от остальных, ввести в оборот новый объект.

Заметьте, что наше определение звучит так: атрибуты — это свойства, о которых бизнес должен знать и отслеживать их. Другими словами, некоторые атрибуты существуют, но не относятся к выполняемой задаче. Все такие атрибуты должны быть вам известны на тот случай, если они понадобятся. (Кроме того, некоторые необходимые сейчас атрибуты могут стать ненужными в будущем в связи с изменением процессов.)

В табл. 6.5 показана та же, что и в предыдущих таблицах, последовательность задач и ее атрибуты. Не относящимися к делу атрибутами можно считать, например: возраст, рост, вероисповедание, национальность и номер страховки. Заметьте, что данный объект способен находиться в разных состояниях.

Табл. 6.5. Пример последовательности задач и ее атрибутов

Примерная задача	Атрибут	Значения в некотором состоянии
У жильца есть имя и адрес	Имя Фамилия Адрес	Дэн Шелли 100 Microsoft Way
Жилец определяет тип заказа	Тип	Gold Club
Жилец работает в некоторой компании (важно для некоторых типов заказов)	Компания	Microsoft
Жилец может быть **курящим или некурящим**	Некурящий/курящий	Некурящий

Взаимосвязи

Взаимосвязь — это логическая связь, объединяющая объекты. Для эффективного проектирования и сборки частей системы в единое целое необходимо определить все взаимосвязи.

Взаимосвязи показывают, как объекты ведут себя по отношению друг к другу. Каждая взаимосвязь может быть одного из трех типов: целое/часть, взаимодействие, обобщение/специализация. При их поиске следует принять во внимание взаимосвязи всех типов.

Как показано на рис. 6.4, объект *жилец* можно выделить из более общего объекта *человек,* он может быть частью *списка жильцов* и жить в некоторой *гостинице.*

Рис. 6.4. Определение взаимосвязей

Подведем итог: на стадии анализа определяются объекты, сервисы, атрибуты и взаимосвязи, входящие в сценарии. Этот процесс изображен на рис. 6.5. Отметим, что в некоторых книгах такой процесс называют *анализом «существительное — глагол»* (или *«объект — действие»*).

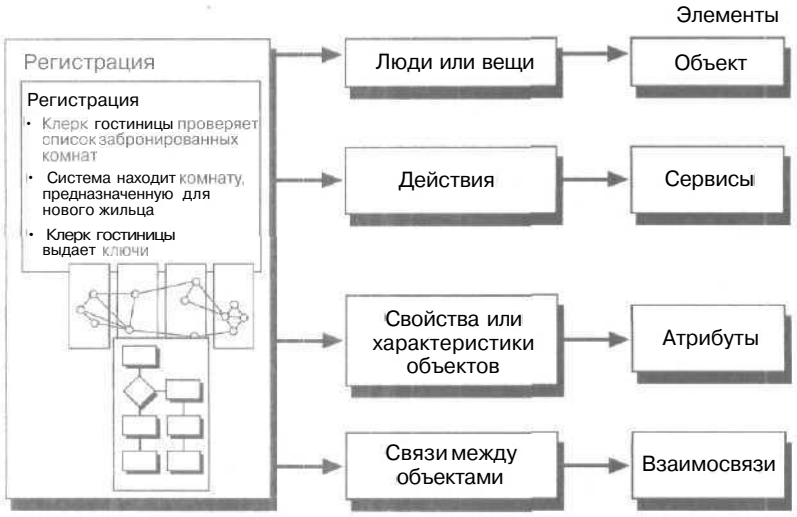


Рис. 6.5. Процесс идентификации

Стадия анализа завершена, когда решены следующие задачи:

- выявлены сервисы и подготовлен их список;
- выявлены объекты и подготовлен их список;
- определены атрибуты и подготовлен их список;
- определены взаимосвязи и подготовлен их список.

Второй этап: рационализация

Главная задача рационализации — создать те сервисы и объекты, которых не было на предыдущих стадиях проектирования. Они нужны, потому что предполагается их существование или они потребовались для контроля. Затем проектная группа «вычищает» и проверяет проект.

Дополнительные сервисы и объекты

На этом этапе создаются объекты и сервисы, которых не было на предыдущих стадиях проектирования. Например, жилец (соответствующий объект) работает в какой-либо компании (также объект), но связь между ними существует не всегда (ведь он не обязан пользоваться кредитной картой данной компании), В этом случае можно создать объект «тип пребывания», который эффективно заменит отсутствующий неявный объект и сохранит взаимосвязь между жильцом и его компанией, если таковая существует.

Контроль

Проектная группа не должна забывать и о *контроле*. Он позволяет удостовериться, что сервисы работают в правильном порядке и в нужное время. В распределенных системах для упорядочения сервисов и взаимосвязей объектов применяют транзакции. В других случаях для проверки целостности сценария, координации сервисов и управления связями объектов используют контрольные объекты.

Например, сценарий регистрации в гостинице — транзакция или, что то же самое, набор взаимосвязанных объектов. Гарантировать учет взаимосвязей и в случае разрушения какой-либо из них исправить ее (например, отменить бронирование комнаты) позволит специальный объект или сервис.

Поэтому проектная группа обязательно должна учитывать необходимость дополнительных бизнес-объектов, отвечающих за контроль, упорядочение выполнения и зависимости. Кроме того, можно отделить стабильные объекты и сервисы от часто меняющихся.

«Очистка» объектов

После создания объектов нужно «очистить» проект от «мусора»:

- уничтожить объекты, не относящиеся к поставленной задаче;
- объединить избыточные объекты;
- выявить неявно используемые объекты;
- разделить атрибуты и объекты;
- рассмотреть управление транзакциями;
- отделить роли и их исполнителей от объектов.

После первого этапа специализации объектов с использованием анализа «существительное — глагол» требуется «очистка», так как не

все существительные адекватно описывают решение. Те из них, что останутся, должны относиться к физическим лицам (например, клиенты, служащие), к оборудованию или к концепциям бизнес-транзакций (например, выделение комнат, авторизация платежей или продвижение по службе).

В процессе «очистки» можно использовать сервисы. Нужно проверить объекты, задавая вопросы «На что воздействует этот сервис?» и «Отвечает ли кто-нибудь за выполнение данного сервиса?». Объекты без сервисов или только с одним сервисом требуют дальнейшего исследования.

Проверка

Проверка — это тестирование полноты и корректности проекта на уровне объектов. При этом объекты проверяются как по отдельности, так и в совместной работе. Тестирование независимых объектов упрощает задачу их интеграции в единое целое, так как их тщательно проверили еще до сборки. Тестирование взаимодействия объектов гарантирует, что работа, предписанная сценарием, будет выполнена. ,

Рис. 6.6. Связь объектов со сценариями

Проектная группа должна изучить предусловия и постусловия для каждого отдельного объекта. На этом этапе важен ответ на вопрос: «Даны некоторые предварительные условия. Соответствуют ли в этом случае постусловия требованиям сценария?»

Однако такое тестирование не затрагивает сборки. Поэтому проектная группа должна рассмотреть различные комбинации объектов, которые способны решить задачи, описанные в сценариях. Причем набор таких взаимозависимых модулей иногда очень сложен. Существует простой способ проверки сценариев, оперирующих большим количеством объектов, — выполнить сценарий полностью, убедившись, что некоторый набор объектов удовлетворяет всем его требованиям.

Проектная группа может выполнить сценарий и определить, какие сервисы и в какой последовательности нужны для успешного его завершения. Выявление необходимых для работы сценария сервисов следует начинать с первого обращения к объекту.

Стадия рационализации завершена, когда решены следующие задачи:

- выявлены неявно используемые объекты и сервисы и проведена «очистка» проекта;

- проверены объекты и сервисы, проведена «очистка» объектного модуля.

Физическое проектирование

Физическое проектирование — это процесс описания компонентов, сервисов и технологий, используемых для получения решения. Его цель — сопоставить логический проект с рамками существующих технологий, изучить возможности реализации проекта и производительность приложения. Результат этого этапа проектирования — набор компонентов, проект пользовательского интерфейса для определенной платформы и физический проект базы данных.

Физическое проектирование можно сравнить с третьим и последним этапом проектирования дома: архитектурный план уже готов, и разработчик намечает схемы электропроводки, водопровода, отопительной системы и вентиляции.

На стадии физического проектирования проектная группа:

- классифицирует требования (таким образом упрощается сегментация и оценка работы, необходимой для создания системы) и полностью разъясняет их разработчикам;
- связывает логический проект и его реализацию, описывая решение в терминах реализации;
- оценивает разные варианты реализации;
- создает гибкий проект на основе сервисов;
- добивается совместимости с производственной архитектурой;
- проверяет соответствие логического проекта схемам использования и сценариям.

Физический проект — это основа функциональных спецификаций, которые необходимы для контроля качества. Он очень важен, так как предоставляет разработчикам последний шанс проверить проект до написания кода. Когда код уже создается, вносить изменения в проект, конечно, все еще можно, но при этом значительно возрастают затраты (как времени, так и ресурсов).

Физическое проектирование завершается в тот момент, когда собрано достаточно информации для начала разработки или развертывания. Поэтому эти процессы начинаются до завершения физического проектирования; более того, физическое проектирование может продолжаться и в начале стадии «Разработка», что дает дополнительные преимущества в «очистке» проекта. Единственное условие — окончательный проект должен быть утвержден до стабилизации решения.

Результаты физического проекта служат исходным документом для других задач фазы «Планирование». Они:

- требуются для примерных расчетов стоимости проекта, его графика и необходимых ресурсов;
- позволяют распределить проектные работы по промежуточным этапам и внутренним выпускам модели процесса разработки MSF;
- нужны для анализа рисков на ранней стадии.

Заметим, что основные документы модели физического проектирования служат исходными данными для подготовки результатов этапа «Одобрение плана проекта»:

- спецификации топологии и компонентов становятся частью функциональных спецификаций;
- оценки и задачи входят в основные план и график проекта;
- оценки риска входят в основной документ оценки рисков.

При разработке, пересмотре и утверждении основного плана и графика проекта возникают вопросы, требующие изменения физического проекта. Например, при ограничении времени и ресурсов иногда приходится изменять характеристики и функции приложения.

Стадия физического проектирования в модели MSF состоит из четырех этапов: исследования, анализа, рационализации и реализации, (Исследование позволяет выявить технологии, которые можно использовать в приложении.) Каждый этап заканчивается конкретными результатами; единственное исключение — этап реализации, результаты которого входят в окончательный физический проект,

На стадии исследования:

- определяются физические ограничения инфраструктуры;
- определяются физические требования к решению;
- изучаются риски, возникающие в результате конфликта между требованиями и физическими ограничениями.

На стадии анализа:

- выбираются возможные технологии реализации;
- создается эскиз модели развертывания, включающий сетевые и компонентные технологии, а также технологии данных.

На стадии рационализации:

- определяется способ комплектации и развертывания;
- выполняется декомпозиция объектов на компоненты и сервисы;
- компоненты распределяются в соответствии с топологией;
- происходит «очистка» плана комплектации и развертывания.

На стадии реализации:

- выбирается программная модель;
- определяется интерфейс компонентов;
- компоненты описываются на языке разработки;
- изучается структура компонентов.

Результат исследования, анализа, рационализации и реализации — базовый физический проект, включающий основные спецификации.

Шаг 1: исследование

Цель исследования:

- выявить физические ограничения инфраструктуры и условий, в которых будет эксплуатироваться приложение;
- по возможности избежать рисков, связанных с конфликтом между ограничениями и условиями работы.

К этому этапу разработки уже должны быть известны требования, предъявляемые к приложению, — сколько людей с ним будут работать, сколько транзакций в день оно будет обслуживать и т. д. Чтобы изучить требования подробнее, могут понадобиться специально созданные прототипы приложения.

Физические ограничения определяются информационной архитектурой организации, которая, как мы уже говорили в главе 1, не всегда хорошо документирована. Если это так и в вашем случае, то стадия исследования — отличная возможность для ее изучения.

При составлении списка требований и соответствующих им ограничений нужно изучить проблему с точки зрения приложения и с точки зрения инфраструктуры.

Такой метод позволит вам ничего пропустить. Вот какие данные включаются в список требований и ограничений:

- производительность;
- затраты и прибыль;
- особенности развертывания;
- возможности сопровождения;
- выбранные технологии:
- надежность;
- пригодность;
- безопасность,

Текущая физическая инфраструктура исследуется с помощью *топологий* — карт, отражающих различные ее особенности. При планировании приложения на физическом уровне иногда полезна топология сети, данных и компонентов (их примеры приведены в следующем разделе, посвященном анализу). Топология инфраструктуры, созданная на этапе исследования, отражает ее *текущее* состояние.

Далее нужно изучить риски, возникающие из-за конфликтов или расхождений трех топологий. Эта процедура разбивается на пять этапов.

1. **Выявление конфликтующих требований инфраструктуры и условий работы приложения.** При сравнении физических требований и ограничений вы заметите, что одни из них конфликтуют, другие -

нет, а некоторые вообще несущественны. Например, инфраструктура требует использования TCP/IP, а приложение работает по другому протоколу (конфликт или недостающий элемент), TCP/IP (нет конфликтов и недостающих элементом) или ему все равно, по какому протоколу работать (несущественное требование).

2. **Предварительная оценка рисков для выявления конфликтов и недостающих элементов.** Что имеет место — недостающий элемент (отсутствие какого-либо из пунктов списка, что в дальнейшем может быть исправлено) или конфликт (появляется необходимость выбора альтернативы)? Насколько значительны недостающий элементы и конфликты?
3. **Расстановка приоритетов для определения важности различных элементов исследуемого проекта.** Какие требования необходимы? Какие ограничения возникают из-за промышленной архитектуры? На данном этапе следует изучить *бизнес-процессы* с точки зрения требований и ограничений.
4. **Выработка предварительных вариантов решения проблемы.** При возникновении конфликтов или недостающих элементов есть только два пути:
   - не исправлять их (продолжать работу и посмотреть, что произойдет);
   - изменить либо требование, либо ограничение,
5. **Выявление рисков, связанных с упомянутыми в списке ограничениями и условиями работы.** На этом этапе важно не допускать ошибок, иначе возникнут новые непредвиденные вопросы,

Результат этой работы — предварительная оценка рисков и план по их сокращению, которые войдут в основной документ оценки рисков.

Стадия исследования завершена, когда выполнены следующие задачи:

- выявлены ограничения и условия работы;
- определены топологии сети, данных и компонентов, а также физические требования приложения;
- изучены риски, возникающие при конфликтах и недостающих элементах в ограничениях и требованиях;
- риски оценены и составлен план их сокращения.

Шаг 2: анализ

Главная цель этой стадии — выбрать технологии для реализации, основываясь на требованиях к приложению. Все эти технологии не обязательно используют при разработке — окончательный выбор делается позже, Но когда список возможных технологий определен, можно начинать разработку модели развертывания.

При анализе технологий в первую очередь изучают вопросы бизнеса, затем вопросы промышленной архитектуры и, наконец, сами технологии.

К вопросам бизнеса относятся:

- **возможности** — действительно ли данная технология удовлетворяет потребностям бизнеса;
- **стоимость продукта** — полная стоимость продукта (не забудьте о стоимости лицензий и обновлений);
- **квалификация пользователей** — выясните, нужно ли обучение (стоимость и время) или консультации (затраты и доступность) и какой уровень приемлем;
- **модернизация существующего продукта или разработка нового** — важно принимать решения, основываясь на информации о рисках. Зрелый продукт принят рынком, он ясен, у него есть история, он стабилен и, наконец, к нему уже выпушено множество справочных и обучаюших материалов. Инновационный продукт всегда «самый новый и лучший», однако часто опережает свое время. В идеале продукт должен быстро проходить все циклы развития, чтобы всегда соответствовать современным технологиям;
- **сопровождение** — это касается как технологии, так и построенного на ее основе решения. Что это означает для проекта и предприятия? Например, чтобы сопровождать продукт, необходимы соответствующие людские и финансовые ресурсы.

Кроме того, следует рассмотреть вопросы развертывания, конкурентоспособность продукта и время выхода на рынок.

К вопросам производственной архитектуры относятся:

- **соответствие целям архитектуры** — приложение должно учитывать цели и принципы архитектуры. Все его задачи основаны на четырех моделях архитектуры: бизнеса, приложения, информации и технологии;
- **строгое соответствие архитектуре** — архитектура предприятия определяет планы для текущего и будущего состояния системы. Приложение должно соответствовать моделям приложения, информации и технологии архитектуры предприятия;
- **возможности роста** — масштабируемость приложения следует рассматривать как с точки зрения положения на рынке, так и с точки зрения увеличения сегмента рынка, занятого данным продуктом. При этом, возможно, разрастется и сама компания;
- **взаимодействие** — приложение должно быть совместимо с другими информационными системами организации. Не мешать работе других систем недостаточно — должен быть определен комму-

никационный интерфейс, посредством которого новое приложение могло бы взаимодействовать с другими приложениями.

К технологическим вопросам относятся:

- **языки программирования** — выбирая язык для разработки компонентов, рассмотрите разные варианты для разных задач проекта, выясните, поддерживает ли язык программирования реализацию взаимосвязанных компонентов, которые при необходимости придется заменять и обновлять;
- **стандарты взаимодействия компонентов** — платформы и стандарты взаимодействия связаны друг с другом. Выбирая стандарт взаимодействия (способ «общения» компонентов), изучите возможности кросс-платформенной интеграции с точки зрения производительности приложения. На этом этапе стоит рассмотреть несколько технологий;
- **методы доступа к данным** — выбирая способ взаимодействия компонентов с хранилищами данных, изучите особенности производительности, стандартизации и перспектив метода. Также учтите разнообразие поддерживаемых хранилищ данных и управление доступом к ним. Выбирая способ хранения данных, принимайте решение, учитывая структуру и местонахождение информации;
- **системные сервисы** — поддерживают инфраструктуру для распределенного решения. Выявите типы сервисов, необходимых для построения решения, и определите, какие из них обеспечиваются штатными системными средствами;
- **операционные системы** — обратите внимание на сервисы операционной системы — их использование может значительно упростить разработку приложения. Кроме того, операционная система должна обладать достаточным уровнем защиты и масштабируемости; но не забывайте, что в разных операционных системах — разные методы доступа к их сервисам,

Теперь проектная группа, учитывая перечисленные технологии, может составить предварительную модель развертывания, состоящую из топологий сети, данных и компонентов. На этой стадии физического проектирования все разработанные топологии носят предварительный характер — они еше не утверждены.

Сетевая топология — это инфраструктурная карта, где отмечено расположение оборудования и указаны связи между отдельными устройствами. Такую карту, отражающую текущее состояние, стоит составить еше на стадии исследования. Сейчас же, возможно, потребуется ее изменить в соответствии с новым физическим проектом, На рис. 6.7 показан пример сетевой топологии.

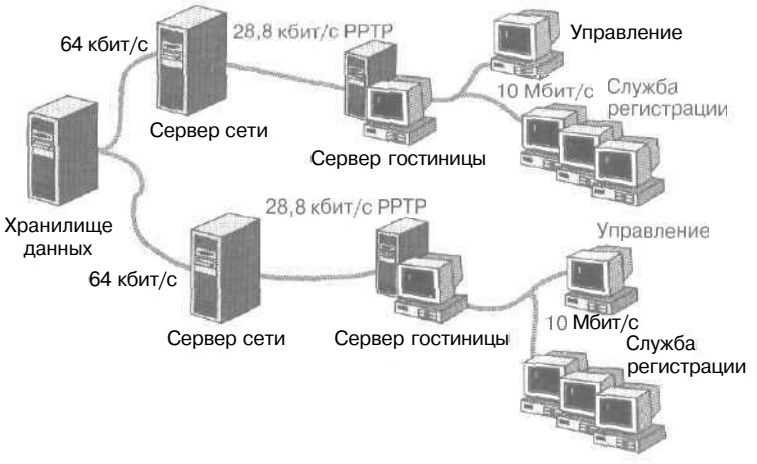


Рис. 6.7. Пример топологии сети

Топология данных — это карта размещения данных, где отмечено местонахождение хранилищ данных в соответствии с топологией сети. Как и для сетевой топологии, карта, отражающая текущее состояние, разработана еще на стадии исследования, но из-за необходимости применения новых стратегий или технологий развертывания данных, возможно, потребует изменений в соответствии с новым физическим проектом. На рис. 6.8 показан пример топологии данных,

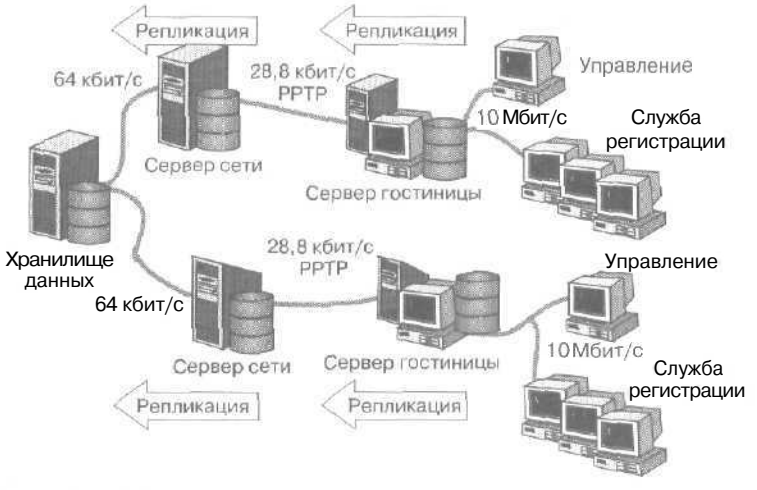


Рис. 6.8. Пример топологии данных

Топология компонентов — это карта, где отмечено местонахождение самих компонентов и их сервисов в соответствии с топологией сети. К этому моменту уже должна существовать текущая версия данной топологии, составленная на стадии исследования. Теперь же нужно нанести на нее новые компоненты и сервисы, необходимые приложению, и обновить ее в соответствии с изменениями других топологий. На рис. 6.9 показан пример топологии компонентов.

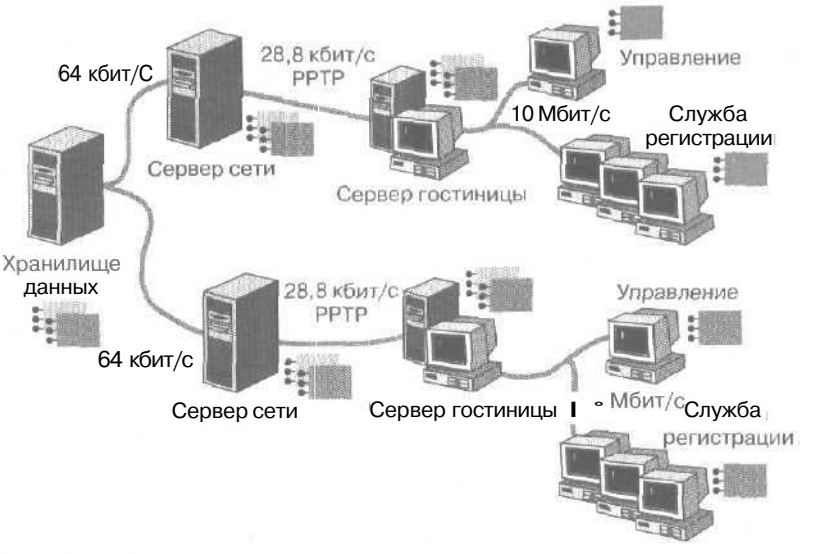


Рис. 6.9. Пример топологии компонентов

Стадия анализа завершена, когда решены следующие задачи:

- составлен список используемых технологий;
- создана предварительная модель развертывания, включающая предполагаемую топологии сети, данных и компонентов.

Шаг 3: рационализация

К настоящему моменту проведено исследование (и даже проанализированы его результаты), поэтому пора утверждать проект. Однако прежде надо определить стратегию развертывания и комплектации.

Вы можете подумать: «Нельзя же планировать комплектацию и развертывание, когда еще не готова модель компонентов», — и окажетесь правы. Но именно по этой причине нельзя закончить работу над моделью компонентов без ясного плана комплектации и развертывания. Задачи стадии рационализации — интерактивные и итерационные. В некоторых случаях имеет смысл начать с составления модели компонентов, но иногда стоит прежде изучить комплектацию.

Единственное, что можно посоветовать, — начните с той задачи, которая кажется вам самой сложной и наиболее важной.

Определяя стратегию комплектации и развертывания, придерживайтесь трех основных принципов. Во-первых, изучите разные способы комплектации и их обоснования или причины, по которым следует выбрать какую-то конкретную стратегию. Это может быть:

- тип сервиса;
- масштабируемость;
- производительность:
- управляемость;
- возможность повторного использования;
- бизнес-контекст;
- степень детализации.

Во-вторых, согласуйте стратегию с моделью программирования. Этот процесс — интерактивный, так как на данном этапе модель программирования еще не ясна.

В-третьих, выявите возможные изменения проекта, способные повлиять на стратегию; при этом учтите причины, изученные на первой стадии создания этой стратегии. Например, если в основу вы положите производительность приложения, это не лучшим образом отразится на его масштабируемости. Этот конфликт придется устранять.

Теперь проектная группа готова выбрать компоненты. Хотя для этого разработано много методик, мы советуем вам сначала обратиться к трем логическим сервисным уровням модели MSF: пользовательскому, прикладному и уровню данных. Как показано на рис. 6.10, сервисы распределены в соответствии с топологией. Компоненты, размещенные в одном и том же месте, можно собрать в группы.

После завершения построения модели компонентов проектная группа готова и к их развертыванию. Повторим, что развертывание основано на трех уровнях сервисов: пользовательском, прикладном и уровне данных. Причем их применение не предполагает развертывания компонентов в трех разных местах — можно создать многоуровневое приложение, работающее на одном компьютере. По этой же причине стратегия развертывания сложного приложения, распределенного по разным организациям, иногда включает 10, 20 и даже больше пунктов развертывания.

На стадии рационализации нужно проверить предполагаемые компоненты, их комплектацию и развертывание и ответить на вопросы: «Соответствуют ли компоненты стратегии комплектации?», «Соблюдены ли требования к приложению и цели проекта?», «Соответствуют ли компоненты логическому проекту?» и «Соответствует ли весь проект архитектуре предприятия?».

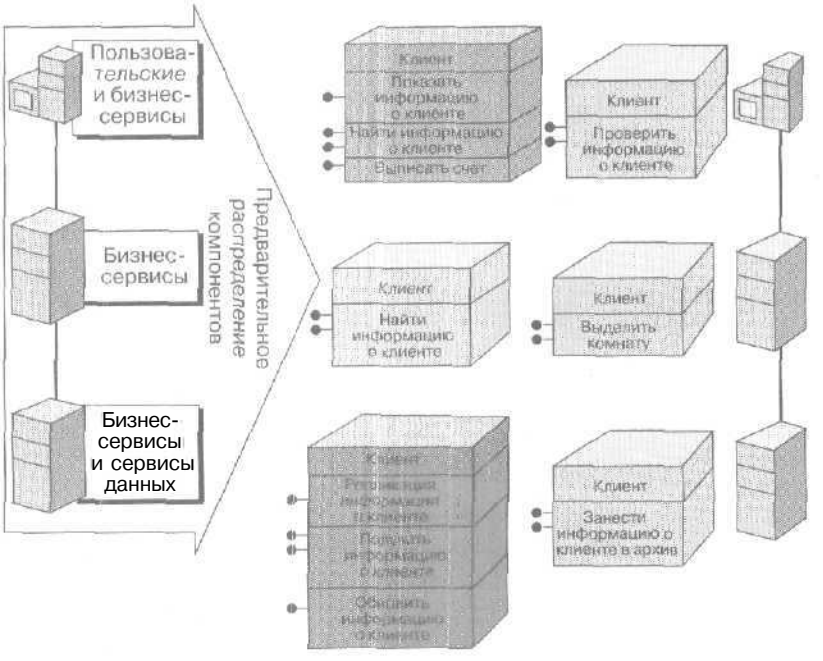


Рис. 6.10. Пример модели компонентов

Для проверки проекта применяют прототип приложения или его пробную версию. Причем по мере появления новых компонентов их нужно включать в прототип и изучать его работу.

Хотя проверка проводится в течение всего этапа рационализации, в конце стоит выполнить окончательную проверку и «очистку» компонентов. За этот процесс отвечают разработчики и тестеры. В некоторых случаях требуется проверить качество компонентов; для независимой экспертизы иногда привлекается сторонний специалист,

На что же такой эксперт должен обратить внимание? План считается удачным, если:

- его компоненты содержат все сервисы, присутствующие в логическом проекте;
- учтены все взаимосвязи компонентов и сервисов;
- развертывание компонентов по топологиям соответствует их особенностям;
- набор компонентов характеризуется сильными элементными и слабыми межкомпонентными связями;

- сервисы сгруппированы в компоненты с учетом баланса их местонахождения, особенностей комплектации и технологических ограничений;
- в компонентном плане учтены масштабируемость сверху вниз и снизу вверх (обе очень важны!).

Элементные и компонентные связи

Одна из целей проектирования — добиться сильных элементных и слабых компонентных связей. *Элементная связь* — это связь между элементами компонента, а *компонентная* — связь между разными компонентами.

О компоненте, чьи внутренние элементы — обычно это сервисы — тесно связаны друг с другом, говорят, что у него сильная элементная связь. В зависимости от причины связь может быть полезной или нет.

Полезными считаются связи;

- **функциональные** — компонент выполняет только одно действие. Это самая сильная элементная связь;
- **последовательные** — компонент содержит операции, которые должны выполняться в определенном порядке и использовать одни и те же данные;
- **коммуникационные** — операции совместно используют данные, но никак друг с другом не связаны. Такой тип элементной связи минимизирует накладные расходы системы на связь;
- **временные** — операции объединены, так как выполняются одновременно,

Некоторые связи бесполезны, а иногда и вредны:

- **процедурные** — операции сгруппированы, так как выполняются в определенном порядке. В отличие от последовательных связей, операции не используют одни и те же данные;
- логические — несколько операций сгруппированы в один модуль, которому требуется управляющий флаг (например, в случае сильно вложенных операторов IF или CASE), передаваемый ему для выполнения определенной функции. Операции сгруппированы искусственным образом;
- **случайные** — операции сгруппированы без видимых связей друг с другом.

Для определения значимости компонентных связей можно использовать следующие параметры:

- **размер** — минимизируйте количество связей и сложность интерфейса;
- **близость** — используйте прямые связи;

- видимость — определяйте все связи явно;
- гибкость — используйте независимые интерфейсы.

Чтобы достичь сильных элементных и слабых компонентных связей, постарайтесь создавать независимые компоненты. Сильные элементные связи позволяют лучше определить функции и поведение компонентов. Полезно организовать сервисы по их назначению, чтобы компоненты содержали только те из них, которые имеют отношение к их работе. Свободные компонентные связи обеспечивают большую гибкость и независимость, что приводит к созданию ясных и простых интерфейсов. Удачный метод — организовывать связи между компонентами так, чтобы каждый компонент при доступе к данным взаимодействовал с минимальным числом других компонентов.

Хотя уточнение моделей проводится в конце стадии рационализации, к нему, возможно, придется вернуться на этапе разработки и стабилизации. Комплектацию и последовательность развертывания иногда приходится корректировать после тестирования по какому-либо критерию — например, по производительности. Поэтому, если система не удовлетворяет требованиям в отношении производительности, стоит пересмотреть план комплектации и развертывания.

Стадия рационализации завершается, когда решены следующие задачи:

- определена стратегия комплектации и развертывания;
- в рамках создания компонентной модели объекты преобразованы в компоненты, основанные на сервисах,
- с целью создания окончательной модели развертывания, включающей топологии сети, данных и компонентов, компоненты распределены по топологии;
- уточнены модели комплектации и развертывания.

Шаг 4: реализация

На последнем этапе физического проектирования — этапе реализации — проектная группа создает модель программирования для разработчиков, а также интерфейсы и внутреннюю структуру компонентов.

Модель программирования (иногда ее называют программными спецификациями или стандартами):

- описывает способы использования выбранных технологий;
- учитывает некоторые особенности спецификаций компонентов, что помогает в дальнейшем согласованно работать над проектом;
- конкретизирует различные элементы решения.

При разработке программной модели следует учесть многие вопросы (некоторые из них перечислены в табл. 6.6); подробнее о них — в следующих главах.

Табл. 6.6. Некоторые элементы программной модели

Элемент	На что обратить внимание
Технологии реализации	Язык программирования; прикладные интерфейсы; серверы и серверные технологии; другие технологии, влияющие на реализацию. На технологии реализации следует обратить внимание потому, что некоторые из них требуют для успешного применения специфической программной модели. Например, для работы с Microsoft Transaction Server пригодны только однопоточные внутрипроцессные компоненты с единственной точкой входа
Объекты с состоянием и без него	Объекты с состоянием хранят информацию *(состояние),* отражающее выполнение клиентских вызовов, а объекты без состояния — нет. При выборе типа объектов нужно учесть его масштабируемость, сложность и производительность
Внутренние и внешние функции	Внутренние вызовы выполняются быстро и напрямую; внешние на том же компьютере также выполняются достаточно быстро и обеспечивают безопасное взаимодействие процессов. Внешние вызовы на разных компьютерах безопасны, надежны и могут использовать гибкий протокол на основе *вызова удаленных процедур в распределенной среде* (Distributed Computing Environment Remote Procedures Calls, DCE-RPC)
Режимы с подключением и без него	В средах с подключением различные компоненты, участвующие в решении задачи, должны быть постоянно соединены друг с другом. Если соединение прервется, возникнет ошибка. Приложения реального времени обычно работают в режиме с подключением. Приложения и компоненты, написанные для использования в среде без подключения, способны установить соединение, только когда оно им необходимо
Развертывание	Три логических уровня не обязательно преобразовывать в три физических уровня. Логические уровни можно распределить по физическим — например, некоторые бизнес-сервисы разместить на клиенте

(продолжение)

Элемент	На что обратить внимание
Синхронные и асинхронные программные модели	В синхронной программной модели выполнение компонента, вызвавшего интерфейс, блокируется до завершения выполнения вызова и возвращения управления компоненту. В асинхронной программной модели компоненты способны передавать друг другу сообщения и продолжать работу, не дожидаясь ответа. Компоненты для работы в асинхронном режиме сложнее программировать, но они обеспечивают лучшую масштабируемость приложения, поскольку не блокируются и не ждут завершения порожденных ими процессов
Модель потоков	Выбор эффективной модели потоков — дело нелегкое, так как она зависит от назначения компонента. Компонент, ответственный за ввод/вывод, может поддерживать модель произвольных потоков, обеспечивая быстрый отклик на действия клиентов за счет вызовов интерфейсов во время скрытых операций ввода/вывода. С другой стороны, объект, взаимодействующий с пользователем, способен использовать для синхронизации вызовов COM с оконными операциями модель распределенных потоков
Обработка ошибок	Каждая модель программирования и развертывания ограничивает выбор методов обработки ошибок
Защита	Каждый компонент или сервис имеет четыре варианта защиты: компонентный (на уровне методов, интерфейсов или компонентов), на уровне базы данных, на уровне пользовательского контекста (интерактивный метод, использующий системную защиту, или фиксированная зашита в рамках приложения) и ролевой (основан на исполняемых ролях — например, клерк или генеральный директор)

Примечание Не существует программной модели, одинаково пригодной для всех компонентов, поэтому часто приходится применять разные модели в зависимости от требований различных компонентов.

Следует обратить внимание еще на один важный вопрос, не учитываемый в программной модели, — квалификацию и опыт разработчиков, реализующих избранную модель.

Когда модель программирования описана в общих чертах, можно определить внешнюю структуру компонентов. Они описаны в интерфейсе компонента, который:

- является контрактом, реализующим связь типа «поставщик — потребитель» между компонентами;
- является средством доступа к основным сервисам;
- реализует один или несколько сервисов;
- включает основные атрибуты объекта,

Спецификация интерфейса компонента должна описывать все способы доступа к компоненту и, по возможности, примеры использования каждого из них (рис. 6.11). Обратите внимание, что интерфейс должен быть документирован и понятен разработчикам, которые будут использовать компонент.

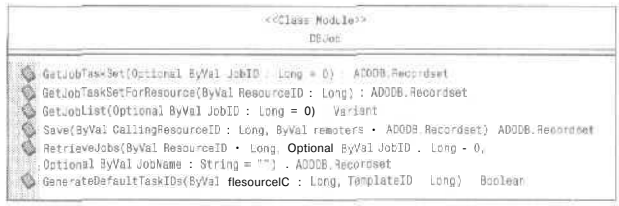


Рис. 6.11. Спецификация интерфейса компонента

При создании интерфейса компонента обратите внимание на то, что:

- опубликованный интерфейс считается контрактом и не должен изменяться;
- модификацию существующего интерфейса надо публиковать как новый компонент или как новую версию интерфейса;
- заказчик интерфейса должен поддерживать типы данных опубликованных атрибутов.

Интерфейсный контракт определяет параметры, типы данных, стандарты взаимодействия и описание интерфейса. Степень специализации зависит от потребностей пользователя и, в конечном счете, от необходимости повторного использования.

Теперь проектная группа может приступить к разработке внутренней структуры компонентов. Обратите внимание, что внутренняя структура важна только при создании компонентов, а при их сборке имеет значение лишь интерфейс. При определении внутренней структуры компонентов важны многие факторы. Самый важный — язык или средство реализации компонента. Остальные факторы служат

критериями, определяющими внутреннюю структуру компонента. Например, при создании первого многоуровневого приложения для крупной организации, планирующей разрабатывать другие приложения, которые будут использовать те же сервисы, внутренняя структура компонентов должна быть спроектирована таким образом, чтобы гарантировать возможность их повторного применения.

Основные результаты стадии реализации видны после определения внутренних структур компонентов; в дальнейшем их включают в физический проект. Этап реализации завершается, когда решены следующие задачи:

- выбрана программная модель;
- определены внешние структуры компонентов, в том числе интерфейсы, атрибуты и сервисы;
- определены внутренние структуры компонентов.

На этом проектирование считается законченным. Его создание длится долго: от реальных требований пользователей через абстракции концептуального и логического проектирования к реальности физического и окончательного проекта приложения. Результаты проектирования ложатся в основу функциональных спецификаций, которые будут включены в план и график проекта, а также в основной документ оценки рисков.

Управление рисками

На фазе «Анализ» группа разработчиков начала выявлять и анализировать риски, используя при этом процесс, проиллюстрированный на рис. 6.12. Ранее уже был подготовлен документ оценки рисков — в общем виде, так как на этапе «Анализ» проект еще не разработан в деталях.

Рис. 6.12. Пять этапов управления рисками

Один из результатов фазы «Планирование» — пересмотренный документ оценки рисков. На этой стадии проекта можно еще раз провести контроль рисков, уточняя план и, возможно, корректируя предварительные оценки. По мере разработки проекта этот процесс периодически повторяется, при этом некоторые риски исключают, остальные отслеживают: иногда обнаруживаются новые риски. Когда подготовлен пересмотренный документ оценки рисков, полезно включить в повестку всех последующих совещаний обсуждение контроля рисков. При этом используются те же пять этапов управления рисками (рис. 6.12.).

При создании начального документа оценки рисков на стадии «Анализ» группа определила всевозможные риски, Тогда же подсчитали и *рейтинг риска*, равный произведению его *вероятности* на *влияние*. На основе такого анализа разработаны планы борьбы с десятью главными рисками, причем для каждого плана исследовались пять основных параметров:

- **информация** — достаточно ли мы знаем о риске;
- **приемлемость** — можем ли мы смириться с последствиями риска;
- **исключение** — можем ли мы избежать риска;
- **снижение** — можно ли уменьшить вероятность наступления или последствия риска;
- **план действий** — что можно сделать, если риск реализовался.

После этого разрабатываются планы снижения рисков или планы действий в случае наступления риска.

Некоторые проектные группы, создав план реагирования на риски, считают, что на этом работа окончена. Это заблуждение, потому что этапы отслеживания и управления так же важны, как и этап планирования.

Создав документ оценки рисков, продолжайте отслеживать их состояние с учетом развития ситуации. Для каждого риска нужно ответить на следующие вопросы;

- все ли мы сделали для снижения риска;
- изменился ли риск под воздействием внешних обстоятельств;
- изменились ли в результате наших усилий или действия внешних сил вероятность наступления или последствия риска;
- если что-то изменилось, как это отражается на рейтинге риска и, следовательно, на его приоритете;
- не возникло ли непредвиденной ситуации;
- что нужно сделать, исходя из собранной информации.

На этапе управления рисками проектная группа изучает изменения, выявленные на предыдущем этапе. В результате этого ее сотрудники разрабатывают новый план снижения рисков, активизируют

планы действия при наступлении риска, повторно анализируют риск или, если он реализовался или был преодолен, исключают его из списка.

Управлениерисками—динамическийпроцесс,представляющий собой жизненно важную составляющую модели MSF.

Этап «Одобрение плана проекта» и его результаты

Цель фазы «Планирование» — достичь этапа «Одобрение плана проекта», представляющего собой кульминацию работы проектной группы на данном этапе и знаменующего достижение согласия с клиентом по основным вопросам проекта.

Для достижения этого этапа необходимы четыре документа:

- функциональные спецификации, описывающие разрабатываемый продукт и его назначение;
- основной план проекта, описывающий выполнение проекта, включая последующие промежуточные этапы и планируемые результаты;
- основной график проекта, регламентирующий время, необходимое для достижения следующих этапов;
- пересмотренный документоценкирисков,описывающий возможные риски и действия группы по отношению к ним.

Кроме того, мы рекомендуем включить в этот список создание прототипа, демонстрирующего корректность проектных решений.

Примечание Под результатом понимается не только текстовый документ, Это может быть диаграмма, приложение, снимок с экрана, сообщение электронной почты — лишь бы он был информативным.

Каждый этап модели MSF означает соглашение между заказчиком, проектной группой и другими участниками проекта. На этапе «Одобрение плана проекта» это достижение соглашения по пунктам, перечисленным в табл. 6.7.

Табл. 6.7. Соглашения, достигнутые на этапе «Одобрение плана проекта»

Пункт соглашения	Связанный с ним документ
Что нужно сделать, чтобы продукт соответствовал потребностям бизнеса	Функциональные спецификации
Приоритетные задачи	Функциональные спецификации
Сколько времени потребуется для завершения проекта	График проекта
Как создавать продукт и кто будет этим заниматься	План проекта

(*продолжение*)

Риски	Пересмотренный документ оценки рисков
Опорные точки и основные результаты	План проекта

Достигнув этапа «Одобрение плана проекта» и рассмотрев основные результаты фазы «Планирование», заказчик и проектная группа могут решить, что прибыль от продукта не соответствует затратам на его создание, и по этой причине закончить его разработку. Вероятность такого исхода демонстрирует важность фазы «Планирование». Хорошо выполненная фаза «Планирование» позволяет заказчику и группе уверенно продолжить разработку проекта, зная, что на первом этапе сделано все, чтобы повысить шансы на успех.

Промежуточные этапы

Советуем разбить фазу «Планирование» на части, введя промежуточные этапы. Так как планы и графики разрабатываются на основе функциональных спецификаций, имеет смысл сначала создать черновик спецификаций. После этого каждый член группы сможет разработать черновики своих планов и графиков, которые затем будут включены в основные документы. По мере реализации стадий процесса проектирования и фазы «Планирование» черновые документы будут пересматриваться и дополняться.

Функциональные спецификации

Функциональные спецификации — главный результат фазы «Планирование». Эти подробные спецификации продукта служат контрактом между заказчиком и проектной группой. Мы не преувеличим, если скажем, что суть всей предыдущей работы — их создание, а всей последующей — их реализация. На основе функциональных спецификаций создаются все остальные основные документы фазы «Планирование»: план проекта, график проекта и документ оценки рисков.

За функциональные спецификации отвечает менеджер программы, но это не значит, что он должен сам создать этот документ. Функциональные спецификации отражают понимание продукта всей группой разработчиков и включают информацию ото всех ее участников.

При создании функциональных спецификаций следует избегать следующих типичных ошибок:

- недостаточной или излишней детализации;
- создания нереального проекта;
- преждевременного утверждения спецификаций;
- чрезмерных затрат времени на пересмотр функциональных спецификаций;

- отказа от передачи информации об изменениях заказчику, членам проектной группы и другим участникам проекта;
- неучастия некоторых членов группы в проектировании.

Хотя большинство из этих ошибок очевидны, их часто не замечают и... совершают. Поэтому во время разработки функциональных спецификаций отпечатайте этот список и повесьте его на видное место,

Содержание функциональных спецификаций

Функциональные спецификации должны содержать разделы, перечисленные в табл. 6.8. Кроме них стоит включить в список дополнительные сведения, относящиеся к данному проекту.

Табл. 6.8. Основные разделы функциональных спецификаций

Раздел	Описание
Резюме проекта	Точка зрения проектной группы на продукт, его обоснование и основные ограничения. Основывается на документе «Концепция», разработанном на фазе «Анализ»
Цели проекта	Описание целей группы. Разработчики используют их при изучении таких вопросов, как производительность, надежность, своевременность, удобство применения и доступность продукта. Эти цели формулируются на стадии «Анализ»
Требовании всех сторон	Описание того, что, по мнению заказчика, пользователей и других участников проекта, должно делать приложение. Требования надо упорядочить по значимости и, кроме того, устранить конфликты между ними
Резюме по использованию	Описывает, кто и как будет использовать продукт. Это не что иное, как объединенные сценарии использования, построенные на этапе проектирования
Функциональные возможности	Точное описание характеристик продукта, включая эскиз пользовательского интерфейса, средства навигации и подробное описание функциональных возможностей
Зависимости	Описание внешних объектов, от которых зависит продукт. В него входят как высокоуровневые (например, взаимодействие с корпоративными системами), так и низкоуровневые (например, разделяемые компоненты) объекты

(продолжение)

Описание графика	Краткое описание основного графика проекта; определяет этапы, основные результаты и дату выпуска продукта
Риски	Перечень рисков; риски, требующие дополнительного изучения, должны быть специально отмечены
Приложения	Описывают все, что не вошло в предыдущие разделы, Набор результатов процесса проектирования, используемый при разработке функциональных спецификаций

Пересмотр функциональных спецификаций и выработка согласованного мнения

Пересмотр функциональных спецификаций — важный этап их разработки. На этой стадии:

- каждый заинтересованный в функциональных спецификациях член группы может принять участие в их пересмотре;
- множество разных людей проверяют выполнение требований;
- все участвующие в проекте узнают, что получится в конце разработки;
- ведутся переговоры и достигаются соглашения,

Основная цель пересмотров функциональных спецификаций – обеспечить всех членов группы исходными данными. Если в пересмотрах участвуют члены группы, ответственные за разные направления разработки, то и спецификации будут отвечать требованиям всех направлений. Кроме того, таким образом все участники проекта обязательно изучат соответствующие документы.

Обратите внимание на слово «пересмотры» (во множественном числе) в предыдущем абзаце. Действительно, на стадии «Планирование» потребуется по крайней мере дважды пересмотреть функциональные спецификации, поскольку одним из промежуточных документов является лишь их черновик. В идеале функциональные спецификации необходимо пересматривать и обновлять регулярно.

По окончании создания функциональных спецификаций должно быть выработано *согласованное мнение* о проекте. Оно гарантирует всем участникам проекта, что функциональные спецификации точно отражают их ожидания (табл. 6,9).

Табл. 6.9. Соглашения, необходимые для принятия функциональных спецификаций

Роль	Пункт соглашения
Заказчик	Решение отвечает бизнес-требованиям. Заказчик может принять графики проекта и оценки планируемых затрат, изучив описание проекта, представленное в функциональных спецификациях
Менеджер продукта	Решение отвечает существующим требованиям. Менеджер продукта считает, что в функциональных спецификациях описан продукт, удовлетворяющий сформулированным требованиям
Менеджер программы	Ответственности группы и трафики реалистичны. Менеджер программы считает, что всем ясен фронт работ и что работы будут выполнены в соответствии с графиком
Разработчик	Решение можно реализовать. Разработчик на фазе «Планирование» исследовал риски, связанные с его работой, и полагает, что они не помещают закончить проект к намеченной дате. Разработчик может построить для себя график, основываясь на текущих функциональных спецификациях
Тестер	Решение можно протестировать и стабилизировать Тестер располагает стратегией тестирования платформ, сценариев и данных и обязуется протестировать все элементы, описанные в функциональных спецификациях
Инструктор	Решение удовлетворяет требованиям в отношении использования; потребности в сопровождении определены. Инструктор ясно представляет себе требования пользователей приложения и специфику его сопровождения. Он обязуется разработать систему поддержки пользователей, описанную в функциональных спецификациях
Логистик	Развертывание и сопровождение решения возможны. Логистик имеет ясное представление об организационном, прикладном и системном интерфейсах и обязуется развернуть систему
Все стороны	Все согласны с датой выпуска

Основной план проекта

Основной план проекта описывает создание проекта, в том числе подробные планы, составленные разными членами проектной группы. Предназначен для синхронизации работы группы.

Цель основного плана проекта:

- объединить планы разных членов группы;
- описать разные направления деятельности;
- синхронизировать планы различных подгрупп.

За основной план проекта отвечает менеджер программы, считающийся координатором планирования проекта и работы над ним. По каждому направлению составляются отдельные планы, которые включаются в основной.

Основной план проекта должен содержать разделы, перечисленные в табл. 6.10.

Табл. 6.10. Разделы основного плана проекта

Раздел	Описание
Разработка	Описание способов разработки составляющих продукта, описанных в функциональных спецификациях. К ним относятся инструменты, методологии, последовательности событий и методы тестирования
Тестирование	Стратегия тестирования, описание конкретных областей, которые необходимо протестировать, и требуемых ресурсов (оборудование и люди)
Обучение	Стратегия и планы разработки обучающих материалов
Поддержка пользователей	Стратегия поддержки и ее составные элементы, необходимые для работы пользователей (например, мастеры и справочная служба)
Контакты	Маркетинговые и рекламные мероприятия
Развертывание	Стратегия и подробный план по подготовке пользователей и персонала до и во время развертывания

Примечание Обычно основной план проекта не используется для управления проектом напрямую из-за громоздкости и недостаточной детализации. Вместо него в рамках каждого направления деятельности проектной группы используются отдельные планы, которые синхронизируются с основным.

Основной график проекта

Мы поставили описание основного графика проекта на *третье место*, чтобы подчеркнуть: график по значимости *уступает* функциональным спецификациям и основному плану проекта. Группа разработчиков не сможет подготовить график, пока не определит функциональные возможности продукта и не спланирует методы их реализации.

Отсюда, однако, не следует, что эти документы просто вытекают один из другого. Процесс их корректировки может, а иногда и должен, быть итерационным. Сначала конкретизируется набор функций, затем составляется план и, наконец, разрабатывается график. Если же оказалось, что при таких спецификациях время выпуска надолго откладывается, проектная группа должна пересмотреть функциональные спецификации, а затем основной план проекта и его график, повторяя эту процедуру до тех пор, пока все три документа не будут согласованы.

Помните, что работу можно планировать с учетом даты выпуска продукта, но попытка *изначально* составить график, а затем в соответствии с ним разрабатывать план проекта и функциональные спецификации ни к чему хорошему не приведет. График должен базироваться на двух предыдущих документах, а дата выпуска, определенная на стадии «Анализ», — только ориентир.

За основной график проекта отвечает менеджер программы, но он его не составляет. Он лишь объединяет детальные графики подгрупп или отдельных членов группы в единый график проекта. Наиболее важен график разработки. На его основе составляются все остальные графики.

В основной график проекта входят:

- график разработки;
- даты выпуска продукта (как окончательного, так и промежуточных версий);
- график тестирования;
- расписание обучения работе с продуктом;
- график поддержки пользователей;
- расписание рекламных мероприятий;
- график развертывания.

Принципы построения графика

Предлагаемые MSF принципы составления графиков подробно обсуждаются во всех главах этой книги, однако мы сочли необходимым еще раз кратко описать их. Ознакомьтесь с четырьмя принципами модели управления рисками MSF: это поможет вам создать реалистичные и выполнимые графики.

*Построение «снизу - **вверх»***

Работу планируют те, кто ее выполняет. Их оценки должны быть отражены в основном графике проекта. Это не только повышает точность оценок, но и увеличивает вероятность одобрения графика всей группой.

*Построение на **основе фиксированной** даты выпуска*

Фиксированная дата выпуска исключает простые оправдания, ограничивает возможность нежелательных компромиссов между ресурсами и функциональными возможностями продукта, и устанавливает четкие границы того, что будет разработано и до какой степени. Обратите внимание, что дата выпуска выбирается на основе *реальных условий*, а не *произвольно*. Достижение результатов к определенной дате требует применения остальных трех принципов составления графиков,

Учет рисков

В графиках, составленных на основе рисков, в первую очередь выполняются задачи, связанные с наибольшим риском. Этот метод отличается несколькими серьезными достоинствами:

- как правило, самые значительные риски связаны с высоким уровнем неопределенности; при их первоочередном изучении у группы хватит времени на разработку методов их снижения (если таковые существуют);
- если проектная группа изучит риск, грозящий остановить работу, и не найдет методов его снижения или исключения, проект можно свернуть на ранней стадии, тем самым сэкономив ресурсы;
- понимание особенностей проекта, наиболее важных для заказчика, и выявление рисков, наиболее опасных для проекта, позволяет достичь понимания с заказчиком;
- попытки снизить риски, способные остановить проект, часто требует создания пробных систем для изучения воздействия этих рисков еще на стадии планирования.

Очевидно, что для составления графика на основе рисков необходимо на стадии анализа рисков упорядочить их по степени опасности.

Учет неопределенности

График и неопределенность, или непредсказуемость, на первый взгляд противоречат друг другу. Смысл этого принципа в том, что, раз уж жизнь полна неожиданностей, график должен учитывать возможность появления непредсказуемых факторов. Три правила, описанные ниже (рис. 6.13), помогут вам составлять такие графики.

- **Запланируйте резерв времени** — менеджер программы должен предусмотреть резерв времени, который позволит справиться с неожиданностями. Хотя этот метод весьма спорный и не нравится

менеджерам, без него не обойтись. Проект без резерва времени обречен на неудачу. Объем такого резерва зависит от уверенности менеджера программы в правильности оценок затрат на выполнение работ, представленных остальными членами группы. При составлении графика «снизу—вверх» определяется «внутренняя» дата выпуска. Добавив к ней резервное время, вы получите «внешнюю» дату — для заказчика. Однако резервное время назначается и управляется менеджером программы, а не отдельными исполнителями — они должны ориентироваться на собственные даты.

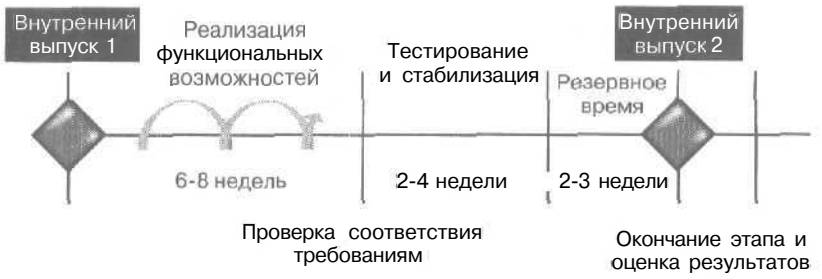


Рис. 6.13. Пример графика проекта

- **Используйте промежуточные выпуски** — это альфа- и бета-версии приложения. Разделение проекта на небольшие части позволит вам получать информацию о ходе его выполнения чаще и вовремя. При этом продукт станет стабильнее, а график работ — более предсказуемым. Использование предварительных версий программы, проходящих собственную стадию стабилизации, позволит вам изучить состояние проекта и потренироваться в выпуске продукта.
- **Разбивайте проект на отдельные задачи** — это позволит постоянно контролировать ход его выполнения. Требуйте промежуточных результатов или отчетов за короткие периоды времени. Выполнять небольшие задачи и управлять ими намного проще. К тому же, чем короче продолжительность выполнения задачи, тем меньше ошибок при ее решении. Например, делать недельную работу шесть недель не только нелепо, но и непросто. Проектная группа должна разделить функции приложения на связанные друг с другом конкретные задачи, характеризующиеся началом и концом, одним результатом и одним ответственным. Время выполнения каждой задачи надо ограничить требуемыми на ее решение усилиями, но оно не должно превышать одной-двух недель. Это позволяет контролировать выполнение всех задач по мере выполнения проекта и в соответствии с ходом разработки корректировать графики.

Пересмотренный документ оценки рисков

К концу фазы «Планирование» у проектной группы должно сложиться ясное представление о рисках, связанных с проектом, а также об их влиянии и значимости. Поэтому пересмотренный документ оценки рисков должен быть более подробным.

За этот документ и его пересмотр отвечает менеджер программы. Он включает в документ оценки рисков уточненную информацию, полученную от других членов группы. В результате формируется сводка конкретных оценок, дающая общее представление о рисках проекта.

Пересмотренный документ оценки рисков помогает синхронизировать оценки рисков разных членов группы. На его основе принимаются решения, относящиеся к рискам и их упорядочению по значимости. Однако управление рисками проводится отдельно по каждому направлению деятельности группы, так как этот процесс на уровне документа оценки рисков слишком сложен.

Резюме

Фаза «Планирование» является основой процесса разработки архитектуры. Проектная группа собрала информацию о требованиях и выработала представление о приложении еще на стадии «Анализ» модели процесса разработки MSF. На фазе «Планирование» группа проектирует архитектуру приложения — фундамент, на котором основывается разработка исполняемого кода приложения. Достигнув этапа «Одобрение плана проекта», все члены группы уже знают, как будет разрабатываться приложение и сколько времени это займет.

В этой главе мы рассмотрели этапы концептуального, логического и физического проектирования, составляющие процесс проектирования MSF. На каждой из перечисленных стадий с нескольких точек зрения изучаются уровни приложения, соответствующие пользовательскому, прикладному уровню и уровню данных модели приложения MSF. Кроме того, мы описали документы, составляемые на фазе «Планирование»: функциональные спецификации, основной план проекта и основной график проекта.

Закрепление материала

1. Зачем нужна фаза «Планирование»?
2. Опишите три этапа, входящие в процесс проектирования MSF.
3. Какую информацию содержат функциональные спецификации?
4. Что такое план проекта?

Практикум 6. **Планирование**

– Итак, настал час исповеди. Все изучили материалы к сегодняшнему совещанию?

Дэн сидел во главе стола с чашкой кофе в руках. Он только что вставил в проектор слайд, иллюстрирующий фазу модели MSF «Планирование». Первое совещание, посвященное ей, началось пятнадцать минут назад, и группа уже обсудила повестку дня и пересмотрела результаты фазы «Анализ».

Помедлив немного, заговорил Тим:

– Не буду врать, Дэн. Я к ним даже не прикасался — у нас сначала «упал» сервер, а потом возникла проблема с вирусом в Кливленде. Я просто не успел просмотреть бумаги.

– Почему же ты не принял свое знаменитое бодрящее средство? — ухмыльнулась Джейн.

– Я бы мог, но подумал, что, если все же запихну информацию себе в голову таким способом, то просплю все совещание.

Когда смех утих, робко вмешалась Марта:

— Я все прочитала, но это было трудно. Честно говоря, я практически ничего не поняла.

Джейн, сидящая напротив, кивнула в знак согласия.

– Ничего страшного, Марта, — сказал Дэн. — Ведь разработкой занимаешься не ты, и не Джейн, и даже не Тим. Поэтому прежде чем продолжить работу, мы кое-что проверим. Билл, — добавил он, повернувшись, — тебе как разработчику стоило читать. Ты с этими документами справился?

– Я их просмотрел в среду вечером, — сердито сказал Билл, открывая свой блокнот. — Похоже, будет много проектной работы. От того, что мы испишем кучу бумаги, программа сама по себе не появится. И еще: я согласен, что RMS — важный проект, но не важнее приложений. Почему мы тратим время на составление этих планов, вместо того чтобы приняться за работу?

Не успел Дэн ответить, как заговорила Мэри-Лу:

– Ты так злишься, потому что хочешь уйти сегодня пораньше, чтобы порыбачить.

Билл удивленно спросил, откуда она это знает, на что Мэри-Лу ответила:

– Когда я ставила машину в гараж, я заметила твой большой синий джип с лодкой, который занял пять мест на парковке!

Все засмеялись, кроме Билла, который выглядел довольно глупо. Тим повернулся к нему и сказал:

— Она тебя раскусила, старина.

— Не беспокойся, Билл. Если мы поработаем как следует, то потом и отдохнем хорошенько, — сказал Дэн. — Я знаю, что ты почти всегда приходишь на работу раньше меня, а уходишь позже. Так что когда мы закончим — порыбачишь всласть. Не забудь поймать рыбки и для меня. Между тем, Билл затронул важные вопросы, — продолжил Дэн, подойдя к доске, — и я думаю, что мы должны обсудить их до того, как займемся планированием.

Фергюсон и Барделл

Строительство • Проектирование • Управление проектами

Проект создания системы управления ресурсами

Повестка дня

Дата: 9 апреля 1999 г **Тема:** планирование 1

I. Уточнение повестки

II. Обзор фазы «Планирование»
- проектирования
- функциональные спецификации
- основной план проекта
- основной график проекта

III. Обзор процесса проектирования
- концептуальное проектирование
- логическое проектирование
- физическое проектирование

IV. Концептуальный проект системы управления ресурсами

V. Вопросы и ответы

VI. Задания и ответственные

Чикаго • Детройт • Милуоки • Цинцинати • Индианаполис • Луисвилль

Он нарисовал на доске модель пропесса разработки MSF и сказал:

– Билл прав, мы могли бы пропустить некоторые этапы MSF и быстрее выпустить приложение RMS. Так почему же я настаиваю, чтобы мы делали все «по учебнику»? Потому что у нас в плане несколько крупных проектов, при работе над которыми без MSF не обойтись. Я бы не хотел рисковать и изучать MSF на примере такого проекта. Я хочу, чтобы мы быстро освоили MSF, чтобы все поняли ценность методики и поверили в нее. Этот проект идеально подходит как учебный. Он может показаться пальбой из пушки по воробьям, но зато мы освоим модель MSF шаг за шагом и не только сами изучим ее, но и сможем рассказать о ней всем остальным.

– Например, Джиму Стюарту? — спросил Тим. жадно поглощая пончик.

— И Джиму, и группе разработчиков Билла, и начальнику проектного отдела, и всем, кто внимательно следит за этим проектом. Им известны наши обещания, и они с интересом наблюдают, выполним ли мы их. Пусть они знают, что можно подать запрос в ИТ-отдел и за разумное время получить ответ. Кроме того, я хочу, чтобы Джейн, Мэри-Лу и Марта объяснили им, что такое «разумные временные рамки». Я хочу, чтобы проект принес какие-то дивиденды помимо денежной прибыли.

Дэн повернулся и нарисовал под схемой модели треугольник.

– Есть еще одна причина делать все, как написано в книге, — это влияние методики MSF на производственную архитектуру «Фергюсон и Барделл». — Он написал в треугольнике «Производственная архитектура», нарисовал стрелку от треугольника к модели и обратно и добавил: — Работа над проектом частично основана на производственной архитектуре, а в результате выполнения проекта архитектура изменится. То есть эти объекты динамично взаимодействуют друг с другом.

Дэн написал над стрелкой слово «документация» и сказал, обращаясь к Биллу;

– Вот одна из причин необходимости бумажной работы. Все результаты нашей работы над проектом RMS будут включены в документацию на нашу архитектуру.

— Я слышала о производственной архитектуре, — сказала Мэри-Лу. — Кто-то мне говорил, что этот самоуверенный Кевин Кеннеди работает над созданием проекта производственной архитектуры компании. Как будто ему популярности не хватает.

— Слухи верны только отчасти. — ответил Дэн, пропустив нападки на Кевина мимо ушей. — Нельзя *создать* производственную архи-

тектуру, потому что она уже существует. А Кевин со своей группой просто *документируют* ее и планируют ее пересмотр. Проект **RMS** связан с их работой, и результаты нашей фазы «Планирование» будут в ней использованы. В наших планах — сотрудничество с ними.

– Есть два способа не увязнуть в трясине планирования и документирования, — продолжил Дэн, сев за стол. — Во-первых, *сохранять равновесие*. Вы должны подробно изучить все действительно важные особенности проекта, выдерживая необходимую степень детализации. Чем шире и значительнее проект, тем больше требуется времени на составление планов и подготовку документации. Во-вторых, *действовать по графику*. В начале работы мы составили предварительный план, в соответствии с которым предполагаем достичь определенных опорных точек. Он поможет нам не тратить лишнее время на какой-либо один этап разработки. Нам придется помнить о необходимости пошевеливаться, чтобы не выбиться из графика. – Дэн сделал паузу и спросил: — Теперь все ясно? Всем понятно, зачем нужна фаза «Планирование»?

— Я себя чувствую как неврастеник из старого анекдота, — сказала Джейн и, видя замешательство Дэна, добавила: — Мой муж, психиатр, однажды спросил меня, знаю ли я, в чем разница между психом и неврастеником. Я ответила: «Нет». Псих, сказал он, говорит: «Дважды два — пять», а неврастеник: «Дважды два — четыре, но это мне не нравится». Вот и я, — продолжила Джейн, когда смех утих, — как неврастеник. Мне все понятно, но очень не нравится. Похоже, будет много работы.

— Да, Джейн, много — ответил ей Дэн. — На фазе «Планирование» нас ждет кропотливая, иногда утомительная работа. Но это ключ к успеху всего проекта. От качества планирования зависит качество приложения.

– И чем быстрее мы начнем, тем быстрее мои ребята начнут писать код, — нетерпеливо сказал Билл. — Ты нас всех убедил, Дэн, даже меня. Мне больше нравится программировать, но я все осознал, так что давайте перейдем к следующему пункту повестки дня.

– Хорошая идея, Билл, — ответил Дэн, меняя слайд. — Давайте взглянем на то, что нас ожидает.

Знакомство с фазой «Планирование»

– Как видите, фаза «Планирование» располагается между фазами «Анализ» и «Разработка». Ее назначение — преобразовать общий план проекта в конкретные планы для разработчиков. Одно дело сказать: «Нам нужно приложение, которое делает то-то и то-то», и совсем другое: «А вот так-то мы его будем делать».

– Вот путь, по которому мы пройдем, составляя планы, — продолжил Дэн, показав на слайде схему. — Именно таким образом мы освоим процесс проектирования MSF (об этом я расскажу через минуту). Наша задача — создать функциональные спецификации, главный результат фазы «Планирование». На их основе мы составим план и график проекта. Все эти три документа пригодятся нам при последующей разработке. Вопросы есть?

— Значит, в функциональные спецификации мы запихнем все детали приложения? — спросила Марта.

– Именно так, — ответил ей Дэн. — Но окончательный график мы сможем составить только тогда, когда все согласятся с созданными нами планами. Кто-нибудь помнит, откуда берутся планы и графики? — спросил он, перевернув страницу своего блокнота.

Марта и Мэри-Лу начали лихорадочно листать свои блокноты, а Марта при этом бормотала:

— Где-то я видела что-то о составлении графиков «снизу — вверх» и «сверху — вниз», но никак не могу найти.

Тим наклонился, заглядывая в бумаги Дэна, потом повернулся к Марте и театральным шепотом произнес: «Посмотри на странице 56». Он улыбнулся Дэну, так как все сразу же нашли нужную страницу, а Марта торжествующе сказала:

– Проект и график плана — это сводка наших личных планов и графиков.

– Точно, даже несмотря на то, что тебе подсказали с галерки, – смеясь, ответил Дэн. Он заменил слайд и продолжил: — Давайте определим, что каждому надо сделать и какие планы и графики вы должны составить. Последний документ относится к оценке рисков. Его мы составим ближе к концу фазы. Вопросы есть?

Марта и Мэри-Лу подняли руки практически одновременно, но Мэри-Лу ее сразу же опустила.

— Спорим, что ты хотела спросить то же, что и я? — сказала Марта, после чего обратилась к Дэну: — Вот смотрю я на все это и не понимаю, как я справлюсь с такой работой, я же не разработчик. Как я буду оценивать возможности тестирования приложения, а Мэри-Лу возможности по его сопровождению?

— Хороший вопрос, Марта. Не беспокойся, я приготовил для вас дополнительные материалы и несколько книг. Кроме того, я буду с вами заниматься индивидуально, как и с Джейн во время концептуального проектирования, о котором я скоро расскажу. Я сведу вас с людьми, которые выполняли такую работу в фирме, где я работал прежде. Они предлагали помощь, когда понадобится.

Мэри-Лу подняла большой палец: «Смотрю ты ко всему подготовился. Спасибо!»

– Это было нетрудно. Запомните: я хочу, чтобы вы работали успешно. А теперь давайте наметим, как мы будем проектировать приложение.

Знакомство с процессом проектирования

На следующем слайде, который показал Дэн, был изображен такой же круг, как и на предыдущем, иллюстрирующем модель MSF, но разделен он был на три части, а не на четыре.

– Это процесс проектирования MSF. Он состоит из трех стадий: концептуального, логического и физического проектирования. Причем эти этапы не строго последовательны, а перекрываются. Другими словами, мы можем начать составление логического проекта до того, как полностью закончим концептуальный, и работать над ними одновременно. Все три проекта взаимозависимы, то есть, если мы изменим что-то в логическом проекте, нам придется заново просмотреть и концептуальный, чтобы узнать, как на нем отразились эти изменения,

— В чем разница между этими тремя стадиями? — спросила Мэри-Лу.

— Отличий много, но главное, что они представляют разные точки зрения на одно и то же приложение. Концептуальный проект – это мнение пользователей, логический — проектной группы, а физический — разработчиков. Еще одно отличие — уровень абстракции. Процесс проектирования похож на программу трансформации трехмерных объектов. Мы берем что-нибудь конкретное — потребности пользователей, их желания и методы реализации — и подаем это на вход процесса. В результате все эти объекты преобразуются в абстрактный логический проект. Затем мы берем абстрактную модель и трансформируем ее в нечто более конкретное, материальное — в приложение. При этом мы должны убедиться, что вход и выход этих процессов связаны друг с другом — необходимо, чтобы любая характеристика приложения в физическом проекте имела обоснование в логическом и концептуальном проекте, и наоборот.

– Похоже на курс философии, который я слушала в колледже, — качая головой, сказала Джейн. — Я уже запуталась, а ведь мы еще не добрались до сути дела. Может, приведешь какой-нибудь пример?

– Хорошо. На простом примере вы сможете во всем разобраться. Итак, работа, которую мы выполняем каждый день, — резервное копирование файлов данных. Джейн делает это вручную, пусть она и расскажет об этом.

— Я делаю это сама не потому, что не доверяю тебе, Тим, — смущенно сказала Джейн. — Просто, это касается денег и их учета, и я боюсь, как девчонка.

— Это ужасно, — ответил Тим, пытаясь выглядеть обиженным. — Заем мое горе еще одним пончиком.

— На самом деле все просто, — отсмеявшись, сказала Джейн. — Я открываю папку с моими файлами, сортирую их по дате, копирую те, что были изменены сегодня, на Zip-дискету и забираю ее домой.

— Отлично, — сказал Дэн, сделав заметки в блокноте. — Запомните, что на концептуальной стадии мы рассматриваем приложение с точки зрения потребителя. Это понадобится для создания схем использования и сценариев, описывающих что и как делает клиент. Позже я расскажу, как сценарии и схемы использования связаны друг с другом. А теперь опишем процесс резервного копирования с точки зрения пользователя — это позволит нам написать приложение для Джейн.

— А что это будет? Командный файл, приложение на Visual Basic или что-то другое? — спросил Тим.

— Сейчас мы этого не знаем, да нам и не нужно, — ответил Дэн. — Мы выберем технологию реализации позже. Кто-нибудь возьмется описать то, что делает Джейн? — спросил он, подойдя к доске.

— Похоже, что процесс состоит из двух этапов, — ответила Марта, подняв руку. — Сначала она выбирает файлы, а потом их копирует на дискету.

— Ты пропустила еще один этап, — сказал Билл. — Как она выбирает файлы?

— По дате, конечно, — ответила ему Марта. — А, я поняла. Она уже знает дату, но приложение должно ее выяснить. Нам понадобится добавить в начало получение текущей даты.

— Хорошо, — сказал Дэн и записал это на доске. — Теперь все согласны, что получилось точное описание приложения с точки зрения пользователя?

Все дружно кивнули.

— Ну, раз все согласны, приступим к логическому проектированию. Если бы приложение было крупнее, мы продолжили бы концептуальную разработку его отдельных частей. Однако это не помешает нам, закончив работу над описанием, немедленно передать ее для логического проектирования.

— Итак, наша задача на стадии логического проектирования. — продолжал Дэн, вытерев доску, — преобразовать описания, полученные при концептуальном проектировании, в абстракции, называемые *объектами*. Мы должны изучить, какие нам нужны *сервисы, атрибу-*

ты и какие *взаимосвязи* существуют между сервисами. Определения этих терминов вы найдете в своих бумагах.

Пока все изучали определения, Дэн нарисовал несколько прямоугольников на доске, после чего сказал:

– Теперь, когда вы знаете, что мы ищем, расскажите мне, какие объекты возможны в нашем примере.

– Ну, есть просто файлы, — сказала Марта, рисуя что-то в своем блокноте. — А еще файлы с атрибутом «Измененная дата», которые нам и нужны.

– Есть объект «Пользователь», — добавила Мэри-Лу, — но, кажется, у него нет ни сервисов, ни атрибутов. Нам понадобится сервис «Получить дату». И еще где-то нужен сервис «Копировать файлы», но я никак не пойму, какому объекту он принадлежит.

– Это нормально, Мэри-Лу, — сказал Дэн, продолжая рисовать на доске. — Скоро ты увидишь, что иногда приходится создавать объекты только для того, чтобы сервисы имели хоть какого-нибудь хозяина. Еще есть предложения?

— Мне кажется, что в этом случае есть и другие объекты, — вступил Билл, до этого что-то рисовавший в блокноте. — Например, файлы являются частью *файловой системы*. К тому же, кроме сервиса «Копировать файлы» нам понадобится сервис «Выбрать файлы». А как насчет обработки ошибок, если в дисководе не оказалось диска? Кроме того, надо бы удостовериться, что файлы скопированы полностью.

— Отлично, Билл, — сказал Дэн, нанеся новые описания на доску. — На одном из этапов логического проектирования изучаются всевозможные объекты и сервисы, что мы сейчас и сделали. Теперь давайте посмотрим на схемы использования и объекты и проверим, все ли мы учли.

Группа несколько минут приводила диаграмму в порядок: были нарисованы объекты для сервисов-«сирот» и стрелки, показывающие процесс выполнения приложения. Наконец Дэн сказал:

– Все поняли, как из концептуального проекта получается логический?

Все кивнули, и он продолжил:

– Теперь пришло время физического проектирования. Мы определим возможные технологии для реализации нашего приложения и рассмотрим их достоинства и недостатки; исходя из различий между физическими требованиями к приложению и физическими ограничениями, накладываемыми производственной архитектурой и существующими технологиями. Кто предложит технологию реализации приложения КФД?

— А что такое КФД? — удивленно спросил Тим.

— Копирование файлов Джейн. Мы же называем все проекты сокращенно, так что я решил быть последовательным. Что скажете? На чем мы будем писать это огромное приложение масштаба предприятия?

— Я — за Visual Basic, — сразу же сказала Мэри-Лу.

— Ну, это слишком, — возразил Билл. — Я бы написал командный файл, используя 4DOS, заменяющий command.com. В 4DOS можно выбирать файлы по дате одной командой.

— Немного промахнулись, чиф, — сказал, ухмыляясь, Тим. — Командный файл — неплохое решение, но нам потребуется больше возможностей по обработке ошибок, чем предлагает 4DOS. К точу же, в нашей компании 4DOS не одобряют. А вот WinBatch одобряют. Используя его, ты получишь настоящее приложение для Windows, которое просто написать и отладить; его даже можно скомпилировать.

— Похоже, ты победил, — сказал Дэн. — Тим рассмотрел все достоинства и недостатки технологии, учитывая производственную архитектуру, Что скажешь, Билл?

— Ничего не слышал о WinBatch, но если все так, как он сказал. то это действительно лучшее средство. Только я хочу знать, — уже обращаясь к Тиму, произнес Билл, — откуда ты так много о нем узнал?

— Да я использую его уже не первый год, автоматизируя небольшие задачи, — насмешливо сказал Тим. — То, что ты называешься «программистом», не означает, что только ты умеешь программировать.

Тут ему пришлось увернуться от бумажного комка, просвистевшего у него над ухом.

— Ну, Билл, не убей нашего начальника сетевого отдела, — засмеялся Дэн. — Он нам еще пригодится. Я знаю, — продолжил он, повернувшись к Джейн и Марте, тихо наблюдавшим за дискуссией, — это довольно сложно, но не беспокойтесь. Ведь физический проект — это точка зрения разработчиков, именно поэтому на данном этапе они играют ведущие роли. Когда мы дойдем до физического проектирования RMS, я дам вам кое-что почитать. Тогда вы сможете понять обсуждаемые проблемы и подготовить собственные документы.

— На данный момент, — Дэн стер с доски все записи, — если бы мы действительно разрабатывали приложение КФД, мы бы закончили работу над физической моделью и использовали ее для создания функциональных спецификаций. После этого мы попросили бы разработчиков составить план и график проекта, на основе которых составили бы свои. Затем мы бы их объединили, изучили возникшие

конфликты и встретились с заказчиком для утверждения плана. На этом фазу «Планирование» можно считать законченной, и началась бы фаза «Разработка». Вопросы есть?

— Концептуальный, логический, физический проекты, функциональные спецификации, план проекта, график проекта. Это все? — перечислила Марта.

— Не забывай пересмотр документа оценки рисков, — добавила Джейн.

— Принято, — ответила Марта.

Концептуальное проектирование

— Продолжим, — сказал Дэн, заглянув в план совещания. — В некоторых случаях проектирование растягивается на недели и даже месяцы. В случае с проектом RMS мы попытаемся уложиться в три совещания. Поэтому сегодня разберемся с концептуальным проектом,

— К счастью, — продолжил он, когда Джейн встала, — Джейн выполнила эту работу, так что мы сможем закончить концептуальное проектирование уже сегодня.

Джейн раздала по стопке документов каждому из присутствующих.

— Первый этап концептуального проектирования — исследование, во время которого изучаются основные процессы и виды деятельности, а также профили пользователей. Я собрала эту информацию и попыталась организовать ее, — сказала она и села на свое место. — Второй этап — анализ. Здесь пересматривается информация, полученная при исследовании, разрабатываются сценарии для всех схем использования и подготавливаются описания данных, например, параметров среды, в которой протекает процесс. Я попыталась написать несколько сценариев, но думаю, что их нужно проверить.

— Ого, Джейн, ты проделала огромную работу! — восхищенно сказал Тим. — И сэкономила нам кучу времени.

— Мне помогла Мэри-Лу, особенно с профилями пользователей, — ответила ему Джейн.

— Вы обе заслуживаете благодарности, — сказал Дэн, пролистав хорошо организованные документы. — Когда вы начали работать, я не был уверен, что вы сможете сделать все вовремя. Отличная работа.

Джейн и Мэри-Лу улыбнулись.

— Ну что ж, приступим.

Следующие 45 минут все изучали документы и приводили сценарии к окончательному виду. Наконец Дэн сказал:

— Ну, вот мы и закончили — это был последний документ.

Все дружно откинулись на спинки стульев: отдых так отдых.

— Я не хотела этого говорить, так как, похоже, мы уже заканчиваем, — вдруг сказала Марта, — но неужели в концептуальном проектировании нет еще какого-либо этапа? Что-нибудь вроде оптимизации?

– Ты права, есть, — утвердительно кивнул Дэн. — Я сейчас сижу и думаю, стоит ли его проводить или нет. Давайте я расскажу вам о нем, и потом мы вместе решим, нужен ли он нам.

Он вкратце объяснил процесс оптимизации и спросил:

– Требуется ли нашим сценариям оптимизация? Другими словами, хотите ли вы создать помимо сценариев, отражающих текущее состояние, их будущие версии?

На минуту все замолчали. Затем Тим сказал:

— Думаю, достаточно поработать только над сценарием генерации расписаний. Мне он кажется слишком громоздким. Его можно улучшить. Если мы этого не сделаем, большинство пользователей посчитают это ошибкой.

- Ну что ж, давайте, — согласился с ним Билл. — Прямо сейчас, пока мы не забыли схемы использования.

Еще двадцать минут все сосредоточенно пыхтели, но не только над сценарием, ответственным за расписание, но и над другими сценариями, производящими распределение ресурсов и печать счетов.

– Еще что-нибудь изменим? — в десятый раз спросил Дэн.

Никто не ответил.

— Хорошо, тогда на какое-то время мы с этим покончили, — сказал Дэн, встав со стула и направившись к шкафу.

– Что значит «на какое-то время»? Нам придется повторять эту работу? — немного запальчиво спросил Тим.

— Что касается этого проекта, то вряд ли. — ответил ему Дэн, складывая документы в шкаф, — ведь RMS не такая сложная система, хотя и важная. Но всегда существует вероятность, что на следующих этапах мы обнаружим некие факты, которые заставят нас внести коррективы. Ну, тогда уж ничего не попишешь. Запомните, что лучше сделать все на фазе «Планирование», чем столкнуться с проблемами при разработке или стабилизации.

Дэн сложил все бумаги и повернулся к членам группы:

– Здесь документы, которые вы должны прочитать за выходные. В них подробно описаны этапы логического и физического проектирования. Будьте готовы к понедельнику. Кроме того, те, кто не просмотрел предыдущие документы, должны сделать это, чтобы не отставать от остальных, — сказал он, обращаясь к Тиму.

– Похоже, выходные укорачиваются, — вздохнул Тим, положив голову на стол.

- Но не для меня, — сказал Билл, собирая свои вещи и документы. — Я еду на *рыбалку*.

Проходя мимо Тима, он похлопал его по плечу и добавил:

— Я закреплю весло поперек лодки, положу на него ноги и буду читать эту штуку, пока не наловлю достаточно рыбы.

- Хороший план, чиф, — сказал Дэн, раздавая документы другим. — Между прочим, не забудь о совещании с группой производственной архитектуры днем в понедельник.

— Не могу дождаться.

- Похоже, он уже встречался с Кевином Кеннеди, — заметила Мэри-Лу, выходя из комнаты.

Логическое проектирование

Почти все выходные Дэн готовился к совещанию, что стало ясно всем членам группы, когда они обнаружили на столе в его кабинете новые стопки бумаги и скоросшиватели. Свежий кофе и пончики тоже были в наличии.

- Кажется, я знаю человека, который был занят все выходные, причем не рыбалкой, — сказала Мэри-Лу, сев за стол и просмотрев документы.

— Да, я потратил на это несколько часов в выходные, — ответил Дэн, наливая себе кофе. — Мне хочется побыстрее перейти к логическому проектированию, а для этого требовалась предварительная работа с объектами и сервисами.

— Я приезжала сюда в субботу забрать кое-какие вещи, — сказала Мэри-Лу. передавая Тиму пончик. — Это не твою машину, Дэн, я видела на стоянке?

- Мою, но я приехал не из-за RMS, — спокойно ответил Дэн. - Было совещание совета директоров.

Все посмотрели на Дэна, но он перекладывал бумаги из старого скоросшивателя в новый и не заметил вопросительных взглядов, обращенных к нему.

— Новые скоросшиватели — для материалов о MSF, — сказал Дэн. - Мне кажется, стоит хранить их отдельно. Пусть они будут у вас под рукой во время работы над другими проектами, когда папка с бумагами о RMS вам уже не понадобится.

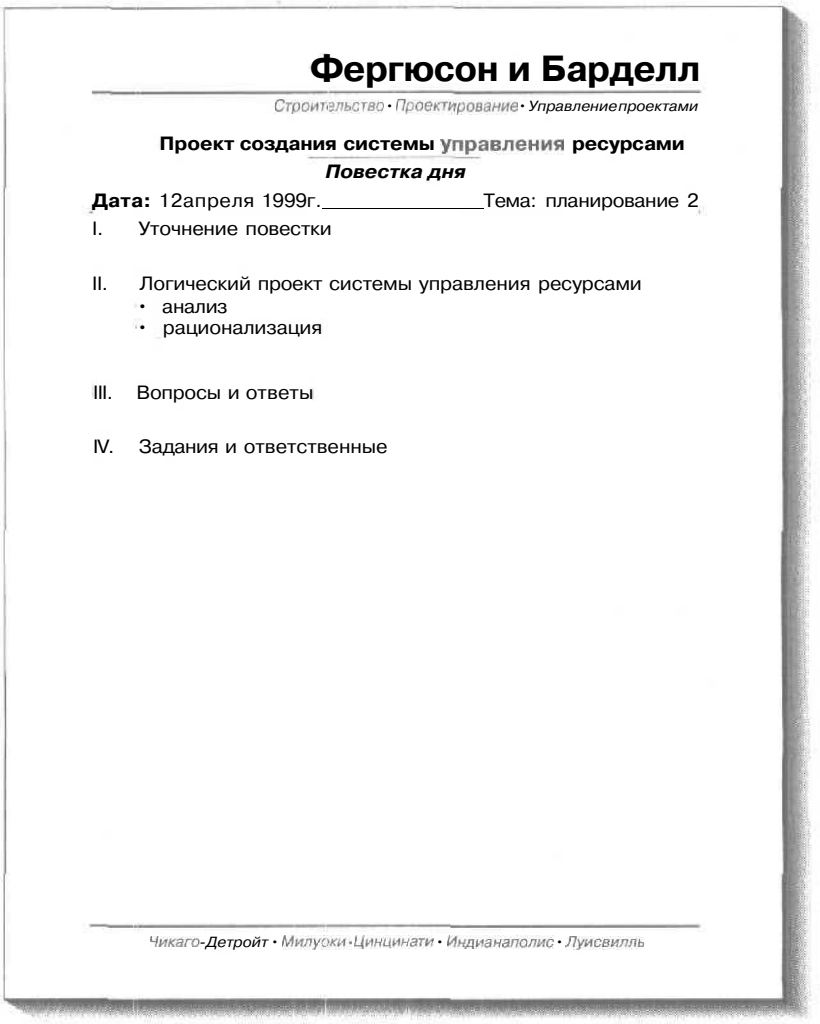
Все стали перекладывать бумаги. В этот момент Джейн наклонилась к Мэри-Лу и тихо сказала:

— Когда закончим, подойди ко мне в офис. У меня для тебя новости!

- Что за новости? — спросила Мэри-Лу. — Очередные слухи о Джеффе из юридического отдела?

– Нет, прямо перед совещанием я получила интересное сообщение по электронной почте.

– Все готовы? — прервал их разговор Дэн. — Тогда откройте свои блокноты. Мы вкратце обсудим этап логического проектирования. Это единственный пункт в сегодняшней повестке дня.



Фергюсон и Барделл

Строительство • Проектирование • Управление проектами

Проект создания системы управления ресурсами

Повестка дня

Дата: 12апреля 1999г._____Тема: планирование 2

I. Уточнение повестки

II. Логический проект системы управления ресурсами
- анализ
- рационализация

III. Вопросы и ответы

IV. Задания и ответственные

Чикаго-Детройт • Милуоки • Цинцинати • Индианаполис • Луисвилль

Пока Дэн писал на доске слова «Анализ» и «Рационализация», остальные перелистывали свои блокноты, выбирая место для новых записей.

— Так, — сказал он, повернувшись к группе, — в логическом проектировании нет этапа исследования. Кто-нибудь может сказать, почему?

— Мы используем результаты концептуального проектирования как начальные данные для логического проектирования, значит, исследование нам не нужно, — ответил Тим. откинувшись на спинку стула.

— Браво, Тим, — похвалил его Дэн. — Я рад, что ты хорошо подготовился в этот раз.

— Я изучал материал не один, — сказал Тим, широко улыбаясь, — а это намного легче.

Зная, что Тим и Билл большие друзья, Дэн подумал, что неплохо бы узнать, сколько же им удавалось прочесть между поклевками, но, оставив эти мысли при себе, он продолжил:

— Тим прав. Анализ будет основан на результатах нашей пятничной работы. Сейчас мы определим все объекты, сервисы, атрибуты и взаимосвязи, как явные, так и неявные. Все помнят определения этих терминов? Тогда начнем работу.

Почти час группа трудилась, про себя благодаря Дэна, подготовившего все документы. Когда они закончили, Джейн сказала:

— Все не так плохо, как я думала. Это все?

— Не совсем, Джейн, — ответил Дэн, подходя к доске. — Мы определили объекты и сервисы, но есть еше один этап — распределение сервисов по уровням. После этого нам, возможно, придется переделать некоторые объекты и сервисы.

— Я так и знал, что на этом работа не закончится, — жалобно простонал Тим.

— Эти уровни — пользовательский, прикладной и данных — из новых документов? — спросила Мэри-Лу. — Зачем они нужны? Зачем разбивать приложение на части?

— Чтобы решить проблемы гибкости, масштабируемости и повторного использования, которые мы изучали раньше, — не дал ответить Дэну Билл. Он подошел к доске и спросил: — Можно я расскажу?

— Конечно, чиф. Вперед! — ответил Дэн и уступил Биллу место, хотя и был несколько удивлен.

Билл минут десять объяснял, что такое многоуровневая разработка, рассказав даже об объединении объектов и сервисов в компоненты.

— Как видите, у многоуровневой разработки так много достоинств, что мы решили использовать ее для всех наших приложений, даже автономных, — закончил он.

— Ты хочешь сказать, что, даже если приложение запускается только на одном компьютере и использует локальную базу данных, вы применяете принципы многоуровневой разработки? — спросила Марта.

– Почти всегда, — ответил ей Билл. — Ведь это очень удобно. Например, мы пишем простую программу-оболочку на Visual Basic для доступа к базе данных Access. В спецификациях ясно сказано, что она не должна быть многопользовательской и размер данных не должен превышать некоторый предел. А теперь представь, что другие пользователи тоже захотели работать с этим приложением, причем с теми же данными. До того, как ты узнаешь об этом, к приложению уже обратятся 10 или 20 пользователей, а объем данных превысит все разумные пределы. Если же мы применим многоуровневый подход, то не составит особого труда изменить сервисы так, чтобы они работали с SQL Server. Главная прелесть в том, что можно писать разные клиентские интерфейсы, работающие с одними и теми же прикладными сервисами и сервисами данных. Например, у нас есть два клиента — Win32 и Web — работающие с одним набором бизнес-сервисов. Web-клиент более привлекателен, так как его легко развернуть. Достаточно просто выложить на сервер все необходимое и, когда пользователь первый раз обратится к Web-странице, нужные для работы компоненты загрузятся на его компьютер и автоматически зарегистрируются. Больше не надо ходить от компьютера к компьютеру, 800 раз устанавливая приложение, не надо мучиться с программами установки. Экономия времени может быть огромной, особенно в таких крупных организациях, как наша.

— Ого! — подумал Дэн, — похоже, чиф всерьез взялся за дело.

Но вслух он сказал:

– Билл, похоже, ты привязался к MSF.

– Ну, почему же именно к MSF, — сдержанно улыбаясь, ответил Билл. — К MSF я еще только присматриваюсь, а с многоуровневыми приложениями год назад меня познакомил мой друг (мы вместе когда-то программировали на Коболе). Я спросил, чем он занят. Он рассказал, что пишет многоуровневые приложения, применяя технологии Microsoft. Тогда я смог только промычать что-то вроде: «Это здорово». Я понял, что совсем не знаю новинок разработки, и не только я, но и все мои ребята. Тогда я решил, что стоит подучиться самому и подтянуть всех остальных из моей команды. Оказалось, что многие из них уже опередили меня и ждут, когда я их догоню. Я говорил с ними о многоуровневой разработке в начале проекта RMS, и они убедили меня, что стоит попробовать. Мы уже написали с ее использованием несколько небольших программ, но первым крупным приложением будет RMS. Поэтому, Дэн, я бы сказал, что я знаю немало о многоуровневой разработке и о соответствующих технологиях Microsoft. Что же касается MSF, старый морской волк только решает, достойна ли эта тема его внимания.

Все, включая Дэна, засмеялись.

— Да, чиф, тебя нельзя обвинить в том, что скрываешь свои чувства. Прекрасно, надеюсь, ты определишься до окончания проекта. Но все же спасибо за отличную вводную лекцию по многоуровневой разработке.

— Он прав, Билл. Даже *я* все поняла. Может тебе стать учителем? — заметила Джейн, пока Билл садился на свое место.

- Нет, — ответил, улыбаясь, тот, — им нельзя ходить на рыбалку по пятницам.

— Хорошо, — продолжил Дэн, — Билл наметил путь. Поэтому еще раз взглянем на уже проделанную работу и закончим на сегодня, распределив сервисы по уровням.

Все поработали еще 30 минут, перемещая объекты и сервисы на пользовательский, прикладной уровни и уровень данных. Как Дэн и ожидал, пришлось кое-что изменить, а некоторые объекты разделить на два для разных уровней.

Наконец, Дэн сказал:

- Похоже, все готово. Теперь проверим. Начнем с пользовательского уровня и выясним, все ли схемы использования мы учли.

После проверки Дэн откинулся на спинку стула:

- Отлично! Кажется, у нас получилась хорошая рабочая версия проекта. Она может измениться на стадии физического проектирования, но на данный момент она завершена. Я красиво все оформлю, — продолжил он, встав, — и отправлю вам завтра утром. Билл и другие разработчики встречаются сегодня днем с группой производственной архитектуры, после чего мы приступим к физическому проектированию. Все должны подготовить к среде черновики своих частей функциональных спецификаций, плана и графика проекта. График разработки вы получите после сегодняшнего совещания. Вопросы есть? Нет? Билл, встретимся с тобой, Бет и Сэмом в 14:00, а с остальными - в среду.

Совещание группы архитектуры и разработчиков

Когда Дэн вошел в зал совещаний, он удивился тому, как расположились участники встречи. С одной стороны стола сидели члены группы ПА. а с другой Билл и два его сотрудника. «Похоже на переговоры по разоружению, Хм, неплохое сравнение», — подумал Дэн, устраиваясь во главе стола.

- Надеюсь, все хорошо пообедали и готовы приступить к работе, — сказал он, открывая свой блокнот. — Вы просмотрели материалы. которые я выслал на прошлой неделе?

— Я-то их посмотрел, — раздраженно ответил Билл, — но кое-чего не понял. Мы ломаем голову над проектом RMS, а группа ПА тем временем, похоже, хочет указать нам направление выбора технологий. Пока ты не пришел, мистер Кеннеди рассказывал нам о замечательных новых технологиях, которые мы будем использовать. С каких это пор отдел долгосрочного планирования вмешивается в наши дела?

Кевин Кеннеди не успел и рта раскрыть, как раздался спокойный голос Ричарда Каплана:

– Почему ты так удивлен, Билл? Я же каждую неделю высылал тебе информацию о работе нашей группы, поэтому в рассказе Кевина ты не должен услышать ничего нового. Что касается Кевина, то он любит важничать, но это не значит, что мы не будем принимать в расчет его вклад в работу. Что касается повестки дня, то сегодня мы собираемся обсудить нашу совместную работу, описанную в разосланных Дэном документах.

Несмотря на то, что слова Дика прозвучали довольно резко, ни Билл, ни Кевин не обиделись, а наоборот выглядели смущенными.

— Дик, как обычно прояснил ситуацию, — сказал Дэн. — Наша цель — не выяснение преимуществ точки зрения одной группы перед другой. Как говаривали в аспирантуре, обе группы должны быть *информированы* о работе друг друга. Архитектурная группа последние шесть недель документировала архитектуру и планировала приоритетные проекты этого года. Как вы понимаете, среди них оказался и RMS. Поэтому наша задача — скоординировать усилия. Это и есть цель нашего совещания.

— Откровенно говоря, Билл, из-за того, что проект RMS так важен, он может повлиять на нас даже сильнее, чем мы на него, — сказала Джо Браун. — Наше исследование показало, что приоритетные проекты способны изменить производственную архитектуру компании.

– Но даже если это и так в нашем случае, нам придется сравнить направления разработки проекта RMS и развития компании «Фергюсон и Барделл», — включился в разговор Кевин. — Ведь все, что мы создадим в рамках проекта RMS, должно соответствовать долгосрочным планам компании.

Он посмотрел на Дэна и добавил:

– Не беспокойся, Дэн, я не забыл о двунаправленном потоке информации, о котором ты так много говорил.

– Наши бизнес-планы должны основываться на технологических планах и разработках, — обратился он к Биллу.

Билл покачал головой,

– Никогда бы не подумал, что кто-то из бизнес-отдела это скажет, — сказал он. — Думаю, что ты не совсем потерян для общества.

– Некоторые с тобой не согласятся, Билл, но все равно спасибо за доверие, — засмеялся Кевин.

Дэн с радостью отметил готовность к сотрудничеству, за что он был очень благодарен Дику Каплану.

— Ну что ж, теперь, когда мы восстановили отношения между двумя отделами, приступим к работе, — сказал он. — Я бы хотел, чтобы ребята из архитектурной группы поведали нам о своем видении проекта **RMS.**

Фергюсон и Барделл

Строительство • Проектирование • Управлениепроектами

Проект создания системы управления ресурсами

Повестка дня

Дата: 12апреля 1999г. **Тема:** производственная архитектура и проект

I. Уточнение повестки

II. Производственная архитектура «Фергюсон и Барделл»
- Бизнес-перспектива
- Прикладная перспектива
- Информационная перспектива
- Технологическая перспектива

III. Проект «Система управления ресурсами»

IV. Рекомендации

V. Вопросы и ответы

VI. Задания и ответственные

Чикаго • Детройт • Милуоки • Цинцинати • Индианаполис • Луисвилль

Четверо сотрудников группы ПА несколько минут показывали свои разработки и планы, способные повлиять на проект RMS. Билл, Бет и Сэм делали заметки в своих блокнотах и задавали вопросы, шла оживленная беседа. Джо рассказала, что проект RMS — основной в их планах на год, после которого, возможно, начнется работа нал проектом по созданию системы для обучения персонала и контроля за его квалификацией. Сэм заметил, что такая система отлично бы вписалась во вторую версию RMS.

Дженни Сакс просветила остальных о планах компании по сокращению используемых технологий, что поможет уменьшить затраты на обучение и сопровождение. Она также отметила, что тестовая группа уже работает с Windows 2000, но переход компании на эту операционную систему планируется не раньше осени 2000 года.

- Нам придется использовать COM+ в первой версии RMS? Понадобится ли Windows 2000 в первой или второй версии проекта? - поинтересовалась Бет у Сэма.

Сэм покачал головой:

- Так мы упростим некоторые компоненты прикладного уровня, но это недостаточная причина, чтобы ради этого менять график развертывания, — ответил он. — Будем работать с тем, что уже есть, а когда выйдет Windows 2000, перепишем эти компоненты заново.

Дик рассказал о некоторых обнаруженных им пробелах:

— Главное расхождение наблюдается между планами и ресурсами. Вообще-то почти каждый менеджер проекта недоволен тем, что невозможно сопоставить ресурсы и проекты в масштабах компании. Некоторые из них надеются, что RMS поможет в этом, — Он вынул из своей сумки пакет с бумагами и передал его разработчикам. - Часть моей работы — сопоставить предложенную модель данных и словарь. Ее результаты еще не утверждены, но мне кажется, что вы захотите взглянуть на них и проверить, соответствуют ли они вашим наработкам по RMS.

Три разработчика просмотрели предложенные им документы.

— Дик, это великолепно! — воскликнула Бет. — Многое совпадает с нашими данными, но есть и кое-что новенькое. Мы изучим твои документы получше, и я пришлю тебе наши комментарии,

- Кевин, что скажешь о направлении развития бизнеса? — спросил Дэн. — О каких стратегических планах нам нужно знать?

— Самое главное — это расширение компании, — ответил Кевин, раздав свои документы. — Как видите, через полтора года наша компания будет размешаться в десяти офисах, а не в шести, как сейчас. Также мы примем на работу около 250 человек, таким образом, штат увеличится более чем на 30%,

— Ого! Порядочный прирост, — удивился Сэм. — И у всех этих людей будут расписания? Это многое изменит, возможно, даже некоторые готовые планы.

Он взглянул на Бет и Билла, они кивнули, соглашаясь.

— Отличный момент, чтобы перейти к следующему пункту повестки дня, — сказал Дэн. — В чем должно заключаться сотрудничество групп RMS и ПА? Билл, почему бы тебе не ввести членов группы ПА в курс дела?

Билл вкратце описал различные направления проекта, рассмотренные разработчиками, в том числе и вопрос об использовании Win32- или Web-клиентов. Затем ему пришлось ответить на несколько вопросов: сотрудники группы ПА выясняли связь этих решений со своей работой. Потом Дэн сказал:

- Похоже, что обе группы достаточно хорошо разобрались в работе друг друга. Теперь давайте перейдем к согласованию вопросов и рекомендаций.

- Какие сведения от группы ПА могут повлиять на ваш проект? — обратился он к разработчикам.

— Думаю, что самое главное — это расширение компании, — сказал Сэм. — Мы планировали, что приложением будут пользоваться не больше 300 человек. Мы считали, что этого достаточно, учитывая разницу часовых поясов между офисами и то, что расписания обычно обновляются с пятницы до понедельника. В таком случае у нас был выбор между Win32- и Web-клиентами. Теперь, когда планируется прибавление еще 100 или 200 пользователей, нам придется ограничиться Web-клиентом, переложив как можно больше функций на серверные приложения и таким образом уменьшив сетевой трафик.

- Но, Сэм, — нахмурившись сказал Дэн, — насколько я помню, некоторые важные функции Win32-клиента не реализованы в Web-клиенте по причине сложности. Если вы ограничитесь Web-клиентом, не слишком ли увеличится время разработки?

Сэм и Бет дружно кивнули. Все замолчали, обдумывая проблему. Дик Каштан даже перестал рисовать в блокноте.

— Логический проект у тебя с тобой? — спросил он у Билла.

— Конечно. А зачем он тебе? — ответил Билл, открывая свою сумку.

— Дай-ка посмотреть, — сказал Дик. Билл передал ему бумаги, и Дик минуту изучал их.

- Кажется, я нашел решение, — сказал он, показывая свой рисунок. — Вот различные функции приложения RMS. Посмотрите сначала на пользовательский уровень, потом на прикладной и, наконец, на уровень данных и скажите, вам ничего не бросается в глаза?

Все, в том числе Билл, Бет и Сэм, некоторое время разглядывали рисунки. Наконец, Джо сказала:

— Я не разработчик, но, похоже, тут два отдельных приложения, использующих одну и ту же базу данных.

- Что ты имеешь в виду? — спросила его Бет.

Джо взяла рисунки и переложила их в другом порядке.

— Видишь? Вот тот набор объектов работает только при вводе времени, его проверке, рассмотрении и утверждении расписания, — сказал она и показала на вторую группу рисунков. — А эти объекты работают с ресурсами: определяют их назначение, выделяют их проектам и проверяют их наличие.

Она встала, сложив руки:

— Очевидно, есть некоторые общие данные, так как в обоих случаях мы работаем с одними и теми же ресурсами и проектами но назначение этих групп объектов разное.

Дик медленно кивал головой, слушая обсуждение с закрытыми глазами. Вдруг Сэм щелкнул пальцами:

- Вот именно! — сказал он, повернувшись к Биллу и Бет. — Мы напишем *оба* клиента! С помощью Web-клиента пользователи будут вводить время, а с Win32-клиентом будут работать только менеджеры. Ведь они распределяют ресурсы, находясь в офисе, и их число ограничено. Для этого можно применить Transaction Server, что гарантирует получение всех расписаний, даже если сервер «зависнет».

Сэм был доволен, что придумал такой замечательный выход из затруднительного положения. но Билл не разделял его чувств:

— *Два* клиента? Я не уверен, что мы один-то сделаем вовремя, а ты хочешь два! Тогда мы точно не уложимся ни в какие графики!

- Уложимся, Билл, — сказала Бет. — Если мы распределим сервисы таким образом, мы сумеем разделить и работы, связанные с каждым уровнем и клиентом, а значит, сможем работать над ними параллельно. Ведь есть еще желающие поучаствовать в нашем проекте. Поэтому мы сделаем больше за меньшее время и при этом не испортим масштабируемость приложения.

- Что скажешь? — с сомнением в голосе спросил Билл у Дэна.

- Я думаю, это сработает, Билл. Мне понравилась идея специализированного Web-клиента, его проще написать. Что касается параллельных действий, то это вопрос не технический, а управленческий. А я знаю хорошего менеджера, который может включиться в наш проект.

Билл видел, что Бет и Сэму очень понравилось новое решение, и он сдался, подняв руки вверх:

– Хорошо, хорошо, мы попытаемся. Но учтите, если ничего не получится, будем делать все по-моему — эмуляция терминала и ассемблер, — добавил он, вызвав всеобщий смех.

– Ты помнишь, Билл, что на фазе планирования создается пробная версия приложения? — спросил Дэн, — Неплохо бы, чтобы пара энтузиастов изучила все тонкости, прежде чем мы окончательно перейдем к концепции двух клиентов. Не надо затейливого интерфейса, полной функциональности — пусть просто проверят, как такой Web-клиент повлияет на работу наших серверов.

– Вы возьметесь за это дело? — спросил Билл у своих программистов. Они взглянули друг на друга и согласно кивнули.

— Славненько, так и сделаем. Вот отличный пример того, как производственная архитектура компании влияет на разработку, — сказал Дэн и повернулся к членам архитектурной группы. — А что вы скажете? Рассказ разработчиков как-нибудь повлияет на вашу работу?

– Конечно, — ответила Дженни. — Мы изучали вопросы, касающиеся нашей интрасети. Она нужна, но не как простое хранилище статичной информации. Мы обсуждали, стоит ли перенести в интрасеть некоторые бизнес-процессы, и уже приняли решение. Теперь наша задача — выбрать технологии для этого переноса.

— Другой вопрос, который изучал отдел долгосрочного планирования, — добавил Кевин, — расширение сферы обслуживания за пределы наших офисов. Через девять месяцев мы собираемся проводить национальную рекламную кампанию. Здесь возникает вопрос, как во время кампании организовать связь между столькими людьми. Дженни и Тим уже подумали над этим, разработав несколько способов выхода из положения. Я думаю, что на этом совещании нам надо обсудить не только как укладывать провода, но и что по ним передавать. Я более чем уверен в способности группы информационных технологий придумать творческое решение бизнес-проблем.

– Кевин, кажется, это самые добрые слова, которые я когда-либо слышал от тебя в адрес других отделов! — сказала Джо, улыбаясь. – Надо почаще забирать тебя от генерального директора.

Все замолкли, ожидая, чем ответит Кевин на этот укол, но он просто сказал: «Иногда даже вундеркинду нужно учиться, чтобы не сбиться с пути к успеху», — и улыбнулся, услышав чей-то одобрительный шепот.

— Думаю, что это совещание полезно для обоих отделов, — продолжил Дэн. — Через неделю архитектурная группа закончит первую версию плана архитектуры, и, я уверен, они захотят увидеть окончательные функциональные спецификации. Есть вопросы или замечания?

Дик Каплан, до этого момента задумчиво смотревший на стену и не замечавший разговора, обратился к Дэну;

— Я бы хотел обсудить кое-что с тобой и группой разработчиков, если у вас есть время. У меня есть несколько идей, касающихся вашего проекта.

- Вы можете задержаться на минутку? — спросил Дэн разработчиков, и те кивнули в знак согласия. — Хорошо. Если больше нет вопросов, то все свободны.

Пока члены архитектурной группы собирали свои вещи, Дик подошел к Дэну и разработчикам, обсуждавшим планы по созданию двух клиентов.

- Ну, Дик, мы все — внимание, — сказал Дэн. — Ты уже нам помог, что же ты еще придумал?

— Что вы собираетесь использовать для хранения данных? — спросил Дик у Бет.

- Конечно же, SQL Server, — ответила она. — Это же корпоративный стандарт.

— А мне кажется, что есть вариант получше.

— Дик, ты знаком с продуктами Microsoft не так хорошо, как мы, и я не собираюсь использовать какую-то другую базу данных, — раздраженно сказал Билл. — Я лишь четыре месяца как избавился от приложений для dBase III. Мы выбрали SQL Server по многим причинам, и я не хочу изучать другие базы данных, программировать для них и сопровождать их.

— Билл, ты не дослушал меня до конца, — спокойно ответил Дик. — Я не говорил о другой базе данных. Я говорил о другом хранилище данных. Чувствуешь разницу?

— Дик, ты говоришь загадками, — совсем рассердился Билл. - Быстрее выкладывай свои соображения!

— Он прав, Билл, — вмешался Дэн, — есть разница. Мы считаем, что единственно возможное место для хранения данных — база данных, но это не совсем так. Ведь данные хранятся в файловых системах, сообщениях электронной почты, отсканированных документах, в Интернете и во многих других местах.

— Вот именно, — сказал Дик и повернулся к Бет и Сэму. — Какие основные элементы данных являются общими для обоих потоков приложения RMS?

- Ресурсы, проекты и время, — почти одновременно ответили они.

Дик кивнул и продолжил:

— А какое доступное всем хранилище мы уже используем для хранения информации о контактах и расписаниях?

На мгновение все задумались, но вдруг заговорил Сэм:

– Exchange! На нашем сервере Exchange хранятся сведения о пользователях и их расписаниях.

— Мне кажется, что вы могли бы применить для хранения данных Exchange, — продолжил его мысль Дик. — Особенно, если вы вспомните о возможности этого приложения тиражировать информацию по предприятию и о массе готовых объектов и сервисов.

– Ты еще не упомянул о его интеграции с Web-технологиями, – сказал Дэн. — Отличное предложение. Ты сегодня работаешь за двоих.

Раздражение Билла прошло и, выходя из комнаты совещаний, он спросил у Дика:

– Откуда ты столько знаешь о проблемах программирования?

— У каждого из нас свои интересы, причем зачастую окружающие о них и не подозревают. Это как с Луной — одну сторону мы видим, а другая скрыта от нас, — спокойно ответил ему Дик. — Никто не знает все о других. Не беспокойся из-за этого.

Когда они подошли к кабинету Дика, Билл, поддавшись порыву, спросил:

– У тебя здесь есть копия твоей диссертации? Дай почитать. Похоже, это интересно.

– Не думал, что тебе интересен Кант, — удивленно сказал Дэн,

– Хочу узнать побольше о человеке, рассуждающем о скрытых возможностях человека. Кроме того, я слышал, что Кант был отличным рыболовом.

Физическое проектирование

Билл и другие разработчики почти весь вторник работали над физическим проектом, пытаясь подготовить все документы к совещанию в среду. С утра им помогал и Дэн, но в 10:30 его вызвал Джим Стюарт.

После обеда Дэн заглянул к разработчикам:

– Как дела?

– Неплохо, — ответил Билл. — Мы определили два хранилища данных: Server и Exchange. Разработали элементы данных и упорядочили их. подготовили краткие описания большинства компонентов и их интерфейсных контрактов. Сейчас работаем над комплектацией.

– Что насчет стратегии развертывания? — спросил Дэн.

– Есть кое-какие идеи, — выглянул из-за компьютера Сэм, — но хотелось бы тщательнее протестировать прототипы, перед тем как включить их в окончательный план развертывания.

– У вас готовы новые прототипы?

— Да, — кивнул Билл. — Сэм и Бет остались вчера вечером и набросали прототип, который мы сейчас используем для составления планов,

- Впечатляет, — сказал Дэн. — Как думаете, будет, что показать завтра? Как насчет графика?

— О, что показать — *будет,* — усмехнулась Бет. — Но вот *что* это будет — уже другой вопрос.

- Что касается графика, — добавил Билл, — я над ним работаю с помощью Бет и Сэма, но. боюсь, у меня для тебя плохие новости, Мы не видим, как можно уложиться в предварительные сроки, определенные на стадии «Анализ».

— Даже учитывая резерв времени? — спросил Дэн.

— Да, — кивнул Билл, — и даже работая над разными компонентами параллельно. Два клиента, два хранилища данных, интеграция с бухгалтерским пакетом для выписки счетов — это слишком много. Нам потребуется как минимум еще месяц.

Дэна как будто обухом по голове ударили.

«Только этого мне не хватало, особенно после утреннего совещания», — мрачно подумал он и, вздохнув, сказал вслух:

- Билл, я уже говорил, что мы будем использовать планирование «снизу — вверх». Если вы уверены, что нужен дополнительный месяц, я вас поддержу. Но мы должны проинформировать заказчика и пользователей о том, что им придется подождать. чтобы получить приложение со всеми необходимыми функциями. У меня только один вопрос - вам нужен ровно месяц или, может быть, два?

- Только один. Дэн, — поразмыслив, ответил Сэм. — В прототипах мы не столкнулись с фатальными проблемами, поэтому одного месяца хватит.

— Я на вас рассчитываю. — тихо сказал Дэн и удалился.

— Что это с ним? — спросил Сэм Билла. — Я его никогда таким не видел.

Билл неопределенно хмыкнул, он тоже не понимал, что случилось. «Наверно мы узнаем все завтра», — подумал он и занялся расписаниями.

На следующее утро все, кто имел отношение к разработке RMS, собрались в зале совещаний. но им пришлось подождать: Дэна не было. Машина Дэна стояла на парковке, свет в его кабинете горел, но, где он сам, никто не знал. Наконец в 8:15, когда Тим уже второй раз звонил Дэну на пейджер, директор по ИТ явился с блокнотами в руках и кислой миной, Он положил свои блокноты на стол, сел и некоторое время пристально смотрел на других.

Даже когда Тим нараспев сказал: «И тебе, Дэн, доброе утро!», Дэн не обратил на него внимания. Затем он встряхнулся и произнес:

- Извините. В мыслях я еще на прошлом совещании. Извините.

- Мы уже начали беспокоиться, — сказала Джейн.

— Да, — добавила Марта. — Мы подумали, что ты берешь пример с Тима.

Тим показал ей язык, что вызвало всеобщий смех. Марта подошла к кофеварке и спросила:

- Хочешь кофе с пончиком?

- Неплохо бы, — ответил Дэн, раздавая всем новые материалы. — Я с семи часов на ногах и даже не перекусил.

Марта принесла ему чашку, он отхлебнул кофе:

— Спасибо, Марта. Из-за этого совещания я не только не успел поесть, но и опоздал к вам. Мы не могли прийти к согласию. Когда я ушел, обсуждение все еще продолжалось, но я сказал, что вы меня ждете.

- Надеюсь, ничего страшного не произошло, — сказала Марта, передав Дэну пару пончиков и сев на свое место.

- Ну, Марта, я уже давно понял, что если нет проблем, то ты либо ничего не делаешь, либо умер. Но к несчастью, проблем уже множество, поэтому нам нужно постараться, чтобы не появились новые, — он откусил пончик, некоторое время задумчиво жевал, затем продолжил: — Возможно, и мы столкнемся с проблемами. Но, надеюсь, все будет хорошо. Сейчас же нам надо завершить проект и составить план. Начнем.

Он отряхнул сахар с рук и прошелся вдоль стола.

- Билл, за физический проект отвечают разработчики, так почему же ты ничего не рассказываешь о своей работе?

Билл встал и раздал документы, посвященные различным частям проекта.

- Наша встреча с группой ПА в понедельник была очень плодотворной. Изучая вопросы, связанные с масштабируемостью, мы решили создать как Web-клиент...

— Отлично! — вставила Мэри-Лу.

— ... *так и* Win32-клиент, — закончил Билл, ликующе посмотрев на Мэри-Лу.

- Погоди, — удивилась Джейн. — вы собираетесь делать *два* клиента? Зачем?

— Я думаю, что вам стоит взглянуть на вторую и третью страницу, где описаны все процессы и технологии, — ответил Билл и подождал пока все члены группы изучат схемы.

- Ну и ну, для хранения данных о ресурсах и контроля за временем вы используете Exchange, — сказала Мэри-Лу, изучив документы. — Из-за этого некоторые рабочие пчелки станут несчастными.

— Почему? — спросила Джейн.

– Потому что они не хранят свои календари в Exchange. Они используют бумажные расписания, автономные органайзеры и т. п. Им не захочется все менять.

— Я гарантирую, что мои сотрудники с радостью откажутся от бумажных календарей, записок и начнут полагаться не на свою несовершенную память, а на автоматический инструмент, который сократит их двухчасовые планы до нескольких минут, — громко сказал Тим. — А если они смогут делать все это дома через Интернет, то наверняка решат, что умерли и попали в рай.

Фергюсон и Барделл

Строительство • Проектирование • Управлениепроектами

Проект создания системы управления ресурсами

Повестка дня

Дата: 14 апреля 1999г **Тема:** планирование 3

I. Уточнение повестки

II. Физический проект
- клиенты
- потоки
- хранилища данных

III. График разработки

IV. Предварительные функциональные спецификации

V. Предварительный план проекта

VI. Предварительный график проекта

VII. Пересмотр документа оценки рисков

VIII. Вопросы и ответы

IX. Задания и ответственные

Чикаго • Детройт • Милуоки • Цинцинати • Индианаполис • Луисвилль

- А Джим Стюарт придет в восторг от того, что мы придумали еще один способ использования Exchange помимо электронной почты, — сказала Джейн. — Мы всучили ему Exchange под видом базы для решений масштаба предприятия, но кроме папки для резервного копирования пока ничего не сделали.

— Итак, мы говорим, что изучили все преимущества и недостатки данной технологии и решили, что она лучше остальных как с точки зрения бизнеса, так *и с* технологической точки зрения, — сказал Тим, откинувшись на спинку стула и улыбаясь.

- Ну, Тим, — сказал Дэн. — Похоже, ты наверстал упущенное.

- Хотя, как и раньше, не без посторонней помощи, — ответил ему Тим, заложив руки за голову.

- Тем не менее ты нас догнал, чему я очень рад. Кстати, ты прав. Вопросы баланса между достоинствами и недостатками различных технологий разработчикам приходится решать на этапе физического проектирования. Вернемся к нашему обсуждению, пусть Билл продолжит,

Следующие 20 минут группа работала над проектом, изучая его компоненты, их объекты и сервисы. Когда Билл объяснил, как многоуровневое проектирование наряду с некоторыми технологиями Microsoft позволит создать пул подключений к хранилищу данных, заговорила Мэри-Лу:

- Так вот, о чем весь день думал Сэм! Это же значительно увеличит масштабируемость, и проблемы роста начнутся гораздо позже.

— Именно так, — сказал Дэн. — Кроме того, такой метод разработки позволит нам изменять части проекта, не затрагивая его в целом. А некоторые компоненты мы сможем применить в других приложениях, за счет чего будущие проекты будем разрабатывать быстрее.

- Ну, а что со временем, чиф? — спросил Тим. — Как выглядит графиктеперь, когда есть два клиента и несколько хранилищ данных? И почему в нем не упоминается Transaction Server? Кроме того, я не понимаю, как нам удастся выполнить всю работу к намеченному сроку, даже используя всех своих людей.

- Вообще-то, мы и не сможем, — хмуро сказал Билл. — Я уже говорил об этом Дэну. Как я ни пытался, мне не удалось найти способ закончить работу в срок. Нам понадобится дополнительный месяц.

— *Но у вас его нет!* — прогремел голос из коридора, и в зал вошел Джим Стюарт.

— Джим, какой сюрприз! — сказал Дэн, не поднимаясь со стула. Хотя он старался не повышать тон, в его голосе чувствовалось раздражение, замеченное всеми, кроме Джима. Не обращая внимания на Дэна, Джим обратился к Биллу:

— Повторяю, мы не можем ждать приложение RMS еще целый месяц. Вы должны закончить его вовремя, а то и раньше! — финансовый директор все больше возбуждался. Он ударил кулаком по столу и почти закричал: — А если вы не справитесь с этой задачей, я найду того, кто справится!

— Хватит! — резко сказал Дэн. — Во-первых, сэр, это мое совещание, и вы не должны врываться на него без стука. Во-вторых, Билл работает на меня, а не на вас, поэтому незачем угрожать моим сотрудникам. *И* наконец, я работаю не *для* вас, а *с* вами. Если вам непонятно, что я сказал, нам придется поговорить отдельно, без моих сотрудников. Вам все ясно?

В комнате стало тихо. После довольно долгой паузы, Джим закрыл лицо руками и прислонился к шкафу.

— Извини меня, Дэн, — сказал он.

— Мистер Стюарт, садитесь на мой стул, — встал со своего места Тим.

Джим сел, сделал глоток воды, принесенной ему Тимом, и, вздохнув, повторил:

— Прости, Дэн. Кажется, я вышел из себя. Я спустился посмотреть, работаете ли вы еще, Совещание совета директоров закончилось, и я хотел рассказать вам о принятых решениях. Но когда Билл сказал о дополнительном месяце, я сорвался.

Он снова вздохнул и обратился к Биллу:

— Похоже, я только говорю глупости, а потом извиняюсь. Если я буду продолжать в том же духе, понадобится специальное приложение для учета моих срывов. Извините, Билл.

— Все в порядке, пока число оскорблений и извинений одинаковы, — сердито сказал Билл, однако они пожали друг другу руки.

— Ну что ж, Джим, — Дэн оперся на шкаф, — наше совещание практически закончено. Так что успокойся и расскажи нам, почему проект RMS стал так важен, что задержка на один месяц вызвала у тебя истерику.

— «Истерика» — точно сказано. Именно это слово описывает мое состояние, — он снова вздохнул и спросил: — Кто-нибудь из вас знает, что произошло в пятницу в отделе продаж?

Мэри-Лу и Джейн обменялись взглядами, и Джейн сказала:

— Мы кое-что слышали, но без подробностей. Что-то связанное с потерей заказчика?

— Да, но тут важно, кого и как мы потеряли, — ответил Джим. Он встал и начал мерить шагами комнату. — Дело вот в чем. Один из наших сотрудников из отдела продаж работал над проектом большого дома неподалеку отсюда. Если вы когда-либо имели дело с отле-

лом продаж, то представляете, о чем я говорю. Они подготовили планы и предварительные расценки два месяца назад, передали их заказчику, после чего тот замолчал... на месяц. Ему намекнули, что неплохо бы прореагировать, но заказчик молчал. Таким образом, мы оставались в полном неведении. До пятницы. В пятницу же мы получили сообщение от генерального директора компании-заказчика. Он сказал, что проект — наш. *если* мы сможем начать работу над ним в понедельник. Оказалось, что их перестанут финансировать, если они не продемонстрирую какую-нибудь активность. Что еще хуже, мы должны согласиться с этим к 15:00 в пятницу. И кроме того, от нас потребовали *гарантии*, что в понедельник мы сможем выделить для работы над этим проектом отдельную группу. А если наши сотрудники вовремя не прибудут на место, нам придется заплатить *значительный штраф*.

Джим перестал расхаживать по комнате, положил руки на спинку стула и обернулся к собравшимся.

– Наш сотрудник был просто разъярен. У него оставалось всего 5 часов, чтобы дать ответ, а ему еще надо было удостовериться, что мы справимся с задачей. Он обошел начальников всех отделов, пытаясь набрать людей. Кое-как ему это удалось — многим нравятся крупные проекты. Но в результате сформировать группу полностью мы не смогли. Немного недоставало инженеров и не хватало опытных менеджеров проекта и инженеров-строителей. Мы решили не рисковать, поэтому в 14:30 нам пришлось, к своему стыду, отправить формальный отказ.

Все увлеченно слушали историю Джима, только Билл удивился:

– Мы уже не раз теряли проекты из-за нехватки ресурсов! Что тут такого? Это происходило раньше и будет происходить в будущем. И вообще, у них вместо мозгов опилки, если они думают, что можно прыгнуть так высоко, почти не разбежавшись.

– Подожди, Билл, это еще не все, — тихо сказал Дэн. — Это еще не конец истории.

— Да! — подтвердил Джим, сев за стол. — Ты прав, Билл, мы теряли заказы и раньше из-за нехватки ресурсов. Этот же случай так ужасен, потому что... у нас *были* ресурсы. Мы просто о них не знали.

– М-м-м, ненавижу, когда такое случается, — сказал Тим, гримасничая.

– Оказывается, — продолжил Джим, — два наших менеджера проекта закончили свою работу, но это нигде это не было зафиксировано. По их графикам выходило, что они будут заняты как минимум еще месяц, но в одном из их проектов возникли проблемы с поставками, а второй они завершили раньше срока. Они отправили начальнику отдела кадров сообщения о том, что вернутся на этой неделе, но того не оказа-

лось на работе из-за болезни, и он не прочитал их. Единственный способ отслеживать занятость менеджеров проекта — по таблице на стене кабинета их начальника, но он в эту таблицу не внес изменений,

— А что с инженерами-строителями? — спросила Марта.

— Оказалось, что в таблицу твоего начальника занесены не все сведения о квалификации подчиненных. Пять наших инженеров-электриков имели степень магистра по электротехнике и степень бакалавра по строительству. Двое из них некоторое время работали инженерами-строителями, прежде чем получили степень магистра. Но узнали мы обо всем этом слишком поздно,

— А когда все это открылось, мы еще могли отменить свой отказ? — спросил Дэн. Он все еще стоял у шкафа, скрестив руки,

— Секретарша позвонила начальнику отдела кадров домой около 2 часов дня, — вздохнул Джим. — После этого он проверил почту и обнаружил сообщения менеджеров. Нам он позвонил в 14:45.

— А как вы узнали о квалификации инженеров-строителей? — снова спросила Марта.

— Один из них услышал наши жалобы на нехватку инженеров-строителей и сообщил о себе и двух других. Затем и об остальных стало известно. А времени было уже 16:45. Вот тогда стало по-настоящему жарко, — Джим качал головой, все еще потрясенный беспорядком в компании,

— Насколько крупный контракт мы потеряли? — тихо спросила Джейн.

— 4,2 миллиона долларов, — еле слышно ответил Джим, уставившись в стол.

— Ну и ну, — свистнул Тим. — Можно было бы заставить всю комнату новыми серверами.

— Это тоже еще не все, — мрачно сказал Дэн. — Расскажи остальное, Джим. Расскажи им о прошедших выходных.

— Вы все равно бы узнали через день-два, так что, я думаю, ничего не случится, если я расскажу вам сейчас, — сказал Джим и сделал паузу. — Когда весть об этом происшествии достигла высшего руководства, ситуация накалилась. Совет директоров собирался в выходные, в понедельник и сегодня утром. И эти совещания не были приятными, уж поверьте мне. В результате исполнительный директор «оставляет свои пост в связи с переходом на другую работу». Очевидно, эта неудача была последней каплей. Его место займет Джо Браун, но только когда завершит свою теперешнюю работу. Сейчас же я не только финансовый директор, но и временный исполнительный директор. Это основные перестановки в руководстве, но грядут и другие, в том числе они коснутся и менеджеров проекта. У управляюще-

го комитета, в который входит и Дэн, есть всего две недели, чтобы подготовить первый вариант полностью документированного методического руководства, Мы пытались убедить генерального директора, что он не может диктовать сроки выполнения таких проектов, но у него не было настроения выслушивать наши доводы в пользу проектирования «снизу — вверх». А когда ты ушел, Дэн, началось обсуждение RMS. Непонятно, кто отвечает за этот проект, а поскольку ты все еще ходишь в любимчиках, то избежал наказания. И последнее, что сказал мне босс: «Стюарт, мы должны усиленно работать над проектом RMS, потому что мы больше не можем терять такие контракты. Я назначаю тебя ответственным за проект RMS. А если ты не сможешь его выполнить, я найду того, кто сможет!»

Все вздрогнули от таких жестких слов. О характере генерального директора ходили легенды.

— Так вот почему ты вышел из себя, когда услышал о задержке проекта? — тихо сказал Билл.

— Не очень хорошо получилось, ведь вам обоим я доверяю больше, чем кому-либо другому, — ответил Джим, не взглянув ни на Билла, ни на Дэна. — Но я просто не знаю, что делать. Если мы не выполним проект RMS в намеченные сроки, то «оставить свой пост в связи с переходом на новую работу» придется кое-кому еше.

Дэн решительно подошел к доске и нарисовал треугольник. Его стороны он подписал: «Функции», «Ресурсы» и «График». Затем он повернулся к группе и сказал:

— Джим, ты помнишь это?

— Конечно, — ответил Джим, чуть улыбнувшись, — это золотой треугольник. Одна из лучших концепций, которую я когда-либо знал.

— Он может пригодиться не только в начале проекта, Джим. Используем его при внесении в проект изменений, — Дэн нарисовал рядом со словом «График» большой минус. — Чтобы написать приложение, нам понадобится дополнительный месяц. Но ты говоришь, что у нас его нет. Значит сторона «График» уменьшается. Поэтому, чтобы восстановить равновесие в треугольнике, мы должны что-то изменить. Например, урезать функциональные возможности приложения, или привлечь дополнительные ресурсы, или изменить проект, или откорректировать график каким-то другим способом,

— Мне нравится наш проект, — продолжил Дэн, вернувшись на свое место у шкафа, — и я не хотел бы его менять. Поэтому нам остается либо сузить набор функций приложения, либо привлечь новые ресурсы. Но при этом не забывайте про мифический человеко-месяц. Ведь дополнительные ресурсы, привлеченные на такой краткий срок и в такой небольшой проект, могут не дать нужного результата.

Ты — заказчик, и тебе решать, как восстановить равновесие. Мы можем многое предложить, но решение — за тобой. Как поступим?

Взгляды всех участников совещания обратились на Джима. Наконец, не поднимая головы, он произнес:

– Уберем бухгалтерский учет.

– Что? — воскликнула Джейн. — Это же одна из главных функций RMS! Ее нельзя выбросить! Как мы будем выписывать счета?

– Так же, как и сейчас, — ответил Джим, в чьем голосе снова появились сердитые нотки. Он остановился, глубоко вздохнул, повернулся к ДжеЙн и продолжил: — Джейн, учет очень важен для тебя и для твоих сотрудников, но взгляни на это шире. Мы же оставляем сведения о расписаниях. У нас все-таки будет база данных ресурсов, контроль за ресурсами и автоматическая генерация расписаний. И все это мы сделаем в срок. Это решение — правильное, и я, как заказчик проекта и как действующий исполнительный директор, его принимаю. Каким бы оно ни было трудным, но я его принимаю.

Он повернулся к Дэну и добавил:

— Вроде бы мы говорили о второй версии? Нельзя ли сделать бухгалтерский учет одним из ее приоритетных направлений?

— Можно. Мы планируем начать работу над второй версией сразу же по окончании работы над первой. — согласился с ним Дэн.

– Так и сделаем, — сказав это Джим, встал и направился к Дэну, чтобы скрепить рукопожатием решение. — Спасибо за помощь и понимание. Извините, что я уже ухожу, но мне надо потушить еще не один пожар, а у вас вроде бы все хорошо. Думаю, что мы достигнем намеченного этапа в пятницу. С нетерпением буду ждать от вас новостей. Удачи!

— М-да, не хотелось бы мне быть на его месте, — сказал Тим, когда Джим ушел.

— Да, похоже, несколько месяцев он будет балансировать на краю пропасти, — сказал Дэн, возвращаясь на свое место во главе стола. — Наша же задача сделать все возможное, чтобы подготовить RMS вовремя и не стать камнем, который утянет Джима вниз.

— Вот, что я вам скажу, — добавил Дэн, взглянув на повестку дня, — Я хочу кое-что изменить в ней. Я полагал, мы сможем разработать черновики основных документов, но из-за такой задержки все мы не подготовим. Поэтому сегодня мы рассмотрим некоторые части функциональных спецификаций, которые я распределю между вами. Завершив работу над своим разделом, вы вышлете его остальным по электронной почте. Я же соберу их в единый документ, который мы утвердим в пятницу.

После небольшой паузы он продолжил:

— Билл, эти изменения сильно повлияли на твои план и график, а все брали его за основу для своих планов. Поэтому не стоит работать над основными планом и графиком проекта, пока возможны поправки. Я хочу, чтобы ты исправил свои планы и расписания и как можно быстрее разослал их всем остальным. Мы внесем изменения в свои документы, а я объединю их в один. Что скажете?

— Ломать — не строить, — ухмыльнулся Билл. — Я смогу разослать исправленные документы уже сегодня.

— Великолепно, — Дэн сделал пометки в блокноте. — Давайте распределим функциональные спецификации. Джейн возьмется за краткий обзор проекта и описание его целей. Мэри-Лу определит требования и составит резюме по использованию. Билл и Тим поработают вместе над функциональными возможностями и зависимостями. Я займусь графиком, а Марта изучит риски. Ее работа нам пригодится в пятницу при составлении документа оценки рисков. Таким образом, в пятницу у нас будут три модели проекта, функциональные спецификации, план проекта и его график, которые мы сможем изменить в случае необходимости. Затем мы займемся оценкой рисков, руководствуясь выкладками Марты. И наконец, вместе с Джимом обсудим вопросы, связанные с очередным этапом. Вопросы есть?

Вопросов не было, поэтому Дэн закончил совещание словами:

— Не забудьте прислать ваши документы как можно раньше, чтобы у меня было время объединить их и отослать вам обратно к завтрашнему вечеру. Дайте мне знать обо всех ваших незаконченных делах, которые я постараюсь передать другим. Вы должны сконцентрироваться только на этом проекте.

— У меня дома куча нестиранного белья, — сообщил Тим, вызвав всеобщий смех.

— Вот поэтому мы и выдали тебе портативный компьютер, — ответил ему Дэн. — Возьми его с собой в прачечную и работай в многозадачном режиме. Может, когда будешь смотреть за тем, как вертится в стиральной машине твое белье, вспомнишь о модели процесса.

— Мой способ стирки больше похож на живой пример управления рисками, — возразил Тим, быстро схватив пончик перед тем, как уйти.

— Если ты будешь питаться только пончиками, — сказала Марта, выходя, — тебе не придется ходить в прачечную, ты будешь периодически и довольно часто менять весь свой гардероб.

Тим только усмехнулся и последовал за ней.

Этап «Одобрение плана проекта»

— Итак, мы все учли?

Совещание группы RMS длилось уже час, были изучены разделы функциональных спецификаций, внесены незначительные изменения, но большей части документов еще не касались.

— Сначала я волновалась из-за разделения функциональных спецификаций на части, — сказала Джейн, — но потом, когда я работала над своим разделом, мне показалось, что все это мы уже сделали. И, похоже, то же почувствовали остальные.

— Мы так долго обсуждали проект, — согласился с ней Тим, — что записывать его на бумагу, как мне показалось, уже поздно.

Он взглянул на Дэна и сразу добавил:

— Но это, конечно же, не так.

— Хорошо поработал, Тим, — сказал Дэн и обратился ко всем остальным. — Думаю, что так работать можно, ведь я сказал, что функциональные спецификации должны полностью описывать проект, но не более того. Я ценю, что никто не пустился в многословные описания.

— Легче было составлять план и расписание после того, как ты получила исправленные планы Билла? — улыбнулся он Джейн.

— Нет, стало еще труднее, — ответила Джейн, и несколько человек кивнули в знак согласия. — В функциональных спецификациях описано то, что мы уже обсуждали. Наши же планы и графики относятся только к нашей сфере деятельности. Я никогда не обучалась на менеджера проекта, поэтому мне было сложно оценить усилия, необходимые для выполнения моих задач. Кроме того, я никогда не составляла план взаимодействия и поэтому не знала, что в него стоит включить. Примеры из проекта, которым ты занимался на прежней работе, неплохи, но сам проект сильно отличается от нашего, и мне было сложно его использовать.

— И у меня такая же история, — добавила Марта. — Я, конечно, ценю твою помощь при составлении плана тестирования, но такая работа все же сильно отличается от того, чему меня учили. Я до сих пор не уверена, что правильно все сделала.

— Честно говоря, ваши планы очень хороши для новичков в области MSF, — сказал Дэн. — Я их прокомментирую, внесу некоторые предложения и напечатаю заново. Давайте посмотрим, что написал я, и затем продолжим работу.

Еще целый час все рассматривали планы и графики, начав с планов разработчиков. Дэн уже подготовил основной график проекта, и когда он его раздал, все сразу же заметили несколько конфликтов и неточностей, Когда все в подробностях изучили расписание, исправили некоторые ошибки и сверили его с календарем компании, Тим, потянувшись, сказал:

– Боже мой, ну и работенка! Никогда не думал, что планирование проекта так выматывает, — он встал, чтобы выпить воды. — Кажется, сегодня к нам должен был присоединиться Джим?

— Сначала я отправлю ему окончательные документы, — ответил Дэн, не поднимая глаз, так как он переносил последние изменения в свою копию основного графика проекта. — Он изучит их, пока мы будем работать над оценкой рисков, и придет к нам в конце совещания, около 11.

Размашисто сделав последнюю пометку, Дэн вызвал помощника, чтобы тот отнес документы Джиму.

– Ну, вот и все! Сделаем перерыв минут на десять. Тим, не ешь больше пончики — мы сегодня обедаем с Джимом, за все платит «Фергюсон и Барделл».

– Не волнуйся, — усмехнулся Тим. — Еще ни разу пончики не отбили у меня аппетит. Вот серверы — бывало, а пончики — никогда.

После перерыва Марта рассказала о документе оценки рисков, Группа изменила некоторые оценки и добавила еще один риск — нереальных ожиданий менеджеров, сражаться с которым выпало на долю Джейн с помощью ее плана взаимодействия. Джейн притворилась возмущенной:

– Вот здорово, большое спасибо! Придется работать с большими шишками, — поняв, что она только что сказала, она смущенно посмотрела на Дэна. — Ой! Извини, Дэн, я не тебя имела в виду. Ты один из них, но на шишку не похож. По крайней мере, пока.

— Да ничего, — отмахнулся Дэн. — Я понимаю, что ты чувствуешь. Но если уж кого посылать к шишкам, так это тебя, Джейн.

– Ну что ж, пошли, — сказал Дэн, взглянув на часы. — Я знаю, где Джим будет нас ждать. Неплохо бы придти заранее.

Обернувшись к Биллу, он добавил:

— Один из наших рисков — производительность. Как продвигается работа с прототипом?

– Вообще-то она уже завершена, — гордо объявил Билл. — Бет моментально написала компоненты, основные были готовы еще вчера утром. Весь день мы их изучали, запускали разные тесты. Похоже, мы одновременно сможем обрабатывать запросы от 500 пользователей, а если добавим несколько серверов и распределим компоненты, то и это не предел.

— Интересно, это не ваше тестирование так замедлило печать моих документов? — спросила Мэри-Лу. — Когда вы этим занимались?

— С десяти до полудня, — улыбка постепенно сходила с лица Билла, — потом с 14:30.

— Похоже, — кивнула Мэри-Лу. — Я очень быстро напечатала десять страниц в 13:30. Но пять страниц в 14:45 печатались целую вечность.

— Вот и еще одна потенциальная проблема, — подумал Дэн, но решил не говорить этого Биллу. Отметив про себя, что стоит спросить его на следующей неделе о тестовой среде, Дэн улыбнулся и сказал: — Чиф, это просто здорово! Ты действительно смоделировал столько пользователей?

— Да. Сэм написал тестовый сценарий, и мы хорошенько поработали над приложением, — ответил Билл, на лицо которого вернулась улыбка. — Бэт нашла способ поправить компонентную модель, и нам удалось выжать из программы дополнительные 12%.

— Отличная работа, Билл, — сказал Дэн. — Это касается и вас всех тоже. Я знаю, вы много работали сверхурочно, особенно с последними изменениями. И очень ценю вашу работу.

В этот момент Джим Стюарт просунул голову в комнату.

— Прошу разрешения подняться на борт, адмирал.

— Разрешаю, — одновременно ответили Дэн и Билл. Они изумленно посмотрели друг на друга и засмеялись.

— Извини, сила привычки, — сказал Билл.

— Да ничего, чиф. Мне стоило уступить моему старшему помощнику, — ответил Дэн и повернулся к подошедшему Джиму. — Что ты думаешь об окончательных версиях документов проекта, которые я тебе послал?

— Все очень хорошо, — удовлетворенно сказал Джим. — Этот проект документирован лучше всего из тех, которыми занималась наша компания. Если его реализация будет так же хороша, как и план, м-м-м... Но у меня есть несколько предложений. Посмотрите их сейчас? — добавил он, вынув из папки несколько бумаг.

— Конечно, — ответил Дэн. Рассказ Джима о четырех предлагаемых им изменениях вызвал короткую дискуссию, во время которой три исправления были приняты, а четвертое отвергнуто.

— А теперь вы готовы пойти пообедать? — спросил Джим.

— Еще нет, — ответил Дэн. Он взял схему и открыл ее на странице со списком из шести пунктов.

— Настало время проверить, достигли ли мы этапа «Одобрение плана проекта». Не забудьте, основные этапы — это важные события в жизни проекта. Именно в эти моменты заказчик и проектная группа приходят к согласию по некоторым вопросам, На фазе «Планирование» их шесть. Скажите мне, пришли ли мы к согласию и где оно зафиксировано.

Все молча смотрели на список. Затем Джейн встала, взяла у Дэна маркер и прошлась по каждому из пунктов списка, делая рядом с ними пометки.

- Что нужно сделать, чтобы удовлетворить требования бизнеса? Это в функциональных спецификациях. Функции приложения в порядке их приоритета? Тоже в функциональных спецификациях. Сколько потребуется времени? Это в основном графике проекта. Как и кто разрабатывает приложение? В основном плане проекта. Риски проекта? В документе оценки рисков. Этапы и документы оставшейся части проекта? В основном плане проекта, — сказала она и оглянулась вокруг. — Думаю, мы закончили.

- Нет, ты не права, — проворчал Билл. — Нам надо сделать еще кое-что, прежде чем считать фазу «Планирование» завершенной.

Все с интересом взглянули на него, и Билл продолжил:

- Мы должны решить, стоит ли продолжать работу,

Все согласно кивнули, вспомнив завершение фазы «Анализ», а Джим сказал:

- Все очень быстро меняется, не так ли, Дэн?

— Да, — улыбнулся тот, — но именно поэтому мы работаем над проектом именно так, и мы за него отвечаем. Думаю, мы готовы идти дальше, слово за тобой. Каково твое решение? Стоит ли начинать фазу «Разработка»?

— Да, — без колебаний ответил Джим. — Хотя кое-что мы из проекта выбросили, он по-прежнему нужен. Кроме того, мне нравится смотреть, как вы работаете, и любопытно, что за продукт вы выпустите.

- По правде говоря, мне тоже это интересно, — сказал Дэн и повернулся к своей группе, — Теперь все в порядке. Мы начинаем стадию «Разработка».

Пожав руку Джиму, он добавил:

- Кажется, я слышал что-то насчет обеда?

- Нас уже ждет столик, — ответил Джим, надевая пиджак и поправляя галстук. — Чем быстрее мы придем, тем быстрее начнем трапезу.

- Пойдемте быстрее! — сказал Тим, направляясь к выходу. Сразу же в комнате появились Бэт и Сэм.

- Рад, что вы смогли присоединиться к нам, — встретил их Дэн.

— Билл сказал, что, по его мнению, мы должны перестать питаться только пиццей и газировкой — настало время попробовать что-нибудь еще, — объяснил Сэм.

Дэн выключил свет, и все направились к лифту.

Часть III. Разработка

ГЛАВА 7

Технологии пользовательского уровня

В этой главе

Из этой главы вы узнаете, как создавать эффективный *пользовательский интерфейс* (User Interface, UI) и пользовательские сервисы, Мы расскажем об уже существующих технологиях, влияющих на дизайн пользовательского уровня модели приложения MSF, об их перспективах и о воздействии Web-технологий на современные методы проектирования приложений.

В начале главы обсуждается проектирование пользовательских сервисов и разные типы пользовательского интерфейса, Вы узнаете о влиянии требований к приложению на выбор методов его реализации, о создании и применении приложений со стандартным пользовательским интерфейсом и об особенностях Web-интерфейсов. В заключение мы рассмотрим некоторые конфигурации и методы, которые разработчики могут применять при объединении пользовательского и прикладного сервисных уровней.

При работе над этой главой мы использовали наш собственный опыт проектирования архитектуры приложений и их реализации, а также следующие материалы:

- Мэри Киртлэнд (Mary Kirtland) «Designing Component-Based Applications» (Microsoft Press, 1998);
- «Microsoft Visual Basic 6.0 Programmer's Guide» (Microsoft Press. 1998).

Изучив материал Этой главы, вы сможете:

- ✓ создавать эффективные проекты пользовательского интерфейса;
- ✓ проанализировав требования к пользовательскому интерфейсу, выбрать соответствующую технологическую модель;
- ✓ охарактеризовать технологии реализации пользовательских сервисов;
- ✓ оценить влияние технологий пользовательского уровня на развертывание приложения.

Выбор пользовательского интерфейса

Пользовательский интерфейс (User Interface, UI) — это часть приложения, получающая информацию от пользователя и отображающая ее. Обычно к приложению обращаются люди, хотя иногда это может быть и другое приложение (второй вариант мы рассматривать не будем).

Много раз мы слышали от разработчиков: «Зачем беспокоиться о пользовательском интерфейсе, неужели не ясно, как работать с компьютером? Это же просто, а если кому-то кажется сложным, то пусть не работает с этой программой!» Такие утверждения можно оспаривать бесконечно, но безрезультатно. Хотя тем же программистам известно, что хороший интерфейс значительно повышает эффективность труда пользователей.

Прежде чем выбрать внешний вид приложения, то есть то, что мы, собственно, и называем пользовательским интерфейсом, важно продумать, как он будет представлен пользователю. Информацией пользователей обеспечивает пользовательский сервисный уровень. И хотя он в действительности не содержит UI, для простоты мы будем считать, что пользовательский интерфейс включен в этот уровень.

Пользовательский уровень отвечает за отображение данных, поступающих от прикладных объектов, а также за отображение объектов данных и получение информации от пользователей. Архитектура Microsoft Windows Distributed interNet Applications Architecture (DNA) поддерживает множество архитектур пользовательского уровня, от обычных Win32-приложений до Web-приложений на языке HTML. Во всех этих случаях для доступа к сервисам прикладного уровня используется COM.

Все бизнес-функции приложения инкапсулированы в компоненты прикладного (среднего), а не пользовательского уровня. Такое разделение упрощает повторное использование этих функций в случае

изменения пользовательского уровня. Оба этих уровня взаимодействуют посредством COM. Для реализации интерфейса на пользовательском уровне применяют элементы управления ActiveX на основе COM. Причем одни и те же ActiveX-элементы хорошо работают как в обычных, так и в Web-приложениях для среды Microsoft Windows. Именно это делает возможным повторное применение кода пользовательского уровня в приложениях с разными типами интерфейса.

Пользовательский уровень

При выборе архитектуры пользовательского уровня ответьте на несколько вопросов.

- Ограничивают ли требования к приложению тип пользовательского интерфейса?
- Влияют ли средства защиты (например, брандмауэры) на взаимодействие рабочих станций с серверами?
- Какие операционные системы, установленные на рабочих станциях, надо поддерживать?
- Какие Web-обозреватели надо поддерживать?
- Можно ли на рабочих станциях устанавливать и запускать компоненты COM?
- Доступны ли рабочим станциям удаленные COM-компоненты?

Ответы на эти вопросы помогут вам, как разработчику, выбрать наиболее подходящий для приложения тип пользовательского интерфейса.

Требования к приложению

Тип пользовательского интерфейса зачастую зависит от требований к приложению, что облегчает задачу проектной группе. Кроме того, потребности клиентов определяют и платформу, на которой будет реализован презентационный уровень. Например, если пользователи будут обращаться к приложению по Интернету, Web-архитектура — оптимальный выбор. Разработчики могут написать приложение, взаимодействующее с сервером через Интернет, но при этом пользователи должны иметь доступ к приложению с помощью Web-обозревателя.

В пользовательских интерфейсах корпоративных приложений часто много общего, ведь они зависят от принятых на предприятии стандартов. Например, с целью сокращения затрат многие компании применяют стандартный пользовательский интерфейс для всех внутренних разработок. Другие выбрали Web-интерфейс — пользователям много проще путешествовать по Web-страницам, чем работать с обычными приложениями. Кроме того, стоимость сопровождения таких программ, как правило, ниже стоимости сопровождения обычных приложений для среды Windows.

Средства защиты

Средства защиты позволяют:

- аутентифицировать пользователей;
- контролировать доступ к компонентам приложения и данным;
- шифровать информацию.

Подробно о вопросах защиты мы расскажем в главе 11; здесь же коснемся их кратко, так как брандмауэры и политика зашиты влияют на взаимодействие пользовательских рабочих станций с серверами и, соответственно, на выбор архитектуры приложения. Вопросы защиты касаются в основном Web-приложений, с которыми пользователи работают через Интернет. Однако они имеют отношение и к программам, запускаемым в *глобальных вычислительных сетях* (Wide-Area Network, WAN).

Мы предлагаем четыре метода для разрешения проблем, связанных с брандмауэрами и прокси-серверами.

- **Перенастройка брандмауэра** — брандмауэр можно настроить так, чтобы он пропускал DCOM-трафик и разрешал прокси-серверам «открывать» IP-адреса пользовательских рабочих станций. Однако обычно такие методы не применяются, поскольку разработчики не управляют прокси-серверами, а изменение конфигурации брандмауэра чаше всего запрещено корпоративными стандартами,
- **Туннелирование** — DCOM-трафик можно инкапсулировать в HTTP, который, в свою очередь, использует TCP/IP. При таком туннелировании обычно применяется порт 80, а так как большинство брандмауэров пропускают трафик этого протокола, применение TCP/IP решит проблему. Однако придется настроить и прокси-серверы, разрешив туннелирование для порта 80. Такой способ тоже иногда не годится, так как разработчики не контролируют прокси-серверы.

Примечание Поддержка TCP/IP-туннелирования включена в Microsoft Windows NT 4.0 Service Pack 4 и в Microsoft Windows 2000, Клиентская поддержка реализована в Microsoft Windows 95/98.

- **Использование *сервисовудаленного доступа к данным* (Remote Data Services, RDS)** — RDS обеспечивает маршалинг объектов ADO **Recordset** посредством DCOM или HTTP. Вообще говоря, RDS может выполнять маршалинг клиентских вызовов серверных бизнес-объектов, использующих интерфейс **IDispatch.** При этом применяется COM, что позволяет обходить проблемы брандмауэров и прокси-серверов. Такой метод годится для любых приложений.

(Позже мы обсудим использование RDS для доступа к удаленным бизнес-объектам.)

- **Изменение структуры приложения** — приложения можно проектировать так, что они не будут вызывать компоненты COM через брандмауэр или прокси-сервер. Стандартный путь — Web-приложение, реализованное, как правило, посредством активных серверных страниц (Active Server Pages, ASP), клиентами которого служат Web-обозреватели. ASP-страницы используют сценарии для создания и работы с бизнес-объектами, с помощью которых необходимые Web-страницы создаются и передаются пользователям. В этих страницах разрешены сценарии и компоненты, выполняющиеся на клиенте, но они никогда не обращаются к удаленным COM-объектам, поэтому брандмауэры в данном случае не задействованы (далее в этой главе мы расскажем, как ASP-страницы получают доступ к бизнес-объектам).

Сетевой трафик от серверов может проходить через брандмауэры, разрешающие доступ к серверу только по определенным протоколам. Для доступа к удаленным компьютерам пользователей иногда требуется прокси-сервер. В таких ситуациях сложно использовать DCOM-компоненты для взаимодействия прикладного и пользовательского уровней; чтобы брандмауэр пропускал DCOM-трафик, нужно его переконфигурировать, разрешив трафик этого типа. При доступе через прокси-сервер также возникают трудности, так как большинство из них скрывают IP-адрес пользователя, который нужен для нормальной работы сетевых протоколов, использующих DCOM.

Ограничения, накладываемые клиентскими операционными системами

Клиентские операционные системы следует учитывать на самой ранней стадии проектирования. Если для пользовательского уровня достаточно операционных систем Win32, то можно применять любые технологии, описанные в этой главе.

Одним из потенциальных ограничений является отсутствие поддержки COM на клиентской платформе. В таком случае приложение не сможет использовать клиентские COM-объекты и устанавливать соединение с удаленными COM-объектами. Но даже в этом случае клиентская платформа (например, Microsoft Windows 3.1 или Macintosh) не будет поддерживать DCOM (чтобы использовать DCOM в Windows 95, нужно установить пакет DCOM 95). В таком случае удобнее применить распределенный пользовательский уровень, где для генерации Web-страниц применяются ASP-страницы.

Другое ограничение — COM-компоненты распространяются в виде двоичных файлов в формате определенной платформы. Таким образом, при использовании клиентских COM-объектов необходима

отдельная поставка компонентов для каждого типа клиентской платформы. В противном случае придется создавать специальную версию пользовательского уровня, не обращающуюся к этим компонентам.

И последнее ограничение — сервисы пользовательского интерфейса и просмотра Web на разных платформах могут отличаться. Поэтому для работы с несколькими платформами нужно использовать нейтральный каркас пользовательского интерфейса. В случае Web-приложения следует применять ограниченный набор команд HTML и сценарных языков. Иначе потребуется создание приложений пользовательского уровня для каждой платформы.

Ограничения, накладываемые Web-обозревателями

Возможно, приложению потребуется поддерживать несколько Web-обозревателей даже в одной операционной системе. Поэтому при написании Web-приложений важно как можно раньше определить, какие элементы HTML, языки сценариев, объектные модели, компоненты и т. д. поддерживаются выбранными обозревателями. Разработчику придется либо найти общее для всех обозревателей подмножество функций, либо написать пользовательский уровень отдельно для каждого из них.

Средства определения обозревателя, предоставляемые активными серверными страницами Microsoft Internet Information Server (IIS), позволяют динамически создавать Web-страницы, использующие все возможности конкретного обозревателя. Таким способом разработчику удается выяснить, с каким обозревателем работает клиент, запросивший ASP-страницу, и какие функции поддерживает обозреватель. Для этого применяется компонент Browser Capabilities, который выбирает информацию по имени обозревателя из файла инициализации. В этот файл можно внести информацию обо всех дополнительных функциях конкретного обозревателя.

Разные обозреватели обладают разными возможностями, поэтому выбор поддерживаемых обозревателей поможет вам определить способы реализации функций Web-приложения. Следует изучить различия в объектных моделях разных обозревателей, так как в результате нестыковок будут неправильно работать сценарии. Если вы хотите создать нейтральное (не зависящее от выбора обозревателя) приложение, используйте сценарные языки, совместимые со стандартом European Computer Manufacturers Association (ECMA) — например, Microsoft JScript.

Примечание Сценарии, выполняющиеся на сервере, можно писать на любом языке, поддерживаемом Web-сервером: их выполняет сервер, а не обозреватель.

Важно знать, какие обозреватели популярны у пользователей. Если приложение распространяется по корпоративной интрасети, где регламентировано применение определенного обозревателя — например, Microsoft Internet Explorer 5.0, можно с уверенностью задействовать все его возможности. В противном случае придется учесть в проекте функции других обозревателей. Например, для компании, занимающейся электронной коммерцией, недопустимо отвергать клиента только потому, что у него другой обозреватель. Дабы расширить пользовательскую аудиторию, приходится приспособляться к старым версиям обозревателей и HTML (такой метод называется *добровольной деградацией)*. Пользователям устаревших обозревателей нужно предоставить возможность просмотра текстовой версии узла или, по крайней мере, сообщить, что им следует обновить свой обозреватель, дав ссылку на узел, с которого можно загрузить его последнюю версию.

Примечание Как правило, в США, Канаде, странах Европейского союза и Японии пользуются новыми версиями обозревателей (как минимум, Internet Explorer 4.x или Netscape Navigator 4.x).

Если разработчики ограничиваются HTML 3.2, им будут недоступны некоторые функции манипулирования данными и разметкой, но зато созданные страницы будут работать почти на всех платформах, даже в мобильных операционных системах (например, Microsoft Windows CE). Однако и при этом отображение страниц разными обозревателями отличается или, по крайней мере, не совпадает. Ведь разрешение экрана может варьироваться от 640x480 до 1600x1200 пикселов, что сильно влияет на размер окна обозревателя и соответственно на внешний вид страницы. Отображение страниц зависит и от глубины цвета, меняющейся от 16 до многих миллионов цветов. Так как механизмы визуализации разных обозревателей отличаются, то не совпадает и внешний вид страниц, а устаревшие обозреватели иногда вообще не отображают некоторые элементы. Именно поэтому мы советуем разработчикам тестировать свои приложения при разных разрешениях экрана, на разных платформах и в разных обозревателях — только так можно выявить все ограничения. Если выбранная версия HTML подходит для всех клиентских платформ, то и приложение будет работать на них без проблем. Если же разработчики не знают о разных платформах и версиях обозревателей, применяемых пользователями, или не могут их контролировать, то они не смогут и гарантировать, что страницы будут отображаться правильно.

COM-компоненты на клиенте

При установке и запуске COM-компонентов «родного» приложения на рабочих станциях пользователей проблем, как правило, не возни-

кает — компоненты устанавливаются одновременно с соответствующим Win32-приложением. С Web-приложениями немного сложнее. Если обозреватель поддерживает клиентские COM-компоненты, то они загружаются и автоматически устанавливаются при первом обращении к ним.

Элементы управления ActiveX

Элементы ActiveX — это COM-объекты, загружаемые с Web-сервера и выполняемые на компьютере пользователя с помощью обозревателя. Некоторые пользователи или организации, в которых они работают, считают автоматическую загрузку и установку исполняемых программ на клиентскую рабочую станцию небезопасной, поэтому запрещают эти действия. Если вы создаете приложение именно для таких заказчиков, ограничьтесь COM-компонентами (в том числе и ActiveX-элементами), выполняющимися на клиенте только при условии, что они уже установлены. Некоторые пользователи самостоятельно принимают решение о загрузке и установке каждого компонента. В этом случае можно использовать клиентские COM-компоненты (о том, как компьютеры-клиенты получают к ним доступ, мы расскажем чуть позже).

Распределенные COM-компоненты

Дабы компьютеры пользователей получали доступ к удаленным компонентам, они должны выполнять COM-вызовы с использованием DCOM или RDS. А значит, необходимо, чтобы клиентские обозреватели поддерживали создание и сценарные вызовы COM-объектов. Кроме того, для доступа к удаленным компонентам обычно требуется дополнительное ПО или изменение реестра. Если приложение применяет для доступа к COM-компонентам связывание по виртуальной таблице или раннее связывание, на компьютеры пользователей придется установить специальные библиотеки-заместители/представители или библиотеку типов (методы связывания и COM-компоненты подробно рассмотрены в главе 8).

Большинство приложений для определения имени удаленного сервера используют информацию из реестра, поэтому при работе с удаленными компонентами иногда возникают проблемы с защитой. Методы доступа к удаленным компонентам описаны далее в этой главе.

Примечание При создании Web-приложения обязательно убедитесь, что все необходимые функции доступны через интерфейс **IDispatch** — большинство языков сценариев поддерживают позднее связывание только с его помощью. Если же необходимы функции другого интерфейса, придется пересматривать архитектуру приложения.

Соединения с Интернетом и интрасетью

Если в качестве платформы выбран HTML, пользовательский уровень придется кодировать для нескольких обозревателей. Так как доступ к этому уровню осуществляется с помощью TCP/IP, на решение разработчиков влияют:

- типы пользовательских соединений;
- необходимое число соединений;
- различие скоростей пользовательских соединений.

Распространенные типы соединений таковы:

- аналоговые модемы с пропускной способностью до 53 кбит/с;
- ISDN-модемы, до 112 кбит/с;
- кабельные модемы, до 10 Мбит/с;
- DSL-модемы, до 6 Мбит/с;
- Tt и другие постоянные соединения, от 1,5 Мбит/с;
- спутниковые тарелки, до 10 Мбит/с;
- корпоративные сети Ethernet, обычно 10 или 100 Мбит/с.

Выбор архитектуры пользовательского уровня

Выбор интерфейса для сервисов пользовательского уровня иногда представляет собой трудную задачу. Этот раздел поможет вам решить, какой интерфейс — обычный или Web — выбрать; часто приходится реализовать оба варианта.

Пользовательский **уровень «родных»** приложений

Приложения, клиентская часть которых обращается к средствам операционной системы, называют *«родными»* для этой ОС. Для выполнения своих функций эти приложения используют прикладные интерфейсы операционной системы. При написании 32-разрядных Windows-приложений для таких операционных систем, как Windows 95, Windows 98, Windows NT и Windows 2000. применяется API Win32. Таким образом, если проектная группа выбрала конкретную клиентскую платформу, можно создать «родное» для нее приложение.

Среда Win32 предоставляет разработчикам большие возможности для создания пользовательского интерфейса. Специализированные Win32-приложения доступны огромной аудитории. Операционные системы располагают мощными средствами поддержки дополнительных технологий, которые можно применять в приложениях:

- технологии поддержки мультимедиа DirectX;
- технологии доступа к данным Open Database Connectivity (ODBC) и OLE DB.

Для разработки «родных» приложений можно применять различные языки программирования и среды разработки, в том числе Microsoft Visual C++, Microsoft Visual Basic и Microsoft Visual J++. Когда

приложение готово, его компилируют, собирают и устанавливают на клиентские системы. Приложения на этих языках являются истинно компилируемыми, поэтому для создания пользовательского интерфейса, состоящего из сложных элементов, многие из которых графические, наилучший выбор — «родное» приложение.

Поскольку пользовательские интерфейсы приложения требуют применения определенных функций операционной системы, их устанавливают на локальных компьютерах. При этом обязательно учитывают, что приложение может оперировать как большими объемами локальных данных, так и удаленными данными, передаваемыми по низкоскоростным каналам. Например, программы семейства Microsoft Office могут содержать исполняемые файлы и DLL объемом 120 Мб. Однако файлы данных и серверные приложения обмениваются лишь небольшим объемом информации, таким образом сокращая сетевой трафик. Большинство современных обычных приложений нужно устанавливать и настраивать. Поэтому, если планируется постоянная корректировка приложения, лучше выбрать другую архитектуру.

При применении «родных» приложений проще встраивать новые технологии. Например, в Office 2000 очень легко добавить средства XML, причем вам не придется создавать отдельные библиотеки для каждого приложения, входящего в состав этого пакета. Если вы хотите использовать возможности других приложений, установленных на компьютерах-клиентах, лучше выбрать обычное приложение, которое способно взаимодействовать с другими программами.

Пользовательский уровень с Web-интерфейсом

Web-интерфейс предлагает почти универсальные методы распространения и готовые средства отображения. Свободно распространяемые Web-обозреватели обладают простыми интерфейсами, увеличивающими эффективность развертывания приложений. Установка и последующие обновления приложения — самые сложные этапы работы. Web-интерфейс позволяет значительно сократить затраты времени на эти процедуры, так как приложения распространяются среди пользователей не вручную. Поэтому часто предпочтение отдается именно Web-интерфейсу — особенно когда большое значение имеют распространение, развертывание и постоянное сопровождение.

Важно помнить, что Web-интерфейсы изначально предназначались для отображения информации на экране. В последнее же время, благодаря появлению языков сценариев, к их возможностям добавилось и выполнение программ. Перемещение по документам осуществляется посредством гиперссылок, которые способны вызвать выполнение программы на данной странице или в другом файле, как, например, при использовании динамического языка HTML (DHTML).

С добавлением DHTML и сценариев, выполняющихся на клиенте, набор элементов управления, доступных Web-приложениям, значительно расширился. Тем не менее, если приложение предназначено для интенсивных вычислений или отображения большого объема информации в графическом виде, Web-интерфейс — не лучшее решение.

Возможности большинства современных интрасетей позволяют при разработке применять HTML 4.0, DHTML и *иерархические таблицы стилей* (Cascading Style Sheets, CSS). Необходимый набор средств определяется на фазе «Анализ». Все функции приложения тестируются в нескольких обозревателях с использованием прототипов, созданных на фазе «Планирование». На фазах «Планирование» и «Разработка» следует изучить различные типы соединения и определить, как скорость загрузки графики, апплетов, компонентов и других элементов страниц влияет на производительность приложения.

Комбинированный пользовательский уровень

Не следует рассматривать «родной» и Web-интерфейсы как взаимоисключающие варианты. Ведь, как и описанное в нашем практикуме приложение, любая программа может обладать двумя интерфейсами. Иногда они похожи, или же один из них отличается какими-то дополнительными функциями. Обратите внимание, что даже выбор двух интерфейсов позволяет повторно использовать значительную часть пользовательских сервисов. Это одно из достоинств пользовательского уровня, созданного для взаимодействия пользовательского интерфейса и бизнес-объектов,

Основы проектирования интерфейса

Этот раздел целиком позаимствован из книги «Microsoft Visual Basic 6.0 Programmer's Guide» (Microsoft Press, 1998).

Чтобы построить прекрасный пользовательский интерфейс, не обязательно быть художником. Большинство принципов проектирования пользовательского интерфейса совпадают с правилами создания художественного произведения. Ведь такие понятия, как композиция, цвет и т. д., одинаковы и для экрана компьютера, и для листа бумаги, и для холста.

Современные средства программирования позволяют создавать пользовательский интерфейс, просто перетаскивая элементы управления на форму или Web-страницу. Однако советуем вам не отказываться от предварительного планирования. Сначала лучше прорисуйте пользовательский интерфейс на бумаге, определите необходимые элементы, их относительную важность и взаимосвязи.

Принципы проектирования одинаково подходят как для обычных интерфейсов, так и для интерфейсов на основе Web. Программный

код, реализующий эти принципы для разных типов приложений, иногда сильно отличается, однако последовательность проектирования пользовательского интерфейса остается неизменной.

Элементы пользовательского интерфейса

Многие реализации пользовательского интерфейса имеют общие элементы. Их правильное применение повысит эффективность приложения. Хотя внешний вид этих элементов в обычном и Web-интерфейсах может быть разным, функционируют они одинаково.

Стили интерфейса

В мире Windows-приложений не все пользовательские интерфейсы выглядят и ведут себя одинаково. Существует три основных стиля и один дополнительный.

- **Однодокументный интерфейс (Single-Document Interface, SDI)** — один из примеров интерфейса такого типа — приложение WordPad из состава Windows. В этой программе можно открыть только один документ. Его надо закрыть прежде, чем вы сможете открыть новый.
- **Многодокументный интерфейс (Multiple-Document Interface, MDI)** - таким интерфейсом обладают, например, Microsoft Word 2000 и Microsoft Excel 2000. В них разрешается одновременно открывать несколько документов, каждый в своем окне. MDI-приложения характеризуются наличием в меню команды Window для переключения между окнами.
- **Интерфейс в стиле Explorer** — это окно, состоящее из двух панелей, в одной из которых отображается дерево (слева), а другая представляет собой область отображения текущего элемента дерева (справа). Таков интерфейс Проводника (приложения Explorer) из состава Microsoft Windows.
- **Отчет** — этот стиль обычно считается дополнительным. Информация в таком интерфейсе отображается в любом формате — графическом, табличном, текстовом или комбинированном, Многие приложения позволяют отображать отчеты или печатать их на принтере.

При выборе интерфейса разработчики должны принять во внимание как назначение приложения, так и работу пользователей. Например, для простой программы «часы» лучше всего выбрать стиль SDI, так как маловероятно, что кому-то потребуется сразу несколько часов одновременно (в крайнем случае, можно запустить второй экземпляр приложения). Для приложения, обрабатывающего страховки, лучше применить MDI-интерфейс, так как возможна работа сразу с несколькими документами — например, для их сравнения. Интерфейс в стиле Explorer используется во многих новых приложениях, так как позволяет искать и просматривать множество документов, изображений или файлов. Отчеты представляют собой «снимок» состояния информации в определен-

ный момент времени и обычно не предполагают непосредственного взаимодействия пользователя с отображаемой информацией.

Диалоговые окна

Большинство приложений взаимодействуют с пользователем. Для запроса данных, необходимых для работы программы, в Windows-приложениях служат диалоговые окна. Это форма специального типа, которая отображает информацию и, как правило, требует в ответ каких-либо действий со стороны пользователя. Обычно, чтобы продолжить работу с приложением, диалоговое окно нужно закрыть.

Запросы приложения варьируются от простого выбора «да/нет» до выбора из списка с сотнями элементов. Для реализации этих действий служат элементы управления, перечисленные в табл. 7.1.

Табл. 7.1. Элементы управления и их назначение

Элемент управления	Описание
Метка (label)	Только отображает текст
Текстовое поле (text box)	Текст, который пользователь может редактировать. Обычно применяется для ввода информации, например, пароля пользователя
Кнопка (command button)	Кнопка (обычно прямоугольной формы), выполняющая команду пользователя по ее щелчку
Флажок (check box)	Набор вариантов, из которых можно выбрать один или несколько. Показывает состояние какого-либо условия — включено/выключено, правда/ложь, да/нет. Если флажки сгруппированы, они независимы друг от друга — пользователь может выбрать любое число вариантов
Переключатель (option button)	Набор вариантов, из которых можно указать только один. Переключатели работают только в группе; выбор одного из них автоматически отключает остальные
Список (list box)	Прокручиваемый список вариантов. Обычно он отображается вертикально в одну колонку, но можно применять списки и из нескольких колонок. Если число элементов списка превышает число отображаемых элементов, должны быть предусмотрены полосы прокрутки. Из списка можно выбрать несколько элементов, удерживая клавишу Ctrl, Кроме того, список бывает раскрывающимся

(продолжение)

Элемент управления	Описание
Поле со списком (combo box)	Прокручиваемый список, объединенный с текстовым полем. Похож на обычный список, но пользователю разрешается как вводить информацию в текстовое поле, так и выбирать элемент из списка
Ползунок (slider control)	Позволяет указать значение на шкале. Такими элементами управления обычно задают громкость звука или регулируют цвета изображения
Индикатор (progress indicator)	Показывает, какая часть процесса выполнена к настоящему моменту. Появление на экране этого элемента управления говорит о том, что приложение выполняет какую-то работу. Если для этого требуется много времени, следует предусмотреть вывод времени, необходимого для завершения работы

Если пользователю предоставляется право выбора, желательно, чтобы существовал вариант по умолчанию.

Композиция

Композиция или размещение элементов пользовательского интерфейса определяет не только его эстетику, но и удобство применения. Композидия подразумевает:

- размещение элементов управления;
- согласованность элементов управления;
- понятность элементов;
- использование разделителей;
- простоту дизайна.

Размещение элементов управления

В большинстве интерфейсов не все элементы равнозначны. Интерфейс нужно проектировать так, чтобы пользователь сразу же понимал, какие элементы важнее других. Поэтому размещение должно подчеркивать иерархию: важные элементы следует помешать на видное место, а маловажные или редко используемые — на менее заметное.

В большинстве языков мира текст читается слева направо и сверху вниз. Это верно не только для книг, но и для экрана компьютера. Поэтому большинство пользователей первым делом обращают внимание на верхний левый угол экрана, следовательно, именно там стоит расположить самые важные элементы интерфейса. Например, рассмотрим форму, содержащую информацию о клиенте. Поле с его именем нужно поместить там, где оно сразу бросится в глаза. Кнопку

OK следует разместить в правом нижнем углу экрана, так как она потребуется только в конце работы с формой.

Важно также объединение элементов управления в группы. Группировать элементы следует в соответствии с назначением. Например, кнопки перемещения по базе данных надо объединить, а не разбрасывать по всей форме, так как их функции взаимосвязаны. То же касается и сведений о клиенте — например, поля с именем и адресом обычно объединяют в одну группу. Элементы одной группы, как правило, ограничивают рамкой.

Часто пользователи переходят от одного элемента управления к другому с помощью клавиши Tab, поэтому в таких формах ввода, как диалоговые панели, важен осмысленный порядок обхода.

Согласованность элементов пользовательского интерфейса

Согласованность элементов — важнейшее качество интерфейса, гарантирующее гармонию. Недостаток согласованности и нелогичность интерфейса вносят в приложение беспорядок, лишают его организованности, что порождает сомнения в его надежности.

Для достижения согласованности следует на ранних фазах проектирования выработать стратегию и соглашения о стиле. Типы элементов управления, их размеры, способы группирования и шрифты надо определять заранее. Для решения этих вопросов можно использовать пробную версию системы или ее прототипы (см. главы 5 и 6). Итак, какие же вопросы следует согласовать?

- Тип элемента управления — не стоит использовать все доступные элементы управления, хотя, конечно, трудно удержаться от соблазна. Выбирайте только те. которые наилучшим образом подходят для приложения. Несмотря на то, что для представления списочной информации годится и список, и поле со списком, и табличная форма, и представление в виде дерева, лучше выбрать и везде применять только один из этих элементов.
- Свойства — важно согласовать свойства элементов управления. Например, если в одном месте у текстового поля белый фон, то в другом месте не стоит без веских причин использовать серый.
- Тип формы — для удобства работы с приложением важно согласовать формы. Две формы — одна с серым фоном и с трехмерными эффектами, а другая с белым фоном и плоскими элементами – внесут сумятицу и несогласованность.

Понятность элементов

Назначение *интуитивно понятных элементов* можно определить по их внешнему виду. Такие элементы сопровождают нас во всех сферах жизни — например, одного взгляда на рукоятки велосипедного руля с выемками для пальцев хватит, чтобы понять, в каком месте держать руль.

Интуитивно понятные элементы стоит применять и в пользовательском интерфейсе. Например, кнопки с трехмерными эффектами так и хочется щелкнуть. Если бы они выглядели плоскими, было бы труднее понять, что это кнопки. Однако группа разработчиков по каким-то соображениям может спроектировать в определенных приложениях, например, мультимедиа-программах, плоские кнопки. Но тогда их надо применить *во* всем приложении.

Другой пример интуитивно понятного элемента — текстовое поле, Пользователи ожидают, что белое поле, окруженное рамкой, предназначено для редактирования текста. Однако можно отобразить его и без рамки, так что оно будет выглядеть, как нередактируемая метка,

Использование разделителей

Разделители подчеркивают важность некоторых элементов, в результате чего работа с приложением становится более удобной. Обычно разделителями служит пустое пространство вокруг или между данными формы. Элементы управления и данные загромождают интерфейс, и становится трудно найти нужные поля и информацию. Разделители же позволяют подчеркнуть важность элементов формы,

Заранее определенные расстояния между элементами и их выравнивание по вертикали и горизонтали делают интерфейс удобнее. Текст в журналах верстается в равноотстоящих друг от друга колонках, что облегчает его чтение; точно так же упорядоченный интерфейс лучше воспринимается, и с ним проще работать.

Простота дизайна

Вероятно, самое важное в дизайне интерфейса — его простота. Если интерфейс приложения сложен, то, скорее всего, и самим приложением трудно пользоваться. И с эстетической точки зрения ясный и простой дизайн всегда предпочтительнее,

Создание интерфейса на основе существующих методов работы — обычная ошибка многих разработчиков. Например, фирма создает приложение для заполнения страховых документов. Естественно желание создать интерфейс, полностью копирующий бумажные документы. В этом случае возникает несколько проблем:

- форма и размер бумажного документа отличаются от его аналога на экране;
- копирование документа ограничивает возможности разработчиков текстовыми полями и флажками;
- у приложения нет никаких преимуществ по сравнению с бумажными документами.

Гораздо лучше создать новый пользовательский интерфейс, оставив при этом возможность печатать документы, идентичные бумаж-

ным. Если применить группировку полей, разбить исходный документ по вкладкам или связанным формам, всю информацию можно будет вводить, не применяя полос прокрутки. Кроме того:

- пользователю придется вводить меньше информации, если спроектированы списки с заранее загруженными вариантами ответа;
- редко применяемые функции можно вынести в отдельную форму;
- пользователю не придется обращаться к каждому элементу интерфейса, если для них будет задано значение по умолчанию (конечно же, обязателен механизм изменения этого значения);
- сложные или редко выполняемые задачи можно упростить с помощью мастера,

Работа с приложением — лучший тест на простоту его интерфейса. Если пользователь не может выполнить какую-либо задачу самостоятельно, вероятно, придется переделать интерфейс.

Цвет и изображения

Использование цвета делает интерфейс значительно более привлекательным, но важно не переусердствовать. Так как большинство современных мониторов способны отображать миллионы цветов, возникает соблазн использовать их как можно больше. Тут-то и появляются проблемы, особенно если все вопросы, касающиеся цвета, не решены на начальных стадиях проектирования интерфейса.

Цветовые предпочтения у всех разные, вкусы пользователей и разработчиков далеко не всегда совпадают. Цвета могут вызывать сильные эмоции, а значения отдельных цветов часто несут конкретную смысловую нагрузку. Поэтому лучше остаться консерватором и применить приятные, нейтральные тона.

Выбор цвета зависит от предполагаемых пользователей приложения и от настроения, которое хотят передать дизайнеры. Ярко-красные, зеленые и желтые цвета годятся для детских программ, но банковские служащие вряд ли оценят их по достоинству.

Яркие цвета в небольших количествах позволяют привлечь внимание к важным элементам интерфейса. Однако их число следует ограничить, а цветовая схема должна соответствовать стилю приложения. По возможности применяйте 16-цветную палитру, так как цвета не включенные в нее, не отображаются на 16-цветных мониторах.

Еще одна проблема — цветовая слепота пользователей. Многие люди не различают комбинации некоторых основных цветов — например, красного и зеленого. Поэтому они не увидят красный текст на зеленом фоне.

Изображения и значки

Картинки и значки добавят привлекательности приложению, но обращаться с ними нужно осторожно. Изображения могут передать информацию вместо текста, но люди относятся к ним по-разному.

Панели инструментов со значками, изображающими различные функции, очень полезны, но если пользователи не смогут сразу же понять, что на них нарисовано, работа только замедлится. При выборе значков для панелей инструментов стоит придерживаться уже действующих стандартов. Например, во многих приложениях значок New File (Создать файл) — это лист бумаги с загнутым уголком. Хотя можно придумать и лучшую метафору для этой функции, другое изображение только запутает пользователя.

Важно учитывать и национальные традиции. Во многих программах для обозначения почтовых функций применяется картинка, изображающая почтовый ящик с флажком, стоящий у многих частных домов в США. Но это ясно лишь американцам; пользователям из других стран этот символ может быть непонятен.

При создании значков и изображений лучшее правило — простота. Сложные многоцветные картинки, как правило, плохо отображаются на значке панели инструментов размером 16x16 пикселов.

Шрифты

Так как для передачи информации в основном применяется текст, выбор легко читаемого при разных разрешениях и на разных мониторах шрифта — важная часть проектирования пользовательского интерфейса. Лучше всего задействовать простые шрифты везде, где возможно. Декоративные шрифты, например, Script, хорошо выглядят на бумаге, но не на экране. К тому же текст, написанный таким шрифтом малого размера, трудно читать.

Обычно в интерфейсе используют стандартные шрифты Windows: Arial, Times New Roman или System. В противном случае вместе с приложением придется распространять и применяемый шрифт, так как на компьютерах, где он не установлен, система заменит его на другой, что иногда сильно ухудшает внешний вид приложения.

Примечание При проектировании приложения для пользователей разных стран важно исследовать шрифты, популярные в этих странах. Также следует изучить ширину текста, так как при переводе на другой язык текст может увеличиться, иногда даже на 50%.

Повторим, что при выборе шрифта, как и многих других элементов интерфейса, важна согласованность. Общая рекомендация — не использовать более двух шрифтов двух или трех размеров, иначе ваше

приложение будет похоже на требование выкупа, составленное из букв, вырезанных из разных газет.

Удобство использования

Удобство приложения определяют пользователи. Проектирование интерфейса — итерационный процесс. Создать идеальный интерфейс с первой попытки очень сложно. Привлечение пользователей к проектированию на ранних стадиях поможет разработать отличный, удобный интерфейс с меньшими усилиями.

Примеры для подражания

Начиная проектировать пользовательский интерфейс, изучите другие приложения, особенно те из них, что хорошо продаются. Чтобы сделать их удобными, затрачено много ресурсов, проведено множество исследований. У интерфейсов таких приложений много общего - панели инструментов, всплывающие подсказки, строки состояния, контекстно-зависимые меню и диалоговые панели с вкладками.

При проектировании стоит учесть и собственный опыт разработчиков как пользователей программного обеспечения. Однако предпочтения небольшой группы разработчиков могут не совпадать со вкусами широкого круга пользователей, поэтому все свои идеи надо проверять на прототипах.

Кроме того, большинство успешных приложений предоставляет пользователю возможность выбрать способ выполнения того или иного действия в зависимости от его предпочтений. Например, скопировать файл в программе Explorer можно средствами меню и клавиатурных команд или перетащив файл мышью. Разнообразие вариантов только добавляет привлекательности приложению; поэтому стоит предусмотреть возможность выполнения всех функций как с помощью клавиатуры, так и мыши.

Правила организации пользовательского интерфейса Windows-приложений

Одно из главных достоинств операционных систем Windows состоит в том, что у всех приложений общий интерфейс. Пользователям, научившимся работать с одним из таких приложений, не составит труда изучить другое, но гораздо сложнее будет привыкнуть к приложению, интерфейс которого им непривычен.

Хороший пример общих элементов интерфейса — меню. В большинстве Windows-приложений меню File (Файл) слева, а меню Help (Справка) — справа, между ними находятся другие меню, например, Edit (Правка) и Tools (Сервис). Возможно, кто-нибудь скажет, что название Documents (Документы) лучше, чем File (Файл), или что меню Help (Справка) нужно поместить первым. Ничто не помешает

разработчикам отступить от стандартов, но это наверняка вызовет замешательство у пользователей и снизит популярность приложения.

Не менее важно и размещение команд. Например, пользователи привыкли, что команды Copy (Копировать), Cut (Вырезать) и Paste (Вставить) находятся в меню Edit (Правка), поэтому они будут недовольны, если вы поместите эти команды в меню File (Файл). Лучше не отступать от устоявшихся стандартов без особой на то причины.

Доступность функций

Один из важнейших параметров, который проверяется при тестировании удобства использования программы. — *доступность* различных функций. Если пользователь не может понять, как применять функцию (или не может ее найти), эта функция бесполезна. Например, как выяснилось, многие из тех, кто работал с Windows 3.1, не знали, что комбинацией клавиш Alt+Tab можно переключаться между открытыми приложениями. Ничто в интерфейсе не помогло им узнать об этом.

Чтобы проверить доступность функции, попросите пользователей выполнить определенную задачу, не объясняя, как это сделать. Если им не удастся справиться с первой попытки, нал доступностью этой функции придется еще поработать.

Средства навигации

Все пользователи знают, как перемещаться по приложению средствами меню. Очень важно правильно использовать эту систему, как, собственно, и другие элементы интерфейса. Чтобы работать с программой было удобно, не следует отступать от стандартов Windows. Сейчас, благодаря обширной поддержке обозревателями языков сценариев, даже в Web-программах удается сохранить стиль меню «родных» приложений, которые, в свою очередь, могут позаимствовать у Web метафоры ссылок и страниц, Повторим еще раз, система навигации приложения должна быть простой и логичной.

Модель помощи пользователям

Как бы ни был хорош интерфейс, возникают ситуации, когда пользователю не обойтись без помощи. Поэтому модель помощи пользователям должна включать в себя как встроенную справочную систему, так и печатную документацию. Кроме того, можно добавить всплывающие подсказки, статусные строки, подсказки «Что это такое?» и мастера.

Модель помощи пользователям, как и другие части приложения, надо проектировать на ранних стадиях процесса разработки. Включенные в нее функции зависят от сложности приложения и от опыта пользователей.

Создание пользовательского интерфейса

Определив вид и тип интерфейса, разработчики приступают к его реализации. Тестируют его с помощью прототипов — их распространяют среди пользователей (см. главу 5). Облегчает написание программы *интегрированная среда разработки* (Integrated Development Environment, IDE). В такой среде есть мастеры, шаблоны и инструменты рисования, которые ускорят создание пользовательского интерфейса, взаимодействующего с другими частями пользовательского уровня.

Реализация пользовательского уровня «родных» приложений

Создание обычных приложений зависит от выбранного языка программирования. Среды разработки Microsoft позволяют создавать интерфейсы на языках Visual Basic, Visual C++ или Visual J++. Такие интерфейсы основаны на элементах управления, различных библиотеках и API операционной системы, входящих в состав библиотек языка программирования. Так как в большинстве многоуровневых приложений реализуется Web-интерфейс, мы не будем подробно описывать создание «родных» приложений. Если вам этот вопрос интересен, советуем прочитать следующие книги: «Microsoft Visual Basic 6.0 Programmer's Guide» (Microsoft Press, 1998), «Microsoft Visual C++ 6.0 Programmer's Guide» (Microsoft Press, 1998), «Desktop Applications for Microsoft Visual Basic 6.0» (Microsoft Press, 1999), «Desktop Applications for Microsoft Visual C++ 6.0» (Microsoft Press. 1999), «Visual Basic 6.0 Win32 API Tutorial» (Wrox Press, 1998) и «Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques» (John Wiley and Sons, 1996).

Реализация Web-интерфейса

Как и все приложения масштаба предприятия, Web-приложения должны динамически отображать информацию из одного или нескольких источников. Для создания таких программ можно использовать ASP-компонент из комплекта IIS. В IIS версии 4.0 технология ASP интегрирована с MTS.

Внимание! Типичная ошибка начинающих ASP-программистов — избыточная реализация функций прикладного уровня на основе сценариев ASP. Чтобы сохранить масштабируемость приложения и упростить кодирование сценариев, бизнес-функции следует реализовать в виде COM-компонентов.

Знакомство с ASP

В интернет-приложениях Web-обозреватель реализует пользовательский уровень на основе HTML-страниц. Запросы этого уровня передаются Web-серверу по протоколу HTTP. В ответ на запрос клиентского Web-обозревателя на сервере активируются ASP-страницы, которые способны сгенерировать и вернуть обозревателю необходимую HTML-страницу, ASP можно применять и для создания интерфейса. Следовательно, ASP считается частью пользовательского уровня. Однако в ASP-страницах должны содержаться сценарии, выполняющиеся на сервере и использующие бизнес-объекты среднего уровня. Эти бизнес-объекты способны, в свою очередь, вызывать объекты данных, таким образом получая доступ к уровню данных. С другой стороны, HTML и клиентские сценарии, реализующие пользовательский уровень, реализуются в ASP-страницах. В любом случае ASP необходим для взаимодействия между пользовательским и прикладным уровнями.

Преимущества ASP

ASP обладает достоинствами, позволяющими избежать некоторых проблем при разработке высококачественных и высокопроизводительных Web-сайтов.

Привычная модель программирования

Основное достоинство ASP — привычная модель программирования, похожая на HTML, но дополненная средствами создания сценариев. Сценарии, выполняющиеся на сервере, разрешается писать на любом языке, если на сервере установлен выполняющий его модуль. IIS поддерживает языки Visual Basic Script Edition и Microsoft JScript, соответствующие стандарту ECMA.

Сложные алгоритмы, как и код, вызывающий несколько COM-объектов, должны реализовываться COM-компонентами, а не серверными ASP-сценариями. Сценарии ASP интерпретируются во время выполнения, поэтому чем они меньше, тем выше их производительность. Фактически ASP-код для доступа к объектам автоматизации может использовать только интерфейс **IDispatch.** Но бизнес-объекты компилируются и способны воспользоваться преимуществами связывания по виртуальной таблице, что увеличивает производительность по сравнению с ASP-кодом, применяющим **IDispatch.** Кроме того, бизнес-объекты можно повторно задействовать в других приложениях — например, в Win32-версии клиента.

Защита

Активные страницы выполняются на Web-сервере и полностью контролируют, какие данные будут переданы Web-клиенту. Таким образом защищается ваша интеллектуальная собственность, будь то информация или код, — на клиентских компьютерах нельзя отследить

ни местонахождение данных, ни происхождение HTML-страниц. Такой механизм обеспечивает уровень защиты, приемлемый для многих приложений,

Различия обозревателей и дисплеев

ASP устраняет проблемы, связанные с различиями обозревателей. Например, разные обозреватели поддерживают разные версии HTML, в том числе и DHTML, а некоторые из них используют собственные расширения HTML и DHTML. Некоторые обозреватели поддерживают Java-апплеты, другие — элементы управления ActiveX, а третьи вообще не поддерживают клиентские компоненты. Какие-то обозреватели поддерживают клиентские сценарии на языках JavaScript или Visual Basic Scripting Edition, другие же не поддерживают никаких сценариев.

Кроме того, различия в разрешении экрана и глубине цвета у разных пользователей дополнительно усложняют разработку независимых Web-страниц.

Динамическое содержание

Средствами ASP можно создавать как динамическое, так и статическое наполнение. Первое основано на данных из серверных источников, к которым клиентский компьютер не должен обращаться напрямую. Это связано с низкой масштабируемостью клиентских соединений. Что еще важнее, при обращении к корпоративным базам данных пользователей Интернета появляются проблемы, связанные с безопасностью информации. По этой причине создатели интернет-приложений позаботились, чтобы HTML-страницы генерировала защищенная серверная программа. Если Web-клиенту понадобятся дополнительные данные, то она передаст их ему после получения сервером соответствующего запроса.

ASP облегчает генерацию динамического наполнения, разрешая выполнение сценариев как на клиенте, так и на сервере. При доступе к ASP-странице запускаются серверные сценарии. По их окончании генерируется результат, который и передается клиенту. Серверные сценарии либо управляют тэгами HTML (в файлах .asp), которые включаются в такие ответы, либо генерируют HTML-страницы «на лету». ASP определяет возможности обозревателя и отображает информацию в соответствии с ними.

С появлением Java-апплетов, элементов ActiveX, DHTML и ASP стало возможно создавать клиентские Web-приложения, поддерживающие постоянное соединение с сервером. Вместо того чтобы генерировать новую страницу каждый раз, когда клиенту нужны новые данные, информация передается между клиентскими и серверными сценариями. Внутренние источники информации при этом закрыты

для пользователей Интернета, так как к ним обращаются только серверные приложения.

Компоненты автоматизации

В ASP включено несколько компонентов автоматизации, выполняющих стандартные действия, например, определение характеристик обозревателя, разбор параметров, обмен файлами жетонов «cookie» и обмен данными между страницами. Так как ASP поддерживает компоненты автоматизации, совместимые с серверными сценариями, относительно простым делом стало создание страниц, обрабатывающих формы HTML. Кроме того, для сложных операций можно создать отдельные компоненты и использовать их в серверных приложениях.

ASP и MTS

ASP — мощный инструмент, позволяющий устранить разрыв между клиентскими функциями представления и серверными бизнес-функциями в трехуровневых приложениях. В IIS 4.0 технология ASP интегрирована с сервером транзакций Microsoft Transaction Server (MTS). MTS предлагает IIS- и ASP-приложениям сервисы, управляющие процессами, потоками и ресурсами, и позволяет выполнять страницы ASP-приложений как транзакции, со всеми их преимуществами.

Серверный сценарий ASP-страницы считается первым уровнем бизнес-функций. Он способен использовать для выполнения запроса COM-объекты. Чтобы убедиться, что после одновременной работы нескольких объектов ресурсы обновлены корректно, стоит применять транзакции, в процессе которых может быть обработана любая ASP-страница IIS 4.О. После этого будет инициирована новая транзакция. Все объекты, созданные на странице, автоматически регистрируются MTS. Транзакция завершается после интерпретации всего файла и выполнения всех серверных сценариев. Если она прерывается, то всем ресурсам возвращаются значения, которые они имели до выполнения страницы. Такая отмена транзакции особенно полезна для страниц, обновляющих базы данных.

IIS использует MTS для координации процессов серверных приложений. Каждое приложение, размещенное в отдельном виртуальном каталоге, можно запустить либо в рамках процесса IIS, либо в процессе MTS, причем последнее повышает надежность сервера (в случае сбоя одного приложения оно будет закрыто и перезапущено, причем перезапуск приложения не повлияет на другие программы и, что еще важнее, на весь сервер).

Распределение приложений по отдельным процессам также упрощает поддержку сервера, В этом случае при обновлении приложения

придется закрыть только один процесс, что не отразится на работе сервера и других программ.

MTS управляет всеми процессами приложений, однако средства IIS позволяют создать отдельные IIS-приложения. Функции управления такими программами инкапсулированы в COM-компоненте Web Application Manager (WAM) и выполняются до передачи управления MTS. WAM загружает библиотеки ASP и взаимодействует с ними при получении HTTP-запросов. При этом каждое приложение, запущенное в процессе IIS или в каком-либо другом процессе, содержит экземпляр WAM. Если конфигурация IIS позволяет запускать приложения раздельно, разработчики могут вызывать административные объекты MTS с целью:

- создать новый MTS-пакет;
- сгенерировать новый *глобальноуникальный идентификатор* (GUID);
- добавитьWAM к новому пакету;
- настроить пакет для запуска в отдельномпроцессе.

Приложения, выполняющиеся в процессе IIS, регистрируются в стандартном внутреннем пакете MTS. При получении HTTP-запроса IIS проверяет, соответствует ли запрашиваемый *универсальныйуказатель ресурса* (UniformResource Locator, URL) IIS-приложению. В случае положительного ответа на этот вопрос IIS ищет во внутренней WAM-карте соответствующий WAM-объект. Если приложение не запущено и экземпляр WAM не существует, его нужно создать. С помощью идентификатора приложения, который генерируется при его регистрации, IIS создает новый WAM-объект, после чего MTS запускает новый процесс и берет на себя управление объектами и ресурсами приложения. Если запрашиваемый URL не соответствует приложению, он обрабатывается обычным образом.

Особенности ASP-приложений

Как мы уже говорили, ASP-страницы используются для того, чтобы вызывать серверные бизнес-объекты и передавать Web-страницы клиенту. ASP-страницы состоят из текста, сценариев и директив. Сценарии и директивы отделяются символами «<%» и «%>». Для обозначения серверных сценариев применяется тэг <SCR1PT>. Все остальные элементы отображаются на возвращаемой клиенту странице «как есть».

ASP-страницы выполняются после получения запроса от клиента. Объекты, созданные сценариями, существуют только во время обработки страницы, если они не были явно сохранены в переменных Session или Application, Прежде чем сохранять данные в этих переменных, изучите их влияние на масштабируемость приложения.

В рамках страницы разрешается создавать любые объекты, необходимые для выполнения работы. Однако повторим, что большинство бизнес-функций следует инкапсулировать в компонентах, не применяя для этого сценарии. Серверные сценарии должны вызывать бизнес-объекты и создавать Web-страницы, возвращаемые клиентскому компьютеру.

Границы транзакций для всех объектов, созданных страницей, определяются в процессе ее обработки. Если какой-либо объект прерывает транзакцию, вместо запрашиваемой страницы должно быть передано сообщение об ошибке. Чтобы включить страницу в транзакцию, начните ее с директивы TRANSACTION:

```
<%@LANGUAGE="VBSCRIPT" TRANSACTION=REQUIRED%>
```

Для передачи клиенту информации, зависящей от того, выполнена транзакция или прервана, используются события **OnTransactionCommit и OnTransactionAbort**.

Для создания объекта в серверном сценарии используется метод **CreateObject.** При этом объект будет создан с соответствующим контекстом транзакции и возможностью доступа к внутренним ASP-объектам **Response и Request.** Чтобы прервать транзакцию, достаточно вызвать из серверного сценария метод **ObjectContext** SetAbort.

Использование клиентских компонентов

ActiveX-элементы — это небольшие COM-компоненты, передаваемые клиенту ASP-страницей. Такие компоненты, выполняемые на клиенте, можно применять в связке с клиентскими сценариями для запуска пользовательских и бизнес-сервисов. Причем данный процесс ничем не будет отличаться от работы с удаленными бизнес-объектами. (Удаленные объекты данных подробно описаны в главе 9.) Однако в данном случае следует уделить особое внимание зашите кола и загрузке.

Если компоненты загружаются и устанавливаются с помощью механизма загрузки Internet Component Download, их следует упаковать в автоматически устанавливающийся исполняемый файл или архив (.cab-файл) в соответствии с документацией Microsoft Platform Software Development Kit (SDK). Загружаемый файл должен быть подписан цифровой подписью, благодаря чему пользователь сможет убедиться, что файл получен из надежного источника и не поврежден. Все компоненты библиотеки также должны быть включены в установочный пакет. Если распространяется средство разработки, то в его документации надо указать библиотеки, используемые генерируемыми с его помощью компонентами. Кроме того, следует описать процесс подготовки этих компонентов к загрузке.

Независимо от того, предназначены ли компоненты для загрузки или нет, рекомендуем убедиться, что они безопасны при применении в сценариях и при инициализации. Это необходимо, поскольку некоторые обозреватели отказываются работать с небезопасными компонентами. Чтобы пометить компонент как безопасный, можно использовать записи реестра или реализовать интерфейс IObjectSafety, подробно описанный в документации Platform SDK.

Методы доступа

Доступ к «родным» приложениям

Важная особенность «родных» приложений — то, что для доступа к ним используется операционная система, Поэтому все исполняемые файлы, DLL и дополнительный код приложения должны быть известны и доступны клиентской операционной системе. Такие программы обычно запускаются щелчком значка или из меню Programs (Программы). Чтобы доступ к программе стал возможен, необходимо выполнить процедуры установки и настройки приложения.

Доступ к Web-приложениям

Доступ к Web-серверу средствами обозревателей имеет некоторые особенности. К Web-приложениям обращаются с помощью обозревателя по их имени в форме указателя ресурса. Это имя преобразуется посредством *системы доменных имен* (Domain Name System, DNS) в IP-адрес, после чего с Web-сервером устанавливается TCP/IP-соединение, использующее коммуникационный протокол HTTP.

Устаревшие или не поддерживающие протокол HTTP 1.1 обозреватели запрашивают все изображения, страницы, элементы управления и апплеты по отдельности, устанавливая каждый раз новое соединение. В результате передается лишняя информация и увеличивается время загрузки данных. Обозреватели, поддерживающие HTTP 1.1, способны выполнять несколько запросов за одно соединение, не загружая лишние данные и экономя время.

Преобразованное с помощью DNS имя и IP-адрес некоторое время (определенное пользователем) хранятся в памяти клиентского компьютера. При запросе следующего ресурса обозреватель сначала ищет соответствующий ему IP-адрес в памяти. Если он его не находит, то снова обращается к серверу DNS,

Масштабирование Web-сервисов

Раньше, при применении статических Web-страниц, процесс масштабирования Web-сервера был проще. Создавались копии узла на нескольких серверах, причем каждый из них имел свой IP-адрес (рис. 7.1). С помощью специальной процедуры все эти адреса связывались с од-

ним доменным именем так, что при каждом новом запросе сервер DNS возвращал новый адрес, циклически распределяя запросы между серверами. Таким образом, все запросы равномерно распределялись между серверами и обрабатывались ими посредством этого простого метода *распределения нагрузки* (load balancing). Однако при работе с такой системой не удавалось выяснить, с каким сервером установлено соединение в данный момент.

Сейчас, когда применяются динамические страницы и сценарии, прежний способ неприменим. Web-серверы поддерживают постоянные соединения, многие страницы используют переменные уровня сеанса для взаимодействия с пользователем. Очевидно, что циклическое распределение нагрузки, подразумевающее перемещение между серверами во время одного сеанса, ведет к потере клиентской информации. Дело в том, что другой сервер не содержит данных предыдущего соединения и даже может потребовать новой аутентификации пользователя.

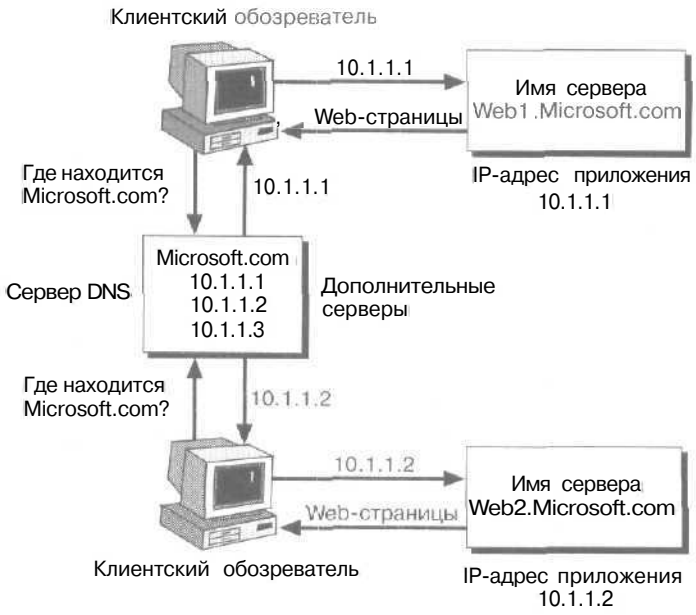


Рис. 7.1. Конфигурация Web-сервера, использующего систему распределения нагрузки

Простейшее решение данной проблемы — запретить переходы между серверами. Это достигается средствами так называемых *«липких»* (sticky) соединений. Клиентский компьютер преобразует указа-

тель ресурса в постоянный IP-адрес сервера. Этот центральный адрес - на самом деле промежуточный, контролирующий серверы, которым передаются запросы (рис. 7.2). Входящие запросы передаются от такого промежуточного сервера на другой, причем повторные запросы того же клиента всегда передаются одному и тому же серверу.

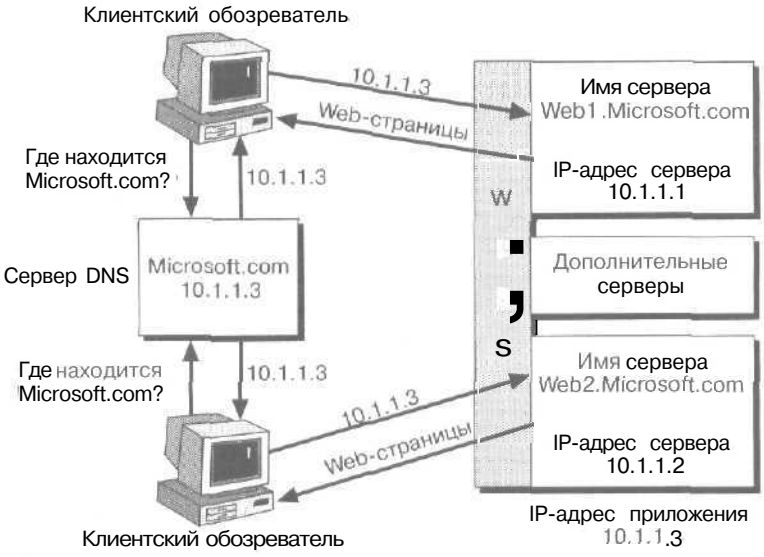


Рис. 7.2. Добавление серверов с целью повышения производительности

Компания Microsoft разработала систему распределения нагрузки Windows Load Balancing Service (WLBS), а Cisco Systems — отказоустойчивую систему Load Director, поддерживающие «липкие» соединения. WLBS постоянно отслеживает состояние Web-серверов, выявляет неисправные и запрещает передачу им запросов. Чтобы добавить в систему новый сервер, достаточно настроить его и добавить в кластер WLBS. Эти масштабируемые технологии позволяют справиться с ростом числа пользователей приложения, практически не меняя его код.

Связывание данных

Чтобы вернуть отсоединенные объекты ADO **Recordset** клиентскому компьютеру, используют RDS. Для присоединения этих объектов к HTML-элементам Web-страниц можно применять связывание данных DHTML, появившееся в Internet Explorer 4.0. К тому же связывание данных и клиентские сценарии позволяют пользователям про-

сматривать объекты **Recordset,** не подключаясь повторно к Web-серверу для получения дополнительных записей.

Поддержка связывания данных, встроенная в Internet Explorer 4.0 и более поздние версии, не ограничена какими-либо определенными типами данных. Она применима для любого типа данных, для которого существует *объект — источник данных* (Data Source Object, DSO). DSO отвечает за передачу информации между клиентом и сервером, за ее обработку и, кроме того, предлагает объектную модель доступа из сценариев. Для определения источника данных, полей, связанных с элементами HTML, формата данных и количества отображаемых на странице записей используются такие атрибуты DHTML, как DATASRC, DATAFLD, DATAFORMATAS и DATAPAGESIZE.

Объект **RDS.DataControl** служит объектом-источником данных для объектов **Rowset** OLE **DB и Recordset** ADO. Он поддерживает как двухуровневую, так и трехуровневую модель доступа к данным. В двухуровневой модели информация об источнике данных внедрена в Web-страницу, и каждый клиент устанавливает соединение с этим источником. В трехуровневой модели для возвращения отсоединенного объекта **Recordset** клиенту, связанному с объектом **DataControl**, используются бизнес-объекты.

Visual Studio 6.0 поддерживает связывание данных для объектов ADO **Recordset.** Разработчики могут посредством объекта ADO **Data Control** устанавливать соединение между элементами управления, связанными с данными, и объектом **Recordset.** Сначала создается объект **Data Control,** Затем его свойство **Recordset** назначается возвращенному бизнес-объектами объекту **Recordset.** Свойство **DataSource** каждого связанного с данными элемента управления формы или диалоговой панели передается объекту **Data Control.** Кроме того, это свойство позволяет присоединить к каждому элементу управления объект **Recordset.**

Связывание пользовательского и прикладного уровней

В многоуровневых приложениях каждый уровень должен иметь возможность общаться со своими соседями. Следовательно, пользовательский и прикладной уровень должны уметь передавать друг другу управление и обмениваться информацией. В обоих уровнях можно использовать объекты. Кроме того, многие ошибочно полагают, что пользовательский уровень и интерфейс — единственные части приложения, затрагивающие компьютер-клиент. На самом деле довольно часто на клиентский компьютер устанавливаются и бизнес-объекты.

Доставка бизнес-объектов на клиентский компьютер

Для создания объектов компоненты необходимо правильно настроить. Прежде всего, разработчику надо либо подготовить идентификатор класса CLSID и имя удаленного сервера для вызова создающей объект функции **CoCreateInstanceEx**, либо занести эту информацию в реестр клиентского компьютера. Для доступа к компонентам средствами Visual Basic и большинства языков сценариев эта информация должна содержать такие записи реестра для объекта, как **ProgID**, **CLSID и RemoteServerName**.

Примечание Бизнес-объекты, работающие в среде MTS, легче всего обеспечить данной информацией, запустив на каждом клиентском компьютере программу их установки (описание MTS см. в главе 8). Специальная программа установит и зарегистрирует все библиотеки типов или DLL-заместители, необходимые для связывания по виртуальной таблице или позднего связывания при доступе к компонентам.

Чтобы автоматически загрузить программу установки с помощью сервиса Internet Component Download, в Web-страницах применяется тэг <OBJECT>. Его параметр CODEBASE указывает обозревателю местонахождение загружаемой программы. Чтобы автоматически создать записи реестра об удаленных компонентах, нужно создать объект посредством тэга <OBJECT>; загрузить и записать в реестр информацию о компонентах с помощью сценариев невозможно,

Помимо регистрации загружаемых компонентов, следует изучить и вопросы безопасности ActiveX-элементов и бизнес-объектов, используемых на Web-страницах. Необходимо, чтобы программы установки имела цифровую подпись, которая позволит обозревателю выяснить происхождение программы и убедиться в се целостности. При определенных параметрах обозревателя неподписанные компоненты и вовсе не будут загружаться.

Примечание Документация Platform SDK и документация некоторых средств разработки (например, Visual Basic) содержит информацию о цифровой подписи и средствах ее создания.

ActiveX-элементы должны быть помечены как;

- **безопасные при запуске сценариев** — это уведомит пользователей, что сценарии, выполняющиеся на клиенте, не будут применять компоненты приложения для нанесения вреда компьютеру пользователя или для несанкционированного доступа к информации;

- **безопасные при инициализации** — это уведомит пользователей, что элементы Web-страницы не причинят никакого вреда компьютеру пользователя.

Проще всего пометить компонент как безопасный при запуске сценариев и при инициализации, добавив в раздел реестра CLSID следующие подразделы:

```
Implemented Categories\{7DD95801-9882-11CF-9FA9-00AA006C42C4}
Implemented Categories\{7DD95802-9882-11CF-9FA9-00AA006C42C4}
```

Если компоненты не помечены как безопасные, обозреватель не сможет создавать объекты и вызывать методы,

Доступ к бизнес-объектам в «родных» приложениях

Способы доступа обычных приложений к удаленным бизнес-объектам с помощью DCOM не слишком отличаются от доступа к COM-компонентам (подробное описание DCOM — в главе 8). Разработчики просто вызывают функции **CoCreateInstance**, **CoCreateInstanceEx** или любую другую функцию, создающую объект, из используемого ими языка программирования. После создания DCOM-объекта его методы можно вызывать, как обычно.

Примечание Метод **CreateInstance** контекста объекта следует использовать только для создания MTS-объектов из среды MTS. С помощью этого метода нельзя создавать удаленные бизнес-объекты, вызывая его в приложении, запущенном на клиентском компьютере: такие приложения сами являются клиентами, работающими вне среды MTS.

«Родные» приложения-клиенты должны хранить интерфейсные указатели на бизнес-объекты MTS, а не создавать их заново каждый раз, когда они требуются. Это позволит избежать расходов на поиск удаленных серверов и установление соединения. Обычно объекты создаются при инициализации приложения или при первой попытке доступа к нему.

Интерфейсные указатели, возвращаемые после создания объекта, нужно сохранять в переменных приложения. Эти указатели не следует освобождать до закрытия приложения, иначе при вызове методов возникнет ошибка из-за невозможности доступа к удаленному серверу. В этом случае освобождение и повторное получение интерфейсных указателей позволит воспользоваться средствами MTS для обработки ошибок на сервере.

Доступ к бизнес-объектам из Web-приложений

COM-объекты на Web-странице **можно создать** с помощью:

- **HTML-тэга <OBJECT>** — *в* этом случае объекты создаются при отображении страницы; этот метод обычно используется для видимых объектов (например, ActiveX-элементов);
- **сценария** — в этом случае объекты создаются только во время выполнениясценария.

Эти способы создания COM-объектов годятся как для клиентских. так и для удаленных COM-объектов. Для вызова методов объектов независимо от способа их создания применяются клиентские сценарии. Они получают доступ к объекту с помощью интерфейса **IDispatch**.

Чтобы получить доступ к объекту из клиентского сценария при использовании тэга <OBJECT>, следует указать идентификатор класса CLSID бизнес-объекта и его идентификатор. Создать экземпляр компонента можно следующим тэгом:

```
<OBJECT ID="objCustomer" CLASSID="clsid:6FED8869-EAC5-11D1-80F4
00C04FD61196"> </OBJECT>
```

Код сценария, создающего объект, зависит от выбранного языка. В VBScript вызывается функция **CreateObject:**

```
<SCRIPT LANGUAGE="VBScript">
Dim objCustomer


Set objCustomer = CreateObject("bus_CustomerC.Customer")
...
Set objCustomer = Nothing
</SCRIPT>
```

В JScript применяется объект **ActiveXObject**:

```
<SCRIPT LANGUAGE="JScript">
var objCustomer;
objCustomer = new ActiveXObject("bus_CustomerC.Customer");
...
objCustomer = '';
</SCRIPT>
```

После создания объекта можно обращаться к его методам, используя его идентификатор (ID) как имя переменной:

```
<SCRIPT LANGUAGE="VBScript">
```

```
Dim rsCustomer
Set rsCustomer = objCustomer.GetByEmail("someone@microsoft.com")
</SCRIPT>
```

Доступ к удаленным объектам с помощью RDS

Еще один способ доступа к удаленным объектам с клиентских компьютеров — вызов с помощью RDS через DCOM или HTTP. RDS отлично подходит для обращения к удаленным объектам на Web-страницах. Однако, как и в большинстве языков сценариев, вызывать можно только методы, доступные через интерфейс **IDispatch.**

Чтобы использовать компоненты в связке с RDS поверх DCOM, их нужно пометить как безопасные при запуске сценариев и при инициализации. На клиентском компьютере достаточно внести в реестр запись, связывающую **ProgID** этого компонента и его CLSID.

Чтобы использовать бизнес-объекты в связке с RDS поверх HTTP, их идентификаторы **ProgID** надо добавить в раздел реестра **ADCLaunch** на сервере:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\ADCLaunch
```

Для обращения к бизнес-объектам используется объект **RDS.Dataspace**. Следующий фрагмент кода иллюстрирует создание удаленного объекта и вызов его метода **GetByEmail** с помошью RDS по протоколу HTTP:

```
<SCRIPT LANGUAGE="VBScript">
Dim rdsds, objCustomer, rsCustomer
Set rdsds = CreateObject("RDS.Dataspace")
Set objCustomer = rdsds.CreateObject("bus_CustomerC.Customer",
"http://webservername")
Set rsCustomer - objCustomer.GetByEmail("someone@microsoft.com")
</SCRIPT>
```

Метод **C'reateObject** объекта **DataSpace** создает клиентский прокси-объект, который вызывает методы средствами интерфейса **IDispatch** поверх DCOM или HTTP. Первый параметр определяет **ProgID** создаваемого объекта, второй — имя сервера. Этот параметр управляет способом вызова объекта — через DCOM или HTTP. Если имя сервера задано в виде *\\machineName*, используется DCOM; если имя задано в виде указателя ресурса *http://machineName*,применяется HTTP. При вызове поддерживается защищенный HTTP-доступ по протоколу Secure Sockets Layer (SSL).

Примечание Более подробное обсуждение RDS, включая примеры кода, вы найдете в документации Microsoft Data Access SDK из состава Platform SDK. Еще одну замечательную книгу (Kamaljit Bath «Remote Data Service in MDAC 2.0») можно загрузить с узла *http://msdn.microsoft.com.*

Резюме

Мы начали главу с обсуждения важных вопросов, возникающих при разработке пользовательского интерфейса и реализации пользовательских сервисов. Далее затронули вопросы защиты, операционных систем, обозревателей коммуникаций.

Мы рассказали об основах проектирования пользовательского интерфейса, упомянули о том, что он должен быть простым и логичным, обсудили основные элементы интерфейса и их применение. Кроме того, описали разработку дизайна интерфейса, в том числе композицию, удобство применения и модель помоши пользователю.

Далее мы обсудили критерии выбора типа интерфейса приложения, затронув не только характеристики «родных» и Web-приложений, но и способы их создания и доступа к ним. В заключение мы рассмотрели механизмы взаимодействия пользовательского и прикладного уровней многоуровневых приложений.

Закрепление материала

1. Что такое композиция?
2. Что понимают под удобством интерфейса?
3. Перечислите вопросы, возникающие при проектировании пользовательского уровня.
4. Какими способами можно применять COM-объекты в пользовательском уровне?
5. Каким образом ASP взаимодействует с пользовательским и прикладным уровнями и уровнем данных?

ГЛАВА 8

Технологии прикладного уровня

В этой главе

Бизнес-объекты служат для инкапсуляции бизнес-правил и служебных функций приложения. Реализация компонентов — несложное упражнение в кодировании, если только они спроектированы в соответствии с рекомендациями глав 2–6. Однако, чтобы бизнес-объекты работали корректно, на стадии разработки необходимо уделить особое внимание некоторым вопросам, особенно если разработка ведется в среде Microsoft Transaction Server (MTS):

- использованию контекста объекта для управления состоянием;
- использованию явно определенных интерфейсов;
- функциональному составу компонентов;
- сохранению состояния в границах транзакции;
- контролю ошибок;
- программному управлению защитой.

Эти вопросы обычно приходится решать для бизнес-объектов; для объектов данных они как правило неактуальны.

Кроме того, в этой главе описана *компонентная объектная модель* (Microsoft Component Object Model, COM) и ее применение в бизнес-сервисах приложения. Сначала в физический проект, использующий COM, интегрируют бизнес-правила и процессы, а затем сетевые технологии и элементы, воздействующие на среду.

Эта глава основана на нашем собственном опыте проектирования и реализации архитектуры приложений и следующих материалах:

- Мэри Киртлэнд (Mary Kirtland «Designing Component-Based Solutions» (Microsoft Press, 1998);
- документы отдела COM компании Microsoft.

Изучив материал этой главы, *вы* сможете:

- ✓ разобраться в Microsoft Component Object Model **(COM);**
- ✓ проанализировать роль COM **в** моделях физического и логического проектирования;
- ✓ разобраться в использовании COM-объектов а транзакциях **под** управлением MTS;
- ✓ разобраться в использовании объектов в сетях;
- ✓ проанализировать влияние среды на прикладной уровень, что поможет создать оптимальный физический проект.

Обзор бизнес-сервисов

Как вы уже знаете из главы 6, перед реализацией бизнес-сервисов *(прикладного уровня)* нужно создать предварительные спецификации основных классов, или *бизнес-объектов,* и распределить их по компонентам. При этом нужно учесть следующие обстоятельства:

- независимо от способа хранения информации, бизнес-объекты инкапсулируют реальные бизнес-операции;
- бизнес-объекты контролируют упорядочение и последовательность применения бизнес-правил, а также целостность выполняемых ими транзакций;
- каждый методбизнес-объекта должен выполнять только одно элементарное действие; каждое такое действие должно быть реализовано только в одном методе;
- вызовы методов бизнес-объектов или объектов данных выполняют высокоуровневые операции;
- бизнес-объект должен функционировать корректно независимо от того, вызван он из транзакции или нет;
- бизнес-объекты, вызываемые *непосредственноиз* презентационного уровня, не должны сохранять свое состояние при вызове методов; бизнес-объекты, вызываемые *в пределах* презентационного уровня, могут сохранять свое состояние в рамках транзакции (в пределах своего сервисного уровня);
- важно сократить до минимума сетевой трафик между удаленными презентационными уровнями и бизнес-объектами;
- так какбизнес-объекты являются «часовыми», контролирующими доступ к данным, для ограничения доступа к бизнес-объектам следует применять ролевую защиту;
- модели обработки ошибок, вызванных методамибизнес-объектов, строятся на основе транзакционных моделей (например, MTS).

Проектирование бизнес-объектов с учетом этих особенностей, помимо упрощения разработки, гарантирует и возможность их повторного использования.

Компонентная модель

Для начала, чтобы вам было легче разобраться в некоторых технологиях Microsoft, мы опишем их основу — COM. Эта глава не претендует на полноту материала — здесь представлены только основные концепции, без которых разработчик не может приступить к проектированию архитектуры приложения. В дальнейшем изучении COM вам помогут книги, перечисленные в библиографическом списке в конце книги.

Примечание Существует еще одна объектная модель — Common Object Request Broker Architecture (CORBA). Подробно она описана в книге Алена Поупа (Alan Pope) «The CORBA Reference Guide» (Addison-Wesley, 1998).

Почему COM?

Главная цель COM — предоставить разработчикам возможность создавать приложения, собирая их из уже готовых частей-компонентов. Компоненты — это не что иное, как физическая реализация бизнес-объектов. Например, в приложении для заполнения бланков заказов можно использовать компонент-таблицу, упрощающий ввод заказываемых товаров. Другой компонент мог бы сам находить город клиента по введенному почтовому индексу. Еще один компонент мог бы рассчитывать налог с продаж. Однако при создании такого приложения в реальности разработчики сталкиваются с некоторыми техническими трудностями.

Во-первых, непонятно, как разместить компонент на компьютере или в сети и, если это удалось, как его запустить. Эти вопросы входят в компетенцию службы каталога для компонентов. Если нет какого-либо стандарта, резко возрастают расходы на изучение компонентов. Кроме того, из-за несогласованности алгоритмов, размещающих компоненты и создающих объекты, затрудняется повторное использование готовых компонентов. Ведь приложение должно уметь находить и выполнять не только компоненты, созданные для внутреннего применения, но и компоненты от сторонних производителей.

Вторая трудность состоит в создании стандартов взаимодействия приложений с компонентами. Как и в предыдущем случае, если таких стандартов нет. накладные расходы на изучение использования компонентов будут препятствовать повторному использованию кода.

Идеальный механизм взаимодействия компонентов не должен зависеть от их местонахождения; он должен работать независимо от того, существует ли этот объект в процессе собственного или другого приложения на локальном или на удаленном компьютере. Реализация удаленной связи и взаимодействия между процессами очень сложна, но стандарты могут значительно сократить время, затрачиваемое на создание такого кода.

Третья трудность — языковая независимость. Все составляющие объекта — выделение памяти, названия методов, типы параметров, соглашения о вызовах и т. д. — должны быть определены так, чтобы объект можно было создать на одном языке программирования, а вызвать его из приложения, написанного на другом языке. При этом разработчики не должны тратить время на выяснение языка или инструментальных средств, с помощью которых был создан компонент. Без широкой поддержки инструментальных средств разные объектные модели разбивают рынок компонентов на мелкие части, что увеличивает затраты на поиск, приобретение и разработку компонентов.

И наконец, последняя проблема — сохранение возможности создания новых версий приложений и компонентов. Приложения, разработанные в разное время, могут использовать одинаковые компоненты, что ведет к конфликтам при запуске на одном компьютере. Кроме того, по мере совершенствования компонентов должна сохраняться совместимость версий. Из-за постоянных обновлений может возникнуть проблема исправления существующих компонентов, расширения их функций и замены новыми компонентами.

Теперь мы обсудим основные элементы и термины COM, часто используемые при разработке приложений.

Объекты

«Объект» — один из самых перегруженных терминов программирования. Но, как и в большинстве объектно-ориентированных моделей, COM-объекты — это экземпляры некоторого класса, представляющего какой-то реальный объект. Точное определение класса мы дадим позже, однако можно сказать, что класс — это тип объектов. Например, рассмотрим три класса с разными характеристиками: **Customer** (клиент), Order (заказ) и **SalesTaxCalculator** (подсчет налога с продаж). В таком случае каждый объект Customer представляет собой какого-либо реального клиента, каждый объект Order — экземпляр заказа и т. д.

Каждый объект обладает идентификатором, состоянием и поведением. Идентификатор — это уникальное имя или метка, отличающие один объект от другого. Состояние — данные объекта. Поведение — набор методов, запрашивающих или изменяющих состояние объекта.

Чтобы разобраться в этих понятиях, рассмотрим объекты C++, являющиеся экземплярами классов C++. В каждом классе определены переменные и методы, присущие только объектам данного класса. При создании объекта для его переменных-членов выделяется непрерывный блок памяти, адрес этого блока становится идентификатором объекта, а его содержимое — состоянием. Размешенные в других блоках памяти методы определяют поведение объекта.

Многие объектные модели языков программирования похожи на модель из C++, но модель COM отличается от нее. Напомним две трудности, связанные с COM, — независимость от языка программирования и местонахождения. Оказывается, что при удаленном вызове и взаимодействии между процессами нельзя однозначно идентифицировать объект по его адресу в памяти. Кроме того, достичь совместимости методики размещения переменных объекта в памяти для всех языков программирования и инструментальных средств практически невозможно.

Из-за этих трудностей подход к объектам в COM отличается от традиционного. В COM открытый интерфейс объекта и его реализация полностью разделены. Приложения могут взаимодействовать с COM-объектами только с помощью их открытых интерфейсов, используя при этом стандартный указатель на интерфейс. По этой причине COM игнорирует местонахождение состояния объекта и внутренние блоки памяти. Кроме того, так как указатель на интерфейс — единственное средство доступа к определенному объекту, то и идентификатор объекта должен быть связан с этим указателем.

Интерфейсы

Чтобы разобраться в COM, нужно изучить интерфейсы — наборы определений, описывающих поведение объекта. При определении интерфейса разрабатываются только спецификации набора функций, но не их реализации. Объявление интерфейса — это контракт между объектом, в котором данная операция реализована, и вызывающим объектом. Контракт предусматривает, что если в некотором компоненте реализован некоторый интерфейс, то вызывающий объект может рассчитывать на то, что этот компонент будет вести себя в полном соответствии со спецификацией интерфейса. Причем в таких спецификациях должно присутствовать строгое определение как синтаксиса методов интерфейса, так и его семантики.

Чтобы быть COM-интерфейсом, интерфейс должен удовлетворять следующим требованиям:

- иметь уникальный идентификатор;
- быть производным от интерфейса **IUnknown**;
- не должен изменяться после публикации.

Идентификаторы COM

Для размещения компонента и обращения к нему необходимы уникальные идентификаторы. Можно было бы использовать строковые идентификаторы, но при этом возникает несколько проблем.

Одна из самых опасных проблем связана со сложностью выбора действительно уникального идентификатора. Даже при наличии соглашения об именах остается возможность, что какой-нибудь разработчик использует уже применяемый идентификатор с другой целью. Для гарантии уникальности имени префиксы должны выдаваться каким-либо центральным органом — например, по одному префиксу для каждой компании. Этим компаниям, в свою очередь, потребуется централизованный реестр имен, который поможет им не допустить появления одинаковых названий. Как видите, простой подход оказывается слишком сложным.

Вместо символьных имен в COM применяются *глобальноуникальные идентификаторы* (Globally Unique Identifier, GUID). Это генерируемые системой 128-разрядные целые числа, уникальность которых обеспечивается алгоритмом их генерации.

Примечание Согласно спецификации COM, можно без повторения создавать 10 000 000 идентификаторов в секунду на каждом компьютере в течение 3 240 лет,

---

GUID можно сгенерировать с помошью утилиты GUIDGEN из комплекта Microsoft Platform Software Development Kit (SDK). Это приложение создает GUID, вызывая функцию прикладного интерфейса **CoCreateGuid**, после чего предлагает выбрать способ его представления. Например, если выбрать *статический постоянный* (static const) идентификатор — обычный метод для включения идентификатора в файлы исходного кода на C++, — получится следующее:

```
// {4C5ECD60-9DDF-11d4-881F-0020AF15C467}

static const GUID GUID_Sample = { 0x4c5ecd60, 0x9ddf, 0x11d4,

{ 0x88, 0x1f, 0x0, 0x20, 0xaf. 0x15, 0xc4, 0x67 } };
```

Первая строка содержит GUID в символьной форме, в которой его обычно и видят пользователи. Во второй и третьей строке GUID представлен R виде константы, которую можно использовать в коде C++.

Примечание Большинство средств разработки автоматизируют процесс создания каркаса COM-компонентов. Они же генерируют соответствующий GUID в формате этого каркаса.

Каждый интерфейс идентифицируется по его GUID. Чтобы обратиться к интерфейсу, мы используем его GUID, который в данном случае называется *идентификатором интерфейса* (interface identification, IID). В качестве IID могут выступать, например, такие значения: {45D3F4B0-DB76-11d1-AA06-0040052510F1} или {45D3F4B1-DB76-11d1-AA06-0040052510F1}.

Для простоты каждый интерфейс согласно стандарту должен иметь символьное название. Обычно такие названия начинаются с буквы «I» — например, **IComputerSalesTax**. Конечно, уникальность символьных имен не гарантируется, но маловероятно, что два разных интерфейса с одинаковыми названиями будут использоваться в одном исходном файле.

Описание интерфейса

Может возникнуть вопрос: как определять интерфейсы, чтобы разработчики компонентов знали, как их реализовать, а разработчики приложений — как их использовать? В COM нет ограничений на средства определения интерфейсов, однако на практике COM-интерфейсы обычно описываются на *языке определения интерфейса* (Interface Definition Language, IDL).

Как и C++, IDL — это язык, описывающий точный синтаксис интерфейса. Определение интерфейса начинается с ключевого слова, перед которым в квадратных скобках помещены атрибуты интерфейса. Если атрибутом является объект, следует использовать IDL-расширения COM; атрибут UUID задает идентификатор описываемого интерфейса,

Примечание Атрибут называется UUID по той причине, что язык описания интерфейсов COM базируется на версии IDL Open Software Foundation. В этой версии вместо термина GUID применяется термин *«универсально уникальный идентификатор»* (Universally Unique Identifier, UUID).

Не все языки программирования поддерживают все типы данных, которые можно определить с помощью IDL, поэтому большинство интерфейсов использует ограниченный набор типов данных. Подробнее этот вопрос обсуждается далее в этой главе. Методам COM можно передавать любое количество параметров, а возвращать они могут данные любой сложности. Помимо типа данных и названия, для каж-

дого параметра можно определить дополнительные атрибуты, несущие информацию о способах копирования данных. IDL — удобный текстовый формат документирования синтаксиса интерфейса. С помощью компилятора Microsoft IDL (MIDL) код IDL можно компилировать, При этом будут сгенерированы файлы определения интерфейса, понятные разным средствам разработки:

- **заголовочный файл** — в заголовочных файлах определены типы IDL, которые могут использоваться при объявлении указателей на переменные интерфейса или при наследовании класса; эти файлы можно включать в программы на C и C++;
- **библиотека типов** — двоичное представление кода IDL, которое используется в языке программирования, например, в Microsoft Visual Basic, для выяснения синтаксиса описанного в ней интерфейса;
- **исходный код DLL-заместителей и представителей** — исходный код библиотек, которые служат для удаленного вызова методов и взаимодействия между процессами.

Несмотря на простоту использования, IDL — все еще новый язык для многих разработчиков, поэтому в большинство средств разработки включены функции, облегчающие его применение. Некоторые системы, например, Visual Basic, полностью скрывают IDL от разработчиков, позволяя определять интерфейсы с помощью синтаксиса системы и автоматически генерируя библиотеки типов. Другие инструментальные средства позволяют создавать файлы IDL с помощью мастеров, помогающих определить интерфейс и его методы.

Определение интерфейса посредством IDL — лишь первый шаг на пути к языковой независимости. Имея файл IDL или сгенерированные из него заголовочный файл или библиотеку типов, можно создать интерфейсное или клиентское приложение при условии, что выбранный язык программирования поддерживает используемые типы данных. Но, чтобы клиентский компьютер и разработанная программа могли взаимодействовать, они должны одинаково «понимать» назначение интерфейсного указателя и способ вызова методов.

COM как двоичный стандарт

Компонентная модель является двоичным стандартом благодаря ориентации на спецификации и наличию реализации. COM основана на спецификации, так как COM-объекты не зависят от языка программирования и местонахождения. Интерфейс COM-компонента можно определить с помощью IDL, соответствующего заголовочного файла или библиотеки типов.

Модель COM основана на реализации, так как обладает системными сервисами размещения и вызова компонентов. Кроме того,

COM поддерживает удаленный вызов и взаимодействие между процессами. COM определяет точное двоичное представление интерфейса, в соответствии с которым все средства разработки должны создавать интерфейсы объектов.

Интерфейсный указатель клиента в действительности ссылается на таблицу указателей, называемую *виртуальной таблицей* (vtable). Каждый указатель этой таблицы ссылается на двоичный код метода интерфейса точно так же, как это происходит в таблице виртуальных функций C++.

Каждый COM-объект содержит указатель на виртуальную таблицу для всех поддерживаемых им интерфейсов. Клиент, запрашивающий интерфейсный указатель на объект, получает *указатель* на указатель в виртуальной таблице, а *не сам* указатель на виртуальную таблицу. Это происходит из-за того, что компонент должен каким-либо образом идентифицировать объект, с которым он работает.

При создании COM-объекта выделяется один блок памяти для указателей на виртуальную таблицу и свойств (данных) объекта. Компонент может идентифицировать объект, распознав связь между положением указателя на виртуальную таблицу и всем блоком памяти объекта. В дальнейшем COM с помощью интерфейсного указателя устанавливает, что первый параметр, передаваемый методу, является указателем на данный объект.

К счастью, большинство языков и средств программирования, поддерживающих COM, автоматически преобразуют интерфейсные указатели и виртуальные таблицы в понятия, свойственные данным средствам разработки. Например, в языке C++ интерфейсы соответствуют абстрактным базовым классам, наследуя которым, можно реализовать интерфейсы. Вызов методов COM с помощью интерфейсного указателя полностью идентичен вызову методов C++ посредством указателя на объект. Приведем другой пример: интерфейсы Visual Basic практически не доступны из самого языка Visual Basic. Их можно реализовать, применив ключевое слово **implements.** Чтобы использовать COM-объект, нужно сначала определить переменную типа «интерфейс», а затем создать объект. После такой процедуры можно вызывать функции как обычно.

Двоичный стандарт, помимо возможности интерпретации определения интерфейса, обладает языковой независимостью и практически полной независимостью от местонахождения. Как уже упоминалось, при его использовании нет разницы между внутренними, внешними и удаленными вызовами. В пределах одного процесса интерфейсный указатель может ссылаться на виртуальную таблицу и методы. Но на разных компьютерах и в разных процессах такой способ.

скорее всего, не будет работать — в этом случае интерфейсный указатель должен ссылаться на виртуальную таблицу библиотеки-представителя. Представитель на клиенте распознает методы, использующие удаленные или внешние (межпроцессные) вызовы объекта сервера или представителя, а этот объект, в свою очередь, выполняет внутренний вызов оригинального объекта. При таком подходе все вызовы методов кажутся клиентам и компонентам одинаковыми.

Интерфейс IUnknown

В виртуальной таблице содержатся три метода — **QueryInterface, AddRef и Release,** не определяемые разработчиком. Эти методы интерфейса **IUnknown** определяют основное поведение всех COM-интерфейсов, причем клиенты могут рассчитывать на это повеление, так как все COM-интерфейсы являются производными от **IUnknown**. С его помощью устраняются технические трудности, связанные с взаимодействием объектов; кроме того, он реализует три важные функции: перемещение по интерфейсам, выяснение их версий и управление временем существования объекта.

Перемещение по интерфейсам

Для перемещения по интерфейсам служит метод **QueryInterface**. Как упоминалось выше, COM-объекты могут поддерживать несколько интерфейсов. В этом случае, если у разработчика есть один интерфейсный указатель и он хочет получить другой, он может запросить его у объекта с помощью метода **QueryInterface.** Этот метод поддерживается всеми интерфейсами как производными от **IUnknown**.

При использовании метода **QueryInterface** клиент передает объекту соответствующий идентификатор интерфейса IID. Если объект поддерживает данный интерфейс, он возвращает указатель на него. В противном случае объект возвращает ошибку. **QueryInterface** — это невероятно мощный механизм, позволяющий независимо созданным клиентам и компонентам согласовывать способ взаимодействия. С его помошью устраняются и проблемы определения версий, описанные в следующем разделе.

Версии интерфейса

Из-за того, что компоненты и приложения создаются независимо друг от друга, опубликованный интерфейс должен быть неизменным. Любые модификации, даже числа методов, могут привести к проблемам, Например, новое клиентское приложение может ошибочно считать, что интерфейс включает пять методов. Если оно обратиться к устаревшему компоненту всего с четырьмя функциями, возникнет ошибка. Именно из-за возможности таких конфликтов COM-интерфейс не должен изменяться.

Новый интерфейс считается новой версией. Существующие приложения, скорее всего, не будут совместимы с ним, но так как исходный интерфейс остается неизменным, реализация интерфейсов в компонентах не повлияет на эти приложения. В новых программах можно реализовать поддержку обновленных интерфейсов и разрешить им доступ к новым функциям. Если же клиент обратится к старому компоненту, он сможет безопасно узнать об отсутствии новых методов с помощью **QueryInterface**.

Таким образом, COM-интерфейс нельзя изменять. При создании его новой версии нужно просто добавить в компонент новый интерфейс, сохранив поддержку старого.

Время существования объекта

Кажется логичным, что объект, созданный на клиентском компьютере, владеющем интерфейсным указателем на него, должен быть уничтожен на том же самом компьютере. Однако процесс удаления ненужного объекта довольно сложен. Например, какой-нибудь клиент может получить с помощью метода **QueryInterface** несколько интерфейсных указателей на один объект. При этом он не сможет определить момент, когда все интерфейсные указатели больше не используются и можно безопасно уничтожить объект — ведь объект, помеченный для удаления, может потребоваться другому клиенту. Ни одно приложение не может выяснить, что другие приложения больше не используют объект, но они могут помочь в этом самому объекту.

Для решения этой проблемы применяется третья функция интерфейса **IUnknown** — *управление временем существования объекта,* обычно называемая *счетчиком пользователей.* Он отслеживает число клиентов, использующих интерфейс. При создании нового интерфейсного указателя вызывается функция **AddRef**, увеличивающая счетчик пользователей на единицу. Клиентский компьютер, не нуждающийся в интерфейсном указателе, вызывает метод **Release** интерфейса **TUnknown,** таким образом уменьшая значение счетчика на единицу. Когда счетчик пользователей оказывается равным нулю, объект уничтожается. Как видите, такое управление временем существования объекта решает все проблемы как в случае одного клиента с несколькими интерфейсными указателями, так и в случае нескольких независимых клиентов. На компьютерах, поддерживающих эту функцию, можно создать объект и получить интерфейсный указатель, а затем вызывать его методы. По окончании работы с объектом необходимо освободить указатель, вызвав функцию **Release** интерфейса **IUnknown**.

Классы

COM-объекты — это экземпляры COM-классов, которые в свою очередь, представляют собой поименованную реализацию одного или

нескольких COM-интерфейсов. Имена классам назначаются с помощью одной из разновидностей GUID — *идентификаторов класса* (CLSID). Как и IID, CLSID уникальны, но пользоваться ими сложнее. Поэтому COM-классам назначают и символьные имена, называемые *программными идентификаторами* (ProgID).

С каждым COM-классом связан *объект класса* (так называемая фабрика классов), способный создавать его экземпляры. В спецификациях COM определены стандартная функция API, создающая объекты класса (**CoGetClassObject**), и стандартный интерфейс для передачи им сообщений (**IClassFactory**). Таким образом, клиентам достаточно одного механизма создания COM-объекта любого типа. Самый важный метод интерфейса **IClassFactory** — **CreateInstance**, создающий объект и возвращающий интерфейсный указатель. Существует два способа создания COM-объекта: клиент может вызвать функцию **CoGetClassObject,** получив таким образом указатель на интерфейс **IClassFactory**, либо передать интерфейсный указатель объекту с помощью метода **CreateInstance** интерфейса **IClassFactory** с последующим удалением этого интерфейса. Так как второй способ применяется довольно часто, для облегчения труда разработчиков была добавлена функция **CoCreateInstanceEx**, сама выполняющая нужные вызовы.

COM-классы отличаются от классов большинства языков программирования тем, что после создания объекта его класс никого не интересует. Взаимодействие с объектом осуществляется с помощью открытых интерфейсных указателей, которые не распознают закрытых классов реализации, использованных для создания объекта. Строгое разделение интерфейса и его реализации — ключевое свойство COM. Эта концепция «черного ящика» значительно упрощает написание клиентов, так как им не нужно знать, как работают компоненты.

Компоненты

COM-компоненты — это двоичные модули для создания COM-объектов. Компонент, отвечающий за определенный CLSID, содержит COM-класс, код, реализующий объект класса и, как правило, код, добавляющий соответствующие записи в системный реестр.

Примечание Иногда компоненты называют серверами, но мы не будем использовать этот термин, чтобы не перепутать сервер-компонент с сервером-компьютером.

В Microsoft Windows 95, Windows 98 и Windows NT существует три типа компонентов: сервисы, исполняемые файлы и библиотеки. Компоненты-сервисы стоит применять, если они должны быть запущены все время, даже если никто не зарегистрирован в системе. Исполняе-

мые файлы применяются, если помимо создания COM-объектов приложение должно обладать пользовательским интерфейсом. Примером такого компонента является Microsoft Word. В большинстве же случаев — например, в трехуровневых приложениях — компоненты собираются в виде библиотек DLL. Элементы управления ActiveX презентационного уровня и транзакционные компоненты, реализующие бизнес-функции, также реализуются в виде библиотек

Кроме того, компоненты можно разделить на три категории но их отношению к клиенту.

- **Внутрипроцессные** — запускаются в том же процессе, что и клиент. Они реализуются в виде DLL.
- **Локальные компоненты** — запускаются в разных процессах на одном клиентском компьютере. Они могут быть как исполняемыми файлами, так и сервисами.
- **Удаленные компоненты** — работают на удаленных компьютерах. Такие компоненты могут быть реализованы в виде исполняемых файлов, сервисов и библиотек. Запускают DLL на удаленном компьютере с помошью *процесса-представителя* или приложения, способного запускать DLL. И в COM, и в MTS существуют стандартные представители DLL-компонентов.

Далее в этом разделе, посвященном COM, мы рассмотрим компоненты, реализованные в виде библиотек, так как они чаще всего встречаются в многоуровневых приложениях.

Структура DLL-компонентов

Помимо реализации COM-классов и объектов, что обязательно для компонентов всех типов, в DLL-компоненте должны присутствовать четыре точки входа:

- **DllGetClassObject** — возвращает интерфейсный указатель на объект класса, определенного в компоненте;
- **DllCanUnloadNow** — показывает, активны ли объекты, созданные компонентом. В случае их активности библиотека должна оставаться в памяти; в противном случае ее можно выгрузить, освободив ресурсы компьютера;
- **DllRegisterServer** — создает элементы реестра, необходимые COM-классам компонента;
- **DllUnregister Server** — удаляет все элементы реестра, созданные с помощью **DllRegisterServer.**

COM самостоятельно вызывает функции **DllGetClassObject и** DllCanUnloadNow, так что приложения не должны вызывать их напрямую. Функции **DllRegisterServer и** DllUnregisterServer обычно вызывают программы установки и средства разработки.

Потоки

COM поддерживает несколько потоковых моделей для компонентов. *Потоковая модель* определяет количество запущенных потоков и способы синхронизации объектов. Windows 95, Windows 98 и Windows NT — многопоточные среды, поэтому компоненты для них нужно писать аккуратно.

Все потоковые модели COM основаны на понятии *отделенного потока* (apartment). *Отделенный поток* — это контекст исполнения объекта. Каждый объект все время своего существования размещается только в одном отделенном потоке; потоки же, в свою очередь, входят в состав процесса. Кроме того, до вызова COM-методов потоки данного процесса должны быть связаны с отделенным потоком с помощью функций **CoInitialize** или **CoInitializeEx.**

Все обращения к объекту происходят из его отделенного потока. Если приложение должно вызывать функции в разных потоках, COM синхронизирует доступ к объекту. Кроме того, при использовании *одиночных отделенных потоков* (Single-Threaded Apartment, STA) для межпотоковых вызовов необходим маршалинг. Он подробно описан в разделе «Удаленная активация и маршалинг». Кратко же можно сказать, что маршалинг — это процесс, перехватывающий вызовы, преобразующий стек вызовов в стандартный формат и после выполнения некоторых действий снова преобразующий его в вызовы методов отделенного потока объекта, Поэтому встроенные в процесс межпотоковые вызовы могут значительно снизить производительность приложения, что еще раз свидетельствует о важности знания потоковых моделей COM.

В Windows 95, Windows 98 и Windows NT COM поддерживает два типа отделенных потоков. Одиночный отделенный поток связан с одним потоком и создается при вызове функций **CoInitialize** или **CoInitializeEx**. В процессе могут существовать несколько таких потоков; первый из них называется *главным.* У процесса может быть и один множественный отделенный поток, с моделью которого можно связать несколько потоков. Чтобы воспользоваться моделью множественного отделенного потока, называемой также *моделью свободных потоков,* нужно присвоить ключу реестра **ThreadModel** значение FREE.

По умолчанию, если раздел реестра **ThreadModel** отсутствует, с STA на протяжении всего времени существования связан один поток, причем одновременный доступ к объектам модели отделенного потока, размещенным в STA, невозможен, так как они запущены в одном потоке. Независимо от количества объектов в отделенном потоке, за один раз может осуществляться вызов только одного метода. Кроме того, в данный момент времени в STA разрешено существова-

ние только одного экземпляра объекта. Таким образом облегчается труд разработчиков, которым не нужно синхронизировать доступ к состоянию каждого объекта — при необходимости можно использовать локальную память потока. Чтобы повысить масштабируемость приложения, можно присвоить ключу реестра **ThreadModel** значение Apartment, что позволит создавать несколько экземпляров объекта в одном STA. При этом глобальные данные компонента и точки входа DLL должны быть реентерабельными.

Синхронизация в STA основана на сообщениях Windows. Вызовы ставятся в очередь специальных сообщений и обрабатываются созданным COM скрытым окном. В результате у потоков, связанных с STA, должны быть *циклы обработки сообщений,* получающие сообщения из очереди, транслирующие их и передающие другим частям приложения. Если у потока нет такого цикла, вызовы не обрабатываются. Модель синхронизации STA не препятствует повторному входу, а наоборот обеспечивает реентерабельность. Она полностью совпадает с моделью, используемой в оконных процедурах. При вызовах методов из других потоков или процессов COM продолжает передавать сообщения, поэтому STA может обработать входящие вызовы и гарантировать ответ окон потока. Вызванные при этом объекты могут без проблем обращаться к другим объектам.

Компоненты, удовлетворяющие модели отделенных потоков, легко написать, но из-за ограничений на параллельные операции их производительность в многопользовательской среде невелика. В таком случае лучше подойдет модель свободных потоков. COM не синхронизирует доступ к объектам во множественном отделенном потоке, для обслуживания параллельных вызовов к которому потоки создаются динамически. Это позволяет получить одновременный доступ к объектам свободных потоков, что увеличивает производительность приложения.

Создать реентерабельный код довольно сложно — ведь глобальные переменные и состояния объектов должны быть защищены. Кроме того, нужно убедиться в реентерабельности функций библиотек компоновки. COM значительно облегчает эту работу, предоставляя возможность выбора потоковой модели при написании кода и разрешая все несоответствия между объектной моделью и потоковой моделью, поддерживающей вызовы.

Эта функция особенно интересна для встраиваемых в процесс компонентов, которые обычно используют потоки процесса клиента, а не создают новые. Клиент может создать объект, вызвав методы **CoInitialize** или **CoInitializeEx** и определив тем самым отделенный поток, связанный с вызывающим потоком. Но возникает вопрос: как

COM убедится в том, что отделенный поток, из которого осуществлен вызов, совместим с потоковой моделью нашего объекта? При создании компонента поддерживаемая им потоковая модель помещается в элементе реестра раздела **InprocServer32**:

```
HKCR\CLSID\{4503F4B1-DB76-11d1-AA06-0040052510F1}\InprocServer32
   @="salestax.dll"

   ThreadingModel="Apartment"
```

По значению элемента **ThreadingModel** COM может определить, в каком отделенном потоке должен быть создан объект (см. табл. 8.1). Например, если вызов осуществляется из STA и значение элемента **ThreadingModel** равно Apartment, объект будет создан в обращающемся к нему STA. Все вызовы объекта будут прямыми, поэтому не потребуется маршалинг. Если же вызов осуществляется из множественного отделенного потока и элемент **ThreadingModel** имеет значение Apartment, объект будет создан в новом STA, а все обращения к нему будут подвергаться маршалингу.

Табл. 8.1. Модели встраиваемых потоков

	Тип отделенного потока, из которого осуществляется вызов		
Значение параметра ThreadingModel	Главный одиночный отделенный поток	Одиночный отделенный поток	Множественный отделенный поток
Не определено	Главный STA	Главный STA	Главный STA
Apartment	Главный STA	Вызывающий STA	Создается новый STA
Free	Множественный отделенный поток	Множественный отделенный поток	Множественный отделенный поток
Both	Главный STA	Вызывающий STA	Множественный отделенный поток

Большинство современных приложений и компонентов либо поддерживают модель отделенных потоков, либо являются однопоточными (используют главный STA). Первые характеризуются оптимальным сочетанием простоты разработки и производительности. Далее в этой главе описаны функции MTS, позволяющие масштабировать такие компоненты для работы с большим числом пользователей.

Модель программирования COM

Модель программирования COM довольно мошна, но в то же время проста, хотя иногда трудно разглядеть эту простоту за всеми ее сервисами. Поэтому мы не будем обсуждать их, а поговорим о самой модели программирования,

COM, OLE и ActiveX

Возможно, разработчики лучше знакомы с OLE и ActiveX, чем с COM, поэтому нетрудно запутаться в значении этих терминов и их связи друг с другом. И это не удивительно, ведь за последнюю пару лет Microsoft изменила определения этих терминов, хотя сами технологии не менялись. Поясним различия.

- COM — фундаментальная модель компонентных объектов, появившаяся в 1992 году. В COM входят только элементы, определенные в ее спецификации, которую можно изучить на Web-узле Microsoft *(www.microsofl.com/com/resources/specs.asp)*.
- OLE — надстройка над COM и, фактически, механизм реализации составных документов. Примером использования OLE может служить документ Microsoft Word, в который вставлена таблица Excel.
- **ActiveX** — маркетинговое название появившихся в 1996 году интернет-технологий на основе COM. В то немного безумное время все основанные на COM элементы, собирались под зонтиком ActiveX, что только внесло дополнительную путаницу. Сейчас все более или менее нормализовалось, и термин ActiveX используется только для обозначения элементов управления ActiveX — специфической технологии на основе COM, применяемой для работы с программными элементами управления. Они используются, например, при помещении элемента управления в форму Visual Basic или при встраивании тэга <OBJECT> в HTML-страницу.

Автоматизация

Изучая COM, мы познакомились с отличительной чертой COM-объектов и интерфейсов — клиентские приложения еще при сборке должны знать COM-интерфейсы, которые они будут использовать. Ведь программа не может создать произвольный COM-объект и обращаться к его интерфейсам, она может обращаться только к интерфейсам, о которых имела представление при своем создании.

При таком положении дел полезно определять тип используемых объектов и соответствующие им интерфейсы во время выполнения. Эту возможность предоставляет *технология автоматизации* (Automation), изначально разработанная для программного управления приложениями из макросов и сценариев, например, в Excel или Word. На момент публикации данной книги в Word и Excel использовался общий язык

создания макросов — Microsoft Visual Basic for Applications. Такие языки не могут содержать информацию обо всех интерфейсах объектов, поэтому она предоставляется приложениям с помощью автоматизации.

Интерфейс IDispatch

Автоматизация определяет стандартный COM-интерфейс программного доступа к объектам — **IDispatch,** После его реализации клиенты смогут использовать любые функции компонентов, к которым они получают доступ с помощью метода **Invoke** интерфейса **IDispatch** (рис. 8.1).

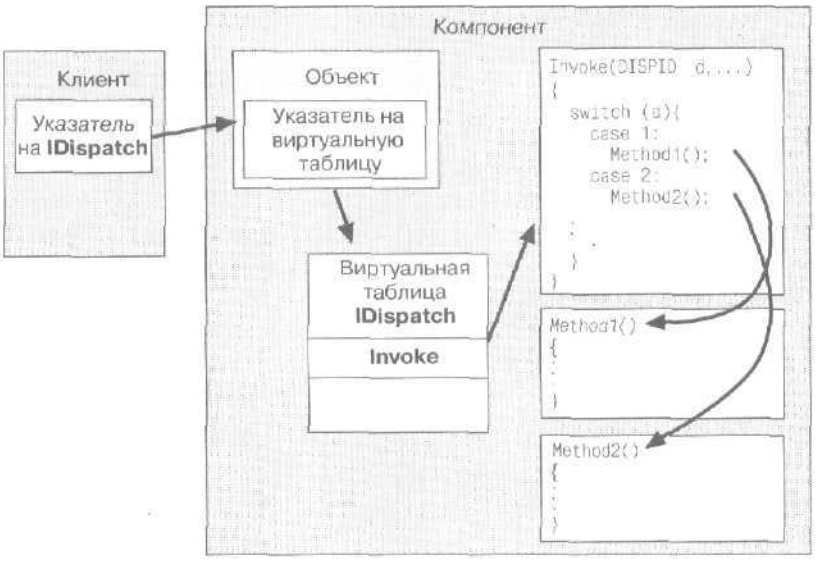


Рис. 8.1. Вызов методов автоматизации с помощью интерфейса IDispatch

В своей простейшей форме, называемой *поздним связыванием,* автоматизация позволяет клиентам обращаться к объектам, не владея никакой информацией об их методах. Клиентское приложение обычным образом создает объект и запрашивает интерфейс **IDispatch.** Чтобы вызвать функцию объекта, нужно передать ее полное название методу **GetIDsOfNames** интерфейса **IDispatch.** Если запрашиваемая функция поддерживается, будет возвращено идентифицирующее ее число — *диспетчерский идентификатор* (Dispatch ID, DISPTD). После этого клиент сохраняет параметры функции в стандартной структуре и передает ее вместе с DISPID методу **Invoke** интерфейса **IDispatch.**

С помощью DISPID объект определяет вызываемую внутреннюю функцию, которой затем передает параметры, выделенные из пере-

данной ему структуры. Если параметры окажутся некорректными, метод **Invoke** может немедленно завершиться, не допустив возникновения сбоя. Полученные после работы внутренней функции данные и/или информация об ошибке сохраняются в стандартной структуре данных и возвращаются в возвращаемых параметрах метода **Invoke.**

Этот довольно запутанный процесс, тем не менее, очень полезен для интерпретируемых языков. Для использования объектов автоматизации интерпретатору сценариев нужно уметь создавать объекты, вызывать методы с помощью интерфейса **IDispatch,** выявлять ошибки и уничтожать объекты. Ему не придется конструировать стековый фрейм лля разных соглашений о вызовах, интерпретировать интерфейсные указатели и выяснять, какие функции есть у объекта. Все это можно скрыть внутри интерпретатора, предоставив в распоряжение разработчиков сценариев простую программную модель.

Можно скрыть большинство таких функций и внутри компонента. Многие средства разработки, поддерживающие COM, по умолчанию создают компоненты, использующие автоматизацию. Если же и язык программирования совместим с COM, как. например, Visual Basic, можно полностью скрыть детали реализации **интерфейса IDispatch,** оставив разработчикам только написание открытых функций. Если какой-либо каркас приложения поддерживает COM, то в нем, как правило, реализован стандартный интерфейс **IDispatch** и определен механизм передачи DISPID внутренним функциям. В этом случае, как и в предыдущем, разработчикам остается написать только нужные им методы.

Но за такую гибкость позднего связывания приходится платить. Во-первых, все параметры, передаваемые методу **Invoke,** должны иметь тип **VARIANT.** Такие данные, помимо самого значения, содержат тэг, определяющий их тип. В автоматизации определен набор типов, которые можно поместить в **VARIANT.** Типы, не входящие в этот список, перед передачей методу в качестве параметра нужно преобразовать.

Примечание На практике проблемы, связанные с этим ограничением, как правило, не возникают, так как набор поддерживаемых автоматизацией типов очень широк. Кроме того, в Windows NT 4.0 SP4, Windows 98 и Windows 2000 можно добавить в этот набор структуры, определенные пользователем.

Во-вторых, объекты могут работать только с одним интерфейсом **IDispatch.** Хотя с помощью разных IID можно добавить дополнительные интерфейсы, основанные на **IDispatch,** ни один язык сценариев не получит к ним доступ. Поэтому большинство объектов автоматизации разрешают клиентам пользоваться только одним программным

интерфейсом. Такое положение вещей может повлиять на проект приложения, особенно при наличии ограничений, связанных с зашитой,

И последнее: при использовании позднего связывания возрастают накладные расходы. Каждый вызов метода приводит к двум обращениям к объекту. Первый — к функции **GetIDsOfNames** для поиска DISPID, второй — вызов метода **Invoke.** Кроме того, дополнительные издержки связаны с преобразованием параметров в структуру при передаче методу **Invoke** и обратным преобразованием при возврате из него. Как правило, такие накладные расходы допустимы в интерпретируемых средах, но чаще всего неприемлемы для других клиентов.

Библиотеки типов

Из-за того что многие клиентские приложения компилируются, им не требуется позднее связывание. В этом случае объекты и методы определяются еще во время разработки.

Вместо того чтобы вызывать **GetIDsOfNames** со всеми накладными расходами, можно определить все DISP1D в коде программы. Такая форма автоматизации называется *ранним связыванием.* При этом привязка компонентов к клиентским приложениям осуществляется во время их сборки. Чтобы раннее связывание работало, среда разработки должна уметь определять DISPID вызванных методов. В идеале она должна определять и правильность передаваемых параметров, что устранило бы большинство ошибок. Для этого необходимы описания методов компонента, которые и хранятся в библиотеках типов.

Диспетчерские интерфейсы

Как и в случае обычных COM-интерфейсов, интерфейсы автоматизации, включая стандартные — для создания и перемещения по информации о типе, — можно определять с помощью IDL. Однако вместо ключевого слова **interface** применяется ключевое слово **dispinterface**. Оно указывает, что интерфейс реализован на основе **IDispatch,** и поэтому можно использовать только типы, поддерживаемые автоматизацией. Методы диспетчерских интерфейсов не заносятся в виртуальную таблицу интерфейса.

Благодаря ключевому слову **dispinterface** разработчики могут явно отделить свойства от методов: свойство представляет собой атрибут, а метод — действие. Например, рассмотрим диспетчерский интерфейс **Rectangle**, содержащий свойства **Height и Width** и метод **Move.** причем свойства реализованы в виде пары функций доступа: одна из них считывает значение, другая его записывает. Обе функции имеют один и тот же диспетчерский идентификатор DISPID; для определения типа операции — чтение или запись — методу **Invoke** передается специальный флаг. Возможность таких действий, способных упрос-

тить не только синтаксис свойств, но и применение объектов, полезна в языках, поддерживающих автоматизацию. Например, в Microsoft Visual Basic Scripting Edition (VBScript) свойства практически неотличимы от переменных:

```
set rect = CreateObject("Shapes.Rectangle")
rect.Left = 10
rect.Top = 10
rect.Height = 30
rect.Width = rect.Height
```

Как уже упоминалось выше, необходимость вручную кодировать определения интерфейсов на IDL возникает редко. Среды разработки, например, Visual Basic, позволяют делать это посредством стандартных конструкций языка, сразу же создавая библиотеки типов. В некоторых других средах такая работа выполняется мастерами, генерирующими правильно отформатированные IDL-файлы, которые можно скомпилировать в библиотеку типов с помошью M1DL.

Двойные интерфейсы

Для компилируемых клиентов раннее связывание значительно эффективнее позднего. Однако, если приложения написаны на языке, поддерживающем связывание по виртуальной таблице, вызов метода **Invoke** может привести к увеличению накладных расходов. Эту проблему решают *двойные интерфейсы* (dual interfaces), обладающие характеристиками как интерфейсов на основе виртуальных таблиц, так и диспетчерских интерфейсов.

Двойные интерфейсы определяются с помощью ключевого слова IDL **interface.** Все они имеют атрибут **dual,** показывающий, что типы параметров методов интерфейса должны поддерживаться автоматизацией. Кроме того, все они являются производными от **IDispatch.** Методы, определенные в двойном интерфейсе, входят в его виртуальную таблицу, поэтому клиенты, использующие связывание по виртуальной таблице, могут вызывать их напрямую. Кроме того, в двойных интерфейсах присутствует и функция **Invoke,** поэтому клиенты, использующие только раннее или позднее связывание, также смогут работать с ними. Метод **Invoke** для определения вызываемой функции по-прежнему использует идентификатор **DISPID.**

Как видите, практически нет причин, по которым стоило бы отказаться от применения двойных интерфейсов. На момент публикации этой книги почти все средства разработки поддерживали их, а большинство совместимых с COM пакетов генерировали их по умолчанию. Двойные интерфейсы позволили без дополнительных усилий

разработчиков повысить производительность клиентских приложений, использующих связывание по виртуальной таблице. Кроме того, двойные интерфейсы могут использовать маршалинг, что избавляет от необходимости установки DLL-заместителей или представителей на клиентских компьютерах.

COM в распределенных средах

При изучении модели программирования COM мы выяснили, что COM-компоненты можно без дополнительных усилий запускать на удаленных компьютерах. Независимость от местонахождения реализована на уровне потоков, процессов и компьютеров. Однако при вызове компонентов на разных компьютерах следует рассмотреть некоторые дополнительные вопросы. В этом разделе мы опишем *распределенную компонентную модель* (Distributed COM, DCOM) и механизм реализации независимости от местонахождения.

Примечание Формально DCOM — это протокол выполнения вызовов объектов на разных компьютерах. Однако часто этим термином называют всю концепцию взаимодействия удаленных объектов с помощью COM. В этом разделе мы ограничимся обсуждением общих вопросов работы распределенных приложений на основе COM.

При выполнении удаленных вызовов возникают вопросы, которые просто не встречаются в случае одного компьютера. Во-первых, по причинам безопасности системы открытый доступ к установленным компонентам может быть нежелателен. Во-вторых, пользователи, обладающие правом доступа к этим компонентам или к компьютерам, на которых они установлены, должны быть обучены работе с ними.

Защита

В COM определены стандартные средства взаимодействия объектов со службами защиты операционной системы. При этом модель защиты COM не зависит от существующих служб ОС.

Средства защиты COM решают два вопроса: кому разрешено запускать компоненты и как обеспечить безопасность вызовов интерфейсного указателя (должна обеспечиваться как безопасность активации, так и безопасность вызовов).

Безопасность активации

Диспетчер сервисов (Service Control Manager, SCM) проверяет безопасность активации при каждом запросе активации объекта. *Активация объекта* — это либо его создание, либо получение интерфейсного указателя на опубликованный объект, например на зарегистрирован-

ный объект класса или на объект из существующей таблицы объектов. Мы рассмотрим вопросы безопасности в первом из этих случаев.

Для проверки безопасности запроса на активацию SCM использует информацию реестра (или информацию, динамически полученную от опубликованного объекта). Сначала SCM проверяет параметры компьютера, чтобы выяснить, разрешены ли удаленные запросы активации объектов. Если проверка прошла удачно, SCM изучает параметры защиты компонента. Более подробно эти параметры описаны в разделе «Регистрация».

По существу реестр может содержать *список контроля доступа* (Access Control List, ACL), где перечислены пользователи, которым разрешается активировать данный компонент. Чтобы разрешить активацию объекта, SCM проверяет наличие клиента в списке контроля доступа. Если для какого-либо компонента такой список отсутствует, используется стандартный список.

Если все проверки прошли успешно, SCM при необходимости запускает компонент и активирует объект; в противном случае доступ запрещается. Чтобы определить контекст защиты и идентификатор пользователя, который должен использоваться объектом для запуска, SCM обращается к информации реестра. Этот идентификатор становится идентификатором клиента для всех запросов активации, выполняемых впоследствии данным объектом.

Безопасность вызовов

Получив указатель на объект, клиент может обращаться к нему. COM обеспечивает проверку безопасности всех обращений к методам посредством интерфейсного указателя. При этом проверке подлежат два аспекта вызова. Первый — аутентификация и авторизация осуществляющего вызов объекта, которые ничем не отличаются от проверок при активации за исключением того, что используется другой ACL, а компонент и клиент располагают некоторыми средствами управления частотой выполнения проверок. Второй относится к целостности и конфиденциальности данных, то есть обеспечению сохранности сетевых пакетов, содержащих COM-методы, и предотвращению доступа к данным пакетов во время их передачи.

Выполнение проверок при каждом вызове метода может привести к падению производительности, поэтому COM предоставляет программам возможность самим определять, когда и как обеспечивать защиту вызовов. И клиентские, и серверные приложения могут установить стандартные параметры защиты вызовов для процесса с помощью функции **CoInitializeSecurity**. В изменяемые ею параметры входят список контроля доступа для авторизации и уровень аутентификации, определяющий се частоту и необходимость контроля цело-

стности и защиты данных. Если приложение не вызовет функцию **CoInitializeSecurity** явно, это будет сделано автоматически перед активацией объектов; информация для вызова по умолчанию берется из реестра. Как и при обеспечении безопасности активации, сначала проверяются параметры компонента, а в случае их отсутствия применяются стандартные параметры.

В распоряжении компонентов и приложений имеются средства контроля безопасности вызовов не только на уровне процесса, но и на уровне отдельных интерфейсов и методов. Для этого используются стандартные интерфейсы **IClientSecurity** и **IServerSecurity**, реализующие более совершенные способы управления параметрами защиты. Подробную информацию о них можно найти в книгах Гая и Генри Эддонов (Guy & Henry Eddon) «Inside Distributed COM» (Microsoft Press, 1998) и «Inside COM+» (Microsoft Press, 1999). Однако большинству приложений вполне достаточно управления параметрами защиты на уровне процесса. Позднее мы опишем используемую в MTS ролевую модель защиты, которая, базируясь на модели зашиты COM, значительно упрощает безопасный доступ к компонентам.

Регистрация

Для обеспечения безопасности в COM используются некоторые разделы реестра. В Windows и Windows NT параметры этих разделов настраиваются с помощью утилиты DCOMCNFG.EXE (рис. 8.2). С ее помошью можно установить параметры защиты на уровне компьютера или приложения с использованием соответствующих параметров реестра.

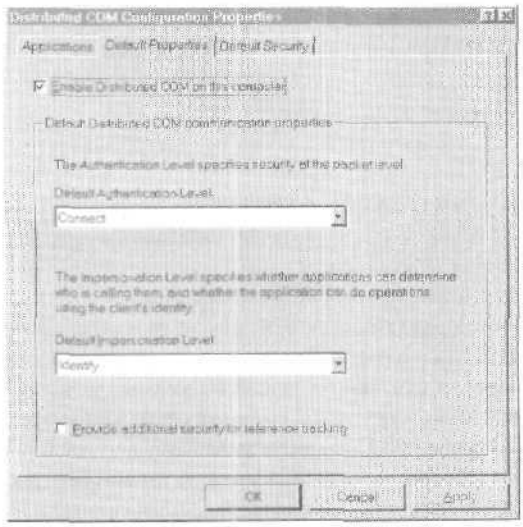


Рис. 8.2. Утилита настройки безопасности DCOM

Некоторые параметры DCOM можно настроить на уровне компьютера. Они хранятся в разделе реестра HKEY_LOCAL_MACHINE\Software\Microsoft\Ole. В табл. 8.2 перечислены основные параметры, их назначение и способ их установки с помощью DCOMCNFG. Эти значения используются по умолчанию при отсутствии параметров для запускаемого приложения.

Табл. 8.2. Элементы реестра, отвечающие за защиту на уровне компьютера

Элемент реестра	Назначение	Настройка
EnableDCOM	Глобальные правила активации для компьютера	На вкладке **Default Properties** установите флажок **Enable Distributed COM On This Computer**
LegacyAuthenticationLevel	Уровень аутентификации по умолчанию для сетевых пакетов	На вкладке Default Properties выберите соответствующий параметр из списка **Default** Authentication Level
DefaultLaunchPermission	Стандартный список контроля доступа для **проверки** безопасности активации	На вкладке **Default** Security в группе **Default Launch Permissions** щелкните кнопку **Edit Default и** отредактируйте список пользователей, имеющих право запуска приложений
DefaultAccessPermission	Стандартный список контроля доступа для проверки безопасности вызовов	На вкладке **Default Security** в группе **Default Launch Permissions** щелкните кнопку **Edit Default и** отредактируйте список пользователей, имеющих право доступа к компонентам

С помощью DCOMCNFG можно настроить параметры для каждого приложения. С точки зрения COM, приложение — это просто процесс, в среде которого выполняются компоненты. Каждое приложение идентифицируется посредством глобально уникального идентификатора, который в данном случае называется AppID.

Зная ApplD приложения, можно настроить для него параметры защиты, которые будут использоваться по умолчанию (если функция **CoInitializeSecurity** не вызвана явно). Эти параметры хранятся в разделе реестра HKEY_CLASSES_ROOT\APPID\{appid-guid} на сервере. В табл. 8.3. перечислены элементы реестра, отвечающие за безопасность, их назначение и способы их настройки в окне **Application Properties** утилиты DCOMCNFG. Чтобы открыть это окно, выберите

приложение в списке на вкладке **Applications** и щелкните кнопку **Properties.**

Табл. 8.3. Элементы реестра, отвечающие за защиту на уровне приложения

Элемент реестра	Назначение	Настройка
RunAs	Учетная запись, используемая для запуска процесса	На вкладке **Identity** выберите учетную запись пользователя, применяемую для запуска приложения
Launch Permission	Стандартный список контроля доступа для проверки безопасности активации	На вкладке **Security** установите флажок **Use Default Launch Permissions** или **Use Custom Launch Permissions,** щелкните кнопку **Edit** и выберите учетные записи пользователей, которым разрешено запускать приложение
AccessPermission	Стандартный список контроля доступа для проверки безопасности вызовов	На вкладке **Security** установите флажок **Use Default Access Permissions** или **Use Custom Access Permissions,** щелкните кнопку **Edit** и выберите учетные записи пользователей, которым разрешен доступ к приложению
Authentication-Level	Уровень аутентификации, применяемый к сетевым пакетам (в Windows NT SP4 и более поздних версиях)	На вкладке **General** выберите из раскрывающегося списка **Authentication Level** нужный уровень аутентификации приложения

Примечание Необходимо зарегистрировать AppID удаленно используемого DLL-компонента — иначе его параметры не удастся менять с помощью DCOMCNFG.

Помимо параметров защиты в реестре хранится информация о местонахождении компонента. Она записывается в реестр при установке компонента на сервере. При этом все, что нужно серверу, - это путь к COM-компоненту, который для каждого идентификатора класса хранится в разделах InprocServer32 или LocalServer32. Для запуска компонента в качестве сервиса понадобятся дополнительные элементы реестра.

Если клиентское приложение при вызове функции **CoCreateInstanceEx** не указывает имя сервера, в реестре компьютеров, запрашивающих удаленные объекты, нужно хранить дополнительную информацию. В частности, им потребуется AppID с параметром **RemoteServerName**. Клиенту могут потребоваться и элементы реестра, описывающие DLL-заместители/представители, используемые для маршалинга интерфейсов. Однако удаленные компоненты не могут сами сохранять в реестре информацию о себе, так как они не установлены на клиентском компьютере.

Существуют два способа создания элементов реестра на Windows-клиенте. Если компоненту не нужны библиотеки-заместители (или представители), можно передать на клиентский компьютер регистрационный файл с нужной информацией. Другой способ — передать установочную программу, которая запишет данные в реестр и установит необходимые заместители. Если же программа установки или регистрационный файл создали AppID, но не указали местонахождение компонента, значение параметра **RemoteServerName** назначают с помощью утилиты DCOMCNFG.

Удаленная активация и маршалинг

Если определены параметры защиты и местонахождение компонентов, можно удаленно создавать объекты. Рассмотрим этот процесс подробнее.

Как уже упоминалось, создать объект очень просто: сначала указателю **IClassFactory** присваивается ссылка на объект класса, затем вызывается его метод **CreateInstance**, в результате чего на созданный объект будет ссылаться интерфейсный указатель. За поиск объекта класса отвечает диспетчер сервисов.

При создании удаленного объекта задействованы два диспетчера сервисов: сначала диспетчер клиентского компьютера выясняет, что запрошен удаленный объект, и связывается с диспетчером сервера. Диспетчер сервера находит объект класса и возвращает интерфейсный указатель диспетчеру клиента (при условии, что все проверки безопасности прошли удачно).

Клиентский SCM может определить, что был запрошен удаленный объект, двумя способами: либо приложение явно указывает имя удаленного сервера при вызове, создающем объект, либо при поиске информации о местонахождении компонента в реестре SCM обнаруживает ссылку на удаленный компьютер (в параметре **RemoteServerName). В** любом случае SCM получит название удаленного сервера и свяжется с ним для получения интерфейсного указателя, с помощью которого приложение сможет вызывать методы объекта. Клиентское

приложение лишь вызывает внутрипроцессные объекты — в этом и состоит вся прелесть независимости от местонахождения.

Можно разработать множество размещенных в коде приложения механизмов, создающих иллюзию внутренних вызовов. Но настоящий интерфейсный указатель передать с удаленного компьютера на клиентский нельзя — ведь на нем этот адрес не имеет смысла. Поэтому клиентскому приложению передается интерфейсный указатель на объект-представитель, который в действительности является встроенным в процесс объектом, Он-то и взаимодействует с соответствующим объектом-представителем в процессе компонента.

Клиентское приложение, вызывающее какой-либо метод, на самом деле обращается к методу объекта-представителя, который преобразует переданные параметры в стандартный формат посредством *маршалинга*. Затем представитель с помощью подходящего механизма взаимодействия процессов передает запрос процессу компонента, который передает его своему представителю. Тот, в свою очередь, распаковывает параметры и вызывает метод реального объекта. Этот процесс называется *обратным маршалингом* или *демаршалингом* (unmarshaling). Точно так же, но только в обратном порядке, возвращаются результаты вызова метода.

Одна из целей COM — скрыть сложность взаимодействия компьютеров и процессов, для чего и служат объекты-заместители и объекты-представители. Маршалинг выполняется не только при взаимодействии разных компьютеров, но и разных процессов и отделенных потоков. Обычно заместители и представители генерируются автоматически при обработке IDL-определений интерфейсов компилятором MIDL. Такие библиотеки вызывают системные функции, инкапсулирующие как маршалинг, так и реальные вызовы. Для обращений к другому компьютеру используется механизм *вызова удаленных процедур* (Remote Procedure Call, RPC). В случае обращения к другому локальному процессу применяется *упрощенный механизм вызова* (Lightweight RPC, LRPC). Если запрашивается вызов из другого отделенного потока, COM автоматически переключает контекст участвующих в этом процессе потоков и синхронизирует доступ к ним.

Обычно достаточно установить и зарегистрировать библиотеки-заместители и представители на соответствующих компьютерах, причем для каждого интерфейса эти библиотеки должны быть зарегистрированы на всех компьютерах, использующих его. Библиотеки, создаваемые MIDL, могут быть саморегистрирующимися — в этом случае для занесения соответствующей информации в реестр достаточно запустить утилиту REGSVR32, указав библиотеку в качестве параметра,

Пакеты MTS

Пакеты — это наборы COM-классов, выполняющих связанные функции. Они также являются простейшими административными элементами MTS, определяющими границы процесса и зашиты.

В MTS существуют два типа пакетов: *пакеты библиотек* и *серверные пакеты*. Пакеты библиотек запускаются в рамках процесса клиента, использующего их COM-классы. Серверные пакеты выполняются как отдельные процессы под управлением MTS. Каждый класс может быть установлен только в одном пакете, находящемся на компьютере. Чтобы распределить нагрузку по нескольким клиентам, можно установить пакет на разных компьютерах.

Примечание Хотя компонент может включать в себя класс, находящийся в нескольких пакетах, на практике этого следует избегать. COM-классы одного компонента обычно имеют взаимозависимости, которые не нарушаются только в случае работы объектов этих классов в одном и том же процессе. Кроме того, реализация класса в нескольких пакетах усложняет администрирование и сопровождение.

Утилита администрирования MTS по умолчанию устанавливает все COM-классы компонента в один пакет. Она считает термин «компонент» синонимом термина «COM-класс». Чтобы не было расхождений с документацией к этой программе, мы будем считать, что все COM-классы компонента собраны в одном пакете. В дальнейшем мы будем изучать администрирование компонентов, а не COM-классов.

Физически пакет — это набор библиотек и элементов каталога MTS, в котором хранится информация о пакетах, компонентах, интерфейсах, ролях и т. д, Данные этого каталога дополняют связанную с COM информацию реестра. Поэтому в MTS 2,0 каталог реализован с использованием реестра. При создании пакетов и добавлении к ним компонентов с помощью программы администрирования MTS информация о них добавляется в каталог.

Чтобы упростить перенос пакета с одного компьютера на другой, можно сохранить информацию о нем в файле (рис. 8.3). Такие текстовые файлы с расширением .pak (пакетные файлы) содержат все элементы каталога, относящиеся к пакету, его ролям, компонентам и интерфейсам. Конечно же, сами библиотеки в таких файлах отсутствуют, но там есть ссылки на них, с помощью которых утилита администрирования MTS находит и регистрирует необходимые библиотеки.

Чтобы MTS мог использовать пакетные компоненты, они должны быть реализованы в виде DLL. Кроме того, в них должна присутствовать функция **DllRegisterServer**, регистрирующая CLS1D, Prog1D, ин-

терфейсы и библиотеки типов компонента и заносящая все необходимые данные в реестр. MTS вызывает эту функцию после установки компонента на сервере. В библиотеках типов должны быть описаны все классы и интерфейсы компонента. Большинство средств разработки генерируют компоненты, удовлетворяющие всем вышеперечисленным требованиям.

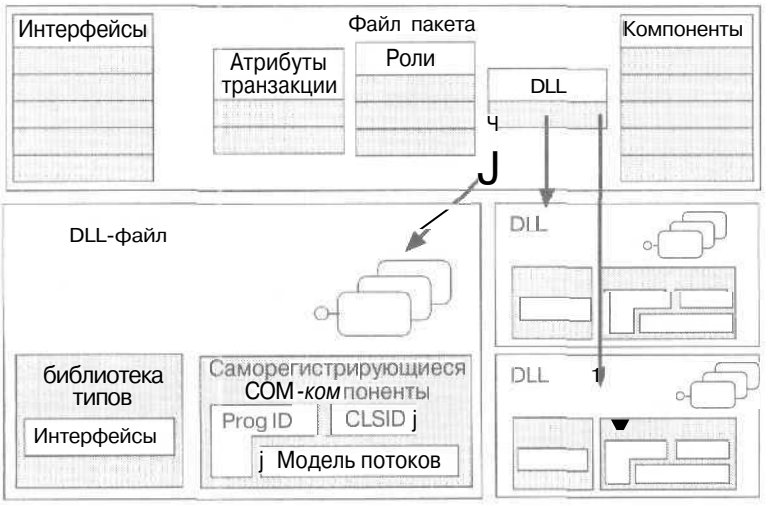


Рис. 8.3. Экспортируемый **пакетный файл MTS**

Проектирование пакетов MTS

Разработка пакета начинается на стадии физического проектирования (см. главу 6). Основная цель такого проектирования — зафиксировать ограничения, накладываемые такими требованиями к приложению, как авторизация вызовов компонента или запуск компонентов на определенном компьютере. Вовсе не обязательно завершать работы по физическому проектированию до начала реализации компонентов. По мере создания приложения физический проект пересматривается и уточняется, компоненты размещаются в пакетах, а ограничения на их развертывание документируются. На этом этапе нужно учесть;

- активацию;
- совместно используемые ресурсы;
- изоляцию ошибок;
- защиту.

Все эти вопросы рассмотрены ниже.

Активация

Серверные компоненты можно активировать двумя способами; в процессе клиента и в процессе пакета, управляемом MTS. Вы уже знаете, что для запуска компонента в процессе клиента используются пакеты библиотек. Обратите внимание, что клиентом в данном случае называется либо другой компонент, либо приложение, запущенное на компьютере, на котором установлен пакет библиотеки. Такие пакеты не поддерживают ни декларативную защиту, ни изоляцию процессов. Поэтому обычно они применяются в компонентах-утилитах, используемых разными приложениями. Пакеты библиотек могут быть полезны и в случае, когда нежелательны накладные расходы, связанные с разделением процессов, и не требуется авторизация.

Серверные пакеты используются для запуска компонентов в отдельных процессах под управлением MTS. Они поддерживают и декларативную защиту, и создание пула ресурсов, и многое другое. Большинство пакетов MTS — серверные. Компоненты приложений группируются в пакеты на основании общих требований к защите, использованию ресурсов и изоляции ошибок.

Определяя необходимое число пакетов, нельзя забывать о равновесии. Когда пакетов много, возрастает гибкость развертывания приложений. Однако каждый серверный пакет должен управлять пулами ресурсов, совместно используемыми свойствами и т. д., что заметно увеличивает накладные расходы. Кроме того, внешние вызовы (вызовы функций другого процесса) связаны с гораздо большими затратами, чем внутрипроцессные вызовы в рамках одного потока. К тому же усложняется задача управления системой. На этом этапе разработки следует создавать пакеты, отталкиваясь от используемых ресурсов, их защиты и изоляции ошибок, а на фазе тестирования производительности при необходимости изменить распределение компонентов по пакетам.

Совместно используемые ресурсы

Компоненты, использующие одни и те же ресурсы (например, базы данных), должны быть собраны н один серверный пакет. Не забывайте, что MTS управляет пулами ресурсов каждого процесса. Поэтому у каждого пакета свой пул потоков, пул соединений с базой данных, *диспетчер общих свойств* (Shared Property Manager, SPM) и т. д. Если два компонента, использующие одну базу данных, размещены в разных пакетах, они не смогут создать пул соединений с базой данных. Но это возможно, если они будут находиться в одном пакете. Такое проектирование пакетов значительно повышает производительность и масштабируемость приложения. Кроме того, как и в вышеописанном случае, компоненты из разных серверных пакетов не смогут со-

вместно использовать общие свойства посредством диспетчера — для этого они должны быть в одном пакете.

При проектировании пакетов следует рассмотреть и размещение ресурсов. В принципе, компоненты должны находиться как можно ближе к ресурсам: таким образом снижается сетевой трафик. Например, если два хранилища данных находятся на разных компьютерах, следует подумать о помещении объектов данных в разные пакеты в зависимости от используемого ими хранилища. Это позволяет установить каждый пакет поблизости от его хранилища данных. Требования и рекомендации по развертыванию и размещению компонентов должны содержаться в документации физического проекта, распространяемой вместе с приложением.

Изоляция ошибок

Размещение компонентов в пакетах гарантирует, что ошибка в одном компоненте не повлечет за собой отказ другого. MTS уничтожает процесс, если обнаруживает его повреждение. Ошибки компонента также могут заставить MTS завершить процесс, причем состояние всех созданных объектов и SPM будет потеряно. Поэтому если в приложении есть компоненты, сохраняющие промежуточное состояние, подумайте об их размещении в отдельном серверном пакете.

Защитная изоляция

В MTS доверие определяется на уровне пакетов, Обращения к пакету могут быть защищенными, но вызовы внутри него считаются надежными. Таким образом, требования к защите приложений сильно влияют на проект пакета. Если обращения к компоненту требуют авторизации, клиенты и компоненты должны быть размещены в разных пакетах. В одном пакете могут находиться только компоненты, способные безопасно обращаться друг к другу.

Роли системы защиты MTS определяются на уровне пакета. Если у нескольких компонентов они совпадают, их можно поместить в один пакет. Как правило, такой метод группирования безопасен, ведь эти компоненты будут применять одни и те же пользователи, а внутренние обращения считаются безопасными. Кроме того, такой подход упрощает администрирование, так как системному администратору не придется назначать множество ролей для разных пакетов.

Помимо прочего, каждый серверный пакет MTS запускается со своим идентификатором, а их компоненты — с идентификатором пакета. Компоненты, запускаемые с разными идентификаторами (например, при наличии нескольких типов прав доступа к ресурсу), должны быть помешены в разные пакеты. Хотя идентификатор пакета в процессе разработки неизвестен, стоит зафиксировать в документа-

ции все рекомендации наряду с разрешениями, необходимыми его компонентам. Например, если объекту данных нужен доступ для чтения/записи, зафиксируйте этот факт.

Реализация COM в среде MTS

Рассмотрим основные концепции создания компонентов, запускаемых в среде MTS. Четыре основные задачи этого процесса таковы:

- создание однопользовательскихвнутри процессных COM-компонентов;
- использование явных интерфейсов для определения открытых интерфейсов объекта;
- использование методов **ObjectContext** для сообщения MTS о том, что состояние объекта может быть изменено;
- управление ошибками с помощью транзакций.

Все компоненты, запускаемые в среде MTS, должны быть внутрипроцессными. Для каждого класса компонента должна существовать фабрика классов и библиотека типов.

Помимо изложенных выше требований, MTS-компоненты должны быть однопользовательскими. Управление потоками осуществляется с помощью *диспетчера MTS* (MTS Executive, mtxex.dll), поэтому разработчикам не нужно заботиться о многопоточности своих компонентов. Не следует самостоятельно создавать потоки внутри компонента — достаточно писать его из расчета, что к компоненту в каждый момент времени будет обращаться только один объект.

Напомним, что с каждым внутрипроцессным компонентом связан элемент реестра **ThreadModel**, описывающий распределение объектов по потокам во время выполнения приложения. Если значение этого элемента не задано, будет использоваться модель одиночного отделенного потока (STA), а все объекты будут выполняться в главном потоке процесса. Доступ к объектам главного потока, которые к тому же склонны к взаимным блокировкам, синхронизирует COM, поэтому этот подход снижает масштабируемость приложения. Однако при использовании нереентерабельных библиотек придется смириться именно с этой моделью.

Оптимальный подход к компонентам в среде MTS — модель разделенных потоков (**ThreadModel**=Apartment). В этом случае каждый объект размещается в отдельном потоке на все время своего существования, а в любом числе потоков можно создать несколько экземпляров объекта, Глобальные данные компонента и точки входа в DLL должны быть реентерабельны и защищены от одновременного обновления разными потоками. Такой подход улучшает масштабируемость, не усложняя реализацию компонентов. Нет необходимости создавать

полностью реентерабельные компоненты, поддерживающие модель свободных потоков; если же они реализованы, они не должны создавать потоки самостоятельно. К счастью, современные средства разработки могут генерировать каркас приложения, удовлетворяющего всем вышеперечисленным требованием.

Примечание Как насчет Java и COM, спросите вы? Java отлично подходит для создания COM-компонентов. Благодаря тому, что Microsoft Virtual Machine for Java и Microsoft Visual J++ поддерживают COM, разработчики могут реализовать компоненты на основе Java-классов. В MTS существует пакет, обеспечивающий возможность работы с Java (com.ms.mtx). Подробнее об этом см. в статье Пола Стэффорда (Paul Stafford) «Writing Microsoft Transaction Server Components in Java» (Microsoft Developer Network Library).

Стандартные заготовки, предоставляемые средствами разработки, подходят для компонентов, используемых в трехуровневых приложениях. При их использовании разработчики могут больше времени уделить алгоритмам программы, не кодируя вручную основную инфраструктуру компонента (если вас устраивает производительность стандартного кода). Так как COM — двоичный стандарт, можно изменить любой компонент, не затрагивая его клиентов, при условии, что открытый интерфейс не изменялся — даже если вы перешли на другой язык программирования. В этой книге описывается стандартная реализация COM в Microsoft Visual Studio. Информацию о настройке компонентов и подробные описания COM можно найти во многих книгах (большинство из них адресованы программистам на C++); например, в книге Дэвида Круглински (David Kruglinsky) «Programming Visual C++, Fifth Edition» или Гая и Генри Эддонов (Guy & Henry Eddon) «Inside Distributed COM» (Microsoft Press, 1998).

После генерации каркаса внутрипроцессного компонента к нему можно добавить реализации классов. Как известно, доступ к классам осуществляется с помощью интерфейсов, однако в некоторых языках программирования — например, в Visual Basic — методы и свойства можно открывать непосредственно на уровне класса; их преобразование в COM-интерфейсы выполняется автоматически. Старайтесь избегать такого стиля — определяйте интерфейсы с помощью IDL. Мы уже упоминали, что интерфейсы представляют собой контракт между объектами и что после публикации их нельзя изменять. Однако после того, как определения интерфейсов будут реализованы, в них можно вносить изменения, не заботясь о последствиях. Раздельное определение интерфейсов очень удобно, когда над приложением трудятся несколько разработчиков. В таком случае удается быстро со-

здать тестовые объекты и проверить правильность кода приложения, обращающегося к данному интерфейсу.

К реализации методов интерфейса предъявляются два требования. Во-первых, MTS должен быть уведомлен о завершении изменения состояния объекта. Во-вторых, ошибки должны быть обработаны объектом, в котором они возникли; кроме того, следует уведомить MTS о том, успешно или нет выполнен метод. С этой целью используется контекст объекта. Метод **SetComplete** интерфейса **IObjectContext** сигнализирует об успешном завершении работы объекта, а метод **SetAbort** — о возникновении ошибок.

При вызове метода **SetAbort** объекта, участвующего в транзакции, она прерывается, а затронутые ресурсы возвращаются в предшествующее ей состояние. Такой механизм обработки ошибок чрезвычайно полезен в случае совместной работы независимых объектов. Если хотя бы один из них не сможет завершить свои действия, вся операция отменяется.

Так как большинство объектов не сохраняют свое состояние во время вызовов методов, структура этих методов должна выглядеть следующим образом:

```
MyMethod(…)
On Error Goto ErrorHandler
CtxObject = GetObjectContext()

Do work
If error condition then
Raise error

CtxObject.SetComplete
Return success

ErrorHandler:
CtxObject.SetAbort
Return error
```

Примечание Помимо вызова **SetAbort** в случае сбоя должна быть возвращена и обычная COM-ошибка, которая позволит клиентам оценить целесообразность продолжения работы. Вызов метода **SetAbort** влияет только на завершение транзакции.

Все вышеописанное относится как к прикладным объектам, так и к объектам данных. Теперь, имея каркасы компонента и классов при-

ложения, мы можем перейти к более интересному занятию — реализации алгоритмов приложения.

Основные прикладные сервисы Windows NT

В Windows NT представлено много сервисов, которые раньше приходилось реализовывать разработчикам приложений. Среди них:

- системные сервисы, защищающие приложение от воздействия других приложений;
- управление памятью;
- сервисы виртуальной памяти;
- кэширование файлов.

В этом разделе мы расскажем об архитектуре Windows NT и способах взаимодействия приложений с операционной системой с целью использования прикладных сервисов.

Пользовательский режим и режим ядра

Архитектура Windows NT разделена на две части: *пользовательский режим* и *режим ядра*. Как известно, Windows NT контролирует доступ к памяти и другим устройствам. Поэтому приложения, запущенные в пользовательском режиме, не могут обращаться к оборудованию напрямую. Они могут получить доступ к ресурсам (файлам, принтерам, коммуникационным портам и даже процессору) только с помощью исполняющей подсистемы Windows NT (Windows NT Executive Service), работающей в режиме ядра,

На рис. 8.4 проиллюстрирована архитектура Windows NT, а также механизм доступа к ресурсам системы.

Доступ ко всем ресурсам контролирует исполняющая подсистема. Обращения к таким ресурсам, как файлы, системные устройства и сеть, обрабатывает диспетчер ввода/вывода. Диспетчер виртуальной памяти отвечает за управление памятью компьютера и настройку ее защищенных областей для каждого приложения, запущенного в пользовательском режиме. В микроядре реализованы основные сервисы, такие как обработка прерываний и управление потоками.

В пользовательском режиме выполняются приложения и пользовательские процессы. Передать информацию из такого приложения в режим ядра удается только одним способом — с помощью исполняющей подсистемы. Ни обойти систему защиты Windows NT, ни получить прямой контроль над физическими ресурсами нельзя.

Пользовательский режим состоит из нескольких подсистем, в которых размещаются приложения. Эти подсистемы, в свою очередь, передают сообщения исполняющей подсистеме и принимают сообщения, поступающие от нее. В них входят подсистемы OS/2, POSIX и подсистема защиты.

Из рис. 8.4 ясно, что Win32-приложение взаимодействует с соответствующей подсистемой, посылая ей сообщения. Подсистема передает запросы сервису Windows NT Executive, который, проверив безопасность операции (на предмет нарушения ею целостности системы), выполняет ее.

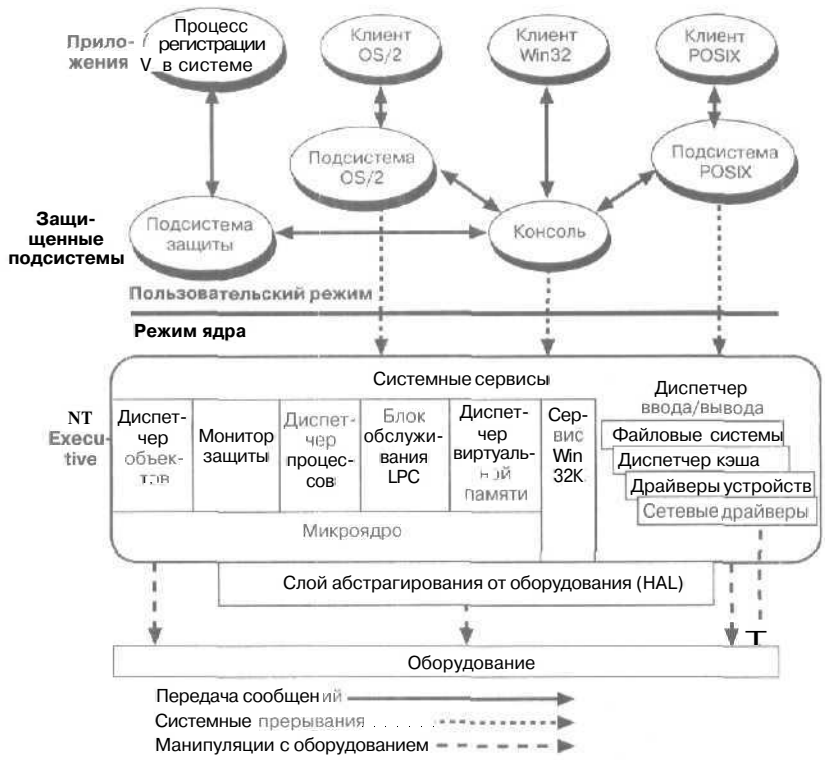


Рис. 8.4. Архитектура Windows NT

Виртуальная память

Памятью также управляет исполняющая подсистема Windows NT. Диспетчер виртуальной памяти выделяет приложениям память, управляет виртуальной памятью и файлом подкачки. Каждой программе выделяется свой адрес в памяти, а объем памяти, доступной приложениям, в разных операционных системах отличается:

- **Windows NT Server 4.0** может выделить каждому приложению 4 Гб памяти; приложение может адресовать 2 Гб;
- **Windows NT Server 4.0 Enterprise Edition** может также выделить 4 Гб памяти, но позволяет приложениям адресовать 3Гб;

- **Windows 2000** Server поддерживается расширенную архитектуру работы с памятью Enterprise Memory Architecture, благодаря которой приложение может адресовать до 64 Гб памяти.

Процесс выделения виртуальной памяти. проиллюстрированный на рис. 8.5, в общих чертах таков.

1. Приложение передает запрос на сохранение данных в памяти.
2. При обращении приложения диспетчер виртуальной памяти проецирует запрос на неконфликтующий адрес. Затем диспетчер перехватывает запрос, определяет число требующихся страниц и проецирует неиспользуемую физическую память на незанятое адресное пространство виртуальной памяти приложения. При этом диспетчер виртуальной памяти скрывает от приложения организацию физической памяти.
3. В случае нехватки физической памяти диспетчер виртуальной памяти применяет метод *подкачки страниц по требованию,* при котором редко используемые страницы копируются в *файл подкачки* (pagefile.sys) на жестком диске. В освобожденную таким образом память помещаются данные, затребованные обратившимся к памяти приложением.
4. Если возникает потребность в данных из файла pagefile.sys, они копируются обратно в память. Новая область ОЗУ проецируется на виртуальный адрес, затребованный приложением.

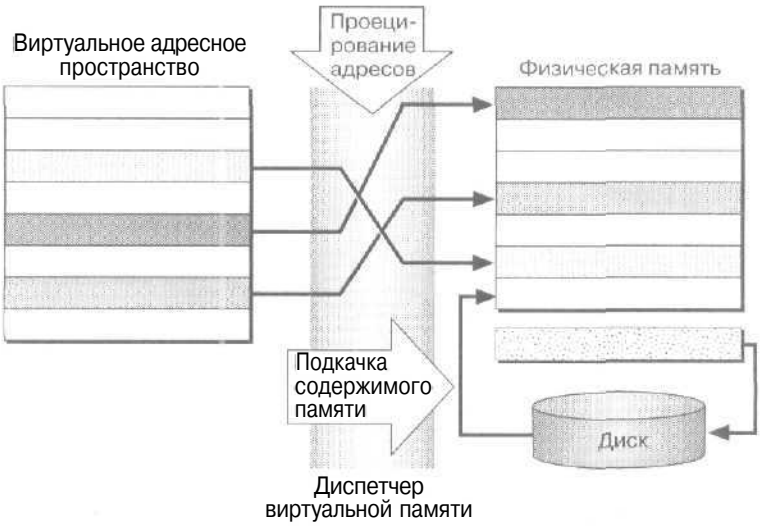


Рис. 8.5. Подсистема виртуальной памяти

Кооперативная многозадачность

Многозадачность — это кажущаяся одновременной работа нескольких приложений и процессов на однопроцессорной системе. Существует два типа многозадачности: кооперативная и вытесняющая.

Операционные системы семейства Windows 3.1 использовали кооперативную многозадачность. Операционная система предоставляет приложению доступ к системным и процессорным ресурсам, но если они требовались другой программе, ему приходилось освобождать их. Очевидно, что плохо написанное приложение могло без труда остановить систему, не уступая контроль над ресурсами.

Вытесняющая многозадачность

В Windows 95, Windows 98, Windows NT и Windows 2000 применяется вытесняющая многозадачность. Здесь ресурсы контролирует операционная система, а не приложения — например, в Windows NT процессором и ресурсами управляет исполняющая подсистема. Именно этот сервис предоставляет доступ к ресурсам. А так как он полностью контролирует все оборудование, ни одно приложение не сможет захватить все процессорное время полностью, не уступая его другим программам, Поэтому разработчикам больше не надо проектировать приложения для работы в средах с кооперативной многозадачностью — всю работу взяла на себя операционная среда.

Многопоточность

В многопоточных приложениях процесс способен выполнять несколько задач сразу, Исполняющая подсистема Windows NT выделяет каждому потоку часть процессорного времени. Например, в электронных таблицах может быть функция пересчета значений. Благодаря многопоточности, такой пересчет, размещенный в одном потоке, не помешает вводу данных, реализованному в другом потоке. В противном случае пользователю пришлось бы ждать обновления значений. На рис. 8.6 изображены однопоточный и многопоточный процессы.

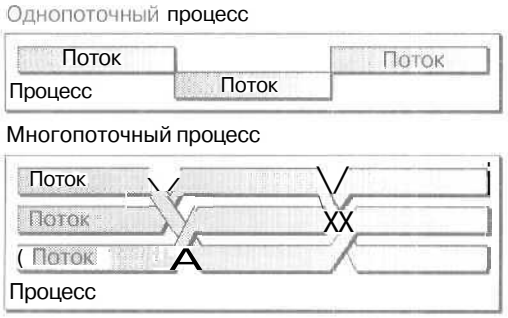


Рис. 8.6. Многопоточность в Windows NT

COM+ в Windows 2000

В предыдущих разделах мы описали технологии Microsoft для создания компонентных трехуровневых распределенных приложений. Вы узнали, что в стратегии Microsoft центральное место занимает программная модель, базирующаяся на COM и MTS. В сентябре 1997 года Microsoft объявила о будущем объединении этих технологий в одну под названием COM+. Первая версия COM+ появилась в Windows 2000, Мы кратко опишем функции COM+ 1.0 и ее роль в разработке распределенных приложений.

Внимание! Когда писалась эта глава, Windows 2000 и COM+ существовали только в бета-версиях; в окончательных выпусках некоторые из описываемых функций могли измениться.

В COM+ все объекты — и удаленные, и локальные — создаются одинаково. Эта технология основана на архитектуре активации и перехвата обшего назначения, которая позволяет перехватывать обращения к объекту и связывать с ним контекст. Во время активации COM+ с помощью атрибутов, хранящихся вместе с компонентом или в информации о конфигурации приложения, выявляет используемые объектом сервисы, отслеживает их и определяет, требуется ли для обращения к ним работа других сервисов. В случае положительного ответа нужные службы вызываются автоматически. Вместо использования отдельных оболочек контекста, как это делается в MTS, COM + объединяет перехваты в обычную архитектуру заместителей/представителей, традиционно применяемую для вызовов в пределах потоков, процессов и компьютеров.

Хотя эта архитектура носит общий характер и все объекты связаны с контекстом, для использования сервисов COM+ можно настроить только классы, реализованные в DLL-серверах. Классы, способные вызывать сервисы COM+, называют настроенными, все остальные - ненастроенными. Настроенные классы размещены в новых суррогатах COM+, использующих стандартную суррогатную архитектуру COM и обладающие свойствами суррогатов как из COM, так и из MTS.

Единая программная модель

Единая архитектура позволяет работать со всеми объектами одинаково, независимо от того, используют они сервисы COM+ или нет. Чтобы убедиться в передаче контекстной информации одним объектом другому, в MTS нужно вызвать метод **CreateInstance** контекста объекта. В COM+ для этого подойдет обычная функция API, создающая объект, например, **CoCreateInstanceEx** или **CreateObject** в Visual Basic. Таким образом, приведенный ниже код:

```
Dim ctxObject As ObjectContext
Dim subObject As ImyInterface
Set ctxObject = GetObjectContext()
Set subObject = ctxObject.CreateInstance("MyComponentProgID")
```

можно заменить таким;

```
Dim subObject As ImyInterface
Set subObject = CreateObject("MyComponentProgID")
```

Кроме того, COM+ поддерживает более общий механизм создания объектов, основанный на *моникерах* (monikers). *Моникер* ~ это COM-объект, реализующий стандарт именования объектов. Разработчик передает такому объекту символьное имя, а моникер находит объект. Все это существовало и в COM, но в COM+ появился стандартный моникер, определяющий способы создания новых моникеров. Такой метод очень полезен, так как может быть в любой момент расширен для поддержки новых функций. Причем это расширение достигается простой установкой нового компонента с последующим информированием разработчиков о совместимых именах. С другой стороны, улучшить системные функции, например. **CoCreateInstanceEx**, добавив к ним новые возможности, довольно сложно. Благодаря новому моникеру, объекты создают следующим образом:

```
Dim subObject As ImyInterface
Set subObject = GetObject("new:MyComponentProgID")
```

Разработчики трехуровневых приложений на основе MTS не заметят изменений в написании COM+-приложений — они коснулись только создания объектов. COM+ направлен на использование в многоуровневых приложениях, поэтому его программная модель практически совпадает с моделью MTS. Компоненты, реализующие сервисы COM+, должны удовлетворять тем же требованиям, что и для запуска в MTS. К тому же, все существующие компоненты **MTS** будут работать и в COM+.

Как и в MTS, в COM+ придается особое значение декларативной, основанной на атрибутах модели доступа к сервисам. Устанавливают эти атрибуты с помощью входящих в COM+ сервисов администрирования. Атрибуты и другие метаданные каждого класса хранятся в библиотеке компонента, дополняющей библиотеку типов. После регистрации компонента информация из библиотеки компонента кэшируется в регистрационной базе данных, используемой COM+ для определения способа создания объекта.

В библиотеку компонента можно включить не только информацию для регистрации, но и полностью описать компонент, сделав его самодокументированным. Данные таких библиотек используются сервисами COM+, регистрирующими компоненты; при этом отпадает потребность в создании кода, напрямую изменяющего элементы реестра в разделе HKEY_CLASSES_ROOT. Так как большая часть такого кода автоматически генерируется средствами разработки, описанный выше механизм никак не влияет на процесс разработки до появления инструментальных средств, поддерживающих **COM+.** Для совместимости с уже существующими компонентами COM+ будет поддерживать и старые саморегистрирующиеся компоненты.

Как уже говорилось, с каждым объектом COM + во время его создания связывается контекст. Кроме того, каждый объект COM+ содержит набор пар «атрибут — значение», описывающий среду, в которой этот объект выполняется. Точный набор таких атрибутов, сохраняемых в контексте, зависит от параметров класса. Контекст объекта применяется для определения момента вызова сервисов. Он используется COM+ в зависимости от сервисных атрибутов компонентов, классов, интерфейсов и обращающегося к нему объекта. Значениями атрибутов контекста объекта нельзя управлять во время периода выполнения, однако разработчики могут передать ему запрос и вызвать его методы для программного доступа к сервисам COM+.

Основные сервисы COM+

COM+ — это комбинация спецификаций и сервисов **COM.** Среди новых и переработанных сервисов:

- серверы;
- транзакции;
- защита;
- администрирование;
- распределение загрузки;
- очередь компонентов;
- события;
- базы данных в памяти.

Подробнее эти сервисы рассмотрены в практикуме 7.

Резюме

Основная цель COM — дать разработчикам возможность собирать приложения из уже готовых частей (компонентов) независимо от их местонахождения и языка программирования, на котором они реализованы. Модель COM определена так, что приложения и компоненты можно перерабатывать независимо друг от друга.

Модель программирования COM основана на объектах, интерфейсах, классах и компонентах. COM-объекты — это экземпляры COM-классов, которые, в свою очередь, являются именованными реализациями COM-интерфейсов. Интерфейс определяет набор взаимосвязанных методов и служит контрактом между реализующим его объектом и клиентом. Все COM-интерфейсы являются производными от **IUnknown**. Он управляет временем существования объекта и перемещением по интерфейсам. COM-компоненты — это двоичные файлы, на основе которых можно создавать COM-объекты. В состав компонента входят COM-классы, реализация объектов, необходимых для создания экземпляров класса, и код, записывающий в реестр информацию о местонахождении классов. Большинство современных компонентов, в том числе многие системные сервисы Windows, совместимы с автоматизацией и используют двойные интерфейсы. Такие компоненты доступны практически всем языкам разработки и средам программирования, включая языки сценариев. COM — основа трехуровневой архитектуры Windows DNA.

Закрепление материала

1. Что такое спецификация COM?
2. Каким образом COM работает на разных компьютерах?
3. Опишите механизм зашиты в MTS.
4. Что такое COM+?
5. Перечислите основные сервисы COM+ 1.0.

Практикум 7. Лекция об основах COM+

Материал для этого практикума взят из технического документа «Designing COM+ Applications», подготовленного отделом COM+ корпорации Microsoft.

— Дэн, как я рад, что мы встретились!

- Билл, почему-то мне кажется, что это не случайно.

- Да, ты прав, — угрюмо усмехнулся Билл. — В прошлый раз. когда мы встречались с группой по промышленной архитектуре, я сказал, что в приложении RMS не будет использоваться новая версия COM, появившаяся в Windows 2000. Тем не менее мы применяем массу технологий на основе COM+, и разработчики задают мне вопросы об этой новинке. Мы не уверены, что наш проект заработает в среде COM+.

- Билл, ты говоришь, как твои молодые программисты. Они постоянно повторяют, что новые технологии изменят мир, но он как-то умудряется остаться прежним. Почему ты считаешь, что COM+ так сильно повлияет на RMS?

— Думаю, моя группа с удовольствием поучаствовала бы в спасении мира, но для начала неплохо бы было спасти нашу компанию. Ты знаешь, как важна часть MSF, отвечающая за выпуск версий приложения? Мы должны быть уверены, что наш проект не только хорош сейчас, но останется таким и в будущем.

Дэн помедлил, поняв какое огромное впечатление произвела модель MSF на разработчиков, и сказал:

- Похоже, ты *сможешь* и старую собаку научить новым трюкам, Ты прав. Знаешь, в прошлом году я был на презентации COM+ и познакомился там с интересными людьми. Думаю, я смогу вам помочь.

Время учиться

Люди заносили стулья в конференц-зал. «Много сегодня народу, - подумал Дэн. — Тиму придется драться за каждый пончик».

В этот момент по комнате прошелестело: «Пончики, пончики...», и в комнату вошли Тим и Билл, груженые десятью большими коробками.

- Эй, босс, сегодня мы взяли их побольше! — крикнул Тим, а народ начал подходить к коробкам.

— Похоже, что пончиков не останется — многие заинтересовались COM+, — засмеялся Дэн.

- Когда ты прислал приглашение на видеоконференцию, я подумал, что стоит переслать его еще нескольким сотрудникам компании. Некоторые из нас уже читали книгу «Designing Component-Based Applications» и с удовольствием послушают ее автора.

На стене появилось огромное изображение Мэри Киртлэнд.

— Привет, Мэри, — сказал Дэн. — Уже все в сборе.

— Спасибо, Дэн. Рада тебя видеть. Пожалуй, я начну.

Создание приложений на базе COM+

- Здравствуйте, я — Мэри Киртлэнд, менеджер отдела COM+ компании Microsoft. Я расскажу вам о создании приложений на базе COM+.

Сначала немного — о некоторых прикладных сервисах, используемых в COM+ для создания приложений на базе архитектуры Windows DNA. Затем более подробно — о процессе разработки приложений на основе технологий COM+.

Начну с Windows DNA. Это — трехуровневая архитектура, на основе которой можно создавать распределенные приложения. Она обеспечивает нас каркасом, а уж думать о самой программе и использовании в ней технологий придется самим. COM+ поддерживается сервисами приложений операционной системы и применяется для создания прикладной логики программы. Пользовательский уровень и уровень данных пересекаются, но COM+ в основном затрагивает прикладной уровень.

Windows DNA — архитектура трехуровневых приложений, которая, как мы считаем, отлично подходит для создания масштабируемых распределенных приложений. Кроме того, она позволяет собирать приложения из компонентов.

Компоненты — это повторно используемые двоичные файлы, предоставляющие приложениям и другим компонентам различные сервисы. Это решение позволяет разрабатывать программы по частям и упрощает их расширение и модификацию. В данный момент мы не говорим о конкретных компонентах COM или COM+, а только о неких распределенных модулях. Со временем вы отберете требуемые вашему приложению сервисы COM и COM+.

Средства, поддерживающие Windows DNA, помогут вам создать такие компоненты, выполняя за вас некоторую часть работы. Они способны генерировать каркасы, описывающие инфраструктуру компонента. Помимо нее, вам понадобятся промежуточные сервисы, которые обеспечат поддержку транзакций и защиту. Они сэкономят вам время, выполняя вспомогательные, но необходимые задачи. И наконец, вряд ли ваша программа не будет ни с чем взаимодействовать. Windows DNA упростит процесс взаимодействия с данными и приложениями.

Основной прикладной сервис — COM+, среда выполнения COM-компонентов. Средствами COM+ вы сможете объявить сервисы или выяснить, какие из них доступны. Инфраструктуру, обеспечивающую работу сервисов COM+, разработала компания Microsoft. Основной элемент этой инфраструктуры — *координатор распределенных транзакций* (Distributed Transaction Coordinator, DTC), позволяющий объединять компоненты в функциональные модули. Он позволяет убедиться, что ресурсы, которые должны обновляться вместе, изменены либо все сразу, либо вообще не изменены (даже если модуль включает нескольких компонентов). Кроме того, существуют и средства администрирования, упрощающие развертывание и комплектацию многокомпонентных приложений.

Попросту говоря, COM+ — это новый этап эволюции хорошо всем знакомой технологии COM и сервера транзакций MTS (он поставляется в составе пакета Option Pack Windows NT). Microsoft объединила эти технологии в одну модель программирования COM+ и включила ее в состав Windows 2000. Большая часть моей лекции посвящена разработке приложений для MTS. Описанные мною принципы пригодятся вам, даже если ваша программа должна работать в Windows NT 4 или ваши клиенты не могут использовать новые функции Windows 2000.

Вместе с COM+ используются еще некоторые важные сервисы. Это — Internet Information Server и *активные серверные страницы* (Active Server Pages, ASP), применяемые для динамической генера-

ции HTML-страниц и их передачи интернет-клиентам. Компоненты инкапсулируют большую часть функций, выполняемых ASP-страницами, а для вызова компонентов применяются сценарии. Для реализации представительского уровня используется HTML — этот язык позволяет отображать информацию одинаково на всех платформах.

Еще один прикладной сервис — Microsoft Message Queue Server (MSMQ) — предназначен для передачи сообщения в определенную вами стандартную инфраструктуру и позволяет определять формат сообщения и объединять слабо связанные приложения. Еще одно преимущество MSMQ заключается в том, что сообщение доставляется только один раз, причем выбирается наименее затратный механизм маршрутизации. Кроме того, вы можете назначать сообщениям приоритеты. Как видите, технология MSMQ превосходит методы работы с сообщениями в DCOM.

В COM уже введено понятие объекта и интерфейса. Основное свойство COM таково: доступ к объекту осуществляется с помощью его интерфейса. Чтобы создать объект, к которому вы хотите обращаться, нужно использовать *фабрику классов* (class factory). Для доступа к интерфейсам применяется стандартный интерфейс **IUnknown**, на основе которого должны создаваться все объекты. Классы объединяются в двоичные модули, называемые COM-компонентами. В такие компоненты включена регистрационная информация, необходимая для их установки на компьютере. Они также содержат данные о методах упаковки объектов, необходимые для их загрузки и вызова операционной системой.

MTS может похвастаться новинками, в частности декларативным программированием и управлением состояния. Используя декларативное программирование, вы вправе работать с базовыми компонентами, обращаясь к ним посредством интерфейсов. Клиенты получают интерфейсные указатели и с их помощью взаимодействуют с объектами. Вы можете определить информацию о способах использования сервисов данного компонента на уровне самого компонента, класса, интерфейса, а в COM+ — и на уровне метода. Атрибутами бывают, например, простые запросы или права пользователя.

На основании этих атрибутов и некоторой информации о клиентском компьютере удается создать контекст, в котором будут работать объекты. По существу, контекст обеспечивает объектам безопасные условия работы. Кроме того, контекст предоставляет операционной среде и MTS данные о транзакциях и защите, а также некоторую дополнительную информацию, необходимую для корректного применения сервисов к объектам. Если вы создадите вложенный компонент или объект, вам придется разработать для них и контексты, изучив

атрибуты нового объекта и контекстную информацию всех существующих объектов. Полученные при этом атрибуты и контекстная информация объединены для создания контекста нового объекта.

Основная идея — сделать понятия атрибутов и контекста COM+ основой модели программирования. Вместо обращения к API за получением доступа к определенным сервисам, которые способны, в свою очередь, обращаться к другим API, конфигурирующим их, вы просто объявляете атрибут. Другими словами вы описываете действие, а способ его выполнения определяет система.

Другая давно обсуждаемая особенность MTS — управление состоянием. Мы часто говорим, что MTS больше годится для объектов без состояния, но это не совсем так. Вам надо учесть, где будет находиться ваш объект. Основная идея — отказаться от хранения состояния объектов при пересечении границ транзакции, чтобы не допустить утечки информации из одной транзакции в другую. Вы должны быть уверены, что данные останутся корректными и их целостность не нарушится.

Вам придется изучить различные типы информации о состоянии. Сохранять состояние разрешается в клиенте, а не внутри объектов. В этом случае клиент передиет информацию объекту или компоненту, вызвав соответствующий метод интерфейса. Можно сохранять состояние в каждом объекте, в виде его переменных-членов (например, в классах *C++* или Visual Basic). Но. повторю, нельзя предполагать, что состояние сохранится вне границ транзакции. Таким образом, все хорошо, пока время жизни объекта меньше, чем время жизни транзакции.

Если же вам нужно сохранить состояние вне границ транзакции, но вы не хотите получать его от клиента, воспользуйтесь общим промежуточным состоянием — оно годится и для хранения информации нескольких клиентов. За управление такими состояниями отвечает *диспетчер общих свойств* (Shared Property Manager). Проиллюстрирую его работу: представьте место на сервере, которое вы используете для хранения и контроля за информацией. Вам, скорее всего, придется передать от клиента ключ, запрашивающий состояние. Иначе ваши объекты будут сами искать его и запрашивать состояние из хранилища, которое для повышения гибкости и масштабируемости приложения может быть постоянным (например, базой данных). При этом вы сможете считывать состояние только тогда, когда требуется. Если же вам удастся повторно или совместно использовать соединения с таким постоянным хранилищем, масштабируемость еще увеличится.

Другая важная с точки и зрения разработчика компонентов особенность модели программирования заключается в том, что вы можете объявить о завершении работы с состоянием объекта. Для этого достаточно вызвать методы SetComplete и SetAbort после завершения

вашего метода. Такая процедура информирует MTS, что разрешено уничтожить объект и освободить занятые им ресурсы.

Как я уже говорила, COM+ — это просто новый этап в развитии COM и MTS, вернее, результат их объединения. Хочу еще раз подчеркнуть, что основные модели программирования COM+ и MTS совпадают. Пару лет назад мы считали, что COM+ изменит способ создания компонентов. Теперь же в Windows 2000 стало ясно, что COM+ играет заметно большую роль — это не просто метод создания компонентов, а новый путь развития программного обеспечения. На самом деле разработчики продолжают писать свои программы так же, как и раньше, когда MTS и COM существовали по отдельности. Даже с помощью COM+ вам придется создавать внутрипроцессные компоненты и открывать интерфейсы. Если вы захотите использовать автоматизацию, вам, как и прежде, придется реализовывать интерфейс IDispatch и писать регистрационный код. Все, что вы делали раньше, вам придется делать и в COM+ и Windows 2000.

Однако появились новые сервисы, благодаря которым расширился диапазон типов приложений, которые вы можете написать. Я кратко перечислю их, рассказав о способах их использования.

Улучшены и существующие сервисы MTS. Например, если вы не привыкли использовать транзакции и активацию по запросу или если они вам не нужны, вашу работу облегчат новые *сервисы транзакции COM+* (COM + Transaction Services). Microsoft разделила эти сервисы, и теперь вы можете выбрать именно то, что вам необходимо. Если вам не нравится активация по запросу, примените, например, ролевую защиту. Ее поддержка также улучшена, и теперь разработчик получил в распоряжение более подробную информацию о цепочке вызовов. В общем, с COM+ все стало проще, даже если вы продолжаете пользоваться сервисами MTS.

Примеры архитектур COM+-приложений

Примером архитектуры высокоуровневых приложений может служить простая программа с Win32-клиентом, взаимодействующая с прикладным уровнем и уровнем данных посредством COM+ Windows 2000. Такое приложение состоит из нескольких компонентов, работающих в среде COM+. Связь между его уровнями осуществляется посредством DCOM, что позволяет Win32-клиенту запрашивать и создавать объекты определенных типов. После получения интерфейсного указателя пользовательский и прикладной уровни получают право взаимодействовать напрямую. Бизнес-объекты могут, в свою очередь, использовать и другие объекты. В большинстве случаев они обращаются к объектам данных, которые инкапсулируют функции доступа к хранилищам информации. Для использования SQL Server

и взаимодействия с ним объектов данных разрешается применить, например, ADO. Существует множество технологий для работы с разными типами хранилищ. Однако ADO — самая простая и гибкая из них с точки зрения программирования. Кроме того, обратиться к этой высокоуровневой и понятной модели можно из сценариев и компонентов, написанных на разных языках программирования. Сейчас я рассажу о некоторых альтернативах ADO.

Последнее время все чаше встречаются Web-приложения, в которых в роли прикладного уровня, напрямую взаимодействующего с пользовательскими сервисами, выступает IIS, возможно, вместе с ASP-страницами. Во многих случаях такой пользовательский уровень создается средствами HTML или динамического HTML (DHTML); при этом для запуска программ применяются клиентские компоненты. Связь между Web-приложениями и I1S осуществляется не только посредством DCOM, но и с помощью HTTP. В этом случае для взаимодействия ASP-страниц и пользовательского уровня применяются обычные HTTP-запросы и ответы, а ASP-страницы вызывают бизнес-объекты. ASP-страницы интерпретируются, поэтому, если вы хотите выполнять сложные задачи, причем неоднократно и в разных средах, очень важно инкапсулировать код сценариев в виде компонентов. Как я уже упоминала, такие бизнес-объекты способны взаимодействовать с объектами данных, которые, в свою очередь, взаимодействуют с уровнем данных.

Другой интересный вариант — создание двух независимых приложений, явно друг с другом не связанных, но иногда взаимодействующих и совместно использующих информацию. Обычный пример — прием заказов и доставка товаров. Процесс приема заказа не должен быть связан с подтверждением доставки, временем ее выполнения, лицом, доставляющим товар и т. п. Чтобы не возникало таких ограничений, нужно применять передачу сообщений от одного приложения другому. В данном случае это может быть клиентский интерфейс, взаимодействующий с программой, которая ставит сообщение в очередь. Теперь за его передачу другому приложению отвечает MSMQ. Каждая программа вправе обращаться к своему хранилищу данных, а иногда даже к одному и тому же.

Новые возможности разработки

А сейчас я познакомлю вас с основными функциями, расширяющими возможности разработчиков: *компонентами в очереди* (Queued Components), *слабо связанными событиями* (Loosely Coupled Events. LCE), *базой данных в памяти* (In-Memory Database, IMDB), *диспетчером общих свойств* (Transactional Shared Property Manager), созданием пула объектов и динамическим распределением нагрузки.

Компоненты в очереди

Эта новинка COM+ представляет собой оболочку, скрывающую очереди сообщений в обычной модели программирования COM. Вам не придется явно писать код для постановки сообщения в очередь, его синтаксического разбора и последующей передачи соответствующему приложению, вы просто создаете «знающий» об очередях интерфейс компонента. Причем отправка сообщения и возврат ответа не зависят от времени. Однако, если приложение вызывает метод клиента, «знающего» об очереди, оно получит сообщение и сможет его обработать, Независимо от вида сообщения, любые действия выполняются только после его поступления. Таким образом реализовано асинхронное, или независимое от времени, поведение объектов в COM+,

Если же вы собираетесь продолжать работу во время длительных операций или уменьшить время отклика на запрос клиента, попытайтесь отделить эти функции от операций взаимодействия клиента с приложением в реальном времени. В подобных случаях мы советуем послать компоненту сообщение, позволить ему выполнение соответствующей работы, после чего, возможно, обновить в хранилище данных информацию о состоянии. В противном случае вы можете послать еше одно сообщение специализированному приложению, знающему, как завершить операцию, выполняемую вами вне очереди.

На этой стадии разработки нужно решить, будете ли вы изменять формат сообщений. Если нет, то к вашим услугам — все преимущества компонентой, использующих очередь, вам не придется изучать интерфейс MSMQ и вы сможете применить знакомую модель программирования COM. Если же вы захотите изменить формат сообщений, что иногда требуется в случае взаимодействия с внешним приложением, поддерживающим только свой формат, вам придется применять API MSMQ.

Слабо связанные **события**

Объединение независимых приложений выполняют и с помощью слабо связанных событий — публикационно-подписочной модели событий, разработанной Microsoft. Эта модель отлично работает в сочетании с компонентами в очереди. Типичная реализация предполагает, что приложение будет ограничено в своих действиях с момента возникновения события до момента отклика на сообщение. Если вы не хотите ждать, пока все подписчики обработают событие, доверьте эту работу компонентам в очереди. Для этого нужно запустить событие посредством сообщения, что позволит системе обработать его, когда появится возможность.

Одно из достоинств модели слабо связанных событий заключается в том, что в отличие от точек соединения ActiveX-элементов она допускает постоянные подписки, которые не обязательно реализовать в каждом компоненте. Поэтому для управления подпиской не нужно включать в состав компонентов дополнительные алгоритмы — все делает система.

Однако не забывайте об одном ограничении LCE — невозможности широковещания. Поэтому, если у события десятки тысяч подписчиков, вряд ли стоит использовать LCE. С другой стороны, если подписчиков немного, передача информации посредством LCE будет эффективной. Кроме того, разработаны средства распараллеливания, с помощью которого отделенные потоки посылаются событиям каждого подписчика быстро, как только возможно. Хотя такая процедура и не является истинным широковещанием, она позволяет передавать сообщения подписчикам, не дожидаясь завершения ее обработки всеми подписчиками. Повторю, это идеальный способ объединения слабо связанных элементов.

Основное назначение такого распараллеливания — мониторинг. Вы можете генерировать события в ответ на необычные происшествия, например, при нарушении доступа. Если какие-либо элементы требуют проверки, вероятно, придется их уничтожить — например, если ресурсы ограничены или возник сбой, вызвавший спад производительности системы. Таким образом вы можете разработать программу, отслеживающую определенные события, благодаря чему удастся разрешать возникающие проблемы. Сами компоненты не различают элементы, с которыми они работают. — они могут только сообщить вам о потенциальных проблемах.

База данных в памяти

Еще одна новинка COM+, отсутствующая в MTS, — база данных в памяти IMDB, представляющая собой кэш данных, упрощающий повторное получение информации из хранилища и, таким образом, приближающий ее к бизнес-объектам. База данных в основном предназначена для частого считывания редко изменяемых данных — например, из таблицы, содержащей список почтовых индексов, городов и стран. Вы получаете информацию из хранилища и передаете ее серверу приложений (среднему уровню, на котором выполняются прикладные алгоритмы), дальнейший же доступ к данным в памяти осуществляется гораздо быстрее. Л так как память недорога, появляется возможность снизить сетевой трафик. Часто считываемые элементы помещают в кэш и извлекают информацию прямо из памяти, не тратя время на постоянные запросы. IMDB оптимизирована для

режима считывания данных, так как в Windows 2000 нет средств поддержки распределенного кэша (поэтому на каждом компьютере заводится свой кэш, в который и поступают данные из хранилища). Кроме того, все изменения информации должны проходить с участием IMDB.

Стоит отметить, что вам не удастся обновить базу данных и распространить эти изменения на каждый кэш. Если вы собираетесь модифицировать данные, убедитесь, что данный компьютер сможет обработать внесенные изменения. В противном случае придется самостоятельно разрабатывать механизм обновления кэшированной информации независимо от числа копий, так как в первые версии IMDB он не включен. Однако вы вправе использовать эту технологию в качестве кэша сквозной записи. Если вы разделили данные и клиентов так, что все изменения происходят только на одном компьютере, пропустите информацию через IMDB и передайте ее в хранилище позднее.

Диспетчер общих свойств

Диспетчер общих свойств MTS управляет общим промежуточным состоянием, которое иногда нужно сохранять при выходе за границы транзакции или разделить между компонентами/клиентами. Одна из новинок COM+ — транзакционный диспетчер общих свойств, представляющий собой надстройку над IMDB и использующий для доступа к ее функциям привычные интерфейсы диспетчера обших свойств. Теперь у вас есть выбор — использовать IMDB с помощью OLE DB или ADO. К тому же для доступа к информации о состоянии разрешается применять объектный подход посредством интерфейсов диспетчера общих свойств. Преимущество диспетчера обших свойств COM+ в том, что он охватывает весь компьютер, а не каждый процесс в отдельности. Поэтому в COM+ несколько приложений могут совместно обращаться к одной информации, а в MTS вся информация относится к одному процессу или серверному пакету,

Напомню, что диспетчер свойств помогает управлять данными, которые нужно хранить вне рамок транзакции, но не обязательно в постоянном хранилище. Транзакционный диспетчер общих свойств идеально подходит для постоянно корректируемых данных, например, счетчиков на Web-страницах или часто создаваемых идентификаторов. В противном случае эти элементы требовали бы обновления базы данных при каждом их изменении, а это приводит к спаду производительности. Транзакционный диспетчер общих свойств позволяет кэшировать данные в памяти компьютера, производя их обновления и записывая их в базу данных по требованию, так что вам не придется изменять базу данных при каждом обращении к ней.

Создание пула объектов

Когда-то мы хотели включить в MTS средства эмуляции пула объектов. Идея заключалась в создании объектов до того, как они потребуются приложениям, и сохранении их в резерве. Когда они понадобятся, MTS быстро бы получил их из пула, не заставляя пользователей создавать и инициализировать новые объекты. К сожалению, пул не был реализован. После деактивации объекта MTS уничтожает его, и при следующем обращении приходится создавать новый объект.

В COM+ такие пулы поддерживаются, поэтому приложения могут создавать пулы однотипных объектов для каждого процесса. При запросе объекта или вызове метода деактивированного объекта сначала изучается пул. Если же при таком использовании объекта возникают проблемы, приложение удалит его из пула.

Если у вас есть объекты, создание которых сопряжено с большими издержками, их можно создать заранее в фоновом режиме, ускорив таким образом, доступ к ним. Пулы позволяют контролировать максимальное число существующих объектов. Иначе вероятна ситуация, что, когда 10 000 клиентов запросят какой-либо объект, COM+ попытается создать 10 000 объектов независимо от возможностей системы. Определение максимального количества объектов позволит вам управлять использованием ресурсов сервера.

Однако имейте в виду, что у пулов есть и недостатки — в них нельзя поместить объекты, связанные с потоками, то есть почти 90% всех создаваемых объектов. Например, таким свойством обладают все компоненты Visual Basic. Поэтому, если вы применяете Visual Basic версий 5.0 или 6.0, на поддержку пула объектов можете не рассчитывать. Надеюсь, что в следующих версиях COM+ это ограничение будет снято, и компоненты, поддерживающие создание пулов, можно будет написать на любом языке программирования. Пока же разрешается использовать Visual C++ и Visual J++ версии 6.0.

Реализуя пулы объектов, не забывайте три важных условия. Во-первых, если вы знакомы с моделями потоков, для доступа к объектам, поддерживающим пулы, применяйте модель свободных потоков, нейтральную модель или обе сразу. Во-вторых, бездумное использование пулов объектов не принесет пользы. И в-третьих, пулы объектов не гарантируют оптимальной производительности приложений.

Учитывая эти условия, вы должны изучить производительность приложения и оценить выгоды от применения пула объектов. Создайте простую версию вашего компонента, особенно если программируете на Visual Basic, не поддерживающем пулы объектов, и с ее помощью выясните, достаточно ли он производителен. Если результат такого исследования будет отрицательным, попробуйте применить дру-

гой язык программирования и только после этих попыток обращайтесь к пулам.

Пулы объектов, скорее всего, помогут там, где затраты на создание объектов (а также на инициализацию компонентов) велики или для этих действий не хватает ресурсов.

Динамическое **распределение** нагрузки

При увеличении числа пользователей MTS требуются дополнительные серверы, При этом копии приложения или пакета выполняются на отдельных компьютерах, а нагрузка распределяется статически. В этом случае определенные клиенты всегда работают на определенном компьютере, либо же система распределяет запросы по разным компьютерам. Но когда число пользователей возрастает, сохранять конфигурацию системы становится сложно. К тому же MTS нечувствителен к некоторым проблемам, например, к отказу отдельного компьютера. Поэтому, если клиент связан только с одним компьютером и на нем произошел сбой, то работа клиента остановится. Причем, к сожалению, не предусмотрены средства для нахождения другого компьютера, на котором он смог бы закончить работу.

В COM+ реализовано динамическое распределение загрузки: вы определяете группу компьютеров, на которых установлен один и тот же набор компонентов, и разрешаете клиенту доступ к любому компьютеру из этой группы. Управляет таким доступом специальная программа, запушенная на маршрутизаторе. Ее алгоритм основан на времени отклика. Она в фоновом режиме собирает статистику и в зависимости от нее выбирает компьютер, производительность которого достаточна для создания объекта.

Распределение нагрузки выполняется не при каждой активации или деактивации объекта. Это решение имеет свои достоинства, ведь распределение нагрузки при активации компонентов без состояния связана с высокими накладными расходами. Если вы распределяете нагрузку только при создании объекта, клиент сможет обращаться к нему напрямую. Поэтому вызванный метод работает на том же самом компьютере: активирует на нем объект, выполняет необходимую работу и завершается. В результате эффективность операций оказывается выше, чем в случае распределения нагрузки при каждом вызове метода.

Основные причины, по которым нагрузка распределяется по этому алгоритму, таковы: во-первых, повышается масштабируемость приложения, во-вторых, появляются дополнительные возможности при выходе отдельных компьютеров из строя. Теперь, если один компьютер выйдет из строя, его заменит другой, входящий в кластер COM+. Большинство написанных вами COM+-компонентов поддер-

живают распределение нагрузки. Единственное, за чем вы должны следить, — неявная привязка к определенному компьютеру. То есть вам придется контролировать зависимость от определенных путей к файлам, названий серверов или информации о состоянии. Вы должны убедиться, что перенесенное на другой компьютер приложение все же будет работать даже при отсутствии информации о состоянии.

Ну вот, вы и узнали немного о сервисах приложений и новых функциях COM+. В заключение я расскажу о вопросах, возникающих при проектировании приложений.

Особенности проектирования приложений

При проектировании приложения первым делом определите бизнес-проблемы, которые вы пытаетесь решить. После этого изучите функциональные требования. Например, схемы и сценарии использования помогут вам прояснить требования пользователей к приложению.

Не забудьте выяснить бизнес-цели и нефункциональные требования. Бизнес-цели — это главные приоритеты, на которых вам надо сконцентрироваться. Обязательно учтите увеличение числа клиентов, так как система должна справляться с возрастающим числом обращений к ней. Определив бизнес-цели, вы сможете оценить необходимую производительность и решить, как удовлетворить требования заказчиков. Есть и еще один подводный камень — финансовые ограничения — например, клиенты, не обладающие большими ресурсами, возможно, захотят использовать существующие компьютеры или, согласившись заменить их, ограничатся самыми дешевыми из современных моделей, Иногда возникают конфликты между платформами клиента и сервера — например, если сервер работает под управлением Windows 2000, а на клиентских компьютерах установлены Windows 98 и Internet Explorer 5.0. Кроме того, на проект приложения сильно влияет и тип обозревателя, выбранного заказчиком.

Основное нефункциональное требование — производительность. Часто разработчики все внимание уделяют скорости работы приложения, завышая требования к производительности. В действительности же лучше изучить время отклика, ожидаемое пользователем. Кроме того, не забудьте оценить необходимое количество пользователей и требуемое приложению время системы. Такие данные должны выражаться в определенных цифрах, тогда вам удастся точно протестировать производительность системы. Таким образом, вы соберете прототип приложения, проведете тесты, выявив «узкие» места, после чего сможете оптимизировать его в соответствии с требованиями к производительности, что, естественно, лучше бездумной оптимизации всех параметров подряд.

Еще одно ограничение касается развертывания. Вряд ли вы будете создавать программу «с нуля». Как правило, уже существуют базы данных и другие приложения, с которыми вам придется взаимодействовать. Причем они работают на своих компьютерах, связываясь с ними определенными способами. В большинстве случаев изменить их нельзя.

Ограничения, накладываемые топологией, также создают трудности при разработке приложений. Какой тип сети используется? Вам вряд ли разрешат изменить сетевые соединения, скорее наоборот — вы должны подгонять свое приложение под топологию сети.

Еще одно нефункциональное требование — безопасность, в том числе защищенный доступ не только к приложениям, но и к данным. Например, существуют ли типы данных или какие-нибудь записи базы данных, к которым никто не должен обращаться? Если да, значит, некоторые учетные записи пользователей и подробная информация по ним должна быть доступна только управляющему ими менеджеру. Если же вы управляете другими учетными записями, то все, что вы получите, — только сокращенная информация о пользователе, например, название учетной записи и имя ее владельца. Но о своей учетной записи вы можете узнать все. Допустимы и другие ограничения — запретить определенным людям изменять информацию, но разрешить ее читать.

Есть и еще один вопрос, на который вам придется искать ответ: нужно ли защищать данные каким-либо способом, например, шифровать их? Есть ли ограничения на передачу информации в открытом виде? Думаю, всем ясно, что номера кредитных карт не стоит передавать открыто — эти данные лучше зашифровать. Система способна выполнить эту операцию на низком уровне (например, используя пакетное шифрование RPC), но такой способ сопряжен со значительными издержками, и его трудно встроить в модель программирования COM+.

На уровне приложения можно задать, что в данном интерфейсе вот этот конкретный метод будет зашифрован, и для его расшифровки нужно выполнить такие-то действия. В некоторых случаях предпочтительна именно защита данных на уровне приложения, где ее можно определить как часть контракта между клиентом и компонентом. Это гораздо эффективнее, чем без разбора шифровать все данные, передаваемые по сети. Кроме того, вы должны подумать, как разделить функции приложения на транзакции и потоки. Существует ли жесткая последовательность выполнения операций? Что вызывает завершение операции и начинает новую? Должны ли какие-либо опе-

рации выполняться совместно? Например, если я снимаю или перевожу деньги со своего счета в банке, я хочу быть уверена, что деньги будут сняты с моего счета и переведены на счет получателя. Мне бы не хотелось, чтобы мои деньги были сняты и никуда не переведены. Если они не попали к получателю, то пусть они остаются на моем счете.

Важно представлять себе информацию в виде блоков, даже когда вы рассматриваете ее на высоком или абстрактном уровне. Почему? Потому что вы не добьетесь максимальной гибкости приложения при переходе к определению ограничений на уровне методов или их вызовов. Именно поэтому надо изучить требования к транзакциям для каждого метода, а также подумать о том, как они связаны друг с другом (что особенно важно при их распределении по интерфейсам), какие интерфейсы входят в определенный компонент и какие компоненты формируют пакет. Хотя эти требования и не связаны со всеми сценариями напрямую, они также накладывают ограничения на ваш проект.

Изучив функциональные и нефункциональные требования, приступайте к моделированию. Теперь пора изучить логические объекты, которые входят в состав системы, определить, подходят ли они друг другу, какие из них можно сохранять и какие сервисы потребуются для создания вашего приложения. Разработав основные концепции, «очищайте» их до тех пор, пока не определите компоненты, которые нужно реализовать.

Я знаю, что многие из вас применяют объектные технологии, в том числе объектно-ориентированный анализ и проектирование, для создания логических моделей. Хотя эти методы и хороши, но все же они не всегда вписываются в процесс разработки компонентов. Обычно, когда вы создаете объектные логические модели, вы уделяете особое внимание концепциям и элементам вашей системы, определяете, как они работают.

Когда же вы изучаете вашу модель с точки зрения проектирования компонентов, анализ в основном ориентирован на сервисы. Не всегда удается решить, какими сервисами должны обладать объекты. Ведь несколько объектов могут работать вместе, выполняя какую-либо операцию. Поэтому с позиций чистого объектно-ориентированного проектирования нельзя определить способы объединения объектов в компоненты. Другими словами, не следует всегда применять модели, основанные на объектах, так как вам придется изменять их при сборке COM-компонентов.

Объекты в среде COM+

В среде COM+ существуют объекты двух типов: бизнес-объекты и объекты данных.

Бизнес-объекты инкапсулируют реально существующие бизнес-операции. Важно, чтобы эти операции были собраны в одном месте. Например, некоторые объекты могут инкапсулировать какое-либо бизнес-правило. Если бы оно размещалось в разных местах, то при его изменении обновить систему было бы очень сложно. Остальные объекты могут выполнять функции контроля — отвечать за последовательность правил и гарантировать, чтобы при применении определенного правила происходило обращение к объекту. Часто только контрольные объекты непосредственно взаимодействуют с пользовательским уровнем.

Но вот вы начали работу нал бизнес-объектом. Достаточно трудно угодить всем пользователям, поэтому изучите только реальных клиентов — людей, которые будут работать с вашим объектом уже сегодня, — и реализуйте необходимые им функции. Обдумывая набор функций объекта, не старайтесь угодить потенциальным клиентам, потребности которых немного отличаются. Создавайте одну версию объекта для ваших теперешних клиентов, а когда появятся новые, вы добавите новый интерфейс или даже создадите новый объект, удовлетворяющий их требованиям.

Кроме того, четко разделяйте методы так, чтобы они выполняли одну и только одну операцию, но делали это очень хорошо. Такие методы и объекты, выполняющие набор связанных действий, намного проще использовать повторно.

Не забудьте тщательно продумать способ передачи информации о состоянии между объектами. Спросите себя: «Как объект получит информацию? Данные будут храниться на сервере или передаваться клиентом? Откуда я буду их получать: из хранилища информации или от пользовательского уровня?» От того, как мы общаемся с бизнес-объектом, зависит и внешний вид интерфейса. Если информация сохраняется клиентом, вам, скорее всего не понадобятся громоздкие функции, передающие данные от клиентского уровня бизнес-объекту, который получит данные, обработает их и «забудет». Если же данные находятся в хранилище, клиент передаст бизнес-объекту определенную ключевую информацию, на основании которой тот получит данные из хранилища или промежуточной общей области. Обработав их, он вернет результат обращавшемуся к нему объекту.

При наличии нескольких высокоуровневых бизнес-объектов приходится хранить большое количество информации о состоянии каждого объекта. Обычно это делается для внутренних бизнес-объектов, недоступных пользовательскому уровню напрямую и используемых только в контексте транзакции. Вы можете, например, применить

такой метод при создании главной записи из нескольких вспомогательных — допустим, счета, в котором указано несколько товаров. После обработки вы захотите его сохранить с помощью объекта данных. Ну что ж, создайте счет, и все будут по очереди добавлять в него товары. Пока все это происходит в контексте одной транзакции, дела идут великолепно, и вы можете сохранять информацию о состоянии в каком-нибудь объекте. Если же вы ожидаете, что пользователь будет передавать информацию об отдельных товарах, вы не захотите сохранять ее в состоянии объекта. Ведь тогда этот бизнес-объект должен быть активным и занимать ресурсы сервера все время до тех пор, пока пользователь не решит, что закончил работу. Кроме того, пользователь может заняться другим делом или вообще пойти обедать, а бизнес-объект никуда не денется, будет ждать окончания работы. Проще сказать: «Вот вся информация, обрабатывай ее». Это типичное решение при переходе от пользовательского уровня к прикладному.

Также вам придется поразмыслить над взаимодействием методов или сервисов. Какие из них используют общие ресурсы? Как разделяются обязанности? Какие методы при совместной работе образуют транзакцию? (Ответ — объединенные в одну высокоуровневую операцию.) При необходимости вы должны будете поместить в один бизнес-объект методы, совместно использующие ресурсы, а методы, разделяющие одни и те же требования к транзакциям, поместить в один пакет или настроить их так, что, реализуя высокоуровневое поведение, они смогут работать вместе.

Не стоит забывать и о требованиях к защите. В MTS это труднее, чем в COM+ в Windows 2000, так как ее нельзя применять к определенному методу. В этой среде вы можете назначить пользователям разные права доступа к компонентам и интерфейсам, но не вправе ограничить доступ к методам. Такую возможность предоставляет COM+, но, несмотря на это, вам, вероятно, придется сгруппировать функции с похожими требованиями к безопасности. Вам следует рассмотреть и вопросы защиты данных. Можете ли вы передавать информацию в виде обычного текста или требуется шифрование? От ответа на этот вопрос зависит интерфейс бизнес-объектов, особенно тех из них, что общаются с внешним миром.

Теперь я расскажу об объектах данных. Они отвечают за точность и согласованность данных определенного типа. Если что-то случится с информацией этого типа, вы сможете выявить источник проблемы. Если же вы распространите ответственность за данные на несколько объектов и будете считать, что с ними все в порядке, то можете серьезно испортить хранилище информации. А несогласованные данные,

разбросанные по разным местам, очень трудно исправить. Таким образом, объекты данных — последняя линия обороны, ответственная за сохранение целостности информации. Объекты данных способны проверять согласованность данных средствами системных сервисов (например, с помощью правил проверки целостности в базе данных SQL Server). Но лучше сосредоточить ответственность за правильность данных и не рассеивать ее по бизнес-объектам.

Что же следует помнить об объектах данных? Во-первых, существует проблема хранения состояния. Дискуссия по этому вопросу об объектах с состоянием и без него только запутывает людей, выбирающих между MTS и COM+. Обычно объект данных существует только в операции или в памяти, естественно, во время существования транзакции. Запомните одно правило: и в MTS, и в COM + разные транзакции не могут совместно использовать состояние (в рамках отдельной транзакции это разрешается). Поэтому, вам, вероятно, понадобятся объекты данных с состоянием, определенные свойства которых вы вправе установить и к которым вы сможете обращаться посредством составных данных. Метод сохранения позволит вам показать, что пришло время записать информацию в хранилище. Либо у вас есть выбор — применить то же решение, что и в случае бизнес-объектов: передать всю информацию объекту, проверить ее целостность и переслать в хранилище. По завершении транзакции объект станет ненужным. Все это я сказала, чтобы вы лучше поняли важность выбора между сохранением информации в пределах транзакции в промежутке между вызовами методов и передачей всей информации сразу для каждого метода.

Типичное поведение объектов данных — создание, считывание, обновление и удаление. Если ваш объект должен быть защищен, вам придется решать, помещать ли операции обновления, создания и удаления в тот же интерфейс или в тот же компонент, что и операцию чтения. Требования к транзакциям у функций, изменяющих базу данных, иногда отличаются от требований к функциям, выполняющим запросы. И в MTS, и в COM+ вы вправе выбрать уровень транзакции только на уровне компонента. Следовательно, если все операции помешены в один компонент, их поведение определяется уровнем транзакции компонента. В таком случае вам, вероятно, понадобится включить поддержку транзакций и удостовериться, что объекты, которым передаются запросы, не входят в состав других транзакций. После этого можно запрашивать объекты, не опасаясь издержек, связанных с завершением транзакций. Объекты, выполняющие обновление, надо включить в ту же транзакцию, что и объекты данных. При огра-

ничении доступа полезно разделить различные элементы — таким образом вы разграничите полномочия пользователей (например, разрешите им выполнять запросы, но запретите обновлять данные).

Как и в случае с бизнес-объектами, вы должны выявить сервисы, использующие сходные ресурсы. Например, сервисы, обращающиеся к одной и той же или похожим базам данных, лучше поместить в один объект данных. Следует убедиться, что архитектура объектов данных допускает совместное использование соединений с базой данных: они требуют больших системных ресурсов и их число ограничено. Именно поэтому так важен совместный доступ к информации. Как я уже говорила, при проектировании объектов данных основным принципом является транзакционное поведение.

Транзакции требуют больших затрат, поэтому, если они вам не нужны — скажем, вы обновляете только одну таблицу, рассчитывая на выполнение этой работы хранилищем данных, или вообще ничего не изменяете, — от транзакций лучше отказаться. С другой стороны, если вы обновляете несколько таблиц или не уверены, что компонент обрабатывает запросы правильно, вам, вероятно, потребуются транзакции. Помните: мы не избавили вас от этого этапа проектирования, мы всего лишь упростили его реализацию.

Но и это еще не все. Вам надо решить, кому разрешено работать с объектами данных. Тогда вы сможете разделить операции на методы, применив к ним ролевую защиту. Для усиления защиты можно распределить их и по интерфейсам в зависимости от поддерживаемых платформ и обрабатываемых ошибок, Но при этом следует помнить, что из сценария доступ возможен только к стандартному интерфейсу **IDispatch.** Поэтому после распределения поведения объекта по нескольким интерфейсам клиенты не смогут обратиться к некоторым методам. Это может быть как плохо, так и хорошо. Обязательно удостоверьтесь, что сервисы, которые должны быть доступны всем, помещены в основной интерфейс. В COM+ можно выборочно назначать параметры защиты для определенных методов, даже входящих в диспетчерские интерфейсы таким образом, что обращение к ним будет вызывать ошибку. Однако скрыть сам факт существования метода вы не в состоянии. Если же вы все-таки хотите спрятать определенный метод, распределите его по разным интерфейсам или даже по разным компонентам.

Обязательно решите, кто будет применять объекты данных и будут ли к ним обращаться напрямую из пользовательского уровня. Будут ли они задействованы бизнес-объектами, которым вы доверяете? Можете ли вы положиться на поведение данного объекта? Если нет, всю ответственность за информацию надо возложить на объект

данных. Кроме того, вам необходимо разобраться, на каком языке писать клиенты. Если вы собираетесь применяться сценарии, которые смогут обращаться только к интерфейсу IDispatch, все функции должны быть доступны с помощью этого интерфейса.

Добавлю несколько слов о пользовательском уровне. В нем обычно содержатся ActiveX-элементы или компоненты автоматизации в зависимости от того, происходит ли отображение данных или они обрабатываются в фоновом режиме. Выполняются они, как правило, внутри процесса, поэтому их можно запустить в среде обозревателя или в контексте Win32-приложения.

При создании приложений, использующих компоненты на стороне клиента, следует изучить способы их установки через Интернет. С этим связано немало трудностей. Компоненты, работающие в среде обозревателя, должны быть безопасными при шифровании и инициализации. В противном случае некоторые клиенты не смогут получить созданные в обозревателе объекты — их защита настроена так, что создание объектов на странице запрещено. Пометив объекты как безопасные, вы повысите вероятность нормальной работы компонентов.

Другая проблема связана с тем, что к объектам пользовательского уровня обращаются клиенты-сценарии, а, следовательно, доступ к ним осуществляется с помощью интерфейса **IDispatch.** Конечно, очень хорошо, что вы распределили объекты по нескольким интерфейсам с компактными определениями типов. Но именно поэтому вам придется разрабатывать механизм передачи информации, необходимой сценариям, обращающимся только к **IDispatch**. Последнее время среди разработчиков COM-компонентов ведутся ожесточенные споры относительно двойных интерфейсов, которые обладают как свойствами **IDispatch,** так и собственными. Если у вас только один программный интерфейс, вы можете сделать его двойным, благодаря чему клиенты с поздним связыванием получат быстрейший доступ к интерфейсу. В то же время сценарии будут обращаться к нему посредством IDispatch. Рассмотрим и другие случаи. Если у вас только **IDispatch,** все операции сильно замедлятся, а если лишь разработанный вами интерфейс, то невозможен доступ из сценариев. Если вы создали несколько интерфейсов, все сильно усложняется. Ведь даже если все они двойные, сценарий видит единственный интерфейс — диспетчерский. Таким образом, вывод очевиден: стандартный интерфейс должен быть либо двойным, либо **IDispatch** (его смогут использовать сценарии),

Также вам придется решить, на методах или на свойствах будут основаны ваши интерфейсы. Интерфейсы на базе свойств отлично подходят для работы на пользовательском уровне, ведь они выполняются внутри процесса. В случае подключения по сети лучше приме-

нять решение, основанное на методах — это позволит сократить число двусторонних перемещений объектов. Например, если вы задали цвет, размер и другие атрибуты объекта, доступ к этим свойствам будет осуществляться с помощью интерфейса сценариев, основанного на IDispatch.

Объединение компонентов

Разработчики чаще всего обсуждают способы проектирования компонентов, но не следует забывать и об их объединении. Поэтому сейчас я расскажу о соединении физических и логических проектов вашего приложения.

Сначала поговорим о доступе к данным. В этой области нужно применять основные принципы инкапсуляции и абстрагирования. скрывая, таким образом, некоторые элементы уровня данных — местонахождение информации, физическую структуру БД и ее схему - от бизнес-объектов. Постройте с помощью объектов данных ясную и логичную модель, в которой бизнес-объекты будут работать, не зная, какова физическая схема данных, где находится информация и какой способ доступа к ней лучше — пусть на эти вопросы отвечают объекты данных.

Как же реализовать доступ к информации на сервисном уровне доступа к данным? Сначала нужно выбрать подходящую технологию, а их много — DAO, RDO, ODBC, OLE DB, ADO. Кажется, возьмешь любую букву, добавишь к ней DA и получишь сокращение, обозначающее какую-нибудь технологию доступа к данным. Так как же определить подходящую? Посмотрим на них с точки зрения простоты разработки, что очень важно, если необходимо выполнить работу в сжатые сроки или разработчики практически не знакомы с данной областью деятельности. Возможно, вам понравится технология, поддержка инструментальных средств которой лучше, чем у остальных, или с которой проще работать, а вы как раз хотите облегчить создание компонентов определенного типа. С другой стороны, такая простота, как правило, не в ладах с высокой производительностью — написать программу легко, но будет ли она эффективна? Поэтому стоит проверить выбранную технологию и, если ее производительность вас не удовлетворит, повнимательнее присмотреться к другим.

К тому же надо учитывать и планы на будущее. Сможете ли вы использовать компонент повторно? А обновить его в случае возникновения проблем? Естественно, нужно учесть и другие особенности работы в вашей компании, Главное — не забыть о нефункциональных требованиях и бизнес-целях, а также изучить соответствие технологии производственной архитектуре и стандартам разработки.

Еще один важный вопрос — отличие доступа к данным с помощью API по сравнению с другими способами, например, OLE DB или ADO. Они коренным образом отличаются по простоте разработки, по стандартам поддержки инструментальными средствами и по гибкости. Если вы использовали API и вам потребовалось изменить основное хранилище информации, вам придется переделать и все объекты данных. В случае OLE DB или ADO достаточно перенастроить некоторые параметры.

Возможно, вас интересует сравнение динамических операторов SQL и хранимых процедур. Последние более производительны, но динамическое составление запросов SQL только прибавит вашему приложению гибкости. Кроме того, хранимые процедуры привязаны к определенной базе данных, поэтому, если существует вероятность изменения хранилища информации или вам нужно несколько таких хранилищ, лучше выбрать динамический SQL.

В COM+ нужно учитывать, будете ли вы обращаться к данным напрямую или с помощью IMDB. Собираетесь ли вы в основном считывать информацию? Есть ли у вас хранилище, которое надо обновлять? Будет ли использоваться интерактивная работа *с* данными?

Не забудьте о взаимодействии с внешними пользователями вашей системы — как людьми, так и программами. Протокол, применяемый для связи с ними, зависит от пользовательского интерфейса. Если информация передается средствами обозревателя, то можно задействовать HTTP, так как в ответ на HTTP-запросы возвращаются HTML-страницы, для отображения которых, собственно, и предназначены обозреватели. Если же вы собираетесь взаимодействовать с Win32-приложением, лучше применить DCOM, предназначенный для взаимодействия приложений и компонентов прикладного уровня.

Стоит изучить и вопросы, касающиеся брандмауэров и защиты в целом. DCOM отлично подходит для связи по протоколу TCP, но для этого необходимы доверительные отношения между объектом, осуществляющим вызов, и вызываемым объектом. Кроме того, DCOM не очень хорошо работает через брандмауэр. В такой ситуации лучше пользоваться HTTP — он позволяет «пройти» через брандмауэр и подключиться к серверу, после чего обработать объекты данных на нем.

Некоторые ограничения накладывает операционная система. Например, не все протоколы годятся для связи с мэйнфреймом IBM. Позаботьтесь и о местонахождении взаимодействующих объектов. Близко ли они расположены друг от друга? Насколько быстр соединяющий их канал? Может, это медленная телефонная линия? Или все-таки высокоскоростное соединение? Ответив на эти вопросы, вы сможете выбрать подходящий протокол.

Я уже говорила о HTTP и DCOM, но никто не запрещает вам применять и *сервисы удаленного доступа к данным* (Remote Data Services, RDS), обеспечивающие доступ к COM-компонентам как по HTTP, так и с помощью DCOM. В таком случае вам не придется заранее выбирать протокол доступа к бизнес-объектам и данным.

Кроме того, определите способы передачи сообщений — с помощью MSMQ или посредством комбинации MSMQ и продуктов MQ Series компании IBM — или же рассмотрите какие-то другие технологии, например, низкоуровневые протоколы RPC или сокеты (однако в этом случае не забудьте просчитать, уложитесь ли вы в график работ).

Я уже говорила, что производительность приложения улучшится, если вы разделите независимые функции, а не объедините их в одну большую операцию. Да, COM — синхронная модель: вызвав метод, вам приходится ждать окончания его работы. Все в ней тесно связано в том смысле, что клиент узнает, к какому интерфейсу обращается, переходит к данному объекту и затем получает отклик от метода. Но, допустим, вашему приложению такая синхронность не нужна, Вы хотите просто сказать объекту: «Вот информация, займись ею в свободное время». В этом случае присмотритесь к MSMQ или компонентам в очереди (выбор между ними определяется необходимостью сообщений нестандартного формата),

Если ваше приложение предназначено для хранения информации и предоставления ее пользователям, подумайте об использовании событий или рассмотрите возможность публикации формата сообщений. Иногда полезно объединить эти методы, используя поставленные в очередь сообщения вместе с LCE.

При разработке независимых частей приложения нужно продумать способы передачи информации о состоянии. Если операция выполняется долго, как вызвавший ее объект узнает о ее завершении? Можно отослать ему асинхронное сообщение или же воспользоваться статусной информацией из таблицы данных. Кроме того, не забудьте обдумать способы восстановления частично завершенных операций. Например, вы заказали товар и получили сообщение, подтверждающее поступление заказа, Естественно, вы считаете, что со временем он будет выполнен. Но что произойдет, если доставка заказа в данный момент невозможна? Тут-то и должен сработать восстанавливающий механизм: «Доставка товара задерживается. Если заказанный продукт вам больше не требуется, сообщите нам, и мы аннулируем ваш заказ». За такое восстановление частично выполненных операций должны отвечать некоторые бизнес-алгоритмы.

Позаботьтесь и о защите. Вы собираетесь устанавливать связь между этими узлами? А надо ли защищать передаваемую между ними информацию?

В какой-то момент вы с удовлетворением констатируете, что разложили все по полочкам: здесь — нужные нам компоненты, там — необходимые сервисы. Но как распределять их по компонентам? Сначала отберите классы, реализующие определенные интерфейсы. Затем сгруппируйте их в компоненты, которые представляют собой физические пакеты, составляющие приложение. Чаще всего в одном компоненте у вас находится только один класс, хотя иногда в него можно поместить несколько совместно работающих классов. И наконец, объедините компоненты в приложения и процессы.

На этом этапе работы нужно изучить доверительные отношения и подумать о развертывании. Во время работы программы выполняются проверки безопасности, но при этом все внутренние модули приложения являются доверенными. Приложение развертывается как единое целое, поэтому установка двух разных компонентов на двух разных компьютерах возможна только в случае, когда они входят в состав разных COM+-приложений. Следовательно, если вам нужна проверка безопасности, вы должны убедиться, что вызов компонента вышел за границы приложения. К тому же есть и дополнительные сложности — каждый класс на каждом компьютере может входить только в одно приложение. Таким образом, можно выделить два типа программ: выполняющиеся в независимом процессе и выполняющиеся в процессе вызвавшего их приложения. Это и позволяет вам создавать приложения на основе библиотек.

Вы должны решить, где будет происходить активация — например, выделить для этого специальный компьютер. Какие компоненты совместно используют ресурсы и, скорее всего, должны находиться в одном процессе? Убедитесь, что воспользовались всеми основными операциями по созданию пула, которые, как правило, применяются к процессу. Также удостоверьтесь, что, если в компоненте возникнет сбой, приведший к его краху, остальная система продолжит работу. Этого можно добиться, ограничив процесс одним или несколькими компонентами.

Я уже упоминала защитную изоляцию — распределение компонентов по приложениям как метод реализации зашиты. Кроме того, может потребоваться привязка приложений и процессов к определенному компьютеру. А вот как вы это будете делать, зависит от ограничений, связанных с развертыванием. Также может понадобиться настройка системы в соответствии с требованиями к производительно-

сти. Для достижения достаточной масштабируемости, возможно, придется реализовать распределение нагрузки.

Таким образом, вывод очевиден: не принимайте решение слишком рано. Распределяя компоненты по приложениям и процессам, сохраняйте гибкость. С другой стороны, заранее предусмотрите все ограничения. Ведь может получиться, что ваше гибкое приложение откажется работать из-за того, что было установлено без учета предусмотренных вами ограничений.

Ну, кажется достаточно, Дэн. Надеюсь, я помогла вам получше познакомиться с COM+ и проектированием приложений.

Этими словами Мэри закончила свою лекцию.

— Конечно, Мэри. Судя по довольным лицам, всем очень понравилось твое выступление, — завершил совещание Дэн.

Билл подошел к видеомагнитофону и осторожно вынул кассету, обращаясь с ней так, как будто она сделана из чистого золота:

— Все было великолепно, Дэн. Я сделаю несколько копий для своего отдела. Им не потребуется брать кассету напрокат, они смогут ее посмотреть в любое время.

— Как же вы, разработчики, все усложняете, — сказал Тим, не упустив возможность посмеяться над Биллом. — Я выложу запись на наш новый сервер Microsoft NetShow, и твои разработчики с помощью обозревателя смогут смотреть ее хоть каждый день.

— Звучит неплохо, — выходя из комнаты, добавил Дэн. — Я оставлю вас одних — хорошенько все обсудите. Вперед, за работу, закончим наш проект!

ГЛАВА 9

Технологии уровня данных

В этой главе

Эта глава посвящена особенностям проектирования, связанным с данными, включая реализацию сервисов доступа к данным. Мы познакомим вас с характеристиками различных технологий доступа и рассмотрим способы их использования, обсудим нормализацию данных и вопросы целостности, влияние бизнес-правил на данные и поясним, где эти правила лучше реализовать. Потом мы перейдем к технологиям доступа к данным, хранящимся в традиционных системах и приложениях *управления ресурсами предприятия* (Enterprise Resource Planning, **ERP),** например, **SAP** R/3 фирмы SAP AG, И наконец, коснемся базы данных в памяти (IMDB), которая позволяет повысить производительность работы с данными.

При написании этой главы мы использовали собственный опыт проектирования и реализации приложений, а также следующие материалы:

- раздел «Visual Studio Developing for the Enterprise» документации Visual Studio 6.0;
- материалы группы Enterprise Integration Group компании Microsoft;
- статью MSDN «The SAP DCOM Component Connector»;
- Мэри Киртлэнд (Mary Kirtland) «Designing Component-Based Applications» (Microsoft Press, 1998).

Изучив материал этой главы, вы сможете:

- ✓ перечислить технологии доступа к данным;
- ✓ различать реляционные и нереляционные хранилища данных;
- ✓ охарактеризовать особенности моделирования данных;
- ✓ перечислить модели нормализации данных;
- ✓ выбрать наиболее подходящую для каждого типа приложений технологию доступа к данным;
- ✓ перечислить технологии доступа к данным для хост-систем;
- ✓ охарактеризовать связи DCOM с **SAP;**
- ✓ перечислить функции IMDB.

Уровень данных

Число способов хранения информации непрерывно и весьма быстро увеличивается. Не так давно данные хранились только на мэйнфреймах и в системах управления базами данных (СУБД). Теперь же информация располагается и в почтовых хранилищах, и в файловых системах, и в Web-документах, и в графических файлах.

При поиске наилучшего метода распространения данных по отделам организации вы, вероятно, захотите поместить всю информацию в одном универсальном хранилище, предназначенном для данных всех типов.

Универсальное хранилище

Универсальное хранилище решает проблему выбора методов доступа. Однако при его использовании возникают дополнительные трудности в создании механизма извлечения данных любых типов. Универсальное хранилище также не справляется с обработкой информации, расположенной в других местах, если ее объем измеряется в терабайтах. Кроме того, всегда существует риск, что все хранилище откажет из-за одного сбоя.

Вообще-то существует общий метод доступа — *интерфейс открытого доступа к базам данных* (Open Database Connectivity, ODBC), но хотелось бы, чтобы универсальный метод, в отличие от ODBC, поддерживал все типы данных, а не только реляционные базы данных и запросы SQL.

Интерфейсы прикладного программирования

Интерфейс прикладного программирования (Application Programming Interface, API) — это набор вызовов, используемых приложениями для запроса выполнения низкоуровневых сервисов операционной систе-

мы. В этом разделе мы опишем методы манипулирования источниками информации с помощью различных интерфейсов доступа к данным. Каждая компания-разработчик СУБД, чтобы упростить работу с ней, готовит специальный API. Доступ к данным, не поддерживаемым СУБД, обеспечивают специальные API, например, API *служб каталога* (directory services) Microsoft Windows NT (ADSI), Messaging API (MAPI) для информации в хранилищах почтовых сообщений и различных файловых API. Методы доступа, предназначенные для соответствующих хранилищ, позволяют добиться полного контроля над информацией. Однако для этого разработчик должен знать, как применять каждый из этих методов, и понимать все детали работы API, связанного с определенным хранилищем. Поэтому, если предполагается работа с несколькими хранилищами данных, придется обучить персонал всем способам работы с ними, что резко увеличивает затраты,

Разработчикам доступен и другой вариант — работать с общими нейтральными API, например, ODBC. Преимущество такого метода в том, что для доступа к СУБД требуется изучить только один API, а приложения смогут обращаться к данным нескольких СУБД.

Универсальный доступ к данным

Разработанная компанией Microsoft *универсальная архитектура данных* (Universal Data Architecture, UDA) предназначена для организации высокопроизводительного доступа к хранящимся на предприятии данным любого типа — структурированным и неструктурированным, связанным и несвязанным. UDA — набор COM-интерфейсов, реализующих концепцию доступа к данным. Она основана на интерфейсах OLE DB для создания компонентов, работающих с базами данных. OLE DB позволяет хранилищам информации открывать свои функции без необходимости имитировать реляционный источник данных. Кроме того, в рамках этой технологии универсальным сервисным компонентам (например, специализированным обработчикам запросов) разрешено расширять функции более простых поставщиков данных. Поскольку основная цель OLE DB — эффективный доступ к информации, а не простота использования, в UDA добавлен интерфейс прикладного уровня Microsoft ActiveX Data Objects (ADO). ADO поддерживает двойные интерфейсы, которые можно применять в языках сценариев, в C++, в Microsoft Visual Basic и других средствах разработки. Об этой технологии — чуть позже.

UDA — платформа и набор средств разработки, определяющий стандарты и технологии, связанные с доступом к хранилищам данных масштаба предприятия. Эта архитектура лежит в основе концепции разработки приложений Microsoft. Кроме того, UDA обеспечи-

вает высокопроизводительный доступ к различным реляционным и нереляционным информационным хранилищам и обладает простым интерфейсом, не зависящим от языка реализации.

При использовании UDA вам не придется перемещать данные в одно хранилище и работать с продуктами только одного производителя, так как UDA поддерживает все основные базы данных. Эта технология создана на основе ODBC, Remote Data Objects (RDO) и Data Access Objects (DAO), но по сравнению с ними ее возможности значительно расширены.

Компоненты доступа на основе UDA

Microsoft Data Access Components (MDAC) — это реализация UDA, включающая как ADO, так и компонент доступа OLE DB для ODBC. Это позволяет ADO обращаться к любой базе данных, снабженной драйвером ODBC (фактически ко всем основным базам данных). Существуют компоненты доступа OLE DB и для других типов хранилищ — например, почтовых хранилищ Microsoft Exchange, служб каталога Windows NT и файловой системы Microsoft Windows, использующей Microsoft Index Server. Как показано на рис. 9.1, ADO как механизм доступа к информации позволяет написать приложения для уже существующих и новых структурированных и неструктурированных данных независимо от их местонахождения.

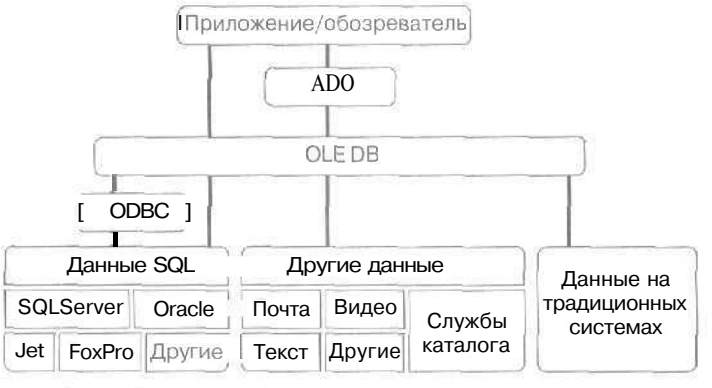


Рис. 9.1. Структура UDA

Моделирование данных

Моделирование данных — процесс идентификации, документирования и реализации требований к данным, при котором существующие модели и процессы пересматриваются с целью определения возможности их повторного использования. Кроме того, создаются но-

вые модели и процессы, реализующие требования к приложению. Основные этапы моделирования данных таковы:

- определение структуры данных и связанных с ними процессов (например, логическая организация данных):
- определение типов данных, их размеров и значений по умолчанию:
- обеспечение целостности данных с помощью бизнес-правил и ограничений;
- подготовка операционных процессов, например, проверок защиты и резервного копирования;
- выбор технологии хранения данных (реляционной, иерархической, индексируемой и т. д.),

Важно понимать, что моделирование данных тесно связано со структурой процессов в рамках организации. Так, новое понимание специфических элементов данных, используемых организацией, часто ведет к трудностям, связанным с принадлежностью информации, включая ответственность за ее хранение, точность и своевременность. Иногда при проектировании данных приходится изучать взаимодействие информационных систем предприятия — при скоординированном планировании возрастает эффективность, сокращаются затраты и появляются новые стратегические возможности,

Примечание Как мы уже отмечали в главе 6, сбор информации начинается еще при выявлении схем и сценариев использования на стадии концептуального проектирования и при описании объектов, клиентов и сервисов на этапе логического проектирования.

Определение структуры данных

Данные характеризуют реальные информационные ресурсы приложения: людей, продукты, заказчиков, имущество, отчеты и, наконец, структуры данных, которые приложение распределяет по категориям, организует и поддерживает.

Процесс определения структуры данных — итерационный. Сначала разработчики в обших чертах представляют механизм обработки информации приложением. По мере расширения знаний о бизнес-процессах он уточняется и становится яснее. Описание каждого элемента данных обычно состоит из:

- названия;
- общего описания;
- владельца (кто за него отвечает);
- характеристик;

- логических событий, процессов и связей (как и когда данные создаются, изменяются и используются).

Кроме того, следует изучить способы количественного определения элементов данных. Перечислим их основные спецификации или атрибуты:

- местонахождение (адрес, страна, полка на складе);
- физические параметры (вес, размеры, объем, цвет, материал, плотность ткани);
- концептуальные параметры (название, категория, серийный номер);
- связи (деталь, состоящая из нескольких элементов, автор, написавший несколько книг);
- ценность (стоимость).

На данном этапе выполняется анализ существующей структуры данных, документирование и тщательный пересмотр. В результате получается документ, в котором есть ответы на вопросы: «Кто, что, где, когда и как?». Таков, R общих чертах, первый этап исследования данных.

Определение характеристик данных

По мере исследования структур данных можно группировать некоторые их элементы, выявляя характеристики и связи:

- определять таблицы, строки и столбцы;
- выбирать ключевые поля;
- создавать связи между таблицами;
- определять типы данных.

Определение таблиц, строк и столбцов

Независимо от способа физического хранения, данные, как правило, организуются в виде таблиц или файлов, состоящих из нескольких строк (записей) и столбцов (полей). Своим видом они напоминают электронные таблицы (табл. 9.1). Каждая строка содержит информацию о конкретном человеке, продукте, заказчике, имуществе или связанных с ними предметах.

Табл. 9.1. Пример таблицы «Люди»

Строка	Имя	Адрес	Телефон
1	Дэн Шелли	100 Microsoft Way	(ххх) ххх-хххх
2	Тим О'Брайан	100 Microsoft Way	(ххх) ххх-хххх
3	Мэри-Лу Моррис	100 Microsoft Way	(ххх) ххх-хххх
4	Джейн Клэйтон	100 Microsoft Way	(ххх) ххх-хххх

Выбор ключевых полей

С помощью *ключевого поля* создается индекс, позволяющий быстро получать информацию. Такое поле бывает как уникальным, так и повторяющимся (если в таблице несколько копий одних и тех же данных). Если же для каждой строки существует уникальный идентификатор, ключевое поле называется *основным.* Например, в табл, 9.2 число, идентифицирующее писателя (Au_id) — основное ключевое поле, так как определяет автора. Ответ на запрос, использующий Au_id, вы получите очень быстро.

Табл. 9.2. Пример таблицы «Авторы»

Таблица авторов
Au_id (ключевое поле)
Au_name
Au_address
Au_phone

Создание связей между таблицами

Базы данных, как правило, состоят из нескольких таблиц, связанных друг с другом. Например, в таблице «Книги» можно указать ISBN книг библиотеки, их названия и год выпуска. Кроме того, неплохо бы знать и название издательства, выпустившего их. В таком случае вместо того, чтобы дублировать информацию об издательстве для каждой книги, стоит создать новую таблицу «Издательства», включив идентификатор издательства (Pu_id) в таблицу «Книги» (он для нее будет *внешним)* и таким образом связав их (табл. 9.3).

Табл. 9.3. Пример связанных таблиц «Книги» и «Издательства»

Таблица «Книги»	Таблица «Издательства»
Ti_isbn (ключевое поле)	Pu_id
Ti_title	Pu_name
Ti_yearpublished	Pu_address
Pu_id (внешнее ключевое поле)	Pu_phone

В данной таблице продемонстрирована связь *«один-ко-многим»*: каждая строка таблицы «Книги» связана только с одним издательством из одноименной таблицы, но одна строка таблицы «Издательства» может соответствовать нескольким записям в таблице «Книги». На рис. 9.2 изображены все типы связей: *«один-к-одному»*, *«один-ко-многим»* и *«многие-ко-многим»*.

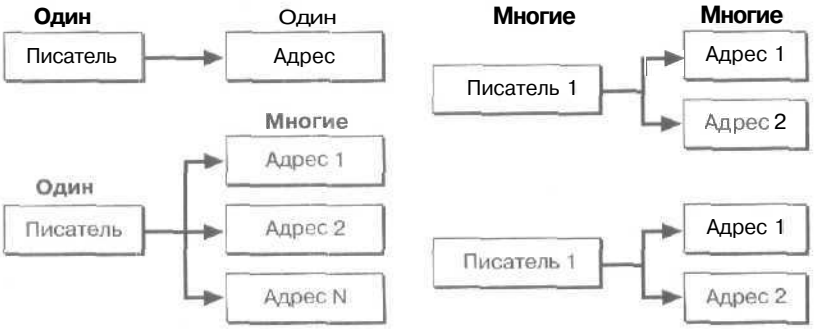


Рис. 9.2. Связи «один-к-одному», «один-ко-многим» и «многие-ко-многим»

Мы ничего не сказали об управлении связями между таблицами, В зависимости от окончательной реализации для этого используют ограничения, накладываемые Microsoft SQL Server на объединение таблиц и внешние ключевые поля, или специализированный код, напрямую считывающий структуру файла и обеспечивающий целостность ссылок в бизнес-объектах приложения.

Определение типов данных

Тип данных — это именованная категория, характеризующая набор значений, способ обозначения этого набора, а также интерпретирующие и модифицирующие его операции. Типы данных бывают:

- **предопределенными** — они встроены в СУБД. Например, в SQL Server предопределены такие типы, как integer, datetime, bit, char и varchar;
- **производные** — определяются с помощью таких средств СУБД, как *язык моделирования данных* (Data Modeling Language, DML). Производные типы основываются на предопределенных и уже существующих производных типах данных. Разработчики придумывают их структуру и название. Производные типы позволяют добиться согласованности типов данных, хранящихся в выбранных столбцах, переменных или параметрах.

Типы данных важны еще и потому, что обеспечивают соответствие значения правильному типу и гарантируют, что оно не выходит за рамки определенного диапазона. Различные хранилища информации и языки программирования поддерживают множество типов данных, например, *символьные* (character), *символьные переменной длины* (variable character), *целые* (integer), *целые двойной длины* (double integer), *числа с плавающей запятой* (floating point numbers), *битовые* (bit fields) и *бинарные поля* (binary fields).

При определении типов данных разработчики должны убедиться, что диапазон значений этих типов соответствует хранимой информации в данный момент и будет соответствовать впоследствии. Например, в SQL Server приложение способно хранить только 255 разных значений в типе данных «tinyint» и более 2 миллиардов в типе «integer». Кроме того, расширение односимвольного обозначения заказчика до двух символов может иметь разрушительные последствия — ведь придется изучить и обновить все обращающиеся к нему функции и компоненты.

Выбрав для полей соответствующий тип данных, вы сэкономите место в базе данных и число операций объединения таблиц. Возьмите на вооружение общее правило: выбирайте тип данных наименьшего размера.

При определении типов данных следует учитывать:

- допустимый минимум и максимум значений;
- значения по умолчанию;
- пустые (NULL) значения;
- ожидаемое увеличение значений;
- ожидаемые и, по возможности, неожиданные изменения.

В реляционных СУБД типы данных считаются дополнительным средством реализации бизнес-правил. Например, доллары нельзя сложить с цветами. Хотя никто еще не делал этого намеренно, но реляционная СУБД автоматически выявит несоответствие типов и запретит такую операцию.

Обеспечение целостности данных

Для обеспечения целостности данных, хранимых и используемых в структурах приложения, следует контролировать все процессы, обращающиеся к информации. Реализовать это позволяют следующие основные концепции:

- нормализация данных;
- определение бизнес-правил доступа к данным;
- обеспечение целостности ссылок;
- проверка данных,

Нормализация данных

Проектировщик базы данных должен структурировать данные, устранив, таким образом, ненужное дублирование информации и ускорив ее поиск. Такой процесс упорядочения таблиц, ключевых полей, столбцов и связей, повышающий эффективность базы данных, называется *нормализацией*. Эта процедура применима как к реляционным, так и к индексируемым файлам.

Нормализация — сложный процесс, подчиняющийся множеству специальных правил и проходящий с разными уровнями интенсивности. Дадим ее полное определение: нормализация — это процесс удаления повторяющихся групп, минимизации избыточности, устранения составных ключевых полей для частичных зависимостей и отделения неключевых атрибутов. На основании этого определения можно сформулировать простое правило: «Каждый атрибут (столбец) должен относиться к ключевому полю, только к нему и ни к чему другому». Другими словами, каждая таблица должна описывать только один тип объектов (например, человека, место, заказ или продукт).

Перечислим достоинства нормализации:

- **целостность данных** — отсутствуют избыточные ненужные данные;
- **оптимизация выполнения запросов** — нормализованные таблицы объединяются быстро и эффективно;
- **быстрое создание индексов и сортировка** — в таблицах мало столбцов;
- **быстрое обновление** — размеры индексов для таблиц уменьшаются;
- **более эффективная многопользовательская работа** — блокировка таблиц затрагивает меньший объем данных.

Простые базы данных можно нормализовать, применив следующее правило: таблицы с повторяющейся информацией надо разделить на несколько (таким образом, устраняется избыточность данных). Пусть, например, для книжного магазина требуется разработать приложение, предназначенное для контроля информации о каждой книге. Оно должно работать со следующими данными:

- имя автора;
- адрес автора;
- телефон автора;
- название книги;
- ISBN;
- год выпуска;
- адрес издательства;
- телефон издательства.

Можно, конечно, создать одну таблицу с полями для каждого элемента данных. Однако при этом возрастает избыточность. Например, если несколько книг разных авторов выпустило одно издательство, информация об издательстве будет повторяться. Если эти поля включить в одну таблицу, то, скорее всего, появятся дубликаты записей. Применяя принципы нормализации, данные следует разбить на четыре группы (табл. 9.4).

Табл. 9.4. Нормализованные таблицы «Авторы», «Книги автора», «Книги» и «Издательства»

Таблица «Авторы»	Таблица «Книги автора»	Таблица «Книги»	Таблица «Издательства»
Au_id (ключевое поле)	Au_id (внешнее ключевое поле)	Ti_isbn (ключевое поле)	Pu_id (ключевое поле)
Au_name	Ti_isbn (внешнее ключевое поле)	Ti title	Pu_name
Au_address		Ti_year-published	Pu_address
Au_phone		Pu_id (внешнее ключевое поле)	Pu_phone

Ключевые поля обеспечивают связь таблиц. Например, с помощью таблицы «Книги авторов» создана связь «многие-ко-многим» между таблицами «Книги» и «Авторы» (перу автора иногда принадлежат несколько книг). Эта таблица позволяет найти ISBN книги, написанной автором (по Au_id) или же, наоборот, определить автора некоторой книги (по полю Ti_isbn).

Можно и не создавать таблицу «Книги авторов», а добавить атрибут Au_id в таблицу «Книги». Такое решение ничем не лучше и не хуже выбранного нами, В обшем, эта работа творческая; разработчики создают сбалансированный проект, в котором надо учесть ожидаемые типы запросов, особенности многопользовательской работы и производительность индексов.

Определение бизнес-правил доступа к данным

Доступ к данным контролируется бизнес-правилами, причем каждое новое приложение должно применять те же правила, что и первое, то есть уже готовые зависимости и связи. В общем, бизнес-правила, отвечающие за доступ к данным, нужно проектировать с особой тщательностью. создавая независимые, хорошо скоординированные процессы.

Бизнес-правила доступа к данным требуются приложениям в следующих случаях:

- при вставке, обновлении, удалении и просмотре данных;
- при проверке данных;
- при управлении защитой данных;

- при доступе к данным из нескольких источников;
- для обеспечения целостности ссылок.

Бизнес-правила стоит использовать каждый раз, когда приложение вставляет, обновляет, удаляет или просматривает данные. Таким образом обеспечивается полный контроль за обновляемой информацией. Например, если приложение сохраняет новый заказ в файле счетов, бизнес-правило должно автоматически проверить кредитоспособность заказчика, прежде чем добавить в счет новые товары.

Обеспечение целостности данных — это процесс проверки значений полей и связанных с ними значений в файлах. Благодаря ему можно быть уверенным, что в числовых полях находятся именно числа, причем они не выходят за рамки соответствующего диапазона, а все связи с другими файлами не нарушены. Поместив такие алгоритмы проверки в бизнес-правила, вы создадите приложение, возвращающее корректные данные и легко адаптирующееся к новым требованиям,

Иногда требуется ограничить доступ к данным приложения, разграничив права пользователей. Для управления такими правами отлично подходят бизнес-правила.

Если приложение отслеживает данные со сложными взаимосвязями, советуем применить бизнес-правила, упрощающие доступ к нескольким источникам данных. Такое правило автоматически найдет все нужные хранилища информации и реорганизует их, упростив, таким образом, работу с ними. Например, предположим, что приложение должно определить максимально возможные платежи по каждому пункту медицинской страховки. В таком случае нужно проверить страховую историю плательщика, учесть длительность страховки и ежегодные выплаты. Естественно, что для выполнения этих операций можно создать повторно используемое бизнес-правило доступа к нескольким источникам информации, на основе которого будут выполняться все проверки такого рода.

Чаще всего бизнес-правила используют для проверки ссылок в индексированных файлах. Обычно такими файлами, в частности проиндексированными с помощью *метода доступа к виртуальному хранилищу* (Virtual Storage Access Method, VSAM), управляет ядро хранилища данных, поэтому приложение должно содержать код, обрабатывающий ограничения, удаления внешних ключевых полей и другие действия, связанные со ссылками. Такой способ обеспечения целостности годится и для реляционных баз данных, особенно если существующие триггеры, ограничения и хранимые процедуры либо не отвечают требованиям, либо слишком сложны.

Обеспечение ссылочной целостности

Для обеспечения ссылочной целостности необходимо, чтобы все внешние ключи одной таблицы указывали на существующие строки другой. При этом гарантируется синхронизация обеих таблиц при операциях обновления и удаления. Допустим, что в приложении имеются две таблицы: «Книги» и «Издательства» (табл. 9.5). Для сохранения ссылочной целостности их надо синхронизировать. Другими словами, каждый идентификатор издательства (Pu_id), находяшийся в таблице «Книги», должен существовать и в таблице «Издательства», Строку Pu_id из таблицы «Издательства» удалить нельзя, так как разорвутся ссылки таблицы «Книги». Однако выполнить такое действие и при этом сохранить целостность ссылок все же удается, если вместе со строкой Pu_id из таблицы «Издательства» удалять все строки таблицы «Книги», содержащие такой же идентификатор Pu_id.

Табл. 9.5. Примеры таблиц «Книги» и «Издательства»

Таблица «Книги»	Таблица «Издательства»
Ti_isbn (ключевое поле)	Pu_id (ключевое поле)
Ti_title	Pu_name
Ti_yearpublished	Pu_address
Pu_id	Pu_phone

Точно так же приложение не должно заносить в таблицу «Книги» строку, содержащую идентификатор Pu_id, не существующий в таблице «Издательства». Поэтому прежде чем добавить новую строку в таблицу «Книги», следует проверить ее соответствие данным таблицы «Издательства».

Реализация ссылочной целостности зависит от ядра хранилища данных и от требований к приложению. Исторически, еще со времен приложений, использовавших VSAM-файлы, за работу этого процесса отвечает код самой программы. Поэтому даже в разработках на основе SQL Server и Oracle не обязательно применять триггеры, внешние ключевые поля, ограничения целостности и функции каскадного удаления. Вместо этого ссылочную целостность можно поддерживать с помощью кода приложений.

Примечание Как правило, функции, отвечающие за ссылочную целостность, реализуют в объектах и сервисах доступа к данным, Впрочем, иногда их помещают в объекты и сервисы прикладного уровня.

Проверка данных

Проверка данных гарантирует правильность и точность информации. Ее можно реализовать по-разному: в коде пользовательского интерфейса, в коде приложения, средствами СУБД и бизнес-правил. Существует несколько типов проверки данных,

- **Проверка типов данных** — одна из простейших форм проверки данных, позволяет получить ответ на вопросы «Состоит ли строка из корректных символов?» или «Состоит ли число из цифр?». Обычно такие проверки выполняет пользовательский уровень приложения.
- **Проверка диапазона значений** — дополняя проверку типа, гарантирует, что значение находится между допустимыми минимумом и максимумом. Например, при работе с символами могут быть разрешены только латинские буквы от A до Z. Все остальные — недопустимы. Как и в случае проверки типа, проверку диапазона обычно выполняет пользовательский уровень, хотя можно создать и соответствующее бизнес-правило,
- **Проверка кодов** — иногда очень сложна. Как правило, для нее нужна специальная справочная таблица. Например, при создании приложения, рассчитывающего налог с продаж, может потребоваться таблица, содержащая региональные коды налогов, Ее стоит включить в бизнес-правило или реализовать непосредственно в базе данных.
- **Комплексная проверка** — применяется, когда простой и основанной на справочных таблицах проверок недостаточно. Обычно реализуется в виде бизнес-правил. Например, приложение, обрабатывающее медицинские страховки, может получить запрос на выплату 123,5 долларов, но его возможность может зависеть от того, какой объем медицинских расходов уже оплачен в текущем году. В такой ситуации проверка данных распространяется на оценку способов оплаты, зависящих от ограничений полиса и головыхначислений.

Внимание! Будьте внимательны при создании приложений, которые В дальнейшем предполагается *локализовать*. Большинство проверок данных может оказаться в файле ресурсов, который при локализации заменяется.

В старых файловых структурах данные часто повреждаются (например, числовые поля оказываются пустыми или содержат нечисловые символы). Поэтому весьма полезно создать утилиту, проверяющую правильность записей базы данных — в противном случае результат работы приложения будет непредсказуем.

Операционные процессы

Регулярные операционные процессы, проверяющие целостность данных, — обязательная составляющая любого приложения, нуждающегося в сопровождении. Операционные процессы включают:

- обслуживание баз данных;
- резервное копирование данных.

Обслуживание базы данных

Если разработчики отвечают и за обслуживание баз данных, в их обязанности входит периодическое выполнение определенных действий. В случае реляционных баз данных это очистка журнала, проверка пиковой загрузки памяти и процедурного кэша, сжатие файлов и проверка связей между таблицами и индексами. В иерархических базах данных надо проверять разорванные связи, тщательно исследуя структуру всех записей. VSAM-файлы следует расширить, добавив в индекс заранее выделенные позиции (их количество определяется тенденцией роста файла).

Примечание При использовании SQL Server анализ и коррекция базы данных выполняется с помощью *операторов проверки целостности* (Database Consistency Checker, DBCC),

Резервное копирование данных

Резервное копирование позволяет восстановить поврежденные данные. Существует несколько способов его реализации:

- создание копии;
- зеркал ирование;
- репликация.

При первом способе данные копируются на внешнее устройство, такое как диск или лента, в особом формате. Так как при этом сохраняется вся база данных, изменения, внесенные после резервного копирования, не учитываются (если в копию не включен журнал транзакций). Этот метод очень популярен. Однако есть и недостаток: резервное копирование следует проводить по крайней мере раз в сутки.

Большинство реляционных баз данных, в том числе и SQL Server, поддерживают *создание зеркальной копии* — постоянное дублирование всех транзакций с одного устройства на другое, называемое зеркалом. Такой метод незаметен и не требует обслуживания. Основное его преимущество — мгновенное восстановление после сбоев. Однако, если сетевое соединение с зеркалом нарушится, база данных перестанет функционировать. Кроме того, этот метод не исключает периодического копирования на обычное резервное устройство.

Примечание Существует еще одна отказоустойчивая технология, похожая на создание зеркальной копии — кластеризация. При этом два компьютера становятся единым хранилищем информации. Создав кластер из серверов, на которых расположена база данных, вы устраните опасности, связанные с физическим отказом одного из компьютеров, но не решите проблему резервного копирования информации.

Репликация — это копирование всей базы данных или ее частей на несколько удаленных компьютеров. Это не только метод резервного копирования информации, но и способ распространения данных по сети, распределения нагрузки и минимизации риска сбоев. Основное назначение репликации — поддержка согласованности информации в распределенных базах данных, поэтому обычно ее не рассматривают как средство резервного копирования.

Выбор технологии хранения данных

Существует огромное число технологий хранения данных. Самые популярные — индексируемая, иерархическая и реляционная. Такие типы хранилищ отличаются не столько способом физического хранения и получения данных, сколько своими концептуальными моделями.

Индексируемые базы данных (например, VSAM), обеспечивающие исключительно быструю выборку информации, удобны для реализации последовательных списков, произвольных выборок данных и сложных файловых взаимосвязей. С помощью индекса легко считать файл полностью или одну запись. Индексированные базы данных, как правило, поддерживают только хранение и выборку информации. За целостность ссылок и проверки данных должно отвечать приложение.

Иерархические базы данных подходят для реализации обращенных древовидных структур, например, списков материалов или штатного расписания. Доступ к иерархическим данным чрезвычайно быстр, так как структуры данных связаны напрямую. Кроме того, в них встроен механизм обеспечения целостности ссылок. Однако для их реализация иногда нужен опытный разработчик, который сможет компенсировать характерное для таких баз данных отсутствие возможностей моделирования сложных связей.

Реляционные базы данных давно стали стандартом хранения информации. Они намного предпочтительнее остальных технологий из-за их удобства. К тому же они обладают стандартным интерфейсом — *языком структурированных запросов* (Structured Query Language, SQL) — в рамках которого возможна согласованная работа многочисленных инструментальных средств. Кроме того, в реляционных базах

данных предусмотрены механизмы обеспечения ссылочной целостности, проверки данных и сопровождения БД.

При разработке приложений масштаба предприятия можно применять базы данных на мэйнфреймах — например, реляционные, файлы VSAM и AS/400. Средства SNA Server позволяют работать с распределенными базами данных и базами мэйнфреймов в сети. Эти технологии, интегрированные в Windows-приложения, расширяют возможности хранения информации.

Компоненты доступа к данным

Работа над приложением начинается с создания концептуального проекта, включающего модель объектов, определение специфических данных и требований к связям между ними. Зная эти требования, вы распределите объекты по компонентам, соблюдая следующие правила:

- за точность, полноту и согласованность данных отвечают владеющие ими объекты данных;
- объекты данных должны работать корректно независимо от того, является ли обращающийся к ним объект транзакционным;
- следует внимательно изучить способы использования объектов данных и издержки доступа к информации, связанные с сохранением состояния во время вызова методов (этот пункт особенно важен, так как объекты данных не способны сохранять состояние вне границ транзакции);
- чем меньше двусторонних перемещений объектов, тем лучше, поэтому объекты данных должны поддерживать сетевые операции. Кроме того, сокращение таких перемещений улучшает производительность приложения.

Для каждого хранилища информации определен присущий только ему метод доступа. Всякий разработчик баз данных создает свой API доступа к информации. Например, к данным, хранящимся не в базе данных, можно обратиться средствами API службы каталога Windows **NT,** MAPI (если это почтовые сообщения) или API файловой системы. Использовать все возможности хранилища разрешается только в одном случае — применяя его собственный API. Однако при этом приходится изучать множество методов доступа. К тому же, разработчики должны разобраться во всех функциях API, выбрать из них наиболее эффективные, научиться пользоваться средствами диагностики и настройки хранилища. Естественно, затраты на обучение персонала всем API доступа к данным, применяемым в вашей организации, становятся очень высокими.

Разработчикам распределенных приложений приходится решать две технические проблемы, связанные с доступом к данным. Во-первых, им редко предоставляется возможность начать работу «с нуля». Ведь, как правило, приложения должны обращаться к уже существующим данным, хранящимся в различных форматах. Например, какая-то часть информации содержится в СУБД, а другая — в менее структурированной форме. На рынке программного обеспечения представлено множество СУБД, в том числе и традиционные СУБД для мэйнфреймов [Information Management Systems (IMS) и DB2], и клиент-серверные СУБД (Oracle и SQL Server), и СУБД для персональных компьютеров (Microsoft Access и Paradox). Одно приложение может использовать несколько СУБД, а для хранения других данных — текстовые файлы, *файлы индексно-последовательного доступа* (Indexed Sequential Access Method. ISAM), электронные таблицы, сообщения электронной почты и файлы других форматов. Разработчикам же распределенных приложений придется придумывать способ объединения различных источников информации.

Во-вторых, распределенные приложения обращаются к удаленным источникам данных. В большинстве случаев пользователи работают на разных компьютерах, причем не на тех, где хранится информация. Поэтому, чтобы минимизировать сетевой трафик, необходим эффективный механизм доступа к удаленным данным. Это приобретает особую важность при увеличении числа пользователей приложения до нескольких тысяч или миллионов, многие из которых подключены к сети как через глобальные вычислительные сети с довольно высокой пропускной способностью, так и с помощью медленных модемныхсоединений.

В состав компонентов доступа к данным MDAC входят:

- ADO — программный интерфейс данных уровня приложения;
- **Remote Data Service (RDS)** — механизм кэширования данных на стороне клиента, ранее называвшийся Advanced Data Connector (ADC);
- **компонент доступа Microsoft OLE DB для ODBC** — компонент доступа к базам данных ODBC средствами OLE DB;
- **диспетчер драйверов ODBC** — библиотека, реализующая ODBC API; непосредственно вызывает соответствующие драйверы ODBC;
- **драйверы ODBC** — драйверы баз данных SQL Server, Access и Oracle.

ODBC

Избежать применения «родных» методов доступа к данным можно посредством независимых API, подобных Microsoft ODBC, ODBC —

это программный интерфейс на основе языка C для доступа к информации СУБД средствами языка SQL. Диспетчер драйверов ODBC преобразует вызовы ODBC в вызовы собственного API базы данных, позволяя приложению найти необходимый драйвер и использовать поддерживаемые им методы доступа к базе данных.

Основное преимущество ODBC заключается в том, что разработчикам приходится изучать только один API, посредством которого они смогут получить доступ ко многим СУБД. Кроме того, приложения могут получать информацию из нескольких СУБД. К тому же тип используемой базы данных не обязательно определять заранее — окончательное решение не поздно принять и при развертывании приложения.

К сожалению, у такого метода есть и недостатки. Во-первых, для каждого хранилища информации должен существовать собственный драйвер ODBC, причем он должен поддерживать запросы SQL, даже если собственный API базы данных их не использует. Во-вторых, API ODBC работает с данными в реляционной форме. Оба эти ограничения способны вызвать проблемы с неструктурированными и нереляционными хранилищами информации. И наконец, API ODBC — стандарт; другими словами, независимо от широты возможностей СУБД драйверу будет доступен только ограниченный набор ее функций, включенный в стандарт. Ведь модификация API — очень сложный процесс. Комиссия должна принять предложения, утвердить способы их реализации в драйверах и определить способ, посредством которого приложение будет узнавать, поддерживает ли данный драйвер новую спецификацию. Драйверы необходимо обновить; а приложениям приходится проверять наличие их обновленных версий.

На практике механизм ODBC используется довольно широко, он отлично подходит для приложений, работающих с традиционными реляционными базами данных. Эту технологию поддерживает большинство крупных производителей СУБД. Однако, как только приложение выходит за рамки реляционных СУБД (РСУБД), для его реализации требуются более сложные методы.

OLE DB

В OLE DB включено несколько интерфейсов доступа к данным. Они организованы так, что работа компонентов доступа зависит от возможностей хранилища информации. Технология OLE DB основана на COM, поэтому арсенал ее функций легко расширить, добавив новые интерфейсы. Чтобы узнать, поддерживаются ли хранилищем какие-то специфические функции, клиенты могут воспользоваться стандартным методом **QueryInterface**. Возможность расширения функциональных возможностей — важное преимущество OLE DB перед ODBC.

На рис. 9.3 проиллюстрирована высокоуровневая архитектура OLE DB, состоящая из трех основных элементов; потребителей данных, сервисных компонентов и компонентов доступа.

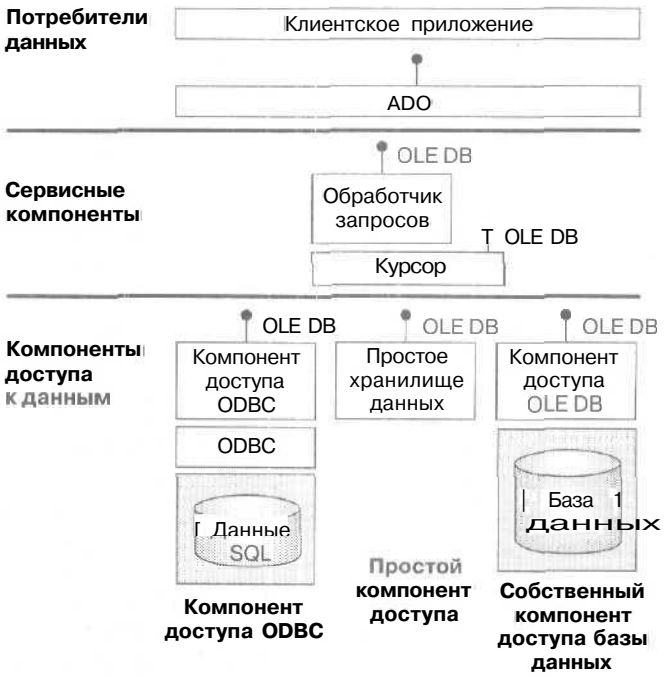


Рис. 9.3. Архитектура OLE DB

Потребители данных — это COM-компоненты, обращающиеся к информации с помощью компонентов доступа OLE DB. Компоненты доступа, также основанные на COM, инкапсулируют специализированные функции управления данными — обработку запросов, управление курсором и транзакциями. Технология OLE DB спроектирована так, что сервисные компоненты можно реализовать независимо от компонентов доступа, которые, в свою очередь, разрабатываются как автономные продукты и подключаются по необходимости. Например, простой компонент доступа способен только передавать данные, но не запрашивать, сортировать или фильтровать их.

В *сервисных компонентах* реализована обработка запросов SQL. Они выполняют такие запросы к простым компонентам доступа каждый раз, когда это необходимо потребителям.

Компоненты доступа OLE DB — это COM-компоненты, ответственные за передачу информации из хранилищ. Они представляют

все данные в виде виртуальных таблиц, или *наборов записей* (rowset). Эти компоненты обращаются к хранилищу информации посредством его собственных методов доступа или же с помощью универсальных API, например, ODBC.

В основе объектной модели OLE DB лежат семь объектов (рис. 9.4), которые реализуются компонентами доступа или сервисными компонентами, а используются потребителями данных.

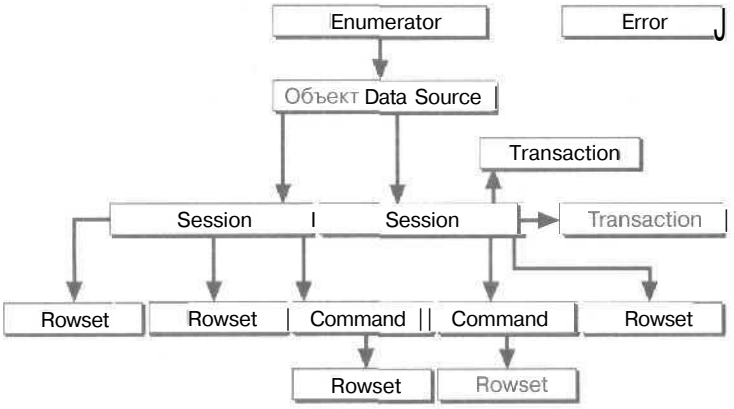


Рис. 9.4. Объектная модель OLE DB

Объект **Enumerator** находит источник информации. С его помощью пользователи, которым неизвестны какие-либо источники информации, получают список таких источников и подчиненных объектов **Enumerator.** Например, в файловой системе каждый файл соответствует источнику данных, а каждый подкаталог — подчиненному объекту **Enumerator.** Клиент ищет в полученном списке нужный ему источник информации и, найдя его, создает объект **Data Source.**

Объект **Data Source** «умеет» подключаться к хранилищам информации, например, к файлам и СУБД. Каждый компонент доступа OLE DB реализует класс **Data Source с** уникальным идентификатором CLS1D, используя который, потребитель данных может напрямую (не обращаясь к объекту **Enumerator)** создать объект **Data Source,** вызвав для этого функцию **CoCreateInstance**. Хотя CLSID каждого класса **Data Source** уникален, все классы должны быть основаны на определенном наборе интерфейсов OLE DB. Это требование очень важно, так как клиент может стандартизовать использование доступных источников информации. С помощью объекта **Data Source** он задает название источника данных, к которому хочет подключиться, и предоставляет информацию для аутентификации. После создания

объект **Data Source** раскрывает все возможности лежащего в его основе компонента доступа.

Средствами объекта Data **Source** можно создать объект **Session,** представляющий собой соединение с источником данных. Его основная функция — определение границ транзакции. Он также отвечает за создание простейших объектов доступа к данным **Command и** Rowset. Один объект **Data Source** разрешается связать с несколькими объектами Session.

Если компонент доступа OLE DB поддерживает запросы, он должен реализовать объект **Command.** Запросы генерируются объектами **Session** и отвечают за подготовку и выполнение текстовых команд. С одним объектом **Session** можно связать несколько объектов **Command.** Не забывайте, что OLE DB не различает командные языки, поэтому объект **Command** просто преобразует команды в вызовы выполняющего их компонента доступа.

Команды, возвращающие данные, создают объекты **Rowset,** которые также разрешается создавать напрямую объектом **Session.** Объект **Rowset,** широко используемый в OLE DB, представляет табличные данные. Все объекты такого типа должны реализовывать определенный набор интерфейсов OLE DB, которые позволяют клиентам последовательно перебирать строки объекта **Rowset.** получать информацию о его столбцах, связывать с ними переменные и получать информацию об объекте в целом. Реализовав другие интерфейсы, можно добавить в **Rowset** дополнительные функции, например, обновление объекта и непосредственный доступ к определенным строкам.

В модель OLE DB входит еще один объект — Error, который могут создавать любые ее объекты. Он содержит подробную информацию об ошибке, которую нельзя передать средствами объекта HRESULT, возвращаемого COM-методами. Объект **Error** основан на стандартном механизме обработки ошибок автоматизации IErrorInfo. В OLE **DB** этот механизм расширен. Теперь за один вызов вам удастся получить информацию сразу о нескольких сбоях, а компоненты доступа имеют право возвращать специфические сообщения об ошибках.

Объектная модель OLE DB — мощный и гибкий механизм единообразного доступа к данным любого типа. Благодаря ей компоненты доступа реализуют максимальное количество функций, в том числе последовательный доступ, простые наборы строк и все операции СУБД. Средства этой модели позволяют разработчикам писать компоненты доступа к данным, использующие только базисные функции. Кроме того, они получают возможность создавать компоненты, оптимизированные для определенной СУБД и использующие только ее модель программирования.

Объекты данных ActiveX

COM-интерфейсы объектной модели OLE DB не совместимы с автоматизацией, и следовательно, OLE DB нельзя использовать напрямую во многих языках и средствах программирования (например. Microsoft Active Server Pages), Поэтому в UDA определена еще одна программная модель уровня приложений — ADO. Все интерфейсы ADO — двойные, поэтому их разрешено использовать в любом языке программирования, поддерживающем COM. ADO рекомендуется применять для доступа к хранилищам данных в многоуровневых приложениях Microsoft Windows DNA.

Объектная модель ADO

Объектная модель ADO изображена на рис. 9.5. ADO основана на OLE DB, поэтому эти технологии очень похожи. Кроме того, она наверняка покажется знакомой и разработчикам, ранее использовавшим другие модели доступа к объектам данных, например, DAO или RDO. В отличие от последних, объектная модель ADO — не иерархическая. Все ее объекты, за исключением Error и Field, допустимо создавать независимо друг от друга, что упрощает их повторное использование в различных контекстах. Кроме того, такая независимость делает возможным решение задач разными способами.

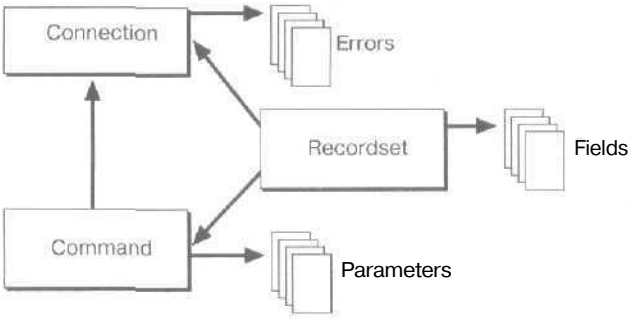


Рис. 9.5. Объектная модель ADO

Примечание Компания Microsoft разработала несколько моделей доступа к данным. DAO состоит из объектов автоматизации, осуществляющих доступ к ядру баз данных Microsoft Jet, используемому как в Access. так и в базах данных ODBC и ISAM. В RDO также входят объекты автоматизации, но они осуществляют доступ к реляционным источникам данных ODBC. Обе эти технологии поддерживаются Visual Basic 5.0. Однако, несмотря на эту поддержку, в новых приложениях лучше использовать ADO.

Объект Connection реализует сеанс подключения к хранилищу данных. По сути, это объединение объектов **Data Source и Session** OLE **DB. В** его состав входит метод Execute, значительно упрощающий выполнение простых операций. Кроме того, объект **Connection** можно подключить к объектам **Command и Recordset,** средствами которых также удается обращаться к данным.

Объекты **Command ADO** полностью совпадают с одноименными объектами OLE DB — они подготавливают и выполняют параметризованные команды источников данных. Подготовка заключается в сохранении команд в некоторой обработанной форме, благодаря которой удается быстро перейти к их выполнению. В объект **Command** входит набор **Parameters,** содержащий один или несколько объектов **Parameter,** каждый из которых отвечает за единственный параметр команды. Объекты ADO **Command** доступны только при условии, что в компоненте доступа OLE DB, лежащем в их основе, реализованы объекты **Command** OLE DB.

Основу ADO составляют объекты **Recordset.** Как и объекты Rowset из OLE DB, они представляют табличные данные. Методы **Connection и Command** возвращают не только информацию из хранилищ данных, но и доступные только для чтения объекты **Recordset,** предназначенные для последовательного просмотра. Однако более гибкие объекты **Recordset** можно создать и напрямую, подключить их затем к объектам **Connection** и (выборочно) **Command** и заполнить их посредством соответствующих методов. **Recordset** поддерживает множество способов контроля объема данных, получаемых из источника информации за определенное время. Кроме того, этот объект может управлять типом и продолжительностью блокировки источников данных, а также определять момент их обновления.

Объект **Recordset** ссылается на набор строк и столбцов, точнее — на набор столбцов, связанных с определенной строкой (текущей). Доступ к отдельным столбцам позволяет получить набор Fields, состоящий из объектов **Field** — по одному на каждый столбец. Кроме того, с каждым объектом **Recordset** связан курсор — объект, возвращающий приложению строки данных. Он указывает текущую позицию в объекте **Recordset,** тем самым определяя возвращаемую строку. В ADO поддерживается несколько типов курсоров — от простых однонаправленных до динамических, с помощью которых разработчики просматривают изменения, внесенные пользователями.

Программирование с применением ADO

Модель ADO состоит из семи объектов, перечисленных в табл. 9.6.

Табл. 9.6. Объекты модели ADO

Объект	Описание
Connection	Управляет соединением с источником данных
Command	Определяет команды, выполняемые в отношении источника данных
Recordset	Представляет набор записей из источника данных или результат выполнения команды
Field	Представляет столбец данных одного типа. Каждый объект **Recordset** содержит набор Fields, состоящий из объектов Field — по одному на каждый столбец
Parameter	Представляет параметр, связанный с объектом **Command,** основанном на параметризованном запросе или хранимой процедуре. Каждый объект **Command** содержит набор **Parameters,** состоящий из объектов **Parameter** — по одному на каждый параметр команды
Property	Динамическая характеристика объекта ADO, определенная компонентом доступа OLE DB. Все объекты **Connection, Command,** Recordset и **Field** содержат наборы **Properties,** состоящие из объектов **Property** — **по** одному на каждую динамически определяемую характеристику
Error	Содержит подробную информацию об ошибках доступа к данным, произошедших при выполнении операции. Каждый объект Connection содержит набор **Errors.** состоящий из объектов **Error** — по одному на каждую ошибку компонента доступа OLE DB

Соединения

Для соединения с источником данных используется объект Connection. Если источник содержит ODBC-данные, подключение к нему происходит с помощью метода Open, которому передаются *имя источника данных* (Data Source Name, DSN), идентификатор пользователя и пароль, или имя файла DSN.

Объекты данных должны обращаться к информации с помощью фиксированного идентификатора; идентификатор клиента лучше не использовать. Этот метод значительно упрощает администрирование и позволяет создавать пулы соединений с базами данных для многочисленных клиентских запросов. Если же вы хотите ограничить доступ к базам данных, достаточно просто ограничить доступ к бизнес-объектам, с которыми взаимодействуют клиенты, или к самим объектам данных.

Самый простой способ передачи параметров методу **Open** объекта **Connection** — включить в исходные коды название файла DSN, в котором описаны источник данных, идентификатор пользователя и пароль. В этом случае администратор сможет откорректировать информацию об источнике данных или учетной записи, не меняя исходный код компонента.

Объект **Connection** устанавливает тип доступа к базе данных. Его свойство **Mode** указывает тип соединения (только для чтения, только для записи, для чтения и записи) и совместного использования данных. Разработчики могут задавать эти свойства перед установкой соединения.

Как правило, соединение устанавливается непосредственно перед обращением к базе данных; затем его как можно быстрее разрывают, не оставляя на все время существования объекта. Такой подход приемлем, даже несмотря на издержки, связанные с подключением к базе данных, так как в диспетчер драйверов ODBC входят сервисы создания пулов соединений. Получая запрос на подключение, такой сервис сначала проверяет, не содержится ли в пуле подходящее неиспользуемое соединение. Если таковое находится, диспетчер драйверов возвращает это соединение; в противном случае он создает новое. Если соединение некоторое время (по умолчанию 60 секунд) не используется, диспетчер драйверов разрывает его и удаляет из пула. В настоящее время размер пула ограничен только количеством доступных соединений и объемом свободной памяти. Им можно управлять, устанавливая значения *таим-аута* (ODBC pooling time-out), основанного на приблизительной частоте подключений.

Диспетчеру драйверов не разрешено повторно применять соединения, установленные разными пользователями. Поэтому разработчикам приходится подключаться к базам данных с помощью фиксированных идентификаторов их объектов данных. Если при подключении передается идентификатор клиента, каждому такому клиенту потребуется свое соединение, что значительно снижает масштабируемость трехуровневой архитектуры и Microsoft Transaction Server (MTS). Кроме того, нельзя повторно использовать соединения, выходящие за границы процесса. Как уже говорилось в главе 8, необходимо, чтобы компоненты, обращающиеся к одному источнику данных, выполнялись в одном процессе — только тогда разрешается повторное использование соединений.

В документации ADO описаны методы поддержки транзакций **BeginTrans**, **CommitTrans** и **RollbackTrans** объекта **Connection.** Компоненты, выполняющиеся в среде MTS, ни в коем случае не должны их применять. Вместо них для управления транзакциями и получения

результата их выполнения следует использовать методы ObjectContext, SetComplete и SetAbort.

Доступ к данным

В ADO доступ к данным осуществляется как методом **Execute** объекта **Connection,** так и посредством объектов **Command и Recordset.** Метод **Execute** служит для выполнения заданной команды источника информации — SQL-запроса (если этот источник содержит ODBC-данные) или хранимой процедуры без параметров. Результат их работы возвращается в виде объекта **Recordset** с однонаправленным курсором только для чтения (курсоры подробно описаны в разделе «Объекты Recordset»).

Примечание Хранимые процедуры значительно повышают производительность, особенно в случае сложных операций с данными. Однако их следует использовать только для доступа к информации — прикладные алгоритмы надо реализовать в бизнес-объектах.

Если же необходимо выполнять параметризованные хранимые процедуры и команды или сохранить составную команду для использования в будущем, применяются объекты **Command.** Соединение с источником данных устанавливается с помощью свойства **Active Connection** этого объекта. Выполняемая команда задается в свойстве **CommandText.** а запускается методом **Execute.** Результат ее работы возвращается в виде объекта **Recordset** с однонаправленным курсором только для чтения. Если у команды есть параметры, они должны быть заданы в наборе **Parameters.** Для предварительной компиляции команды (скажем, для многократного использования) служит свойство **Prepared.**

И наконец, можно обращаться к данным, используя объект **Recordset** напрямую. Это самый гибкий из всех способов манипулирования информацией.

Объекты **Recordset**

В ADO практически все манипуляции с данными осуществляются средствами объектов **Recordset.** Они либо возвращаются функцией **Execute** объектов **Connection и** Command, либо создаются непосредственноразработчиком.

Объект Recordset содержит набор строк и столбцов. Как уже говорилось, в каждый момент времени он ссылается на столбцы, связанные с определенной строкой. Получить какой-либо из этих столбцов можно с помощью набора **Fields.** Перемещение по строкам объекта Recordset осуществляется посредством связанного с ним курсора.

Объекты **Recordset** исключительно полезны в трехуровневых приложениях. В этом случае состояние, хранящееся на сервере, не может

совместно использоваться разными методами. Такие серверные состояния должны подключаться к базе данных, получать из нее информацию, отсоединяться и возвращать данные вызвавшему их объекту. Объекты **Recordset** позволяют решить эту проблему с помощью ADO.

Блокировка **объектов** Recordset

Объекты **Recordset** поддерживают различные типы блокировки при обновлении записей. Тип блокировки указывается в свойстве **LockType**:

- **adLockOptimistic** — оптимистическая блокировкаотдельных записей при вызове метода **Update;**
- **adLockPessimistic** — пессимистическая блокировка отдельных записей при чтении;
- **adLockBatchOptimistic** — оптимистическая пакетная блокировка объекта Recordset только при вызове метода **UpdateBatch**;
- **adLockReadOnly** — блокировка только для чтения (применяется по умолчанию). При ее использовании обновление данных объекта **Recordset** запрещено.

Примечание Существует несколько методов перемещения по строкам объекта **Recordset.** Два из них — **MoveNext и MovePrevious** — позволяют перемещаться на одну запись вперед и назад. Достижение конца или начала файла определяется по свойствам EOF и BOF. Эти методы используются в объектах **Recordset,** поддерживающих динамическое позиционирование. Если же они поддерживают закладки, то, обратившись к свойству **Bookmark,** можно получить уникальный идентификатор текущей записи объекта **Recordset**. позволяющий при необходимости вернуться к этой записи. Кроме того, к записи можно перейти по ее порядковому номеру.

Самый простой способ заполнения объекта **Recordset** — подключить его к объекту **Connection** с помощью свойства **ActiveConnection**, после чего вызвать метод Open объекта **Recordset.** Если же информация получена не из источника данных OLE DB, объект **Recordset** разрешается заполнить программно.

Типы курсоров ADO

ADO поддерживает четыре типа курсоров.

- **Динамические курсоры** позволяют разработчикам просматривать добавленные, измененные и удаленные пользователями данные. Все перемещения по объекту, не основанные на закладках, разрешены. Закладки доступны, если их поддерживает компонент доступа OLE DB,

- **Ключевые курсоры** похожи на динамические, но не позволяют просматривать данные, добавленные и удаленные пользователями. Такие курсоры всегда поддерживают закладки.
- **Статические курсоры** представляют статическую копию набора записей. Просмотр добавленных, измененных и удаленных данных не поддерживается. Эти курсоры всегда поддерживают закладки и, следовательно, допускают любые перемещения по объекту **Recordset.**
- **Однонаправленные курсоры** похожи на статические, но разрешают перемешаться по объекту Recordset только вперед.

Функции объекта **Recordset,** доступные разработчикам, зависят от заданного при его создании типа курсора,

---

Примечание Не каждый компонент доступа OLE DB поддерживает все типы курсоров. При использовании компонентов доступа для ODBC допустимые типы курсоров определяются свойствами драйвера ODBC соответствующей СУБД. Драйвер ODBC для SQL Server поддерживает все четыре типа курсоров.

Отсоединенные объекты Recordset

Отсоединенные объекты **Recordset** используют оптимистическую блокировку; ими можно манипулировать на стороне клиента с помощью клиентской библиотеки курсоров. (В этом случае клиентом считается как презентационный, так и прикладной уровень.) Пакетные обновления таких объектов выполняются методом UpdateBatch. Однако его применение требует особой осторожности, так как всегда существует риск, что два клиента начнут изменять записи одновременно. Если такой конфликт все-таки произойдет, будет возвращено сообщение об ошибке. Поэтому разработчики должны определить в интерфейсе компонента способы обработки частичных обновлений — будут ли генерироваться ошибки транзакции или же их обработает клиент.

Для создания отсоединенного объекта Recordset нужно до установления соединения присвоить свойству CursorLocation объекта **Connection** или **Recordset** значение adUseClient. После этого следует получить необходимые данные и освободить **объект** ActiveConnection. Если вы хотите позволить клиенту изменять информацию, создайте объект **Recordset,** использующий либо пакетную оптимистическую блокировку (adLockBatchOptimistic), либо статический **(adOpenStatic)** или ключевой (adOpenKeyset) курсор.

Набор Fields

Как мы уже говорили ранее, доступ к столбцам текущей строки объекта **Recordset** можно получить посредством набора **Fields. К** полю об-

ращаются либо по его названию, либо по числовому индексу. Если используются источники данных ODBC, названия полей соответствуют названиям, заданным в списке полей оператора SQL SELECT. Числовой индекс определяется позицией поля в этом списке. Получив объект **Field,** разработчики получают или устанавливают его данные с помощью его свойств. Обычно применяется свойство **Value,** содержащее значение поля.

Для обработки длинных двоичных или символьных данных служат методы **GetChunk и AppendChunk** объекта **Field. GetChunk** используется для получении части данных, а **AppendChunk** — для ее записи. Доступность этих методов определяется свойством **Attributes** объекта Field.

Обработка ошибок

Любая операция ADO может вызвать сбой, поэтому очень важно обрабатывать ошибки внутри методов. Эта обработка состоит из двух этапов: возвращения кодов ошибок для каждого вызова и применения стандартного механизма COM для передачи сообщений **IError-Info**. Коды ошибок компонентов доступа OLE DB (например, коды ошибок ODBC) сохраняются в наборе **Errors,** связанном с объектом **Connection**. Один вызов ADO способен сгенерировать в этом наборе сразу несколько ошибок. Поэтому для получения всех сообщений нужно последовательно пройти весь набор **Errors.**

Перед каждым вызовом, который может вызвать ошибку. ADO очищает объект **ErrorInfo**. Однако набор **Errors** очищается и заполняется заново только при появлении нового сбоя или при вызове метода Clear. Существуют методы и свойства, генерирующие предупреждения, но не прерывающие выполнение программы. В таком случае перед их вызовом следует очистить набор **Errors** — только после этого стоит изучить свойство **Count** и выявить появление новых сообщений. Предупреждения выдают методы **Resync, UpdateBatch, CancelBatch и Filter** объекта **Recordset.**

Сервисы удаленных данных

Хотя в ADO набор записей предоставляет доступ только к одной строке таблицы, перемещение по курсору не обязательно ведет к обращению к хранилищу. Объект **Recordset** способен кэшировать данные, что очень важно для создания масштабируемых распределенных приложений.

Рассмотрим в качестве примера интернет-магазин, к которому одновременно обращаются тысячи пользователей. Каталог его товаров, скорее всего, хранится в базе данных. Если для каждого пользователя устанавливать отдельное соединение с этим каталогом на все время его пребывания в магазине, то количество покупателей будет ограничено возможностями базы данных. Если же подключение ис-

пользовать только для передачи блоков данных, одно соединение сможет обслужить нескольких посетителей.

Кроме того, есть и еще один выход — переслать на компьютер покупателя большой блок информации, благодаря чему число обращений к серверу БД сократится. Очевидно, что последний способ не только снизит сетевой трафик, но и улучшит время отклика магазина на запросы.

Для воплощения этого подхода в жизнь применяются отсоединенные объекты Recordset и *сервисы удаленного доступа к данным* (Remote Data Services, RDS). Архитектура RDS проиллюстрирована на рис. 9,6,

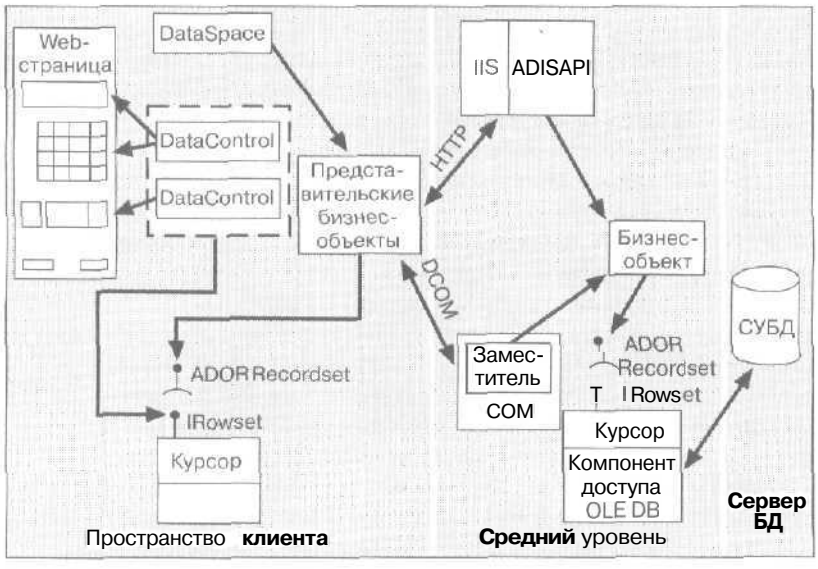


Рис. 9.6. Архитектура RDS

Отсоединенный объект **Recordset** просто отделен от объекта **Connection.** Отсоединенные объекты **Recordset** не поддерживают блокировку хранилищ информации — они кэшируют все данные. В случае записи изменений **Recordset** в хранилище OLE DB сначала изучает все модифицированные строки на предмет конфликтов при обновлении. Другими словами, OLE DB проверяет, не изменена ли строка уже после кэширования объекта **Recordset.** Если результат окажется положительным, о конфликте сообщается приложению, которое и должно предпринять действия по его устранению.

В RDS предусмотрены клиентские курсоры. К тому же эта технология обладает очень эффективными сервисами маршалинга наборов

записей между компьютерами по Интернету или интрасети. Благодаря этим функциям серверное приложение выбирает данные и передает их клиентскому, которое, в свою очередь, сможет обращаться к ним, как будто подключено к реальному хранилищу информации. Кроме того, в RDS разрешается связывать объекты **Recordset** с элементами управления, что значительно упрощает разработку клиентских приложений этого типа.

Помимо прочего, в RDS имеются три компонента, облегчающие создание приложений: **RDS.DataControl**, **RDSServer.DataFactory** и **RDS.DataSpace**. Объект **RDS.DataControl** используется для связывания элементов управления ActiveX с объектами **Recordset.** Его средствами приложение может просматривать и изменять наборы записей. **RDSServer.DataFactory** — универсальный бизнес-объект, взаимодействующий с источниками данных. Он не содержит бизнес-правил и прочих специальных прикладных алгоритмов.

Объект **Recordset** создается двумя способами. Во-первых, его косвенно создает объект **RDS.DataControl.** В свойства последнего нужно занести информацию о сервере данных и запросе. Затем, после вызова метода **Refresh,** средствами объекта **RDSServer.DataFactory** (обращаться к которому непосредственно не требуется) будет создан объект **Recordset.** Во-вторых, можно определить свой бизнес-объект, возвращающий отсоединенные наборы записей. Сначала посредством объектов **RDS.DataSpace** на стороне клиента создается представитель бизнес-объекта. После этого приложение вправе вызывать любые методы, возвращающие объект **Recordset**, получив который, оно должно настроить свойство **Recordset** объекта **RDS.DataControl.**

В этой книге мы описываем только второй способ. Бизнес-объекты среднего уровня средствами объектов данных создают отсоединенные наборы записей, которые затем передаются презентационному уровню. Клиентские приложения презентационного уровня применяют RDS для связывания объектов RDS с элементами управления.

В интрасети для пересылки объектов **Recordset** между клиентским компьютером и сервером разрешается использовать DCOM. Однако, если те же действия осуществляются через Интернет, применяется HTTP, функции для работы с которым включены в RDS. Для этого на Web-сервере нужно установить ISAPI-расширение Advanced Data 1SAP1 (ADISAPI). ADISAPI обрабатывает запросы от представителей клиентских бизнес-объектов. Он создает серверные объекты, вызывает методы, генерирующие объекты **Recordset,** и преобразует данные в наиболее эффективную для передачи клиенту форму. ADISAPI и представители бизнес-объектов полностью отвечают за все детали пересылки информации по протоколу HTTP.

Выбор технологии доступа к данным

Практически все приложения обращаются к данным. Реализовать доступ к информации в автономных приложениях очень легко, но гораздо сложнее сделать это в приложениях масштаба предприятия. Вам придется учитывать удаленные источники данных, информация в которых хранится в разных форматах, да и сами эти хранилища реализованы по-разному.

Разработчики часто задают вопрос: «Какая технология доступа к данным лучше подходит для моего приложения?» Чтобы ответить на него, нужно учесть два момента: важность повторного использования кода и способность разработчиков реализовать выбранный интерфейс. Часто в поисках решения с высокой производительностью и управляемостью программисты применяют экзотические технологии, сопровождение которых связано с большими затратами. Новые же технологии, как правило, сокращают время разработки и упрощают кодирование, не снижая производительность и функциональность приложений.

В большинстве случаев эффективны практически все технологии Microsoft для доступа к данным. Тем не менее у каждой из них есть свои сильные и слабые стороны. Поэтому окончательный выбор можно сделать, только изучив их,

ADO

ADO — новейшая технология Microsoft для доступа к данным. Она и ее партнер OLE DB подходят для любой работы с данными. Если вы разрабатываете новое приложение, используйте ADO.

Если рассматривается переход от какой-либо технологии доступа к ADO, нужно выяснить, оправдают ли выгоды, полученные от ADO, затраты на преобразование приложений. Ведь преобразовать автоматически старый код, написанный с использованием RDO или DAO, невозможно. Однако любое решение, уже реализованное средствами других технологий доступа к данным, можно реализовать и посредством ADO. Таким образом, в долгосрочных проектах следует применять ADO.

RDO

Если ваше RDO-приложение хорошо работает, причин что-либо менять в нем нет. Чтобы его расширить для доступа к данным нового типа, стоит рассмотреть возможность применения ADO и, следовательно, преобразования приложения. В новых приложениях, как уже говорилось ранее, следует использовать ADO.

ODBCDirect

ODBCDirect подойдет, если приложение должно выполнять запросы и хранимые процедуры реляционных ODBC-совместимых СУБД или если ему нужны только специфические функции ODBC (например, пакетные обновления или асинхронные запросы). Однако все функции ODBCDirect присутствуют и в ADO.

От технологии ODBCDirect не стоит отказываться, если разработчики имеют большой опыт ее применения, если ее средствами реализован существующий код и даже если требуется расширить возможности уже использующих ODBCDirect-приложений. Однако технология ODBCDirect неприменима для доступа к нереляционным источникам данных. В конечном счете, всегда можно перейти на ADO, воспользовавшись преимуществами этой технологии в проектировании и кодировании и заодно повысив производительность приложения.

DAO

DAO — единственная технология доступа к данным, поддерживающая 16-разрядные операции. Естественно, что при создании 16-разрядных приложений другого выбора у вас нет.

DAO можно использовать для обращений к данным ODBC и Microsoft Jet, однако доступ к ним, осуществляемый с применением OLE DB или ADO, быстрее и требует меньше ресурсов.

От технологии DAO не стоит отказываться, если разработчики имеют большой опыт ее применения, ее средствами реализован существующий код и даже если требуется расширить возможности приложений, уже использующих базы данных Microsoft Jet. Но к информации из других источников данных обратиться с помощью DAO нельзя. В конечном счете вам придется перейти на ADO.

ODBC

ODBC стоит применить, если требуется высокая производительность, полный контроль над интерфейсом и невысокая ресурсоемкость.

Писать программы с помощью API ODBC очень сложно, но это позволяет добиться полного контроля над источником данных. В отличие от других технологий (ADO, RDO или ODBCDirect) ODBC не считается «пуленепробиваемым». Но, хотя, используя его, очень легко допустить ошибки, этот API обладает отличным механизмом обработки сбоев с подробными сообщениями об ошибках. В общем, разработка, отладка и сопровождение приложений на API ODBC требуют огромных знаний, опыта и времени — для написания большого объема кода. Поэтому разработчики предпочитают иметь дело с менее сложными, высокоуровневыми объектными интерфейсами, например, ADO.

ODBC не подходит для нереляционных данных (таких как ISAM-данные), так как не поддерживает поиск записей, настройку диапазонов и просмотр индексов. Эта технология просто не рассчитана на такой тип информации. Однако обратиться к ISAM-данным и данным Microsoft Jet можно средствами ODBC-драйвера для Microsoft Jet. При этом ISAM-данные конвертируются в реляционные, и, таким образом, обеспечивается поддержка необходимых функций. Естественно, что из-за появления дополнительного уровня — ядра Microsoft Jet — производительность этого метода низка.

Если приложению нужен быстрый доступ к существующим ODBC-данным и разработчики согласны написать огромное количество строк сложного кода (или уже имеют заготовки для повторного использования). ODBC — это то, что нужно.

Выбор стратегии доступа к данным

Прежде чем окончательно выбрать технологию доступа к данным, следует рассмотреть следующие вопросы.

- **Разработчики создают новый проект или корректируют существующее приложение, в котором используются устаревшие технологии доступа к данным?**

  При модификации приложения заманчиво сохранить действующую технологию доступа к данным. Кажется, что такое решение разумно и не сопровождается значительными затратами. Однако здесь есть и «подводные камни» — весьма сложно добавлять источники информации другого типа. В новых проектах стоит использовать ADO,

- **Где находятся данные? В Интернете, на удаленном сервере или же хранятся локально, на компьютере пользователя?**

  Если данные располагаются на локальных системах, весьма вероятно, что перемещать их на выделенный сервер не нужно. Если же данные находятся в удаленных источниках, стоит подумать об управлении соединением. Что случится, если приложение не сможет подключиться? Должно ли оно использовать технологии асинхронного доступа к информации, такие как ADO и RDO?

- **Какова квалификация разработчиков? Работа ли они с ADO, RDO, DAO или ODBC? Стоит ли обучать их использованию ADO?**

  Если предполагается применять ADO, окупятся ли затраты на обучение разработчиков (скажем, за счет сокращения стоимости сопровождения ПО)?

- **Требуется ли приложению доступ к реляционным и нереляционным источникам данных? Есть ли компоненты доступа OLE DB к каждому из них?**

  Если ответ положительный, используйте ADO.

- **Собираются ли разработчики применять MTS?**

  Если да, следует выбрать технологии доступа к данным, которые способны работать на сервере в качестве «диспетчера ресурсов» (в MTS так называется компонент, в котором реализованы интерфейсы, управляющие ресурсами), Например, ADO, RDO и ODBC могут быть диспетчерами ресурсов MTS, а DAO — нет. Также нужно выяснить, должны ли компоненты быть реентерабельными (как в ADO и RDO), так как это необходимо для большинства MTS-компонентов (если, конечно, вы хотите разумно использовать ресурсы и получить достаточно высокую производительность).

- **Все ли приложения уже используют API ODBC?**

  Если разработчики продолжат работу с ODBC, как их приложения будут обращаться к новым источникам данных?

Итак, разработчики должны оценить требования к приложению и выбрать стратегию доступа к данным. Основные характеристики наиболее распространенных технологий приведены в табл. 9.7.

Табл. 9.7. Основные характеристики технологий доступа к данным

Лучший выбор	Требования к приложению	Примечания
ADO	Взаимодействие с данными и приложениями на *мэйнфреймах*	Средствами Microsoft SNA Server можно настроить компоненты доступа OLE DB для таких источников данных, как файлы VSAM, CICS, IMS и AS/400
	Преобразование	Следует изучить возможность преобразования приложений, включив в них поддержку ADO. Естественно, стоит попытаться оставить уже используемую технологию доступа к данным
	Разработка нового приложения	Для новых разработок применяйте ADO
	Универсальный доступ к различным источникам и типам данных	ADO — универсальный интерфейс для всех видов доступа к данным

(*продолжение*)

	Быстрая разработка	Благодаря своей универсальности, логичности и простоте, ADO помогает сократить затраты на разработку. Вы обучаетесь один раз, а преимуществами этой технологии пользуетесь долгое время
	Высокая производительность	ADO обеспечивает высокую производительность
	Поддержка Web (ASP И IIS)	Если приложение создает HTML-страницы с помощью ASP, используйте ADO
OLE DB	Доступ к любым файлам	Можно написать компонент доступа OLE DB практически для любого источника информации. После этого в качестве технологии доступа к данным можно применять ADO
RDO	Быстрый доступ к существующим ODBC-данным	RDO — очень быстрая технология
ODBCDirect	Доступ к ODBC-данным	Производительность ODBCDirect выше, чем DAO
DAO	Усовершенствование существующего доступа к данным	DAO обладает логичной моделью программирования для так называемых DAO-ситуаций, в которых для доступа к данным используется Microsoft Jet. Если разработчики уже написали много кода, поддерживающего DAO, и не хотят обращать внимание на преимущества ADO в проектировании, кодировании и производительности, менять что-либо не стоит
	Работа в 16-разрядной среде	DAO — единственный выбор
ODBC API	Быстрый доступ к существующим ODBC-данным	Если разработчики не против создания и сопровождения сложного кода, ODBC — отличный выбор

Доступ к данным на традиционных системах

Во многих организациях данные распределены по различным устройствам и программным платформам. Хотя перемещение всей информации в единое централизованное хранилище не всегда возможно. преимущества решения, при котором данные объединены хотя бы логически, совершенно очевидны. Ведь такая стратегия отлично подходит для разных моделей программирования, используя которые, разработчики смогут модифицировать клиентские приложения, не зная точного местонахождения данных.

ADO для AS/400 и VSAM

В этом разделе описана роль ADO во взаимодействии Windows-систем с системами VSAM и AS/400 (рис. 9.7). Начнем мы со способов реализации связи с мэйнфреймами: дадим ее краткое описание, коснемся метода связи с системами IBM и средств *управления распределенными данными* (Distributed Data Management, DDM), а также расскажем о взаимодействии OLE DB с DDM при передаче данных с помощью Microsoft SNA Server.

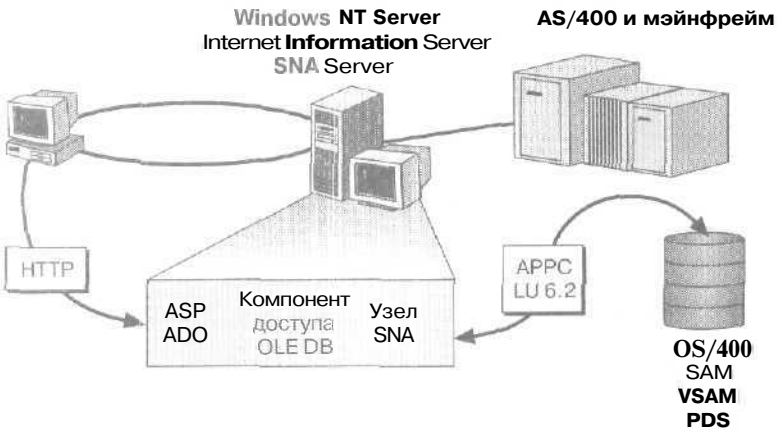


Рис. 9.7. Доступ к данным AS/400

Большинство компьютерных систем используют разные способы хранения, управления и доступа к данным. Из-за этого возникает множество трудностей, связанных с совместным использованием данных, расположенных на различных системах, и одновременным доступом пользователей к нескольким компьютерам. Кроме того, су-

шествует проблема объединения гетерогенных систем (например, SNA Server и Windows NT) в сеть.

Компания IBM разработала архитектуру Distributed Data Management (DDM) — стандартизованный метод доступа, осуществляемого схожими и разными компьютерными системами, а также и разными операционными системами. DDM поддерживает три типа хранилищ информации: *записи* (records), *потоки байтов* (byte streams) и реляционные базы данных.

Как уже говорилось, Microsoft создавала OLE DB для обеспечения совместной работы разных хранилищ данных. Для интеграции систем хранения, управления и доступа к данным независимые производители ПО реализуют OLE DB в своих компонентах доступа.

Как показано на рис. 9.7, Microsoft разработала компоненты доступа OLE DB для AS/400 и VSAM, а также компонент доступа к данным в хранилищах SNA посредством DDM. Он соответствует четвертому уровню архитектур IBM DDM и OLE DB, Компоненты доступа OLE DB для связи между узлом SNA и операционными системами Windows NT используют SNA Server.

Компоненты доступа OLE DB для AS/400 и VSAM поддерживают следующие функции:

- настройку атрибутов и описаний записей файла на мэйнфрейме (информация о столбцах);
- переход к первой или последней записи файла;
- перемещение к следующей или предыдущей записи;
- поиск записи по ее индексу;
- блокировку файлов и записей;
- изменение записей файла;
- вставку и удаление записей;
- сохранение атрибутов файлов и записей.

DDM и OLE DB

В работе DDM участвуют две компьютерные системы: система-источник, где работает запрашивающее данные приложение, и система назначения, где хранятся данные (рис. 9.8). Для доступа к файлам с помощью DDM и OLE DB требуется приложение-источник, получающее записи данных. Для передачи этих записей из файла системы назначения используется специальная информация о местонахождении данных, расположенная в строке инициализации. Последняя требуется для установки соединения с компонентом доступа OLE DB по запросу приложения-источника, за что отвечают стандартные библиотеки ADO. Компонент доступа OLE DB для AS/400 и VSAM ин-

терпретирует запросы OLE DB и транслирует их в команды DDM. Помимо информации о местонахождении источника данных, в строке инициализации содержатся *псевдоним логических модулей* (Logical Units, LU) *системы межпрограммной связи* (Advanced Program to Program Communication, APPC} на компьютере SNA Server, кодовая страница, идентификатор пользователя и пароль. Для подключения к мэйнфрейму компонент доступа использует WinAPPC и SNA Server.

За все вопросы, связанные с защитой данных, обработкой ошибок, блокировкой ресурсов и управлением потоками, отвечают архитектура DDM и протокол APPC. Когда сервер-источник DDM получает данные, он конвертирует их из DDM в типы OLE DB системы-источника.

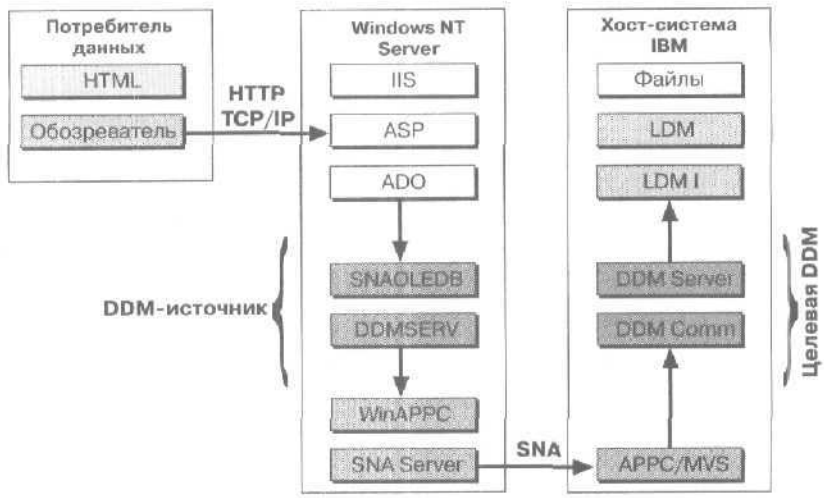


Рис. 9.8. OLE-DB и APPC

Для чтения и записи данных хост-файла приложение должно знать его формат — размер полей и формат их данных. Однако на мэйнфреймах в системе описывается только размер записи. Информация о самом поле от нее скрыта. Поэтому файлы мэйнфреймов часто называют программно-описываемыми или файлами уровня записи. Разработчикам приходится встраивать информацию о формате записей в свои программы. К тому же компонент доступа OLE DB для AS/400 и VSAM обращается к данным не из хост-приложения. Другими словами, для хост-записей не существует ни описания полей, ни метаданных. Компонент доступа OLE DB частично решает эту проблему с помошью хранящегося на локальном компьютере файла, содержащего метаданные для хост-файла. Такой файл *описания столбцов хост-*

данных (Host Column Description, HCD) создают разработчики приложений. В него включают идентификатор (название) файла, тип хост-данных (длина, точность, масштаб), тип данных компьютера (тип данных OLE DB) и CCSID (кодовая страница) хоста. С помощью этой информации компонент доступа OLE DB преобразует записи хоста в записи вызывающего компьютера. Кроме того, он конвертирует данные из EBCDIC в UNICODE, а затем с помощью API *поддержки языков* (National Language Support) — в формат кодовой страницы компьютера назначения.

В OS/400 файлы описываются на уровне записей или полей. Описываемые на уровне полей файлы часто называются файлами с внешним описанием. К ним имеют право обращаться любые приложения. Как правило, разработчики для AS/400 применяют хранимые системой описания записей на уровне полей, таким образом сокращая работу по определению записей. Для описания полей используются три элемента: *спецификация описания данных* (Data Description Specification, DDS), *интерактивная утилита определения* данных (Interactive Data Definition Utility, IDDU) и SQL. С их помощью полям назначаются атрибуты; имя, псевдоним, длина, тип данных, ограничения на проверку данных и текстовое описание. Для AS/400 уже не нужен файл описания столбцов, так как компонент доступа OLE DB для AS/400 и VSAM получает описания записей посредством команд DDM. Несмотря на это, HCD-файлы все же иногда применяются на AS/400; в этом случае их называют простыми файлами AS/400.

Компонент доступа OLE DB для AS/400 и VSAM поддерживает следующие объекты:

- **Enumerator;**
- **DataSource;**
- **Session;**
- **Rowset;**
- **Command;**
- **View;**
- **Index;**
- **ErrorObject.**

Объекты **Transaction и TransactionOptions** не поддерживаются.

Компонент доступа ADO для AS/400 и VSAM поддерживает следующие объекты:

- **Connection;**
- **Recordset;**
- **Field;**
- **Command;**
- **Parameter;**

- Collection;
- **Error.**

COMTI и интеграция данных на мэйнфреймах

Этот раздел посвящен интеграции Windows-клиентов с данными на мэйнфреймах с помощью Microsoft Component Transaction Integrator (COMTI), одного из компонентов SNA Server 4.0. Если COMTI работает вместе с MTS, CICS- и IMS-программы выглядят как обычные компоненты MTS, которые можно использовать для создания распределенных приложений. С помощью COMTI значительно упрощается создание сложных приложений, в которых Web-транзакции работают с данными на мэйнфреймах.

Основа этой технологии — COMTI Component Builder. В него включен COBOL BUILDER, с помощью которого разработчики создают требуемые Кобол-программами определения файлов и генерируют компонентные библиотеки (.tlb-файлы), содержащие соответствующие определения интерфейсов автоматизации. Благодаря тесной интеграции с MTS, при перетаскивании компонентной библиотеки в окно MTS Explorer (или во встраиваемый модуль MMC) создается COM-объект, «умеющий» активизировать транзакции на мэйнфрейме. Это позволяет разрабатывать приложения, использующие технологии Active Server, которые очень просто включить в *транзакционные программы* (Transaction Programs, TP) мэйнфреймов. Похожим образом осуществляют доступ к таким приложениям из Windows-приложений для Интернета и интрасети. При этом разработчикам не приходится изучать новые API и разрабатывать собственные интерфейсы для каждого приложения. Не надо и переписывать приложения для мэйнфреймов, созданные с применением языка Кобол — ведь исполняемый код COMTI не должен запускаться на мэйнфреймах, вся его работа происходит в среде Windows NT Server. Клиентские приложения просто вызывают методы сервера автоматизации и TP, как будто к ним обратилась другая программа мэйнфрейма.

COMTI экономит время, затрачиваемое на программирование специализированных интерфейсов для мэйнфреймов. Действуя как универсальный представитель, COMTI перехватывает обращения к методам объектов и перенаправляет их соответствующим программам мэйнфрейма, при этом обрабатывая и возвращаемые значения. Перехватив вызов, COMTI конвертирует и форматирует параметры метода так, чтобы они были понятны транзакционным программам мэйнфреймов. Причем вся эта работа выполняется в среде Windows NT Server — на мэйнфрейме запускать какой-либо код не нужно. Во время взаимодействия Windows NT Server с мэйнфреймами исполь-

зуются стандартные коммуникационные протоколы (например, LU6.2, поддерживаемый SNA Server).

COMTI способствует взаимодействию компонентов автоматизации и приложений мэйнфреймов. COMTI-компоненты, запущенные в Windows NT Server, выглядят как серверы автоматизации, которые разработчики могут использовать в своих приложениях. На самом деле COMTI работает в качестве представителя, связывающегося с приложениями, работающими под управлением ОС IBM Multiple Virtual Storage (MVS).

Программы, работающие частично на Windows-платформах и частично на мэйнфреймах, называются распределенными приложениями. COMTI поддерживает все такие приложения при условии, что они соответствуют спецификациям автоматизации и DCOM, хотя и не все части программы в обязательном порядке придерживаются этих стандартов.

Доступ к транзакционным процессам на мэйнфреймах осуществляется также с помощью технологии COMTI, поддерживающей сервисы Customer Information Control System (CICS) и Information Management System (IMS) компании IBM. Пример распределенного приложения такого типа — простое считывание базы данных DB2 на мэйнфрейме с целью обновления информации БД SQL Server в среде Windows NT Server.

Клиентское приложение может выполняться под управлением Windows NT Server, Windows NT Workstation, Windows 95/98 и на любой другой платформе, поддерживающей DCOM. Так как DCOM - независимая от языка программирования технология, для создания приложения разрешается использовать любые средства, с которыми знакомы разработчики, включая Visual Basic, Visual Basic for Applications, Visual C++, Visual J++, Delphi, PowerBuilder и Microfocus Object COBOL. Такой клиент сможет обратиться ко всем зарегистрированным в Windows NT Server COMTI-объектам автоматизации (и ко всем остальным объектам автоматизации).

Упрощение применения транзакций средствами COMTI

Хотя COMT1 используют и с простыми приложениями, обращающимися к данным на мэйнфреймах, эта технология намного мощнее, чем кажется на первый взгляд, — она позволяет распространять транзакции из среды Windows NT Server на мэйнфреймы. Windows-программы, применяющие MTS, могут включать CICS-приложения в координируемые MTS транзакции (начиная с SNA Server 4.0 SP2 в COMT1 были включены IMS-транзакции, после того как IBM обеспечила

поддержку Sync Level 2 для IMS посредством IMS 6.0). COMTI полностью интегрирована в MTS, поэтому:

- разработчики Windows-приложений без труда описывают, выполняют и администрируют специальные объекты MTS, обращающиеся к транзакционным программам CICS или IMS;
- разработчики приложений для мэйнфреймов предоставляют Windows-приложениям для Интернета и интрасетей доступ к транзакционным программам мэйнфреймов;
- разработчики MTS-компонентов включают приложения для мэйнфреймов в состав двухфазных транзакций MTS.

Теперь разработчики, использующие MTS, вправе решать, каким частям приложения нужны транзакции, а каким нет. Имея в арсенале такое средство, как COMTI, они могут сделать это же и с мэйнфреймами, обрабатывая как вызовы требующие транзакций, так и не требующие их. COMTI обладает полным набором функций, необходимых для интеграции двухфазных транзакций (в Windows) и Sync Level 2 (на мэйнфреймах). При этом не нужно модифицировать клиентское приложение и помещать на мэйнфрейм исполняемый код, но небольшие изменения в транзакционные программы внести придется. Однако от клиента не требуется различать ссылки на COMTI-компоненты и на остальные MTS-компоненты.

В COMTI включены два открытых интерфейса:

- COMTI Management Console;
- Component Builder.

Стандартный представитель времени выполнения COMTI не только предоставляет каждому COMTI-компоненту интерфейс сервера автоматизации, но и взаимодействует с программами мэйнфреймов. У представителя нет открытых интерфейсов. COMTI Management Console собирает информацию о пользовательской среде и настраивает COMTI для работы с Windows NT Server и средой выполнения транзакционных программ мэйнфреймов MVS. Component Builder (CB) — это простое в использовании средство создания компонентных библиотек (.tlb-файлов). Оно позволяет создавать объявления данных Кобола, используемых в CICS- и IMS-программах для мэйнфреймов. CB — автономное приложение, поэтому для его работы не придется устанавливать Visual Basic или C++.

Во время выполнения COMTI перехватывает вызовы методов компонентной библиотеки COMTI, преобразует и форматирует их параметры и отсылает (или получает) их соответствующей программе на мэйнфрейме. Для такого преобразования параметров используются созданные с помощью CB компоненты. С целью поддержки двухфаз-

ных транзакций COMTI тесно интегрирована с MTS и Microsoft Distributed Transaction Coordinator.

Component Builder на компьютерах пользователей устанавливать не нужно, но COMTI Management Console и представитель времени выполнения должны присутствовать.

Различия между технологиями Windows-платформ и мэйнфреймов

Смысл термина «транзакция» зависит от среды, В MTS — это набор действий, которые *диспетчер распределенных транзакций* (Distributed Transaction Coordinator, DTC) выполняет как неделимую единицу. В CICS-средах определение транзакции носит более общий характер. Любая CICS-программа, связывающаяся с другой CICS-программой с помощью APPC, называется транзакционной. APPC — это набор протоколов, разработанный IBM специально для применения в одноранговых сетях, состоящих из мэйнфреймов, AS/400, кластерных контроллеров 3174 и других «интеллектуальных» устройств. Диапазон сервисов, предоставляемых TP, довольно широк — это взаимодействие терминалов, передача данных, запросы и обновления баз данных.

Для непосредственного взаимодействия одной TP с другой с помощью APPC две программы должны установить связь по протоколу LU6.2. LU6.2 — стандарт де-факто для выполнения распределенных транзакций в средах мэйнфреймов, используемый в подсистемах CICS и IMS. Существует три уровня синхронизации при взаимодействии программ:

- Sync Level 0 практически не обеспечивает контроля целостности сообщений, кроме номеров в последовательности, что позволяет лишь выявить потерянные и повторные сообщения;
- Sync Level 1 поддерживает операцию CONFIRM-CONFIRMED (подтвердите — подтверждаю), что позволяет осуществлять пересылку подтверждений между клиентом и сервером;
- Sync Level 2 поддерживает операцию SYNCPT, которая реализует свойства ACID для двухфазных распределенных транзакций,

Из всех этих трех уровней только Sync Level 2 обеспечивает ту же надежность, что и MTS. Таким образом, в CISC- и MTS-средах под термином «транзакционная программа» не всегда подразумевается использование механизма 2PC — иногда так называют саму программу. Только в случае, когда для транзакций используется Sync Level 2, разработчики приложений для MTS и мэйнфреймов могут быть уверены, что говорят об одном и том же. Точно так же использование термина «транзакции Sync Level 2» свидетельствует о взаимопонима-

нии Windows-программистов и их коллег, пишущих программы для мэйнфреймов.

COMTI поддерживает синхронизацию Sync Level 0 и Sync Level 2, Если вызов метода входит в координируемую посредством DTC транзакцию, для взаимодействия с CICS используется Sync Level 2. В противном случае применяется Sync Level 0.

Другие средства

DCOM-коннектор для SAP

SAP — специализированная система для выполнения бизнес-приложений. Ее средствами пользователи могут управлять SAP-данными посредством специальных интерфейсов. Этот продукт позволяет отображать бизнес-процессы и управлять ими в графической форме.

Одна из последних версий системы SAP, R/3 — объектно-ориентированная. Поэтому в ней стала возможной поддержка программируемых COM-интерфейсов средствами DCOM-коннектора для SAP,

DCOM-коннектор позволяет обращаться из MTS к экземплярам используемых во время выполнения бизнес-объектов или удаленных функций R/3-приложения как к COM-объектам. Сам же DCOM Connector устанавливается в MTS в виде пакета компонентов, позволяющих разработчикам создавать и администрировать COM-объекты.

DCOM-коннектор для SAP R/3 основан на многоуровневой архитектуре и включает сервер R/3-приложений, в который входят бизнес-объекты и удаленные функции SAP, вызываемые из компонентов MTS. Взаимодействие этих функций и объектов с COM реализуют специальные библиотеки. Клиенты могут обращаться к этим компонентам посредством MTS, который, в свою очередь, связывается с сервером приложений R/3.

Соответствие между ABAP-объектами (BAPI) и COM-интерфейсами также устанавливается с помощью DCOM-коннектора. (ABAP — Advanced Business Application Programming Interface, расширенный интерфейс программирования бизнес-приложений.) Термин BAPI относится к неизменяемым объектам. Им обозначают интерфейс между процессами и данными систем бизнес-приложений, Кроме того, SAP Business Repository (BOR) интегрирован с Microsoft Repository (XML Repositories). Тем не менее такое решение масштабируемо, так как компоненты разрешается устанавливать и распределять сколь угодно часто, а соединения помешаются в пул вызовов удаленных функций R/3 (Remote Function Calls, RFC). При использовании клиентов Windows NT защита обеспечивается в контексте MTS, но в рамках одного сеанса можно передать идентификатор

пользователя системе безопасности R/3. Все таблицы (tables) и структуры (structures) представляются в виде наборов записей ADO. Кроме того, поддерживается автоматический маршалинг средствами RDS. На уровне ядра для отображения таблиц ABAP используется OLE DB, причем в качестве интерфейса к OLE DB применяется ADO. DCOM-коннектор автоматически генерирует необходимые компоненты-представители и заместители. Новая версия DCOM Connector будет поддерживать 2PC и вызовы за пределами транзакции, что потребует применения COM+,

Преимущества DCOM-коннектора перечислены ниже.

- **Открытость** — DCOM-коннектор осуществляет взаимодействие между объектами R/3 и COM по гетерогенным сетям посредством четких бизнес-интерфейсов. Он включает тщательно разработанные бизнес-процессы и объекты, необходимые для открытого DCOM-доступа, что упрощает дополнение и расширение R/3 с помощью средств Microsoft (например, создание новых клиентских интерфейсов и расширение бизнес-процессов).
- **Эффективное использование накопленных знаний** - благодаря DCOM-коннектору у разработчиков появился выбор средств расширения — SAP (ABAP-объекты) или COM (Visual Studio, Visual Basic, Visual C++ и Visual J++). Это увеличит эффективность инвестиций в обучение и в огромный рынок готовых COM-компонентов.
- **Упрошенная интеграция** — интеграция бизнес-компонентов R/3 (R/3 Business Components) в продукты сторонних производителей, традиционные системы и собственные приложения стала намного проще. Теперь возможны даже распределенные транзакции, использующие нескольких систем, содержащих несколько баз данных,
- **Гибкость** — DCOM-коннектор работает, даже если сама система R/3 не запущена на платформе Windows NT. Это позволяет добиться значительной экономии за счет использования существующей инфраструктуры. Кроме того, такое решение можно без проблем расширять и модернизировать в соответствии с меняющимися потребностями разработчиков, благодаря масштабируемости серверов каждого сервисного уровня (сервер приложений R/3 и MTS).

База данных в памяти

В COM+ появилась *база данных в памяти* (In-Memory Database, IMDB), таблицы которой хранятся в памяти. Такие базы данных полезны по нескольким причинам, о которых мы и расскажем ниже.

Приложениям часто приходится получать статические данные из неизменяемых таблиц. Например, в таких таблицах хранятся почто-

вые индексы и соответствующие им города, которые автоматически подставляются в заполняемую форму, после введения в нее индекса. Бизнес-объект может использовать таблицу и для проверки соответствия телефонного номера адресу. Естественно, таблица очень большая, и постоянные обращения к ней, а особенно ее извлечение из базы данных, значительно снижают производительность приложения. Эту проблему и решает IMDB, осуществляя кэширование данных — в этом случае таблица загружается в базу данных IMDB только один раз. Объекты данных после соответствующей настройки будут получать информацию уже из кэша, а не из постоянной БД. Так как память относительно дешева, этот подход позволяет без особых затрат повысить производительность приложений.

Хотя кэширование полезно в основном только для считываемых данных, IMDB можно использовать и для записи информации. Такое решение подойдет, если объекты данных расположены не на сервере, а на разных компьютерах. Для уменьшения сетевого трафика IMDB-кэш можно разместить на сервере. Объекты данных при этом обращаются к TMDB как для чтения, так и для записи. Измененные записи будут переданы в постоянное хранилище по завершении транзакции. Это решение особенно эффективно, если базу данных можно разделить на части так, что только один компьютер обновляет только набор записей кэша.

Кроме того, база данных в памяти — альтернатива *диспетчеру общих свойств* (Shared Property Manager, SPM), управляющему общим промежуточным состоянием. В отличие от SPM, IMDB предназначена для хранения информации нескольких серверных процессов. К тому же, так как IMDB является компонентом доступа OLE DB, для обращения к данным IMDB можно применять ADO. Вероятнее всего, разработчикам лучше знакомы интерфейсы ADO, а не SPM, поэтому им проще реализовать общее промежуточное состояние.

На рис. 9.9 изображен механизм работы IMDB. Для настройки серверных процессов IMDB, запускаемых на выбранном компьютере, используется COM Explorer. Эти процессы выполняются как сервисы Windows NT. Они отвечают за управление таблицами IMDB и взаимодействие с постоянной базой данных. Сами таблицы хранятся в обшей памяти, для определения объема которой используется COM Explorer. Кроме того, IMDB предоставляет каждому клиентскому процессу представителя — компонент доступа OLE DB. Эти представители способны непосредственно взаимодействовать с таблицами IMDB только для чтения, но запросы на чтение/запись или на блокировку должны проходить через серверный процесс,

Применение IMDB накладывает и некоторые ограничения, связанные с поддерживаемыми таблицами баз данных. В частности, в таблицах обязательно должны присутствовать основные ключевые поля, а все изменения в них надо вносить средствами IMDB. Из-за этого в таблицах нельзя применять поля с автоматическим приращением (поля-счетчики), метки даты/времени и триггеры. Кроме того, в IMDB отсутствует обработчик запросов и не поддерживается частичная загрузка таблиц — они должны быть загружены полностью. Сортировка и фильтрование таблиц осуществляется посредством объектов ADO **Recordset.**

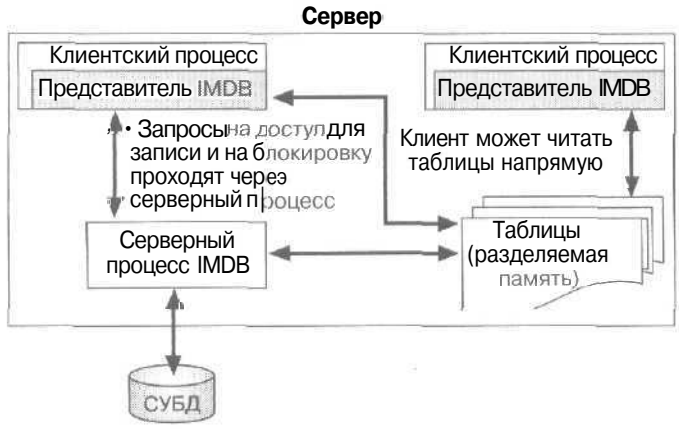


Рис. 9.9. COM и IMDB

Внимание! В COM+ 1.0 IMDB разрешено использовать только на одном компьютере. Информация, кэшируемая с помощью IMDB на нескольких компьютерах, не синхронизируется. Точно так же общее промежуточное состояние, управляемое **IMDB,** является локальным для каждого компьютера. Однако эти проблемы предполагается решить в следующих версиях IMDB, которые будут включать координатор распределенного кэша и диспетчер блокировок,

Но, несмотря на эти ограничения, использовать IMDB довольно просто. Для доступа к таблицам можно применять либо OLE DB, либо ADO, Установив посредством DSN соединение, можно обращаться к обычным методам ADO или OLE DB. Настройка DSN производится с помощью COM Explorer, что позволяет также связать источник дан-

ных IMDB с БД и определить таблицы, загружаемые при запуске сервера TMDB.

Резюме

Универсальная архитектура доступа к данным UDA предлагает универсальные методы доступа к данным независимо от их местонахождения. Она основана на OLE DB, компонентной архитектуре управления данными на системном уровне. Кроме того, в UDA входит ADO - программный интерфейс прикладного уровня для доступа к информации. ADO можно использовать в любом языке или средстве программирования, поддерживающем COM.

Архитектура UDA реализована с помощью компонентов доступа к данным MDAC, в которые входят ADO, компонент доступа OLE DB для ODBC и несколько компонентов ODBC. В модели программирования ADO манипуляции с данными осуществляются посредством объектов Recordset. Именно ADO рекомендуется в качестве технологии доступа к данным при создании новых приложений.

Сервисы доступа к информации реализуются объектами данных, которые отвечают за точность, полноту и согласованность принадлежащих им данных. При этом они должны работать корректно независимо от того, являются ли их клиенты транзакционными или нет. Объекты данных для MTS являются внутрипроцессными COM-компонентами.

Доступ к данным имеет несколько особенностей, гарантирующих целостность информации. Это — нормализация, использование бизнес-правил и целостность ссылок.

Новые технологии, такие, как COMTI (для доступа к хост-данным), DCOM-коннектор для SAP (для доступа к системам управления ресурсами предприятия) и база данных в памяти COM+, значительно упрощают требования к доступу к информации.

Закрепление материала

1. Что такое UDA?
2. Какие компоненты доступа к данным, разработанные Microsoft, вы знаете?
3. Какой компонент доступа к данным рекомендуется использовать?
4. Каким образом обратиться к хост-данным с помощью COM?
5. Что такое IMDB?

ГЛАВА 10

Тестирование и производственный цикл

В этой главе

До начала реализации проекта важно организовать соответствующую рабочую среду. Эта глава — первая из нескольких, посвященных созданию такой среды для поддержки разработки, тестирования, сертификации и производства. Мы опишем этот цикл, именуемый *производственным*, на реальных примерах, а также расскажем о проведении тестирования приложения в контролируемой среде без «нанесения ущерба» производственной среде организации.

Мы рассмотрим особенности тестирования, обсудим некоторые методы выполнения тестов и контроля их результатов, а также методы расширения производственной среды посредством добавления дополнительных серверов.

Наконец, вы узнаете о классификации отказов и сбоев приложений, об основных проблемах, связанных с ошибками, и методах их контроля, классификации и устранения.

При работе над этой главой мы использовали свой собственный опыт создания архитектур приложений и их реализации, а также следующие материалы:

- Microsoft Solutions Framework;
- курс MSF 1516, «Principles of Application Development»;
- Мэри Киртлэнд (Mary Kirtland) «Designing Component-Based Applications» (Microsoft Press, 1998).

Изучив материал этой главы, вы сможете:

- ✓ четко определить этапы производственного цикла;
- ✓ применить к среде разработки соответствующие меры защиты;
- ✓ выявить преимущества объединения разработки, тестирования и сертификации в единый производственный цикл;
- ✓ описать типичный производственный цикл;
- ✓ выявить требования к производительности приложения;
- V описать способы тестирования и настройки программного обеспечения;
- ✓ разобраться в процессе управления ошибками.

Среда разработки

Разработку надо проводить в устойчивой и надежной среде — сегодняшние системы сложны, и их работа жизненно важна для информационной инфраструктуры предприятия. Поэтому управление средой разработки по своей важности не уступает управлению самим проектом. Эта среда должна быть полностью изолирована от производственной среды организации — их совмещение скажется отрицательно как на ежедневной работе организации, так и на процессе разработки приложения.

Чтобы изменения в разрабатываемом приложении не повлияли на производственную среду организации и на пользователей, нуждающихся в стабильной среде, следует применять систему контроля изменений как во время разработки, так и в период сопровождения приложения. Например, на заре возникновения Интернета Web-страницы содержали только текст и изображения, поэтому не составляло никакого труда их откорректировать. Однако сегодняшние Web-страницы намного сложнее, а многие из них представляют собой приложения, которые должны быть постоянно доступны. Поэтому сбой в работе узла может нанести фирме финансовые убытки или привести к потере популярности у пользователей, не сумевших вовремя получить нужную им информацию. Очевидно, что совершенно неприемлемы и простои, связанные с поиском изменений, вызвавших сбой, выявлением виновного и исправлением ошибки.

Слабая система контроля за изменениями бесполезна. Разработчики должны наложить ограничения в местах возможного обхода контролирующих процедур, например, запретить изменение приложения в производственной среде без проверки этих изменений в сре-

дах тестирования и сертификации. Процесс внесения изменений следует хорошо продумать и, по возможности, автоматизировать.

Производственный цикл

Жизненный цикл приложения состоит из четырех этапов, составляющих *производственный цикл*:

- разработка;
- тестирование;
- сертификация;
- эксплуатация.

На каждом из этих этапов работают определенные группы людей и компьютерные системы. Причем участвовать сразу в работе двух или более групп не должен ни один человек и ни один компьютер. Следует разработать набор правил, определяющих, когда и как изменения, внесенные в приложение, проходят производственный цикл. Выполняя эти процедуры, вы будете уверены, что любые проблемы будут обнаружены на стадии тестирования или сертификации, а не позже.

Разработка

Первое и самое главное правило; все изменения вносятся на компьютерах разработчиков. Если этого не придерживаться с самого начала, проект, скорее всего, потерпит неудачу. Это правило обязательное: никакие исключения недопустимы. Естественно, иногда возникает желание его не выполнять — например, если требуется внести важное изменение, а проводить его через весь производственный цикл трудно. Однако даже самые незначительные коррективы могут вызвать разрушительные последствия. Таким образом, выработка правил и неуклонное следование им гарантирует успешное завершение проекта.

Тестирование

Для каждого проекта следует разработать исчерпывающий план тестирования, распространяющийся на все основные функции приложения и гарантирующий их правильную работу. Изменения, внесенные в приложение на стадии разработки, обязательно надо передать группе тестирования. Это может быть простой командный файл или кнопка на Web-странице, активация которой лицом, имеющим на это право, инициирует «модификацию». Затем на тестовых системах необходимо выяснить работоспособность внесенных изменений.

На стадии тестирования проверяется функциональность и работоспособность приложения после внесения изменений.

Сертификация

По завершении основных тестов приложение переносится на системы сертификации. Там оно тщательно проверяется перед передачей

в производственную среду для выявления незамеченных ранее ошибок. Сертификационные тесты намного полнее проверок, выполняемых на тестовых системах. Они проверяют работу функций приложения, его производительность и работоспособность. Кроме того, можно проверить и совместную работу программы с другими приложениями, пользующимися теми же COM-объектами и базами данных. И наконец, здесь тестируется работа приложения в условиях, приближенных к реальным.

На стадии сертификации основное внимание уделяется производительности приложения и его взаимодействию с другими системами. Поэтому сертификационная среда должна быть полнофункциональной и как можно точнее отражать производственную конфигурацию. В частности, в ней необходимо смоделировать все межсетевые протоколы, транспортные сервисы COM, домены или брандмауэры, ограничивающие доступ к компьютерам. Такие элементы среды, как соединения с базой данных, должны точно соответствовать реальным (на производственных серверах).

Примечание Для тестирования и сертификации нужны как компьютерные системы, так и люди. Не следует создавать сертификационную систему, если нет персонала для этой работы.

Эксплуатация

Приложение переносят на производственный сервер только после выполнения всех тестов, доказывающих его стабильность. Преимущества такого производственного цикла очевидны: это гибкость и масштабируемость. Например, если тесты на стадии сертификации показали, что для работы приложения необходим еще один сервер, его удастся вовремя добавить и настроить. Единственные необходимые при этом действия — настройка прикладных систем на новом сервере.

Тем временем разработчики смогут работать над новой версией приложения на своем сервере.

Контроль изменений

Планирование контроля изменений и разграничение прав проще, чем их использование. К счастью, в Windows NT и Windows 2000 встроена система, гарантирующая выполнение таких правил. На рис. 10.1 показаны права разработчиков на четырех этапах производственного цикла, назначенные им средствами файловой системы Windows NT (NTFS). Очевидно, разработчики должны получать полный доступ к исходному коду, хранящемуся на сервере разработки, так как они вносят в эти файлы изменения. Однако доступ к этим же файлам на тестовом, сертификационном и производственном серверах для них

следует ограничить, так как исправления кода, не прошедшие систему контроля изменений, запрещены.

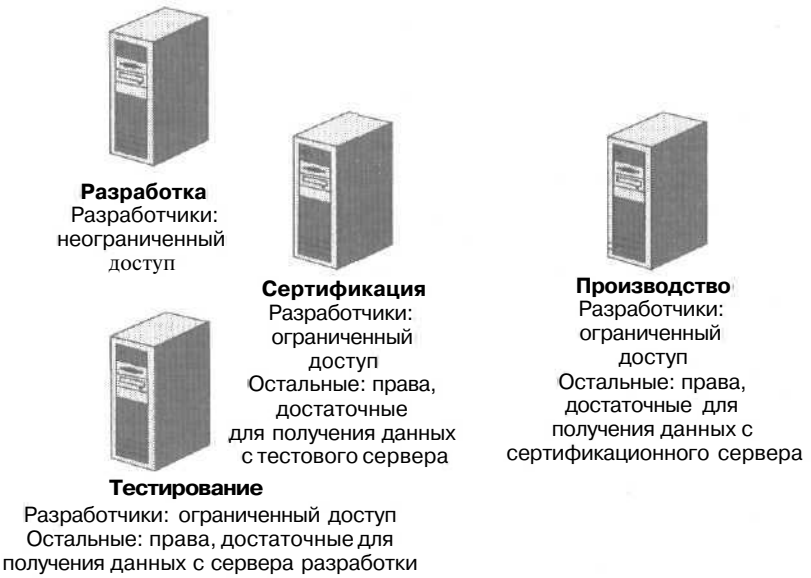


Рис. 10.1. Права разработчиков на разных стадиях производственного цикла

На рис. 10.2 упрощенно показан производственный цикл, в результате которого приложение переносится на три производственных сервера. Проиллюстрируем проблемы, возможные при отсутствии контроля изменений. Допустим, на Web-узле компании находится интернет-приложение, использующее активные страницы (ASP), бизнес-объекты COM и СУБД SQL Server. Пусть приложение состоит из нескольких Web-страниц, содержащих текст и графику, включая изображение льва.

Предположим, что разработчик решил, что эта картинка больше не нужна. Он подключился к серверу, удалил изображение и ссылку на него, не подумав о том, что другой разработчик использовал ссылку на это изображение на другой странице. Но так как изображения теперь нет, все документы, содержащие его, будут выдавать сообщение об ошибке.

А что случится если удалить картинку только с производственных серверов 1 и 2, оставив ее на третьем? Производственные среды не будут согласованы. Следовательно, пользователь, подключившийся к

узлу через производственный сервер 1, увидит льва, а подключившиеся через другие серверы его не увидят.

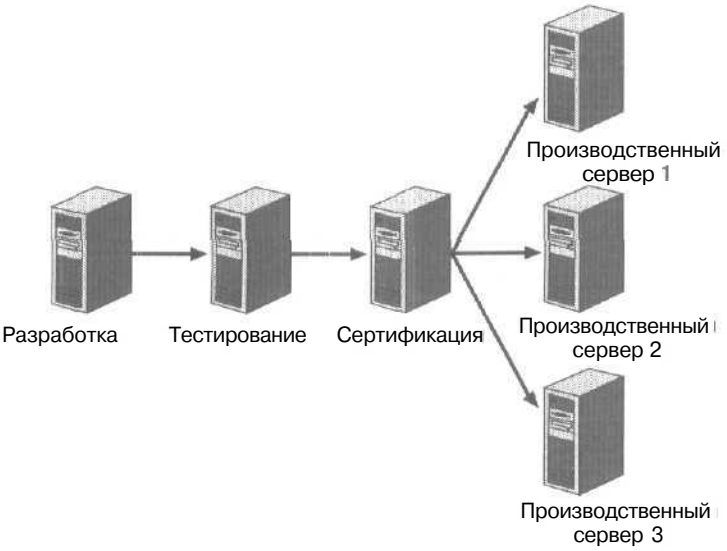


Рис. 10.2. Упрощенный производственный цикл

В нашем примере отсутствие файла изображения доставит пользователям всего лишь небольшие неудобства. Но, например, при удалении бизнес-объекта последствия нерегулируемого внесения изменений окажутся гораздо серьезнее: страницы, использующие его, откажутся работать, что неблагоприятно повлияет на все бизнес-процессы. Даже простая модификация, а не удаление бизнес-объекта может вызвать неприятные результаты.

Расширение производственного цикла

Когда над проектом, особенно Web-приложением, работает много разработчиков, важность разделения производственного цикла на этапы становится очевидной очень быстро. Однако его наличие также важно, если за приложение отвечает всего один человек. Производственный цикл дисциплинирует команду, позволяет протестировать модифицированное приложение перед развертыванием в производственной среде и при необходимости отменить внесенные изменения.

Рассмотрим простой пример производственного цикла. Часто приложения состоят из нескольких компонентов (например, Web-страниц, бизнес-объектов и данных), размещенных на нескольких серверах (например, Web-серверах, индексных серверах, серверах транзак-

ций и баз данных). Для проведения тестирования все эти серверы надо моделировать на этапе сертификации, поэтому сертификационная среда должна полностью совпадать с производственной. Но (как видно из рис. 10.3) в небольших организациях или для небольших приложений ее можно уменьшить.

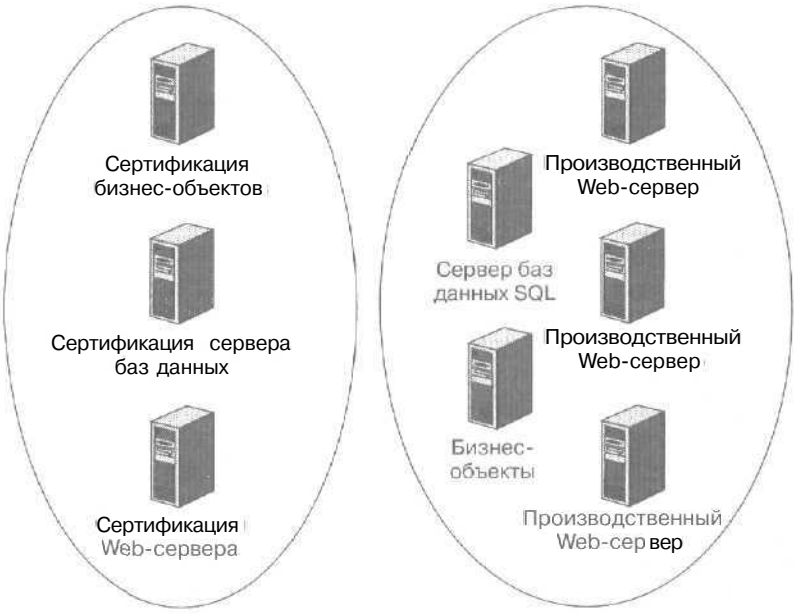


Рис. 10.3. Масштабирование сертификационной среды

В малых средах сертификационную имитацию базы данных можно поместить на производственный сервер, однако данные, используемые в производственных целях, применять ни в коем случае нельзя.

Иногда сертификационную среду допустимо размещать на одном компьютере. Однако если реальная производственная среда состоит из нескольких компьютеров, такая конфигурация не даст информации об их взаимодействии. Для выявления и устранения проблем, связанных с подключением к базам данных, необходимо подготовить несколько серверов, настроенных аналогично производственным.

Внимание! Выполнение тестов производительности, связанных с высоким сетевым трафиком, в производственных сетях иногда замедляет нормальную работу организации. Перед выполнением таких тестов следует изучить возможность разделения сертификационных и производственных сетей.

Тестирование приложений масштаба предприятия

Процесс тестирования представляет собой эксплуатацию приложения в контролируемых условиях и изучение полученных результатов. При этом проверяется работа приложения с нормальными и ошибочными данными и событиями. Следует изучить и реакцию на неожиданные ситуации.

В конечном счете тестирование проводится не только для поиска ошибок, но и для проверки качества продукта. А так как качество — это «соответствие потребностям пользователей в решении их бизнес-задач», процесс тестирования должен способствовать достижению этой цели с помощью проверки корректности работы программы.

Тестирование не ограничено фазой «Стабилизация» модели MSF — оно частично проходит и на стадии «Разработка». К моменту достижения этапа «Одобрение плана проекта» (в конце фазы «Планирование») проектная группа разрабатывает основные планы тестирования и начинает работу над подробными спецификациями тестов каждой функции приложения. Эти спецификации можно создать на этапе «Завершение разработки» (в конце фазы «Разработка»), так как после этого функциональные возможности приложения уже не изменяются.

На стадии «Разработка» проводится *полное тестирование* — тщательная проверка работы всех функций приложения в закрытой среде. На фазе «Стабилизация» выполняется *проверка работоспособности* приложения, его соответствия схемам и сценариям использования, собранным на этапе «Анализ». На этой стадии (называемой бета-тестированием) подключаются пользователи, так как желательно, чтобы оно проходило в условиях, приближенных к реальным, Необходимость устранения ошибок является приоритетной задачей фазы «Стабилизация», а так как на этом этапе продукт готовится к сдаче в эксплуатацию, важнее всего возможность управления ошибками.

Тестирование распределенных приложений несколько сложнее из-за множества взаимосвязей. Каждая зависимость, Web-страница, компонент, база данных, а также код графического интерфейса, программное обеспечение среднего уровня и сетевая инфраструктура должны быть протестированы не только на корректность работы, но и на совместимость друг с другом в разных конфигурациях,

Лучше всего для тестирования распределенных приложений применять метод «снизу — вверх». Сначала каждый компонент проверяют по отдельности. Затем его тестируют на автономном компьютере в среде MTS. И только потом проверяют работу всего приложения в распределенной среде предприятия.

Тестирование компонентов

Сначала компоненты тестируют по отдельности вне среды **MTS** (что напоминает модульное тестирование кода программы). Самый простой

способ — написать программу, вызывающую все функции компонента. Для этого используют любые языки сценариев и *средства быстрой разработки приложений* (Rapid Application Development. RAD) типа Microsoft VBScript и Microsoft Visual Basic. Для изучения проблем, возникающих при одновременных обращениях к компоненту, применяются многопоточные тестовые программы. Цель этого этапа – проверить корректность работы компонента до его переноса в среду, где будет работать распределенное приложение. Кроме того, многие средства программирования не поддерживают все возможности отладки компонентов в MTS, поэтому чем больше вы устраните ошибок на этом этапе, тем лучше.

Компоненты, как правило, используют контекст объекта, а это иногда порождает проблемы во время их тестирования вне MTS (контекст объекта существует только в MTS). Однако и эта проблема решаема. Если окончательную версию компонента можно запускать как в MTS, так и вне его, существование контекста объекта надо проверять перед вызовом каждого метода. Если же он запускается только в MTS, такую проверку проводить не обязательно. В этом случае полезно выполнить условную компиляцию (если выбранный язык программирования ее поддерживает).

Недостаток этого метода очевиден — придется создавать специальную версию компонента, способную работать вне MTS. Из-за этого возникает опасность, что в одной версии компонент будет безошибочен, а в версии для MTS останутся ошибки, однако этот способ — наилучший для тестирования компонентов, написанных средствами Microsoft Visual C++ и Microsoft Visual Basic вне среды MTS.

Локальное тестирование

После тестирования компонентов вне среды MTS их следует протестировать и в MTS, но на одном компьютере. Сначала компоненты тестируются по отдельности, а потом добавляются новые компоненты, пока приложение не будет собрано полностью. Естественно, на этом этапе вы не обнаружите ошибок, связанных с сетью и обеспечением безопасности информации. Тестируя приложение на одном компьютере, вы сможете протестировать только его транзакционное поведение и защиту данных.

Первым делом проверяется взаимодействие транзакций. Изучив реакцию программы на обычные действия, проверяют и ее реакцию на ошибочные. Протестируйте вызовы методов **SetAbort**, **SetComplete**, **EnableCommit** и **DisableCommit**, в том числе и возвращаемые ими коды ошибок, а также обработку сбоев на клиентских компьютерах. Иногда оригинальные компоненты не способны воспроизвести все ошибки. В этих случаях для полной проверки всех возможных ситуаций

нужно подготовить специальную тестовую версию компонента, использующую тот же интерфейс, но генерирующую ошибки.

Чтобы сократить работу по настройке тестовой системы, все компоненты запускают в контексте защиты интерактивного пользователя с отключенной авторизацией. Затем включают авторизацию и повторяют тесты для конкретного пользователя. И наконец, тестируют ролевую защиту.

Средства отладки

Если компонент не работает, как должно, его нужно запустить в отладчике и изучить каждую строку его кода. Для этого необходимо, чтобы компоненты содержали отладочную информацию, а отладчик запускал имитатор MTS.

Трассировка позволяет получить информацию о выходных данных по мере выполнения компонента. Такие данные особенно полезны, если программа запушена не в отладчике или ее исходный код отсутствует. Однако с трассировкой возможны проблемы, если компонент не имеет доступа к рабочему столу интерактивного пользователя, и следовательно, окна сообщений отображаются в области, где их нельзя увидеть или закрыть.

Обязательно проверяйте и все возвращаемые методами значения, и уже по ним определяйте тип ошибки, о которой сообщает COM, Например, при выполнении компонента в распределенных или защищенных средах COM способен сообщить о нарушении прав доступа или коммуникационных ошибках,

Тестирование доступа к данным

Если компоненты, обращающиеся к данным, не могут получить доступ к источникам информации, для выявления причин нужно использовать средства СУБД и ODBC. Если же вы работаете с SQL Server, тестировать подключения к базам данных, выполнение запросов и т. п. следует с помощью утилиты SQL Enterprise Manager. Для изучения операций над базами данных применяют утилиту SQL Trace. Также пригодятся утилиты Visual Data Tools и отладчик SQL из состава Microsoft Visual Studio Enterprise Edition.

Если доступ к информации осуществляется посредством ODBC, драйвер источника данных может разрешить применять диспетчер драйверов ODBC для тестирования доступа к этому источнику по определенному имени источника данных (DSN). Все перехваченные при трассировке сообщения записываются в файл журнала, по информации которого удастся подробно исследовать выполненные команды ODBC.

Если источник данных доступен только вручную, но не из MTS-компонентов, обязательно проверьте его совместимость с MTS. В частности, необходимо, чтобы все драйверы ODBC поддерживали MTS. Если же обращение к хранилищу информации из MTS возможно, но работа транзакций некорректна, значит, диспетчер распределенных транзакций (MS DTC) не запущен или работает некорректно.

Интеграционное тестирование

Теперь, когда приложение работает на одном компьютере, его надо протестировать в распределенной среде. С этого момента начинается сертификация. Чаще всего для MTS-приложений не требуется дополнительный кол, но сама по себе настройка сертификационной среды - отличный способ протестировать комплектацию и развертывание приложения.

Интеграционное тестирование стоит начинать с простейшего варианта развертывания системы, постепенно переходя ко все более сложным. Например, сначала выполните тесты без брандмауэра, а потом добавьте его и повторите все проверки. Кроме того, протестируйте доступ к приложению не только одного клиента, но и сразу нескольких либо с помощью нескольких компьютеров, либо посредством программы, имитирующей их работу.

Методы локального тестирования годятся и для распределенной среды. Большинство средств администрирования Windows NT способны работать как на удаленных, так и на локальных компьютерах. Например, журнал событий для нескольких компьютеров можно просмотреть с одной рабочей станции. Однако существуют и способы, применимые только к распределенным средам. Если приложение работает локально, но создавать удаленные объекты нельзя, вероятно, разорвано соединение с сетью или отсутствует поддержка DCOM. Журнал поможет выявить проблемы защиты, препятствующие созданию объектов и доступу к ним.

Механизм тестирования сетевого соединения зависит от протоколов, используемых в сети. Если для связи посредством DCOM применяется TCP/IP, возможность доступа к компьютеру удастся проверить утилитой Ping (хотя положительный результат и не гарантирует, что DCOM-связь возможна). Утилита DCOM Configuration (DCOMCNFG.EXE) проверяет поддержку DCOM на компьютере. Эта проверка особенно важна в среде Microsoft Windows 95 и Windows 98, так как по умолчанию поддержка DCOM в этих ОС отключена.

Если основные функции COM и MTS не выполняются, следует проверить состояние компьютеров.

Анализ производительности

Как показано на рис. 10.4, определение базовой производительности приложения — одна из первых задач процесса тестирования, после чего можно приступать к разработке тестов ее измерения. Чтобы не выйти за рамки требований, такие тесты надо выполнять в разные моменты процесса разработки. Производительность приложения нужно проверить в тестовой среде, аналогичной эксплуатационной. Если окажется, что требования не соблюдены, следует провести серию экспериментов, выявить причины проблем и устранить их. Этот процесс придется повторять до тех пор. пока производительность приложения не достигнет требуемой.

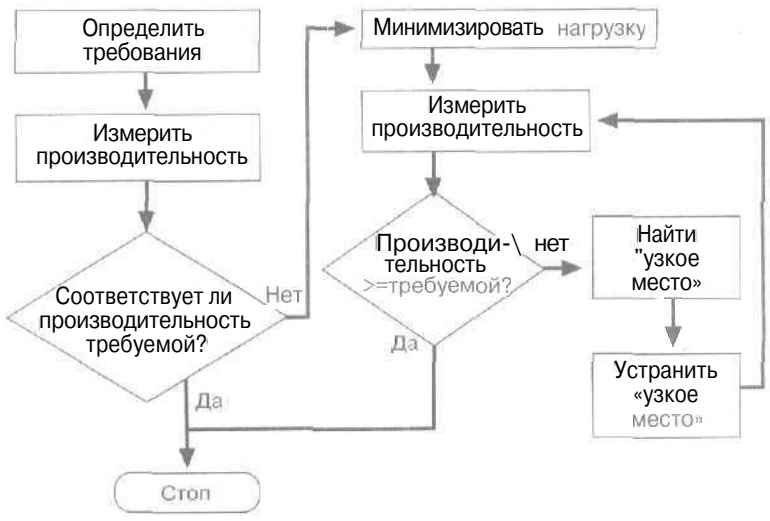


Рис. 10.4. Анализ производительности приложения

Определение требуемой производительности

Требования к производительности необходимо определить заранее, еще до начала разработки и отладки. Для этого следует изучить ограничения, функции, выполняемые приложением, и нагрузку на него. На основании этой информации выбираются соответствующие метрики и определяется необходимая производительность.

Ограничения

Не все в проекте можно изменить для повышения производительности — например, ограничения, связанные с графиком работ, выбором средств или технологий разработки часто непреодолимы. Если приложение следует развернуть к определенной, указанной в контракте дате,

а разработчики имеют большой опыт работы с Visual Basic, но ничего не знают о Visual C++, разработка компонентов на Visual C++ невозможна. Большое влияние оказывают и ограничения, связанные с оборудованием, особенно рабочими станциями пользователей. Но какими бы ни были эти ограничения, их обязательно следует зафиксировать. Если же приведенные в документации ограничения не позволяют достичь удовлетворительной производительности, их следует откорректировать: этим занимаются менеджер программы и заказчик.

В поисках способов повышения производительности допустимо изменение элементов проекта, на которые не наложены ограничения. Например, удастся ли реализовать проект на другом языке программирования? Можно ли использовать иную технологию доступа к данным? Нужны ли транзакции? Не придется ли добавить дополнительные компьютеры? Эти вопросы помогут вам выявить «узкие» места системы.

Сервисы

Обычно один или несколько сервисов приложения отвечают какой-то схеме или сценарию использования. Как правило, каждый сценарий можно описать набором транзакций. Но, даже если транзакции не применяются, система взаимодействует с пользователем. Семантику каждого чувствительного к производительности сценария (описывающего действия пользователя и запускаемые в ответ на них сервисы) надо определить как можно точнее, не забывая изучить сервисы доступа к базе данных и другие системные сервисы. На основе этой информации и проводятся тесты, измеряющие производительность.

После определения сервисов, производительность которых нужно измерить, следует изучить и частоту их использования. Оценив этот параметр, вы сможете создать тесты, соответствующие ожидаемому использованию системы, и повысить точность результатов тестирования производительности.

Нагрузка

Обычно нагрузка на приложение измеряется количеством использующих его клиентов. Кроме того, за единицу измерения иногда берется *время на раздумье* — время, прошедшее с момента ответа на запрос до передачи следующего запроса. Например, если для внесения данных в форму требуется 60 секунд, в этом сценарии время на раздумье равно 60 секундам.

Обязательно учитывайте и изменение нагрузки со временем. Ведь для некоторых приложений она может изменяться. Например, к приложению, обрабатывающему платежи, будет больше обращений в тот период, когда подаются налоговые декларации. Справочному приложению придется нелегко в первую неделю после выпуска новой версии программы. Используя информацию об изменениях нагрузки.

можно выяснить среднюю и пиковую нагрузку. Они позволят выработать требования к производительности.

Что нужно измерять

После изучения ограничений, сервисов и нагрузки следует выработать требования к производительности для вашего приложения. Первым делом надо выбрать соответствующие способы измерения, которые обычно учитывают:

* **общую пропускную способность системы** — измеряется в *транзакциях в секунду* (Transaction Per Second, TPS) для конкретного набора транзакций в условиях определенной нагрузки;
* **время отклика** — интервал между передачей запроса и получением ответа; часто указывается в процентном отношении — например, «отклик на 95% всех запросов должен осуществляться менее, чем за 1 секунду».

Выбрав показатели, нужно сформулировать требования к их значениям, причем последние должны быть реалистичны — вряд ли можно считать серьезными требования типа «как можно быстрее». Пропускную способность системы определяют достаточно простым способом: делят количество клиентов на время, требуемое на размышления. Например, если приложение должно одновременно обслуживать 1200 клиентов, а время на раздумье равно 60 секундам, средняя нагрузка составит 20 TPS (1200/60). При расчете времени отклика необходимо учитывать пожелания пользователей. Например, если после отправки формы они не хотят ждать более 5 секунд, требование ко времени отклика можно определить так: 95% откликов быстрее чем за 5 секунд для модемных соединений со скоростью 28,8 кбит/с.

Измерение производительности

Изучив требования к производительности, выясните, удовлетворяет ли им приложение. Важно исключить из тестов все факторы, не связанные с производительностью. Например, ошибку в программе иногда принимают за проблему с производительностью. Кроме того, имейте в виду, что сравнивать результаты разных тестов можно только в случае корректной работы приложения. Поэтому особенно важно повторить тестирование приложения после внесения изменений в его компоненты. Изучают производительность только после проведения этих тестов. Помните, что изменения затрагивают и оборудование, и сетевой трафик, и конфигурацию программного обеспечения, и системные сервисы и т. д. Естественно, все это нужно контролировать.

Таким образом, при тестировании производительности вам придется фиксировать всю информацию о тестах, включая:

- точную конфигурацию системы, в частности все изменения по сравнению с предыдущим тестированием;
- исходные и обработанные результаты тестов измерения производительности приложения.

Эти сведения помогут выявить не только проблемы с производительностью, но и их возможные причины.

Следует всегда использовать один и тот же набор тестов. Иначе не удастся понять, из-за чего изменились результаты — из-за модификации теста или приложения. Автоматизация тестов позволяет исключить влияние оператора.

Подбор тестов для определения производительности

Во время тестирования измеряют и записывают значения всех метрик. При этом обязательно учитывают и время на раздумье, и информацию о транзакциях, и другие параметры. Тестирование не должно выходить за их рамки и, по возможности, соответствовать реальным условиям. Пусть, например, нужно смоделировать одновременный доступ к приложению нескольких клиентов. Чтобы этот тест был воспроизводимым, можно использовать многопоточную программу, каждый поток которой имитирует одного клиента. Если приложение обращается к базе данных, необходимо, чтобы она содержала близкое к реальному число записей, а тестирование проводилось с произвольными, но корректными значениями полей. Если это условие не выполнено и используется небольшая база данных, из-за кэширования информации результаты не будут иметь ничего общего с реальными. Та же ситуация возникает и в случае ввода или доступа к данным методами, не используемыми на практике (например, при занесении новой информации в ключевое поле в алфавитном порядке).

В состав инструментального пакета MTS Performance Toolkit входит тестовая программа, которая может служить основой для создания тестов. Она поможет вам познакомиться с методами измерения TPS и времени отклика, а также со способами моделирования множества клиентов средствами нескольких потоков. Обычно такие программы нуждаются во входных данных, таких как типичный набор транзакций, время на раздумье, количество клиентов и т. д., однако их можно включить непосредственно в тест.

После создания тестовой программы следует задокументировать неизменяемые условия запуска тестов (например, входные параметры тестов). Также в этот документ стоит внести конфигурации тестовых компьютеров и способы настройки «чистой» (не содержащей изменений, внесенных предыдущими тестами) базы данных. Обычно тестовые программы запускают на отдельном компьютере, чтобы добиться лучшего соответствия производственной среде.

Эталонная производительность

После определения требований к производительности и разработки соответствующих тестов следует провести первую проверку. Чем больше сертификационная среда похожа на производственную, тем вероятнее, что после развертывания приложение будет корректно работать. Поэтому очень важно с самого начала тестирования создать сертификационную среду, максимально приближенную к реальной.

Если повезет, и начальное тестирование покажет, что требования к производительности выполняются, дополнительной работы не потребуется. Чаще всего первые тесты неудовлетворительны, но документация и их результаты станут основой ваших дальнейших действий.

Выявление и устранение проблем

Если требования к производительности не выполняются и после снижения требований к приложению (см. главу 2) или если такое снижение недопустимо, следует проанализировать результаты тестов, выявить проблемы и сформулировать предположение об их причинах. Иногда данных не хватает; в этом случае приходится применять дополнительные средства, измеряющие производительность MTS-приложений.

- **Microsoft Windows Task Manager** — на вкладке Performance окна Task Manager (рис. 10.5} отображается информация об использовании памяти и процессора на компьютере. Все эти данные могут пригодиться для выявления причин низкой производительности.

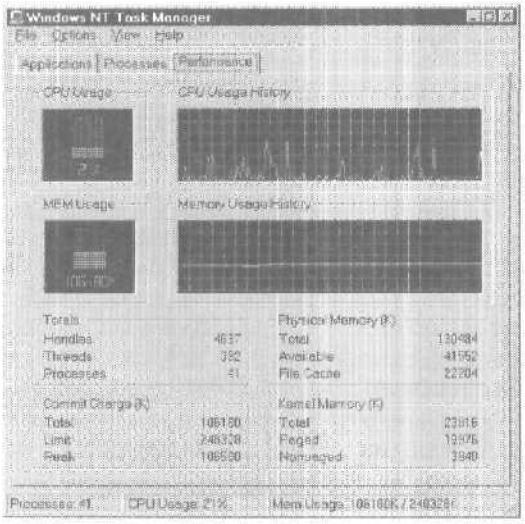


Рис. 10.5. Вкладка Performance окна утилиты Task Manager

- **Панель Transaction Statistics утилиты MTS Explorer** — пригодится для сбора информации о производительности приложения, если оно использует транзакции (рис. 10.6). Позволяет определить количество переданных и прерванных во время теста транзакций, а также минимальное, максимальное и среднее время отклика. Эти значения отклика верны только для транзакций, а не для всего сценария, и при их определении не делается различий между разными типами приложений. Тем не менее эту информацию все же можно использовать для изучения влияния транзакций на производительность приложения.

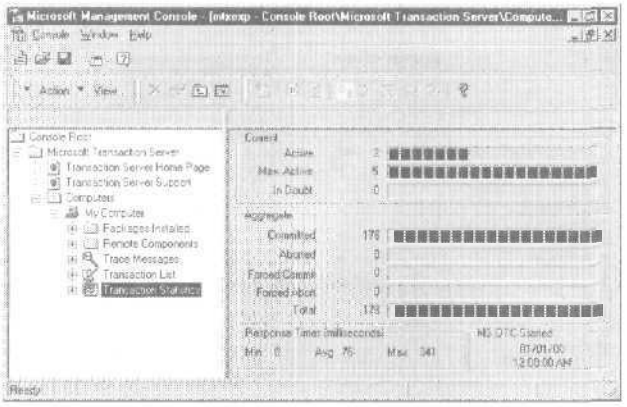


Рис. 10.6. Окно Transaction Statistics утилиты **MTS Explorer**

- **Microsoft Windows Performance Monitor** (PerfMon) — это приложение (рис. 10.7) очень полезно для выявления падения производительности и его причин. Позволяет измерить множество различных параметров, в том числе пропускную способность, длину очереди, перегрузку и другие параметры, связанные как с приложениями, так и с оборудованием.
- Visual Studio **Analyzer** — эта утилита из состава Microsoft Visual Studio 6.0 также измеряет различные параметры системы. Кроме того, она позволяет отслеживать события, связанные с компонентом приложения или взаимодействием между такими компонентами. Утилита способна перехватывать как COM-, так и MTS-события, что помогает выявить проблемы, связанные с некорректной реализацией компонентов (например, так обнаруживают медленно выполняющиеся методы).

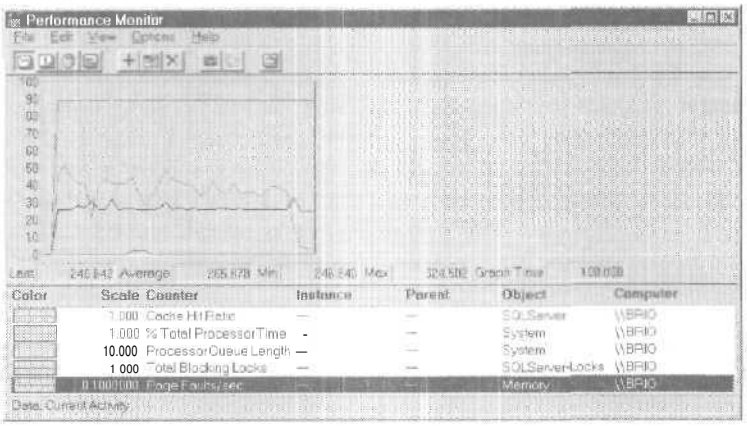


Рис. 10.7. Microsoft Windows Performance Monitor

Примечание Исчерпывающее описание Visual Studio Analyzer см. в документации Visual Studio.

Хотя в данный момент в MTS отсутствуют встроенные средства измерения производительности, определить степень использования памяти, дисков и памяти в этой среде все же можно. Для выявления причин проблем годятся и другие приложения (например, SQL Server).

Причины большинства проблем производительности MTS-приложений — недостаточный объем памяти. малая мощность процессора и низкая пропускная способность дисковой подсистемы. Все их можно выявить средствами счетчиков, описанных в табл. 10.1.

Табл. 10.1. Счетчики производительности

Счетчик	Описание	Возможные проблемы
Memory: Page Faults/Second	Число страниц, отсутствующих в физической памяти	Частота, превышающая 5, указывает на недостаток памяти
Physical **Disk: % Disk Time**	Доля времени, когда диск занят операциями чтения и записи	Значение этого показателя, большее 85%, особенно если значение счетчика Average Disk Queue Length превышает 2, указывает на малую пропускную способность дисковой подсистемы (если, конечно, повышенная дисковая активность не вызвана нехваткой ОЗУ)

(*продолжение*)

Physical Disk: Average Disk Queue Length	Среднее число запросов чтения и записи за определенный интервал	Значение этого показателя, большее 2, особенно если значение счетчика % Disk Time превышает 85, указывает на малую пропускную способность дисковой подсистемы (если, конечно, повышенная дисковая активность не вызвана нехваткой ОЗУ)
System: % **Total** Processor **Time**	Доля времени, когда процессор занят	Значение этого параметра, превышающее 80%, указывает на недостаточную производительность процессора
System: **Processor Queue Length**	Количество потоков, ожидающих своего выполнения в очереди	Значение, превышающее 2, указывает на недостаточную производительность процессора
SQL Server: Cash Hits Ratio	Доля выборки данных из кэша	Если значение этого параметра ниже 80%, SQL Server не хватает памяти
SQL Server Locks: Total Blocking Locks	Числоблокировок БД, останавливающих другие процессы	Высокое значение указывает на проблемы с базой данных

Примечание Существуют и другие счетчики производительности. Более подробно об этом — и документации Windows NT Performance Monitor и в книге «Microsoft Windows NT Workstation 4.0 Resource Kit».

Собрав необходимые данные, можно попробовать выявить проблемы и найти их причины, после чего пересмотреть решение. Иногда этот процесс довольно прост, но чаще информации о производительности недостаточно для устранения проблем. В этом случае следует изменить какой-либо параметр тестовой среды и повторить измерения. Если производительность осталось прежней или ухудшилась, стоит восстановить предыдущее состояние среды и попробовать изменить какой-либо другой параметр.

Типичные проблемы

Опытные разработчики, занимающиеся оптимизацией приложений, хорошо знакомы с типичными проблемами. В частности, специальная группа отдела COM компании Microsoft описала типичные «узкие» места MTS-приложений в документации MTS Performance Kit.

Этот раздел посвящен проблемам, вызывающим спад производительности, и способам их возможных решений.

Проблемы работы SQL Server

Для повышения производительности соединений SQL Server можно настраивать как коммуникационный протокол и как процедуру регистрации в системе.

Клиентский протокол

По умолчанию в SQL Server применяются именованные каналы. Однако производительность и масштабируемость приложения становится лучше при использовании TCP/IP. Для этого средствами программы SQL Server Setup надо включить поддержку TCP/IP. Затем для каждой системы, которая запускает компоненты, обращающиеся к SQL Server, на вкладке Net Library утилиты SQL Client Configuration Utility выберите в качестве протокола по умолчанию **TCP/IP** Sockets. Чтобы изменения вступили в силу, остановите и перезагрузите службу SQL Server.

Регистрация

Если для доступа к SQL Server используется регистрационная запись администратора, при каждой транзакции происходит обращение к базе данных master. Однако она не всегда требуется приложениям. Чтобы исключить издержки, связанные с доступом к БД master, стоит создать для приложения специальную регистрационную запись, задав для нее в качестве БД по умолчанию базу, к которой обращается приложение. Эту регистрационную запись можно использовать для всех источников данных.

Проблемы доступа к данным

Доступ к данным — затратный процесс, но всегда есть способы повысить его производительность. В этом разделе мы расскажем о некоторых проблемах, связанных с обращением к данным.

Файловые имена источников данных

Имя источника данных — простой способ определения базы данных, к которой нужно получить доступ, Однако производительность этого метода очень низка, так как системе приходится постоянно открывать файл описания источника данных и считывать из него параметры соединения с базой данных. Поэтому применяют методы, описанные ниже.

- **Пользовательские и системные DSN.** Их создают с помощью утилиты ODBC Data Source Administrator. Пользовательские и системные DSN усложняют администрирование приложения, так как в этом случае .dsn-файл недостаточно просто скопировать из одного места в другое. Однако повышение производительности обычно «перевешивает» этот недостаток.

- **Встроенные строки команды подключения.** Компоненты модифицируют так, чтобы командная строка, описывающая соединение, была встроена непосредственно в приложение, При этом отпадает необходимость использования DSN при создании объектов ADO **Connection и Recordset.** Такой подход не слишком сильно влияет на администрирование приложения, но, возможно, вам придется собирать заново все компоненты приложения при каждом изменении конфигурации баз данных.

Производительность ADO и OLE DB

Ранние версии ADO и OLE DB практически не поддавались масштабированию при применении многопоточных клиентов, использующих пулы соединений, в частности на многопроцессорных компьютерах. Этот недостаток значительно снижал производительность MTS-компонентов и ASP-страниц. В Microsoft Data Access Components (MDAC) 2.0 и более поздних версиях эта проблема решена, поэтому, по возможности, следует использовать именно эти версии. Кроме того, существуют и другие решения:

- изменить приложение, разместив MTS-компоненты на нескольких серверах;
- использовать в важнейших компонентах прямые вызовы ODBC.

Позднее связывание с компонентами доступа к данным

Как мы уже говорили в предыдущих главах, позднее связывание с COM-компонентами намного медленнее связывания по виртуальной таблице и раннего связывания, так как при этом все вызовы методов клиента осуществляются через интерфейс **IDispatch.** Кроме того, производительность падает из-за необходимости упаковки параметров метода в специальную структуру, необходимую для вызова функции **Invoke.** Поэтому, если используемые вами средства разработки поддерживают раннее связывание и связывание по виртуальной таблице, пользуйтесь именно ими.

Журнал диспетчера распределенных транзакций

Диспетчер MS DTC протоколирует все транзакции. Поэтому, если журнал находится на медленном диске, производительность приложения может упасть. Таким образом производительность приложения, использующего транзакции, можно улучшить, выделив для хранения журнала диспетчера транзакций быстрый диск с высокопроизводительным контроллером.

Несколько одновременно работающих диспетчеров транзакций

По умолчанию каждый сервер, работающий с SQL Server или MTS, использует локальную копию диспетчера транзакций. Издержки, связанные с взаимодействием между ними, могут снизить производи-

тельность приложения. Эта проблема решается посредством использования одного диспетчера. Для этого надо остановить сервис MS DTC, удалить его с локального компьютера и затем с помощью апплета MS DTC Панели управления выбрать компьютер, на котором он должен выполняться.

Другие проблемы MTS

Полный список возможных проблем приведен в документации MTS Performance Toolkit. Мы же опишем только две из них: доступ к реестру и динамическое выделение памяти.

Доступ к реестру

Обращения к реестру сопровождаются значительными накладными расходами, Поэтому, если информация о параметрах приложения хранится в реестре, она должна считываться только один раз. Далее компоненты могут хранить информацию с помощью диспетчера общих свойств, что сделает ее доступной всем объектам процесса, а приложения — в локальных переменных.

Динамическое выделение памяти

Еще один затратный процесс, который, по возможности, должен быть исключен — динамическое выделение памяти. В этом отношении особенно плох Microsoft Visual Basic 5, выделяющий и освобождающий память даже тогда, когда ее можно использовать повторно.

Масштабирование производственной среды

В этом разделе мы рассмотрим некоторые распространенные конфигурации среды развертывания приложений, использующих IIS для создания клиентского интерфейса, MTS как среду выполнения бизнес-объектов и SQL Server в качестве сервера БД. Большая часть этой информации применима и к приложениям, не использующим IIS.

Конфигурация 1: один узел

В этой конфигурации (рис. 10.8} IIS, SQL Server и MTS-приложение установлены на одном сервере, или *узле,* с которым взаимодействуют все клиенты. Развертывание и администрирование такой системы не представляет затруднений. Кроме того, не возникает никаких проблем, связанных с защитой или брандмауэрами, так как все серверные приложения выполняются на одном компьютере.

Производительность такой системы высока, поскольку межсерверные вызовы отсутствуют. Кроме того, ее можно дополнительно повысить, если MTS-приложение работает как внутрипроцессное, Однако в этом случае MTS-приложения не могут использовать ролевую защиту.

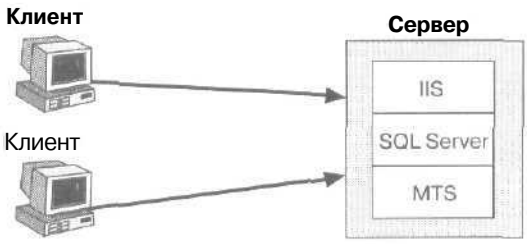


Рис. 10.8. Развертывание приложения на одном узле

У такой конфигурации есть и недостаток: по мере роста числа пользователей ее производительность будет падать. Конечно, можно добавлять память, устанавливать высокопроизводительные накопители или процессоры, но рано или поздно будет достигнуто предельное число этих устройств и придется переходить на многосерверную конфигурацию. Таким образом, конфигурация с одним узлом подойдет только для приложений, обслуживающих небольшое число пользователей.

Конфигурация 2: IIS на отдельном узле

В этой конфигурации IIS запускается на одном сервере, а SQL Server и MTS-приложения — на другом. Web-клиенты связываются с сервером IIS. Такая конфигурация быстрее выполняет запросы статических страниц и простых ASP-страниц, так как IIS выделен большой объем ресурсов. Производительность бизнес-объектов и объектов данных также должна быть высокой, поскольку MTS и SQL Server находятся на одном компьютере. Кроме того, из-за применения двух серверов к такой системе может подключиться больше пользователей, чем к одному узлу.

Но и у этого подхода есть недостатки. Обращения ASP-страниц к MTS-компонентам связаны с запросами к другому серверу и, следовательно, медленнее локальных вызовов и могут привести к падению производительности. Дополнительные трудности могут возникнуть в случае, когда эти два сервера расположены по разные стороны брандмауэра.

Конфигурация 2 подходит для приложений, состоящих в основном из статических Web-страниц и простых ASP-страниц, не очень интенсивно использующих MTS-компоненты и базы данных SQL Server. Если сервер IIS не справляется с возросшей нагрузкой, его можно реплицировать на несколько компьютеров, а для распределения запросов воспользоваться сервисом распределения нагрузки Windows Load Balancing Service.

Самый крупный недостаток этой конфигурации состоит в том, что все серверы IIS взаимодействует с одним сервером MTS/SQL Server. При увеличении количества пользователей это может привести к снижению производительности.

Конфигурация 3: SQL Server на отдельном узле

В третьей конфигурации IIS и MTS-приложения запускаются на одном сервере, а SQL Server — на другом (рис. 10.9). Клиенты подключаются к IIS/MTS-серверу. Так как вызовы между IIS и MTS происходят в пределах одного сервера, отклик на запросы Web-страниц и компонентов быстр. Если же не требуется защита и изоляция процессов, что позволяет запускать MTS-приложения как внутрипроцессные, производительность будет еще выше. Кроме того, приложение может обслуживать большее, чем в случае одного узла, число клиентов.

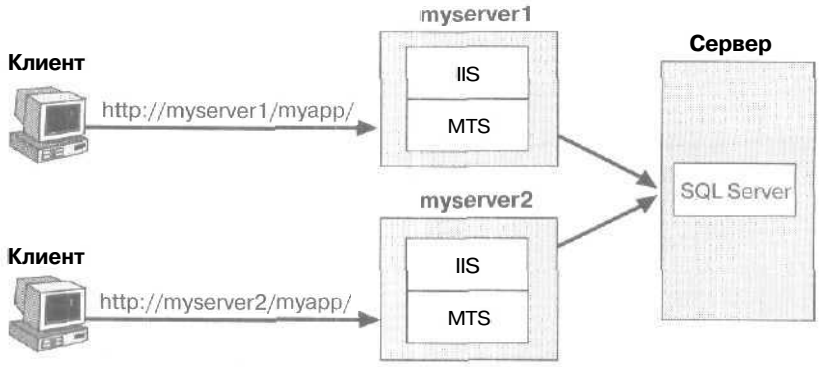


Рис. 10.9. Использование WLBS для распределения запросов клиентов

Недостаток такой конфигурации в том, что обращения к SQL Server адресуются другому серверу. Однако, поскольку такие вызовы обычно связаны с большим объемом вычислений и операций ввода/вывода, они в любом случае будут медленными (по сравнению с вызовами методов компонентов). Поэтому издержки при обращениях к другому серверу не приведут к существенному снижению производительности. В этом случае важно проверить, выполняются ли необходимые требования к производительности. Может потребоваться и корректировка тайм-аута пула соединений ODBC, который должен учитывать длительность установки сетевого соединения.

Как и в случае конфигурации 2, вероятны трудности, если серверы расположены по разные стороны брандмауэра. Производительность можно повысить за счет использования TCP/IP в качестве ком-

муникационного протокола SQL Server. В зависимости от типа брандмауэра и корпоративной политики в отношении проходящего через него трафика, доступ к SQL Server через брандмауэр по протоколу TCP/IP может стать оптимальным решением, или же вообще окажется невозможным. В этом случае лучше всего применить пакетные брандмауэры, позволяющие открывать спецификации TCP/IP-портов.

Конфигурация 3 подходит для приложений с низкой и умеренной нагрузкой, обращающихся к базам данных на выделенном сервере или на сервере, который должен быть изолирован от Интернета. Приложения, работающие в условиях высокой нагрузки, могут быть развернуты в немного видоизмененной среде: IIS/MTS-сервер реплицирован на несколько серверов, а для распределения запросов используется WLBS. Отметим, что, поскольку запросы к базе данных адресуются одному и тому же серверу, рано или поздно будет достигнут предел возможностей одновременного обращения, а значит, и числа пользователей.

Конфигурация 4: каждая база данных на отдельном узле

Обязательно наступит время, когда приложения или базы данных, размещенные на одном компьютере, не смогут обрабатывать запросы пользователей с приемлемой производительностью. В этом случае приложения и базы данных придется разделить на части.

Существует несколько способов такого разделения. В конфигурации 4 используемые приложением БД SQL Server размешены на нескольких серверах (рис. 10.10). На каждом сервере можно разместить несколько баз данных, но каждая база данных может быть помещена только на один сервер. MTS-приложения устанавливаются на компьютерах, находящихся как можно ближе к используемой ими информации. Такое разделение несколько усложняет развертывание системы — ведь теперь каждый компьютер, обращающийся к компонентам, должен знать, где размещен нужный ему компонент.

В этой конфигурации большую часть транзакций можно распределить (если приложение использует транзакции, обращающиеся к нескольким базам данных). Такие распределенные транзакции медленнее транзакций на одном компьютере, так как в процессе их синхронизации диспетчер должен подключаться ко всем серверам. Однако такое увеличение длительности транзакции приемлемо, ведь среднее время ее выполнения при высокой нагрузке системы, скорее всего, уменьшится. Кроме того, так как IIS запущен на отдельном компьютере, проблемы конфигурации 2, связанные с производительностью и брандмауэрами, актуальны и для этого случая. Напомним. что увеличение времени выполнения вызовов методов и сложности

администрирования систем с брандмауэром, как правило, допустимы — ведь при этом увеличивается и число обслуживаемых пользователей. Не забывайте, что каждую рассматриваемую топологию следует тщательно протестировать на соответствие выработанным ранее требованиям к производительности.

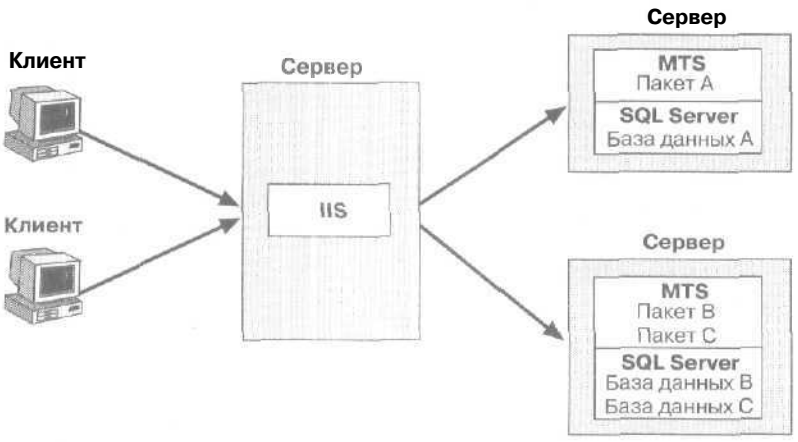


Рис. 10.10. Размещение баз данных SQL Server на нескольких узлах

Конфигурация 5: распределенная база данных

Ситуация усложняется, если приложение использует одну базу данных, таблицы которой нельзя преобразовать в несколько баз данных, или если приложение использует несколько баз данных, каждая из которых расположена на отдельном сервере, но требования к производительности не выполнены. В этом случае нужно создать несколько копий базы данных и разместить их на разных компьютерах. Синхронизируют информацию с помощью средств репликации SQL Server.

Конфигурацию 5 стоит попробовать, если отдельные группы пользователей обращаются в основном к части информации базы данных – например, если информацию могут обновлять пользователи только определенного региона. Периодически такие обновления будут копироваться в главную базу данных, поэтому даже пользователи, находящиеся вне своего региона, получат доступ ко всем необходимым им данным. Для предотвращения конфликтов между изменениями, внесенными в разных местах, ключ (значение, идентифицирующее запись) должен содержать идентификатор сервера.

В одном из вариантов этой конфигурации разделяется только база данных, а приложения остаются в неизменном виде. В этом случае все пакеты MTS-приложений устанавливаются на один сервер, а в

компоненты включают код, определяющий базы данных, используемые во всех операциях. В большинстве ситуаций такой способ не годится, так как при каждом изменении конфигурации базы данных приходится изменять и компоненты.

В другом варианте MTS-приложение реплицируется на все серверы баз данных. Чтобы определить, допустимы ли входные данные для компьютера, компоненты используют конфигурационную информацию сервера. (В противном случае пришлось бы для каждого сервера писать отдельные компоненты.) Клиентские компьютеры и IIS-серверы конфигурируются так, чтобы работать с определенным сервером БД. Например, все клиентские компьютеры организации могут связываться с локальным сервером IIS, который, в свою очередь, будет направлять запросы на специальный сервер, на котором установлены MTS-приложение и база данных SQL Server. Такой вариант (рис. 10,11) отлично подходит для сред, распределенных по географическим регионам.

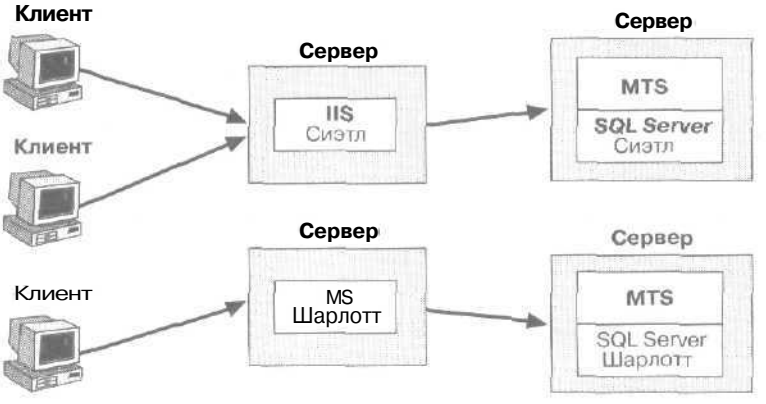


Рис. 10.11. Использование выделенного сервера IIS для доступа к базе данных, распределенной по географическому признаку

Конфигурация 6: распределенное приложение

Возможно, понадобится разделить не только базу данных, но и приложение. В такой конфигурации пакеты MTS устанавливаются с разными базами данных. Проще всего разделить приложение на основе его функций. Например, на рис. 10.12 показан бизнес-объект, установленный на сервере US, и объект данных, установленный поблизости от используемой им информации. Производительность такой системы должна быть высока, так как ASP-страницы обращаются к

бизнес-объектам, а объекты данных — к базам данных SQL Server локально. Только обращения бизнес-объектов к объектам данных выходят за рамки одного сервера.

Комбинируя элементы описанных ранее конфигураций, можно разработать множество вариантов данной конфигурации. Однако при использовании распределения базы данных, репликации, балансировки нагрузки и т. п. ее развертывание и администрирование будет постепенно усложняться. Но ее преимущество — поддержка большого количества пользователей — несмотря ни на что, останется преимуществом. Кроме того, при использовании сложных конфигураций нужно обязательно тестировать систему методами, описанными в главе 13, чтобы выяснить, соответствует ли ее производительность требованиям заказчика.

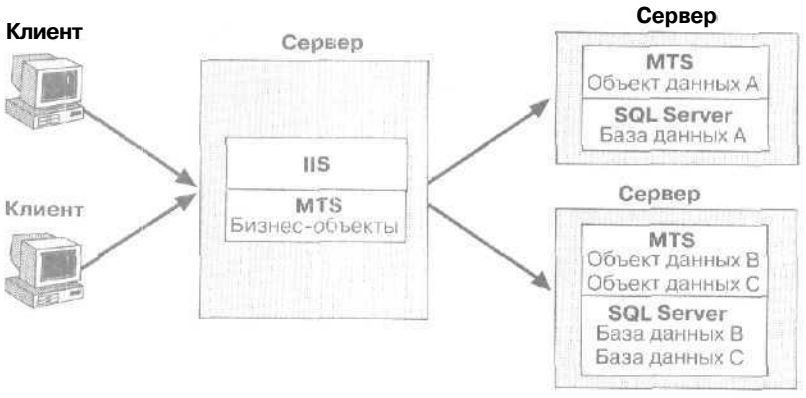


Рис. 10.12. Распределенное приложение

Отказоустойчивость

Если приложение должно быть надежным и всегда доступным, его можно установить на кластере — группе физических компьютеров, действующих как один логический компьютер. Один из компьютеров такой группы работает всегда, Если он выйдет из строя, его заменит другой компьютер кластера. Для управления кластерами из двух компьютеров можно воспользоваться ПО Microsoft Cluster Server (MSCS).

Устранение ошибок

Информация этого раздела позаимствована из курса MSF 1516 «Principles of Application Development».

Ошибками называют все, что должно быть найдено и исправлено до выпуска приложения. Бытует мнение, что ошибки — это дефекты,

неисправности и т. п. Это не так. Дефекты — только один из видов проблем, возникающих при использовании разрабатываемого приложения, в число которых входят:

- необходимость исправлений или улучшений;
- предложения, выходящие за пределы возможностей данной версии приложения;
- проблемы, возникающие из-за тех или иных предпочтений пользователей;
- неизбежные последствия принятых решений;
- дефекты.

Процессы отслеживания ошибок охватывают такие вопросы, как передача сообщений об ошибках, расстановка их по приоритетам, распределение среди разработчиков, исправление и закрытие. Основной принцип стадии «Стабилизация» — нужно всегда точно знать статус приложения. Информацию об этом статусе вы получите как раз в процессе отслеживания ошибок. Ошибки следует найти и исправить, поэтому на фазе «Стабилизация» придется принимать много компромиссных решений — в частности, какие ошибки будут исправлены, а какие нет. Проектная группа должна классифицировать ошибки по их приоритету и опасности, на основе чего выбираются методы их устранения.

Отслеживание ошибок

Прежде чем начать «охоту» за ошибками, нужно организовать какое-нибудь хранилище, например, базу данных, в которую будет заноситься информация о найденных ошибках. Новые ошибки должны помечаться как активные.

Получив сообщения об ошибках, начальник группы разработки должен расставить их по приоритетам и распределить их среди своих сотрудников. На стадии «Стабилизация» он может исправлять незамеченные ранее ошибки сам.

Разработчики должны исправить ошибки, после чего протестировать исправления, чтобы убедиться, что известные ошибки устранены, а новые не появились. После устранения активной ошибки ее статус меняется и зависимости от использованного решения. Затем группа тестирования должна проверить качество исправления и убедиться, что не появилось новых ошибок. Если новые ошибки все-таки возникли, тестеры должны занести их в базу данных и, таким образом, запустить цикл исправления заново. Если же решение проблемы окажется неполным, нужно снова активировать ошибку. Ее можно закрыть только после окончательного устранения. Иногда ошибку не исправляют сразу, а заносят информацию о ней в документацию при-

ложения. Работы же по исправлению откладывают и выполняют их после выпуска программы.

Классификация ошибок

Классификация поможет определить приоритет и опасность ошибок. При этом следует изучать две важных стороны проблемы: ее серьезность (влияние неисправленной ошибки на работу приложения) и приоритет (важность исправления ошибки для сохранения стабильности приложения). Надо не только найти ошибку и сообщить о ней, ее еще нужно классифицировать.

Перечислим основные уровни серьезности проблемы.

- **1 — отказ системы.** Ошибки этого класса приводят к отказам системы, ее зависаниям и прочим остановкам работы, сопровождающимся потерями данных.
- **2 — крупные проблемы.** Ошибки этого класса являются серьезными дефектами приложения, но не всегда приводят к отказу всей системы и потере данных.
- **3 — незначительные проблемы.** Ошибки этого класса — дефекты приложения, редко приводящие к потере данных.
- **4 — тривиальные.** Ошибки этого класса являются незначительными косметическими проблемами, которых большинство пользователей не замечает.

Теперь рассмотрим типичные приоритеты.

- **1 — высший.** Фатальная ошибка. Приложение нельзя сдавать в эксплуатацию, так как разработчики не могут достичь следующего промежуточного этапа.
- **2 — высокий.** Крупная проблема. Приложение нельзя сдавать в эксплуатацию, но разработчики могут достичь следующего промежуточного этапа.
- **3 — средний.** Приложение можно сдавать в эксплуатацию, и разработчики могут достичь следующего промежуточного этапа. Приоритет таких ошибок достаточно низок, поэтому их, как правило, исправляют, только если на это есть время, а риск не уложиться в сроки низок.
- **4 — низкий.** К ошибкам с низким приоритетам обычно относятся пожелания расширения или улучшения функций приложения и ошибки, которые даже не стоит исправлять.

Устранение ошибок

Исправление ошибок не завершает их поиск. Это только промежуточный этап на пути к устранению ошибок, которое происходит после того, как тестеры проверят, что исправление полностью устраняет известную проблему и не создает новых.

Ошибки могут быть следующих типов.

- **Исправленные** — разработчики исправили ошибку, протестировали исправление, проверили код, поправили номер версии программы и отправили ее тестеру, сообщившему об ошибке.
- **Повторные** — ошибка, совпадающая с уже находящейся в базе данных. Для повторной ошибки в БД записывается ссылка на первую запись об ошибке, после чего повторная ошибка считается разрешенной и закрытой.
- **Отложенные** — в текущем выпуске ошибка не будет исправлена, но в следующих версиях программы такая работа может быть проведена. Такое обозначение необходимо, если ошибку нужно исправить, но в данный момент на это нет времени.
- Проектные — поведение программы, классифицированное как ошибка, является корректным и соответствует функциональным спецификациям.
- **Невоспроизводимые** — разработчики не могут проверить наличие ошибки из-за невозможности воспроизвести ее.
- **Неисправляемые** — ошибка не будет исправлена в текущей версии приложения, так как разработчики считают, что эффект от ее исправления не стоит затраченных на это усилий, или менеджерам и заказчикам она кажется малозначимой.

Резюме

Разрабатывать сложные приложения очень трудно. Не стоит усиливать эти трудности, полагаясь на неадекватный производственный цикл. В этой главе мы обсудили преимущества грамотно организованного производственного цикла (стабильность и управляемость), в рамках которого группа последовательно переходит от среды разработки к тестовой, а затем — к сертификационным и производственным средам. Мы также изучили некоторые особенности тестирования приложений масштаба предприятия.

Прежде чем сдавать приложение в эксплуатацию, следует проанализировать его производительность. В этой главе вы узнали, как определить требования к производительности, выбрать соответствующие метрики и подобрать тесты для ее измерения. Мы советуем сначала определить эталонную производительность, с которой потом можно будет сравнивать результаты тестов переработанного приложения. Также мы обсудили способы масштабирования типичного Web-приложения в производственной среде.

И наконец, мы изучили процесс управления ошибками, разобрались, как их отслеживать, классифицировать и исправлять,

Закрепление материала

1. Перечислите этапы производственного цикла.
2. В чем состоят преимущества производственного цикла?
3. Какие вопросы нужно изучить при определении требований к производительности?
4. Что такое эталон производительности?
5. Назовите две категории тестов.
6. Перечислите этапы управления ошибками.

Практикум 8. Тестирование RMS

– Боже мой!

Дэн Шелли не поверил своим глазам, когда на экране появилось еще одно сообщение о сетевой ошибке.

– Это — уже третье за утро! Что случилось?

Он открыл папку My Computer и проверил сетевые диски, пытаясь понять, какой из серверов отказал:

– Бог мой! Это же новый сервер, который купили для AutoCAD. Кто-нибудь мне скажет, в чем дело?

Он немного подумал, затем встал, взял свой блокнот и добавил:

– Наверно, лучше пойти *к* серверу и посмотреть, что произошло. Все равно я ничего не узнаю, если буду сидеть здесь.

Подойдя к большой, застекленной комнате, в которой располагались серверы фирмы «Фергюсон и Барделл», он увидел, что Тим О'Брайан, сетевой менеджер, работает на новеньком сервере, четырехпроцессорном монстре с дисковым массивом RAID 5 и огромным объемом памяти. Тим лихорадочно набирал команды, а стоящий рядом Билл Парди давал ему советы. Причем Тим выглядел совершенно расстроенным.

— Опять не то, Билл, — сказал он. — Из-за тех программ, что установил Сэм, разрушился реестр. Придется восстанавливать его из резервной копии.

– Привет, — сказал Дэн с притворной приветливостью.

– У нас небольшие проблемы с новым CAD-сервером и мы пытаемся их решить, — не давая Тиму раскрыть рта, ответил Билл.

— Правда? — спросил Дэн, не подав вида, что неисправность сервера для него не новость. — Я и не подозревал, что ты консультируешь Тима по работе с сетью. А из-за чего проблемы?

Прежде чем Билл ответил, в разговор вступил Тим:

– Ну, Билл, перестань. Дэн нас не убьет. Он *захочет* это сделать, но ведь не сделает же!

Затем, повернувшись к Дэну, он добавил:

– Дело в том, что разработчикам понадобилось протестировать свои компоненты с помощью сценария, моделирующего разную нагрузку. А так как мы говорили о приобретении такого же сервера и для RMS, они попросили установить эти компоненты на CAD-сервере. Я был против, но они сказали, что на их компьютерах все работает замечательно, и никто ничего не заметит. В общем, я поддался на уговоры и разрешил им поработать здесь. Установка прошла отлично, но когда они запустили свои тесты, сервер «упал». Мы «оживляли» его дважды, но, похоже, в третий раз нам это не удастся. Я пытаюсь восстановить реестр, но если не получится, придется восстанавливать всю последнюю резервную копию.

– Если я правильно понял, — не повышая голоса, обратился Дэн к Биллу. — твои разработчики захотели протестировать компоненты, и ты посоветовал им сделать это на *производственном* сервере?

— А ты им *разрешил?* — уже повернувшись к Тиму, закончил он.

Тим угрюмо кивнул, а Билл попытался что-то сказать в свое оправдание, но Дэн прервал его, подняв руку,

— Ничего не хочу слышать, Билл. Мы поговорим об этом в другое время и в другом месте. Сейчас же мы пойдем отсюда, чтобы Тим побыстрее восстановил сервер. Мне уже надоели постоянные сообщения об ошибках.

— А почему ты получаешь сообщения об ошибках? — удивленно спросил Тим. — Ты же не пользуешься этим сервером.

— А вас не удивило, что я появился здесь сразу же после сбоя сервера? — спросил Дэн. — Может быть вы подумали, что это простое совпадение? Я не афиширую тот факт, что каждую неделю связываюсь со всеми нашими серверами — тестирую производительность и проверяю соблюдение политики безопасности. Сегодня как раз очередь CAD-сервера.

Тут все трое заметили Джима Стюарта, прислонившегося кдвери. Он выглядел спокойным, но мрачным.

— Джим! Не часто тебя увидишь в такой глуши. Заходи, — сказал Дэн, загораживая экран с сообщениями об ошибках. — Что ты здесь делаешь?

– Подсчитываю, — сухо ответил Джим.

«Вот и неприятности», — подумал Дэн, а Тим наивно спросил:

– Что подсчитываешь?

– Подсчитываю, сколько стоит простой сервера, — повысив голос, ответил Джим. — Это можно сделать двумя или тремя способами. Рассказать какими?

«Необязательно», — подумал Дэн, но, решив, что отступать некуда, сказал:

– Конечно, Джим, расскажи.

Тот начал тоном школьного учителя:

— Во-первых, можно применить метод истраченной не по назначению зарплаты. Он самый простой. Ведь я знаю почасовую оплату сотрудников каждого отдела, поэтому ее надо просто умножить на время, когда сервер не работал. Во-вторых, метод потерянной прибыли...

Джим расхаживал по комнате туда-сюда.

«Ненавижу, когда он так ходит», — подумал Дэн, но промолчал.

– Потерянную прибыль подсчитать сложнее, потому что она меняется каждый месяц. Но я могу взять годовое значение, разделить его на общее число рабочих часов — получится почасовое значение, которое я и применю для расчета. И наконец, утраченные возможности. Сколько мы заработали бы за время простоя сервера, если бы продали наши ресурсы? Этот расчет, конечно, теоретический, но часто дает отличный результат.

К этому моменту все трое слушателей выглядели довольно жалко. И хотя Дэн не хотел ничего говорить, ему пришлось спросить;

– И какие цифры у тебя получились?

Джим остановился, посмотрел сначала на Дэна, затем на Билла и Тима и раздраженно сказал:

— Получается вот что. Потерянная зарплата составляет 1 300 долларов в час. Неполученная прибыль — между 4 и 5 тысячами долларов. А утраченные возможности еще больше. Итог таков: если вы не исправите сервер к концу дня, «Фергюсон и Барделл» потеряет как минимум 25 000 долларов. Что вы на это скажете?

Дэн на удивление спокойно ответил:

— Джим, над проблемой работает наш лучший сотрудник. Что касается Билла, то с ним я собирался поговорить позже, но, думаю, что сделаю это прямо сейчас и в твоем присутствии. Поэтому давайте перейдем в соседнюю комнату, а Тим попробует прекратить утечку денег.

Что нам нужно и зачем

Когда они перешли в соседнюю комнату, Джим потребовал объяснений:

— А теперь расскажите мне, что случилось. Почему сервер не работает?

Дэн кратко описал ситуацию, и Джим обратился к Биллу:

— У твоих людей самая дорогая техника во всей компании! И тебе ее не хватает?

— Не хватает, — резко ответил Билл. Тирада Джима вывела его из равновесия, поэтому он решил высказаться.

– Раньше все было по-другому. Когда-то мы писали автономные приложения, которые удавалось компилировать и тестировать на на-

ших же компьютерах. Большие серверы нам требовались только для ускорения компиляции, чтобы не наблюдать все утро, как компьютеры медленно «перемалывают», ну, например, код на Clipper. После компиляции мы тестировали программы локально, применяя базы данных, размещенные на наших же компьютерах. Затем мы помешали приложение в сеть и объясняли всем, как до него добраться и запустить. Для особых неумех приходилось писать командные файлы и включать приложение в их сценарии регистрации.

Но сейчас все изменилось. Наши приложения разделены на компоненты, одни из них выполняются на сервере, а другие — на компьютерах пользователей. Мы применяем базы данных, работающие на серверах, связанных, в свою очередь, с другими серверами, на которых работают Web-приложения, расположены брандмауэры, электронные магазины, хранилища данных и многое другое.

Билл подошел к доске, взял маркер и нарисовал несколько квадратов. Затем он соединил их линиями и обратился к Дэну:

— А сейчас к этой головоломке добавляется еще одна часть, вернее, *несколько* частей. Мы пишем многоуровневое приложение, значит, у нас есть пользовательский и прикладной уровни, а также уровень данных. Нам нужны MTS, SQL Server и IIS, причем запушенные на разных серверах. Но в нашем отделе разработки нет отдельных компьютеров для всех этих приложений, так что же нам делать? Либо мы вообще не будем проводить тесты, либо для тестирования приходится использовать производственные системы, так как только на них запушены все необходимые приложения.

Когда Билл закончил, к нему подошел Дэн:

— Билл, ты же работал с мэйнфреймами, значит, должен знать о четырехэтапном производственном цикле. Почему бы не использовать его в «Фергюсон и Барделл»?

— По одной простой причине — *нет денег*, — ответил Билл, глядя на Джима. — Каждый год мы подаем заявку на создание автономной тестовой лаборатории, но ее никогда не учитывают, составляя бюджет компании. Прежний директор по ИТ беседовал на эту тему с финансовым директором, который должен был представлять нашу идею на совете директоров, но никто не посчитал нужным выделить на нее деньги.

— Намек понял, — кивнул Джим. — Ты совершенно прав. Я действительно не думал, что такая лаборатория нам нужна. Но вообще-то, — добавил он, глядя на Билла и Дэна. — раньше меня и на совещания ИТ-отдела не приглашали.

— Ясно, — ответил Билл и повернулся к Дэну. — Так что же будем делать? Я не хочу быть причиной потерянной прибыли, и, как мне кажется, ты тоже.

- Первым делом, — сказал Дэн, — ты должен запретить своим разработчикам — уже сегодня — даже прикасаться к производственным машинам, независимо от срочности и сложности тестирования. Я же оповещу все остальные отделы об этом запрете, объясню, почему некоторые компьютеры обозначены как производственные, и составлю их список. Также я попрошу Тима проверить параметры защиты этих компьютеров, чтобы их нельзя было вывести из строя случайно.

Немного подумав, он продолжил:

— Закрыв на замок производственные системы, мы изучим, что нам нужно для тестов, и составим калькуляцию — хорошо бы реальную для бюджета нашей компании. Но сейчас я даже не знаю, как это сделать.

— Дэн, а что такое четырехэтапный производственный цикл, о которой ты говорил? — спросил Джим.

— По существу, это система, основанная на четырех уровнях изоляции и критичности, — ответил Дэн. Он стер рисунок Билла и нарисовал четыре квадрата, надписав на них «*Разработка*», «*Тест*», «*Сертификация*» и «*Эксплуатация*». — Разработчики создают либо новое приложение, либо его часть, либо новую версию старой программы на своих компьютерах. Их я обозначил «Разработка». Когда программистам кажется, что приложение уже готово, они устанавливают его на тестовой системе, имитирующей обычную рабочую среду компании. Этот этап я обозначил квадратом с надписью «Тест». В этот момент программу принимает тестовая группа и проводит все необходимые тесты.

Если приложение правильно работает в простой тестовой среде, его переносят на другой компьютер, где оно тестируется более тщательно, в том числе проверяются параметры, связанные с нагрузкой, временем отклика и взаимодействием с другими приложениями. Эта стадия называется *сертификацией*. Только после проведения всех сертификационных тестов программу можно запускать в производственной среде. Естественно, я сильно упрощаю процесс. Ведь в идеале на каждом уровне должна проводиться настройка защиты, чтобы пользователи на предыдущем и последующем уровнях не могли обойти некоторые важные мелочи — например, написание документации и контроль изменений. Тем не менее сразу видны два важных правила: тестирование нельзя проводить на производственных компьютерах и приложения, тестировавшиеся только на компьютерах разработчиков, нельзя устанавливать на производственные системы.

- Проблема же заключается в том, — добавил Билл, — что такое тестирование надо проводить в среде, похожей на реальную производственную среду. Все сервисы, выполняющиеся *в* производственной среде, должны быть представлены и в сертификационной, а если

приложение — многоуровневое, то и в тестовой. Естественно, все это очень дорого. К тому же для работы и управления такой системой понадобятся дополнительные специалисты.

Джим, потирая подбородок, смотрел на доску:

— Билл, раньше я думал, что вы в ИТ-отделе отыскиваете себе новые игрушки только для того, чтобы потратить деньги компании, и даже сейчас я так думаю о некоторых ваших сотрудниках. Но с тех пор, как я работаю с вами над проектом RMS, я стал понимать, что ты и Дэн просто хотите как можно профессиональнее выполнять свою работу. Наверное, и Тим захочет того же, если вам удастся заставить его думать о чем-нибудь, кроме пончиков,

Джим несколько раз прошелся взад и вперед по комнате и сказал:

- Похоже, для тестовой лаборатории понадобится место. Я прав?

А когда Билл кивнул ему в ответ, он спросил у Дэна:

- Ты был когда-нибудь в 34-й комнате?

- Это бывшая столовая для директоров? Мне казалось, что ее занял маркетинговый отдел,

- Ну да, — ответил Джим. — Они собираются переехать туда на следующей неделе. Но они освободят свою комнату...

- ...которая отлично подойдет для нас! — обрадовался Билл. - Это будет здорово: близко, но в то же время полностью изолирована от мира,

— Но вы не сможете занять всю комнату, — предупредил Джим. — На нее претендует отдел продаж. Половины хватит?

- Еще бы, — сказал Дэн. — Не хочу показаться неблагодарным, но место — только полдела. Что мы туда поставим?

- Вы сможете к концу недели разработать реалистичное предложение по созданию высококачественной тестовой лаборатории? - спросил Джим. — Я не обещаю, но думаю, что мне удастся выбить необходимую технику, — он улыбнулся, — особенно, если я им расскажу о потерянных прибылях.

- Как насчет того, чтобы разбить эту заявку на части? — предложил Дэн. — В первой мы опишем, что нам нужно для тестирования RMS, вторую подадим попозже в этом году, а третью — следующей весной. Так мы распределим затраты, которые большей частью придутся на период, когда успехи проекта станут явными.

Джим посмотрел на Дэна и улыбнулся:

- Знаешь, в тебе пропадает прекрасный дипломат. Отличное предложение. Так и сделай, мы представим его Старику вместе. Он бывает резок, но все же, думаю, поймет, что ты хочешь потратить деньги для того, чтобы заработать — или сэкономить — их в дальнейшем. Ты получишь свою лабораторию.

ГЛАВА 11

Защита приложения

В этой главе

С появлением распределенных приложений все важнее становится эффективная защита. Возможности манипуляции данными на локальных компьютерах растут не по дням, а по часам, поэтому каждому разработчику необходимо знать как можно больше о существующих протоколах и приложениях, чтобы завтра создавать новые программы.

Мы начнем эту главу с описания нескольких протоколов, обеспечивающих защиту. Затем расскажем об аутентификации — безопасном и надежном методе идентификации пользователя при регистрации в системе или при обращении к ресурсам — и шифровании, обеспечивающем защиту информации от несанкционированного доступа. Мы также обсудим контроль доступа, который необходим для определения прав пользователей системы, и аудит, позволяющий протоколировать работу пользователей и их доступ к ресурсам. В заключение мы познакомим вас с протоколами, журналами событий и распределенными средами.

При работе над этой главой мы использовали собственный опыт проектирования и реализации архитектуры приложений, а также следующие материалы:

- Мэри Киртлэнд (Mary Kirtland) «Designing Component-Based Applications» (Microsoft Press, 1998);
- «Mastering Enterprise Development Using Microsoft Visual Basic 6» (Microsoft Press, 1998);
- справочная система Microsoft Visual InterDev;
- Microsoft Windows **NT** Workstation 4.0 Resource Kit;
- справочная система Microsoft Internet Information Server 4.0;
- статья Microsoft Technet #Q102716 «User Authentication with Windows NT»;
- документ «Secure Networking Using Windows 2000 Distributed Security Services», предоставленный отделом Microsoft PBS Web;

- статья Марка Битера (Mark Bieter) «Internet Information Server SecurityOverview»;
- *http://www.microsoft.com/ntserver/security/exec/feature/WebSecurity.asp;*
- *http://home. netscape.com/eng/ssl3/draft302txt.*

Изучив материал этой главы, вы сможете:

- ✓ описать существующие методы зашиты приложений;
- ✓ описать существующие методы аутентификации;
- ✓ разобраться в средствах аутентификации Web-сервисов;
- ✓ охарактеризовать методы защищенного доступа;
- ✓ описать способы шифрования информации;
- ✓ перечислить достоинства аудита приложений:
- ✓ описать методы аудита.

Аутентификация

Степень защищенности системы определяется используемым методом аутентификации. Хотя защищенных приложений множество, пользователи могут свести все преимущества защиты на нет, применяя простые пароли типа 1234, или отказавшись от них совсем, или оставив записанный пароль на видном месте. Существуют различные средства защиты от подобной неосторожности пользователей — например, запрет на применение простых паролей или ограничение минимальной длины пароля, однако не все проблемы решаются программно. Но если вы сделали все возможное, чтобы исключить неосторожность пользователей по отношению к их паролям, то *аутентификация*, то есть идентификации пользователя (возможно, неоднократная), — весьма действенное средство. О ней мы сейчас и поговорим.

Средства аутентификации Windows NT

Аутентификация пользователей Windows NT осуществляется с помощью пакета MSV1_0 Authentication Package (<systemroot>\system32\ msv1_0.dll). Пакет MSV1_0 состоит из двух частей. Первая отвечает за получение запроса на регистрацию в системе и определение его источника (то есть от какого компьютера, локального или удаленного, он поступил), Если запрос локальный, MSV1_0 передает его второму компоненту, который идентифицирует пользователя по информации из локальной базы данных системы зашиты. Если же запрос поступил от удаленного компьютера, информация о пользователе передается сервису Netlogon. который, в свою очередь, передает ее второму компоненту MSV1_0 удаленного компьютера, При взаимодействии сервисов

Netlogon, расположенных на двух компьютерах, используется метод *«запрос — ответ»* (challenge/response); при локальной регистрации он не применяется. В конце концов Netlogon возвращает ответ на запрос первому компоненту MSV1_0 локального компьютера.

Примечание Мы описываем сетевую аутентификацию с использованием метода «запрос — ответ» средствами зашиты Windows **NT LAN** Manager (NTLM). Однако MSV1_0 также обрабатывает интерактивную регистрацию в системе, регистрацию сервисов и регистрацию в сети. Причем независимо от типа регистрации при использовании пакета MSV1_0 передается название домена, содержащего учетную запись пользователя, название этой учетной записи и некоторое представление пользовательского пароля. Разные типы регистрации отличаются именно способом представления парольной информации. При интерактивной регистрации и регистрации сервисов задействован первый компонент сервиса MSV1_0 на локальном компьютере, поэтому пароль передается в незашифрованном виде. MSV1_0 преобразует его в два пароля — для LAN Manager и для Windows NT — и передает сервису Netlogon или второму компоненту MSV]_0, который запрашивает пароль у *диспетчера защиты* (Security Account Manager, SAM), сравнивает пароль с введенным при регистрации и идентифицирует пользователя.

Рассмотрим примерную схему регистрации пользователя в Windows NT.

1. Пользователь вызывает диалоговое окно регистрации в системе, нажав клавиши CTRL+ALT+DEL. Нажатие этой комбинации клавиш перед регистрацией защищает от «троянского коня» — программы, выдающей себя за операционную систему и таким образом похищающей имена и пароли пользователей.
2. Пользователь вводит имя и пароль своей учетной записи, после чего процесс регистрации вызывает *локального диспетчера защиты* (Local Security Authority, LSA).
3. LSA запускает пакет аутентификации MSV1_0.
4. Пакет аутентификации опрашивает локальную базу данных и выясняет, является ли учетная запись локальной. Если ответ положительный, выполняется сравнение введенного имени и пароля с содержащимися в базе учетных данных. В противном случае запрос на регистрацию передается сервису Netlogon. Он связывается с таким же сервисом удаленного компьютера (в нашем примере это контроллер домена). Затем первый компонент MSV1_0 локального компьютера связывается со вторым компонентом MSVl_0 удаленного компьютера, инициируя схему «запрос — ответ».

5. После локальной или удаленной проверки учетной записи диспетчер защиты (ему принадлежит база учетных данных) возвращает *идентификатор системы защиты* (Security ID, SID) пользователя и групп, к которым он принадлежит.
6. Пакет аутентификации создает сеанс и передаст информацию с ним, а также идентификаторы пользователя локальному диспетчеру защиты.
7. Если в регистрации отказано, сеанс удаляется, а процессу регистрации передается сообщение об ошибке.
8. Если регистрация удалась, создается маркер доступа, содержащий идентификатор пользователя, группы Everyone и остальных групп, в которые входит этот пользователь. Затем маркер доступа со статусом Success передается процессу регистрации в системе.
9. Сеанс вызывает подсистему Win32, чтобы создать пользовательский процесс, и присоединяет к нему маркер доступа.
10. В интерактивных сеансах Windows NT запускается процесс-оболочка (рабочий стол).
11. После всех проверок маркер доступа передается процессу-оболочке. В Windows NT 4.0 этот маркер, контролирующий все действия пользователя, ассоциируется с процессом ободочки Explorer.exe.

Аутентификация по протоколу Kerberos

Протокол Kerberos разработан в Массачусетском технологическом институте. Основные термины этой схемы аутентификации — *билет* (ticket), *сеанс и доверие*. В Windows 2000 применяется аутентификация по протоколу Kerberos 5. Ниже перечислены преимущества использования Kerberos по сравнению с доменной стратегией защиты Windows NT.

- **Ускоренная аутентификация при первом подключении** — для аутентификации пользователя серверу приложений не нужно подключаться к контроллеру домена, поэтому он быстрее обслуживает запросы регистрации.
- **Делегирование аутентификации для многоуровневых клиент-серверных приложений и архитектур** — при обращении клиента к серверу клиент работает от имени сервера. Если же для выполнения клиентской транзакции один сервер должен подключиться к другому, протокол Kerberos позволяет первому серверу делегировать второму права на подключение от имени клиента. Благодаря делегированию клиент может работать на втором сервере от имени первого.

Примечание Приложения, которым требуются права доступа к процессу, могут пользоваться дополнительными службами защиты Microsoft Windows 2000, а также воздействовать на билеты; приме-

няемые при вызовах методов, с помощью *делегируемой имперсонации* и *маскирования*. Делегируемая имперсонация позволяет серверу подменять собой клиента, а также передавать его билет удаленному компьютеру. Раньше серверы могли имперсонировать клиента только при доступе к локальным ресурсам. Маскирование позволяет скрыть от сервера-адресата идентификаторы промежуточных серверов. Например, клиент А обращается к серверу В, а тот вызывает сервер С от имени клиента А. Если на сервере В включено маскирование, то для доступа к серверу С используется идентификатор клиента А.

- **Транзитивные доверительные отношения при междоменной аутентификации** — пользователю достаточно пройти аутентификацию в одном узле дерева доменов, так как сервисы аутентификации каждого домена доверяют билетам серверов любого другого домена своего дерева. Такое переходящее доверие упрощает управление доменами в больших сетях.
- **Аутентификация в гетерогенных сетях** — пользователи проходят только одну проверку и на ее основании получают доступ к разным системам под управлением разных операционных систем. Например, средства защиты Windows 2000 можно применять для доступа как к Windows 2000, так и к UNIX или мэйнфреймам. Для этого на UNIX-системах и мэйнфреймах нужно установить соответствующие COM-библиотеки. Таким образом, для аутентификации в доменах Windows 2000/Kerberos пользовательской системе не обязательно работать под управлением Windows 2000.

Примечание Переход от NTLM-аутентификации Windows NT 4.0 к Kerberos не составит труда — сервисы Windows 2000 поддерживают оба этих протокола. Переход от систем масштаба предприятия, использующих Kerberos, к интернет-сервисам, применяющим аутентификацию по открытому ключу (описана далее в этой главе), также прозрачен для пользователей.

Web-аутентификация

Почти все существующие обозреватели поддерживают основную форму аутентификации — передачу по Интернету или интрасети имени пользователя и пароля в текстовом виде. К сожалению, при применении такого метода не исключен риск перехвата передаваемой информации. Поэтому, если вы хотите обеспечить защиту доступа к информации, расположенной в сети или на серверах, позаботьтесь об аутентификации на таких Web-серверах. Помимо средств защиты Windows

NT можно задействовать и средства защиты Microsoft Internet Information Server (IIS) 4.0, обеспечивающие корректную идентификацию пользователей, определения их уровня доступа и создания безопасного соединения.

В этом разделе описана технология аутентификации IIS 4.0, а также разработанная Microsoft технология поддержки защищенных каналов, позволяющая обезопасить передаваемые по Интернету или интрасети данные от постороннего вмешательства.

Аутентификация по методу «запрос — ответ» Windows NT

IIS 4.0 поддерживает аутентификацию по методу «запрос — ответ», использующую для передачи учетных данных криптографические методы. При этом сам пароль не передается по сети и, следовательно, его «перехват» невозможен. Этот метод аутентификации поддерживает Microsoft Internet Explorer 2,0 и более поздние версии.

Если аутентификация по методу «запрос — ответ» разрешена, обозреватель пользователя проверяет пароль посредством защищенного обмена информацией с Web-сервером. Если при этом идентификация пользователя прошла успешно, то данные о его учетной записи не запрашиваются. В противном случае обозреватель предлагает ввести имя пользователя и пароль, которые обрабатываются тем же методом. Эти данные запрашиваются до тех пор, пока пользователь не введет правильную информацию или не закроет диалоговое окно запроса.

Примечание Приоритет аутентификации по методу «запрос — ответ» выше, чем у базовой аутентификации, поэтому, если обозреватель поддерживает оба метода, применяется первый из них,

---

Жетоны

Web-приложения способны отслеживать и идентифицировать пользователей с помощью небольших *файлов-жетонов* (cookie), которые пересылаются сервером на пользовательский компьютер. Они, как правило, содержат глобально уникальный идентификатор (GUID), который Web-сервер запрашивает при следующем посещении узла пользователем. Хотя жетоны — простой способ идентификации пользователей, их не стоит применять для доступа к конфиденциальной или секретной информации. Жетон легко перехватить при передаче или просто скопировать с компьютера, после чего злоумышленник без труда выдаст себя за владельца украденного файла жетона. Хотя результаты кражи жетона и имени пользователя и пароля одинаковы, «позаимствовать» жетон гораздо проще.

Цифровые сертификаты

Посредством *цифровых сертификатов* пользователь может обращаться к защищенным Web-узлам, не запоминая свои пароли. IIS поддерживает сертификаты стандарта X.509. Они удостоверяют личность точно так же, как, например, водительские права или пропуск. Сертификаты выпускает специальный *сертификационный орган*; это может быть как отдел вашей фирмы, так и специализированная компания — например, VeriSign или Entrust. Строгость проверки таких удостоверений зависит от уровня зашиты (или доверия), требуемого ресурсом или приложением.

Для аутентификации на основе сертификатов необходим протокол, обрабатывающий удостоверения на клиенте и на сервере, а также управляющий обменом соответствующей служебной информацией.

Серверные сертификаты

Уникальные цифровые идентификаторы, называемые *серверными сертификатами,* составляют основу работы протокола Secure Socket Layer (SSL). Такие сертификаты, полученные от надежной сторонней организадии, позволяют пользователям идентифицировать Web-узел. Сертификат сервера содержит подробную информацию, в том числе название организации, владеющей узлом, название организации, выпустившей сертификат, и уникальный идентификатор — *открытый ключ.* Все эти данные гарантируют подлинность Web-узла и защиту HTTP-соединения. Открытый ключ вместе с еше одним — секретным — ключом составляют *пару ключей SSL,* используемую Web-сервером для создания защищенного TCP/IP-соединения с клиентским обозревателем. Хотя пара ключей играет важную роль в создании защищенных соединений, непосредственно для шифрования данных она не применяется.

Клиентские сертификаты

При использовании SSL Web-сервер идентифицирует пользователей по их *клиентским сертификатам.* Эти файлы похожи на стандартные удостоверения личности, например, водительские права или паспорт. Типичный клиентский сертификат содержит подробную информацию о пользователе и об организации, выдавшей сертификат. При его подписании пользователь вводит пароль, который в дальнейшем требуется при каждой активации сертификата. Как и в случае водительских прав, обладание удостоверением еше не означает, что вы его настоящий владелец. Ключом к доступу считается пароль, поэтому его должен знать только настоящий хозяин сертификата.

Отождествление сертификатов

Клиентский сертификат можно связать с учетной записью пользователя Windows NT, для чего требуется копия сертификата. В дальней-

шем Web-сервер будет аутентифицировать пользователя по его удостоверению, не запуская процедуру аутентификации «запрос — ответ». При таком преобразовании клиентский сертификат «привязывается» к учетной записи Windows NT, описывающей права пользователя. В результате каждый раз, когда пользователь регистрируется в системе по своему сертификату, Web-сервер автоматически «привязывает» его к соответствующей учетной записи Windows NT.

Такой метод отлично подходит для Web-узлов, выпускающих собственные сертификаты с помощью специальных серверов — например, Microsoft Certificate Server из состава Windows NT 4.0 Option Pack.

Отождествление по шаблону

При использовании отождествления по шаблону серверу IIS не обязательно владеть клиентским сертификатом — вместо этого он проводит аутентификацию пользователей по информации, входящей в сертификат, например, по полю SubjectName. В состав IIS 4.0 включен ActiveX-компонент, применяющий ASP для автоматического отождествления по шаблону. Он позволяет, например, создать ASP-страницу, запрашивающую у пользователя разрешение на связывание его сертификата с учетной записью Windows NT Server. В случае положительного ответа запускается процесс отождествления информации сертификата с соответствующей учетной записью Windows NT Server.

IIS 4.0 может аутентифицировать пользователей, регистрирующихся по клиентскому сертификату, преобразуя содержащуюся в нем информацию в учетную запись Windows NT. Причем IIS 4.0 может связать с одной учетной записью как один клиентский сертификат (отображение «один-к-одному»), так и несколько. В последнем случае для получения не всей, а только определенной информации, применяются *шаблоны* (wildcard). Например, для автоматического отождествления сертификатов всех пользователей, выпущенных определенной организацией, с одной учетной записью Windows NT лучше применить шаблон, а не создавать для каждого сертификата отдельную учетную запись.

Программное использование сертификатов

Средства аутентификации клиентов IIS 4.0 шире обычной аутентификации и контроля доступа. Информация сертификата открыта как ASP-, так и ISAPI-приложениям. Это позволяет разработчикам создавать ASP-и ISAPI-приложения для создания персонализированных материалов, контроля доступа и выполнения собственных запросов к БД на основе информации из клиентских сертификатов. Аутентификация по клиентским сертификатам и SSL-шифрование позволяет создать надежный способ идентификации пользователей.

Microsoft Certificate Server

В Internet Information Server 4.0 Option Pack появилась возможность управления сертификатами с помощью Microsoft Certificate Server и использования стандартных криптографических функций, включенных в состав Microsoft CryptoAPI. Перечислим основные функции Certificate Server:

- обслуживание стандартных запросов сертификатов в формате PKCS #10;
- выпуск сертификатов X-509 версий 1 и 3 в формате PKCS#7;
- выпуск сертификатов для SSL-клиентов и серверов;
- выпуск S/MIME-сертификатов;
- выпуск SET-совместимых сертификатов;
- поддержка открытых интерфейсов, позволяющая создавать модули, обслуживающие сертификаты специализированных форматов.

Средства аутентификация SQL Server

В этом разделе мы расскажем о трех методах защиты, используемых Microsoft SQL Server: стандартном, интегрированном и смешанном.

Встроенные средства

Стандартная защита предполагает проверку всех соединений при регистрации в системе. Соединения, проверенные SQL Server, называются *недоверенными* (non-trusted). Стандартную защиту стоит использовать в сетевых средах, к которым подключены разные клиенты, в том числе и не поддерживающие *доверенные соединения.* Кроме того, этот метод обеспечивает совместимость с предыдущими версиями SQL Server.

Средства аутентификации Windows NT

При использовании интегрированной защиты все соединения проверяются с помощью механизма аутентификации Windows NT. Соединения, проверенные Windows NT Server и принятые SQL Server, называются *доверенными* (trusted connections). Интегрированную защиту следует использовать в сетевых средах, все клиенты которых поддерживают доверенные соединения. Клиентами SQL Server в трехуровневых приложениях являются MTS-компоненты.

Примечание В Windows NT 4.0 MTS-компоненты должны проходить аутентификацию на контроллере домена. Этот метод увеличивает трафик, а следовательно, возрастают накладные расходы. Windows 2000 не обладает таким недостатком, так как аутентификация по протоколу Kerberos не требует участия контроллера домена.

Вот как происходит процесс регистрации MTS-компонента при применении интегрированной **защиты.**

1. Для доступа к SQL Server компонент должен представить учетную запись Windows NT. MTS-компоненты используют учетную запись, заданную в свойстве Identity пакета MTS. Если это учетная запись домена, его контроллер проверяет имя пользователя и пароль, а исполняемые файлы пакета MTS запускаются под контролем средств зашиты LAN Manager. Однако, если учетная запись принадлежит рабочей группе, имя пользователя и пароль проверяются в локальной базе данных системы зашиты.
2. Компонент подключается к SQL Server, после чего SQL Server ишет соответствующий идентификатор регистрации в таблице syslogins. Дальнейшие действия зависят от того, существует ли идентификатор регистрации. Если существует, то компонент получает доступ к SQL Server со всеми правами, присвоенными данному идентификатору. В противном случае компонент получает доступ к SQL Server по стандартному идентификатору — гостевой учетной записи. Если учетная запись компонента наделена административными полномочиями Windows NT, эти привилегии сохранятся и в SQL Server. Возможен еще один вариант: идентификатор отсутствует, а стандартного идентификатора не предусмотрено; тогда доступ к SQL Server запрещается.
3. После регистрации доступ к таблицам SQL Server контролируется правами, предоставленными идентификатору регистрации в соответствующей базе данных.

Смешанный режим

В этом режиме SQL Server может проверять запросы регистрации как своими средствами, так и средствами Windows NT. Здесь поддерживаются как обычные (стандартная защита), так и доверенные (интегрированная защита) соединения. Очевидно, что такая защита применяется в смешанных сетевых средах с клиентами разных типов. При этом для клиентов, применяющих только доверенные соединения, запускается процесс регистрации Windows NT. а для клиентов, применяющих обычные соединения, проверки выполняет SQL Server.

В смешанном режиме процесс регистрации компонента происходит следующим образом.

- Когда компонент пытается получить доступ к серверу с помощью доверенного соединения, SQL Server проверяет имя учетной записи. Если оно соответствует имени пользователя компонента, пусто или содержит пробелы, SQL Server применяет средства защиты Windows NT.
- Если имя содержит какое-либо другое значение, компонент должен передать пароль, с помощью которого SQL Server запустит собственный процесс регистрации (соответствующий стандартной

защите). Если же устанавливается обычное соединение, компонент должен сообщить идентификатор и пароль, которые SQL Server проверит собственными средствами.

Примечание Для решений масштаба предприятия, активно использующих Windows NT Server, MTS и SQL Server, рекомендуется применять интегрированную или смешанную защиту. Интегрированная защита упрощает управление регистрацией, позволяя централизовать администрирование учетных записей. Кроме того, разработчикам не приходится ни встраивать в свои компоненты идентификаторы и пароли, ни помещать их в имя источника данных ODBC. В случае же стандартной защиты при любых изменениях параметров регистрации приходится перекомпилировать компоненты или обновлять DSN.

Аутентификация объектов в SQL Server

Мы уже упоминали, что для доступа к SQL Server необходимы идентификатор регистрации и пароль. Сейчас мы покажем, как такие меры защиты можно реализовать в компоненте.

Компоненты осуществляют регистрацию, используя свойство **ConnectionString** объекта **Connection** или передавая методу **Open** объектов **Connection** или **Recordset** параметр **ConnectionString.** Если компонент передает идентификатор регистрации и пароль, выполняется аутентификация средствами SQL Server. В противном случае (если идентификатор и пароль не передаются) соединение устанавливается с помощью средств защиты Windows NT, а в MTS для регистрации используется идентификатор пакета компонентов.

Если подключение осуществляется посредством компонента доступа OLE DB, его следует уведомить об использовании интегрированной зашиты. При этом идентификатор и пароль не передаются, а устанавливается атрибут **Trusted_Connection**:

```
Dim conn as ADODB.Connection
Set. conn - New ADODB.Connection
conn.Provider = "SQLOLEDB"
conn.ConnectionSTring= "Data Source=MYSERVER;' & _
 "Initial Catalog=Pubs;Trusted_Connection=Yes"
conn.Open
```

Шифрование

Разработчикам приходится защищать от несанкционированного доступа не только файлы и каталоги, но и информацию, передаваемую

по сети. Это особенно важно при передаче данных через Интернет. В настоящее время многие пользователи, посещающие коммерческие Web-узлы, неохотно оставляют сведения о себе, например, номера кредитных карт, боясь, что они попадут в руки хакеров. Сейчас мы расскажем о некоторых методах шифрования информации, предназначенных для зашиты информации в подобных случаях.

Протоколы защиты информации

В защищенные протоколы встроены механизмы аутентификации и шифрования. Чтобы воспользоваться этими средствами, приложению достаточно поддерживать протокол.

Дабы упростить рассказ о работе таких протоколов, мы опишем, на каком уровне эталонной модели OSI (Open System Interconnection) они функционируют, а также как они определяют область видимости защищаемых ими данных. Например, в протоколах, работающих на прикладном уровне — к ним относятся и Hypertext Transfer Protocol Secure (HTTPS), и NTLM — защита реализуется для каждого приложения. Некоторые протоколы создают туннели, или виртуальные соединения, позволяющие устанавливать связь между устройствами сети. Таким образом создаются виртуальные частные сети. Все протоколы подобного типа — например, Point-to-Point Tunneling Protocol (PPTP) и Layer 2 Tunneling Protocol (L2TP) — работают на сетевом и канальном уровнях модели OS1.

Помните, что эффективность защиты, обеспечиваемой протоколом, зависит от применяемого метода шифрования. Все основные сведения о протоколах, описываемых в этой главе, вы найдете в табл. 11.1.

Табл. 11.1. Протоколы защиты информации

Протокол	Уровень модели OSI	Средства аутентификации	Методы шифрования	Транспортный протокол
L2P	Канальный	CHAP, PAP, MS-CHAP	RC4, DES	IP, IPX, NetBEUI
PPTP	Канальный	CHAP, PAP, MS-CHAP	RC4	IP, IPX, NetBEUI
HTTP/SSL	Прикладной	X.509	**RC2, RC4, DES**	IP
1PSEC	Сетевой	HMAC-MD5, HMAC-SHA, DH	DES	IP
SBM/NTLM	Прикладной	«Запрос-ответ»	DES, MD4	IP, IPX, NetBEUI

Secure Sockets Layer (SSL)

Конфиденциальность, целостность и достоверность соединений в IIS 4.0 обеспечиваются технологией Microsoft Secure Channel. Часть этой работы выполняет реализованный на сервере протокол SSL 3,0, позволяющий установить защищенный и практически непроницаемый для посторонних канал связи. SSL гарантирует аутентичность содержимого Web-узла, одновременно устанавливая подлинность пользователей, подключающихся к закрытым узлам.

Примечание 11S 4.0 поддержи наст также протокол Private Communication Technology (PCT) 1.0. Он, как и SSL, поддерживает защищенные соединения.

При использовании SSL сервер и Web-обозреватель «договариваются» о применяемом методе шифрования, обмениваясь сертификатами и ключами и устанавливая доступное только им защищенное соединение. Естественно, необходимо, чтобы методы шифрования сервера и обозревателя были совместимы, В результате обмена данными создается специальный ключ (обычно это делает обозреватель), так называемый *ключ сеанса.* Он используется для шифрования и расшифровки передаваемой информации. Степень сложности ключа, или его *стойкость,* измеряется в *битах.* Чем больше битов в ключе, тем выше надежность шифрования. Обычно применяются 40- или 128-разрядные сеансовые ключи.

Ключи создаются попарно — открытый и закрытый. Открытый ключ разрешается передавать кому угодно (часто их распространяют через публичные агентства, например, сертификационные органы), Закрытый же ключ нужно тщательно охранять. В обмене информацией участвуют оба этих ключа. Во время настройки SSL на Web-узле генерируются и конфигурируются соответствующие ключи. Для генерации запроса на создание пары ключей и их последующую активацию применяется утилита IIS Key Manager. Серверный же ключ можно создать из файла запроса ключа средствами Microsoft Certificate Server; этот же файл разрешается использовать для получения ключа от сторонней организации, такой как VeriSign или Entrust.

Теоретически SSL допустимо применять с любым протоколом на базе TCP/IP, запущенным на любом порту, если либо обозреватель, либо сервер «знают», что используется SSL. Однако ради удобства для каждого протокола выделен отдельный порт, что позволяет фильтрующим брандмауэрам пропускать этот трафик.

Internet Explorer шифрует данные (например, HTML-формы, содержащие конфиденциальную информацию) средствами открытого

ключа сервера. Затем он передает их серверу, который расшифровывает их, применяя свой закрытый ключ. Информацию удается расшифровать только с помощью этого ключа.

Дабы применение SSL было эффективным, стоит шифровать только секретную информацию, например, номера кредитных карт и адреса. Ведь на шифрование нужно время, а значит, передача и прием данных идут медленнее. Поэтому в каталоги, доступ к которым защищен средствами SSL, следует помешать только страницы с конфиденциальными сведениями. Кроме того, такие страницы должны содержать минимум элементов, так как шифруется все их содержимое, в том числе и простая графика. Любой лишний элемент страницы значительно увеличивает время ее загрузки.

SSL, IIS 4.0 и Microsoft Proxy Server

SSL создан для исключения перехвата информации, пока она передается от клиента серверу. В результате SSL не может взаимодействовать с прокси-серверами напрямую. Чтобы все-таки «подружить» протокол SSL с прокси-серверами (например, с Microsoft Proxy Server), необходимо разрешить непосредственный доступ через открытый порт прокси-сервера. Только после этого клиенту удастся использовать SSL для подключения к серверу. Естественно, тот надо соответственно настроить. Однако при применении такого метода возникают дополнительные проблемы. Например, если на компьютере с IIS 4.0 установлен Microsoft Proxy Server 2.0, должна быть включена служба Web Publishing, Кроме того, если используется фильтрование пакетов, нужно добавить фильтр для TCP-порта 443. Служба Web Publishing позволяет Web-клиентам взаимодействовать непосредственно с Web-сервером. Создание на прокси-сервере фильтра для порта 443 позволяет пропускать SSL-трафик на Web-сервер, а значит, устанавливать прямые и защищенные соединения с клиентом.

Server Gated Cryptography

Server Gated Cryptography (SGC) — это расширение протокола SSL. позволяющее финансовым организациям оказывать услуги независимо от местонахождения клиента, не внося никаких изменений в обозреватель или серверное программное обеспечение. SGC позволяет на основе семейства продуктов Microsoft BackOffice создавать инфраструктуру, взаимодействующую с популярными клиентскими приложениями — Internet Explorer 3.02, 4.0, 5.0, Microsoft Money 98 и Netscape Navigator 4.0. Это требуется, например, для работы международных банков. Поддержка SGC включена во все клиентское и серверное программное обеспечение, разрабатываемое Microsoft, а предыдущие версии приложений можно обновить на Web-странице Mic-

rosoft SGC по адресу *www.microsoft.com/security/tech/sgc/*Кроме того, SGC-продукты Microsoft полностью совместимы с SGC-продуктами фирмы Netscape.

При проведении онлайновых банковских операций и банку, и заказчику нужна уверенность, что каждый из них действительно тот, за кого себя выдает. Например, обязанность банка — защищать информацию о Джоне Смите ото всех, кроме него самого. В свою очередь и Джон Смит должен получить доказательства того, что действительно подключился к банку, а не к компьютеру хакеров, пытающихся опустошить его счет.

SGC возлагает ответственность определения личности пользователя на банки, большинство которых требуют, чтобы клиент в начале каждой сессии вводил свой идентификатор и пароль. При этом свою подлинность банк доказывает с помощью цифрового сертификата. Как и обычные сертификаты, он содержит данные, уникально идентифицирующие владельца; их генерирует специализирующаяся на выпуске сертификатов организация. Применяя криптографию с открытым ключом, такая сертификационная фирма гарантирует, что, хотя прочитать сертификат может любой, создателем его является только эта фирма (естественно, после тщательного изучения получателя). Все сертификаты защищены от подделки и внесения в них изменений.

До сегодняшнего дня продукты, использующие стойкое шифрование, были доступны не всем банкам. Это связано с существовавшими до недавнего времени ограничениями на экспорт таких средств из США. В отличие от SSL, который выпускается в двух версиях (экспортной, применяющей 40-разрядные ключи, и североамериканской, использующей 128-разрядные ключи), SGC разрешается экспортировать в единой версии, применяющей 128-разрядные ключи. Это стало возможным благодаря решению Министерства торговли США все-таки разрешить экспорт продуктов, использующих стойкое шифрование, но исключительно финансовым учреждениям.

Для получения цифрового сертификата SGC банк должен доказать свою подлинность — предоставить регистрационный номер Dun & Bradstreet DUNS или American Banking Association или документы от правительства штата, провинции или страны, подтверждающие, что данное учреждение действительно является банком. Полный список сертифицирующих организаций приведен на Web-странице Microsoft SGC.

Установка полученного сертификата на Web-узле ничем не отличается от установки сертификата SSL. В начале каждого сеанса сервер банка автоматически перелает свой сертификат клиенту, Процедура прозрачна для пользователя и сервера, Получив сертификат, клиентское приложение проверяет, действителен ли он, и подтверждает

подлинность банка. После этого клиент может начинать работу со своим счетом.

SGC предоставляет право Web-клиенту и серверу выбрать метод шифрования для сеанса SSL. Этот процесс реализован в виде расширения SSL и его преемника, протокола Transport Layer Security (TLS). SGC можно использовать, только когда его поддерживают и серверные, и клиентские приложения. Поэтому при подключении к SGC-узлу обозреватель и сервер инициируют обычное SSL-соединение. Затем на стадии «рукопожатия» клиент провернет цифровой сертификат и ищет в нем специальные данные, указывающие на возможность использования SGC. Если такой информации в сертификате нет, создается обычный SSL-сеанс с применением 40-разрядного ключа. Если же подобные сведения в сертификате есть, клиент отменяет предыдущие действия и создает сеанс, используя один из перечисленных ниже криптоалгоритмов и соответствующие им ключи:

- 128-разрядный RC4;
- 128-разрядный RC2;
- 56-разрядный DES;
- 3DES (тройной DES, пока еще окончательно не разрешенный к экспорту Министерством торговли США).

CryptoAPI

CryptoAPI, входящий в Windows NT 4,0 и Internet Explorer 3.0 и последующие версии, разработан специально для того, чтобы избавить разработчиков от программирования криптографических операций. Его интерфейс Cryptographic Service Provider (CSP) значительно облегчает доступ к криптографическим функциям, позволяя изменять стойкость и тип шифра без изменений в коде приложения.

CryptoAPI освобождает приложения от необходимости заниматься шифрованием. Он обеспечивает простой доступ к криптографическим функциям, таким как шифрование, хеширование и создание цифровых подписей. Все использующие CryptoAPI приложения могут использовать сертификаты стандарта X.509, что позволяет совместимым с ним программам или системам обращаться к Web-серверам с любой платформы, в том числе из сред **UNIX и** Macintosh.

Контроль доступа

Контроль доступа — важная составляющая архитектуры приложения. Она позволяет контролировать, кто и как использует его, Использование защиты гарантирует пользователю и приложению, что обмен информацией осуществляется исключительно в установленных рамках. Приложения должны обеспечить секретность пользовательских

данных, а также защищать свои компоненты и сервисы от несанкционированного доступа.

Защищенное приложение предоставляет доступ к своим сервисам только тем, кому это разрешено. К тому же каждый его компонент, сервис и файл защищен от несанкционированного просмотра и изменения. Лучше всего обезопасить элементы архитектуры приложения и его процессы с помощью встроенных сервисов Windows NT и Windows 2000. В этих операционных системах несанкционированный доступ предотвращается посредством паролей, защиты ресурсов и сервисов, а также с помощью аудита,

Средства контроля доступа Windows NT

Все действия пользователя Windows NT проходят предварительную проверку. Такие проверки система защиты выполняет при попытке открыть файл, при регистрации и при обращениях с других компьютеров. Средства контроля доступа Windows NT перечислены в табл. 11.2.

Табл. 11.2. Основные функции зашиты Windows NT

Функция	Описание	Где обсуждается
Управление пользователями и группами	Контроль за регистрацией и выходом из системы, как локальным, так и удаленным (для пользователей, групп и администраторов)	Раздел «Контроль доступа»
Определение политики защиты	Определение прав пользователей и групп (срока действия паролей, правил доступа по умолчанию и режима аудита)	Раздел «Контроль доступа»
Доступ к файлам и объектам	Защита файлов и каталогов, в том числе на сменных носителях, а также защита файлов операционной системы	Раздел «Защита файлов»
Доступ к реестру	Защита реестра от удаленного доступа	Раздел «Защита реестра Windows NT»

Благодаря этим функциям приложения могут контролировать доступ ко всем типам объектов Windows NT. в том числе объектам, сервисам и ресурсам программ. Защищаемых типов объектов довольно много, к ним относятся:

- локальные и удаленные файлы и каталоги NTFS;
- процессы и потоки;

- именованные и анонимные каналы;
- буферы консольных экранов;
- проекции файлов;
- маркеры доступа;
- почтовые ящики;
- разделы реестра;
- локальные и удаленные принтеры;
- совместно используемые ресурсы Windows **NT;**
- сервисы Windows NT;
- внутрипроцессные синхронизирующие объекты.

Приложение может и не работать с правилами доступа для всех этих объектов. Однако полезно знать, что приложение способно не только определять такие правила, но и программно устанавливать и проверять их.

Разработчику нужно знать не только средства защиты приложений, но, поскольку IIS позволяет обозревателям обращаться к файлам систем под управлением Windows NT, важно понимать и требования к защите Web-приложений. (Защита Web-приложений описана в следующих разделах.)

Контроль доступа

Приложения контролируют доступ к системе посредством четырех методов: учетных записей пользователей, учетных записей групп, правил и прав пользователей и программного контроля доступа.

Учетные записи пользователей

Приложения способны контролировать доступ с помощью аутентификации, например, «запрос — ответ» Windows NT. Если пользователю не удастся зарегистрироваться, он не сможет запускать приложения.

Основу механизма защиты Windows NT составляют учетные записи пользователей. У каждого пользователя, имеющего доступ в систему, есть учетная запись, включающая его имя, пароль и другие параметры регистрации. Администратор может создавать, удалять и отключать учетные записи посредством утилиты Windows NT User Manager.

Примечание Мы рекомендуем не удалять учетные записи, а отключать их. Ведь с помощью отключенной записи получить доступ к сети нельзя. А если учетную запись удалить, все ссылки на нее также будут удалены, что затруднит работу администратора, пытающегося определить права пользователя, и помешает выполнению аудита.

Администраторы (и люди, имеющие соответствующие права. - например, операторы учетных записей) вправе создать необходимые учетные записи и для каждой из них установить ограничения на дос-

туп к приложениям и ресурсам. Очевидно, что правильная настройка учетных записей очень важна.

Как правило, для защиты приложения не нужно создавать или изменять учетные записи. Но, если они все-таки понадобятся, механизм аутентификации Windows NT всегда к услугам разработчиков.

Примечание Программный доступ к базе данных Windows NT SAM осуществляется с помощью интерфейсов Active Directory Service Interface (ADSI), Active User Object (AUO), Lightweight Directory Access Protocol (LDAP) и ActiveX Data Object (ADO).

Группы

Существует еще один способ ограничения прав — объединение пользователей с одинаковыми правами в группы.

Примечание Группы пользователей, обладающих одинаковыми правами, иногда называют *ролями*.

Объединение учетных записей в группы упрощает администрирование. Кроме того, предоставить сразу нескольким пользователям доступ к определенному ресурсу гораздо проще — ведь в этом случае достаточно предоставить соответствующее право группе в целом.

С помощью групп создается модель защиты, позволяющая четко разграничить доступ различных групп к приложениям. Для разработки такой модели нужно с помощью схем использования определить:

- участников, называемых в универсальном языке моделирования UML *актерами* (actor);
- операции доступа к данным и ресурсам, необходимые каждому участнику (часто описываются диаграммами схем использования, диаграммами операций и сценариями);
- группы пользователей в соответствии с их правами.

Например, возможны следующие группы: «Открытая», «Закрытая», «Конфиденциальная», «Секретная», «Совершенно секретная».

Права пользователей

Разрешения позволяют контролировать права пользователей и групп, вводить разные ограничения — например, срок действия пароля или длительность захвата ресурса — и предоставлять разрешение на доступ к защищенным объектам, таким, как файлы, каталоги и сервисы.

Примечание В большинстве защищенных сред срок действия пароля не должен превышать 45 дней, а его длина должна быть не менее 8 символов.

С помощью разрешений удается контролировать доступ к сетевым ресурсам и сервисам — «регистрация на сервере», «запуск сервисов» или «резервное копирование и восстановление данных». Для такого контроля нужно сначала сформировать группу пользователей, после чего, используя утилиту Windows NT User Manager, назначить права доступа к защищаемым объектам. Такие ограничения определяют, кто и при каких условиях получит доступ к ресурсам. Точное назначение правил доступа для пользователей и групп намного улучшает защищенность промышленных приложений,

Программный контроль доступа

Приложения способны контролировать доступ, программно ограничивая права пользователей и групп. Например, определенный компонент можно открыть только для одного пользователя, а для всех остальных доступ к его методам будет закрыт. Для этого в приложение нужно встроить модули защиты, охраняющие методы компонента.

Однако при использовании программного контроля доступа возрастает нагрузка на администратора. Если такие функции жестко «прошиты» в коде объекта, при каждом изменении прав доступа придется заново собирать и распространять все компоненты. Поэтому лучше воспользоваться средствами контроля доступа MTS. MTS значительно упрощает администрирование прав доступа к компонентам, используя уже установленные права и отделяя защиту от самих компонентов.

Защита файлов

Файлы защищают средствами файловых систем, поддерживаемых операционными системами Microsoft: File Allocation Table (FAT) и Windows NT File System (NTFS). Если разработчикам нужен полный контроль доступа к файлам, стоит использовать NTFS — возможности системы защиты FAT очень слабы.

Использование разрешений для файлов NTFS

Приложение может обращаться не только к базам данных, но и к файлам, содержащим дополнительную информацию — .ini, .txt, .prf и т. п. Если каким-либо файлам приложения требуется защита, к ним можно применить явные разрешения — например, No Access (доступ запрещен), Read (разрешено чтение), Change (разрешены изменения) и Full Control (полный контроль).

Явные разрешения назначают только при следующих условиях:

- используется файловая система NTFS;
- разрешен доступ к объекту;
- разрешено изменять правила доступа к объекту;
- вы полностью контролируете объект (право Full Control),

Для назначения правил доступа к файлам обычно используется Windows NT Explorer. Он позволяет без труда установить права для каждого файла в отдельности или для целого каталога сразу (в том числе и для всех его файлов).

Назначить правила доступа к файлам позволяет и утилита командной строки Cacls.exe. Она отображает или изменяет списки контроля доступа. Ее возможности ограничены установкой только разрешений Read, Change и Full Control.

Примечание У утилиты Cacls.exe есть и преимущества — она позволяет добавлять разрешения к уже существующим (например, добавить группу Tester к группам, имеющим право Full Control для определенного каталога и вложенных каталогов). Кроме того, для вложенных каталогов разрешается назначать другие права,

При использовании приложений масштаба предприятия (таких как Web-узлы) с открытым доступом для внешних пользователей нет ничего необычного в том, что сначала защищают файлы сервера, на котором расположено это приложение, и только потом назначают права файлам Windows NT там, где это необходимо.

Внимание! Будьте осторожны при изменении правил доступа для корневого каталога и системных файлов Windows NT. Иначе никто из пользователей не сможет зарегистрироваться на компьютере.

Дополнительные возможности

Защита распределенных компонентов

Если приложение использует DCOM, его компоненты также следует защитить, то есть назначить разрешения для их запуска и доступа к ним, а также обезопасить их от постороннего вмешательства.

Зашита при создании удаленных объектов особенно важна из-за простоты DCOM. Тип объекта — локальный или удаленный — определяется по элементу реестра данного компонента. Причем в конфигурации по умолчанию запускать объекты на удаленных компьютерах имеет право только администратор. Эти правила (а также права доступа) можно изменить после распределения и регистрации компонентов и назначения правил для файлов.

Правила для объектов назначают с помощью утилиты DCOMCNFG, Она записывает в реестр параметры, определяющие, где разрешено запускать компонент и кто получит к нему доступ.

Подробнее о программе DCOMCNFG рассказано в библиотеке **MSDN** Library — ищите по ключевому слову «DCOMCNFG».

Защита сервисов операционной системы

В Windows NT все сервисы защищены — непосредственный доступ к ним разрешен только процессам режима ядра операционной системы. Пользовательским процессам не разрешено обращаться к ним напрямую. Когда приложение запрашивает сервис Windows NT, система зашиты проверяет права пользователя и открывает доступ только в случае их наличия. В Windows NT можно установить явный контроль доступа для всех сервисов операционной системы, в том числе для процессов, потоков, совместно используемых ресурсов, файлов, папок и устройств.

Защита реестра Windows NT

Информация о параметрах приложений, как правило, хранится в реестре. Например, именно в нем хранятся все параметры распределенных компонентов. Так как по умолчанию администраторам разрешен удаленный доступ, возникает риск несанкционированного изменения данных реестра, относящихся к вашему приложению.

Реестр можно защитить следующим образом:

- защитить файлы реестра;
- ограничить сетевой доступ к реестру на каждой рабочей станции, использующей данное приложение.

Примечание В Windows NT 4.x файлы реестра находятся в каталоге SystemRoot\System32\Config.

Стандартная конфигурация рабочих станций Windows NT позволяет администратору получить доступ к реестру с любого компьютера. Если же в нем отсутствует раздел winreg, любой пользователь, имеющий доступ к компьютеру, сможет подключиться к реестру и повредить информацию о конфигурации.

Примечание По умолчанию элемент реестра winreg не определен, поэтому удаленный доступ к реестру разрешен. Однако в Windows NT Server этот элемент присутствует, и следовательно, доступ к реестру разрешен только администратору.

Дабы ограничить доступ к реестру, в реестре каждой рабочей станции, использующей приложение, создают следующий раздел:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentcontrolSet\Control\Secure
PipeServers\winreg
```

Этот элемент позволяет изменить права доступа к реестру.

Внимание! Важно предварительно согласовать с администратором сети все параметры реестра, используемые приложением и компонентами, чтобы избежать конфликта с принятыми мерами защиты.

Защита ASP- и HTML-страниц

Основная функция IIS — при соблюдении определенных условий передавать от локального сервера удаленному обозревателю ASP- и HTML-страницы. Доступ к Web-страницам и графике можно ограничить, назначив права доступа к каталогам и файлам (таким как .htm, .asp, .gif и .jpg).

Доступ к определенным страницам контролируют и программно, установив в объекте Session права на уровне пользователя. Имя пользователя и пароль, введенные при регистрации, будут сравниваться с информацией базы данных системы зашиты. Защищенные приложения используют при этом встроенные группы Windows NT и выполняют проверку в базе учетных записей.

Примечание Количество пользователей Windows NT 4.x не может превышать 100 000 (обычно от 50 000 до 100 000). Однако продукты некоторых других фирм (помимо системы Microsoft Site Server Membership) обеспечивают защиту Web-узлов, поддерживающих несколько миллионов пользователей. Такие большие Web-сообщества поддерживает и Windows 2000, обеспечивая при этом дополнительную защиту и масштабируемость средствами Active Directory.

Назначение прав Web-серверам

С помощью прав доступа для Web-серверов ограничивают просмотр, запуск и использование ASP-страниц. Разрешения применяются ко всем пользователям и не различаются для разных учетных записей. Основную настройку выполняют следующим образом:

- предоставляют пользователям право Read для всех виртуальных каталогов, содержащих .asp-файлы без сценариев;
- предоставляют пользователям право Read и Script для всех виртуальных каталогов, содержащих .asp- и .htm-файлы со сценариями;
- предоставляют пользователям право Read и Execute для виртуальных каталогов, содержащих .asp и другие исполняемые файлы, например, .exe и .dll.

Установив эти права для каталогов, вы дополнительно ограничите доступ к определенным файлам.

Защита данных и приложений в MTS

В трехуровневых и многоуровневых приложениях доступ к данным напрямую из клиентского приложения невозможен. Пользователям разрешено обращаться только к компонентам среднего уровня. При таком разделении доступа на две части (пользователь обращается к компоненту, а компонент — к данным) применяются два типа зашиты:

- **защита** приложения — авторизация доступа пользователей к коду приложения в рамках пакетов MTS; зашита приложения реализуется на прикладном уровне;
- защита данных — авторизация пакетов MTS при доступе к данным; защита данных реализуется на уровне данных.

Защитная изоляция

В MTS элементами доверия являются серверные пакеты. Вызовы пакета можно защитить. Таким образом, требования к защите приложения влияют на проектирование пакетов. Ведь если при обращении к компоненту требуется авторизация, этот компонент и клиент надо помещать в разные пакеты. В один пакет разрешается помещать только те компоненты, обращения к которым безопасны и не требуют авторизации.

Для каждого компонента определяются роли MTS. Компоненты с одинаковыми ролями можно поместить в один пакет. Как правило, такое группирование безопасно. Компоненты должны доверять друг другу, так как к ним обращаются одни и те же пользователи. К тому же создание групп упрощает администрирование, так как системному администратору не приходится запоминать роли каждого компонента.

Кроме того, серверный пакет выполняется целиком, и, следовательно, все его компоненты выполняются под его управлением. Однако иногда требуется отдельное выполнение некоторых компонентов, если им нужны несколько другие права доступа или, например, аудит. Такие компоненты следует помещать в отдельные пакеты. Хотя точный идентификатор запушенного пакета определить во время разработки нельзя, все рекомендации по этому вопросу, а также права, необходимые для работы компонентов, должны быть документированы. Например, если объекту данных пакета нужен доступ для записи и чтения, в документацию следует записать, что используемый для запуска этого пакета идентификатор должен иметь права на запись/чтение.

Ролевая защита MTS

В основе зашиты MTS лежат роли. В рамках многоуровневой модели их преимущества таковы:

- инкапсуляция доступа — доступ к базам данных можно полностью инкапсулировать в MTS-компонентах, так как пользователи не обращаются к этим базам напрямую;

- **создание пулов соединений** — в MTS разрешается применять пулы соединений, что позволяет устанавливать параллельные соединения между базой данных и MTS. Такие параллельные подключения значительно повышают масштабируемость приложения;
- **упрощенная защита** — администрирование зашиты баз данных значительно упрощается, так как не нужно управлять учетными записями пользователей на уровне базы данных. В этом случае при обращении к данным можно использовать идентификаторы пакетов, что намного эффективнее;
- **аутентификация пакетов** — система защиты MTS способна выполнить аутентификацию пакета при его обращении к другому пакету без дополнительной регистрации в системе и без создания дополнительных сеансов.

Декларативная защита

Чтобы реализовать такой вид защиты в Windows NT, нужно создать группы, представляющие разные категории пользователей приложения. При этом в MTS определяют роли компонентов пакета. При создании многоуровневого приложения группам пользователей Windows NT назначаются соответствующие роли MTS. Таким образом, пользователи и группы не связаны с данными напрямую, а изменение пользовательской защиты заключается в модификации групп, имеющих доступ к пакетам MTS. Такая реализация зашиты MTS называется *декларативной*.

Примечание Традиционно компоненты создавались для имперсонации клиента, обращающегося к данным. Однако такой метод полностью противоположен архитектуре многоуровневых приложений, где защита самого приложения отделена от защиты данных. В MTS каждому пакету следует назначить собственный идентификатор, который будут совместно использовать все компоненты этого пакета. Посредством таких идентификаторов разработчики связывают учетные записи пользователей Windows NT с пакетами.

Программная защита

Защита программируется и в MTS-компонентах. Такая *программная защита* более гранулирована. Например, присвоив роль нескольким группам, вы сможете обращаться к ним из компонента для дальнейшего управления защитой.

Программную защиту реализуют посредством двух методов объекта **ObjectContext**.

- **IsSecurityEnabled** — если возвращается значение true, защита включена, если false — выключена.

- **IsCallerRole** – также возвращает одно из двух значений — true или false. Этот метод определяет, принадлежит ли объект роли, обрабатывая строку, содержащую ее название, и сравнивая это название с именем учетной записи пользователя. После сравнения он возвращает значение true, если пользователь входит в данную роль (группу), и false в противном случае.

Примечание По умолчанию, чтобы запретить доступ в случае возникновения сбоя, необходимо реализовать проверку ошибок.

Включение защиты MTS

Вместе с MTS устанавливается системный пакет. Чтобы защита MTS заработала, этот пакет надо включить и связать роль администратора пакета с учетной записью пользователя Windows NT.

Защита MTS действует как на уровне пакетов, так и на уровне компонентов. Причем оба уровня существуют только в двух состояниях; включен и отключен. Параметры пакетного уровня имеют приоритет над параметрами компонентного уровня. Например, если пакетная защита не включена, то не работает и защита компонентов. Однако если активизировать пакетную защиту, компонентная защита может оставаться отключенной.

Примечание Не забудьте, что защита включается при обращении компонента одного пакета к компоненту другого пакета. Если оба компонента находятся в одном пакете, этого не происходит.

Контроль доступа в SQL Server

Независимо от типа используемой компонентом защиты, SQL Server идентифицирует соединение по учетной записи. Разрешение соединения еще не означает разрешения на доступ к базе данных. Таким образом, права компонента определяются его учетной записью.

Каждый идентификатор регистрации связан с набором прав доступа к объектам базы данных, причем эти права могут быть включены или отключены. Если, например, пользователю присвоено разрешение **Select** для таблицы **Authors**, то в отношении таблицы **Authors** он сможет выполнить только оператор Select. Аналогично, если у пользователя нет разрешения **Select** для таблицы **Employees,** ему не удастся выбрать данные из этой таблицы.

Все разрешения нужно присвоить до того, как компоненты получат доступ к базе данных. Разрешения описывают возможности каждой учетной записи. Они обозначают операторы (например, **Select, Insert, Update и Delete),** которые разрешено применять при работе с

таблицами базы данных. Кроме того, они определяют, разрешен ли запуск хранимых процедур.

Если учетной записи разрешено выполнение хранимых процедур, их можно запускать в течение сеанса. Процедура способна выполнять действия, не разрешенные пользователю. Например, если учетной записи разрешено выполнить хранимую процедуру **AddCustomer**, то, даже если у пользователя нет разрешения **Insert** для вставки данных в таблицу **Customer,** ему удастся добавить записи с помощью хранимой процедуры.

Аудит

Аудит — важный элемент эффективной защиты. К сожалению, во многих приложениях им пренебрегают, из-за чего в их системах зашиты остаются «дыры». Помимо аудита системы защиты существует и простой аудит, позволяющий отслеживать все действия приложения.

Примечание Если вы не знакомы с аудитом, обратите внимание на следующее: издержки, связанные с аудитом, иногда очень велики, так как аудит требует значительных ресурсов. Кроме того, если этот «навязчивый» процесс реализовать слишком усердно, возможно значительное падение производительности приложений. Поэтому разработчики должны выбрать уровень аудита, достаточный для управления приложением и в то же время не слишком сказывающийся на его производительности.

Реализуя аудит, проще всего пойти по пути проектировщиков Windows NT. В этой операционной системе журнал событий может протоколировать практически любые события. Однако многие возможности аудита Windows NT по умолчанию отключены. Естественно, при необходимости системный администратор вправе их включить. Тем не менее минимальная информация о системе, такая как системные ошибки или ошибки приложений, а также данные об автозагрузке сервисов, всегда регистрируются в журнале событий.

События, подвергающиеся аудиту, подразделяют на следующие категории.

- **Информация** — сообщения, информирующие о состоянии системы — например, о том, что объем хранилища данных достиг 750 Мб. Кроме того, такие сообщения могут информировать о корректной работе приложений.
- **Успех** — сообщения, информирующие, например, об успешном запуске сервиса приложения в определенное время не только успокаивают администратора, но и позволяют автоматизированным системам мониторинга отслеживать удачное завершение различных операций.

- **Предупреждение** — информируют о возможных проблемах, предупреждая о неожиданных событиях. После такого сообщения нужно тщательно изучить состояние системы и решить, требуется ли предпринять какие-либо действия по предотвращению сбоев.
- **Сбой** — если какая-либо операция не может быть завершена, приложение должно сообщить об ошибке. Плохо, когда разработчики получают сигнал об ошибке от пользователей. именно поэтому приложению вменяется в обязанность отслеживать сбои в работе клиентов и серверов. С точки зрения защиты важно перехватывать сообщения об ошибках, связанных с нарушением полномочий — по ним удается выявить попытки взлома системы. Например, 42 000 сообщений об ошибочных запросах данных за одну минуту может означать, что в защите системы потенциально существует «дыра» или что приложение работает некорректно.

Журналы

Один из самых простых способов проведения аудита — создать текстовый файл, выполняющий роль журнала. Средства для записи в такие файлы есть во всех языках программирования, а также в ADO и ODBC. Рекомендуется, чтобы за размещение регистрационных файлов в системе отвечали системные администраторы. Тогда приложение сможет извлечь информацию о каталоге, в котором хранятся журналы, еще при установке и записать эту информацию в свой раздел реестра.

Журналы событий Windows NT

Журналы Windows NT просматривают с помощью утилиты Event Viewer. В Windows NT три журнала, по одному для каждого типа событий: связанных с зашитой, системой и приложениями. Поддержку этих журналов можно встроить в приложение, что гарантирует запись информации в соответствующий файл. Регистрационные файлы appevent.evt, secevent.evt и sysevent.evt, расположены в каталоге Systemroot\System32\Config. Администратор сервера должен ограничить доступ к ним, чтобы никто не мог изменить их (скажем, злоумышленники для удаления следов неудавшегося взлома).

В Windows NT представлено несколько интерфейсов аудита. Как уже упоминалось, большая часть функций аудита по умолчанию отключена, но некоторые работают всегда:

- **системный аудит Windows NT — это** регистрация пользователей в системе и выход из нее, доступ к объектам и файлам, назначение прав, управление группами и пользователями, изменение параметров защиты, запуск и завершение работы системы, а также управление процессами;

- **аудит файлов и каталогов** — проверяется наличие сбоев при выполнении следующих операций над файлами и каталогами: чтение, запись, выполнение, удаление, изменение правил и изменение владельца;
- **аудит реестра** — проверяется наличие сбоев при проведении следующих операций над реестром: запрос значений, установка значений, создание разделов, перечисление разделов, уведомление, создание связей, удаление, запись DAC и контроль чтения,

Внимание! Избыточный аудит иногда снижает производительность системы. Подробнее об аудите — в книге «Microsoft Windows NT 4.0 Security, Audit and Control» (Microsoft Press, 1998).

Для проведения постоянного мониторинга приложений в программу можно встроить счетчики производительности утилиты Windows NT Performance Monitor, предоставляющий в реальном времени информацию о функционировании приложения. Интерфейсы регистрации дополнительных счетчиков в системе мониторинга производительности описаны в книге «Microsoft Windows NT Workstation 4.0 Resource Kit» (Microsoft Press, 1996).

Распределенные среды

Способ аутентификации пользователей в Windows NT влияет на методы аудита распределенных приложений, Например, при аутентификации средствами Windows NT маркер передается только из пункта А в пункт В. Когда В хочет получить доступ к С, для этого применяется маркер В. Результат такой операции не является транзитивным — то, что С доверяет А и А доверяет В, еше не означает, что С доверяет В. С другой стороны, протокол Kerberos является транзитивным. Поэтому, чтобы успешно осуществлять аудит в распределенных многоуровневых средах Windows NT, разработчики должны разбираться в методе аутентификации.

Очевидно, что разработчикам придется проводить аудит ошибок, Но это не все; от них требуется также:

- отслеживать успешные события;
- отслеживать сбои;
- вести мониторинг доступа к объектам;
- выполнять мониторинг доступа к базам данных;
- проводить диагностику во время выполнения;
- проводить диагностику во время разработки;
- выполнять мониторинг использования.

Проведение аудита защиты, как правило, обусловлено желанием знать, кто пытался использовать приложение, Для этого применяют

программные сервисы защиты MTS. Например, разработчик может создать объект аудита, отслеживающий событие, которое начинается на презентационном уровне, а заканчивается на уровне данных. Такие сервисы MTS позволяют разработчикам использовать более изощренные методы, чем те, которые обеспечивает ролевая защита.

Большая часть требований к защите — даже к программной — реализуется с помощью ролевой защиты. Например, при проектировании банковского приложения разработчики вправе разрешить кассиру снимать по требованию пользователей с их счетов суммы, не превышающие 500 долларов, а снятие же больших сумм должен санкционировать менеджер, В этом случае метод **Withdraw** (снять) выполняет различные действия в зависимости от роли обращающегося к нему объекта. Для реализации такой процедуры применяют методы контекста объекта **IsSecurityEnabled и** IsCallerInRole.

В редких случаях, когда ролевой защиты недостаточно, применяют интерфейс MTS ISecurity**Property,** позволяющий получать доступ к идентификаторам пользователей. Вначале посредством контекста объекта получают указатель на интерфейс ISecurityProperty, после чего вызывается один из следующих четырех методов, возвращающих идентификатор защиты пользователя (SID):

- **GetDirectCallerSID** — возвращает SID внешнего процесса, вызвавшего выполняющийся в данный момент метод;
- **GetDirectCreatorSID** — возвращает SID внешнего процесса, создавшего выполняющийся в данный момент объект;
- GetOriginalCallerSID — возвращает SID базового клиентского процесса, инициировавшего последовательность вызовов, в которую входит и вызов текущего объекта;
- GetOriginalCreatorSID — возвращает SID базового клиентского процесса, инициировавшего текущие действия.

SID — это структура Windows, содержащая информацию о пользователе и группах, в которые он входит. Для синтаксического разбора SID можно использовать API Windows. Полученные таким образом данные применяют для ограничения доступа к компонентам или получения информации для проведения аудита и создания регистрационных файлов. Если идентификатор защиты, полученный от интерфейса ISecurityProperty, больше не нужен, освободите его, вызвав метод **ReleaseSID.**

Получить SID в Visual Basic довольно трудно, поэтому в MTS включен объект SecurityProperty. Он определен в библиотеке типов MTS. Ссылку на него можно получить через свойство **Security** контекста объекта следующим образом:

```
Dim secProperty as SecurityProperty
Set secProperty = GetObjectContext.Security
```

Объект **SecurityProperty** предоставляет доступ только к имени пользователя, а не ко всему SID. К счастью, для аудита и регистрации событий, как правило, больше ничего и не нужно.

Точно так же можно организовать аудит базы данных по параметрам (таким как имя пользователя), передаваемым хранимым процедурам. СУБД может зарегистрировать имя пользователя в соответствующей таблиде посредством дополнительной хранимой процедуры или триггера.

Резюме

Защиту приложений можно условно подразделить на четыре категории: аутентификация, шифрование, защита доступа и аудит.

Мы обсудили аутентификацию, с которой начинается работа пользователей. Разработчики должны идентифицировать и проверять пользователей, а также исполняемую ими при взаимодействии с приложением роль. В программах для Windows NT действуют несколько механизмов аутентификации, в том числе Windows NT Challenge/Response и Kerberos. Кроме того, приложения могут применять и Web-аутентификацию: базовую, «запрос — ответ», файлы-жетоны и цифровые сертификаты, а также аутентификацию SQL Server: стандартную, интегрированную и смешанную.

При передаче информации между пользователями, сервисами и хранилищами данных применяют шифрование с помощью нескольких протоколов и методов, обеспечивающих защиту информации, таких как SSL и Microsoft CryptoAPI.

После идентификации пользователя приложение должно разрешать ему доступ только к необходимым данным, контролируя при этом обращения к системным сервисам, файлам, компонентам и реестру Windows NT. Web- и MTS-приложениям доступно множество средств ограничения доступа.

И наконец, вы изучили аудит приложений — узнали о различных объектах аудита и их регистрации.

Закрепление материала

1. Что такое аутентификация?
2. Перечислите основные методы Web-аутентификации.
3. Что такое шифрование?
4. Что такое зашита доступа?
5. Зачем нужен аудит?
6. Какие службы Windows NT поддерживают аудит?

ГЛАВА 12

Стадия «Разработка» и ее результаты

В этой главе

Эта глава посвящена переходу от фазы «Планирование» к фазе «Разработка» на основе функциональных спецификаций и утвержденного проекта приложения. Причина неудачи множества проектов — низкое качество реализации, поэтому при переходе от планирования к разработке необходимо сосредоточиться на создании надежного кода.

На стадии разработки приложение «начинает дышать», и тогда же выявляются некоторые проблемы. В этой главе мы рассмотрим процесс разработки и участие в нем сотрудников проектной группы. Мы детально опишем тестирование, выявление ошибок и стратегию ориентации на отсутствие дефектов, а также рассмотрим компромиссы, необходимые на этой стадии. Кроме того, мы обсудим реализацию многоуровневых приложений, а также окончательный этап стадии «Разработка» и его результаты.

При работе над этой главой мы использовали собственный опыт проектирования и реализации архитектуры приложений, а также следующие материалы:

- Мэри Киртлэнд (Mary Kirtland) «Designing Component-Based Applications» (Microsoft Press, 1998);
- Джим Маккарти (Jim McCarthy) «Dynamics of Software Development» (Microsoft Press, 1995);
- Стив Магуайр (Steve Maguire) «Debugging the Development Process» (Microsoft Press, 1994);
- Стив Макконнелл (Steve McConnell) «Software Project Survival Guide» (Microsoft Press. 1998);
- учебный курс MSF 1516 «Principles of Application Development» (Microsoft Press, 1998).

Изучив материал этой главы, вы сможете:

- ✓ перечислить промежуточные результаты и документы этапов «Завершение разработки» и «Первое использование»;
- ✓ описать функции участников проектной группы на стадии «Разработка»;
- ✓ обосновать необходимость ориентации на отсутствие дефектов на стадии разработки;
- ✓ описать влияние тестирования на качество программного продукта;
- У разобраться в методике отслеживания ошибок.

Особенности стадии «Разработка»

«Разработка» — третья из четырех стадий модели процесса разработки MSF. Она следует за стадией «Планирование», которая завершается одобрением плана проекта. До сих пор проектная группа занималась в основном концепцией, архитектурой продукта и планированием. На стадии «Разработка» основная задача — выполнение проекта.

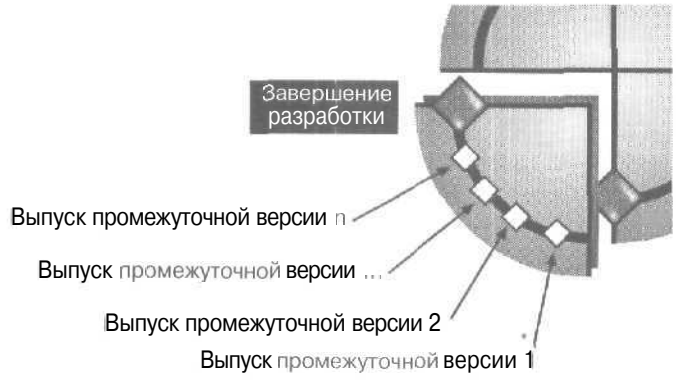


Рис. 12.1. Стадия «Разработка»

Главный вопрос, которым задастся проектная группа на этой стадии: как организовать разработку, чтобы создать спроектированный продукт в запланированные сроки? Ответ на этот вопрос основан на понимании концепции продукта с учетом необходимости выпуска работающего приложения.

Как показано на рис. 12.1, стадия «Разработка» завершается написанием кода и выпуском первой версии приложения. Результаты этапа «Завершение разработки» таковы:

- все необходимые функциональные возможности приложения реализованы (хотя, вероятно, и не самым оптимальным образом);
- продукт прошел первоначальное тестирование; продолжается устранение выявленных ошибок (завершение этой работы на данном этапе не обязательно);
- проектная группа и другие участники проекта согласны с тем, что все реализованные функциональные возможности отвечают концепции и функциональным спецификациям, и реализованы успешно;
- завершена подготовка к тестированию производительности продукта и его стабилизации.

Фаза «Разработка» во многом схожа с другими стадиями модели процесса разработки MSF. Например, как вы помните, фаза «Планирование» завершается подготовкой функциональных спецификаций, плана и графика проекта, а также сводного документа оценки рисков. Эти документы, означающие достижение этапа «Одобрение плана проекта», становятся исходными для стадии «Разработка». Кроме того, они необходимы для оценки различных характеристик процесса разработки. Помните, что эти документы не остаются неизменными — они вполне могут претерпевать изменения по мере выполнения стадии «Разработка». Эта стадия завершается, когда подготовлены пересмотренные варианты этих документов, а также:

- исходный код и исполняемые модули проекта;
- результаты исследования производительности;
- основные составляющие процесса тестирования.

В этой главе мы обсудим, как применять результаты фазы «Проектирование» на стадии «Разработка» для создания продукта и достижения этапа «Завершение разработки», а также промежуточные этапы этой стадии.

Стадию «Разработка» программисты часто называют «настоящей работой». Действительно, основная ее задача — создание работающего продукта.

Связь фаз «Проектирование» и «Разработка»

Проектирование архитектуры продукта на стадии «Проектирование» определяет успех его реализации на стадии «Разработка». Стив Макконнелл в своей книге «Software Project Survival Guide» описывает связь этих стадий, сравнивая их с движением против течения реки и по течению. Разработка архитектуры на стадии «Планирование», считает он, похожа на движение вверх по течению — чем выше вы заберетесь, тем проще будет сплавляться вниз на стадии «Разработка». Этот процесс, начинающийся по завершении стадии проектирования

и заканчивающийся выпуском продукта, будет тем успешнее и проще, чем лучше продумана архитектура.

График проекта описывает предварительный распорядок работ на стадии «Разработка». Функциональные спецификации описывают концептуальный, логический и физический проекты приложения, представляющие собой основу кодирования. Анализ и управление рисками надо продолжать и на стадии «Разработка», поскольку не исключено как появление новых, так и изменение уже известных факторов риска. План разработки реализуется последовательно через промежуточные этапы.

Например, отдельный элемент архитектуры решения по мере перехода от разработки концепции к проектированию и затем к разработке может «вылиться» сначала *к* десяток вариантов физического проекта, а затем — в сотни строк кода. При переходе от проектирования к кодированию проектная группа не способна сразу создать код всех функций, сервисов и объектов — ясно, что это делается последовательно. По мере завершения кодирования отдельные фрагменты связывают друг с другом, создавая приложение.

В процессе разработки каждый вариант приложения считается промежуточной версией. Промежуточные версии, полученные ближе к завершению процесса «Разработка», передают пользователям — это так называемые альфа- и бета-версии.

Основные этапы стадии «Разработка»

Основная цель стадии «Разработка» модели процесса разработки MSF — согласие всех участников проекта с тем, что реализованы все запланированные функциональные возможности продукта. Кроме того. заказчик и проектная группа должны принять решение о том, что реализация всех элементов и продукта в целом завершена.

Распределение обязанностей на стадии разработки

На стадии «Разработка» все участники проектной группы выполняют свои функции, а также выявляют риски и управляют ими. Каждый член группы выполняет свои обязанности, что в сочетании обеспечивает успешное выполнение стадии «Разработка»,

- **Менеджер продукта** — отвечает за выполнение требований заказчика, информирует заказчика и других участников проекта о состоянии проекта и продукта, готовит пользователей к выпуску альфа- и бета-версий.
- **Менеджер программы** — отвечает за связь всех подгрупп проектной группы, координирует выполнение промежуточных этапов проекта, отвечает за подготовку итоговых документов фазы «Раз-

работка», включая пересмотренные функциональные спецификации, план и график проекта, а также сводный документ оценки рисков.

- **Разработчики** — пишут код, реализующий все запланированные функциональные возможности продукта, выполняют предварительное тестирование кода и функциональных возможностей, отвечают за создание альфа- и бета-версий продукта, а также первой окончательной версии в конце этапа «Завершение разработки».
- **Группа обучения пользователей** — выполняет первое тестирование продукта и тестирует производительность работы пользователей, готовит пользователей к выпуску альфа- и бета-версий, разрабатывает справочные и обучающие материалы, отвечает за подготовку итоговых документов, относящихся к применению продукта и его сопровождению.
- **Группа тестирования** — разрабатывает спецификации, планы, примеры использования и сценарии для выполнения первого этапа тестирования функциональных возможностей продукта — проверки промежуточных версий продукта, а также альфа- и бета-версии. Основная задача группы тестирования — выявление и отслеживание ошибок в процессе разработки. Группа тестирования также отвечает за документирование процесса тестирования продукта на всех этапах процесса разработки.
- **Группа логистики** — отвечает за создание условий для работы проектной группы, создает материалы и документацию по сопровождению продукта, занимается развертыванием альфа- и бета-версий продукта.

В процессе разработки следует придерживаться трехэтапного подхода, который мы упоминали при обсуждении стадии «Концептуальное проектирование». Эти этапы не надо считать обязательными - это, скорее, рекомендации. Их названия также в достаточной мере условны.

Первый этап: анализ и рационализация

На этом этапе проектная группа уже располагает достаточно конкретным планом разработки, так что особой нужды в дополнительном исследовании нет. Единственное, что следует проанализировать на этом этапе, — текущий проект продукта.

Кроме того, придется проанализировать график проекта и выявить ресурсы, выделенные на кодирование. Функциональные возможности продукта надо разделить на составляющие и назначить группы, отвечающие за кодирование каждого фрагмента. При распределении обязанностей необходимо учесть структуру сервисов приложения.

Группы, отвечающие за кодирование тех или иных наборов функций приложения, следует формировать с учетом знания языков программирования и средств разработки, необходимых для реализации отдельных фрагментов различных уровней приложения.

Сотрудники группы тестирования анализируют проект приложения и решают, кто будет выполнять различные виды тестов, а также еще раз изучают схемы и сценарии использования для проверки полноты тестирования всех функций продукта. В их задачи также входит подготовка исчерпывающего комплекта тестовых сценариев с учетом схем использования, физического проекта и других требований, например, параметров, выбранных для анализа производительности приложения и эффективности работы пользователей. Кроме того, группа тестирования должна выработать план выполнения тестовых сценариев.

И наконец, на стадии «Разработка» проектная группа выпускает несколько промежуточных версий продукта (мы обсудим этот вопрос подробнее в следующих разделах). После выпуска каждой промежуточной версии следует собрать отклики пользователей, сотрудников групп тестирования и логистики и проанализировать их. Таким образом удастся выяснить, какие проблемы в текущей версии решены, а что еще предстоит сделать.

Второй этап: реализация

Второй этап — «золотое время» разработчиков: начинается реализация проекта приложения. Однако для достижения этапа «Завершение разработки» недостаточно написать код приложения, необходимо еще подготовить соответствующие документы. Группа обучения пользователей готовит материалы, необходимые для обучения и технической поддержки пользователей и для сопровождения приложения, — эти материалы гарантируют успешное развертывание законченного продукта, Хотя обычно основное внимание уделяется традиционным руководствам пользователя и инструкциям по установке, группе обучения иногда приходится готовить и дополнительные материалы — обучающие программы и комплект мастеров, упрощающих работу пользователей.

Третий этап: аттестация

Аттестация продукта — задача всей проектной группы, однако на стадии «Разработка» ее выполняют в основном сотрудники группы разработки и тестирования. Обязательные составляющие этой работы — постоянное тестирование, проверка производительности, отслеживание ошибок и ориентация на отсутствие дефектов.

Поскольку на стадии «Разработка» последовательно выпускается несколько версий приложения, часть проектной группы занимается

аттестацией выпускаемых версий и соответствующей документации. Чтобы добиться наибольшей эффективности аттестации кода приложения, следует максимально автоматизировать процесс тестирования.

Управление рисками

Процесс управления рисками начинается на стадии концептуального проектирования и продолжается до конца жизненного цикла программного продукта. Сводный документ оценки рисков, который является одним из результатов фазы «Планирование», на всех следующих стадиях постоянно уточняется. Процесс уточнения, состоящий из пяти шагов, проиллюстрирован на рис. 12.2. Очевидно, что проектная группа продолжает определять статус различных рисков, выявленных на стадии «Планирование» и перечисленных в сводном документе. Как и прежде, рассматривая каждый риск, проектная группа должна ответить на следующие вопросы.

- Что сделано для снижения риска?
- Изменился ли риск под действием внешних факторов?
- Изменились ли вероятность или последствия какого-либо из рисков под действием внешних факторов или в результате деятельности группы? Если да, как изменилась вероятность реализации риска и его приоритет?
- Исключены ли какие-либо факторы риска?
- Какие действия следует предпринять с учетом ответов на эти вопросы?

Рис. 12.2. Повторный анализ рисков

Мы вновь хотим подчеркнуть важность управления рисками для успешного завершения проекта. Проектная группа должна постоянно заниматься этой работой. Более того, это важнейшая составная часть управления проектом в рамках MSF. На стадии «Разработка»

проектной группе придется заново оценить 10 важнейших рисков проекта и переработать планы их снижения, а также предпринимать постоянные усилия по устранению реализовавшихся рисков.

Этап «Завершение разработки» и его результаты

Основная цель стадии «Разработка» — завершение разработки, о чем свидетельствуют следующие результаты.

- **Пересмотренные функциональные спецификации** — в этом документе отражаются изменения проекта в результате разработки приложения; другими словами, в этом документе проект приведен в соответствие с готовым продуктом. Пересмотренные функциональные спецификации отражают все коррективы, внесенные в проект на стадии разработки и согласованные с заказчиком. Кроме того, здесь описаны новые функциональные возможности, которые будут включены в следующие версии приложения,
- **Пересмотренный план проекта** — здесь описан план выполнения последней фазы проекта и стабилизации продукта, а также перечислены все развертываемые составляющие. Пересмотренный план отражает все изменения, внесенные в проект на стадии «Разработка».
- **Пересмотренный график проекта** — в пересмотренный график включается время, необходимое для стабилизации продукта. Как и пересмотренный план проекта, график отражает все изменения, внесенные на стадии «Разработка».
- **Пересмотренный сводный документ оценки рисков** — здесь отражены потенциальные угрозы и меры, предпринятые группой для их уменьшения, а также описаны как уже известные, так и новые риски, обнаруженные на стадии разработки. Кроме того, в документе следует описать план управления рисками и ход его выполнения до настоящего момента.
- **Исходные тексты приложения и исполняемые модули** — все тексты и модули, составляющие полнофункциональный выпуск продукта. Этот выпуск свидетельствует о завершении разработки и имеет статус версии-кандидата.
- **Средства повышения эффективности работы пользователей и сопроводительные материалы** — описание работы приложения как для пользователей, так и для группы сопровождения. Средства повышения эффективности работы пользователей варьируются от мастеров и документации до материалов для учебных курсов. Сопроводительные материалы для группы сопровождения включают инструкции по установке, конфигурированию, администрированию и использованию приложения.

- **Тестовые материалы** — методы аттестации приложения и список элементов, тестируемых на стадии «Стабилизация». Это планы тестирования, спецификации тестов, схемы и сценарии, обеспечивающие тестирование всего набора функциональных возможностей продукта. Группа тестирования должна создать набор автоматизированных тестовых сценариев для использования на стадии стабилизации.

Промежуточные этапы

Для успешной реализации целей проекта следует разбить стадию «Разработка» на несколько промежуточных этапов разумного размера. Этот способ позволяет сосредоточиться на необходимости выпуска продукта. Хотя группа может добавить свои промежуточные этапы, обычно стадию «Разработка» подразделяют на выпуск промежуточной версии, за которым следуют выпуск альфа- и бета-версий, свидетельствующий о достижении этапа «Завершение разработки».

Промежуточные этапы позволяют постоянно контролировать ход выполнения проекта. Код часто пишется параллельно несколькими группами, и поэтому очень важно иметь возможность оценить результаты каждой из них. Промежуточные контрольные точки заставляют членов группы синхронизировать усилия; этот процесс иногда называют *интеграционным тестированием.*

Обычная частота промежуточных выпусков для проектов среднего размера — раз в 1-3 месяца; для больших проектов ее можно уменьшить до одного выпуска в 2-4 месяца. Число и частота промежуточных выпусков зависит от размера и продолжительности проекта,

Группа должна стремиться как можно раньше добиться создания стабильных версий пользовательского интерфейса и базы данных. Хотя они и не включены в список формальных промежуточных этапов стадии «Разработка», их лучше закончить как можно быстрее. Дело в том, что многие другие работы — например, создание обучающих материалов — в значительной степени зависят от степени завершенности пользовательского интерфейса. Важно и определиться со структурой базы данных, поскольку она диктует компоновку различных функциональных возможностей продукта, а эти решения также принимаются на ранней стадии разработки. Таким образом, чем раньше завершена разработка пользовательского интерфейса и структуры БД, тем меньше изменений придется вносить в код и документацию на следующих этапах работы.

Стадия «Разработка» — итерационный процесс решения четко поставленной задачи. Промежуточные выпуски могут иметь конкретную цель — скажем, тестирование определенных функций, части

пользовательского интерфейса или развертывания продукта. Некоторые промежуточные выпуски иногда специально адресуются пользователям или заказчику. В первых промежуточных выпусках надо реализовать приоритетные функциональные возможности продукта, чтобы продемонстрировать способность группы выполнить задачу. В первых выпусках для внутреннего пользования следует уделить особое внимание архитектурным особенностям, связанным с наибольшим риском или наибольшей неопределенностью, чтобы оценить их осуществимость или. наоборот, выявить изменения, необходимые для минимизации эффектов изменения проекта.

Разбиение большого проекта на составные части помогает группе видеть ежедневные достижения, что позволяет действовать более конструктивно. Кроме того, чем раньше вы подкорректируете проект, тем дешевле это вам обойдется. Промежуточные версии для внутреннего и внешнего использования помогают повысить качество продукта, что закладывает основу для следующих этапов разработки и позволяет группе своевременно исправить ошибки и устранить дефекты проекта,

Промежуточные версии для внутреннего пользования

Промежуточные выпуски для внутреннего пользования укладываются в общую концепцию последовательного создания версий продукта. Их общая цель — постепенное расширение функциональных возможностей стабильной версии. Кроме того, внутренние выпуски помогают проектной группе добиться стабильности продукта и требуемого уровня качества, а также позволяют попрактиковаться в выпуске продукта. Каждая промежуточная версия становится точкой отсчета для последующих этапов разработки и, кроме того, основой для количественных оценок степени прогресса. Сопоставление реализованных функций продукта с функциональными спецификациями позволяет правильно расставить приоритеты и выявить особенности проекта, не проанализированные на стадии «Проектирование».

Внутренние версии создаются в течение всей стадии «Разработка». Каждый следующий выпуск должен расширять функциональные возможности предыдущего, как показано на рис. 12.3. Это способ позволяет рассматривать отдельные выпуски как независимые продукты, ориентируя группу на достижение поставленных целей. Анализ промежуточных выпусков — обязательный элемент самообучения проектной группы, позволяющий ей расширять арсенал удачных решений и избегать повторения ошибок.

Разработка каждого промежуточного преследует некую цель — в частности, определенный уровень качества продукта, позволяющий говорить о завершении этого этапа. Кроме того, необходимо предусмотреть небольшую фазу стабилизации текущей версии продукта, во

время которой группа «подтягивает» ее к принятым стандартам качества и практикуется в стабилизации продукта.

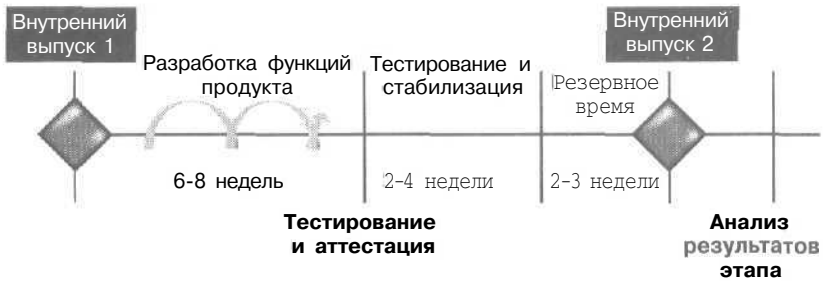


Рис. 12.3. Внутренние версии продукта на стадии «Разработка»

По мере разработки проектная группа постепенно расширяет набор функциональных возможностей последовательных промежуточных выпусков до тех пор, пока не реализует задачи, поставленные в функциональных спецификациях. Отметим, что недостаточно точное планирование на стадии проектирования значительно затрудняет реализацию промежуточных версий.

Промежуточные версии для внешнего пользования

Хотя это и не предусмотрено моделью MSF, мы рекомендуем на стадии «Разработка» создавать и промежуточные версии для внешнего пользования. Они помогают синхронизировать процесс кодирования и усилия разных подгрупп. Кроме того, их появление демонстрирует заказчику, пользователям и другим участникам проекта, что группа движется к поставленной цели.

Как правило, альфа-выпуск (или выпуски) продукта не содержат всех функциональных возможностей. Его цель в другом — отработать выпуск продукта и проверку других материалов, которые должны быть подготовлены на стадии «Разработка» (например, справочные и учебные материалы).

Бета-версия по своим функциональным возможностям очень близка к окончательному варианту продукта, однако, как правило, не оптимизирована.

Пересмотренные функциональные спецификации

Функциональные спецификации считаются основным результатом фазы «Планирование» и являются основой стадии «Разработка». Это своего рода компас, определяющий направление разработки проекта. Продолжая метафору, скажем, что проектная группа не может «точно проложить курс» до самого пункта назначения, однако функциональные спецификации служат ей ориентиром. Промежуточные

этапы фазы «Разработка» — те инструменты, которые помогут вам воплотить функциональные спецификации в жизнь.

Анализ и пересмотр функциональных спецификаций

Пересмотренные функциональные спецификации не обязательно должны быть полностью взаимосвязаны с промежуточными этапами фазы «Разработка». Тем не менее в них необходимо отразить все различия между проектом приложения, описанным в исходных спецификациях, и состоянием продукта на этапе «Завершение разработки».

Не следует удивляться возникновению необходимости внесения новых изменений по окончании каждого промежуточного этапа. Помните, что основная цель функциональных спецификаций — достижение согласия между заказчиком и проектной группой (и внутри проектной группы). Как и в случае исходных спецификаций, все изменения в спецификациях следует проанализировать. В этом принимают участие все, кто участвовал в выработке исходных спецификаций; при этом проверяется соответствие приложения запросам заказчика и пользователей. Помните, что совместное обсуждение служит гарантией достижения согласия, кроме того, функциональные спецификации, помимо своего основного назначения, являются и средством обмена информацией между проектной группой и заказчиком.

Пересмотр функциональных спецификаций позволяет проверить соответствие продукта замыслу и еще раз убедиться, что выполнены основные задачи, поставленные в проекте. Критерии, которыми должны руководствоваться различные члены группы при пересмотре функциональных спецификаций, перечислены в табл. 12.1.

Табл. 12.1. Причина пересмотра функциональных спецификаций

Роль	Причина
Заказчик	Функциональные возможности промежуточной версии продукта не соответствуют бизнес-целям
Менеджер продукта	Несоответствие продукта существующим требованиям; в этом случае менеджер продукта должен настоять на пересмотре функциональных спецификаций, чтобы привести в соответствие требования и готовый продукт
Менеджер программы	Невозможно выполнить обязательства в установленные сроки
Разработчик	Невозможно реализовать функциональные возможности, зафиксированные в спецификациях; возникновение новых факторов риска,

(продолжение)

	препятствующих реализации; несоответствие трафика разработки поставленным задачам
Тестер	Невозможно выполнить тестирование всех функциональных возможностей в полном объеме. На стадии «Разработка» группа тестирования должна руководствоваться графиком труппы разработки и, в частности, датами промежуточных выпусков — тестировать можно лишь то, что уже готово
Инструктор	Несоответствие функциональных возможностей альфа- и бета-версий нуждам пользователей (наличие ошибок при этом не считается решающимфактором)
Логистик	Невозможно развернуть альфа- и бета-версии; по мере выпуска последующих версий труппа логистики должна внести в функциональные спецификации поправки, гарантирующие гладкое развертывание продукта

В качестве примера пересмотра функциональных спецификаций рассмотрим разработку системы управления ресурсами, с которой вы познакомились на практикумах. На стадии разработки были обнаружены проблемы с использованием Web-клиента, отсоединенных объектов ADO **Recordset** и средств зашиты Windows NT. Проблема заключалась в том, что при использовании отсоединенных объектов **Recordset** идентификатор системы защиты текущего пользователя передавался бизнес-объектам промежуточного слоя MTS некорректно. Проектной группе пришлось пойти на изменение архитектуры приложения. Хотя и Web-клиент, и отсоединенные объекты ADO **Recordset** были протестированы на стадии проверки реализуемости концепции, интеграция средств зашиты Windows NT ускользнула от внимания группы. В результате проблема архитектуры осталась невыявленной до начала разработки. К счастью, в конце стадии «Разработка» появились пересмотренные версии системного ПО, в которых проблема устранена, однако группа не успела воспользоваться ими в первой версии приложения. Устранение проблемы с соответствующим изменением архитектуры включено в план выпуска следующей версии продукта.

Пересмотренный план проекта

План проекта также надо пересмотреть с учетом различий между проектом приложения и тем, что удалось реализовать, Хотя план проекта

и не используется в каждодневном управлении, он отражает общие тенденции проекта.

По мере выполнения проекта планы группы и ее подгрупп претворяются *в* жизнь и приводятся в соответствие *с* замыслом проекта. Составление и выполнение основного плана проекта относится к компетенции менеджеров программы. Каждая подгруппа реализует свой план — часть основного плана проекта. Планы группы разработки, группы тестирования и других подгрупп проектной группы по мере выполнения проекта пересматриваются и обновляются с учетом новой информации, появляющейся на стадии разработки приложения.

Пересмотренный график проекта

Основной график проекта включает сроки выполнения следующих этапов:

- разработка;
- выпуск внутренних и внешних версий продукта;
- тестирование;
- подготовка пользователей;
- сопровождение;
- информирование участников проекта;
- развертывание.

На этой стадии главным считается график разработки, поэтому все прочие графики надо привести в соответствие с фактическим графиком работ группы разработки. Любое изменение графика выполнения перечисленных выше работ необходимо отразить в основном графике проекта.

На стадии планирования в рамках создания основного графика проекта группа устанавливает сроки выполнения промежуточных этапов, которые служат ориентиром на стадии разработки. При изменении (чаще всего — несоблюдении) этих сроков все коррективы обязательно отражаются в графике с уведомлением менеджера программы. Тот, в свою очередь, должен оценить, как это повлияет на проект в целом. В случае конфликта между графиками разных подгрупп менеджер программы должен принять компромиссное решение, учитывающее интересы подгрупп. Когда решение принято, новые графики включаются в пересмотренный общий график проекта.

Пересмотренный сводный документ оценки рисков

Сводный документ оценки рисков разрабатывается на стадии разработки концепции. Его созданием и ведением, включая проверку полноты и точности, занимается менеджер программы Пересмотренный документ включает оценки новых факторов риска, обнаруженных

разными подгруппами на стадии «Разработка». Сведя эти оценки вместе, группа получает общее представление о новых рисках.

Сводный документ оценки рисков позволяет синхронизировать выявление и анализ новых факторов риска разными подгруппами проектной группы и помогает правильно расставить их по приоритету. Управление рисками по-прежнему возложено на подгруппы, к компетенции которых относится соответствующий фактор риска.

Код и исполняемые модули

Исходные тексты и исполняемые модули составляют комплект поставки полнофункциональной версии приложения. Как только код, созданный группой разработки, начинает соответствовать функциональным спецификациям, следует начинать работу по контролю качества кода.

Следите, чтобы код и исполняемые модули были эффективны и созданы без нарушения установленного графика. К сожалению, часто проектные группы поступаются качеством кода, дабы закончить разработку к установленному сроку. Следите за качеством кода все время разработки проекта, несмотря на жесткие сроки. Попытки сэкономить на качестве кода в угоду сиюминутным интересам чреваты крайне нежелательным удорожанием всего проекта в будущем.

Избежать снижения качества кода помогают стандарты; они:

- вынуждают разработчиков придерживаться единого подхода к кодированию в любых условиях;
- постоянно напоминают разработчикам о важности качества кода.

Решение о внедрении стандартов и процедуры контроля за их соблюдением сказывается на отношении разработчиков к кодированию. Понимание того, что стандарты — непреложные законы, а не общие необязательные рекомендации, заставит разработчиков соблюдать их. Стандарты — обязательная составляющая должностных обязанностей разработчиков, а не соглашение, придуманное исключительно для удобства их работы.

Традиционно стандарты кодирования охватывают следующие области кодирования:

- именование;
- структура;
- комментарии;
- конкретные требования и запреты.

Их соблюдение гарантирует создание кода, *понятного* не только автору, но и другим программистам. Такой код имеет преимущества, даже если с ним работает только автор. Каждый программист знает,

что, только руководствуясь стандартами, можно писать такой код, который будет понятен автору спустя три месяца после его написания.

Еще одна цель традиционных стандартов кодирования — повышение надежности кода. Этой цели служат конкретные требования и запреты, например, обязательность обработки ошибок или запрет использования оператора GoTo. Элементы программы, файлы, функции, переменные и константы должны отвечать правилам именования; часто применяются и дополнительные соглашения специального характера (скажем, применение венгерской нотации для префиксного указания типа переменной в ее имени). Соблюдение подобных стандартов упрощает управление проектом.

Средства повышения эффективности работы пользователей и сопроводительные материалы

Диапазон средств повышения эффективности работы пользователей широк — от мастеров и документации до материалов для учебных курсов. Проектная группа должна совместными усилиями разработать все материалы, перечисленные в функциональных спецификациях и. работая в тесном контакте с пользователями, добиться стабильности работы промежуточных выпусков продукта.

Вот что пишет Стив Магуайр в своей книге «Debugging the Development Process»:

> *«Если, читая документацию, пользователь не испытывает затруднений, значит разработчики на верном пути. Создавая код, программист должен постоянно помнить о пользователях. Если написанный код кажется программисту недостаточно ясным или эффективным, впечатление пользователя скорее всего будет таким же».*

При тестировании функциональных возможностей промежуточных версий продукта бета-тестеры не должны заниматься теми компонентами, качество кода которых не соответствует критериям качества. В противном случае заведомо некачественные компоненты будут влиять на общее впечатление от продукта, которое складывается у группы тестирования. При бета-тестировании не требуется добиваться идеальной работы приложения — проверяются только конкретные элементы, которые должны быть доведены до группы тестирования.

Тестирование

В этом разделе мы обсудим методы тестирования, которые помогают спроектировать и разработать удачный программный продукт. Мы обсудим тестирование промежуточных версий продукта, а также стратегию отсутствия дефектов, рецензирование кода и ежедневную сборку продукта.

Сводное тестирование промежуточных версий

Без тестирования можно обойтись лишь при «Стабилизации» — во всех остальных случаях это неотъемлемая часть процесса разработки. На этапе одобрения плана проекта проектная группа намечает график тестирования и детально прорабатывает спецификации тестирования отдельных функциональных возможностей продукта.

Создание спецификации тестов надо завершить к этапу «Завершение разработки», поскольку на этом этапе функциональные возможности продукта фиксируются, а их расширение возможно лишь в следующих версиях. Реализацию продукта на этапе «Завершение разработки» следует рассматривать как альфа-версию для стадии «Стабилизация». Не следует путать эту альфа-версию с альфа- и бета-тестированием промежуточных версий на стадии «Разработка». Отметим, что часть операций по тестированию промежуточных версий на стадии «Разработка» повторится на стадии «Стабилизация».

С точки зрения тестирования переход от стадии «Разработка» к стадии «Стабилизация» — это переход от тестирования наличия функций к проверке их работоспособности и эффективности. Приведем некоторые примеры методов тестирования, применяемых на стадии «Разработка».

- **Тестирование модулей** — наиболее распространенный метод тестирования вручную. Отслеживание версий, верификация сборки и регрессионное тестирование, как правило, автоматизируют, поскольку эти операции выполняются регулярно в течение всего проекта. Тестирование модулей и функций позволяет разработчику находить ошибки раньше, чем они попадут к тестерам.
- **Функциональные тесты** — проверка наличия конкретных функций приложения и их работоспособности.
- **Тесты перед сдачей версии модуля** — это простые автоматизированные тесты, которые выполняются перед сдачей новой версии модуля в базы данных исходных текстов проекта.

Иногда используются и другие методы тестирования.

- **Проверка сборки** — обычно выполняется после ежедневной сборки приложения, чтобы удостовериться, что она прошла успешно.
- **Регрессионное тестирование** — автоматизированный тест, проводимый после ежедневной сборки для проверки работоспособности всех функций предыдущей сборки в новой версии.

Теперь рассмотрим примеры методов тестирования, характерных для стадии «Стабилизация».

- **Тестирование конфигурации** — проверяет работоспособность продукта на аппаратно-программной платформе.

- **Тестирование совместимости** — проверка корректности взаимодействия приложения с другими программами.
- **Тестирование под нагрузкой** — работа приложения в условиях критической нагрузки. Цель такого тестирования — определить предельно допустимую нагрузку, включая такие факторы, как использование памяти, дисковой подсистемы, работу в условиях высокой нагрузки на сеть и при большом числе пользователей.
- **Тестирование производительности** — исследование производительности приложения; взаимосвязано с тестированием под нагрузкой,
- **Проверка документации и справочной системы** — выявление ошибок и несоответствий в документации и справочных материалах.

Альфа-тестирование — первая проверка полнофункциональной версии продукта. Бета-тестирование — проверка продукта группой внешних пользователей, часто позволяющая выявить проблемы, незамеченные проектной группой. Несмотря на широко распространенное мнение, что ошибка — это какая-то неточность в программе, программисты считают огрехи программирования лишь частным случаем ошибки. Вообще, ошибкой называется любая проблема, выявленная при использовании разрабатываемого продукта.

Как мы уже отмечали в главе 10, классификация ошибок позволяет определить их приоритет и опасность. При этом следует изучать серьезность проблемы (влияние неисправленной ошибки на работу приложения) и ее приоритет (важность исправления ошибки для сохранения стабильности приложения). Наличие ошибок со степенью серьезности или приоритетом 1 (они определены в главе 10) следует рассматривать как препятствие выпуску текущей версии приложения,

Ориентация на отсутствие дефектов

Как мы уже говорили, ориентация проектной группы на отсутствие дефектов — важнейший фактор успеха проекта. Кроме того, ориентация на отсутствие дефектов свидетельствует о том, что проектная группа стремится создать продукт высшего качества. Именно для этого на стадии разработки постоянно проводится контроль качества.

Ориентация на отсутствие дефектов не означает создания абсолютно бездефектного продукта. Это лишь цель, которую ставит перед собой проектная группа. Если все же продукт считается *бездефектным,* это не означает, что вы не найдете в нем ни одного дефекта — достаточно, чтобы продукт отвечал заданным критериям качества. Бездефектные этапы требуют соответствия продукта установленным критериям качества — в противном случае этап считается не завершенным,

Основное достоинство этого способа — высокий приоритет качества продукта при его разработке. Поскольку высокое качество — ос-

новное требование заказчика, ориентация на отсутствие дефектов одновременно настраивает группу на выполнение требований заказчика.

Ежедневная сборка

Хотя ежедневная сборка приложения — хлопотное дело, она очень полезна. Это просто компиляция и сборка всего кода приложения в исполняемый модуль (или модули). Как следует из названия, эта процедура выполняется ежедневно. На практике обычно сборка проводится через три-четыре дня (но не реже). В своей книге «Dynamics of Software Development» Джим Маккарти отмечает:

> *«Разрабатывая программное обеспечение, легко попасть под влияние иллюзий, однако ежедневная сборка — прекрасное (и совершенно безопасное) средство от них».*

Одно из главных достоинств ежедневной сборки — постоянная доступность приложения всем участникам проекта, что позволяет в любой момент проверить фактическое положение дел. Ежедневная сборка показывает степень готовности проекта в целом, а не только его отдельных частей.

По образному сравнению Джима Маккарти из цитированной выше книги, ежедневная сборка — сердцебиение проекта:

> *«Если сборка провалилась, всем сразу ясно, что дело плохо, и нужно принимать срочные меры».*

Это очень удачная метафора. Достоинств у ежедневной сборки много, но самое главное, что она позволяет контролировать состояние продукта в течение всего процесса. Кроме того, регулярная сборка всех компонентов продукта в единое целое позволяет быстро выявлять неготовые или неработоспособные элементы. Если некоторые элементы не удается присоединить к общему целому, выявляются проблемы интеграции продукта. Необходимость объединения отдельных компонентов в единое целое заставляет отвечающие за них подгруппы синхронизировать свою работу.

И наконец, ежедневная сборка позволяет регулярно проверять состояние и качество продукта. Все большая степень готовности приложения с каждой следующей сборкой — прекрасный стимул для проектной группы и заказчика. Регулярность и частота сборки позволяет быстро выявить проблемы и обеспечивает необходимую степень синхронизации усилий всех членов проектной группы,

Резюме

Разработку удачных программных продуктов часто считают искусством, однако этому можно и нужно учиться. В предыдущих главах мы обсудили множество методов, доступных проектной группе на ста-

дии разработки. Их применение позволяет проектной группе достичь этапа «Завершение разработки».

Достижение этого этапа говорит о готовности проектной группы приступить к завершению разработки продукта, так сказать, к нанесению последних штрихов. Все задачи, поставленные в функциональных спецификациях, решены, и проектная группа готова перейти к фазе «Стабилизация». На этой, следующей за разработкой стадии основное внимание уделяется управлению рисками как основному инструменту создания необходимого продукта.

Закрепление материала

1. Перечислите результаты стадии «Разработка».
2. Что такое промежуточная версия продукта?
3. Перечислите преимущества выпуска промежуточных версий продукта.
4. Перечислите три метода устранения ошибок.
5. Когда проектная группа готова к переходу на стадию «Стабилизация»?

Практикум 9. Разработка

– Всем привет, давайте начнем. Как у нас дела?

Было утро 26 апреля. Группа уже неделю занималась стадией «Разработка». Если не считать проблему с сервером AutoCAD, то неделя сложилась удачно. Все допили свой кофе и заняли места за столом.

– Итак, как у нас дела? — повторил свой вопрос Дэн. — Давайте вспомним свои роли, чтобы каждый знал, на что ему обратить внимание во время обсуждения.

Дэн вставил в проектор слайд с описанием ролей и обязанностей разных подгрупп на стадии «Разработка» и повернулся к аудитории.

Отчет менеджера продукта

— Джейн, начнем с тебя. Как менеджер продукта ты отвечаешь за взаимодействие с заказчиком, за соблюдение его интересов и за подготовку первого бета-выпуска. Как обстоят дела?

– Очень даже неплохо, — ответила Джейн, раскрывая папку. – Мне очень понравилось, как ты это сформулировал, когда мы собирались на прошлой неделе, чтобы обсудить фазу «Разработка»: заказчик должен знать, чего ему ждать, и должен получить именно то, что ожидает. Я уже подготовила материал о нашем проекте для корпоративной газеты, на узле нашей интрасети есть страничка, посвященная RMS (ребята из отдела информационных систем сделали ссылку на нее на своей странице). Мы с Джимом подготовили и разослали

информацию всем менеджерам, которые будут пользоваться административным клиентским ПО; мы сообщили, что продукт готовится к выпуску, когда и чего следует ждать. Кроме того, я подобрала для Билла и его ребят пользователей: одну группу, поменьше, — для первого тестирования и еще одну, побольше, — для второго.

Фергюсон **и Барделл**

Строительство • Проектирование • Управление проектами

Проект создания системы управления ресурсами

Повестка дня

Дата: 26 апреля 1999 г. **Тема:** разработка

I. Уточнение повестки

II. Сообщения членов группы
- Менеджер продукта
- Менеджер программы
- Разработчик
- Тестер
- Инструктор
- Логистик

III. Вопросы и ответы

IV. Задания и ответственные

Чикаго • Детройт • Милуоки • Цинцинати • Индианаполис • Луисвилль

- Похоже, ты ни на Йоту не отступаешь от плана, — сказал Дэн, и присутствующие одобрительно закивали. — Всего один вопрос: что ты думаешь по поводу отсутствия бухгалтерских функций в приложении

и что сделала по этому поводу? Я знаю, что в твоей группе некоторые надеялись, что такие функции будут, и они наверняка разочарованы.

– Пришлось выдержать целую дискуссию, — ответила Джейн. – Мне кажется в такой ситуации не следует увиливать — это все равно не поможет. Я просто рассказала, что на эту работу сейчас нет ни средств, ни времени, но мы постараемся реализовать эти функции во второй версии, которую планируем выпустить до конца года. Наверное, не все мне поверили, но, во всяком случае, пока они не высказывают своих сомнений. Вероятно, причина в том, что мы информируем их о проекте намного больше, чем раньше. Кстати, Билл, — сказала она, повернувшись, — теперь мое честное имя в твоих руках: если ты не добавишь бухгалтерский модуль во вторую версию, мне не поздоровится.

— Я простой исполнитель, — улыбнулся Билл и показал рукой на Дэна. — Скажи боссу — он подписывает мои чеки.

— На самом деле твои чеки подписывает Джейн, да еще и печать на них ставит, — ухмыльнулся Дэн. — Да и мои, между прочим. Так что придется нам с тобой делать бухгалтерский модуль во второй версии.

– Приятно сознавать, что все понимают, кто в доме хозяин. – рассмеялась Джейн.

Отчет менеджера программы

— Идем дальше, — продолжил Дэн, глядя на слайд. — Менеджер программы — это я, поэтому расскажу, чем я занимался. На стадии «Разработка» мне пришлось выполнять три дела: контролировать проект, поддерживать связь со всеми участниками группы и координировать работы, а также планировать выпуск бета-версии.

Склонившись над столом, он оглядел всех участников совещания:

— Мне нужна ваша помощь. Я решил, что на стадии «Разработка» эффективнее заменить общие встречи на беседы «один на один», а всем вместе собираться только раз в неделю, чтобы координировать работу. Я надеялся, что с глазу на глаз разговаривать более полезно и эффективно, поскольку можно обсудить именно то, что делает конкретный сотрудник, не тратя впустую время всех остальных. Именно этим я и занимался всю прошедшую неделю и теперь хочу знать, было ли это полезно, нравится вам этот метол или стоит вернуться к общим встречам?

Никто не хотел отвечать первым, и в зале воцарилось молчание, Наконец Марта сказала:

— Я не могу отвечать за других, но мне больше нравятся встречи «один на один», — все остальные утвердительно кивнули, и Марта продолжила: — Две встречи, которые у нас с тобой были на этой неделе, оказались очень полезными. Каждая из них помогла мне лучше

понять, что я должна делать, и это заняло меньше времени, чем при общей встрече. Я считаю, что этот способ лучше.

— Я думаю, что знаю, почему встречи «один на один» раньше не работали, а теперь вдруг стали эффективными, — сказала Мэри-Лу. — На стадиях разработки концепции и планирования мы должны были работать вместе, чтобы добиться общего понимания проекта и выработать общий план, Полезно было выслушать точки зрения остальных, найти компромиссы и помочь друг другу. Кроме того, нам требовалось почувствовать себя командой, — она сделала небольшую паузу и продолжила: — Теперь мы делаем конкретную работу, у каждого своя задача, и нам не обязательно собираться всем вместе. Но нам важно не потерять командный дух; мы должны помогать друг другу хотя бы пониманием того, что каждый всерьез делает свою работу. Я думаю, что сочетание встреч «один на один» и общей встречи раз в неделю — именно то, что нужно на этой стадии проекта.

— Хорошо сказано, — заметил Билл, не поднимая глаз от стола.

— И рассылка новостей по электронной почте — тоже хорошая идея, потому что мы больше не встречаемся так часто, — добавил Тим и продолжил, улыбаясь: — Жаль, что пончики по почте не пошлешь.

Дэн засмеялся:

— В следующий раз, когда я буду связываться с Microsoft, я спрошу, не планируют ли они включить рассылку пончиков в функциональные спецификации следующей версии Exchange.

Отчет инструктора

Дэн бросил взгляд на слайд с перечислением ролей и продолжил:

— Я хотел бы оставить Билла и Марту напоследок, поэтому давайте послушаем инструктора. Мэри-Лу, как идут твои дела?

Заглянув в свою папку, Мэри-Лу ответила:

— Как я уже говорила на прошлой неделе, в мои обязанности входит подготовка материалов для пользователей, включая документацию и справочные буклеты. Кроме того, я отвечаю за тестирование удобства продукта для пользователей.

Она достала из сумки маленькую папку.

— Вот начало документации для пользователей. В пятницу я встречалась с разработчиками. Они показали мне уже готовые фрагменты и отдали несколько снимков с экрана, над которыми я и поработала в выходные, — она повернулась к Биллу: — Билл, я не могу закончить документацию, пока ты не скажешь мне, что пользовательский интерфейс зафиксирован, и не дашь снимки всех экранов. Когда это случится?

— Надеюсь, что скоро, — ответил Билл, глядя на график проекта. — Мы должны закончить альфа-версию в среду. Я прослежу, чтобы к этому сроку интерфейсы были закончены, и сообщу тебе по почте. Если все пройдет по плану, в четверг утром сможем вместе взяться за работу.

- Хорошо бы, — кивнула Мэри-Лу.

— А что с тестированием удобства продукта для пользователей? - спросил Дэн.

- Мы вместе с Мартой работаем над тестовыми сценариями, - ответила Мэри-Лу, — но не сможем их закончить, пока не появится первая бета-версия, — она бросила взгляд на свои записи и добавила: - Еще мы с Джейн работаем над планом бета-тестирования, что тебе уже известно.

Дэн кивнул:

— Неплохо, Мэри-Лу. Я рад, что ты занимаешься документацией, поскольку тебе же придется обучать менеджеров пользоваться системой. Джейн показала мне твои наработки для занятий. Должен сказать, выглядят они впечатляюще.

Отчет логистика

Дэн повернулся к Тиму:

- Ну, теперь твоя очередь. Кстати, ты больше серверы не ломал? — он ловко уклонился от брошенного Тимом пончика.

- Нет, не ломал и, будь уверен, больше не подпущу ни Билла, ни его любимчиков ни к одному производственному серверу, — раздраженно бросил Тим.

— О чем это вы? — заинтересовалась Джейн.

- Просто с подачи вашего босса сетевой менеджер и начальник отдела разработки получили небольшой урок, — ухмыльнулся Дэн. — Надеюсь, все сделали выводы, и больше мы об этом говорить не будем. — Вновь став серьезным, он спросил: — Как поживает наша тестовая лаборатория?

- Новые машины еще не пришли, однако мы собрали кое-какое оборудование и уже перенесли в помещение лаборатории, — ответил Тим. — Сегодня один из моих ребят настраивает его, В обшем, если новые машины поступят на следующей неделе, как мы рассчитываем, мы успеем подготовить их до выхода бета-версии.

— Отлично, Тим. Надеюсь, все пройдет по плану. — Дэн заглянул в свои записи и спросил: — А как документация для группы сопровождения?

Тим смутился.

— Ох, я так и думал, что что-нибудь да забуду. Прошу прощения, Дэн, — смиренно произнес он, — я за нее еще не принимался.

Дэн кивнул:

- Не переживай, Тим, у тебя еще масса времени. Да ты и не сможешь сделать документацию, пока не будет готова тестовая система, а группа разработки не закончит свою работу. — Сделав пометку в дневнике, он продолжил: — Если они будут работать по графику, на подготовку документации у тебя есть две недели. Хватит?

- Вполне, — кивнул Тим.

Отчет группы разработки

Дэн повернулся к Биллу;

- Ну, чиф, сцена в твоем распоряжении. Расскажи нам, как идет настоящая работа над приложением RMS — или, лучше сказать, приложениями?

Билл открыл лежавшую перед ним папку сантиметров в восемь толщиной, набитую листингами, снимками с экрана, планами и графиками работ, и вытащил из нее четырехстраничную диаграмму, покрытую значками и каракулями.

- Основную часть работ ведут Бэт и Сэм, однако им помогают еще четыре человека из нашего отдела. Мы разделили проект так: Бэт делает бизнес-уровень SQL Server, Win32-клиент на Visual Basic и интерфейс Outlook для Web-клиента. Сэм занимается бизнес-уровнем Exchange, проектирует базу данных и уровень доступа к данным, а также отвечает за Web-интерфейс для расписаний.

Мы решили начать работу с проектирования базы данных SQL, а потом разработать все бизнес-объекты, Как только эта работа будет закончена, мы сможем параллельно заниматься и уровнем доступа к данным, и пользовательским уровнем. Сейчас практически все бизнес-объекты готовы, мы успели справиться и с другими объектами. — Он посмотрел на Мэри-Лу: — Понимаешь, выбранный нами порядок работы таков, что я не знаю, когда мы сможем «заморозить» пользовательский интерфейс; боюсь, это случится в самый последний момент. Единственное, что я могу сделать — это попросить Сэма и Бэт сначала закончить с клиентами, а потом уже заниматься уровнем доступа к данным.

Мэри-Лу кивнула, а Дэн спросил:

- Пришлось ли вносить изменения в проект, который мы разработали на стадии «Планирование»?

— Ничего серьезного, — ответил Билл, отыскав другую страницу в своей папке. — Единственное внесенное нами изменение — это новые методы, о которых мы забыли при проектировании. Например, менеджеры должны утверждать расписание. Мы предусмотрели метод, который позволяет сотруднику передать расписание менеджеру на подпись, однако забыли метод, который позволил бы менеджеру

запросить все расписания на проверку. Поэтому мы добавили такой метод и внесли соответствующие коррективы в проект.

— А что с MTS? — спросила Мэри-Лу. — Вы активно используете транзакции?

— По всему бизнес-уровню, — ответил Билл, — Наше главное правило — когда мы записываем данные в базу данных SQL, хоть новые, хоть измененные, мы используем транзакцию.

— Я знаю, что такое бухгалтерская транзакция, — сказала Джейн, — но вот что такое MTS?

- Microsoft Transaction Server, — пояснила Мэри-Лу, а затем коротко, но очень доходчиво обрисовала, что такое транзакции и как они используются в RMS.

Когда она закончила, Билл сказал:

- Ты сегодня уже два раза отвечала на вопрос так, что понял даже я. Ты очень хороший инструктор, Мэри-Лу.

— Не забывай, Билл, тебе и нашему блестящему инструктору нужно на стадии «Разработка» работать вместе. Однако из некоторых сегодняшних замечаний я понял, что она пока оказалась вне игры. Это так?

Билл расстроился:

— Я так и знал, что ты это скажешь. Я просто работаю, как привык.

— Я знаю, — мягко ответил Дэн, — и не заставляю тебя отказываться от этого. Но инструктор должен работать вместе с вами — это единственный способ быстро подготовить материалы для пользователей и группы сопровождения и ускорить обмен мнениями. Ты же понимаешь, что Мэри-Лу — настоящий профессионал. Она не будет «давить на психику», по крайней мере, пока в этом не возникнет необходимость — впрочем, и это, по моему, только сделает конечный продукт лучше, чего, кстати, мы все и добиваемся. Когда мы закончим, договорись с Мэри-Лу — я хочу, чтобы она точно знала, когда у вас следующая встреча и обязательно присутствовала на ней.

Билл, казалось, был готов дать отпор, но, заметив железные нотки в голосе Дэна, только вздохнул.

— Больше всего это похоже на усиление бюрократии, — сказал он, — но, поскольку мы все делаем по книжке, спорить не приходится.

- Спасибо, Билл, — сказал Дэн. — Ты прав, мы все делаем по книжке — и чтобы научиться, и чтобы показать всем, что MSF работает. В следующих проектах мы будем менять модель, как нам удобно. Пока же давайте научимся ею пользоваться.

Отчет группы тестирования

Дэн снова заглянул в свои записи:

- Я оставил тестирование напоследок, поскольку хотел, чтобы каждый получил представление о том, чем в данный момент занима-

ются остальные. Марта, почему бы тебе теперь не рассказать нам о тестировании RMS?

Марта вытащила из сумки ворох бумаг.

- Как вы все знаете, — сказала она, — разработка программного обеспечения — не моя специальность, поэтому я согласилась на роль тестера с некоторым опасением.

- Мы помним, — сухо ответил Тим.

— Дэн оказался превосходным учителем, — продолжила Марта, — и, к тому же. помог мне скоординировать работу группы тестирования и группы разработки. Помог и Билл, хотя, по-моему, он не совсем рад моему присутствию в группе.

- Ничего личного, Марта, — быстро отозвался Билл. — Ты компетентный и умный сотрудник, и это самое меньшее, что я могу сказать. Я просто думаю, что тестирование работоспособности приложения — обязанность группы разработки. Но, поскольку у Дэна другое мнение, я решил извлечь максимум из такой ситуации.

- Ну, и чем же этот старый скряга помог тебе, Марта? — ухмыляясь, спросил Тим.

— Они с Дэном решили, что нужно прикрепить ко мне программиста в качестве помощника по тестированию. Он сам не занимается проектом RMS, однако консультирует меня по всем техническим вопросам. Я составляю планы и отвечаю за контроль работы и информирование группы, а Майкл — так его зовут — помогает с реальным тестированием. Или, точнее, с той его частью, которая достается нам.

- Я думала, вы тестируете приложение полностью, — заметила Джейн.

— На самом деле, нет, — ответила Марта. Она повернулась к Дэну: - Я подготовила несколько слайдов, чтобы разъяснить процедуру тестирования, если ты не против.

- Конечно же, нет, — ответил Дэн. — Мы все внимание.

Марта быстро объяснила, в чем отличие тестирования наличия функций и их работоспособности, и перечислила конкретные методы, применяемые в обоих случаях. Она подчеркнула, что тестирование модулей — обязанность разработчиков, и отметила, что они же выполняют и функциональные тесты,

— Однако, когда они заканчивают разработку набора функций, — сказала она, — они передают продукт нам для проверки работоспособности.

- Как это, ведь не все объекты готовы? — спросил Тим,

- Они просто пишут «заглушки», которые возвращают управление, — ответила Марта. — Иногда они добавляют окно сообщения, чтобы было видно, что библиотека была вызвана. Для RMS это не так

важно, как для большого проекта, — когда они закончат все, продукт все равно попадет к нам на окончательное тестирование. Если мы что-нибудь обнаружим, найти источник проблемы не составит труда — приложение, слава Богу, не такое уж большое.

Тим вопросительно посмотрел на Билла:

— У Марты в списке есть верификация сборки, но, по-моему, ты ни разу не упоминал, что вы делаете ежедневную сборку.

— А мы и не делаем, — ответил Билл, откинувшись в кресле. — Ежедневная сборка бывает только в книжках, а на практике никто этим не пользуется. Мы как-то попробовали этот метод в одном из предыдущих проектов, однако быстро поняли, что частота сборки должна соответствовать масштабу проекта. В этом проекте мы делаем сборку раз в три-четыре дня.

— И еще, — добавила Марта. — поскольку проект небольшой, мы не выполняем регрессионное тестирование.

— Так чем же тогда заняты вы с Майклом? — спросила Джейн. — Похоже, всю работу делает группа разработки.

— Так и есть, — с улыбкой ответила Марта. — Мы решили, что так будет лучше. Если помните, тестирование играет главенствующую роль только на стадии «Стабилизация». Пока же мы готовим тесты и пропускаем все, что поступает из группы разработки, через нашу тестовую машину, чтобы подготовиться к тестам под нагрузкой и тестам производительности. Кроме того, Майкл и я работаем над базой данных ошибок.

— База данных ошибок? Ты ни разу об этом не упоминала, — Билл от удивления даже выпрямился в кресле.

— Погоди, Билл, до ошибок мы еще доберемся, — сказал Дэн. — Я хочу поподробнее услышать о тестовой машине. Марта вскользь о ней упомянула, но меня интересуют детали.

— Вообще, это отличное изобретение, — улыбнулась Марта. — Майкл написал ее за выходные, чем меня совершенно поразил, однако он заявил, что в этом нет ничего нового. На самом деле это автоматизированный тест компонентов. Мы вводим имя библиотеки, выбираем нужный метод, задаем его параметры и указываем, сколько раз его нужно вызвать. Тестовая машина вызывает метод столько раз. сколько мы просили, и протоколирует результаты в файле. Кроме того, она отслеживает использование памяти, использование диска и другие полезные параметры. Если мы знаем интерфейс, мы можем автоматически протестировать любой объект. Майкл собирается расширить ее так, чтобы она контролировала весь набор функций, но у него пока не было времени, Пока что мы тестируем компоненты, которые нам передают разработчики.

— Были проблемы? — спросила Мэри-Лу.

— Нет, сотрудники Билла пишут практически безошибочный код, — ответила Марта.

— Вот именно, и поэтому не нужно никакой базы данных ошибок, — разгорячился Билл. Глядя на Дэна, он продолжил: — Я думаю, что все разговоры об ошибках, не говоря уже о базе данных ошибок, просто обижают Сэма, и Бэт, и всех остальных, кто работает над проектом. Она же сказала, что получает от нас безошибочный код!

— Это верно, но ведь и безошибочный код может породить проблемы, — ответил Дэн.

Билл оглянулся вокруг, ища поддержки. Неужели они не видят, что у их начальника вдруг помутился рассудок? Потом, повернувшись к Дэну, он сказал:

— Так не бывает, какие могут быть ошибки в безошибочном коде?

— Дай. пожалуйста, определение ошибки, — попросил Дэн Марту.

— Ошибка — это любая проблема, возникшая при использовании продукта, — без раздумий ответила Марта.

— Это обязательно изьян в программе?

— Нет, хотя может быть и изьяном.

— Что нужно сделать со всеми проблемами, возникающими при использованииподукта?

— Их надо проанализировать и разрешить до выпуска продукта.

— Можно привести пример проблемы, которая не является ошибкой программирования?

Марта стала загибать пальцы:

— Запросы улучшения или расширения функциональных возможностей; предложения, выходящие за рамки текущего выпуска; проблемы, вызванные предпочтениями пользователей; наконец, неустранимые последствия архитектурных решений.

— Видишь, Билл, — сказал Дэн, подходя к доске, — иногда и превосходный код не лишен проблем. А справляться с проблемами - лело группы тестирования, хотя она не обойдется без твой помоши и помощи всей проектной группы.

Дэн нарисовал на доске круг.

— Вот это процесс отслеживания проблем. Он, кстати, напоминает методику отслеживания рисков. Сначала тестер — Марта, Майкл или кто-то другой из бета-тестеров — сообщает о наличии проблемы. Мы заносим ее в базу данных. Билл, как глава группы разработки, присваивает проблеме приоритет и код серьезности и назначает разработчика, ответственного за ее устранение. Как только у разработчика появляется уверенность, что проблема разрешена, он передает ее обратно тестеру для проверки. Если все в порядке, проблема считается устраненной; если нет, процесс повторяется, — Дэн повернул-

ся к Марте: — У тебя есть слайд с классификацией приоритетов и серьезности проблем?

Она кивнула и передала Дэну слайд, который он вставил в проектор.

— Как видите, мы классифицируем все проблемы по приоритетам и серьезности, — сказал он, написав на доске слова *«критерий качества»*. — Группа тестирования руководствуется критериями качества, установленными для выпуска продукта. Кстати, каковы критерии качества для RMS, Марта?

— Отсутствие проблем с кодом серьезности и приоритетом 1 или 2, — без запинки ответила Марта.

— Итак, — сказал Дэн, возвращаясь к столу, — именно группа тестирования решает, сможем мы выпустить продукт в срок или нет. Если они решат, что мы сделали свою работу недостаточно качественно, нам придется переделать ее. Эта работа начинается на стадии разработки и завершается на стадии стабилизации. Вопросы?

— Только один, — тон Билла несколько смягчился. — Что, если мы не согласимся с мнением группы тестирования? Что, если мои ребята считают, что все сделано качественно?

— Тогда мы просто выносим вопрос на общее обсуждение, и решение примет проектная группа, — спокойно ответил Дэн. — И помни, Билл, у группы тестирования та же задача, что и у группы разработки, у тебя и у меня — выпустить продукт настолько качественным, насколько это возможно.

Задачка

Дэн сидел в офисе, когда дверь приоткрылась и в комнату заглянул Билл. — Тук, тук, тук, кто-нибудь дома?

— Входи, — сказал Дэн. — В чем дело, чиф?

— У меня извинения и вопрос, — сказал Билл. — У тебя найдется время и для того, и для другого?

Дэн вышел из-за стола, подвинул одно из стоявших перед ним кресел и, показав на него рукой, пригласил Билла сесть. — Не вижу причин для извинений, Билл, а вот для вопроса начальника отдела разработки у меня всегда есть время. В чем дело?

Билл поерзал на стуле:

— Я хочу извиниться за мои комментарии сегодня днем. Теперь они кажутся мне слишком грубыми и резкими, как всегда, и уж точно для них не было никакой причины.

— Извинения приняты; — со смехом ответил Дэн, — однако, честно говоря, мне не показалось, что ты был уж так резок.

— Наверное, вы все уже просто ко мне привыкли, — *с* облегчением вздохнул Билл. — Однако это не значит, что я могу так себя вести, и поэтому хотел извиниться.

– Что тебя волнует, чиф? По-моему, теперь уже все поняли, что мы одна команда и делаем одно дело.

— Я это понимаю, но, к сожалению, только тогда, когда об этом думаю. Может быть, дело в том, что мы работаем не так, как раньше. Может, дело в жестком графике. Ты же сам сказал, что мы работаем «под прессом», а теперь вся моя группа — а я сильнее всех — чувствуем это «пресс» на своей шкуре. А теперь вопрос.

Билл вздохнул, помолчал и продолжил:

– Ты знаешь, что основные разработчики проекта — Бэт и Сэм. Вот с ними-то у меня и проблема, и мне нужен твой совет.

Дэн молча ждал.

– Я думаю, — сказал Билл, — лучше всего сформулировать это так: Бэт и Сэм затягивают работу, но каждый по-своему. Боюсь, если я не найду решения, мы не впишемся в график.

Дэн присвистнул:

— Дело серьезное. Я рад, что ты пришел ко мне, хотя честно скажу, я не ожидал ничего подобного. По-моему, они оба — отличные программисты. Что должно было случиться, чтобы они превратились в тормоз?

— Да в том-то и дело, — сказал Билл. — Понимаешь, они как бы зеркальное отражение друг друга. Вот, скажем, Сэм. Он превосходный программист, что называется, от Бога. Он всегда находит такие решения, глядя на которые можно только восхищаться. Это и есть его сильная сторона — кажется, он интуитивно чувствует, как должно быть устроено приложение. В результате он работает быстрее всех остальных.

– Так в чем же тогда проблема? — удивленно спросил Дэн.

– Все очень просто, — ответил Билл. — Как только он нашел решение, его больше ничего не интересует. Он делает глупые ошибки, не проверяет свой код, и нам приходится чистить его программы. Сам понимаешь, это замедляет всю работу.

— С такими людьми я уже сталкивался, — сказал Дэн. — А в чем же дело с Бэт? Уж она-то точно не делает детских ошибок.

— О да, она пишет безукоризненный код, и в этом-то и состоит ее проблема, — увидев, что лицо Дэна вопросительно вытянулось, он пояснил: — Она пишет превосходный код, Дэн, но на это уходит слишком много времени. Пока она пишет один компонент, Сэм успевает сделать четыре. Когда ее работа готова, она полностью документирована, все отлично работает, и придраться не к чему — однако, если она будет продолжать в том же темпе, ей понадобится еще года два. Я просто не знаю, что делать.

— Хм, — Дэн встал, подошел к окну и некоторое время стоял молча. Потом он повернулся и, прислонившись к подоконнику, сказал: — Понимаешь, Билл, в этом нет ничего удивительного. Большинство программистов принадлежат к одной из этих категорий — они либо гении, которые не желают заботиться о деталях реализации, либо буквоеды, работающие точно по правилам и пишущие больше комментариев, чем кода. Выход один — скрестить их друг с другом. — Он на минуту задумался и щелкнул пальцами: — Да, именно скрестить!

- Как-как?

— Скрестить, и это единственный выход.

— Слушай, Дэн, может ты и прав, но, по-моему, у Сэма уже есть девушка.

- Нет, я имел в виду совсем другое. Я хочу сделать так, чтобы их сильные стороны дополняли друг друга, одновременно компенсируя их слабости, Вот что нужно сделать, — сказал он и вкратце изложил свой план.

- Превосходная идея, — восхищенно сказал Билл. — Знаешь, это одна из лучших идей, какие мне приходилось слышать. Теперь я не удивляюсь, что ребята с твоей предыдущей работы отзывались о тебе так высоко.

Дэн удивленно поднял брови:

— А ты откуда знаешь?

Поднявшись из кресла, Билл с улыбкой ответил:

- Не только у тебя есть друзья в других компаниях.

Регрессионное тестирование

- Входите оба, — сказал Дэн.

Марта и Майкл вошли в комнату и сели к небольшому столу, Дэн закрыл записную книжку и присоединился к ним.

- Как идет тестирование, Марта?

- Вот схема со всеми элементами, тестирование которых мы запланировали, и результатами, — ответила Марта, передавая Дэну стопку скрепленных вместе листков.

- О, выглядит убедительно, — присвистнул Дэн, перелистывая страницы. — Как вы это сделали? Больше всего похоже на отчет Access.

- Так и есть, сэр, — ответил Майкл. — Когда мы стали пропускать через тесты все большее число модулей, контролировать их на бумаге стало все труднее, поэтому я сделал простую базу данных Access. Кроме того, я модифицировал тестовую систему так, чтобы результаты заносились прямо в нее. Это, кстати, позволяет сравнивать результаты при повторном тестировании тех же модулей.

— Ага, простое регрессионное тестирование. Хорошая мысль, — одобрительно сказал Дэн. — Я вижу, вы перечислили и эталонные результаты по всем метрикам.

— Да, мы не видели смысла проводить тестирование, если не с чем сравнивать.

Дэн с интересом переворачивал страницы, просматривая описания тестов и их результаты. Он поднял голову и, взглянув на Марту, спросил:

— Что это за тест на четвертой странице, обведенный кружком? Почему нет результатов?

— Именно об этом мы и хотели поговорить. Тест, на который ты смотришь, касается предложения Билла на последнем общем собрании. Помнишь, он просил предусмотреть метод для вывода списка всех расписаний, поданных на утверждение?

Дэн кивнул:

— Конечно помню. Мне не показалось, что это так уж важно. В чем же причина падения производительности, которая, если не ошибаюсь, видна на графике?

Марта неуверенно поерзала в кресле, а затем посмотрела на Майкла:

— Почему ты сам не расскажешь, Майкл? Я не хочу, чтобы из-за моего недопонимания кто-то еще «сел в лужу».

Прежде чем Майкл успел раскрыть рот, Дэн сказал:

— Нет, Марта, ты руководишь тестированием, и это твоя обязанность четко и ясно описать проблему. Если вы ошиблись при тестировании, мы сможем выяснить это только при анализе проблемы. А если вы правы, то просто хорошо выполнили свою работу и предупредили выпуск неправильно работающего продукта. Так что рассказывай сама, не скрываясь за спиной Майкла. В чем здесь дело?

Марта взяла схему и показала несколько цифр.

— Когда мы начинали тестирование, мы не пользовались метриками для конкретных объектов, а просто фиксировали результаты. Когда же мы приступили к тестированию функций, мы задались эталонными значениями времени отклика и других важных параметров.

— Пока все нормально. Что же случилось?

— Для метода, собирающего список всех расписаний, мы задали предельное время отклика 5 секунд. Смотри, вот в этой колонке.

— Ага, а в следующей колонке тогда должен быть результат теста, правильно?

Марта кивнула:

— Именно. Как видишь, ответ поступает примерно через 2 секунды.

Дэн был явно удивлен. — В чем же проблема? Требования выполнены, причем с запасом. Я. наверное, чего-то не понял,

Майкл покачал головой:

- Нет, сэр, вы все поняли правильно, просто мы еще не успели рассказать о другом тесте, который мы провели.

Дэн улыбнулся:

- Иначе говоря, помолчи и дай нам закончить.

Майкл ухмыльнулся:

- Ну, я не хотел говорить именно так, но вы совершенно точно оценилиситуацию.

Дэн расхохотался:

- Кое-кто обвиняет меня в излишней склонности к дипломатии, однако, похоже, они просто не знакомы с тобой. Хорошо, я помолчу, а вы расскажите о втором тесте.

Марта продолжили:

— Результаты первого теста основаны на тестовых данных, которыми пользовался Билл и его группа при разработке. Майкл и я задумались, что будет, если воспользоваться набором данных побольше. В конце концов, в компании больше 800 сотрудников, поэтому расписания за весь год — это таблица с 40 000 записей только заголовков, не говоря уже о самих расписаниях. Мы решили выяснить, что будет.

- Итак, — взял слово Майкл, — я создал базу данных SQL Server, потом сделал примерно 30000 расписаний со случайными данными. Пришлось часок подумать, как сделать данные действительно случайными, чтобы тест получился корректным. Потом мы проверили компонент на этой базе данных.

- И в результате, — объявила Марта, — все изменилось. Да не просто изменилось, а сильно ухудшилось. Вот число, в последнем столбце.

- Пятнадцать секунд?! — Дэн не мог поверить своим глазам. - Кошмар! Да с таким временем отклика большинство сотрудников решат, что машина зависла и перезагрузят ее. Это никуда не годится, — он замолчал и взглянул на Марту: — Но я по-прежнему не понимаю, причем тут вы. Это же проблема Билла и его команды.

Марта огорчилась еще больше:

— Должна быть, но они не соглашаются.

— А, теперь я все понял, — спокойно сказал Дэн. Он на минуту задумался, а потом спросил: — Ну и что они говорят: что результаты неверны или что метрика некорректна?

— И то, и другое, — ответил Майкл, чуть пожав плечами. — Даже после того, как я объяснил Сэму всю процедуру тестирования в под-

робностях, он заявил, что я сам не понимаю, что говорю. Они считают, что 10 секунд — вполне приемлемое значение, в том числе и для нового сервера, где будет работать SQL Server.

— Я думаю, что 10 секунд — это верхний предел, особенно учитывая, что в следующем году штат компании увеличится до 1200 сотрудников. Однако их слова по поводу нового сервера кажутся разумными. — он на некоторое время замолчал, задумавшись, а потом сказал: - На вашем месте я бы сделал вот что, — он написал в блокноте несколько фраз. Закончив, он повернулся к Марте: — Ну как, я вас убедил?

Она уныло посмотрела на босса и ответила:

- Попробуем, однако я не думаю, что Билл согласится.

- Марта, он не взбесившийся гризли, а ласковая медведица, оберегающая своих медвежат от всяческих опасностей. Покажите ему, что он не прав, и он поймет это. Просто вам придется некоторое время терпеть возмущенное рычание.

Завершение разработки

— О, эклеры!

Тим кинулся к столу и схватил по пирожному в каждую руку:

— Что у нас за праздник, босс? Ты наконец-то научился попадать клюшкой по мячику?

- Нет, любитель пончиков, причина не в этом, хотя в выходные я набрал 100 очков, — ответил Дэн.

Джейн фыркнула:

- Ты, похоже, еще не проснулся, Тим. Мы же собрались по поводу завершения фазы «Разработка»!

— Я и правда забыл, — сконфуженно признал Тим и приветственно вскинул руку с зажатым эклером в сторону вошедшего в комнату Билла: — Мои поздравления, чиф! Эклер?

Билл с отвращением посмотрел на изуродованное пирожное, а все остальные рассмеялись:

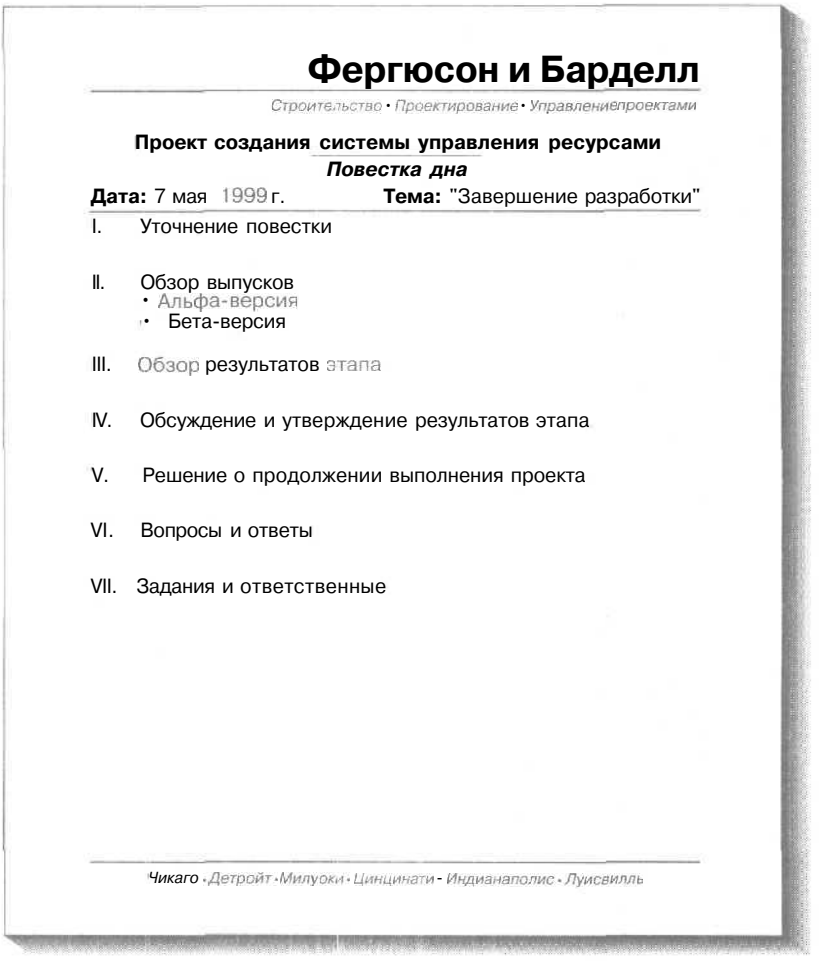
— Нет уж, Тим, тебе придется доесть это самому. Похоже, ты крепко взялся за дело.

- М-м-м, — ответил Тим, запихнув весь эклер в рот.

- Итак, все собрались, можем приступать, — сказал Дэн и заглянул в лежащий перед ним план. — Как обычно, прежде всего пройдемся по программе собрания. Есть вопросы или дополнения?

— У меня есть дополнение, — сказал Билл. — Когда закончим, я хотел бы сказать пару слов.

— Хорошо, Билл, — сказал Дэн, записывая имя Билла в план собрания и попутно раздумывая, что бы это значило.



Фергюсон и Барделл

Строительство • Проектирование • Управлениепроектами

Проект создания системы управления ресурсами

Повестка дня

Дата: 7 мая 1999 г. **Тема:** "Завершение разработки"

I. Уточнение повестки

II. Обзор выпусков
- Альфа-версия
- Бета-версия

III. Обзор результатов этапа

IV. Обсуждение и утверждение результатов этапа

V. Решение о продолжении выполнения проекта

VI. Вопросы и ответы

VII. Задания и ответственные

Чикаго • Детройт • Милуоки • Цинцинати - Индианаполис • Луисвилль

Пока все участники открывали свои папки, Дэн добавил:

— Как Джейн уже заметила, сегодня мы собрались по поводу окончания фазы «Разработка». Давайте для начала рассмотрим две промежуточных версии, которые мы выпустили.

Минут тридцать группа обсуждала альфа- и бета-версии RMS. С альфа-версией работали только участники проектной группы, а бета-версия прошла через руки небольшой группы пользователей, отобранных Джейн и Мэри-Лу. В принципе, все прошло неплохо, а в бета-выпуске было обнаружено несколько важных проблем, которые требовалось решить.

- Кстати, Дэн, то, что ты заставил Марту сделать с бета-версией, не очень-то честно, — с усмешкой сказал Билл, глядя на Дэна. Марта посмотрела на Билла, не понимая, говорит он серьезно или шутит.

- Что ты имеешь в виду, Билл? — невинно спросил Дэн, хотя совершенно точно знал, о чем речь. — Я не припоминаю за собой ничего такого.

- Так ты ничего и не делал, — ответил Билл и, глядя на Марту. добавил: — Все сделали она и Майкл, а ты им лишь давал советы.

- Видишь ли, Билл, это был единственный способ заставить тебя и Сэма понять, что Марта права, — сказал Дэн.

Он опасался, что Билл начнет ругаться, однако тот вдруг широко улыбнулся;

- Это правда, метод оказался очень эффективным. И для меня. и для Сэма.

- Не мог бы кто-нибудь из вас разъяснить остальным, что именно проделали Дэн и Марта?

Дэн и Марта попытались вставить слово, но Билл прервал их;

- Вот как было дело. В одном из тестов Марта и Майкл обнаружили, что время отклика много выше установленного предела; в частности, при сборе всех расписаний. Они попытались указать на это мне и Сэму, но мы, мягко говоря, отшили их, считая, что проблема решится сама собой после ввода в строй нового сервера. Они выждали, пока группа, работавшая над бета-версией, закончит Win32-клиент, а затем устроили ловушку.

— Ну зачем так, — вставил Дэн, — никакой ловушки не было. Мы просто хотели проверить бета-версию в условиях, максимально приближенных к «боевым».

- Насчет «боевых» условий — полная правда, особенно в том, что касается писем и звонков, которые я стал получать, — с усмешкой заметил Билл. — Когда мы установили бета-версию на машинах пользователей. Марта, Майкл и Тим заменили нашу базу данных SQL Server на свою, в которой хранилось 50 000 расписаний. Время отклика выросло до 13 секунд, и пользователи набросились на меня и Сэма.

— А как насчет нового сервера? Он не решил проблему? — спросила Джейн.

— Они и об этом позаботились, — ответил Билл. — Я пришел жаловаться, считая, что они по-прежнему пользуются старым сервером, на котором мы тестировали бета-версию. Оказалось, что Дэн договорился с парнем, который закупал оборудование, и тот поставил новый четырехпроцессорный сервер, раз в пять мощнее того, на который мы рассчитывали. Стало ясно, что проблема не в сервере/Сэму и мне пришлось признать, что причина большого времени отклика — наш код.

— Мне очень жаль, Билл, но я в самом деле не видела другого способа убедить вас, — сказала Марта с явным огорчением.

— Что ж, юная леди, вы сделали именно то, что было нужно. Иногда единственный способ убедить в чем-нибудь такого тупоголового старика, как я, — заманить его в яму, которую он сам выкопал. И никогда не извиняйтесь за то, что хорошо сделали свою работу, — он задумался, а потом сказал: — Все это напоминает мне одну историю, которая случилась, когда я служил во флоте. Я тогда помогал налаживать первые компьютеры для наводки орудий. Наш начальник корабельной артиллерии не верил компьютерам и всегда оспаривал числа, которые выдавал компьютер. Однажды, когда он в очередной раз пришел со своими претензиями, мы решили его проучить и спросили его, по каким данным наводить орудия — по компьютерным или по его. Он ответил — по моим. В результате мы чуть не потопили крейсер. Тут-то он наконец понял, что его упрямство может довести до катастрофы. Кстати, он не единственный морской волк, которому требовался такой урок.

Джейн улыбнулась и погладила Билла по руке:

- Во всяком случае, ты-то сделал выводы.

— Хватит воспоминаний, — заявил Тим. — Я хочу знать, что было дальше.

Билл рассмеялся:

- Когда пользователи вылили на нас с Сэмом целый ушат грязи, мы решили повнимательнее посмотреть на код. Оказалось, что Сэм второпях использовал очень неэффективный метод создания набора записей* . На небольшой таблице это не сказывалось, а когда таблица выросла до 10 000 записей, последствия оказались катастрофическими. Сэм изменил код, и время реакции стало ниже 5 секунд.

Билл пожал руку явно успокоившейся Марте и все зааплодировали.

— Я очень рада, что все так хорошо закончилось, — сказала она. -- Теперь и у меня к тебе есть вопрос.

- В какой-то момент мы с Майклом заметили, что Сэм работает гораздо быстрее, чем Бэт, но качество его кода хуже. Когда результаты какого-нибудь теста оказывались плохими, виноватым почти всегда оказывался код Сэма. Потом вдруг мы заметили, что положение начинает исправляться, пока, наконец, на прошлой неделе не случилось чудо: Бэт практически сравнялась с Сэмом по производительности, а код Сэма стал почти таким же чистым, как у Бэт. Что случилось?

Билл и Дэн переглянулись.

* По-видимому, имеется в виду тип курсора. Сэм. скорее всего, воспользовался динамическим курсором вместо статического, что стало заметно, когда таблица разрослась. — *Прим. ред.*

– Извините, это профессиональный секрет менеджеров, — ответил Дэн. — Именно за это нам с Биллом и платят большие деньги.

– Так нечестно, — воскликнул Тим, — Я тоже хочу, чтобы мне платили много баксов; мне, между прочим, до 2020 года выплачивать кредит за учебу в университете,

Билл наклонился к Тиму и шепотом сказал:

– Всего два слова, Тим: *рецензирование кода.*

– Рецензирование кода? — переспросила Мэри-Лу. — А мне казалось, что мы обсуждали внешнее рецензирование и решили, что у нас нет на это времени.

— Я же не сказал внешнее, — ответил Билл. — Это было внутреннее рецензирование, — усмехнулся он. — На самом деле идея принадлежит Дэну, пусть он и рассказывает.

– Все очень просто, — сказал Дэн. — Я предложил Биллу, чтобы Сэм и Бэт начинали каждый день с анализа кода, сделанного другим. Бэт увидела, что Сэм работает гораздо быстрее, а Сэм понял, что код Бэт гораздо чище. В результате Бэт стала работать быстрее, а Сэм стал писать чише — никто не хотел уступать, так ведь? Затраты времени оказались невелики, а выигрыш в качестве — очень заметным. Конечно, все получилось только потому, что наши разработчики имеют высокую квалификацию, а их начальник обладает талантом не только находить нужных людей, но и помогает повышать квалификацию своим сотрудникам и каким-то образом удерживает их в отделе.

Билл кивнул в знак благодарности.

– Итак, что же дальше? Можем мы считать, что разработка закончена?

Дэн повернулся к проектору и сказал:

— Давайте посмотрим, что мы должны были сделать и что сделали.

Некоторое время все отмечали в списке результатов сделанную работу. Наконец, Дэн сказал:

– OK, похоже мы разобрали все, что нужно. Как насчет задач, которые перед нами стояли. Пусть кто-нибудь перечислит их, чтобы мы решили, все ли задачи выполнены.

Джейн начала перечислять;

– Соглашение о функциональных возможностях; нам надо решить, все ли они реализованы, — она взяла со стола пересмотренные функциональные спецификации и зачитала список. — Давайте обсудим функциональные возможности, запланированные для первого выпуска приложения, и результаты их тестирования в бета-версии.

Когда со списком функций было покончено, Дэн сказал:

– Итак, мы все считаем, что текущая версия содержит все функциональные возможности. Что дальше, Джейн?

Джейн бросила взгляд на свои записи:

— Эталон для оценки эффективности работы пользователей.

— Мэри-Лу и Джейн, это ваша епархия. Все ли готово для поддержки пользователей?

Мэри-Лу кивнула:

— Сначала я волновалась, потому что пользовательский интерфейс все время изменялся. Но, после того как ты посоветовал мне участвовать во встречах группы разработки, я стала продвигаться вперед гораздо быстрее, и наконец мы вписались в график, хотя и с трудом.

— Хорошо, — сказал Дэн. — Я рад, что это сработало. — Он повернулся к Тиму: — Как дела у тебя? Ты и твоя группа готовы к развертыванию RMS?

— Да, у нас более чем удовлетворительный набор документации для группы сопровождения, поверьте мне. Кроме того, инцидент с сервером AutoCAD преподал нам урок относительно тестирования и производственных серверов. Мы поправили свои планы развертывания, учли некоторые нюансы сетевой инфраструктуры и подготовились к развертыванию. Я думаю, мы готовы к следующей фазе,

— Тогда, — сказала Джейн, — можно обсуждать последний пункт повестки: стабилизация, в том числе пилотное развертывание и дополнительное тестирование.

— Это к Марте и Тиму. — сказал Дэн. — Вы готовы к следующей стадии?

Марта кивнула:

— Когда мы только начинали работу, я бы в это не поверила, но сейчас я вполне уверена в нашей методике тестирования и в том, что мы правильно понимаем все вопросы. Мы уже поговорили с Тимом о плане работы и я думаю, что мы готовы.

— Превосходно, — сказал Дэн, закрывая блокнот. Он посмотрел на всех участников совещания и добавил: — Главный результат этого этапа — полностью законченный продукт, готовый к тестированию. Как мы уже отмечали, в нем возможны проблемы, но ничто не мешает нам передать продукт группе стабилизации. Все согласны?

Все закивали, и Билл сказал:

— Я думаю, что нынешняя версия оказалась очень удачной. Сэм, Бэт и я согласны, что ее можно отдавать группе стабилизации. На самом деле, — он залез в портфель и вынул из него компакт-диск, — вчера мы подготовили окончательную версию Win32-клиента для Марты и Тима. Это еще не окончательная версия, но нам она кажется очень удачной. — Он передал диск Марте и добавил: — Тестируйте на здоровье.

— Спасибо, Билл. Надеюсь, мы не найдем ничего серьезного.

– Если найдете, сразу приходите к нам. Мы не хотим потопить крейсер. — Билл повернулся к Дэну: — Помнишь, я хотел сказать несколько слов в конце заседания. Можно?

Дэн собрался кивнуть, но его перебила Джейн:

– Погодите. Мы еще не закончили и не можем закончить без заказчика.

– Это так, — сказала Мэри-Лу, — без Джима ничего не получится. А где он, кстати?

– Я забыл вам рассказать, — сказал Дэн, — Я встречался с ним вчера днем, и мы с ним обсудили все то, о чем говорили с вами сегодня. Он видел бета-версию и согласился, что она отвечает целям, поставленным перед первым выпуском. По его словам, он согласен, но окончательное решение за нами.

– Похоже, уровень доверия растет, — сказал Тим. — Приятная новость.

– На самом деле, — ответил Дэн, — Джим сказал даже больше: если Билл и я согласны, ему этого вполне достаточно.

– Теперь как раз время для моих нескольких слов, — заметил Билл. Он оглядел собравшихся и продолжил: — Вы все помните, что я с самого начала относился ко всей этой игре в MSF скептически. считая ее бессмысленной тратой времени, особенно, что касалось привлечения неспециалистов к управлению программным проектом. В общем, должен признать, что я ошибался.

В самом начале Дэн сказал, что его цель — сделать так, чтобы отдел разработки выглядел лучше всех. Не знаю, удалось ли это, но мне ясно, что мы не выглядим хуже всех. Фаза анализа помогла нам понять, что именно требуется пользователям. Фаза «Проектирование» позволила нам остаться реалистами и не обещать того, что мы не в силах сделать. А совместная работа на стадии разработки позволила избежать серьезных ошибок, которые вскрылись бы год спустя, когда базы данных достигли бы того критического размера, который так ловко выявили Марта и Майкл. Короче, с моей точки зрения, главное достоинство этого метода в том, что он удержал разработчиков от серьезных ошибок, и для меня достаточно одного этого.

После минутного молчания Дэн сказал:

– Билл, спасибо за этот комментарий. Я никогда всерьез не задумывался об этом, но, думаю, ты прав. Разработка по подобной методике позволяет застраховаться от серьезных ошибок. Тем не менее моя задача в другом — чтобы, когда разработка будет закончена, твои ребята оказались героями. Посмотрим, удастся ли нам этого добиться.

Откинувшись в кресле, Тим сказал:

– Эти душеспасительные беседы напомнили мне об эклерах — они такие же сладкие и тягучие. — Ам! — и он проглотил последний эклер.

Часть IV. Выпуск

ГЛАВА 13

Стабилизация продукта

В этой главе

Эта глава посвящена фазе «Стабилизация» модели MSF: проектная группа заканчивает разработку продукта и переходит к завершающим этапам его выпуска.

Последние этапы процесса разработки, связанные с выпуском готового продукта, можно охарактеризовать словами Чарльза Диккенса: «В такие времена испытываются души человеческие». В этой главе мы обсудим переход от этапа «Завершение разработки» фазы «Разработка» к этапу «Выпуск продукта» фазы «Стабилизация», в том числе четыре основных промежуточных этапа этого процесса: устранение проблем, синхронизация результатов, выпуск продукта и его исчерпывающее тестирование. Мы обсудим и промежуточные этапы, которые предстоит пройти проектной группе на пути к завершающему этапу «Выпуск продукта».

Кроме того, в этой главе вы познакомитесь с некоторыми рекомендациями по развертыванию продукта после его выпуска. Мы расскажем о предварительном планировании, пилотном выпуске и его тестировании, сопровождении и устранении проблем.

При работе над этой главой мы использовали свой собственный опыт проектирования архитектуры приложений и их реализации, а также следующие материалы:

- Microsoft Solutions Framework;
- Microsoft Windows 98 Resource Kit;
- технический документ Microsoft «Clustering and Windows NT Load Balancing Services (WLBS)»;
- учебный курс 827a «Administering Systems Management Server 2.0».

Изучив материал этой главы, вы сможете:

- ✓ перечислить роли участников проектной группы в **выпуске** окончательной версии продукта;
- ✓ перечислить промежуточные этапы процесса стабилизации приложения;
- ✓ описать последовательность действий по стабилизации приложения и выпуску продукта требуемого качества;
- ✓ перечислить результаты и документы этапа «Окончательный выпуск продукта»;
- ✓ описать методы развертывания программных продуктов;
- ✓ перечислить инструментальные средства, упрощающие процесс развертывания.

Процесс стабилизации

По завершении написания кода приложения и окончании этапа «Завершение разработки» фазы «Разработка» наступает фаза «Стабилизация», основная цель которой — подготовить выпуск продукта пользователям. В главе 12 мы уже говорили, что выпуск, подготовленный к этапу «Завершение разработки» фазы «Разработка», представляет собой полнофункциональную версию продукта, которая прошла базовое тестирование. Хотя на всех стадиях модели MSF группа старается соблюдать сроки и создавать продукт, отвечающий всем запросам заказчика, на стадии «Стабилизация» эти задачи становятся основными. Как показано на рис. 13.1, именно на этой стадии продукт «доводится» до состояния, необходимого для успешного выполнения этапа «Выпуск продукта».

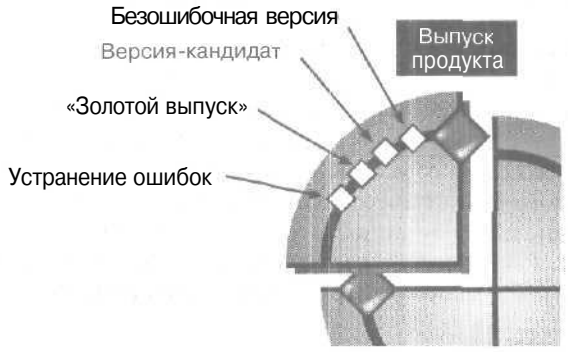


Рис. 13.1. Этап «Выпуск продукта» фазы «Стабилизация»

На стадии стабилизации проектная группа может выпустить несколько промежуточных версий приложения. Они позволяют выявить конкретные проблемы и устранить их. Почти окончательная версия продукта — так называемая *безошибочная версия* (Zero-Bug Release, ZBR). Это первый промежуточный выпуск, в котором все известные проблемы тем или иным способом устранены (зафиксированы, отложены или признаны несущественными). После полготовки этой версии группа проверяет степень готовности продукта; положительный результат этого теста позволяет считать продукт готовым.

Кроме того, на фазе «Стабилизация» завершается подготовка всех остальных компонентов, необходимых для развертывания и сопровождения продукта, включая инструкции для пользователей, руководство по установке и настройке и материалы для подготовки групп сопровождения и эксплуатации. Только при полной готовности кода приложения и всех этих материалов группа может считать свою работу законченной, а продукт — готовым к выпуску,

В процессе стабилизации, как и во всех других фазах разработки продукта, участвует вся проектная группа, но каждый из ее членов отвечает за свой участок. Последние недели и особенно дни перед выпуском продукта иногда требуют героических усилий и постоянного напряжения; группа должна быть готова приложить все силы для выпуска качественного продукта в срок.

В отличие от других стадий процесса разработки, эта не подразделяется на этапы. Их заменяют промежуточные выпуски продукта. Для каждого промежуточного выпуска выполняются одни и те же действия: устранение ошибок, синхронизация всех составляющих конечного продукта, выпуск и его тестирование. Посредством этого итерационного процесса группа создает версию продукта, которая отвечает всем требованиям. Эта версия и считается окончательной.

Распределение обязанностей в группе

Для достижения этапа «Выпуск продукта» необходимы усилия всей проектной группы, тем не менее каждый участник группы отвечает за выполнение конкретных обязанностей. Поскольку основное внимание на этой стадии уделяется созданию окончательной версии продукта, действия каждого из участников направлены именно на это.

В табл. 13.1 перечислены обязанности участников группы на стадии «Стабилизация». Руководитель каждой из подгрупп отвечает за их выполнение и связь с другими подгруппами проектной группы.

Табл. 13.1. Роли участников группы на стадии «Стабилизация»

Роль	Обязанности
Менеджер продукта	Согласование промежуточных версий продукта с заказчиком; планирование выпуска окончательной версии
Менеджер программы	Бета-выпуск и пилотный выпуск продукта, соблюдение графика, согласование функций окончательной версии с заказчиком и группами обучения пользователей и логистики
Разработчик	Поиск, регистрация и устранение ошибок; завершение интеграции компонентов продукта и его тестирование
Инструктор	Подготовка материалов, необходимых для сопровождения продукта и обучения пользователей; подбор инструкторов и координация их работы по обучению пользователей при выпуске промежуточных и окончательной версии
Тестер	Выполнение плана тестирования; поиск, регистрация и классификация ошибок; изъятие устраненных ошибок и верификация их устранения; отдельное тестирование удобства продукта для пользователей, средств установки и конфигурирования
Логистик	Установка, конфигурирование, развертывание и сопровождение промежуточных выпусков; планирование развертывания продукта; обучение группы эксплуатации и сопровождения продукта

Промежуточные этапы

В отличие от предыдущих фаз, естественным образом подразделяющихся на последовательные этапы, фаза «Стабилизация» состоит из промежуточных этапов, каждый из которых завершается выпуском промежуточной версии приложения. Промежуточные версии выпускаются до тех пор, пока группа не удостоверится, что продукт готов к окончательному выпуску. Отметим, что для каждой промежуточной версии проектная группа проверяет готовность и комплектацию. Каждая промежуточная версия тестируется, все обнаруженные проблемы регистрируются и устраняются. Решение об окончательном выпуске продукта принимается руководителями проектной группы, заказчиком и группами эксплуатации и сопровождения.

Этап 1: версии, появляющиеся по мере устранения ошибок

Как уже отмечалось в главе 12, проектная группа распространяет промежуточные выпуски продукта в ограниченной группе пользователей, что позволяет дополнительно протестировать приложение. На стадии «Стабилизация» основная цель — снижение числа ошибок в каждом следующем выпуске продукта. Ее достижение свидетельствует о стабилизации приложения, Хотя число обнаруженных ошибок не обязательно убывает монотонно от версии к версии, группа должна добиваться именно такой динамики.

Этап 2: безошибочная версия

Это первый промежуточный выпуск, в котором все известные проблемы устранены тем или иным способом (зафиксированы, отложены или признаны несущественными). Вполне возможно, что в следующем выпуске число ошибок возрастет, однако безошибочная версия позволяет группе разработки «держать» число проблем во всех следующих выпусках на приемлемом уровне. Выпуск безошибочной версии свидетельствует о том, что и окончательная версия не за горами.

Этап 3: версии-кандидаты

Когда по мнению проектной группы продукт готов к окончательному выпуску, создается *версия-кандидат* (Release Candidate, RC). В нее включены все составляющие продукта; кроме того, эта версия должна быть свободна от неустраненных проблем. Выпуски-кандидаты подвергаются интенсивному тестированию, чтобы выявить и устранить проблемы, препятствующие выпуску окончательной версии. Результаты такого тестирования показывают, можно ли считать версию-кандидат окончательной или группе следует выпустить следующую, Как правило, первая версия-кандидат не становится окончательной, поскольку в ней обнаруживаются ошибки, недопустимые в окончательной версии продукта.

Этап 4: выпуск окончательной версии

Окончательная версия продукта — это версия-кандидат, качество и состав которой устраивают всех участников проекта, в том числе, членов проектной группы и заказчика. Эта версия не требует ни дальнейшей разработки, ни дополнительного тестирования — именно ее «пакуют в коробку». Решение о выпуске окончательной версии никогда не бывает простым. Основная цель — выпуск продукта с заданными характеристиками в установленные сроки, поэтому прежде всего необходимо ответить на вопрос, отвечает ли версия-кандидат требованиям заказчика. Кроме того, надо принять во внимание резуль-

таты анализа проблем, результаты тестирования версии-кандидата и возможность ее сопровождения. Как и всякое ответственное решение, решение о придании кандидату статуса окончательной версии сопряжено со многими рисками и должно приниматься коллегиально.

Управление рисками

На первом этапе планирования развертывания продукта очень важно выявить потенциальные источники риска, оценить их влияние на развертывание продукта и выработать меры по снижению их влияния. Тщательный анализ рисков позволяет проектной группе избежать задержек и подготовиться ко всевозможным осложнениям,

Сводный документ оценки рисков надо пересматривать на всех стадиях разработки, поскольку с течением времени появляются новые факторы риска, а уже известные риски могут изменяться, В табл. 13.2 приведен пример матрицы рисков процесса развертывания продукта.

Табл. 13.2. Пример матрицы рисков процесса развертывания

Проблема	Влияние	Вероятность	Методы снижения
Недостаточная квалификация или малочисленность группы поддержки пользователей	Среднее	Высокая	Снижение темпов развертывания или увеличение числа сотрудников группы поддержки. Инструктаж группы поддержки по самым распространенным вопросам пользователей снизит затраты времени на обработку каждого обращения пользователей и, значит, уменьшит общую нагрузку на группу поддержки
Недостаточная квалификация или малочисленность группы сопровождения	Высокое	Средняя	Снижение темпов развертывания или увеличение числа сотрудников группы сопровождения. Инструктаж группы поддержки по самым распространенным проблемам снизит затраты времени на устранение типичных проблем и, значит, общую нагрузку на группу сопровождения

(продолжение)

Отсутствие у группы логистики необходимых полномочий на внесение изменений в процесс развертывания может нарушить график развертывания	Высокое	Средняя	Менеджер программы должен координировать работу группы разработки и группы логистики с тем, чтобы снизить вероятность возникновения проблем развертывания
Выход из строя сервера (серверов) распространения остановит развертывание	Среднее	Низкая	Необходимо заранее проверить работоспособность серверов распространения и предусмотреть возможность распространения приложения на компакт-диске
Невозможность подключения пользователей к серверам распространения остановит развертывание	Среднее	Низкая	Необходимо заранее проверить возможность подключения пользователей к серверам распространения и предусмотреть возможность распространения приложения на компакт-диске
Слишком высокая нагрузка на серверы распространения может вызвать резкий рост затрат времени на одну установку и нарушить график развертывания	Среднее	Низкая	Необходимо заранее проверить способность серверов распространения справиться с нагрузкой. Следует предусмотреть организацию резервного сервера распространения
Если пользователи столкнутся с проблемами при работе с приложением, меньшее число сотрудников будет им пользоваться	Низкое	Высокая	Чтобы облегчить пользователям знакомство с новым приложением, необходимо увеличить затраты на обучение или применить новые методы обучения

Этап «Выпуск продукта» и его результаты

Достижение этапа «Выпуск продукта» — главная задача проектной группы. Он знаменует завершение работы над продуктом и готовность всех его составляющих к развертыванию. Кроме того, на этом этапе происходит перераспределение ответственности за продукт — от группы разработки к группе логистики и сопровождения. С точки зрения проекта этап «Выпуск продукта» свидетельствует об успешной реализации концепции. Не забудьте отпраздновать этот момент — он означает, что цели проекта достигнуты.

По окончании стадии «Стабилизация» проектная группа начинает этап «Выпуск продукта». Его результаты ложатся в основу процесса развертывания продукта и его эксплуатации.

Для достижения этапа «Выпуск продукта» необходимы следующие результаты:

- **окончательная версия продукта** — исходные тексты и исполняемые модули;
- **документация к окончательной версии** — описание окончательной версии и последних изменений, не отраженных в документации продукта;
- **материалы для сопровождения приложения и поддержки пользователей** — окончательная версия учебных и инструктивных материалов;
- **результаты тестирования** — база данных выявленных проблем необходима как для сопровождения продукта, так и для будущих проектов;
- **архивы проекта** — вся информация, имеющая отношение к продукту и его разработке, независимо от того, вошла она в окончательный выпуск или нет;
- **документация** — вся документация проекта, включая ее версии для каждого промежуточного этапа.

Документация к окончательной версии

При выпуске окончательной версии в продукт всегда вносятся изменения, не отраженные в документации продукта. Чтобы «заткнуть эту брешь», создается документ, где вкратце описаны эти вопросы, а также вопросы совместимости и другие проблемы, которые могут возникнуть при развертывании продукта.

Материалы для сопровождения приложения и поддержки пользователей

Это материалы принимают различные формы — от справочных файлов до учебных брошюр. С их помощью решаются две задачи; подготовить пользователей к выпуску продукта и помочь им освоить про-

дукт. Если пользователи ничего не знают о продукте, например, как с ним работать и у кого это выяснить, крах проекта неминуем. Чтобы такая ситуация не возникла, группа обучения пользователей должна действовать эффективно; ниже мы вкратце опишем основные обязанности, возложенные на нее.

Документация для пользователей

Документация для пользователей — залог успешного развертывания продукта. Весьма полезно заранее объявить о скорой готовности продукта и предоставить активным пользователям возможность познакомиться с информацией о нем. Прежде чем делать подобное заявление, необходимо подготовить соответствующую документацию, организовать удобный доступ к ней и тщательно ее проверить. Документацию можно издать в обычном бумажном виде, однако намного эффективнее и удобнее организовать соответствующий раздел на узле корпоративной интрасети.

Кроме того, следует предусмотреть краткий документ, знакомящий пользователя с основными характеристиками и возможностями продукта. Когда проектная группа приобретет первый опыт развертывания продукта, необходимо опубликовать список самых частых вопросов пользователей с ответами на них. Желательно, чтобы этот документ отражал реальные ситуации, с которыми сталкиваются пользователи продукта, поэтому оптимальный путь — опросить пользователей и собрать их впечатления о продукте. Таким образом вы не только «оживите» документацию, но и вовлечете пользователей в процесс развертывания приложения, упростив тем самым работу группы развертывания и сопровождения.

Обучение пользователей и необходимые для этого материалы

На стадии стабилизации проектной группе следует разработать план обучения пользователей и соответствующую документацию. Формы обучения могут быть самыми разнообразными: самообучение, обучение под руководством инструктора или неформальные беседы «один на один»; конкретный метод определяется масштабом продукта и организации. Независимо от избранной формы, необходимо предусмотреть как первоначальное обучение пользователей на стадии развертывания (первое знакомство), так и дополнительное — после приобретения пользователями некоторого опыта работы с продуктом.

Сопровождающие материалы

Документация приложения может состоять из нескольких частей. Если в состав приложения входит СУБД и данные, пригодится серверная документация. Часто приходится структурировать документацию, описывая отдельно вопросы установки, конфигурирования и т. д. Например, при наличии в составе приложения базы данных надо обяза-

тельно описать меры по резервному копированию БД и ее восстановлению в случае сбоя. Не следует откладывать разработку плана восстановления и составление соответствующего документа до реального сбоя.

Если приложение основано на многоуровневой архитектуре, его эксплуатация в производственной среде часто позволяет или требует оптимизации производительности. Параметры, подлежащие оптимизации, необходимо предусмотреть на стадии разработки и протестировать на промежуточных выпусках продукта, а рекомендации по оптимизации и методику настройки параметров — описать в документации продукта. Не забудьте включить в документацию краткий перечень параметров оптимизации в табличном или матричном виде.

Для клиент-серверных приложений необходимо предусмотреть клиентскую документацию, где основное внимание уделено процессу установки и конфигурирования клиентского ПО. Здесь же стоит перечислить все параметры по умолчанию, используемые при установке. Этот документ создают на стадии разработки и приводят к окончательному виду на стадии стабилизации. Все изменения конфигурации по умолчанию обязательно надо отразить в клиентской документации. Кроме того, для многоуровневых приложений желательно в отдельном документе вкратце описать вопросы взаимодействия сервисов различных уровней,

Результаты тестирования

Результаты тестирования приложения следует собрать и сохранить в архиве. Они пригодятся как при расширении функциональных возможностей приложения, так и при сопровождении.

Архивы проекта

Все результаты выполнения проекта, включая результаты всех промежуточных этапов, должны сохраняться. Эти материалы понадобятся при анализе результатов выполнения проекта и при планировании будущих проектов.

Развертывание

Методы развертывания приложений весьма разнообразны. Для распространения программы установки (или всего выпуска) на клиентские рабочие станции можно воспользоваться такими средствами, как Microsoft Systems Management Server (SMS), регистрационные сценарии, рассылка электронной почтой и распространение через Web. Выбор метода распространения определяется размером приложения, корпоративной средой, темпами развертывания и ожидаемым уровнем подготовки пользователей. В этой главе мы подробно обсудим каждый из перечисленных методов развертывания.

Независимо от выбранного метода, его успех определяется тщательным планированием. Разработка плана развертывания и его тщательное тестирование до рассылки приложения пользователям обеспечит плавный переход продукта из стадии разработки на стадию эксплуатации и сопровождения, которыми завершается процесс создания продукта.

При развертывании приложения посредством рассылки по электронной почте или через Web квалификация пользователей должна быть выше, а времени понадобится больше. Некоторые методы более требовательны в отношении полномочий, необходимых пользователям для установки приложения. Этот вопрос не имеет значения, если пользователи обладают административными полномочиями на своих рабочих станциях, а организация не озабочена вопросами защиты,

Регистрационные сценарии — самый простой, быстрый и эффективный метод развертывания, однако здесь понадобится участие пользователей (им придется зарегистрироваться для запуска сценария). Кроме того, иногда оказывается, что у пользователей недостаточно полномочий для выполнения установки. При отсутствии жесткого контроля безопасности регистрационные сценарии должны работать успешно.

Использование SMS устраняет проблемы, присущие другим методам развертывания. **SMS** не требует участия пользователей в инициализации процесса установки, выполняя эту работу самостоятельно. Кроме того, SMS устраняет и проблему недостаточных полномочий, поскольку установка на клиентской рабочей станции выполняется от имени администратора. SMS упрощает развертывание, позволяя автоматизировать процесс дистрибуции и уведомления клиентов о предстоящем развертывании приложения.

Планирование развертывания

Развертывание любого программного продукта — это риск: вдруг новое приложение несовместимо с существующим программным обеспечением, конфигурацией операционной системы или драйверами аппаратуры. Планирование и тестирование процесса развертывания приложений — важные составляющие процесса разработки, в значительной степени определяющие успех проекта.

При планировании необходимо принять во внимание человеческие и технические ресурсы, необходимые для развертывания приложения, начиная от состава группы поддержки пользователей до сетевой инфраструктуры, требуемой для доставки исполняемых модулей приложения на клиентские рабочие станции. Цель планирования - определить ресурсы, необходимые для развертывания и, насколько это возможно, оценить объемы и сроки использования этих ресурсов.

На стадии тестирования эти параметры оцениваются в контролируемой среде, например, в компьютерной лаборатории или пилотной группе пользователей, работающей в реальной среде. По результатам тестирования иногда приходится пересматривать первоначальные опенки ресурсов, необходимых для развертывания. Помните: успех отдельных этапов развертывания еще не гарантирует популярность продукта в целом, Например, распространение установочной программы может пройти безупречно, но затем недостаток опыта у пользователей приводит к излишней нагрузке на группу сопровождения. Итак, планирование и тестирование — важные составляющие успешного развертывания, которыми не стоит пренебрегать.

Разнообразие производственной среды и географическая удаленность подразделений усложняют развертывание. Приходится учитывать особенности инфраструктуры организации, скорость каналов, нагрузку на серверы и сроки развертывания. Иногда необходимо также разделить подразделения и пользователей на категории по их географическому местонахождению, организационной структуре и сетевой топологии.

Местонахождение подразделений и пользователей

Подразделения и пользователей часто классифицируют по географическому положению, однако классификацию можно проводить и по серверу, отвечающему за проверку их учетных данных.

По географическому признаку проще всего планировать развертывание в том случае, когда у каждого подразделения свой контроллер домена и файл-сервер, работающий основным сервером распространения приложения для этого подразделения. Если же у подразделений нет своих серверов, при составлении плана развертывания необходимо учесть скорость каналов, соединяющих подразделения с серверами распространения.

Планируя развертывание, уделите особое внимание распределению пользователей по группам распространения в соответствии с их местонахождением. Кроме того, при развертывании продукта в организации с несколькими удаленными подразделениями следует учитывать некоторые дополнительные тонкости. Сначала можно ограничиться несколькими точками распространения, но при переходе к широкомасштабному развертыванию заранее, хотя бы за неделю, передайте код приложения на серверы распространения. Этот запас времени позволит вам убедиться, что его передача прошла успешно. и в случае необходимости повторить ее.

Подразделения

При развертывании приложения по подразделениям также не все просто. Компания, например, может потребовать развернуть приложение

на всех рабочих станциях подразделения, удаленных друг от друга на значительное расстояние. В такой ситуации требуется особо тщательное планирование. Клиентские рабочие станции надо сгруппировать по серверам распространения; в некоторых случаях стоит разбить группу клиентов на несколько групп распространения, В процессе развертывания позаботьтесь о равномерности поступления приложения на клиентские рабочие станции — все пользователи подразделения должны получить приложение практически одновременно. До начала распространения проектной группе следует убедиться в работоспособности всех серверов распространения и соединений с клиентами.

Скорость каналов

Планируя развертывание, чаще всего упускают из виду именно скорость каналов связи. А от нее зависит соблюдение графика развертывания приложения и его успех. Наибольшее влияние на развертывание оказывают два фактора:

- скорость соединений междусервером-источни ком кода развертываемого приложения и удаленными серверами распространения;
- скорость соединений между серверами распространения и клиентскими компьютерами.

Планируя развертывание кода приложения с сервера-источника на удаленные серверы распространения, необходимо выявить самый медленный канал. При грамотном планировании распространение кода можно провести в период наименьшей загрузки сети производственными операциями. Таким образом вы снизите риск перегрузки сети. Если скорость каналов слишком мала, можно запланировать распространение кода на выходные дни или на время, когда сеть не используется для производственных нужд. Если объем распространяемого пакета велик, а скорость соединений неудовлетворительна, следует рассмотреть альтернативный метод распространения кода, например, с помощью переносного жесткого диска, компакт-диска или другого емкого переносного носителя информации.

Нагрузка на серверы

Помимо скорости соединений, при планировании развертывания следует учесть нагрузку на серверы распространения. Ее надо отслеживать, чтобы гарантировать выполнение серверами обычных обязанностей (сервер БД, контроллер домена и т. п.).

По мере роста числа подключений к серверу распространения, скорость установки, как правило, падает. Выбор сервера с адекватным процессором, достаточным объемом памяти и высокой производительностью сетевого соединения позволяет снизить этот эффект. Кроме того, для снижения нагрузки на сервер можно ограничить число вероятных подключений к серверу в заданный интервал времени.

График развертывания

При развертывании масштабных приложений график развертывания планируется достаточно подробно, а затем уточняется по результатам тестирования. Сбор статистики развертывания и установки на первых стадиях развертывания играет очень важную роль в выработке реалистичного графика. Необходимо определить следующие параметры:

- скорость соединения с сервером распространения и между ним и клиентами;
- нагрузку на сервер;
- число установок на каждой фазе развертывания.

При сборе данных часто упускают из вида такой источник информации, как обращения в службу поддержки во время развертывания. Конечно, некоторые обращения в службу поддержки не следует учитывать (например, расспросы о графике развертывания или о порядке установки). Однако контроль числа обращений на разных стадиях развертывания помогает выявить ошибки в графике развертывания или возникновение непредвиденных факторов, замедляющих процесс. Небольшое число обращений — позитивный признак, свидетельствующий, что развертывание можно продолжать по графику, не снижая число установок в единицу времени или даже несколько увеличив его.

Преобразование данных

После установки приложения пользователям понадобится помощь в переносе или преобразовании данных предыдущей версии приложения или из других источников данных. Конкретные детали этой процедуры зависят от приложения, однако существует множество инструментальных средств, упрощающих перенос и преобразование данных.

Это процесс необходимо максимально автоматизировать и упростить. Мы рекомендуем три метода.

- **Совместимость «сверху — вниз»** — самый простой метод миграции данных — обеспечить совместимость нового приложения с прежним форматом данных. В этом случае пользователи смогут просто открывать существующие документы в новом приложении.
- **Общий формат** — этот метод требует сохранения существующих данных в формате, понятном и предыдущей, и новой версии приложения. Когда формат разработан, эта процедура не вызывает никаких затруднений. Преобразование файлов в общий формат можно возложить на пользователей, но лучше автоматизировать процесс и сразу преобразовать все файлы. Следует иметь в виду, что автоматическое преобразование может оказаться непростым делом, особенно если файлы рассредоточены на множестве клиентских рабочих станций.

- **Специализированное приложение** — его можно разработать специально для *преобразования существующих* данных в формат, понятный новому приложению, Такое приложение может помимо преобразования данных в новый формат выполнять их верификацию и удаление неверных данных. Если новое приложение работает с данными в реляционной БД или с хранилищем данных, полезно преобразовать данные средствами *сервисов преобразования данных* (Microsoft Data Transformation Services, DTS), Этот набор утилит из состава Microsoft SQL Server 7.0 позволяет переносить и преобразовывать данные из различных источников реляционной и другой природы (не ограничиваясь базами данных SQL Server 7,0).

В процессе развертывания документы, созданные с помощью новой версии приложения, могут понадобиться на компьютере, где новая версия еще не установлена. Самый простой способ избежать этого — выполнять развертывание по подразделениям.

Независимо от того, какой метод переноса данных вы выбрали, следует ознакомить пользователей со всеми деталями этой процедуры, в особенности если преобразование данных возлагается на них.

Развертывание промежуточных выпусков продукта

Для развертывания промежуточных выпусков необходимо, чтобы сам процесс установки был полностью отработан. Для этого программа установки должна корректно работать с разными версиями ОС, которые могут встретиться на клиентских компьютерах большой организации, и устанавливать соответствующие версии библиотек и инициализационных файлов, необходимых приложению. Обязательно проверьте комплектность установки файлов версии приложения, чтобы обеспечить корректную установку последовательных версий на один и тот же клиентский компьютер,

В процессе развертывания промежуточных версий приложения необходимо проконтролировать совместимость *приложения* с другими приложениями, установленными на клиентском компьютере. Это обычно весьма трудно в связи с разнообразием установленных приложений. Как правило, совместимость с каждым конкретным приложением требует отдельного тестирования, поэтому чаще всего приходится ограничиваться тестированием совместимости с наиболее распространенными приложениями.

Развертывание промежуточных выпусков продукта позволяет оценить нагрузку, которая ляжет на группы поддержки и сопровождения при развертывании окончательной версии. Это основной показатель при определении скорости развертывания и его сроков. Проектная группа должна предусмотреть время и ресурсы на обучение групп

поддержки и сопровождения. Обучение может проходить в виде обсуждения наиболее типичных ситуаций и проблем или ознакомлении с инструкциями сотрудников этих групп. Независимо от избранного метода подготовки следует помнить, что корректная оценка нагрузки на эти группы является обязательной для одобрения плана развертывания и успеха этой процедуры.

Предварительный инструктаж пользователей может значительно снизить число обращений в группу поддержки и упростить процедуру развертывания для пользователей. Обучение пользователей работе со справочной системой приложения позволяет отделить вопросы по установке приложения от вопросов, связанных с его использованием.

Методы развертывания

Существует масса методов развертывания приложений. В этом разделе мы обсудим несколько наиболее эффективных, в том числе Microsoft Systems Management Server (SMS), регистрационные сценарии, распространение по электронной почте и через Web.

Microsoft **Systems Management** Server

Применение Microsoft Systems Management Server — наиболее эффективный метод развертывания приложений, позволяющий администратору тщательно планировать и всесторонне контролировать этот процесс. Кроме того, SMS снимает всякую ответственность за распространение с пользователей, позволяя сделать это вообще без их участия.

При подготовке к развертыванию приложения средствами SMS необходимо учесть несколько факторов. Масштабы развертывания и размер приложения оказывают значительное влияние на систему и на то, как она справится с развертыванием. Грамотно разработанная процедура распространения средствами SMS распределяет серверы распространения в соответствии с местонахождением пользователей и скоростью каналов. Для разработки процедуры распространения средствами SMS необходимо распределить пользователей приложения по группам в зависимости от географического местонахождения, отдела или клиентской операционной системы. На рис. 13.2 проиллюстрировано распространение пакета приложения и инструкций по его установке от основного сервера-источника SMS к локальным серверам распространения и к точкам доступа клиентов. Клиентские компьютеры периодически опрашивают сервер доступа и подключаются к серверу распространения по заданному администратором расписанию для установки пакета.

В процессе тестирования процедуры распространения средствами SMS необходимо протоколировать время, необходимое для передачи кода приложения с сервера-источника на серверы распространения. При этом следует анализировать все возможные соединения, от са-

мых медленных до самых быстрых. Сбор этой информации на стадии тестирования позволит оценить время, которое понадобится на распространение кода приложения при полномасштабном развертывании. Для оценки нагрузки на серверы необходимо протестировать ситуацию с несколькими установками с одного сервера распространения. Эти данные позволяют оценить максимальное число установок, которые способен поддерживать сервер,

Рис. 13.2. Процесс развертывания программного обеспечения средствами SMS

Кроме того, в процессе тестирования следует учесть реакцию пользователей, участвующих в тестировании, на установку приложения. Эта информация поможет оценить нагрузку на группу поддержки при полномасштабном развертывании.

Регистрационные сценарии

Еще один метод развертывания приложений — сценарии регистрации. Это весьма эффективный метод распространения приложения среди множества пользователей, однако при его применении возникают вопросы, которые следует тщательно продумать:

- возможность многократной повторной установки приложения;
- требует ли приложение однократной установки на клиентский компьютер или установку придется повторять для каждого пользователя;
- как организовать развертывание для групп пользователей, не совпадающих с группами Windows **NT;**
- как при установке приложения на рабочую станцию, где работает несколько пользователей, зафиксировать параметры конкретного пользователя?

Развертывание приложений с помощью сценариев регистрации контролируется администратором, однако само развертывание требует участия пользователей (для запуска процедуры установки они должны по крайней мере зарегистрироваться на сервере). При создании сценариев необходимо принять во внимание распределение пользователей по группам, наличие разных версий клиентских операционных систем и другие особенности развертывания. Часть сценария, ответственная за собственно установку приложения, должна быть достаточно «интеллектуальной», чтобы определить, установлено ли приложение на клиентский компьютер или еще нет. Сценарий должен предусматривать создание ярлыков для пользователей и возможность многократной установки.

Тестирование развертывания посредством сценариев регистрации не представляет затруднений. Собрав тестовую группу пользователей, можно попросить их зарегистрироваться для начала установки. Процесс установки следует подвергнуть мониторингу для анализа вероятных проблем с последующей модификацией сценария (если понадобится). Необходимо также определить нагрузку на серверы в утренние часы, когда пользователи, как правило, регистрируются в системе.

Распространение по электронной почте

В некоторых случаях приемлем и еще один способ развертывания – распространение по электронной почте. Для этого метода (и не только для него) характерны следующие особенности:

- использование этого метода требует доработки установочного пакета;
- установочный пакет должен распознавать версию операционной системы клиентского компьютера;
- процесс развертывания должен предоставлять возможность подключения на выбор к одному из серверов распространения;
- от системы электронной почты требуется поддержка клиентов, подключающихся через сервер удаленного доступа и по медленным (например, телефонным) соединениям;
- этим методом допустимо распространять лишь сравнительно небольшие приложения, которые можно вложить в сообщение электронной почты.

Самый существенный недостаток развертывания средствами электронной почты заключается в том, что ответственность за запуск процедуры установки возлагается на пользователя. В принципе этот недостаток можно преодолеть, предусмотрев возможность автоматического запуска процедуры установки по получении письма (этот способ не позволит пользователю отказаться от установки, удалив письмо).

На стадии тестирования развертывания по электронной почте необходимо собрать информацию об успешности установки, о числе отказов и о времени, затраченном на установку. Как и развертывание с помощью регистрационных сценариев, этот метод приводит к резкому росту нагрузки на серверы распространения в утренние часы, когда основная масса пользователей получает первую почту.

Распространение через Web

Распространение через Web — еще один полезный метод развертывания приложений. Как и другие методы, он должен работать с несколькими серверами распространения. Этот метод также возлагает ответственность за запуск процедуры установки на пользователя, который должен регулярно проверять соответствующую Web-страницу. При достаточной квалификации пользователей этот метод является одним из самых эффективных. Залог успеха развертывания через Web — создание понятной всем пользователям и эффективной Web-страницы развертывания.

При тестировании развертывания через Web проектной группе следует изучить мнение пользователей разной квалификации относительно дизайна Web-страницы развертывания, простоты ее использования и обшей эффективности процедуры. Для учета компьютеров, где установка прошла успешно, необходим аудит.

Эксплуатация приложения

Для повышения доступности приложения пользователям часто необходимо применять средства распределения нагрузки и методы обеспечения отказоустойчивости.

Для этого можно привлечь *систему распределения нагрузки Windows NT* (Windows NT Load Balancing Service (WLBS), которая обеспечивает распределение нагрузки и повышает отказоустойчивость распределенных Web-приложений. WLBS обеспечивает равномерное распределение клиентских запросов на 32 серверах с поддержкой отказоустойчивости — если один из серверов выйдет из строя, его нагрузка автоматически переносится на другие серверы кластера. После устранения проблемы сервер можно снова ввести в состав кластера WLBS. Тот же метод применяется и для обеспечения отказоустойчивости базы данных за счет объединения серверов БД в кластер.

Тестирование системы распределения нагрузки и средств обеспечения отказоустойчивости следует проводить на этапе тестирования развертывания. Основная задача тестирования — оценка приемлемости производительности, обеспечиваемой этой системой.

Сосуществование данных и версий

Эти вопросы как правило возникают при переходе на другую операционную систему. Например, при переходе организации с Novell NetWare на Microsoft Windows NT процесс развертывания приложений должен учитывать, что некоторое время будут сосуществовать несколько операционных систем. Как правило, при этом приходится обеспечивать доступ к данным с обеих платформ. Если число сотрудников, работающих с данными, невелико, следует установить приложение на их компьютеры или в начале, или в самом конце. После этого не забудьте продублировать данные, чтобы обеспечить наличие идентичных копий на каждой из платформ. Если же с данными работает большое число пользователей, может понадобиться шлюз, обеспечивающий доступ к данным на старых системах с новых систем. По окончании перехода на новую ОС необходимость в хранении копий или в шлюзе отпадет.

Резюме

В этой главе мы рассказали о фазе «Стабилизация» и обсудили развертывание продукта по достижении этапа «Выпуск продукта».

Вы узнали о методах реализации фазы «Стабилизация», гарантирующих выпуск продукта с заданными свойствами в поставленные сроки. Рекомендуется последовательное прохождение четырех этапов: устранение ошибок, синхронизация готовности всех составляющих продукта, окончательный выпуск и полное тестирование. Каждый этап завершается созданием промежуточной версии продукта, каждая из которых приближает проектную группу к этапу «Выпуск продукта».

Кроме того, мы описали три фазы процесса развертывания продукта: планирование, тестирование и собственно развертывание, Адекватное планирование упрощает процесс развертывания и улучшает первое впечатление пользователей от приложения. Как мы уже не раз отмечали в этой книге, когда план составлен, его надо последовательно выполнять и уточнять. И наконец, реализация плана развертывания завершает процесс разработки — теперь продукт из рук разработчиков переходит к пользователям.

Закрепление материала

1. Какова основная цель стадии «Стабилизация»?
2. Как группа продвигается к выпуску продукта?
3. Перечислите основные этапы стадии «Стабилизация».
4. Перечислите промежуточные выпуски продукта.
5. Каковы результаты стадии «Стабилизация»?
6. Какие методы рекомендуется применять для развертывания приложений в организации?

ГЛАВА 14

Обсуждение проекта

В этой главе

Широко известно, что «дорога в ад вымощена благими намерениями». Несмотря на лучшие побудительные мотивы, многие компании не удосуживаются провести организованное обсуждение проекта по его окончании. Это одна из причин того, что во многих из этих фирм проекты разработки становятся практически неуправляемыми.

Эта глава посвящена методам организации обсуждения завершенных проектов. Мы покажем, что практика рецензирования завершенных проектов позволяет повысить качество управления этими проектами. Мы рассмотрим взаимосвязь обсуждения завершенных проектов с *моделью зрелости разработки программного обеспечения* (Capability Maturity Model for Software, CMM) и поясним, насколько велико значение рецензирования проектов для выработки оптимальных приемов работы.

Кроме того, в этой главе вы познакомитесь с практическими рекомендациями по проведению формального рецензирования, с критериями отбора участников этой процедуры и методами ее организации. Мы покажем, как организовывать и проводить соответствующие совещания, как собирать и документировать информацию. В завершение мы обсудим создание и роль специальной обзорной группы в больших проектах.

При работе над этой главой мы использовали свой собственный опыт проектирования и реализации архитектуры приложений, а также следующие материалы:

- Демпси (Dempsey), Дворак (Dvorak) и Михан (Meehan) «Escaping the IT Abyss» (The McKinsey Quarterly, № 4, 1997);
- МаркПоук (Mark C. Paulk), Чарльз Вебер (Charles V. Weber), Билл Кэртис (Bill Curtis) и Мери Бет Крисси (Mary Beth Chrissis) «The

Capability Maturity Model: Guidelines for Improving the Software Process» (Addison-Wesley, 1995);

- Джойс Штатц (Dr. Joyce Statz) «Microsoft Solutions Framework and the Capability Maturity Model» (Microsoft/TeraQuest, 1999).

Изучив материал этой главы, вы сможете:

- ✓ обосновать необходимость и перечислить выгоды рецензирования проекта;
- ✓ охарактеризовать связь обсуждения завершенных проектов с моделью зрелости практики разработки программного обеспечения;
- ✓ описать практические методы рецензирования проектов.

Зачем рецензировать проект

Начнем с цитаты из статьи «Как избежать хаоса в управлении информационными проектами» Демпси и соавторов.

« Обсуждение эффективности реализации проектов в области информационных технологий поможет лучше справиться со следующим проектом. Многие компании утверждают, что они проводят рецензирование законченных проектов, однако мало кто делает это достаточно эффективно. Обычно руководство компании ссылается на то, что «большинство сотрудников уже ушли или теперь занимаются другими проектами» — так сказал нам один директор по ИТ. Другой сообщил, что «цели проекта так часто менялись, что теперь уже трудно его рецензировать». Эти причины не помеха при тщательном планировании, наличии четко поставленных целей, аккуратной отчетности, четком соблюдении графика и постоянном контроле.

Компании — лидеры в сфере информационных технологий отличаются тем, что информационными проектами управляют бизнес-менеджеры, ориентированные на достижение результата и контролирующие проект на каждой стадии. Отчетность на каждом промежуточном этапе позволяет менеджеру сопоставлять задачи и результаты. Опыт и извлеченные уроки применяются в последующих проектах».

Итак, подготовиться к будущему можно только изучая прошлое. Приведем пример: кандидат на должность на собеседовании утверждает, что у него двадцатилетний опыт работы в какой-то области. Задайте ему вопрос: у него действительно двадцатилетний опыт или двадцать годичных опытов? Если не извлекать уроков из прошлых успехов и неудач, прогресс замедлится или вовсе прекратится.

Разбор проекта следует начинать с так называемого *вскрытия,* которое проводится сразу по окончании проекта. При разборе следует тщательно проанализировать ход выполнения проекта и выявить «плюсы» и «минусы» работы на всех стадиях. Разбор только что завершенного проекта — первый шаг к расширению арсенала наиболее эффективных методов работы, что позволит будущим проектным группам более точно выявлять потенциальные риски и справляться с ними.

Достоинства рецензирования

Рецензирование идет на пользу и компании, и отдельным сотрудникам. Как уже упоминалось, самый важный результат рецензирования — повышение квалификации сотрудников посредством расширения арсенала наиболее эффективных методов работы. Кроме того, рецензирование имеет и другие достоинства — они перечислены ниже.

- **Логическое завершение проекта** — формальное завершение проекта особенно важно в случае, когда группа, выполнявшая проект, сразу переключается на работу над новым проектом или расформировывается.
- **Возможность обмена мнениями** — обсуждая проект, его участники получают возможность «облегчить душу». Если не предоставить менеджерам и сотрудникам возможность высказаться организованно, это может случиться само собой, что чревато нежелательными последствиями. На встрече следует говорить о работе коллектива, а не о действиях отдельных сотрудников. Такая встреча *ни в коем случае* не должна стать сведением счетов или поиском виновных.
- **Улучшение морального климата** — сотрудники получают возможность поделиться как отрицательными, так и положительными мнениями по поводу работы над проектом. Разработка программного обеспечения традиционно считается скорее индивидуальной, чем коллективной деятельностью; в рамках подхода, изложенного в этой книге, укрепление коллективного духа — важнейшая задача,
- **Анализ методов работы** — для будущих проектов очень важен анализ слабых и сильных сторон проекта с точки зрения проектной группы. Такой анализ — основа выработки корпоративных стандартов и оптимальных методов работы. Помните, что модель процесса разработки MSF требует пристального внимания к методам — это позволяет повысить эффективность новых проектов за счет применения оптимальных методов. Результаты анализа методов выполнения предыдущих проектов очень полезны на первых стадиях разработки нового проекта. Следует позаботиться о сборе этих результатов, например, посредством организации соответствующей библиотеки, доступной всем разработчикам.

- **Сбор мнений и откликов** — результаты анализа методов работы станут основой для повышения эффективности выполнения следующих проектов. Такая обратная связь необходима при применении модели зрелости разработки программного обеспечения, к рассмотрению которой мы переходим.

Модель зрелости

Эту модель разработал д-р Джойс Штатц (компания TeraQuest) в рамках выполнявшихся с середины 80-х годов в Университете Карнеги-Меллон исследований эволюции компаний от «младенчества» к «зрелости». Эта модель применима к разным сферам бизнеса, поскольку у них много обших черт. Модель зрелости разработки программного обеспечения — адаптация модели Штатца к практике разработки ПО. Она применяется при разработке ПО, управлении проектами, приобретении ПО и разработке информационной инфраструктуры.

Модель предлагает структурированное представление соответствующей предметной области, обычно в виде иерархии из пяти уровней. Цель модели — постепенное повышение эффективности организации бизнес-процессов. Каждый из уровней модели суммирует оптимальные методы работы на данном этапе и образует основу для их качественного роста, необходимого для перехода на следующий уровень.

Уровни модели зрелости разработки программного обеспечения перечислены в табл. 14.1.

Табл. 14.1. Пять уровней модели зрелости разработки ПО

Уровень	Чему уделяется основное внимание	Основные характеристики
***5:* оптимизация**	Постоянному совершенствованию бизнес-процессов	Предотвращение дефектов Управление технологическим развитием Управление процессами
4: управляемость	Качеству продукции и процессов	Управление процессами на основе количественных показателей Управление качеством программного обеспечения
3: структуризация	Управлению проектами	Анализ требований Планирование хода выполнения проекта Отслеживание хода выполнения проекта и его предсказуемости

(продолжение)

2: повторяемость	Четкости инженерного процесса	Анализ требований Планирование хода выполнения проекта Отслеживание хода выполнения проекта и его предсказуемости Управление взаимодействием с субподрядчиками Обучение Рецензирование ПО
1: начальный	Поиску героев	Работа над проектом требует героических усилий

Подробное описание модели выходит далеко за рамки этой книги. Тем не менее отметим, что модель процесса разработки MSF — превосходное руководство для перехода организации от первоначальной стадии модели Штатца к следующим уровням зрелости. Модель зрелости разработки программного обеспечения подробно обсуждается в книге Марка Поука, Чарльза Вебсра. Билла Кэртиса и Мери Бет Крисси «Модель зрелости, или как повысить эффективность проектов разработки программного обеспечения» (Addison-Wesley, 1995).

Мы упомянули о модели Штатца, чтобы еще раз подчеркнуть важность обсуждения проектов для эволюции организации по ступеням зрелости. Для организаций, находящихся на первом уровне модели, типично «изобретение велосипеда» в каждом проекте. Рецензируя выполненные проекты и применяя полученный опыт в следующих проектах, вы избегнете этого широко распространенного недостатка.

Главные задачи организаций, уровень развития которых — вторая ступень модели Штатца, — точное планирование проектов и эффективное отслеживание хода их выполнения. Обсуждение проекта позволяет организациям сопоставить планы с их фактическим выполнением, а значит, повысить эффективность планирования и управления проектами. Игнорирование обсуждения скорее всего приведет к повторяющейся от этапа к этапу недооценке или переоценке ресурсов и графиков будущих проектов.

В своей статье «MSF и модель зрелости» (Microsoft/TeraQuest, 1999) д-р Штатц отмечает, что рецензирование завершенных проектов является обязательным условием перехода организации на третий уровень иерархии:

> *«Хотя большинству организации второго уровня присуща высокая дисциплина при выполнении проектов, опыт, приобретенный сотрудниками, часто остается неиспользованным, хотя он мог бы найти*

эффективное применение в следующих проектах. Накопление такого опыта на уровне организации и стандартизация процессов и подходов, применимых ко всем проектам, — краеугольный камень третьего уровня модели зрелости.

По мере выполнения проекта группа приобретает бесценный опыт. Его надо сделать достоянием организации и активно применять в следующих проектах. Организация должна постоянно выявлять оптимальные методы и способы, годные для широкого круга проектов».

На четвертом и пятом уровнях модели основная задача рецензирования — сбор мнений и откликов. Выявление каких-либо проблем завершенного проекта позволит изменить процессы так, чтобы снизить или полностью исключить влияние этих проблем на следующие проекты.

Модель зрелости разработки программного обеспечения становится теоретической основой для повышения эффективности процессов в рамках организации и, значит, для ее эволюции *к* зрелости. В рамках этой модели рецензирование завершенных проектов — важнейший инструмент повышения эффективности процесса разработки программного обеспечения.

Рекомендации по проведению встречи

Большинство обзорных встреч имеют много общего. Не претендуя на полноту, перечислим основные вопросы, которые следует учесть при планировании такой встречи:

- время;
- форма;
- продолжительность;
- организация;
- участники.

Время проведения встречи

Встречу, посвященную обсуждению проекта, по логике вещей надо проводить по завершении работы над проектом. С другой стороны, анализ сделанного и приобретение опыта важны на всех стадиях проекта, в особенности по окончании всех основных этапов процесса разработки, в том числе этапа «Завершение разработки» фазы «Разработка».

Хотя нет никаких строгих требований в отношении времени проведения формального рецензирования, не следует откладывать эту процедуру, но и не стоит затевать ее сразу по окончании проекта — в этом случае обсуждение скорее всего коснется последней фазы проекта. Кроме того, в этот момент воспоминания еще слишком свежи,

и эмоциональные переживания могут воспрепятствовать объективному анализу проекта. Если же встречу отложить надолго, члены проектной группы забудут специфику проекта. Более того, с течением времени вызвавшие наибольшие затруднения особенности проекта, как правило, теряют значимость в глазах его участников.

Некоторые рекомендации по выбору сроков проведения обзорной встречи после выпуска продукта в зависимости от различных характеристик проекта приведены в табл. 14.2.

Табл. 14.2. Выбор сроков проведения обзорной встречи

Характеристики проекта	Через 2 недели после выпуска	Через 5 недель после выпуска
Объем	Небольшой	Крупный
Продолжительность	Короткая (до 3 месяцев)	Значительная (более 3 месяцев)
Активность участников	Низкая	Высокая
Наличие членов проектной группы	Некоторые уже работают над другими проектами	Все уже работают над другими проектами

Форма проведения встречи

Форма проведения встречи определяется характеристиками проекта. Можно провести встречу небольшой группы участников проекта в неформальной обстановке или, предварительно обсудив проект с избранными участниками, устроить общее собрание по результатам анализа.

В идеале обзорная встреча — это хорошо продуманное и организованное мероприятие с четко поставленными целями и продуманной повесткой. Неформальные встречи полезны для укрепления коллективного духа проектной группы, однако они редко помогают определить оптимальные методы работы, ради чего, собственно, все и затевается.

Тщательное планирование обзорной встречи и подготовка участников позволит сосредоточиться на изучении опыта, не превращая обзорную встречу в бессмысленный обмен жалобами.

Продолжительность встречи

Обзор проекта часто организуют в форме нескольких мероприятий, длящихся целый день, однако лучше планировать одну встречу продолжительностью не более двух часов. На продолжительность встречи влияет и выбор времени для ее проведения — иногда требуется несколько встреч, посвященных отдельным вопросам работы над

проектом. Следует отметить, что в любом случае основную подготовку к рецензированию надо закончить до проведения встречи.

Организация встречи

Выбор походящей обстановки для обзорной встречи позитивно скажется на обсуждении и моральном климате в группе. Рекомендуется круглый стол или несколько столов, расставленных по кругу, что подчеркнет равноправие всех участников группы без доминирующей персоны «во главе». Помещение должно быть достаточно велико, чтобы вместить всех членов проектной группы. Если возможны конфликтные ситуации, лучше заранее выбрать помещение попросторнее - эмоциональные всплески в ограниченном объеме быстро охватывают всех участников встречи.

Обстановка должна способствовать созидательной ориентации участников встречи. Если люди будут чувствовать дискомфорт или угрозу, вся их энергия уйдет на самозащиту, а до непредвзятого анализа сильных и слабых сторон проекта просто «не дойдут руки». Основное внимание надо уделить результатам коллективной работы группы над проектом, а не оценке вклада отдельных участников.

Состав участников

В принципе в обзорной встрече должны участвовать все, кто, пусть даже немного, работал над проектом. Лучше всего, если позволят условия, собрать вместе абсолютно всех участников проекта, чтобы все они смогли высказаться.

Если же такое общее собрание невозможно — скажем, число участников слишком велико — попробуйте организовать серию мини-встреч сотрудников, работавших над разными частями проекта. Затем соберите представителей этих групп для итогового обсуждения.

Подготовка обсуждения

В повестку обзорной встречи следует включить следующие вопросы:

- удалось ли соблюсти график выполнения проекта;
- насколько эффективно использовались привлеченные ресурсы:
- какие сильные и слабые стороны работы на каждой стадии выполнения проекта могут отметить члены проектной группы;
- что не удалось;
- как, по мнению участников, улучшить процесс разработки;
- какие рекомендации проектная группа может дать тем, кто будет работать над следующими проектами.

Перечисленные выше вопросы не обязательно обсуждать в каком-то конкретном порядке, однако они составляют основу плана обсуж-

дения любого проекта. Менеджер программы должен спланировать процесс рецензирования до начала встречи, добавив к этим базовым вопросам все, что он считает нужным.

Всем участникам проекта надо предоставить возможность свободно изложить свою точку зрения о их роли, затратах времени на проект и др. Помните: организация встречи должна способствовать обсуждению ролей, а не индивидуумов, игравших эти роли, проекта в целом, а не его последних (самых горячих) дней.

Как правило, достаточно подготовить повестку дня и общие рекомендации по обсуждению за неделю до встречи и ознакомить с ними всех участников. Следует всячески приветствовать комментарии и замечания по повестке от участников встречи. Если таковые поступят, менеджер программы должен в обязательном порядке пересмотреть повестку и регламент.

В случае небольшого проекта или неформальной встречи повестку достаточно обсудить в начале заседания. Часто полезно сначала ознакомиться с обзорами предыдущих проектов — это поможет участниками встречи заранее понять, чего от них ждут.

Ведущий

Часто бывает полезно пригласить на роль ведущего обзорной встречи человека, не принимавшего участия в проекте. Если это неудобно иди невозможно. ведущего встречи должен выбрать менеджер программы,

Обязанности ведущего — поддерживать порядок, обеспечивать соблюдение повестки и не позволять участникам встречи нападать друг на друга. Кроме того, ведущий должен следить, чтобы участники обсудили все вопросы и чтобы все они в равной степени участвовали в дискуссии и укладывались в регламент,

Запись дискуссии

В идеале все, что говорится на встрече, надо каким-то образом фиксировать, причем лучше если участники встречи будут знать, что это делается. Большие листы бумаги на стенах, лекционная доска с мелом или компьютеры — все это в той или иной форме можно использовать для записи дискуссии.

Важна и психологическая сторона: внимание участников встречи будет сосредоточено на том, что говорится, а не том, кто это говорит. Кроме того, полезно вести протокол встречи — тогда любую новую мысль или тему, не отмеченную в повестке, можно просто записать на полях. Тем самым важные мелочи не потеряются, но и не отвлекут участников встречи от повестки.

И наконец, наличие протокола заставит участников высказывать свои мнения корректно, кратко и ясно. Небольшая, но важная деталь — ведение протокола лучше доверить человеку, не принимавшему участия в проекте: он будет фиксировать мнения и мысли, а не обдумывать их.

Группа, организующая обсуждение проекта

Планируя обсуждение проекта, менеджер программы может создать специальную группу, Этот метод особенно хорош в больших проектах, когда проектная группа на самом деле состоит из нескольких больших групп.

Существуют разные способы подбора обзорной группы. Можно создать дисциплинарные подгруппы для обсуждения разработки, тестирования и других отдельных составляющих проекта. С другой стороны, ничем не хуже и смешанные подгруппы, члены которой работали над разными этапами проекта.

Подготовка к встрече

После организации обзорной группы нужно дать ей возможность провести одну или несколько встреч до общего собрания. На этих предварительных встречах должны присутствовать руководители проектной группы и менеджеры, чтобы выработать график и сформулировать цели обзора проекта. На встрече обзорной группы следует составить список участников общей встречи.

Сбор информации

Процесс сбора информации к обзору проекта — это обмен мнениями (скажем, по электронной почте) всех участников проекта. Основное внимание надо уделить слабым и сильным сторонам разработки проекта и рекомендациям на будущее. Если нужно, в ходе подготовки к общей встрече обзорная группа может побеседовать с отдельными членами проектной группы.

Анализ

Одна из важнейших обязанностей обзорной группы — записать и проанализировать всю информацию, собранную при подготовке и в ходе обзорной встречи.

Выработка рекомендаций

Для выработки рекомендаций обзорная группа должна проанализировать важность поставленных вопросов с точки зрения достижения целей проекта и эффективности его выполнения. Рекомендации следует давать только по важнейшим вопросам.

Проблемы и вопросы, указывающие на «минусы» проекта, надо тщательно изучить. Если существуют методы, позволяющие устра-

нить или смягчить этот недостаток, обзорная группа должна привести их в качестве рекомендаций. Если готового решения проблемы нет, обзорная группа должна порекомендовать альтернативные методы решения задачи.

Представление и обсуждение рекомендаций

В конце работы обзорная группа должна просуммировать все, что удалось выяснить, и представить список рекомендаций для руководства и менеджеров проекта. Все дополнения, замечания и исправления должны быть учтены для включения в окончательный отчет по проекту. Рекомендации для будущих проектов должны быть обсуждены и одобрены всей проектной группой.

Результаты обсуждения

Как мы уже упоминали, документ, суммирующий результаты обзора проекта, крайне важен для групп, которые будут выполнять следующие проекты.

Прежде чем приступать к обсуждению проекта, следует назначить ответственного за подготовку такого документа. В документ следует включить распорядок встречи, список участников, перечень рекомендаций и все важное, что было отмечено лри обсуждении. Не следует сокращать или подвергать цензуре мнения участников обсуждения — чем точнее они зафиксированы в документе, тем лучше он отражает мнение членов группы о проекте.

Когда документ будет подготовлен, необходимо дать всем членам проектной группы возможность просмотреть его до публикации, чтобы они могли прокомментировать документ, дополнить его или порекомендовать что-то изменить.

И наконец, самое главное. Одна из самых больших ошибок — подготовить документ, суммирующий результаты обсуждения проекта, положить его в папку и навсегда забыть о нем. Если организация заинтересована в повышении эффективности разработки программного обеспечения и хочет добиться успеха, не следует совершать эту ошибку.

Приступая к новому проекту, полезно изучить рекомендации. сформулированные в результате обзора предыдущих проектов, — это превосходный способ не повторять ошибок. Точно так же по мере выполнения проекта не забывайте обращаться к обсуждению соответствующих стадий предыдущих проектов. Помните: использование опыта, накопленного вашими предшественниками, — не только дань уважения им, но и экономия собственных сил и времени.

Резюме

Группы, игнорирующие обсуждение завершенных проектов, не только «не добегают до финишной черты», но и лишаются ценного опыта и перспектив роста. При эффективной организации рецензирование становится мощным средством повышения квалификации разработчиков и методом повышения эффективности проектирования и разработки программных продуктов. В процессе рецензирования группа, обсуждая успехи и неудачи, вырабатывает наиболее эффективные приемы управления проектами, и поэтому обзор проекта — инвестиции в будущее, которые вернутся сторицей при реализации следующих проектов.

Закрепление материала

1. Перечислите выгоды от рецензирования завершенных проектов.
2. Опишите взаимосвязь обсуждения завершенных проектов с моделью зрелости разработки программного обеспечения.
3. Как на практике организовать обсуждение проекта?
4. Опишите способы рецензирования больших проектов.

Практикум 10, Выпуск приложения

Дэн осторожно заглянул в комнату, где работали Марта, Майкл и Тим:

— Сколько ошибок сегодня, Марта?

Марта бросила, не глядя на него:

— Одиннадцать.

— Пять главных без изменений? — Дэну не хотелось отрывать ее от работы, однако на этом этапе лучше задать пару вопросов по ходу дела, чем организовывать специальную встречу.

— Без изменений.

— Загляни ко мне перед уходом, идет?

— Идет, — ответила Марта, не отрываясь от экрана.

Дэн вышел из комнаты группы тестирования и отправился в свой офис. Другой менеджер на его месте наверное бы разволновался, однако в предыдущих проектах Дэн видел и не такое. «Эндшпиль — вот как это называется», — подумал он. В юности он играл в шахматы и прекрасно помнил возбуждение, охватывавшее его в конце партии. Помнил он и то, как концентрировал все силы для решающего удара. «Мы знаем, какие ходы нужно сделать, и постараемся сделать их. Но вот что неизвестно, так это не возникнет ли очередная проблема, которая потопит наш корабль».

Число ошибок начало уменьшаться три дня назад. К счастью, проект был невелик, и достичь этой стадии оказалось несложно, даже несмотря на подготовку пилотного выпуска. Разработчики быстро обнаружили около 20 проблем, от которых за день осталось 10. Затем частота выявления ошибок резко снизилась. За последние два дня особого прогресса в устранении приоритетных проблем не было, Никто не мог понять причины пяти ошибок, стоявших в списке первыми, и что самое неприятное, несколько ошибок явно были связаны с новым сервером, специально купленным для проекта.

Вернувшись в свой офис, Дэн подошел к плакату с подробным графиком проекта и глубоко задумался. Он рассчитывал, что безошибочная версия будет готова к четвергу, однако теперь ясно, что этой надежде не суждено сбыться. До сих пор резерв времени ни разу не понадобился; может быть, пришла пора? «Не торопись, — сказал Дэн самому себе. — Дождись вечера и послушай, что скажут Марта и разработчики».

Готовы ли мы к встрече с пользователями?

В этот момент в дверь офиса постучали, а затем в нее заглянула Джейн, из-за плеча которой высовывалась голова Мэри-Лу.

– Входите обе и садитесь, — сказал Дэн.

Дамы расселись вокруг небольшого стола в офисе Дэна.

– Что у вас на уме? — спросил он.

Первой заговорила Мэри-Лу:

— Я хочу показать тебе документацию и справочные файлы; может, у тебя найдутся замечания или предложения,

— А мне нужна сводка состояния проекта для Джима и для публикации в интрасети, — добавила Джейн, — но мне не хочется трогать Марту и Тима: они, по-моему, и так «в загоне».

— Это точно, — ответил Дэн. — Боюсь, Марта как раз сейчас теряет остатки веры в нашу группу разработки. Тестеры обнаружили несколько ошибок, проявляющихся на новом сервере, и никто не может понять, в чем дело. — Он посмотрел на Мэри-Лу. — Но это не значит, что все остальные не должны выполнять свою работу, не так ли? Давайте я посмотрю, что вы сделали.

Он заглянул в документы, которые принесла Мэри-Лу. «Здорово! Хорошо продумано, отлично организовано, удачный подбор снимков — на экране выглядит очень привлекательно. Отличная работа!»

Он знал, что Мэри-Лу с ее опытом подготовки документации справится на отлично.

Пока Дэн восхищался документацией, Джейн и Мэри-Лу сидели молча.

- Я думаю, что все в порядке, — наконец сказал Дэн. — Единственное, что меня заботит — это то, что вам придется подождать, пока мы доберемся до промежуточного этапа «Безошибочная версия». Только тогда мы сможем удостовериться, что после устранения ошибок нам не придется переделывать документацию.

— Хорошо, что ты это сказал. Я попрошу типографию подождать с печатью день-два.

Джейн заволновалась:

— Дэн, а это действительно так серьезно? Мне бы не хотелось говорить об этом Джиму. И, кроме того, наши пользователи не обрадуются, узнав о задержке.

- Тебе не придется никому ничего объяснять, — ответил Дэн. - Я люблю открытость, но в то же время считаю неразумным сообщать кому-либо о наших успехах и неудачах до завершения проекта. Подобные сведения могут создать у людей неверные представления о проекте.

С заказчиком и пользователями работали Джейн и Мэри-Лу, поэтому Дэн в мягкой форме постарался высказать им, как следует вести себя с теми и с другими.

— Сейчас никому ничего сообщать не нужно. Давайте дождемся, пока у нас появится определенность — либо мы справимся, и тогда скажем, что все готово, либо нам придется обсуждать изменение графика. Мы все-таки выпустим продукт без ошибок, даже если для этого придется отложить выпуск.

- Пока я просто рассказываю всем, что мы добиваемся максимальной производительности продукта, и до сих пор это объяснение всех устраивало, — сказала Мэри-Лу. — Конечно, если мы пересмотрим функциональные возможности или сильно затянем выпуск, придется объясняться.

Дэн кивнул в знак согласия.

— Что же касается Джима, — сказал он, глядя на Джейн, — я просто скажу ему, что есть еще пара ошибок, над которыми мы работаем, и мы будем держать его в курсе.

Дэн понимал, что с Джимом пора поговорить — в конце концов, нельзя держать в неведении одно из первых лиц в компании, особенно если это лицо было одновременно финансовым и исполнительным директором.

— Договорились, — сказала Джейн. Выходя из офиса Дэна, она добавила: — Ах да, еще вот что. Пока идет тестирование, мы редко встречаемся, поэтому я плохо информирована. Что еще осталось сделать?

Дэн показал на один из плакатов на стене:

— В принципе, когда мы добьемся безошибочности текущей версии, мы подготовим окончательный внутренний выпуск, чтобы проверить работу программы установки. Кроме того, на этой стадии мы проведем окончательное тестирование. Затем выпустим версию-кандидат, установим ее на компьютерах пользователей и протестируем с помощью сценариев, разработанных Мартой и Майклом. Если не обнаружатся новые ошибки, если не восстанут из пепла старые и если мы справимся со всеми рисками, организуем встречу, где решим, готовы мы к выпуску продукта или нет. Если решение будет положительным, мы объявим, что окончательная версия готова, а Тим и его группа проведут развертывание.

— Физическое развертывание, — поправила его Мэри-Лу. — Останутся еще бизнес-вопросы и логистика.

— Верно, спасибо за напоминание, — улыбнулся Дэн.

— По-моему, в плане проекта предусмотрен еще один этап: обсуждение проекта, — вмешалась Джейн.

Дэн кивнул, но прежде чем он успел открыть рот, чтобы пояснить Джейн смысл последнего этапа проекта, в офис вошли Марта и Тим. Оба выглядели подавленными, а Марта была к тому же явно перевозбуждена.

— Пошли-ка отсюда, — шепнула Джейн, взяв Мэри-Лу за руку, и они быстро скрылись за дверью.

Свежий взгляд

Когда Марта, тянувшая за собой Тима, как буксир баржу, набросилась на Дэна, он попытался изобразить полное спокойствие и дружелюбие:

— Привет, ребята! Давно не виделись.

Марта проигнорировала и тон, и приветствие.

— Дэн, у нас проблема! Случилось именно то, чего я больше всего боялась. Код не работает, и никто не может помочь — ни Тим, ни Билл, ни Сэм, ни Бэт, ни даже Майкл. А уж я и подавно не знаю, что делать! Мне не нужна такая ответственность! Я не могу уследить, чтобы все делалось правильно, — она была в ярости. — О чем я думала! Я знала, что не надо соглашаться.

На Тима было жалко смотреть. Он целиком присоединился к мнению Марты.

— Дэн, я не понимаю, что происходит. Версия прекрасно работала в тестовой лаборатории, но когда мы перенесли ее на многопроцессорный сервер, время отклика упало. Потом пошли сообщения об ошибках. А сегодня утром мы дождались и синего экрана, — он вздохнул. — Единственное, что приходит в голову — это проблема с поддержкой многопроцессорности. Боюсь, придется отправить сервер

обратно, перенести приложение на однопроцессорную систему и надеяться, что нам хватит ее производительности.

Дэн понял, что пора брать ситуацию в свои руки.

— Прежде всего давайте не будем посвящать всю компанию в свои проблемы, Идемте в мой офис, вы успокоитесь, и тогда мы поговорим, — он взял Тима и Марту под руки, затащил в свой офис, плотно закрыл дверь и усадил своих коллег за стол. Дэн ясно видел, что Тим и Марта попали в одну из типичных ловушек фазы «Стабилизация».

— Я понимаю твое состояние, Марта. Поверь мне, любой, кто написал хоть строчку кода, попадал в подобную ситуацию. Решать задачи типа этой, не имея возможности понять, что происходит, — тяжелая работа. Ты не можешь решить, кто виноват: аппаратура, операционная система, какой-нибудь кривой драйвер или программа, которую мы написали. Однако чем больше ты злишься, тем меньше у тебя шансов справиться с проблемой, — он повернулся к менеджеру сетевого отдела: — А ты, Тим, нарушаешь сразу два правила: во-первых, делаешь слишком много предположений и, во-вторых, хватаешься за соломинку. Я ожидал от тебя большего.

Пораженческие настроения Тима мгновенно сменились раздражением:

— А что я еще мог подумать?

— Я не знаю, — ответил Дэн, — потому что у меня нет вашего опыта отладки этого конкретного приложения. — Тим стал успокаиваться, а Дэн продолжал: — У меня просто больше опыта с подобными ситуациями. Давай подумаем: предположим, ты консультант, которого пригласили в некую компанию, чтобы выяснить, в чем дело, и выработать меры по устранению этой проблемы. С чего бы ты начал?

Тим на минуту задумался и начал загибать пальцы:

— Сначала я бы выяснил, все ли компоненты используемой аппаратуры указаны в списке совместимого оборудования, и поискал бы отличия конфигурации от стандартной и другие странности. Потом я бы проверил работу аппаратуры.

- Хорошо, — ободрил его Дэн, — это верное начало. Что дальше? Давай начнем с описания проблемы.

- Можно я попробую? — сказала Марта. Ее уныние сменилось заинтересованностью, она любила решать задачки. — У нас есть специализированное приложение, взаимодействующее с несколькими сервисами, — скажем, MTS и SQL — на сервере под управлением Windows NT Server. Приложение отлично работает на одном сервере, но перестает работать на другом. Второй сервер — многопроцессорный, что может иметь отношение к делу, а может и не иметь.

— Можно я скажу? — спросил Тим. Марта кивнула. — Когда мы пытались устранить проблему, сервер стал работать все нестабильнее, пока наконец сегодня утром система не «упала».

— Итак, — спросил Дэн, — что можно предпринять?

— Для начала попробуем исключить многопроцессорность. Можно перенести приложение на другой многопроцессорный сервер и посмотреть, будет ли оно работать, — предложила Марта.

— Для начала неплохо, но нереально: новый сервер появится не раньше, чем через месяц, а такой пробоины наш график не выдержит, — напомнил Дэн. Он повернулся к Тиму: — Ты сказал, что сервер работал все менее стабильно. Это тебе ни о чем не говорит?

— Да нет, все наши многопроцессорные серверы работают очень стабильно.

— Они чем-нибудь отличаются от этого? — спросил Дэн. — Версией операционной системы, сервисными пакетами, заплатками, может чем-нибудь еще в этом духе?

— Я проверил все заплатки — все в порядке, — сказал Тим, — правда, я не проверял версию. Ее установил мой сотрудник, когда сервер поступил к нам. — Тим задумался. — Версия... А что если... Дэн, я воспользуюсь твоим компьютером?

— Конечно.

Тим бросился к компьютеру Дэна,

— Я зарегистрируюсь по своей учетной записи, ладно? — Дэн кивнул, а Тим, зарегистрировавшись, запустил несколько приложений.

Дэн и Марта наблюдали, как Тим раскладывает на экране несколько окон. Он посмотрел список файлов и проверил их свойства. Через минуту-другую он щелкнул пальцами:

— Вот оно! Дэн, вот в чем дело!

— В чем? — изумленно спросила Марта.

Дэн уже понял, что скажет Тим. но решил не перебивать его:

— Давай, Тим, не заставляй нас ждать.

— Смотрите, — сказал Тим, показывая на одно из окон, — это папка на производственном сервере MTS, А это та же папка на новом сервере. Видите разницу?

— В папках разные файлы, — сказала Марта.

— Это нормально, — ответил Тим. — Если серверы не абсолютно идентичны, состав папок и должен различаться. Дело не в этом, однако так этого не увидеть. — Он щелкнул в каждом из окон кнопку для отображения списка файлов со свойствами: — А теперь видите?

— У некоторых файлов отличаются даты, — сказала Марта. — Это важно?

- Иногда очень важно, — сказал Дэн, кивая Тиму. — Это значит, что на серверах установлены разные версии приложения. Чаще всего это не имеет значения, однако, если одно приложение устанавливает предыдущую версию файла, который использует другое приложение, та может начать работать некорректно.

— С серверами это часто случается — все эти заплатки, новое ПО и все такое, — Тим покачал головой, — Я сказал своим ребятам, что сервер нужно привести в рабочее состояние и установить все необходимые приложения и сервисы, но я не пояснил, в каком порядке это делать, какие файлы сохранить, а какие перезаписать. Я предполагал, что они сами справятся.

- Опять «предполагал», — улыбнулась Марта, глядя на Тима. - Теперь ты знаешь, чем кончаются твои предположения.

- Хорошо-хорошо, я все понял, — ответил ей Тим, поднимая руки.

- Ну и что мы будем делать дальше? — спросила Марта.

- Закажем пончики, — ответил Дэн.

Марта удивилась. Тим криво улыбнулся и пояснил:

— Он хочет сказать, что знает одного начальника сетевого отдела, который сегодня ночью будет учиться ничего не принимать на веру: ему придется переустановить все программное обеспечение на одном большом сервере.

Отклики пользователей

После того как Тим переустановил все программное обеспечение на сервере, работа над фазой «Стабилизация» пошла гораздо быстрее. Остальные ошибки в пилотном выпуске были устранены за несколько дней. Группа логистики под руководством Тима провела последний внутренний выпуск и вместе с Мартой — его тестирование.

Убедившись, что приложение готово, в выходные проектная группа провела тестовое развертывание в Чикагском офисе. Во время развертывания они дополнительно протестировали Win32-клиент, а сотрудники ИТ-отдела подключались к серверу через Интернет и подавали тестовые расписания.

Тестовое развертывание закончилось успешно. Новых ошибок не обнаружилось, а работа программы установки не вызвала нареканий. Группа была обрадована успехом, но с празднованием решили подождать до понедельника — к этому сроку Мэри-Лу должна была провести первые две лекции для сотрудников компании.

В понедельник проектная группа вместе с Сэмом и Бэт собралась в офисе Дэна. Они собирались все вместе пойти на ленч, как и по окончании фазы «Планирование», однако решили подождать Мэри-

Лу и Джима. Те должны были вернуться со второй лекции (по работе с Win32-клиентом для менеджеров). Чуть раньше они уже провели первую лекцию, посвященную работе с расписаниями с помощью Web-интерфейса.

Дэн сидел за столом и вертел в руках ручку. Сэм, Бэт, Тим и Марта играли в карты за маленьким столом, а Джейн читала, устроившись в уголке. Бил ходил по комнате взад-вперед, сложив руки на груди.

-- Билл, может перестанешь ходить? — попросила Джейн. — Ты ведешь себя как папаша в роддоме.

-— Хорошо хоть подарки не пришлось приносить, — сказала Бэт. Все рассмеялись, и даже Билл улыбнулся

-- Не волнуйся так, Билл, — сказал Дэн. — От этого приложения зависит не твоя карьера.

-- Этот так, — сказал Билл, — однако, после того как предыдущий проект был остановлен, этот должен удаться на «все сто».

Дэн выглянул из офиса.

-— Похоже, сейчас мы все узнаем. Они идут.

Джим и Мэри-Лу вошли в офис, и Мэри-Лу поставила в шкаф папку с учебными материалами. Когда она повернулась, все увидели, что у нее довольно мрачный вид. «Дело плохо, — подумал Дэн, — она обычно очень оживлена, особенно после лекций».

-- Ну, — не удержался Билл, который, похоже, был нетерпеливее всех, — как все прошло?

-- Пошли обедать, — вздохнул Джим, — после еды плохие новости принимать легче.

— Думаю, нам лучше сначала узнать о реакции менеджеров, Джим, даже если новости не слишком хорошие. — сказал Дэн, и все согласились.

Джейн подошла к своей подруге и спросила:

— Мэри-Лу, не подслащивай пилюлю, просто ответь: что они сказали?

Мэри-Лу взглянула на Джима, который, вздохнув, кивнул головой. Она вдруг вскинула руки и воскликнула:

— Они просто влюбились в вашу программу! — Она бросилась обнимать Джейн, а Джим расхохотался, увидев растерянные лица остальных.

— Ах вы мерзавцы! — сказал Дэн Джиму, пока все обнимались. — Боюсь, твой счет за ленч окажется несколько больше, чем ты думаешь, — пошутил Дэн, — тебе придется расплатиться за вашу шутку.

— Без проблем, — рассмеялся Джим. — Я с удовольствием заплачу за любой ленч, чтобы еще раз услышать комментарии, которые слышал сегодня от менеджеров.

— Слушайте, а почему такой восторг? — спросил Тим. — Я понимаю, что мы сделали интересное приложение, но это, в конце концов, не новая версия Windows.

- Восторг вызвала не программа, хотя она им очень понравилась, — сказал Джим. — Они сразу поняли, насколько проще им будет работать, но и это не главное, Самое большое впечатление произвело то, *как* мы это сделали. Проект закончен в срок, в соответствии со спецификацией и, самое главное, с учетом их пожеланий. Им понравилось, что их постоянно информировали, понравилось, как мы провели развертывание и как подошли к их обучению.

- Тем не менее больше всего им понравилось то, что *мы сдержали слово.* Практически каждый выступавший снова и снова говорил, что мы вернули им веру в то, что ИТ-отдел что-то способен сделать, причем в срок. — Он повернулся к Биллу: — Больше всего говорили о тебе, Билл. Они знают, что код писали твои ребята, и все как один утверждали, что были уверены в вас.

- Но это нечестно! — смиренно сказал Билл. Он обвел руками комнату: — Без них — без нас всех — ничего бы не вышло.

— Не трудись объяснять нам это, Билл, — сказал Дэн, хлопая Билла по плечу. — Мы собрали отличную группу, воспользовались удачной методикой и сделали хороший продукт. Если лавры в основном достанутся тебе и разработчикам, это будет правильно. — Он хитро посмотрел на Билла. — Можешь отдать нам долг — иди работай, а мы пойдем обедать за себя, и за тебя. Лады?

- Ни за что, — улыбаясь, ответил Билл. — Я устал от пиццы и кока-колы. Пришло время для других блюд.

По дороге на обед Мэри-Лу сказала Тиму:

- Кстати, о еде. Мне очень понравился значок, который вы придумали для программы. Менеджеры, по-моему, не поняли, а мне он показался очень симпатичным.

- Значок? — изумился Билл. — Последний раз, когда я его видел, там было что-то вроде расписания.

— О нет, — улыбнулась Марта, — перед установкой на компьютеры менеджеров все изменилось.

- И что же там теперь? — спросил Дэн.

- Пончик.

- Пончик?! — взревел Билл. — Откуда он взялся? Это ты придумал? — он показал пальцем на Тима, который спрятался за Сэмом. — Я тебе покажу пончик!

Тим побежал к лифту, а Билл преследовал его по пятам.

- Займите нам место в ресторане, — крикнула им вслед Джейн.

Обсуждение проекта

Три недели спустя группа снова собралась в офисе Дэна на обзорную встречу. Выпуск закончился удачно, и приложение уже вовсю использовалось в компании. Преимущества, которые принесло приложение, были очевидны: все менеджеры отмечали удобство и эффективность управления ресурсами и обработки расписаний.

Собрание открыл Билл:

– Эта встреча очень характерна для нашего проекта. Мы начали его не так, как все предыдущие, и заканчиваем его нетрадиционно. Поэтому вопрос: зачем вообще нужно это обсуждение? — он наклонился к Джейн и произнес театральным шепотом: — Смотри, у Дэна целый проектор причин.

– Конечно же, — ответил Дэн, устраивая слайд на проекторе, – кое-кто из нас уже умеет пользоваться PowerPoint.

Билл скорчил гримасу:

— Ох! Ладно, я помолчу.

– Вот этого-то ты и не должен делать, — сказал Дэн, включая проектор. — Вот причина, по которой мы собрались, и по которой я не хочу, чтобы вы молчали.

Он прочел надпись на слайде: «Обсуждение проекта».

– Это просто формализация всем известного процесса обучения на собственных ошибках. Если мы забудем этот проект и начнем следующий, не подумав, что мы сделали хорошо, а что плохо, мы упустим возможность сделать выводы и научиться работать лучше. Окончание проекта — это момент, который преподаватели называют «шансом научиться», и мы хотим им воспользоваться,

– Мне, например, кажется, что лучше будет ввести в практику шестиэтапный процесс вместо обычного четырехэтапного, — сказал Тим. — Я даже приготовил слайд с описанием этих стадий. — Тим положил слайд в проектор и прочитал названия стадий: — Первая – энтузиазм, вторая — разочарование, третья — паника, четвертая – поиск виноватых, пятая — наказание невиновных, шестая и последняя — награждение непричастных, — под общий хохот Тим выключил проектор и убрал слайд.

— Это ужасно, Тим, — смеясь, сказал Дэн, пока Тим усаживался на место. — Откуда ты это вытащил?

– Это с плаката, который висит у нас дома с незапамятных времен. Я был у родителей на прошлой неделе, увидел плакат и решил, что он отлично подойдет для обзорной встречи. Теперь он висит у меня в офисе.

— Рядом со всей твоей коллекцией плакатов? — *с* усмешкой спросила Джейн. — Знаешь. Дэн, а это отличное начало обсуждения.

Тим изобразил удивление:

– Ты что, думаешь, что моя шутка действительно имеет отношение к обзорной встрече?

— Да нет, правда, — настаивала Джейн. — Эти шесть стадий так рассмешили нас потому, что каждый из нас когда-нибудь принимал участие в проекте, закончившемся именно так. Мы же, пользуясь современной методикой, сделали проект RMS совсем иначе и совсем с другими результатами.

– Хорошо сказано, Джейн, — заметил Дэн. Затем он обратился к остальным: — Давайте побеседуем о том, какие фазы проекта оказались особенно ценными.

Обсуждение некоторых вопросов протекало спокойно, другие вызвали оживление, иногда все говорили одновременно, иногда — напротив, внимательно слушали выступавшего. Через три часа, исписав десять страниц мелким почерком, Дэн положил ручку и сказал:

– Все, ребята. Мне кажется, мы выжали из этого проекта все сколько-нибудь существенное. — Скорчив гримасу и сжимая-разжимая кисть, он добавил: — Во всяком случае, такого мнения придерживается моя рука — никаких сил в ней уже не осталось.

– Надеюсь, кое-какие силы у нее еще остались, — хрипло сказал Билл, вставая со своего места, — они тебе еще понадобятся.

Дэн медленно встал и оказался лицом к лицу с Биллом. Все стихли.

– Когда мы начинали проект, ты сказал, что хочешь сделать его так, чтобы в конце мы оказались героями. Сейчас я хочу сказать так, чтобы все слышали — настоящим героем оказался ты. Работать с человеком, который способен, собрав совершенно разных людей, заставить их использовать все свои знания и навыки на пользу проекту и сделать такой проект, как наш — честь для меня, — и он крепко пожал Дэну руку.

— Спасибо, Билл. Такой комплимент от человека с твоим опытом дорого стоит, — Дэн посмотрел на всех остальных: — Спасибо вам всем. С вами было приятно работать.

– Не надо говорить так, как будто мы больше никогда не встретимся, — сказала Джейн, собирая веши. — И не забывайте, вы обещали моим сотрудникам вторую версию с бухгалтерским пакетом.

Пока сотрудники неторопливо покидали офис, Дэн сделал в блокноте пометку о том, что нужно отправить Джиму Стюарту заявку на финансирование следующего этапа разработки RMS.

Фергюсон и Барделл

Строительство • Проектирование • Управлениепроектами

Проект создания системы управления ресурсами

Повестка дня

Дата: 7 июня 1999 г. _ **Тема:** обзор проекта

I. Уточнение повестки

II. Цели обзора проекта

III. График проекта

IV. Использование ресурсов

V. Плюсы и минусы на разных стадиях проекта
- Выработка концепции
- Проектирование
- Разработка
- Стабилизация

VI. Что можно улучшить

VII. Рекомендации для будущих проектов

Чикаго • Детройт • Милуоки • Цинцинати • Индианаполис • Луисвилль

Приложение

Вопросы и ответы

Глава 1. Производственная архитектура

Закрепление материала

1. Какие проблемы развития информационной инфраструктуры решает производственная архитектура?
   - **Активное участие информационной инфраструктуры в бизнесе** - информационная инфраструктура организации направлена на решение задач бизнеса.
   - **Контроль расходов** — позволяет получать отдачу от всех инвестиций в информационную инфраструктуру и принимать обоснованные решения о будущих инвестициях.
   - **Активность и реактивность** — производственная архитектура расширяет возможности организации по повышению эффективности производственных процессов и взаимодействию с клиентами, поставщиками и партнерами.
2. Каковы основные задачи производственной архитектуры?
   - Логичность.
   - Поддержка как каждодневной деятельности, так и автономных проектов.
   - Преобразование сложившейся структуры информационных систем и приложений организации к состоянию, определенному как долгосрочная цель.
   - Поддержка как текущих, так и перспективных задач.
3. Перечислите фазы применения производственной архитектуры в проектах разработки программного обеспечения.
   - Исследование.
   - Планирование.
   - Разработка.
   - Стабилизация,
4. Перечислите четыре перспективы модели производственной архитектуры MSF.

- **Бизнес-перспектива** — состоит из множества стратегий и планов, цель которых — переход организации от сложившегося состояния к желаемому. Она описывает организацию работ в компании,
- **Прикладная перспектива** — описывает комплект приложений, использующихся в организации.
- **Информационная перспектива** — описывает то, что должна знать организация для решения своих задач и обеспечения нормального функционирования.
- **Технологическая перспектива** — представляет аппаратное и программное обеспечение, необходимое для работы организации. Технологическая перспектива обеспечивает логичное, независимое от производителя описание инфраструктуры и системных компонентов, которые необходимы для поддержки прикладной и информационной перспектив. Она определяет перечень технологических стандартов и сервисов, необходимых для выполнения задач организации.

5. Как выпускать приложения в период разработки проекта производственной архитектуры?

   Производственная архитектура создается не в вакууме, она обязательно должна реагировать на новую информацию, которая выявляется в процессе выполнения текущих проектов. Метод последовательных версий, который подразумевает обратную связь с проектными группами и пользователями, позволяет постепенно улучшать архитектуру. В противном случае быстро меняющиеся условия сделают бессмысленными все усилия как по созданию производственной архитектуры, так и по ее применению в конкретных проектах.

Глава 2. Приложения масштаба предприятия

Закрепление материала

1. Что такое архитектура приложения? Перечислите характеристики удачного архитектурного решения.

   Архитектура — набор основополагающих проектных решений относительно организации программной системы. К ним относятся:

   - выбор структурных элементов и интерфейсов, образующих систему;
   - поведение системы, определяемое взаимодействием этих элементов;
   - объединение структурных и поведенческих элементов в более крупные подсистемы;
   - стиль организации системы.

Удачное архитектурное решение характеризуется гибкостью, простотой, реализуемостью, четким разграничением проблем, сбалансированным распределением ответственности и наличием компромисса между требованиями и ограничениями.

2. Какими способами можно описать архитектуру программного продукта?

   Это:

   - универсальный язык моделирования;
   - шаблоны;
   - антишаблоны.

3. Охарактеризуйте производственные приложения.

   Производственным приложениям присущи такие качества.

   - Сложность — это многопользовательские многокомпонентные приложения, разработанные большим коллективом программистов и предназначенные для обработки огромного количества данных, параллельного выполнения множества процессов и интенсивного использования распределенных ресурсов. Естественно, что используемые в них алгоритмы очень сложны. Такие приложения, как правило, развертываются на различных платформах, взаимодействуют с другими приложениями и эксплуатируются в течение продолжительного времени.
   - Ориентация на бизнес — цель таких приложений — решение конкретных задач бизнеса; они реализуют правила, процессы и объекты, присущие производственным процессам конкретной организации. Именно для этого их разрабатывают, развертывают и эксплуатируют.
   - Важность — производственные приложения должны быть достаточно надежными, чтобы выдерживать непрерывную эксплуатацию. От них требуется исключительная гибкость в вопросах, касающихся масштабирования и развертывания, а также наличие средств обслуживания, мониторинга и администрирования,

4. Что такое шаблоны и антишаблоны?

   Шаблон — это информация, описывающая структуру удачного семейства решений некоторого класса проблем, возникающих в некоторых условиях. Шаблоны предлагают неочевидное и доказательное решение типовой проектной задачи, допускающее повторное использование. Шаблоны проекта бывают порождающими и нейтральными. Первые можно применять при решении практических задач, вторые — только соблюдать.

   Шаблоны, как правило, применяются к созданию архитектуры приложений «с нуля», Антишаблоны, напротив, предназначены

для исправления ситуации в задачах, существующее решение которых неудовлетворительно.

5. Назовите любые пять из основных принципов управления созданием программного обеспечения.
   - Соответствие целям бизнеса.
   - Ориентация на продукт.
   - Приоритет архитектуры.
   - Контекстно-зависимое проектирование.
   - Использование адекватных инструментальных средств на каждой стадии проекта.
   - Коллективная работа,
   - Понимание целей проекта членами проектной группы.
   - Периодическая демонстрация продукта заказчику.
   - Управление рисками.
   - Компонентный подход.
   - Управление изменениями.
   - Зависимость продукта от ожиданий и приоритетов пользователей.
   - Гибкость метода.
6. Перечислите шесть подмоделей модели производственного приложения.
   - Бизнес-модель.
   - Модель разработки.
   - Модель пользователя.
   - Логическая модель.
   - Технологическая модель.
   - Физическая модель.
7. Что такое модель приложения MSF?

   Модель приложения MSF описывает методы проектирования и разработки программного обеспечения на базе многоуровневой архитектуры, основанной на сервисах. Приложение в этой модели рассматривается как сеть совместно работающих, распределенных и повторно используемых сервисов, решающих поставленные бизнес-задачи. Сервисы приложений — это единицы прикладной логики, включающие методы, реализующие необходимые операции, функции и преобразования. Доступ к сервисам осуществляется с помощью опубликованных интерфейсов, соответствующих спецификациям. В модели приложения **MSF** используются сервисы трех типов; пользовательский, прикладной и сервис данных. В результате становится возможной параллельная разработка разных элементов, обеспечивается более полное использование возможностей технологии, упрощается сопровождение и поддержка, а также расширяются возможности развертывания и масштабируемость приложения.

Глава 3. Проектные группы

Закрепление материала

1. Перечислите шесть ролей модели проектной группы.
   - Менеджер продукта.
   - Менеджер программы.
   - Разработчик.
   - Тестер.
   - Инструктор.
   - Логистик.
2. Назовите основные цели и обязанности каждой роли.
   - Менеджер продукта должен вовремя реагировать на потребности заказчика. Его главная задача — сформировать общее представление о поставленной задаче и о том, как ее решать. Он должен ответить на вопрос: «Зачем мы делаем все это?» — и убедиться, что все члены группы знают и понимают ответ на него. Основная обязанность менеджера продукта — выполнение требований заказчика.
   - Задача менеджера программы — вести процесс разработки с учетом всех ограничений. Его главная обязанность — выполнить все стадии разработки так, чтобы нужный продукт был выпущен в нужное время.
   - Задача группы разработки — технологическое консультирование проектной группы и собственно создание продукта. В качестве консультантов они предоставляют исходные данные для проектирования, проводят оценку технологий, а также разрабатывают прототипы и тестовые системы, необходимые для проверки решений и сокращения рисков на ранних стадиях процесса разработки. Разработчики не решают, какими функциями должен обладать продукт — они реализуют функциональные спецификации, что и является их главной задачей.
   - Задача тестеров — испытать продукт в реальных условиях, дабы определить, что в продукте работает и что не работает, и нарисовать таким образом точный «портрет» приложения. Естественно, что для проведения гестов нужно отлично разбираться и в требованиях пользователей, и в том, как их удовлетворить. Тестеры разрабатывают стратегию, планы, графики и сценарии тестирования, которые позволяют убедиться, что все ошибки выявлены и исправлены до выпуска приложения. Основная обязанность группы тестирования — информировать

проектную группу обо всех проблемах продукта и гарантировать, что все проблемы устранены до выпуска продукта.

- Цель группы обучения — повысить эффективность труда пользователей. Поэтому инструкторы «принимают сторону» пользователей подобно тому, как менеджеры продукта представляют интересы заказчика. Однако перед пользователями инструкторы выступают в роли представителей проектной группы. В этом последнем качестве группа обучения отвечает за выпуск удобного, полезного продукта, которому практически не нужна поддержка.
- Группа логистики представляет интересы служб поддержки и сопровождения, справочных служб и других служб канала доставки. Она занимается развертыванием продукта и его сопровождением и контролирует продукт с этой точки зрения в процессе проектирования. Кроме того, ее задача — составление графиков развертывания приложения. Логистики, менеджеры продукта и менеджеры программы совместно определяют порядок передачи продукта пользователям и организации, после чего логистики готовят их к развертыванию приложения.

3. Каковы особенности проектных групп больших и малых проектов?

В крупных проектах часто приходится делить проектную группу на тематические и функциональные подгруппы.

- **Тематические группы** — это небольшие подгруппы из одного или нескольких человек, роли которых различны. Каждой из таких групп выделяется некий набор функциональных возможностей приложения, за все стороны проектирования и разработки которого она и отвечает (включая составление проекта и графика реализации). Например, какой-либо группе можно предоставить решать задачу вывода данных на печать.
- **Функциональные группы** формируются в рамках одной роли. Они нужны в очень крупных проектных группах или при работе над крупными проектами, когда отдельные роли нуждаются в дополнительном подразделении.

В проектной группе за каждое направление должен отвечать как минимум один человек. При реализации крупного проекта возникает затруднение, связанное с эффективным обменом информацией. В небольших организациях или при работе над мелкими проектами роли можно совмещать. Однако в этом случае существует другая проблема — как «усидеть на нескольких стульях» одновременно, не упустив из виду ни одной существенной детали проекта с точки зрения каждой роли. При совмещении роли нужно не забывать два принципа.

- **Нельзя совмещать разработку с другими видами деятельности** - создателей приложения не стоит отвлекать от основной задачи. Если «повесить» на разработчиков дополнительные обязанности, то, скорее всего, график работ будет нарушен, а дату выпуска продукта придется отодвинуть.
- **Конфликт интересов** — нельзя совмещать роли, интересы которых противоположны. Пример — менеджер продукта и менеджер программы. Первый хочет выполнить все требования заказчика, второму же надо уложиться в график и бюджет. Если совместить эти роли, возникает опасность упустить просьбу заказчика о внесении изменений в проект либо, напротив, принять ее без должного анализа влияния на график работ. Таким образом, назначение на эти роли разных людей позволяет соблюсти интересы всех участников проекта.

4. Перечислите стадии процесса разработки и их связь с развитием группы.
   - Заинтересованность.
   - Надежда, оптимизм, готовность к работе.
   - Определение задач и решений.
   - Проявление взаимопомощи.
   - Доверительные уважительные отношения;
   - Единение.
5. Назовите два метода обучения, позволяющие повысить эффективность труда членов группы.
   - **Методическое обучение** — обучение разработчиков методологии разработки программного обеспечения.
   - **Техническое обучение** — обучение технике разработки, включая языки программирования и средства разработки.

Глава 4. Процесс разработки

Закрепление материала

1. Охарактеризуйте модель водопада и спиральную модель.
   - **Модель водопада** представляет процесс разработки в виде строго упорядоченной последовательности этапов. К ним относится сбор требований к системе, анализ, проектирование, кодирование и тестирование модулей, интеграция, тестирование и передача в эксплуатацию. Для перехода к следующему этапу необходимо полностью закончить работу над текущим и тщательно описать его результаты.

- **Спиральная** модель характеризуется итерационным жизненным циклом, который подразделяется на стадии исследования (анализ требований и предварительное планирование), проработки (проектирование приложения), создания (фактическая разработка приложения) и переходный период (оценка и стабилизация приложения). Каждая из этих стадий подразделяется на пять фаз: сбор требований, проектирование, реализация, развертывание и управление. В рамках спиральной модели процесс разработки состоит из упомянутых выше четырех стадий, на каждой из которых эти пять фаз выполняются многократно. Цель итерационного подхода — добиться последовательного уточнения характеристик и архитектуры продукта, которое обеспечивает его постоянное приближение к окончательному виду.

2. Перечислите основные этапы универсального процесса.
   - Требования: сбор бизнес-, технических и прикладных требований к проекту.
   - Анализ: бизнес- и прикладное моделирование на основе собранных требований.
   - Проектирование: создание архитектуры на основе объектно-ориентированного подхода.
   - Реализация: разработка спроектированного приложения (на ранних стадиях — разработка прототипов).
   - Тестирование: проверка сделанной работы.
3. Перечислите фазы и промежуточные этапы универсального процесса.
   - Фаза «Исследование» предназначена для создания модели предметной области.
   - Итерации фазы «Проработка» ставят своей целью создание базовой архитектуры.
   - Итерации фазы «Создание» преследуют цель создания продукта путем последовательного выпуска версий с постепенно расширяющимися функциональными возможностями;
   - Переходный период необходим для проверки готовности продукта к эксплуатации; эта фаза завершается выпуском продукта,
4. Опишите цели и задачи каждой из фаз модели процесса разработки MSF.
   - **«Анализ»** — цель этой фазы выработать единую концепцию проекта для всех его участников. Единая концепция предполагает, что понимание бизнес-проблемы, на решение которой направлен проект, согласовано всеми сторонами, а также выработаны решение, отвечающее ожиданиям заказчика, и обоснованная оценка проектных ограничений.

- **«Планирование»** — на этой стадии определяется архитектура продукта. Она основана на концептуальной, логической и физической проектных моделях. Важная составляющая разработки надежного и хорошо масштабируемого приложения — применение современной многоуровневой архитектуры. Она, кстати, не ограничивает возможности реализации: многоуровневая архитектура применима к монолитным, клиент-серверным и многоуровневым приложениям. На стадии «Планирование» уточняются оценки графика, ресурсов и характеристик продукта. К концу этой фазы группа определяет график работ, ресурсы, которые ей понадобятся, и характеристики продукта, которые будут реализованы.
- **«Разработка»** — на этой стадии основная задача — создание приложения. Итерационный подход, применявшийся на предыдущих стадиях, на этой стадии приобретает особое значение. На стадии «Разработка» группа, как правило, последовательно выпускает несколько версий приложения (альфа-, бета- и окончательную версию) и занимается устранением всех выявленных проблем. Полное устранение всех ошибок и проблем на этой стадии не обязательно — достаточно, если они будут исследованы. Цель этой стадии — создать приложение, отвечающее заявленным требованиям и готовое к тестированию.
- **«Стабилизация»** — на этой стадии тестируются производительность и совместимость приложения, устраняются все найденные проблемы, а также завершается создание всех материалов, необходимых группам эксплуатации и сопровождения. Вообще, одна из задач стадии «Стабилизация» — доделать все, что не закончено. Документация, инструкции по установке, окончательный список проблем с рекомендациями для будущих версий, руководство по развертыванию — все эти материалы на стадии «Стабилизация» приобретают окончательный вид. Важная особенность этой фазы — участие пользователей в тестировании продукта. Кроме того, это период интенсивного обучения групп эксплуатации и сопровождения. На этой стадии группа логистики передает обязанности по сопровождению продукта группам, ответственным за эту работу в организации. Процесс завершается выпуском продукта.

5. Перечислите результаты каждой из фаз модели процесса разработки MSF.
   - **«Анализ»** — концепция, документ оценки рисков, структура проекта. Кроме того, рекомендуется добавить к этому списку

прототип приложения, демонстрирующий корректность и реализуемость концепции.

- «**Планирование**» — функциональные спецификации, основной план проекта, основной график проекта, пересмотренный документ оценки рисков. Кроме того, рекомендуется включить в этот список прототип системы, демонстрирующий корректность проектных решений и позволяющий основным участникам проекта оценить архитектуру приложения и ее реализуемость.
- «**Разработка**» — окончательная версия пересмотренных функциональных спецификаций, пересмотренные план и график проекта, пересмотренный сводный документ оценки рисков, исходные тексты приложения и исполняемых модулей, средства повышения эффективности работы пользователей и сопроводительные материалы, тестовые спецификации и схемы тестирования.
- «**Стабилизация**» — окончательная версия продукта, документация к окончательной версии, материалы для сопровождения приложения и поддержки пользователей, результаты и средства тестирования, архивы проекта, обзор всех основных этапов проекта.

6. Опишите преимущества последовательного выпуска версий.

- **Контакт с заказчиком** — выпуск версий позволяет постоянно поддерживать контакт группы с заказчиком, информируя его о состоянии продукта, и учитывать в следующих выпусках лучшие идеи обеих сторон.
- **Ранняя версия** — группа может быстро выпустить продукт с базовым набором функциональных возможностей и собрать отклики от заказчика и пользователей. Если заказчик видит, что работа над продуктом идет в соответствии с графиком, он гораздо спокойнее относится к переносу некоторых функциональных возможностей в следующий выпуск.
- **Ограниченный круг решаемых вопросов** — выпуск версий позволяет всегда иметь дело с относительно компактным кругом вопросов и решать эти проблемы на стадии выпуска очередной версии.
- **Ясность целей** — выпуск версий ставит перед группой четкие и ясные задачи, что позволяет сосредоточиться на решении вопросов, связанных с текущей версией, и быстрее добиваться результата. Роль версий велика и в уточнении графика — они позволяют разбить работу на небольшие, легко управляемые части, которые имеют ясную цель и дают конкретный результат.
- **Свобода и гибкость** — выпуск версий дает группе больше свободы в выборе приоритетов и придает дополнительную гибкость процессу проектирования, позволяя своевременно реагировать на изменение бизнес-требований. Гибкость подхода

снижает общую неопределенность за счет распределения конкретных составляющих и функциональных возможностей по версиям — например, при изменении требований достаточно просто изменить приоритеты для очередного выпуска. Кроме того, группа всегда располагает стабильной версией продукта.

- **Последовательное и постоянное расширение функциональных возможностей продукта** — выпуск версий позволяет группе планомерно расширять функциональные возможности продукта. Последовательность и планомерность производят чрезвычайно выигрышное впечатление на заказчика.

7. Опишите концепцию компромиссного треугольника и методы достижения компромисса.

   Успех любого проекта зависит от равновесия трех важнейших элементов: ресурсов (люди и деньги), характеристик (продукт и его качество) и графика.

   В начале проекта взаимосвязь этих элементов довольно туманна. На этом этапе группа располагает лишь приблизительными оценками того, что предстоит сделать, какие ресурсы потребуются и когда продукт будет готов. На стадии «Планирование» стороны компромиссного треугольника постепенно приобретают большую определенность. По окончании этой фазы группа должна отчетливо представлять себе доступные ресурсы, характеристики продукта и дату выпуска.

   Важно помнить, что три стороны компромиссного треугольника взаимосвязаны. Изменение одной из них влияет на остальные. Понимание и применение этой концепции дает проектной группе мощный инструмент адекватной реакции на изменение требований или условий в ходе проекта.

Глава 5, Предпроект

Закрепление материала

1. Каковы основные цели фазы «Анализ»?

   Фаза «Анализ»:

   - служит первой фазой планирования и проектирования:
   - позволяет добиться понимания и согласия между всеми участниками с самого начала проекта;
   - помогает группе объединить разные точки зрения в общую концепцию;
   - обеспечивает основу для будущего планирования;

- выявляет факторы, которые заказчик и основные участники считают важнейшими для проекта.

2. Какие роли отвечают за достижение целей фазы «Анализ»?
   - Менеджер продукта: основное действующее лицо фазы «Анализ», отвечает за подготовку документа «Концепция проекта», за работу с заказчиком и его вовлечение в разработку прототипа, а также за управление рисками.
   - Менеджер программы: отвечает за формулировку целей проектирования, описание концепции решения, выявление структуры проекта и управление рисками.
   - Разработчик: отвечает за разработку прототипов, выбор основных направлений разработки, выявление требований и взаимосвязей и за управление рисками.
   - Инструктор: отвечает за сбор требований, касающихся удобства использования, за работу с пользователями, за их вовлечение в разработку прототипов и за управление рисками.
   - Тестер: в обязанности тестера входит разработка стратегии тестирования, формулировка критериев приемки продукта, разработка системы выявления ошибок и управления рисками, и собственно управление рисками.
   - Логистик: отвечает за вопросы развертывания и сопровождения и их влияние на проект и за управление рисками.
3. Каковы основные компоненты концепции?
   - Формулировка концепции.
   - Результаты исследования требований пользователей.
   - Информация о конкурентоспособности решения.
   - Описание функциональных возможностей.
   - Приблизительный график.
4. В чем преимущество использования прототипов?

   Для успешного выполнения проекта необходим эффективный обмен информацией с заказчиком и пользователями. Проектную группу необходимо проинформировать о бизнес-проблемах, а информацию о том, как продукт решит эти проблемы, надо довести до заказчика и пользователей.

   Существенно облегчит такую связь прототип приложения, который полностью или частично демонстрирует концепцию продукта. Его разрабатывают в ходе фазы «Анализ» и включают в результаты этапа «Одобрение концепции». Прототип может представлять собой как работающее приложение, так и просто набор экранов. Прототип помогает разъяснить концепцию и выявить дополнительные вопросы группы, заказчика и пользователей продукта.

5. Что такое риск?

   Риск — это *возможность потерь*. Для конкретного проекта это может быть:

   - снижение качества продукта;
   - увеличение расходов;
   - срыв сроков;
   - невозможность достижения целей проекта.

6. Что такое управление рисками?

   Выявление и превентивное снижение потенциальных рисков проекта, а также создание и использование системы управления рисками, реализовавшимися в ходе проекта.

Глава 6. **План проекта**

Закрепление материала

1. Зачем нужна фаза «Планирование»?

   На фазе «Анализ» проектная группа отвечала на вопрос: «Можно ли разработать технологию решения конкретной бизнес-проблемы, и если да, то как?» В результате группа в обших чертах разобралась в поставленной задаче и наметила ее решение. На этапе «Планирование» ставится другой вопрос: «Сколько времени займет реализация предложенного решения?» Ответом на него должен стать подробный план, согласованный проектной группой и заказчиком.

   Из всех фаз разработки MSF труднее всего довести до конца именно планирование. Фаза «Анализ» — это свободный полет мысли, исследование всех возможных решений, волнительный, но чрезвычайно интересный период. Фаза «Разработка» — время поработать руками, реализовать все задумки: «Наконец-то мы что-то делаем!» А вот фаза «Планирование» очень изматывает, ведь нужно отвечать на сложные вопросы: «Какую функцию отложить до следующей версии? Какая технология еще не созрела для реализации? Не слишком ли высоки затраты?» Фаза «Планирование» — сложная, кропотливая работа, поэтому часто возникает искушение пропустить ее.

   Но именно на этом этапе решается судьба проекта — ждет его успех или крах. И лучше выявить проблемы на бумаге во время проектирования, чем столкнуться с ними при реализации. Намеченные на этапе «Анализ» идеи должны пройти через горнило фазы «Планирование», и только после этого вы можете быть уверены, что эти идеи удастся реализовать на стадии разработки и стабилизации.

2. Опишите три этапа проектирования, входящие в процесс проектирования MSF.

   Процесс проектирования MSF состоит из трех частей: концептуального, логического и физического проектирования, каждая из которых служит основой для одноименной модели. На каждой из стадий проектирования решаются конкретные задачи, а результат описывается в своих терминах.

   - **Концептуальное проектирование** рассматривает задачу с точки зрения пользователей и бизнеса и описывает задачу и ее решение в терминах сценариев.
   - **Логическое проектирование** рассматривает решение с точки зрения проектной группы и описывает его в виде набора взаимодействующих сервисов.
   - **Физическое проектирование** рассматривает решение с точки зрения разработчиков и определяет сервисы и технологии, необходимые для реализации решения.

   Задачи трех стадий процесса проектирования таковы.

   - **Концептуальное проектирование** — исследование потребностей бизнеса и пользователей. Создание с помощью пользователей и других заинтересованных сторон сценариев, полностью отражающих требования к решению.
   - **Логическое проектирование** — определение решения и организация взаимодействия его элементов. На основе сценариев, выработанных при концептуальном проектировании, формулируется абстрактная модель решения. Сценарии концептуального проекта используются для создания объектов и сервисов, прототипов пользовательского интерфейса и логического проекта базы данных.
   - **Физическое проектирование** — уточнение решения с учетом доступных технологий, возможностей реализации и необходимой производительности. На основе результатов предыдущего этапа — логического проектирования — на этой стадии создаются компоненты, спецификации пользовательского интерфейса и физический проект базы данных.

3. Какую информацию содержат функциональные спецификации?

   - **Резюме проекта** — точка зрения проектной группы на продукт, его обоснование и основные ограничения. Основывается на документе «Концепция», разработанном на фазе «Анализ».
   - **Цели проекта** — описание целей группы. Разработчики используют его при изучении таких вопросов, как производитель-

ность, надежность, удобство использования и доступность продукта. Эти цели формулируются на стадии «Анализ»,

- **Требования всех сторон** — описание того, что, по мнению заказчика, пользователей и других участников проекта, должно делать приложение. Требования надо упорядочить по значимости и, кроме того, устранить конфликты между ними,
- **Резюме по использованию** — описывает, кто и как будет использовать продукт. Это не что иное, как собранные вместе сценарии использования, созданные на этапе проектирования.
- **Функциональные возможности** — точное описание характеристик продукта, включая эскиз пользовательского интерфейса, средства навигации и подробное описание его функциональных возможностей.
- **Зависимости** — описание внешних объектов, от которых зависит продукт. В него входят как высокоуровневые (например, взаимодействие с корпоративными системами), так и низкоуровневые (например, разделяемые компоненты) объекты,
- **Резюме графика** — краткое описание основного графика проекта с определением этапов, основных результатов и даты выпуска продукта.
- **Риски** — перечень рисков; риски, требующие дополнительного изучения, должны быть специально отмечены.
- **Приложения** — описывают все, что не вошло в предыдущие разделы.

4. Что такое план проекта?

Основной план проекта описывает процедуру выполнения проекта. в том числе подробные планы, составленные разными членами проектной группы. Предназначен для синхронизации работы группы.

Цель основного плана проекта:

- объединить планы разных членов группы;
- описать разные направления деятельности;
- синхронизировать планы различных подгрупп.

За основной план проекта отвечает менеджер программы, считающийся координатором планирования проекта и работы над ним. По каждому направлению составляются отдельные планы, которые включаются в основной.

Глава 7, Технологии пользовательского уровня

Закрепление материала

1. Что такое композиция пользовательского интерфейса?

   Композиция, или размещение элементов пользовательского интерфейса, определяет не только его эстетику, но и удобство применения. Композиция включает:

   - размещение элементов управления;
   - согласованность элементов управления;
   - понятность элементов;
   - использование разделителей;
   - простоту дизайна.

2. Что понимается под удобством интерфейса?

   Простота работы пользователей с приложением. Обычно в это понятие включается дизайн меню, простота доступа к функциональным возможностям приложения, средства навигации и система помощи пользователям.

3. Перечислите вопросы, возникающие при проектировании пользовательского уровня.

   - Ограничивают ли требования к приложению тип пользовательского интерфейса?
   - Влияют ли средства защиты (например, брандмауэры) на взаимодействие рабочих станций с серверами?
   - Какие операционные системы, установленные на рабочих станциях, надо поддерживать?
   - Какие Web-обозреватели надо поддерживать?
   - Можно ли на рабочих станциях устанавливать и запускать компоненты COM?
   - Доступны ли рабочим станциям удаленные COM-компоненты?

4. Какими способами можно применять COM-объекты на пользовательском уровне?

   При установке и запуске COM-компонентов «родного» приложения на рабочих станциях пользователей проблем, как правило, не возникает — компоненты устанавливаются одновременно с соответствующим Win32-приложением. С Web-приложениями немного сложнее. Если обозреватель поддерживает клиентские COM-компоненты, то они загружаются и автоматически устанавливаются при первом обращении к ним.

Элементы ActiveX — это COM-объекты, загружаемые с Web-сервера и выполняемые на компьютере пользователя с помощью обозревателя. Некоторые пользователи или организации, в которых они работают, считают автоматическую загрузку и установку исполняемых программ на клиентскую рабочую станцию небезопасной, поэтому запрещают эти действия. Если вы создаете приложение именно для таких заказчиков, ограничьтесь уже установленными клиентскими COM-компонентами (в том числе, ActiveX-элементами). Некоторые пользователи самостоятельно принимают решение о загрузке и установке каждого компонента; в этом случае использовать клиентские COM-компоненты можно.

5. Каким образом ASP взаимодействует с пользовательским и прикладным уровнями и уровнем данных?

   В интернет-приложениях Web-обозреватель реализует пользовательский уровень на основе HTML-страниц. Запросы этого уровня передаются Web-серверу по протоколу HTTP. В ответ на запрос клиентского Web-обозревателя на сервере активируются ASP-страницы, которые способны сгенерировать и вернуть обозревателю необходимую HTML-страницу. ASP можно применять и для создания интерфейса, Следовательно, ASP считается частью пользовательского уровня. Однако в ASP-страницах должны содержаться сценарии, выполняющиеся на сервере и использующие бизнес-объекты среднего уровня. Эти бизнес-объекты способны, в свою очередь, вызывать объекты данных, таким образом получая доступ к уровню данных. С другой стороны, HTML и клиентские сценарии, реализующие пользовательский уровень, реализуются в ASP-страницах, В любом случае, ASP необходим для взаимодействия между пользовательским и прикладным уровнями.

Глава 3. **Технологии прикладного уровня**

Закрепление материала

1. Что такое спецификация COM?

   Спецификация описывает модель компонентных объектов и ее реализацию. Она доступна на Web-узле Microsoft по адресу *WWW, microsoftcom/com/resources/specasp.*

2. Каким образом COM работает на разных компьютерах?

   Благодаря сервисам распределенной модели компонентных объектов DCOM. Формально DCOM — это протокол выполнения вызовов объектов на разных компьютерах, однако часто этим терми-

ном называют всю концепцию взаимодействия удаленных объектов COM.

3. Опишите механизм защиты в MTS.

   В MTS доверие определяется на уровне пакетов. Обращения к пакету могут быть защищенными, но вызовы внутри него считаются надежными, Таким образом, требования к защите приложений сильно влияют на проект пакета. Если обращения к компоненту требуют авторизации, клиенты и компоненты должны быть размещены в разных пакетах. В одном пакете могут находиться только компоненты, способные безопасно обращаться друг к другу.

4. Что такое COM+?

   Программная модель, базирующаяся на COM и MTS. Первая версия COM+ поставляется в составе Windows 2000.

5. Перечислите основные сервисы COM+ 1.0.

   Вот они:

   - серверы;
   - транзакции;
   - защита;
   - администрирование;
   - распределение загрузки;
   - очередь компонентов:
   - события;
   - база данных в памяти.

Глава 9. Технологии уровня данных

Закрепление материала

1. Что такое UDA?

   Разработанная компанией Microsoft *универсальная архитектура данных* (Universal Data Architecture, UDA) предназначена для организации высокопроизводительного доступа к данным любого типа (структурированным и неструктурированным, связанным и несвязанным), хранящимся на предприятии. UDA — набор COM-интерфейсов, реализующих концепцию доступа к данным. Она основана на интерфейсах OLE DB для создания компонентов, работающих с базами данных. OLE DB позволяет хранилищам информации открывать свои функции без необходимости имитировать реляционный источник данных. Кроме того, в рамках этой технологии универсальным сервисным компонентам (например, специализированным обработчикам запросов) разрешено расширять функции более простых поставщиков данных. Поскольку основ-

ная цель OLE DB — эффективный доступ к информации, а не простота использования, в UDA добавлен интерфейс прикладного уровня Microsoft ActiveX Data Objects (ADO). ADO поддерживает двойные интерфейсы, которые можно применять в языках сценариев, в C++, в Microsoft Visual Basic и других средствах разработки.

UDA — платформа и набор средств разработки, определяющий стандарты и технологии, связанные с доступом к хранилищам данных масштаба предприятия. Эта архитектура лежит в основе концепции разработки приложений Microsoft. Кроме того, UDA обеспечивает высокопроизводительный доступ к различным реляционным и нереляционным информационным хранилищам и обладает простым интерфейсом, не зависящим от языка реализации.

2. Какие компоненты доступа к данным, разработанные Microsoft, вы знаете?

   Microsoft Data Access Components (MDAC) — это реализация UDA, включающая как ADO, так и компонент доступа OLE DB для ODBC. Это позволяет ADO обращаться к любой базе данных, снабженной драйвером ODBC (фактически ко всем основным СУБД). Существуют компоненты доступа OLE DB и для других типов хранилищ — например, почтовых хранилищ Microsoft Exchange, служб каталога Windows NT и файловой системы Microsoft Windows, использующей Microsoft Index Server.

3. Какой компонент доступа к данным рекомендуется использовать?

   Рекомендованная технология — ADO. ADO и OLE DB обеспечивают доступ к источникам данных практически любого типа, и поэтому Microsoft рекомендует эту технологию для разработки новых приложений.

4. Каким образом обратиться к хост-данным с помощью COM?

   С помощью OLE DB-провайдера для AS/400 и VSAM.

5. Что такое IMDB?

   База данных в памяти (In-Memory Database, IMDB) — один из компонентов COM+, позволяющий кэшировать таблицы БД в памяти.

Глава 10, Тестирование и производственный цикл

Закрепление материала

1. Перечислите этапы производственного цикла,

   Жизненный цикл приложения состоит из четырех этапов, составляющих *производственный цикл*:

- разработка;
- тестирование;
- сертификация;
- эксплуатация.

2. В чем состоят преимущества производственного цикла?
   - Все проблемы обнаруживаются на стадии тестирования или сертификации, а не на стадии эксплуатации.
   - Гарантируется исчерпывающее тестирование.
   - Все изменения вносятся организованно,
3. Какие вопросы нужно изучить при определении требований к производительности?
   - Ограничения.
   - Функции, выполняемые приложением.
   - Нагрузка на приложение.
4. Что такое эталон производительности?

   Результаты выполнения первоначального набора тестов.
5. Назовите две категории тестов.

   На стадии «Разработка» проводится *полное тестирование* — тщательная проверка работы всех функций приложения в закрытой среде. На фазе «Стабилизация» выполняется *тестирование работоспособности* приложения, его соответствия схемам и сценариям использования, собранным на этапе «Анализ». На этой стадии (называемой бета-тестированием) подключаются пользователи, так как желательно, чтобы оно проходило в условиях, приближенных к реальным. Необходимость устранения ошибок является приоритетной задачей фазы «Стабилизация», а так как на этом этапе продукт готовится к сдаче в эксплуатацию, важнее всего возможность управленияошибками.
6. Перечислите этапы управления ошибками.
   - Выявление.
   - Присвоение приоритета.
   - Назначение ответственного.
   - Устранение.
   - Изъятие.

Глава 11. Защита приложения

Закрепление материала

1. Что такое аутентификация?

   Идентификация пользователя и определение его прав.

2. Перечислите основные методы Web-аутентификации.
   - Схема «запрос — ответ» Windows NT.
   - Жетоны.
   - Цифровые сертификаты.
3. Что такое шифрование?

   Преобразование информации, передаваемой по открытым каналам, к виду, гарантирующему ее защиту от несанкционированного доступа.
4. Что такое контроль доступа?

   Контроль доступа — важная составляющая архитектуры приложения. Она позволяет контролировать, кто и как использует систему, Программы должны обеспечивать конфиденциальность пользовательских данных и защищать свои компоненты и сервисы от несанкционированного доступа. Защищенное приложение предоставляет доступ к своим сервисам только тем, кому это разрешено. К тому же каждый его компонент, сервис и файл защищены от несанкционированного просмотра и изменения.
5. Зачем нужен аудит?

   Аудит — механизм отслеживания работы пользователей с приложением, позволяющий выявлять прорехи в системе защиты приложения. Кроме того, информация, собранная в процессе аудита, позволяет оптимизировать приложение для работы в конкретной эксплуатационной среде.
6. Какие службы Windows NT поддерживают аудит?
   - **Системный аудит Windows NT** — это регистрация пользователей в системе и выход из нее, доступ к объектам и файлам, назначение прав, управление группами и пользователями, изменение параметров защиты, запуск и завершение работы системы, а также управление процессами,
   - **Аудит файлов и каталогов** — проверяется наличие сбоев при выполнении следующих операций над файлами и каталогами: чтение, запись, выполнение, удаление, изменение правил и изменение владельца.
   - **Аудит реестра** — проверяется наличие сбоев при проведении следующих операций над реестром: запроса значений, установки значения, создания разделов, перечисления разделов, уведомления, создания связей, удаления, записи DAC и контроля чтения.

Глава 12. Стадия «Разработка» и ее результаты

Закрепление материала

1. Перечислите результаты стадии «Разработка».
   - Пересмотренные функциональные спецификации.
   - Пересмотренный план проекта.
   - Пересмотренный график проекта.
   - Пересмотренный сводный документ оценки рисков.
   - Исходные тексты приложения и исполняемые модули.
   - Средства повышения эффективности работы пользователей и сопроводительные материалы.
   - Тестовые материалы.
2. Что такое промежуточная версия продукта?

   В процессе разработки каждый вариант приложения считается промежуточной версией. Промежуточные версии, полученные ближе к завершению стадии «Разработка», передают пользователям — это так называемые альфа- и бета-версии.
3. Перечислите преимущества выпуска промежуточных версий продукта.

   Для успешной реализации целей проекта следует разбить стадию «Разработка» на несколько промежуточных этапов разумного размера. Хотя группа может добавить свои промежуточные этапы, обычно стадию «Разработка» подразделяют на выпуск промежуточной версии, за которым следуют выпуск альфа- и бета-версий, что свидетельствует о достижении этапа «Завершение разработки».

   Промежуточные этапы позволяют постоянно контролировать ход выполнения проекта. Код часто пишется параллельно несколькими группами, и поэтому очень важно иметь возможность оценить результаты каждой из них. Промежуточные контрольные точки заставляют членов группы синхронизировать усилия; этот процесс иногда называют *интеграционным тестированием.*

   Группа должна стремиться как можно раньше добиться создания стабильных версий пользовательского интерфейса и базы данных. Хотя они и не включены в список формальных промежуточных этапов стадии «Разработка», их лучше закончить как можно быстрее. Дело в том, что многие другие работы — например, создание обучающих материалов — в значительной степени зависят от степени завершенности пользовательского интерфейса. Важно опре-

делиться и со структурой базы данных, поскольку она диктует компоновку различных функциональных возможностей продукта, а эти решения также принимаются на ранней стадии разработки. Таким образом, чем раньше завершена разработка пользовательского интерфейса и структуры БД, тем меньше изменений придется вносить в код и документацию на следующих этапах работы.

Стадия «Разработка» — итерационный процесс решения четко поставленной задачи. Промежуточные выпуски могут иметь конкретную цель — скажем, тестирование определенных функций, части пользовательского интерфейса или развертывания продукта. Некоторые промежуточные выпуски иногда специально адресуются пользователям или заказчику. В первых промежуточных выпусках надо реализовать приоритетные функциональные возможности продукта, чтобы продемонстрировать способность группы выполнить задачу. В первых выпусках для внутреннего пользования следует уделить особое внимание архитектурным особенностям, связанным с наибольшим риском или наибольшей неопределенностью, чтобы оценить их осуществимость или, наоборот, на ранней стадии изменить проектное решение.

Разбиение большого проекта на составные части помогает группе видеть ежедневные достижения, что позволяет действовать более конструктивно. Кроме того, чем раньше вы подкорректируете проект, тем дешевле это вам обойдется. Промежуточные версии для внутреннего и внешнего использования помогают повысить качество продукта, что закладывает основу для следующих этапов разработки и позволяет группе своевременно исправить ошибки и устранить дефекты проекта.

4. Перечислите три метода устранения ошибок.

Исправление ошибок не завершает работу с ними. Это только промежуточный этап на пути к устранению ошибок, которое происходит после того, как тестеры удостоверятся, что исправление полностью устраняет известную проблему и не создает новых. Устраненная ошибка может иметь следующий статус:

- **исправленная** — разработчики исправили ошибку, протестировали исправление, проверили код, изменили номер версии программы и отправили ее тестеру, сообщившему об ошибке;
- **повторная** — ошибка, совпадающая с уже находящейся в БД; в БД записывается ссылка на первую запись об ошибке, после чего повторная ошибка считается разрешенной и закрытой;
- **отложенная** — в текущем выпуске ошибка не будет исправлена, но в следующих версиях программы такая работа может быть

проведена. Такое обозначение необходимо, если ошибку нужно исправить, но в данный момент на это нет времени;

- **проектная** — поведение программы, классифицированное как ошибка, является корректным и соответствует функциональным спецификациям;
- **невоспроизводимая** — разработчики не могут проверить наличие ошибки, так как не удается воспроизвести ее;
- **неисправляемая** — ошибка не будет исправлена в текущей версии приложения, так как разработчики считают, что эффект от ее исправления не стоит затраченных усилий, или менеджерам и заказчикам она кажется малозначимой.

5. Когда проектная группа готова к переходу на стадию «Стабилизация»?

   Когда:

   - все необходимые функциональные возможности приложения реализованы (хотя, возможно, и не самым оптимальным образом);
   - продукт прошел первоначальное тестирование; продолжается устранение выявленных ошибок (завершение этой работы на стадии «Разработка» не обязательно);
   - проектная группа и другие участники проекта согласны с тем, что все функциональные возможности отвечают концепции и функциональным спецификациям и реализованы успешно;
   - завершена подготовка к тестированию производительности продукта и его стабилизации.

Глава 13. Стабилизация продукта

Закрепление материала

1. Какова основная цель стадии «Стабилизация»?

   Подготовить продукт к сдаче в эксплуатацию.

2. Как группа продвигается к выпуску продукта?

   На стадии стабилизации проектная группа может выпустить несколько промежуточных версий приложения. Они позволяют выявить конкретные проблемы и устранить их. Почти окончательная версия продукта называется «безошибочной». Это первый промежуточный выпуск, в котором все известные проблемы тем или иным способом устранены (зафиксированы, отложены или признаны несущественными). После подготовки этой версии выполняется детальное тестирование готовности продукта; положительный результат тестирования позволяет считать продукт готовым и присвоить выпуску статус окончательного,

3. Перечислите основные этапы стадии «Стабилизация».

   В отличие от других стадий процесса разработки, эта не подразделяется на этапы. Их заменяют промежуточные выпуски продукта. Для каждого промежуточного выпуска выполняются одни и те же действия:

   - устранение ошибок;
   - синхронизация всех составляющих конечного продукта;
   - выпуск версии и ее тестирование.

4. Перечислите промежуточные выпуски продукта на стадии «Стабилизация».
   - Один или несколько промежуточных выпусков.
   - «Безошибочная» версия.
   - Один или несколько выпусков-кандидатов.
   - Окончательный выпуск,

5. Каковы результаты стадии «Стабилизация»?
   - **Окончательная версия продукта —** исходные тексты и исполняемые модули.
   - **Документация к окончательной версии —** описание окончательной версии и последних изменений, не отраженных в документации продукта.
   - **Материалы для сопровождения приложения и поддержки пользователей** — окончательная версия учебных и инструктивных материалов.
   - **Результаты тестирования** — база данных выявленных проблем необходима как для сопровождения продукта, так и для будущих проектов.
   - **Архивы проекта** — вся информация, имеющая отношение к продукту и его разработке, независимо от того, вошла она в окончательный выпуск или нет;
   - **Документация** — вся документация проекта, включая ее версии на каждом из промежуточных этапов.

6. Какие методы рекомендуется применять для развертывания приложений в организации?
   - Microsoft Systems Management Server (SMS).
   - Регистрационные сценарии.
   - Рассылку по электронной почте.
   - Распространение через Web.

Глава 14. Обсуждение проекта

Закрепление материала

1. Перечислите выгоды от рецензирования завершенных проектов,
   - **Логическое завершение проекта** — формальное завершение проекта особенно важно в случае, когда группа, выполнявшая проект, сразу переключается на работу над новым проектом или расформировывается.
   - **Возможность обмена мнениями** — обсуждая проект, его участники получают возможность «облегчить душу». Если не предоставить менеджерам и сотрудникам возможность высказаться организованно, это может произойти стихийно, что чревато нежелательными последствиями. На встрече следует говорить о работе коллектива, а не о действиях отдельных сотрудников. Такая встреча *ни в коем случае* не должна стать сведением счетов или поиском виновных.
   - **Улучшение морального климата** — сотрудники получают возможность поделиться как отрицательными, так и положительными мнениями по поводу работы над проектом. Разработка программного обеспечения традиционно считается скорее индивидуальной, чем коллективной деятельностью; в рамках подхода, изложенного в этой книге, укрепление коллективного духа — важнейшая задача.
   - **Анализ методов работы** — для будущих проектов очень важен анализ слабых и сильных сторон проекта с точки зрения проектной группы. Такой анализ — основа выработки корпоративных стандартов и оптимальных методов работы. Помните, что модель процесса разработки MSF требует пристального внимания к методологии — это позволяет повысить эффективность новых проектов за счет применения оптимальных методов. Результаты анализа методов выполнения предыдущих проектов очень полезны на первых стадиях разработки нового проекта. Следует позаботиться о сборе этих результатов, например, посредством организации соответствующей библиотеки, доступной всем разработчикам.
   - **Сбор мнений и откликов** — результаты анализа методов работы станут основой для повышения эффективности выполнения следующих проектов.
2. Опишите взаимосвязь обсуждения завершенных проектов с моделью зрелости разработки программного обеспечения.

Модель зрелости разработки ПО предлагает структурированное представление соответствующей предметной области, обычно в виде иерархии из пяти уровней. Цель модели — постепенное повышение эффективности организации бизнес-процессов. Каждый из уровней модели суммирует оптимальные методы работы на данном этапе и образует основу для их качественного роста, необходимого для перехода на следующий уровень.

Обзор проекта играет важную роль в переходе организации на следующий уровень модели, помогая выявить и устранить проблемы, присущие организациям данного уровня. Главные задачи организаций, уровень развития которых — вторая ступень модели зрелости, — точное планирование проектов и эффективное отслеживание хода их выполнения. Обсуждение проекта позволяет организациям сопоставить планы с их фактическим выполнением, а значит, повысить эффективность планирования и управления проектами. На четвертом и пятом уровнях модели основная задача рецензирования — сбор мнений и откликов. Выявление проблем завершенного проекта в ходе его обсуждения позволит изменить процессы так, чтобы снизить или полностью исключить влияние этих проблем на следующие проекты.

3. Как на практике организовать обсуждение проекта?

Основные вопросы, которые следует учесть при планировании обзорной встречи, таковы;

- время;
- форма;
- продолжительность;
- организация;
- участники.

4. Опишите способы рецензирования больших проектов.

Планируя обсуждение проекта, менеджер программы может создать специальную группу. Этот метод особенно хорош в больших проектах, когда проектная группа на самом деле состоит из нескольких больших групп.

Существуют разные способы подбора обзорной группы. Можно создать дисциплинарные подгруппы для обсуждения разработки, тестирования и других отдельных составляющих проекта. С другой стороны, ничем не хуже и смешанные подгруппы, члены которой работали над разными этапами проекта.

Предметный указатель

Ж

З

И

К

M

H

O

Р

C

T

Библиографический список

1. Administering Systems Management Server 2.0 (Microsoft, 1998).
2. *William Brown, Raphael Malveau, Hays McCormick III, Thomas Mowbra.* AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis (John Wiley and Sons, 1998).
3. *Mark C Paulk, Charles V Weber, Bill Curtis, Mary Beth Chrissis.* The Capability Maturity Model: Guidelines for Improving the Software Process (Addison-Wesley, 1995).
4. *Alan Pope.* The CORBA Reference Guide (Addison-Wesley, 1998).
5. *Laurie Litwack.* Creating a Vision for Your Product (Microsoft).
6. *Steve Maguire.* Debugging the Development Process (Microsoft Press, 1994).
7. *Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.* Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley, 1995).
8. *Mary Kirtland.* Designing Component-Based Applications (Microsoft Press, 1998).
9. Desktop Applications for Microsoft Visual Basic 6.0 (Microsoft Press. 1999).
10. Desktop Applications for Microsoft Visual C++ 6.0 (Microsoft Press, 1999).
11. *Jim McCarthy.* Dynamics of Software Development (Microsoft Press, 1995).
12. *Dempsey, Dvorak, and Meehan.* «Escaping the IT Abyss», The McKinsey Quarterly, N.4 (1997).
13. *Guy and Henry Eddon.* Inside COM+ (Microsoft Press, 1999).
14. *Guy and Henry Eddon.* Inside Distributed COM (Microsoft Press, 1998).
15. *Mark Bieter.* Internet Information Server Security Overview (Microsoft).
16. Mastering Enterprise Development Using Microsoft Visual Basic 6, Microsoft Mastering Series Group (Microsoft, 1998).
17. *Dr Joyce Statz.* Microsoft Solutions Framework and the Capability Maturity Model (Microsoft/TeraQuest, 1999).
18. Microsoft Visual Basic 6.0 Programmer's Guide (Microsoft Press, 1998).
19. Microsoft Visual C++ 6.0 Programmer's Guide (Microsoft Press, 1998),
20. Microsoft Windows NT 4.0 Security, Audit, and Control (Microsoft Press, 1998).
21. Microsoft Windows NT Workstation Resource Kit (Microsoft Press, 1996).
22. *Christopher Alexander, Sara Ishikawa, and Murray Silverstein.* A Pattern Language: Towns, Buildings, Construction (Oxford University Press, 1977).

23. *James Coplien.* Pattern Languages of Program Design (Addison-Wesley, 1995).
24. *Brad Appleton.* Patterns and Software: Essential Concepts and Terminology (*www.enteract.com/~bradapp/docs/patterns-intro.html*, 1997).
25. *Steve McConnell.* Rapid Development: Taming Wild Software Schedules (Microsoft Press, 1996).
26. *Kamaljit Bath.* Remote Data Service in MDAC 2.0 (Microsoft).
27. The SAP DCOM Component Connector (Microsoft, 1997).
28. Secure Networking Using Windows 2000 Distributed Security Services, Microsoft PBS Web Team (Microsoft).
29. *Maty Shaw.* Software Architecture: Perspectives on an Emerging Discipline (Prentice Hall, 1996).
30. *Barry Boehm.* Software Engineering Economics (Prentice Hall, 1981).
31. *Walker Royce.* Software Project Management: A Unified Framework (Addison-Wesley, 1998).
32. *Steve McConneli.* Software Project Survival Guide (Microsoft Press. 1998).
33. *Ron Higuera, YacovHaimes.* Software Risk Management (Software Engineering Institute).
34. *Adele Goldberg, Kenneth S, Rubin.* Succeeding with Objects (Addison-Wesley, 1995).
35. Technical Overview: Clustering and Windows NT Load Balancing Services (Microsoft, 1998).
36. *Dirk Riehle, Heinz Zullighoven.* Theory and Practice of Object Systems (John Wiley and Sons).
37. *ChristopherAlexander.* The Timeless Way of Building (Oxford University Press, 1970).
38. *An Jaaksi, Juha-MarkusAalto, Ari Aalto, Kimmo Vatto.* Tried and True Object Development: Practical Approaches with UML (Cambridge University Press, 1999).
39. *Grady Booch, James Rumbaugh, Ivar Jacobson.* The Unified Modeling Language User Guide (Addison-Wesley, 1998).
40. *Ivar Jacobson, Grady Booch, James Rumbaugh.* The Unified Software Development Process (Addison-Wesley, 1999).

ЛИЦЕНЗИОННОЕ СОГЛАШЕНИЕ MICROSOFT

прилагаемый к книге компакт-диск

ЭТО ВАЖНО - ПРОЧИТАЙТЕ ВНИМАТЕЛЬНО. Настоящее лицензионное соглашение (далее «Соглашение») является юридическим документом, оно заключается между Вами (физическим или юридическим лицом) и Microsoft Corporation (далее «корпорация Microsoft») на указанный выше продукт Microsoft, который включает программное обеспечение и может включать сопутствующие мультимедийные и печатные материалы, а также электронную документацию (далее «Программный Продукт»). Любой компонент, входящий в Программный Продукт, который сопровождается отдельным Соглашением, подпадает под действие именно того Соглашения, а не условий, изложенных ниже. Установка, копирование или иное использование данного Программного Продукта означает принятие Вами данного Соглашения. Если Вы не принимаете его условия, то не имеете права устанавливать, копировать или как-то иначе использовать этот Программный Продукт.

ЛИЦЕНЗИЯ НА ПРОГРАММНЫЙ ПРОДУКТ

Программный Продукт защищен законами Соединенных Штатов по авторскому праву и международными договорами по авторскому праву, а также другими законами и договорами по правам на интеллектуальную собственность.

1. **ОБЪЕМ ЛИЦЕНЗИИ.** Настоящее Соглашение дает Вам право:
   a) **Программный продукт.** Вы можете установить и использовать одну копию Программного Продукта на одном компьютере. Основной пользователь компьютера, на котором установлен данный Программный Продукт, может сделать только для себя вторую копию и использовать ее на портативном компьютере.
   b) **Хранение или использование в** сети. Вы можете также скопировать или установить экземпляр Программного Продукта на устройстве хранения, например па сетевом сервере, исключительно для установки или запуска данного Программного Продукта на других компьютерах в своей внутренней сети, но тогда Вы должны приобрести лицензии на каждый такой компьютер. Лицензию на данный Программный продукт нельзя использовать совместно или одновременно на других компьютерах.
   c) **License Pak.** Если Вы купили эту лицензию о составе Microsoft License Pak, можете сделать ряд дополнительных копий программного обеспечения, входящего в данный Программный Продукт, и использовать каждую копию так, как било описано выше. Кроме того, Вы получаете право сделать соответствующее число вторичных копий для портативного компьютера в целях, также оговоренных выше.
   d) **Примеры кода.** Это относится исключительно к отельным частям Программного Продукта, заявленным как примеры кода (далее «Примеры»), если таковые входят в состав Программного Продукта.
      i) **Использование и модификация.** Microsoft дает Вам право использовать и модифицировать исходный кол Примеров при условии соблюдении пункта (d)(iii) ниже. Вы не имеете права распространять в виде исходного кола ни Примеры, ни их модифицированную версию.
      ii) **Распространяемые файлы.** При соблюдении пункта (d)(iii) Microsoft дает Вам право на свободное от отчислений копирование и распространение в виде объектного кода Примеров или их модифицированной версии, кроме тех частей (или их модифицированных версий), которые оговорены в файле Readme, относящемся к данному Программному Продукту, как не подлежащие распространению.
      iii) **Требования к распространению файлов.** Вы можете распространять файлы, разрешенные к распространению, при условии, что: а) распространяете их в виде объектного кода только в сочетании со своим приложением и как его часть; б) не используете название, эмблему или товарные знаки Microsoft для продвижения своего приложения; в) включаете имеющуюся в Программном Продукте ссылку на авторские права в состав этикетки и заставки своего приложения; г) согласны освободить от ответственности и взять на себя защиту корпорации Microsoft от любых претензий или преследований по закону, включая судебные издержки, если таковые возникнут в результате использования или распространения Вашего приложения; ид) не допускаете дальнейшего распространения конечным пользователем своего приложения. По поводу отчислений и других условий лицензии применительно к иным видам использования или распространения распространяемых файлов обращайтесь в Microsoft.

2. **ПРОЧИЕ ПРАВА И ОГРАНИЧЕНИЯ**
   - **Ограничения на реконструкцию, декомпиляцию и дизассемблирование.** Вы не имеете права реконструировать, декомпилировать или дизассемблировать данный Программный Продукт, кроме того случая, когда такая деятельность (только в той мере, которая необходима) явно разрешается соответствующим законом, несмотря на это ограничение.

- **Разделение компонентов.** Данный Программный Продукт лицензируется как единый продукт. Его компоненты нельзя отделять друг от друга для использования более чем на одном компьютере.
- **Аренда.** Данный Программный Продукт нельзя сдавать в прокат, передавать во временное пользование или уступать для использования в иных целях.
- **Услуги по технической поддержке.** Microsoft может (но не обязана) предоставить Вам услуги по технической поддержке данного Программного Продукта (далее «Услуги»). Предоставление Услуг регулируется соответствующими правилами и программами Microsoft, описанными в руководстве пользователя, электронной документации и/или других материалах, публикуемых Microsoft. Любой дополнительный программный код, предоставленный в рамках Услуг, следует считать частью данного Программного Продукта и подпадающим под действие настоящего Соглашения. Что касается технической информации, предоставляемой Вами корпорации Microsoft при использовании ее Услуг, то Microsoft может задействовать эту информацию в деловых целях, в том числе для технической поддержки продукта и разработки. Используя такую техническую информацию, Microsoft не будет ссылаться на Вас,
- **Передача прав на программное** обеспечение. Вы можете безвозвратно уступить все права, регулируемые настоящим Соглашением, при условии, что не оставите себе никаких копии, передадите все составные части данного Программного Продукта (включая компоненты, мультимедийные и печатные материалы, любые обновления, Соглашение и сертификат подлинности, если таковой имеется) и принимающая сторона согласится с условиями настоящего Соглашения.
- **Прекращение действия Соглашения.** Без ущерба для любых других прав Microsoft может прекратить действие настоящего Соглашения, если Вы нарушите его условия. В этом случае Вы должны будете уничтожить все копии данного Программного Продукта вместе со всеми его компонентами.

3. АВТОРСКОЕ ПРАВО. Все авторские права и право собственности на Программный Продукт (в том числе любые изображения, фотографии, анимации, видео, аудио, музыку, текст, примеры кода, распространяемые файлы и апплеты, включенные в состав Программного Продукта) и любые его копии принадлежат корпорации Microsoft или ее поставщикам. Программный Продукт охраняется законолательством об авторских правах и положениями международных договоров. Таким образом, Вы должны обращаться с данным Программным Продуктом, как с любым другим материалом, охраняемым авторскими правами, с тем **исключением**, что Вы можете установить Программный Продукт на один компьютер три *условна, что* храните оригинал исключительно как резервную или архивную копию. Копирование печатных материалов, поставляемых вместе с Программным Продуктом, запрещается.

ОГРАНИЧЕНИЕ ГАРАНТИИ

ДАННЫЙ ПРОГРАММНЫЙ ПРОДУКТ (ВКЛЮЧАЯ ИНСТРУКЦИИ ПО ЕГО ИСПОЛЬЗОВАНИЮ) ПРЕДОСТАВЛЯЕТСЯ БЕЗ КАКОЙ-ЛИБО ГАРАНТИИ. КОРПОРАЦИЯ MICROSOFT СНИМАЕТ С СЕБЯ ЛЮБУЮ ВОЗМОЖНУЮ ОТВЕТСТВЕННОСТЬ, В ТОМ ЧИСЛЕ ОТВЕТСТВЕННОСТЬ ЗА КОММЕРЧЕСКУЮ ЦЕННОСТЬ ИЛИ СООТВЕТСТВИЕ ОПРЕДЕЛЕННЫМ ЦЕЛЯМ. ВЕСЬ РИСК ПО ИСПОЛЬЗОВАНИЮ ИЛИ РАБОТЕ С ПРОГРАММНЫМ ПРОДУКТОМ ЛОЖИТСЯ НА ВАС.

НИ ПРИ КАКИХ ОБСТОЯТЕЛЬСТВАХ КОРПОРАЦИЯ MICROSOFT, ЕЕ РАЗРАБОТЧИКИ, А ТАКЖЕ ВСЕ, ЗАНЯТЫЕ В СОЗДАНИИ, ПРОИЗВОДСТВЕ И РАСПРОСТРАНЕНИИ ДАННОГО ПРОГРАММНОГО ПРОДУКТА, НЕ НЕСУТ ОТВЕТСТВЕННОСТИ ЗА КАКОЙ-ЛИБО УЩЕРБ (ВКЛЮЧАЯ ВСЕ. БЕЗ ИСКЛЮЧЕНИЯ, СЛУЧАИ УПУЩЕННОЙ ВЫГОДЫ, НАРУШЕНИЯ ХОЗЯЙСТВЕННОЙ ДЕЯТЕЛЬНОСТИ, ПОТЕРИ ИНФОРМАЦИИ ИЛИ ДРУГИХ УБЫТКОВ) ВСЛЕДСТВИЕ ИСПОЛЬЗОВАНИЯ ИЛИ НЕВОЗМОЖНОСТИ ИСПОЛЬЗОВАНИЯ ДАННОГО ПРОГРАММНОГО ПРОДУКТА ИЛИ ДОКУМЕНТАЦИИ. ДАЖЕ ЕСЛИ КОРПОРАЦИЯ MICROSOFT БЫЛА ИЗВЕЩЕНА О ВОЗМОЖНОСТИ ТАКИХ ПОТЕРЬ, ТАК КАК В НЕКОТОРЫХ СТРАНАХ НЕ РАЗРЕШЕНО ИСКЛЮЧЕНИЕ ИЛИ ОГРАНИЧЕНИЕ ОТВЕТСТВЕННОСТИ ЗА НЕПРЕДНАМЕРЕННЫЙ УЩЕРБ, УКАЗАННОЕ ОГРАНИЧЕНИЕ МОЖЕТ ВАС НЕ КОСНУТЬСЯ.

РАЗНОЕ

Настоящее Соглашение регулируется законодательством штата Вашингтон (США), кроме случаев (и лишь в той мере, насколько это необходимо) исключительной юрисдикции того государства, на территории которого используется Программный Продукт.

Если у Вас возникли какие-либо вопросы, касающиеся настоящего Соглашения, или если Вы желаете связаться с Microsoft по любой другой причине, пожалуйста, обращайтесь в местное представительство Microsoft или пишите по адресу: Microsoft Sales Information Center, One Microsoft Way, Redmond, WA 98052-6399.

Microsoft Corporation

Принципы проектирования и разработки программного обеспечения

Учебный курс MCSD

2-е издание, исправленное

Перевод с английского под общей редакцией **А. В. Кузьмина**
Переводчики **А. Е. Долганов, М. Е. Кондрашова**
Консультант Д. **З. Вибе**

Редакторы **Ю. П. Леонова, С. В. Дергачев**

Компьютерный дизайн и подготовка иллюстраций **Д. В. Петухов**

Технический редактор **Н. Г. Тимченко**

Дизайнер обложки **Е. В. Козлова**

ParaType TypeMarketFontLibrary легальный пользователь

Оригинал-макет выполнен с использованием издательской системы Adobe PageMaker 6.0

Главный редактор **А. И. Козлов**

Подготовлено к печати издательско-торговым домом «Русская Редакция»

РУССКАЯ РЕДАКЦИЯ

Лицензия ЛР № 066 422 от 19.03.99 г.
Подписано в печать 22.04.2002 г. *Физ.* п. л. 46.
Формат 60x90 1/16 Тираж 1500 экз.

Отпечатано в ОАО «Типография "Новости"»
107005, Москва, ул. Фр. Энгельса, 46

Книги Microsoft Press на русском языке по программам сертификации Microsoft

Сертификационный экзамен	Издания, необходимые для подготовки к экзамену	Программа сертификации
№70-016 Designing and Implementing Desktop Applications with Microsoft Visual C++ 6.0	Разработка приложений на Microsoft Visual C++ 6.0. Учебный курс MCSD 2-е изд. ISBN 5-7502-0185-6, 704 стр., +CD, 2001 г.	Экзамен MCSD по выбору
№ 70-029 Designing and Implementing Databases with Microsoft SQL Server 7.0	Реализация баз данных Microsoft SQL Server 7.0. Учебный курс MCSE. ISBN 5-7502-0116-3. 528 стр., +CD, 2000 г.	Экзамен MCSE по выбору
№ 70-059 Internetworking with Microsoft TCP/IP on Microsoft Windows NT 4.0	Microsoft TCP/IP. Учебный курс MCSE 3-е изд. ISBN 5-7502-0171-6, 344 стр., +CD. 2001 г.	Экзамен MCSE по выбору
№ 70-100 Analyzing Requirements and Defining Solution Architectures	Принципы проектирования и разработки программного обеспечения. Учебный курс MCSD 2-е изд. ISBN 5-7502-0213-5, 736 стр., +CD, 2002 г.	Обязательный экзамен MCSD
№70-175 Designing and Implementing Distributed Applications with Microsoft Visual Basic 6.0	Разработка распределенных приложений на Microsoft Visual Basic 6.0. Учебный курс MCSD. ISBN 5-7502-0133-3, 400 стр., +CD, 2000 г.	Экзамен MCSD по выбору
№70-210 installing, Configuring, and Administering Microsoft Windows 2000 Professional	Microsoft Windows 2000 Professional. Учебный курс MCSE 2-е изд. ISBN 5-7502-0183-X, 672 стр., 2001 г.	Обязательный экзамен MCSE/MCSA
№ 70-215 Installing. Configuring, and Administering Microsoft Windows 2000 Server	Microsoft Window» 2000 Server. Учебный курс MCSE 2-е изд. ISBN 5-7502-0182-1, 912 стр., 2001 г.	Обязательный экзамен MCSE/MCSA
№70-216 implementing and Administering a Microsoft Windows 2000 Network Infrastructure	Администрирование сети на основе Microsoft Windows 2000. Учебный курс MCSE. ISBN 5-7502-0164-3, 512 стр., 2001 г.	Обязательный экзамен MCSE/MCSA
№ 70-217 Implementing and Administering a Microsoft Windows 2000 Directory Services Infrastructure	Microsoft Windows 2000 Active Directory Services. Учебный курс MCSE. ISBN 5-7502-0139-2, 800 стр., 2001 г.	Обязательный экзамен MCSE
№70-210, №70-215, № 70-216, №70-217	MCSE Windows 2000. Компакт-диск: учебные материалы для подготовки к сертификационным экзаменам №№ 70-210, 70-215, 70-216, 70-217, 2-е изд., ISBN 5-7502-0197-X, 16 стр., +CD, 2001 г.	

Книги Microsoft Press на русском языке по программам сертификации Microsoft

Сертификационный экзамен	Издания, необходимые для подготовки к экзамену	Программа сертификации
№ 70-218 Managing a Microsoft Windows 2000 Network Environment	Управление сетевой средой Microsoft Windows 2000. Учебный курс MCSA. ISBN 5-7502-0212-7, готовится к изданию	Обязательный экзамен MCSA
№ 70-220 Designing Security Tor a Microsoft Windows 2000 Network	Безопасность сети на основе Microsoft Windows 2000. Учебный курс MCSE. ISBN 5-7502-0176-7, 912 стр., +CD, 2001 г.	Обязательный экзамен MCSE по выбору
№ 70-221 Designing a Microsoft Windows 2000 Network Infrastructure	Разработка инфраструктуры сетевых служб Microsoft Windows 2000. Учебный курс MCSE. ISBN 5-7502-0192-9, 992 стр.. +CD, 2001 г.	Обязательный экзамен MCSE по выбору
№ 70-226 Designing Highly Available Web Solutions with Microsoft Windows 2000 Server Technologies	Web-разработка на платформе Microsoft Windows 2000 Server. Учебный курс MCSE. ISBN 5-7502-0196-1, готовится к изданию	Обязательный экзамен MCSE по выбору
№ 70-227 Installing, Configuring, and Administering Microsoft Internet Security and Acceleration (ISA) Server 2000	Microsoft Internet Security and Acceleration Server 2000. Учебный курс MCSE. ISBN 5-7502-0191-0, 544 стр., +CD, 2002 г.	Экзамен MCSE по выбору
№ 70-228 Installing. Configuring, and Administering Microsoft SQL Server 2000 Enterprise Edition	Администрирование Microsoft SQL Server 2000. Учебный курс MCSE. ISBN 5-7502-0202-X, готовится к изданию	Экзамен MCSE по выбору
№ 70-229 Designing and Implementing Databases with Microsoft SQL Server 2000 Enterprise Edition	Проектирование и реализация баз данных Microsoft SQL Server 2000. Учебный курс MCSE. ISBN 5-7502-0149-X, 652 стр., +CD, 2001 г.	Экзамен MCSE по выбору
№ 70-270 Installing, Configuring, and Administering Microsoft Windows XP Professional	Microsoft Windows XP Professional Учебный курс MCSE. ISBN 5-7502-0209-7, готовится к изданию	Обязательный экзамен MCSE
A+ Certification	A+ Сертификация. Учебный курс. ISBN 5-7502-0115-5, 496 стр., 2000 г.	Экзамен CompTIA
Network+ Certification	Компьютерные сети. Сертификация Network+. ISBN 5-7502-0190-2, 704 стр.. + CD. 2002 г.	Экзамен CompTIA

издательство компьютерной литературы
РУССКАЯ РЕДАКЦИЯ

ПРОДАЖА КНИГ
оптом тел.: (095) 142-0571. e-mail: alexg@rusedit.ru;
интернет-магазин http://www.ITbook.ru; тел.: (095) 145-1519;
в розницу магазин «КОМПЬЮТЕРНАЯ и ДЕЛОВАЯ КНИГА»
Москва. Ленинский пр-т. стр. 38. тел.: (095) 778-7269