Электронный учебно-методический комплекс

ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Учебная программа дисциплины

➤ **Учебное пособие**

Лабораторный практикум

Учебное пособие по курсовому проектированию

Методические указания по самостоятельной работе

Банк тестовых заданий в системе UniTest

Красноярск
ИПК СФУ
2008

УДК 004.3
ББК 32.973
Я49

Электронный учебно-методический комплекс по дисциплине «Технологии разработки программного обеспечения» подготовлен в рамках инновационной образовательной программы «Инновационно-образовательный центр технологий поддержки жизненного цикла и качества продукта», реализованной в ФГОУ ВПО СФУ в 2007 г.

Рецензенты:
Красноярский краевой фонд науки;
Экспертная комиссия СФУ по подготовке учебно-методических комплексов дисциплин

Якунин, Ю. Ю.

Я49 Технологии разработки программного обеспечения. Версия 1.0 [Электронный ресурс] : электрон. учеб. пособие / Ю. Ю. Якунин. – Электрон. дан. (3 Мб). – Красноярск : ИПК СФУ, 2008. – (Технологии разработки программного обеспечения : УМКД № 183-2007 / рук. творч. коллектива Ю. Ю. Якунин). – 1 электрон. опт. диск (*DVD*). – Систем. требования : *Intel Pentium* (или аналогичный процессор других производителей) 1 ГГц ; 512 Мб оперативной памяти ; 3 Мб свободного дискового пространства ; привод *DVD* ; операционная система *Microsoft Windows* 2000 *SP* 4 / *XP SP* 2 / *Vista* (32 бит) ; *Adobe Reader* 7.0 (или аналогичный продукт для чтения файлов формата *pdf*).

ISBN 978-5-7638-0975-6 (комплекса)
ISBN 978-5-7638-1435-4 (пособия)
Номер гос. регистрации в ФГУП НТЦ «Информрегистр» 0320802414 от 21.11.2008 г. (комплекса)

Настоящее издание является частью электронного учебно-методического комплекса по дисциплине «Технологии разработки программного обеспечения», включающего учебную программу, учебное пособие по курсовому проектированию, лабораторный практикум, методические указания по самостоятельной работе, контрольно-измерительные материалы «Технологии разработки программного обеспечения. Банк текстовых заданий», наглядное пособие «Технологии разработки программного обеспечения. Презентационные материалы».

Рассмотрены технологии, методики и стандарты, обеспечивающие процесс разработки программных систем. Особое внимание уделено новой технологии, ориентированной на архитектуру, управляемую моделью (MDA), и языкам модельной разработки (UML и OCL).

Предназначено для студентов направления 230100.62 «Информатика и вычислительная техника» укрупненной группы 230000 «Вычислительная техника и информационные технологии».

Рекомендовано Инновационно-методическим управлением СФУ
в качестве учебного пособия

Редактор Н. Н. Вохман

Разработка и оформление электронного образовательного ресурса: Центр технологий электронного обучения информационно-аналитического департамента СФУ; лаборатория по разработке мультимедийных электронных образовательных ресурсов при КрЦНИТ

Подп. к использованию 01.10.2008
Объем 3 Мб
Красноярск: СФУ, 660041, Красноярск, пр. Свободный, 79

Оглавление

ВВЕДЕНИЕ

Процесс современной разработки программного обеспечения ориентирован на жизненный цикл программного продукта. Все существующие в настоящее время технологии, методики и стандарты напрямую или косвенно касаются или регламентируют этапы жизненного цикла как по функциональному наполнению, так и по содержанию.

Процесс разработки программных систем тесно связан с областью управления проектами, потому что любой программный продукт является уникальным результатом. От организации этого процесса напрямую зависят основные характеристики выполнения программного проекта – сроки выполнения, запланированный бюджет, качество выпускаемого продукта.

Но профессиональное управление проектами само по себе не может обеспечить достижение указанных характеристик. Немаловажную роль в этом играет архитектура программной системы, опыт и квалификация участников команды разработки, а также правильное документирование всех процессов разработки программного обеспечения.

В данном курсе уделяется большое внимание российским стандартам, регламентирующим организацию процесса разработки и документирование отдельных этапов. Раскрывается понятие архитектуры программной системы и ее важность. Рассматривается методика разработки, ориентированная на архитектуру, управляемую моделью (MDA), которая раскрывает практическую значимость этого подхода.

1. ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1.1. Основные этапы развития технологии разработки

Технологией программирования называют совокупность методов и средств, используемых в процессе разработки программного обеспечения. Как любая другая технология, технология программирования представляет собой набор технологических инструкций, включающих:

указание последовательности выполнения технологических операций;

перечисление условий, при которых выполняется та или иная операция;

описания самих операций, где для каждой операции определены исходные данные, результаты, а также инструкции, нормативы, стандарты, критерии, методы оценки и т. п. (см. рис. 1.1).

Рис. 1.1. Структура описания технологической операции

Кроме набора операций и их последовательности, технология также определяет способ описания проектируемой системы, точнее модели, используемой на конкретном этапе разработки.

Различают технологии, используемые на конкретных этапах разработки или для решения отдельных задач этих этапов, и технологии, охватывающие несколько этапов или весь процесс разработки. В основе первых, как правило, лежит ограниченно применимый *метод,* позволяющий решить конкретную задачу. В основе вторых – базовый метод или *подход,* определяющий совокупность методов, используемых на разных этапах разработки, или проектируемой системы, точнее модели, используемой на конкретном этапе разработки.

Чтобы разобраться в существующих технологиях программирования и определить основные тенденции их развития, целесообразно рассматривать эти технологии в историческом контексте, выделяя основные этапы развития программирования как науки.

1.1.1. Этап 1. «Стихийное» программирование

Этот этап охватывает период от момента появления первых вычислительных машин до середины 60-х гг. XX в. В это время практически отсутствовали технологии разработки программного обеспечения и программирование фактически было искусством. Первые программы имели простейшую структуру. Они состояли из собственно программы на машинном языке и обрабатываемых ею данных (рис. 1.2). Сложность программ в машинных кодах ограничивалась способностью программиста одновременно мысленно отслеживать последовательность выполняемых операций и местонахождение данных при программировании.

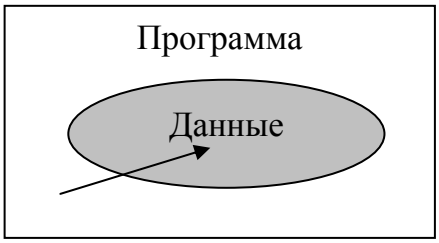


Рис. 1.2. Структура первых программ

Появление ассемблеров позволило вместо двоичных или шестнадцатеричных кодов использовать символические имена данных и мнемоники кодов операций. В результате программы стали более «читаемыми».

Создание языков программирования высокого уровня, таких как FORTRAN и ALGOL, существенно упростило программирование вычислений, снизив уровень детализации операций. Это, в свою очередь, позволило увеличить сложность программ.

Революционным было появление в языках средств, позволяющих оперировать подпрограммами. (Идея написания подпрограмм появилась гораздо раньше, но отсутствие средств поддержки в первых языковых средствах существенно снижало эффективность их применения.) Подпрограммы можно было сохранять и использовать в других программах. В результате были созданы огромные библиотеки расчетных и служебных подпрограмм, которые по мере надобности вызывались из разрабатываемой программы.

Типичная программа того времени состояла из основной программы, области глобальных данных и набора подпрограмм (в основном библиотечных), выполняющих обработку всех данных или их части (рис. 1.3).

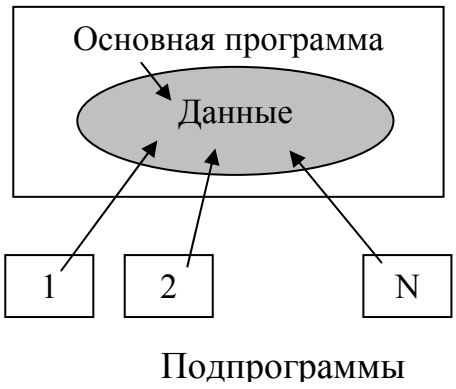


Рис. 1.3. Принцип работы программ с глобальной областью данных

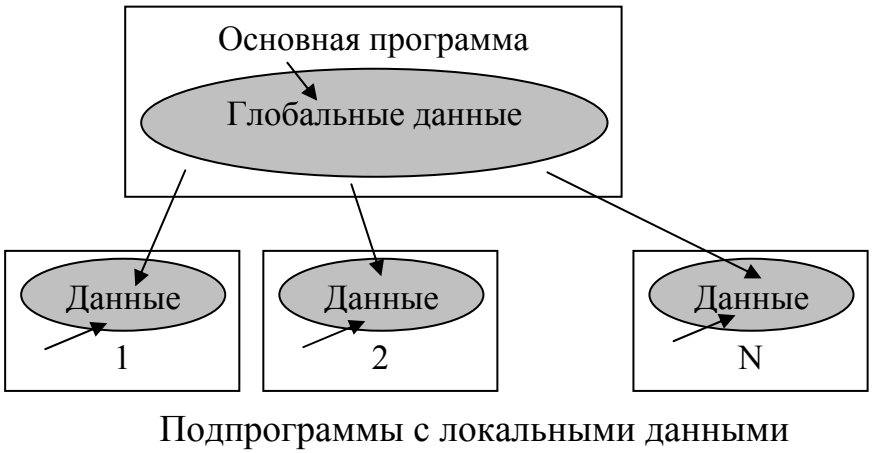


Рис. 1.4. Принцип работы программы,
использующей подпрограммы с локальными данными

Слабым местом такой архитектуры было то, что при увеличении количества подпрограмм возрастала вероятность искажения части глобальных данных какой-либо подпрограммой. Например, подпрограмма поиска корней уравнения на заданном интервале по методу деления отрезка пополам меняет величину интервала. Если при выходе из подпрограммы не предусмотреть восстановления первоначального интервала, то в глобальной области окажется неверное значение интервала. Чтобы сократить количество таких ошибок, было предложено в подпрограммах размещать локальные данные (рис. 1.4).

Сложность разрабатываемого программного обеспечения при использовании подпрограмм с локальными данными по-прежнему ограничивалась возможностью программиста отслеживать процессы обработки данных, но уже на новом уровне. Однако появление средств поддержки подпрограмм позволило осуществлять разработку программного обеспечения нескольким программистам параллельно.

В начале 60-х гг. XX в. разразился «кризис программирования». Он выражался в том, что фирмы, взявшиеся за разработку сложного программного обеспечения такого, как операционные системы, срывали все сроки за-

вершения проектов. Проект устаревал раньше, чем был готов к внедрению, увеличивалась его стоимость, и в результате многие проекты так никогда и не были завершены.

Объективно все это было вызвано несовершенством технологии программирования. Прежде всего, стихийно использовалась разработка «снизу-вверх» – подход, при котором вначале проектировали и реализовывали сравнительно простые подпрограммы, из которых затем пытались построить сложную программу. В отсутствии четких моделей описания подпрограмм и методов их проектирования создание каждой подпрограммы превращалось в непростую задачу, интерфейсы подпрограмм получались сложными, и при сборке программного продукта выявлялось большое количество ошибок согласования. Исправление таких ошибок, как правило, требовало серьезного изменения уже разработанных подпрограмм, что еще более осложняло ситуацию, так как при этом в программу часто вносились новые ошибки, которые также необходимо было исправлять... В конечном итоге процесс тестирования и отладки программ занимал более 80 % времени разработки, если вообще когда-нибудь заканчивался. На повестке дня самым серьезным образом стоял вопрос разработки технологии создания сложных программных продуктов, снижающей вероятность ошибок проектирования.

Анализ причин возникновения большинства ошибок позволил сформулировать новый подход к программированию, который был назван «структурным».

1.1.2. Этап 2. Структурный подход к программированию (60–70-е гг. XX в.)

Структурный подход к программированию представляет собой совокупность рекомендуемых технологических приемов, охватывающих выполнение всех этапов разработки программного обеспечения. В основе структурного подхода лежит декомпозиция (разбиение на части) сложных систем с целью последующей реализации в виде отдельных небольших подпрограмм. С появлением других принципов декомпозиции (объектного, логического и т. д.) данный способ получил название «процедурной декомпозиции».

В отличие от используемого ранее процедурного подхода к декомпозиции структурный подход требовал представления задачи в виде иерархии подзадач простейшей структуры. Проектирование, таким образом, осуществлялось «сверху-вниз» и подразумевало реализацию общей идеи, обеспечивая проработку интерфейсов подпрограмм. Одновременно вводились ограничения на конструкции алгоритмов, рекомендовались формальные модели их

описания, а также специальный метод проектирования алгоритмов – метод пошаговой детализации.

Поддержка принципов структурного программирования была заложена в основу так называемых процедурных языков программирования. Как правило, они включали основные «структурные» операторы передачи управления, поддерживали вложение подпрограмм, локализацию и ограничение области «видимости» данных. Среди наиболее известных языков этой группы стоит назвать PL/1, ALGOL-68, Pascal, C.

Одновременно со структурным программированием появилось огромное количество языков, базирующихся на других концепциях, но большинство из них не выдержало конкуренции. Какие-то языки были просто забыты, идеи других были в дальнейшем использованы в следующих версиях развиваемых языков.

Дальнейший рост сложности и размеров разрабатываемого программного обеспечения потребовал развития структурирования данных. Как следствие этого в языках появляется возможность определения пользовательских типов данных. Одновременно усилилось стремление разграничить доступ к глобальным данным программы, чтобы уменьшить количество ошибок, возникающих при работе с глобальными данными. В результате появилась и начала развиваться технология модульного программирования.

Модульное программирование предполагает выделение групп подпрограмм, использующих одни и те же глобальные данные в отдельно компилируемые модули (библиотеки подпрограмм), например, модуль графических ресурсов, модуль подпрограмм вывода на принтер (рис. 1.5). Связи между модулями при использовании данной технологии осуществляются через специальный интерфейс, в то время как доступ к реализации модуля (телам подпрограмм и некоторым «внутренним» переменным) запрещен. Эту технологию поддерживают современные версии языков Pascal и C (C++), языки Ада и Modula.

Использование модульного программирования существенно упростило разработку программного обеспечения несколькими программистами. Теперь каждый из них мог разрабатывать свои модули независимо, обеспечивая взаимодействие модулей через специально оговоренные межмодульные интерфейсы. Кроме того, модули в дальнейшем без изменений можно было использовать в других разработках, что повысило производительность труда программистов.

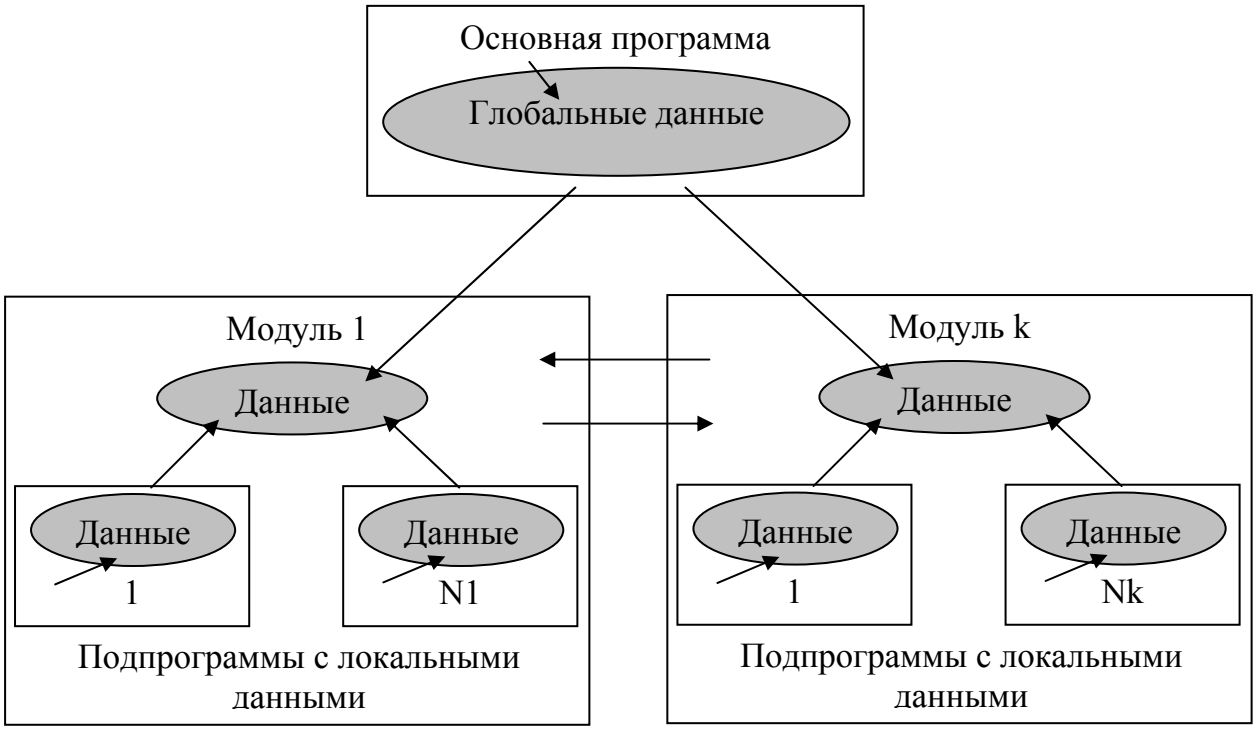


Рис. 1.5. Модульная структура программ

Практика показала, что структурный подход в сочетании с модульным программированием позволяет получать достаточно надежные программы, размер которых не превышает 100 000 операторов. Узким местом модульного программирования является то, что ошибка в интерфейсе при вызове подпрограммы выявляется только при выполнении программы (из-за раздельной компиляции модулей обнаружить эти ошибки раньше невозможно). При увеличении размера программы обычно возрастает сложность межмодульных интерфейсов, и с некоторого момента предусмотреть взаимовлияние отдельных частей программы становится практически невозможно. Для разработки программного обеспечения большого объема было предложено использовать объектный подход.

1.1.3. Этап 3. Объектный подход к программированию (с середины 1980-х гг. до нашего времени)

Объектно-ориентированное программирование определяется как технология создания сложного программного обеспечения, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса (рис. 1.6), а классы образуют иерархию с наследованием свойств. Взаимодействие программных объектов в такой системе осуществляется путем передачи сообщений (рис. 1.7).

Объектная структура программы впервые была использована в языке имитационного моделирования сложных систем Simula, появившемся еще в 60-х гг. XX в. Естественный для языков моделирования способ представле-

ния программы получил развитие в другом специализированном языке моделирования – языке Smalltalk (70-е гг. XX в.), а затем был использован в новых версиях универсальных языков программирования, таких как Pascal, C++, Modula, Java.

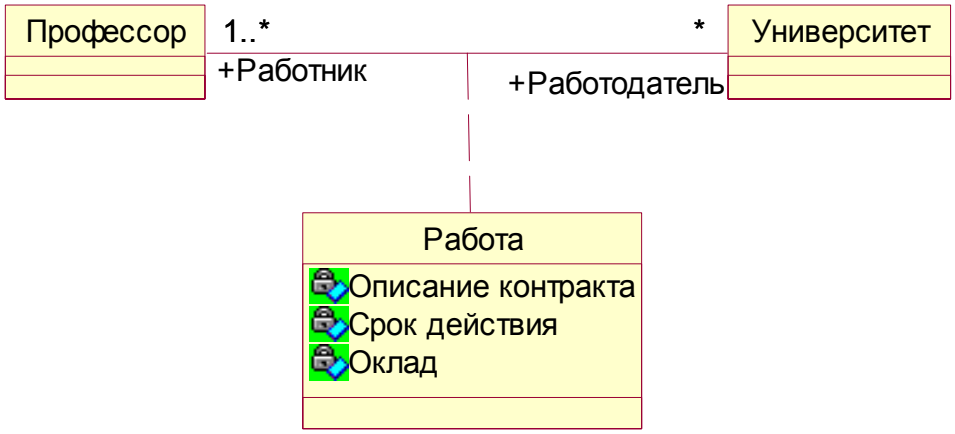


Рис. 1.6. Структура объектно-ориентированной программы в виде связанных классов

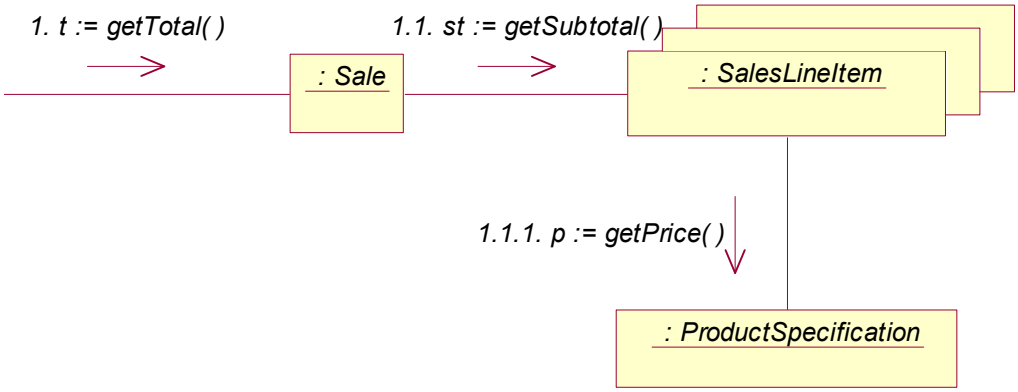


Рис. 1.7. Взаимодействие объектов в объектно-ориентированных программах

Основным достоинством объектно-ориентированного программирования по сравнению с модульным программированием является «более естественная» декомпозиция программного обеспечения, которая существенно облегчает его разработку. Это приводит к более полной локализации данных и интегрированию их с подпрограммами обработки, что позволяет вести практически независимую разработку отдельных частей (объектов) программы. Кроме этого, объектный подход предлагает новые способы организации программ, основанные на механизмах наследования, полиморфизма, композиции, наполнения. Эти механизмы позволяют конструировать сложные объекты из сравнительно простых. В результате существенно увеличивается показатель повторного использования кодов и появляется возможность создания библиотек классов для различных применений.

Бурное развитие технологий программирования, основанных на объектном подходе, позволило решить многие проблемы. Так, были созданы

среды, поддерживающие визуальное программирование, например, Delphi, C++ Builder, Visual C++ и т. д. При использовании визуальной среды у программиста появляется возможность проектировать некоторую часть, например, интерфейсы будущего продукта, с применением визуальных средств добавления и настройки специальных библиотечных компонентов. Результатом визуального проектирования является заготовка будущей программы, в которую уже внесены соответствующие коды.

Использование объектного подхода имеет много преимуществ, однако его конкретная реализация в объектно-ориентированных языках программирования, таких как Pascal и C++, имеет существенные недостатки:
фактически отсутствуют стандарты компоновки двоичных результатов компиляции объектов в единое целое даже в пределах одного языка программирования: компоновка объектов, полученных разными компиляторами C++ в лучшем случае проблематична, что приводит к необходимости разработки программного обеспечения с использованием средств и возможностей одного языка программирования высокого уровня и одного компилятора, а значит, требует наличия исходных кодов используемых библиотек классов;
изменение реализации одного из программных объектов, как минимум, связано с перекомпиляцией соответствующего модуля и перекомпоновкой всего программного обеспечения, использующего данный объект.

Таким образом, при использовании этих языков программирования сохраняется зависимость модулей программного обеспечения от адресов экспортируемых полей и методов, а также структур и форматов данных. Эта зависимость объективна, так как модули должны взаимодействовать между собой, обращаясь к ресурсам друг друга. Связи модулей нельзя разорвать, но можно попробовать стандартизировать их взаимодействие, на чем и основан компонентный подход к программированию.

1.1.4. Этап 4. Компонентный подход и CASE-технологии (с середины 1990-х гг. до нашего времени)

Компонентный подход предполагает построение программного обеспечения из отдельных компонентов – физически отдельно существующих частей программного обеспечения, которые взаимодействуют между собой через стандартизованные двоичные интерфейсы. В отличие от обычных объектов объекты-компоненты можно собрать в динамически вызываемые библиотеки или исполняемые файлы, распространять в двоичном виде (без исходных текстов) и использовать в любом языке программирования, поддерживающем соответствующую технологию. Сегодня рынок объектов стал реальностью: так, в Интернете существуют узлы, предоставляющие большое количество компонентов, рекламой компонентов забиты журналы. Это позволяет программистам создавать продукты, хотя бы частично состоящие из повторно использованных частей, т. е. применять технологию, хорошо зарекомендовавшую себя в области проектирования аппаратуры.

Компонентный подход лежит в основе технологий, разработанных на базе COM (Component Object Model – компонентная модель объектов), и тех-

нологии создания распределенных приложений CORBA (Common Object Request Broker Architecture – общая архитектура с посредником обработки запросов объектов). Эти технологии используют сходные принципы и различаются лишь особенностями их реализации.

Технология COM фирмы Microsoft является развитием технологии OLE (Object Linking and Embedding – связывание и внедрение объектов), которая использовалась в ранних версиях Windows для создания составных документов. Технология COM определяет общую парадигму взаимодействия программ любых типов: библиотек, приложений, операционной системы, т. е. позволяет одной части программного обеспечения использовать функции (*службы*), предоставляемые другой, независимо от того, функционируют ли эти части в пределах одного процесса, в разных процессах на одном компьютере или на разных компьютерах (рис. 1.8). Модификация COM, обеспечивающая передачу вызовов между компьютерами, называется DCOM (Distributed COM – распределенная COM).

По технологии COM приложение предоставляет свои службы, используя специальные объекты – объекты COM, которые являются экземплярами классов COM. Объект COM, так же как обычный объект, включает поля и методы, но в отличие от обычных объектов каждый объект COM может реализовывать несколько интерфейсов, обеспечивающих доступ к его полям и функциям. Это достигается за счет организации отдельной таблицы адресов методов для каждого интерфейса (по типу таблиц виртуальных методов). При этом интерфейс обычно объединяет несколько однотипных функций. Кроме того, классы COM поддерживают наследование интерфейсов, но не поддерживают наследования реализации, т. е. не наследуют код методов, хотя при необходимости объект класса-потомка может вызвать метод родителя.

Каждый интерфейс имеет имя, начинающееся с символа I и глобальный уникальный идентификатор IID (Interface IDentifier). Любой объект COM обязательно реализует интерфейс (Unknown (на схемах этот интерфейс всегда располагают сверху). Использование этого интерфейса позволяет получить доступ к остальным интерфейсам объекта.

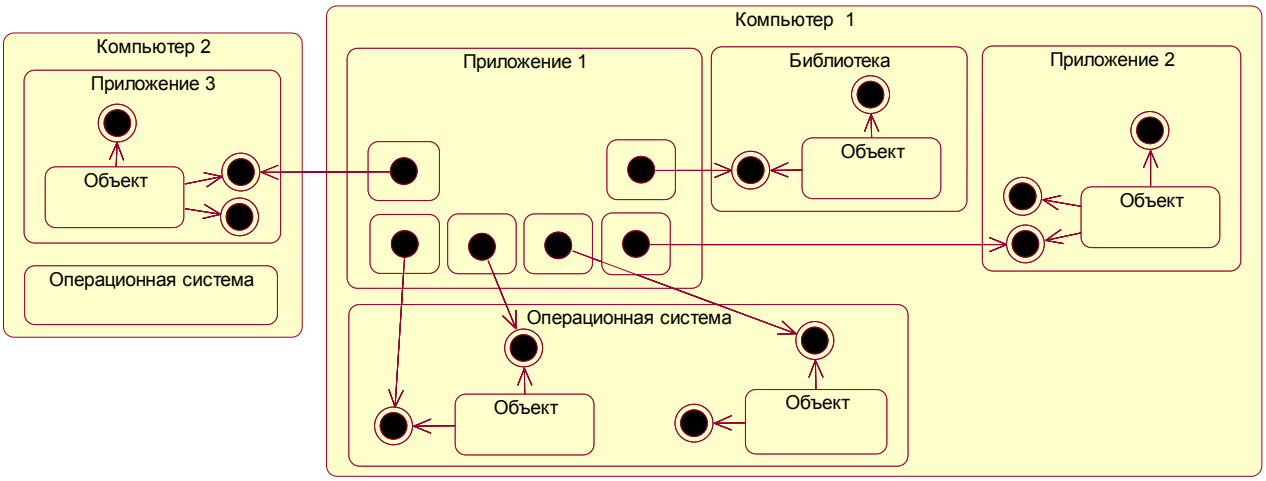


Рис. 1.8. Взаимодействие программных компонентов различных типов

Объект всегда функционирует в составе сервера – динамической библиотеки или исполняемого файла, которые обеспечивают функционирование объекта. Различают три типа серверов:

внутренний сервер: реализуется динамическими библиотеками, которые подключаются к приложению-клиенту и работают в одном с ними адресном пространстве. Это наиболее эффективный сервер, кроме того, он не требует специальных средств;

локальный сервер: создается отдельным процессом (exe), который работает на одном компьютере с клиентом;

удаленный сервер: создается процессом, который работает на другом компьютере.

Например, Microsoft Word является локальным сервером. Он включает множество объектов, которые могут использоваться другими приложениями.

Взаимодействие клиента и сервера обеспечивается базовыми механизмами COM или DCOM, поэтому клиенту безразлично местонахождение объекта. При использовании локальных и удаленных серверов в адресном пространстве клиента создается *proxy-объект* – заместитель объекта COM, а в адресном пространстве сервера COM – *заглушка,* соответствующая клиенту. Получив задание от клиента, заместитель упаковывает его параметры и, используя службы операционной системы, передает вызов заглушке. Заглушка распаковывает задание и передает его объекту COM. Результат возвращается клиенту в обратном порядке.

1.1.5. этап 5. Разработка, ориентированная на архитектуру и CASE-технологии (с начала XXI в. до нашего времени)

До конца XX в. все возможные проектные модели употребляли исключительно для документирования промежуточных и заключительных этапов разработки программного обеспечения, для чего применялись различные графические нотации и технологии, которые впоследствии были использованы для создания стандарта объектного моделирования.

В ноябре 1997 г. после продолжительного процесса объединения различных методик группа OMG (Object Management Group) приняла получившийся в результате унифицированный язык моделирования (Unified Model Language – UML) в качестве стандарта. В 2001 г. члены OMG начали работу над новой версией UML, добавляя в нее недостающие элементы и устраняя недостатки, выявленные в UML1. Версия UML2 была принята в 2004 г. С официальной спецификацией UML можно ознакомится на веб-сайте OMG по адресу www.omg.org.

Идея создания языка UML включала в себя не только реализацию стандарта для документирования и общения разработчиков, но и реализацию

возможности использования UML как языка программирования. Этот момент вызывал массу проблем при осуществлении, так как язык визуального моделирования по определению не мог содержать в себе всей выразительности объектно-ориентированных языков в плане проектирования (программирования) динамики и реализации алгоритмов. Нужен был язык, который на более высоком (абстрактном) уровне смог бы обеспечить разработку на UML. Такой язык был создан – это объектный язык ограничений OCL (Object Constraint Language).

Тенденции развития средств разработки программных систем заключаются в создании таких средств, которые обеспечили бы не только автоматизацию всех этапов и процессов разработки программных систем, но и связь между результатами этапов. Одним из ключевых соединительных узлов является связь между проектными моделями и программным кодом. Когда разработка программных систем начинается от проектирования ее структуры до последующего кодирования и все изменения в функциях разрабатываемой системы реализуются начиная с перепроектирования архитектуры, то такая технология называется ориентированной на архитектуру (Model Driven Architecture – MDA).

Компания Borland начиная с седьмой версии своей среды разработки (Delphi) уже использует набор компонент, реализующий подход, ориентированный на архитектуру, но в этой версии присутствует еще масса недостатков и недоработок. В последней своей версии (Delphi 2006) компания Borland модернизировала технологию, ориентированную на архитектуру, что позволило использовать ее для разработки промышленных приложений, в том числе и для платформы Microsoft Framework .Net.

1.2. Эволюция моделей жизненного цикла программного обеспечения

1.2.1. Каскадная модель

Первоначально (1970–1985 гг.) была предложена и использовалась *каскадная схема разработки программного обеспечения* (ПО) (рис. 1.9), которая предполагала, что переход на следующую стадию осуществляется после того, как полностью будут завершены проектные операции предыдущей стадии и получены все исходные данные для следующей стадии.

Именно такую схему и используют обычно при блочно-иерархическом подходе к разработке сложных *технических* объектов, обеспечивая очень высокие параметры эффективности разработки. Однако данная схема оказалась применимой только к созданию систем, для которых в самом начале разработки удавалось точно и полно сформулировать все требования. Это уменьшало вероятность возникновения в процессе разработки проблем, связанных

с принятием неудачного решения на предыдущих стадиях. На практике такие разработки встречаются крайне редко.

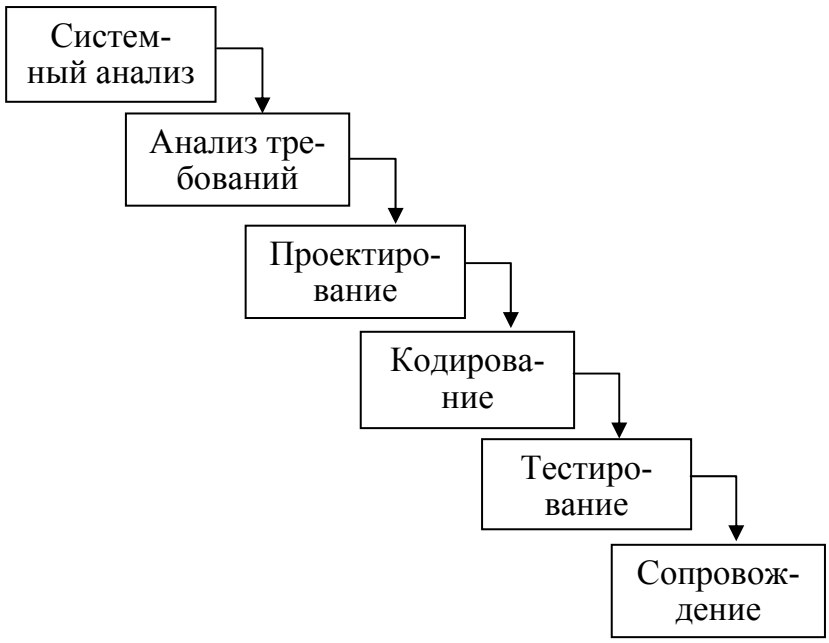


Рис. 1.9. Каскадная модель разработки программного обеспечения

Схема, поддерживающая итерационный характер процесса разработки, была названа схемой с промежуточным контролем (рис. 1.10). Контроль, который выполняется по данной схеме после завершения каждого этапа, позволяет при необходимости вернуться на любой уровень и внести необходимые изменения.

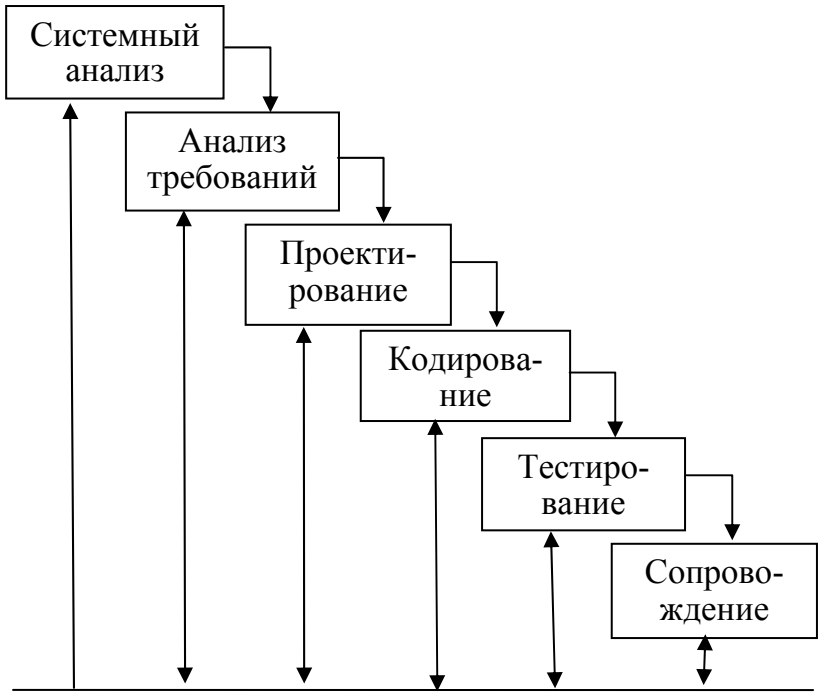


Рис. 1.10. Каскадная модель разработки ПО с промежуточным контролем

Охарактеризуем содержание основных этапов.

Подразумевается, что разработка начинается на системном уровне и проходит через анализ, проектирование, кодирование, тестирование и сопровождение. При этом моделируются действия стандартного инженерного цикла.

Системный анализ задает роль каждого элемента в компьютерной системе, взаимодействие элементов друг с другом. Поскольку ПО является лишь частью большой системы, то анализ начинается с определения требований ко всем системным элементам и назначения подмножества этих требований программному элементу. Необходимость системного подхода явно проявляется, когда формируется интерфейс ПО с другими элементами (аппаратурой, людьми, базами данных). На этом же этапе начинается решение задачи планирования проекта ПО. В ходе планирования проекта определяются объем проектных работ и их риск, необходимые трудозатраты, формируются рабочие задачи и план-график работ.

Анализ требований относится к программному элементу – программному обеспечению. Уточняются и детализируются его функции, характеристики и интерфейс.

Все определения документируются в *спецификации анализа.* Здесь же завершается решение задачи планирования проекта.

Проектирование состоит в создании представлений:

- архитектуры ПО;
- модульной структуры ПО;
- алгоритмической структуры ПО;
- структуры данных;
- входного и выходного интерфейса (входных и выходных форм данных).

Исходные данные для проектирования содержатся в *спецификации анализа,* т. е. в ходе проектирования выполняется трансляция требований к ПО во множество проектных представлений: при решении задач проектирования основное внимание уделяется качеству будущего программного продукта.

Кодирование – перевод результатов проектирования в текст на языке программирования.

Тестирование – выполнение программы для выявления дефектов в функциях, логике и форме реализации программного продукта.

Сопровождение – это внесение изменений в эксплуатируемое ПО. Цели изменений:

- исправление ошибок;
- адаптация к изменениям внешней для ПО среды;

- усовершенствование ПО по требованиям заказчика.

Сопровождение ПО состоит в повторном применении каждого из предшествующих шагов (этапов) жизненного цикла к существующей программе, но не в разработке новой программы.

Как и любая инженерная схема, классический жизненный цикл имеет достоинства и недостатки.

Достоинства классического жизненного цикла:

1. получение в конце каждой стадии законченного набора проектной документации, отвечающего требованиям полноты и согласованности;

2. простота планирования процесса разработки.

Недостатки классического жизненного цикла:

1. реальные проекты часто требуют отклонения от стандартной последовательности шагов;

2. цикл основан на точной формулировке исходных требований к ПО (реально в начале проекта требования заказчика определены лишь частично);

3. результаты проекта доступны заказчику только в конце работы.

1.2.2. Спиральная модель

Спиральная модель (автор Барри Боэм, 1988) базируется на лучших свойствах классического жизненного цикла и макетирования, к которым добавляется новый элемент – анализ риска, отсутствующий в этих парадигмах.

Как показано на рис. 1.11, модель определяет четыре действия, представляемые четырьмя квадрантами спирали.

1. Планирование – определение целей, вариантов и ограничений.
2. Анализ риска – анализ вариантов и распознавание/выбор риска.
3. Конструирование – разработка продукта следующего уровня.
4. Оценивание – оценка заказчиком текущих результатов конструирования.

Интегрирующий аспект спиральной модели очевиден при учете радиального измерения спирали. С каждой итерацией по спирали (продвижением от центра к периферии) строятся все более полные версии ПО.

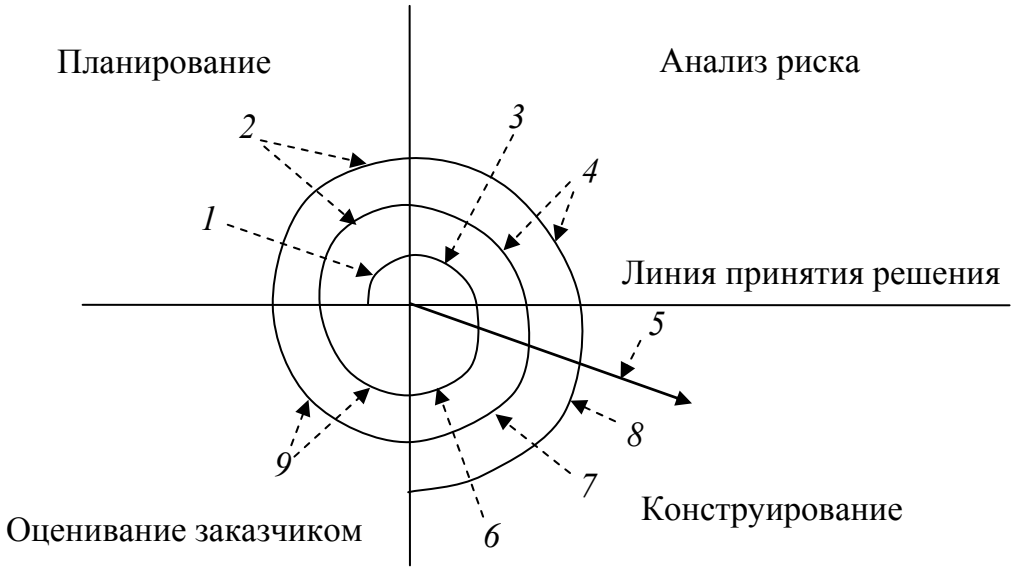


Рис. 1.11. Спиральная модель: *1* – начальный сбор требований и планирование проекта; *2* – та же работа, но на основе рекомендаций заказчика; *3* – анализ риска на основе начальных требований; *4* – анализ риска на основе реакции заказчика; *5* – переход к комплексной системе; *6* – начальный макет системы; *7* – следующий уровень макета; *8* – сконструированная система; *9* – оценивание заказчиком

В первом витке спирали определяются начальные цели, варианты и ограничения, распознается и анализируется риск. Если анализ риска показывает неопределенность требований, на помощь разработчику и заказчику приходит макетирование (используемое в квадранте конструирования). Для дальнейшего определения проблемных и уточненных требований может быть использовано моделирование. Заказчик оценивает инженерную (конструкторскую) работу и вносит предложения по модификации (квадрант оценки заказчиком). Следующая фаза планирования и анализа риска базируется на предложениях заказчика. В каждом цикле по спирали результаты анализа риска формируются в виде «продолжать, не продолжать». Если риск слишком велик, проект может быть остановлен.

В большинстве случаев движение по спирали продолжается, с каждым шагом продвигая разработчиков к более общей модели системы. В каждом цикле по спирали требуется конструирование (нижний правый квадрант), которое может быть реализовано классическим жизненным циклом или макетированием. Заметим, что количество действий по разработке (происходящих в правом нижнем квадранте) возрастает по мере продвижения от центра спирали.

Достоинства спиральной модели:

1. наиболее реально (в виде эволюции) отображает разработку программного обеспечения;
2. позволяет явно учитывать риск на каждом витке эволюции разработки;

3. включает шаг системного подхода в итерационную структуру разработки;

4. использует моделирование для уменьшения риска и совершенствования программного изделия.

Недостатки спиральной модели:

1. повышенные требования к заказчику;
2. трудности контроля и управления временем разработки.

1.2.3. Макетирование

Достаточно часто заказчик не может сформулировать подробные требования по вводу, обработке или выводу данных для будущего программного продукта. С другой стороны, разработчик может сомневаться в приспосабливаемости продукта под операционную систему, в форме диалога с пользователем или в эффективности реализуемого алгоритма. В этих случаях целесообразно использовать макетирование.

Основная цель макетирования – снять неопределенности в требованиях заказчика.

Макетирование (прототипирование) – это процесс создания модели требуемого программного продукта.

Модель может принимать одну из трех форм:

бумажный макет или макет на основе ПК (изображает или рисует человеко-машинный диалог);

работающий макет (выполняет некоторую часть требуемых функций);

существующая программа (характеристики которой затем должны быть улучшены).

Макетирование основывается на многократном повторении итераций, в которых участвуют заказчик и разработчик (рис. 1.12).

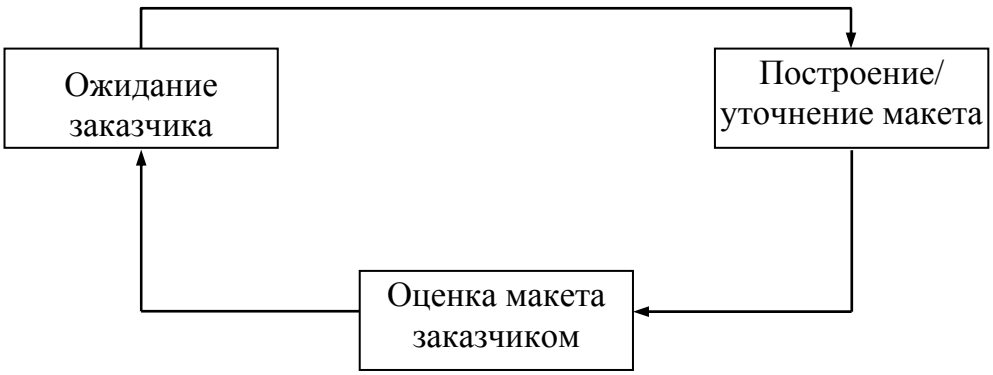


Рис. 1.12. Макетирование

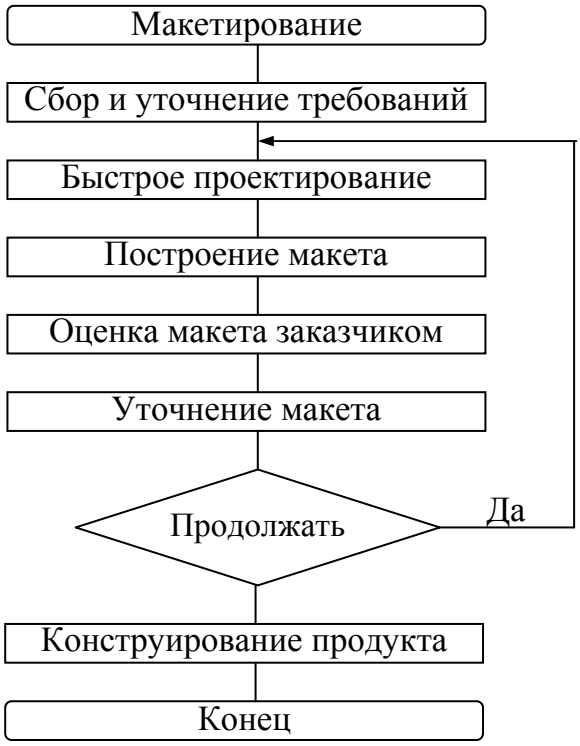


Рис. 1.13. Последовательность действий при макетировании

Последовательность действий при макетировании представлена на рис. 1.13.

Макетирование начинается со сбора и уточнения требований к создаваемому ПО. Разработчик и заказчик встречаются и определяют все цели ПО, устанавливают, какие требования известны, а какие предстоит доопределить.

Затем выполняется быстрое проектирование. В нем внимание сосредоточивается на тех характеристиках ПО, которые должны быть видимы пользователю.

Быстрое проектирование приводит к построению макета.

Макет оценивается заказчиком и используется для уточнения требований к ПО. Итерации повторяются до тех пор, пока макет не выявит все требования заказчика и тем самым не даст возможность разработчику понять, что должно быть сделано.

Достоинство макетирования – обеспечивает определение полных требований к ПО.

Недостатки макетирования:
заказчик может принять макет за продукт;
разработчик может принять макет за продукт.

Поясним суть недостатков. Когда заказчик видит работающую версию ПО, он перестает сознавать, что детали макета скреплены «жевательной резинкой и проволокой»; он забывает, что в погоне за работающим вариантом оставлены нерешенными вопросы качества и удобства сопровождения ПО. Когда заказчику говорят, что продукт должен быть перестроен, он начинает возмущаться и требовать, чтобы макет «в три приема» был превращен в рабочий продукт. Очень часто это отрицательно сказывается на управлении разработкой ПО.

С другой стороны, для быстрого получения работающего макета разработчик часто идет на определенные компромиссы. Могут использоваться не самые подходящие язык программирования или операционная система. Для простой демонстрации возможностей может применяться неэффективный алгоритм. Спустя некоторое время разработчик забывает о причинах, по которым эти средства не подходят. В результате далеко не идеальный выбранный вариант интегрируется в систему.

Очевидно, что преодоление этих недостатков требует борьбы с житейским соблазном – принять желаемое за действительное.

1.2.4. Быстрая разработка приложений

Модель быстрой разработки приложений (Rapid Application Development) обеспечивает экстремально короткий цикл разработки. RAD – высокоскоростная адаптация линейной последовательной модели, в которой быстрая разработка достигается за счет использования компонентно-ориентированного конструирования. Если требования полностью определены, а проектная область ограничена, RAD-процесс позволяет группе создать полностью функциональную систему за очень короткое время (60–90 дней).

RAD-подход ориентирован на разработку информационных систем и выделяет следующие этапы:

бизнес-моделирование. Моделируется информационный поток между бизнес-функциями. Ищутся ответы на следующие вопросы: Какая информация руководит бизнес-процессом? Какая информация генерируется? Кто генерирует ее? Где информация применяется? Кто обрабатывает ее?

моделирование данных. Информационный поток, определенный на этапе бизнес-моделирования, отображается в наборе объектов данных, которые требуются для поддержки бизнеса. Идентифицируются характеристики (свойства, атрибуты) каждого объекта, определяются отношения между объектами;

моделирование обработки. Определяются преобразования объектов данных, обеспечивающие реализацию бизнес-функций. Создаются описания обработки для добавления, модификации, удаления или нахождения (исправления) объектов данных;

генерация приложения. Предполагается использование методов, ориентированных на языки программирования 4-го поколения. Вместо создания ПО с помощью языков программирования 3-го поколения RAD-процесс работает с повторно используемыми программными компонентами или создает повторно используемые компоненты. Для обеспечения конструирования применяются утилиты автоматизации;

тестирование и объединение. Поскольку применяются повторно используемые компоненты, многие программные элементы уже протестированы. Это уменьшает время тестирования (хотя все новые элементы должны быть протестированы).

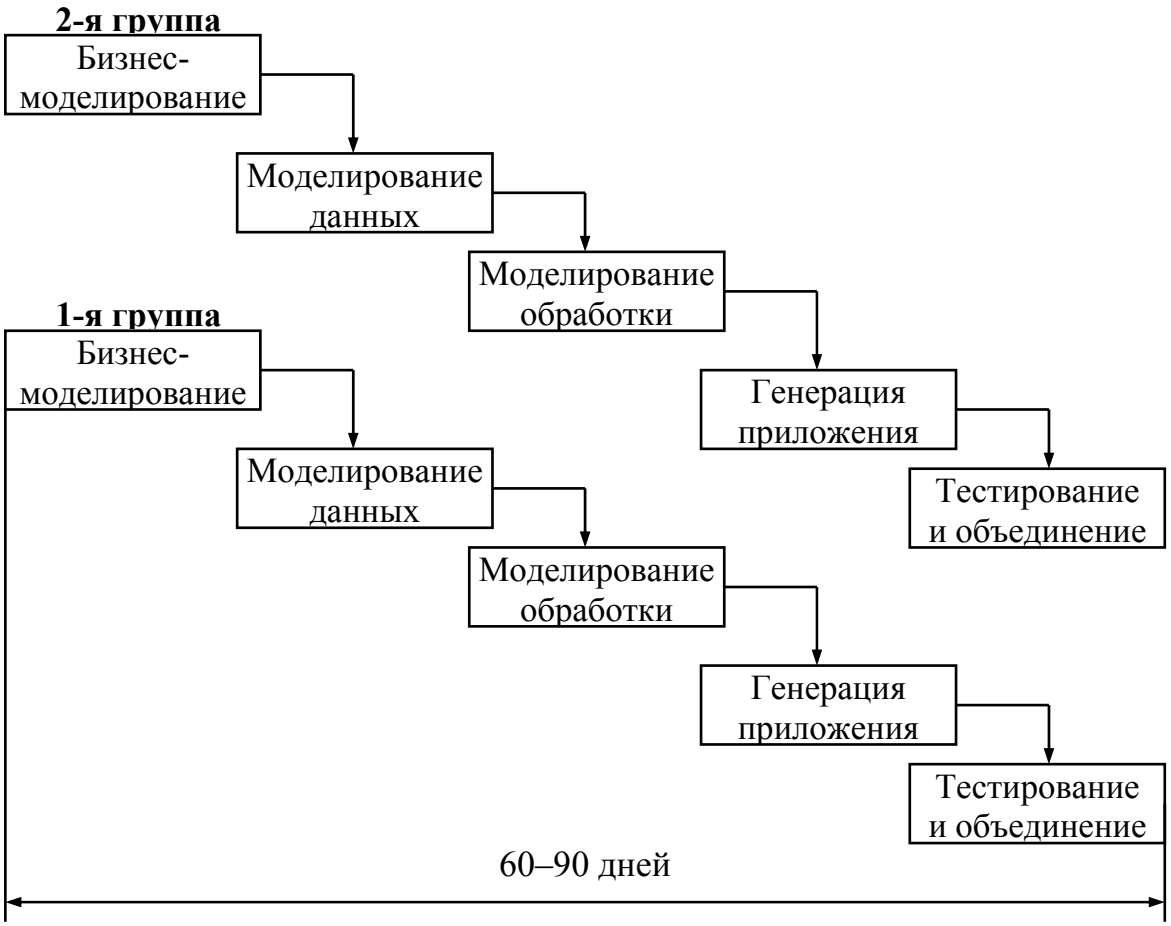


Рис. 1.14. Модель быстрой разработки приложений

Применение RAD возможно в том случае, когда каждая главная функция может быть завершена за 3 месяца. Каждая главная функция адресуется отдельной группе разработчиков, а затем интегрируется в целую систему.

Применение RAD имеет и свои недостатки, и ограничения:

для больших проектов в RAD требуются существенные людские ресурсы (необходимо создать достаточное количество групп);

RAD применима только для таких приложений, которые могут декомпозироваться на отдельные модули и в которых производительность не является критической величиной;

RAD неприменима в условиях высоких технических рисков (т. е. при использовании новой технологии).

1.2.5. Компонентно-ориентированная модель

Компонентно-ориентированная модель является развитием спиральной. В этой модели конкретизируется содержание квадранта конструирования – оно отражает тот факт, что в современных условиях новая разработка

должна основываться на повторном использовании существующих программных компонентов (рис. 1.15).

Программные компоненты, созданные в реализованных программных проектах, хранятся в библиотеке. В новом программном проекте, исходя из требований заказчика, выявляются кандидаты в компоненты. Далее проверяется наличие этих кандидатов в библиотеке. Если они найдены, то компоненты извлекаются из библиотеки и используются повторно. В противном случае создаются новые компоненты, они применяются в проекте и включаются в библиотеку.

Достоинства компонентно-ориентированной модели:

1. уменьшает на 30 % время разработки программного продукта;
2. уменьшает стоимость программной разработки до 70 %;
3. увеличивает в полтора раза производительность разработки.
4. Недостатком такой модели является сложность организации процесса разработки ПО по данной модели.

1.2.6. XP-процесс

Экстремальное программирование (eXtreme Programming, XP) – облегченный (подвижный) процесс (или методология), главный автор которого Кент Бек (1999). XP-процесс ориентирован на группы малого и среднего размера, строящие программное обеспечение в условиях неопределенных или быстро изменяющихся требований. XP-группу образуют до 10 сотрудников, которые размещаются в одном помещении.

Основная идея XP – устранить высокую стоимость изменения, характерную для приложений с использованием объектов, паттернов и реляционных баз данных. Поэтому XP-процесс должен быть высокодинамичным процессом. XP-группа имеет дело с изменениями требований на всем протяжении итерационного цикла разработки, причем цикл состоит из очень коротких итераций. Четырьмя базовыми действиями в XP-цикле являются: кодирование, тестирование, выслушивание заказчика и проектирование. Динамизм обеспечивается с помощью четырех характеристик: непрерывной связи с заказчиком (и в пределах группы), простоты (всегда выбирается минимальное решение), быстрой обратной связи (с помощью модульного и функционального тестирования), смелости в проведении профилактики возможных проблем.

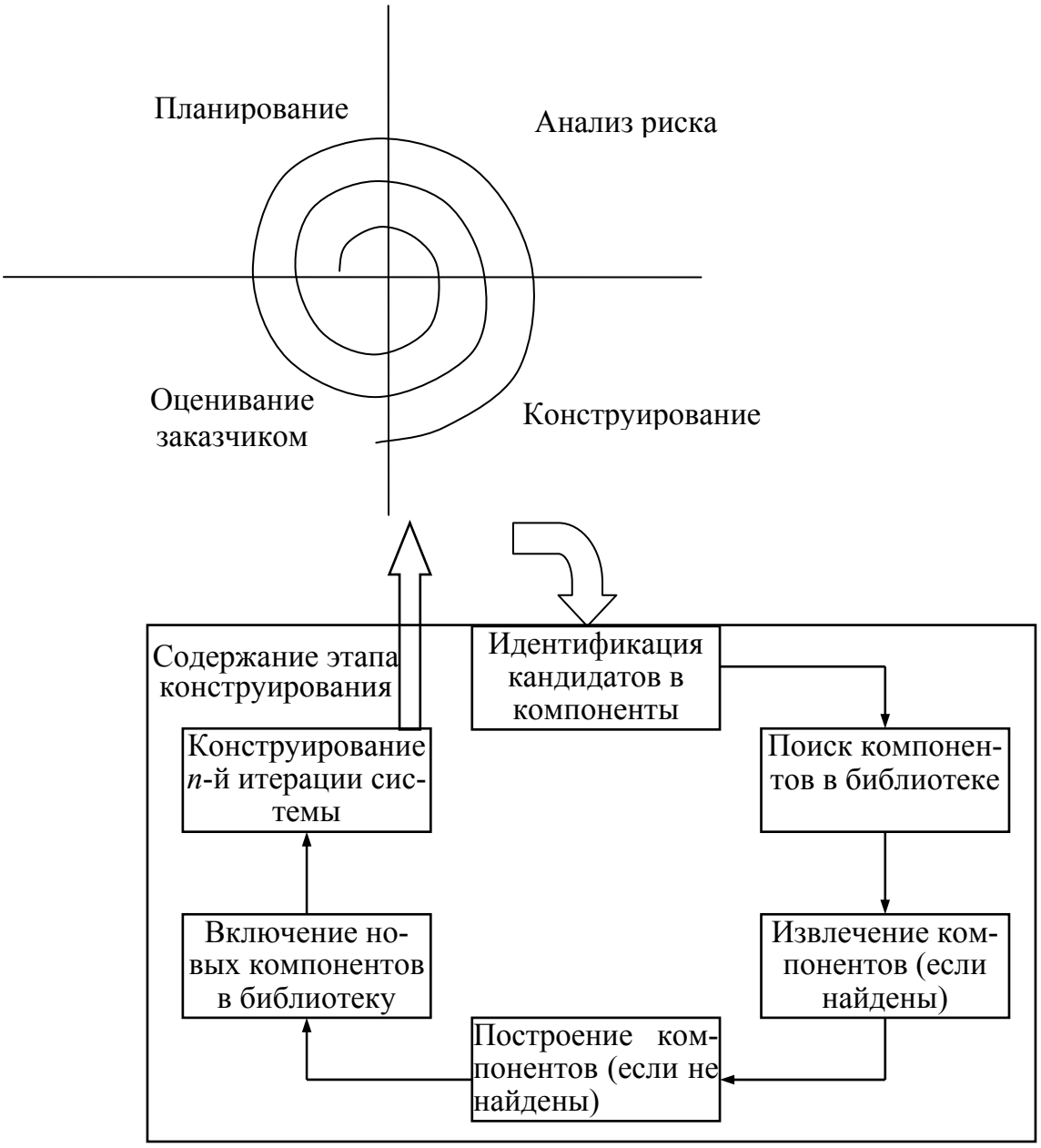


Рис. 1.15. Компонентно-ориентированная модель

Большинство принципов, поддерживаемых в XP (минимальность, простота, эволюционный цикл разработки, малая длительность итерации, участие пользователя, оптимальные стандарты кодирования и т. д.), продиктованы здравым смыслом и применяются в любом упорядоченном процессе. Просто в XP эти принципы достигают «экстремальных значений» (табл. 1.1).

Таблица 1.1

Экстремумы в экстремальном программировании

Практика здравого смысла	XP-экстремум	XP-реализация
Проверки кода	Код проверяется все время	Парное программирование
Тестирование	Тестирование выполняется все время, даже с помощью заказчиков	Тестирование модуля, функциональное тестирование
Проектирование	Проектирование является частью ежедневной деятельности каждого разработчика	Реорганизация (refactoring)
Простота	Для системы выбирается простейшее проектное решение, поддерживающее ее текущую функциональность	Самая простая вещь, которая могла бы работать
Архитектура	Каждый постоянно работает над уточнением архитектуры	Метафора
Тестирование интеграции	Интегрируется и тестируется несколько раз в день	Непрерывная интеграция
Короткие итерации	Итерации являются предельно короткими, продолжаются секунды, минуты, часы, а не недели, месяцы или годы	Игра планирования

Тот, кто принимает принцип минимального решения за хакерство, ошибается; в действительности XP – строго упорядоченный процесс. Простые решения, имеющие высший приоритет, в настоящее время рассматриваются как наиболее ценные части системы, в отличие от проектных решений, которые пока не нужны, а могут (в условиях изменения требований и операционной среды) и вообще не понадобиться. Базис XP образуют перечисленные ниже двенадцать методов.

1. Игра планирования (Planning game) – быстрое определение области действия следующей реализации путем объединения деловых приоритетов и технических оценок. Заказчик формирует область действия, приоритетность и сроки с точки зрения бизнеса, а разработчики оценивают и прослеживают продвижение (прогресс).

2. Частая смена версий (Small releases) – быстрый запуск в производство простой системы. Новые версии реализуются в очень коротком (двухнедельном) цикле.

3. Метафора (Metaphor) – вся разработка проводится на основе простой, общедоступной истории о том, как работает вся система.

4. Простое проектирование (Simple design) – проектирование выполняется настолько просто, насколько это возможно в данный момент.

5. Тестирование (Testing) – непрерывное написание тестов для модулей, которые должны выполняться безупречно; заказчики пишут тесты для демонстрации законченности функций. «Тестируй, а затем кодируй» означает, что входным критерием для написания кода является «отказавший» тестовый вариант.

6. Реорганизация (Refactoring) – система реструктурируется, но ее поведение не изменяется; цель – устранить дублирование, улучшить взаимодействие, упростить систему или добавить в нее гибкость.

7. Парное программирование (Pair programming) – весь код пишется двумя программистами, работающими на одном компьютере.

8. Коллективное владение кодом (Collective ownership) – любой разработчик может улучшать любой код системы в любое время.

9. Непрерывная интеграция (Continuous integration) – система интегрируется и строится много раз в день по мере завершения каждой задачи. Непрерывное регрессионное тестирование, т. е. повторение предыдущих тестов, гарантирует, что изменения требований не приведут к регрессу функциональности.

10. 40-часовая неделя (40-hour week) – как правило, работают не более 40 часов в неделю. Нельзя удваивать рабочую неделю за счет сверхурочных работ.

11. Локальный заказчик (On-site customer) – в группе все время должен находиться представитель заказчика, действительно готовый отвечать на вопросы разработчиков.

12. Стандарты кодирования (Coding standards) – должны выдерживаться правила, обеспечивающие одинаковое представление программного кода во всех частях программной системы.

1.3. Стандарты, регламентирующие процесс разработки программного обеспечения

1.3.1. ГОСТ Р ИСО 9000–2001. Системы менеджмента качества. Основные положения и словарь

1.3.1.1. Предисловие

Стандарт разработан Всероссийским научно-исследовательским институтом сертификации (ВНИИС). Представляет собой аутентичный текст международного стандарта ИСО 9000–2000 «Системы менеджмента качества. Основные положения и словарь».

1.3.1.2. Введение

Семейство стандартов ИСО 9000, перечисленных ниже, было разработано для того, чтобы помочь организациям всех видов и размеров внедрять и обеспечивать функционирование эффективных систем менеджмента качества:

ГОСТ Р ИСО 9000–2001 описывает основные положения систем менеджмента качества и устанавливает терминологию для систем менеджмента качества;

ГОСТ Р ИСО 9001–2001 определяет требования к системам менеджмента качества для тех случаев, когда организации необходимо продемонстрировать свою способность предоставлять продукцию, отвечающую требованиям потребителей и установленным к ней обязательным требованиям. Направлен на повышение удовлетворенности потребителей;

ГОСТ Р ИСО 9004–2001 содержит рекомендации, рассматривающие как результативность, так и эффективность системы менеджмента качества. Целью этого стандарта является улучшение деятельности организации и удовлетворенность потребителей и других заинтересованных сторон;

ИСО 19011* содержит методические указания по аудиту (проверке) систем менеджмента качества и охраны окружающей среды.

Вместе они образуют согласованный комплекс стандартов на системы менеджмента качества, содействующий взаимопониманию в национальной и международной торговле.

Принципы менеджмента качества. Для успешного руководства организацией и ее функционирования необходимо направлять ее и управлять систематически и прозрачным способом. Успех может быть достигнут в результате внедрения и поддержания в рабочем состоянии системы менеджмента качества, разработанной для постоянного улучшения деятельности с учетом потребностей всех заинтересованных сторон. Управление организацией включает менеджмент качества наряду с другими аспектами менеджмента.

Восемь принципов менеджмента качества были определены для того, чтобы высшее руководство могло следовать им с целью улучшения деятельности организации.

1. Ориентация на потребителя

Организации зависят от своих потребителей, и поэтому должны понимать их текущие и будущие потребности, выполнять их требования и стремиться превзойти их ожидания.

2. Лидерство руководителя

Руководители обеспечивают единство цели и направления деятельности организации. Они должны создавать и поддерживать внутреннюю среду, в которой работники могут быть полностью вовлечены в решение задач организации.

3. Вовлечение работников

Работники всех уровней составляют основу организации, и их полное вовлечение дает возможность организации с выгодой использовать их способности.

4. Процессный подход

Желаемый результат достигается эффективнее, когда деятельностью и соответствующими ресурсами управляют как процессом.

5. Системный подход к менеджменту

Выявление, понимание и менеджмент взаимосвязанных процессов как системы содействуют результативности и эффективности организации при достижении ее целей.

6. Постоянное улучшение

Постоянное улучшение деятельности организации в целом следует рассматривать как ее неизменную цель.

7. Принятие решений, основанное на фактах

Эффективные решения основываются на анализе данных и информации.

8. Взаимовыгодные отношения с поставщиками

Организация и ее поставщики взаимозависимы, и отношения взаимной выгоды повышают способность обеих сторон создавать ценности.

Эти восемь принципов менеджмента качества образуют основу для стандартов на системы менеджмента качества, входящих в семейство ИСО 9000.

1.3.1.3. Область применения

Настоящий стандарт устанавливает основные положения систем менеджмента качества, являющихся объектом стандартов семейства ИСО 9000, и определяет соответствующие термины.

Настоящий стандарт может использоваться:

а) организациями, стремящимися добиться преимущества посредством внедрения системы менеджмента качества;

б) организациями, стремящимися получить уверенность в том, что их заданные требования к продукции будут выполнены поставщиками;

в) пользователями продукции;

г) теми, кто заинтересован в едином понимании терминологии, применяемой в менеджменте качества (например, поставщики, потребители, регламентирующие органы);

д) теми сторонами, внутренними или внешними по отношению к организации, которые оценивают систему менеджмента качества или проверяют ее на соответствие требованиям ГОСТ Р ИСО 9001 (например, аудиторы, органы по сертификации/регистрации);

е) теми сторонами, внутренними или внешними по отношению к организации, которые консультируют или проводят обучение по системе менеджмента качества, соответствующей данной организации;

ж) разработчиками соответствующих стандартов.

1.3.1.4. Основные положения систем менеджмента качества

Обоснование необходимости систем менеджмента качества. Системы менеджмента качества могут содействовать организациям в повышении удовлетворенности потребителей.

Потребителям необходима продукция, характеристики которой удовлетворяли бы их потребности и ожидания. Эти потребности и ожидания, как правило, отражаются в технических условиях на продукцию и обычно считаются требованиями потребителей. Требования могут быть установлены потребителем в контракте или определены самой организацией. В любом случае приемлемость продукции в конечном счете устанавливает потребитель. Поскольку потребности и ожидания потребителей меняются, организации также испытывают давление, обусловленное конкуренцией и техническим прогрессом, они должны постоянно совершенствовать свою продукцию и свои процессы.

Системный подход к менеджменту качества побуждает организации анализировать требования потребителей, определять процессы, способствующие получению продукции, приемлемой для потребителей, а также поддерживать эти процессы в управляемом состоянии. Система менеджмента качества может быть основой постоянного улучшения с целью увеличения вероятности повышения удовлетворенности как потребителей, так и других заинтересованных сторон. Она дает уверенность самой организации и потребителям в ее способности поставлять продукцию, полностью соответствующую требованиям.

Требования к системам менеджмента качества и требования к продукции. Семейство стандартов ИСО 9000 проводит различие между требованиями к системам менеджмента качества и требованиями к продукции.

Требования к системам менеджмента качества установлены в ГОСТ Р ИСО 9001. Они являются общими и применимыми к организациям в любых секторах промышленности или экономики независимо от категории продукции. ГОСТ Р ИСО 9001 не устанавливает требований к продукции.

Требования к продукции могут быть установлены потребителями или организацией, исходя из предполагаемых запросов потребителей или требо-

ваний регламентов. Требования к продукции и в ряде случаев к связанным с ней процессам могут содержаться, например в технических условиях, стандартах на продукцию, стандартах на процессы, контрактных соглашениях и регламентах.

Подход к системам менеджмента качества. Подход к разработке и внедрению системы менеджмента качества состоит из нескольких ступеней, включающих:

а) установление потребностей и ожиданий потребителей и других заинтересованных сторон;

б) разработку политики и целей организации в области качества;

в) установление процессов и ответственности, необходимых для достижения целей в области качества;

г) установление и определение необходимых ресурсов и обеспечение ими для достижения целей в области качества;

д) разработку методов для измерения результативности и эффективности каждого процесса;

е) применение данных этих измерений для определения результативности и эффективности каждого процесса;

ж) определение средств, необходимых для предупреждения несоответствий и устранения их причин;

з) разработку и применение процесса для постоянного улучшения системы менеджмента качества.

Такой подход также применяется для поддержания в рабочем состоянии и улучшения имеющейся системы менеджмента качества.

Организация, принимающая указанный выше подход, создает уверенность в возможностях своих процессов и качестве своей продукции, а также обеспечивает основу для постоянного улучшения. Это может привести к возрастанию удовлетворенности потребителей и других заинтересованных сторон и успеху организации.

Процессный подход. Любая деятельность или комплекс деятельности, в которой используются ресурсы для преобразования входов в выходы, может рассматриваться как процесс.

Чтобы результативно функционировать, организации должны определять и управлять многочисленными взаимосвязанными и взаимодействующими процессами. Часто выход одного процесса образует непосредственно вход следующего. Систематическая идентификация и менеджмент применяемых организацией процессов и прежде всего обеспечения их взаимодействия могут считаться «процессным подходом».

Рис. 1.16. Модель системы менеджмента качества, основанной на процессном подходе*

Назначение настоящего стандарта – побуждать принятие процессного подхода к менеджменту организации.

На рис. 1.16 приведена основанная на процессном подходе система менеджмента качества, описанная в семействе стандартов ИСО 9000. Он показывает, что заинтересованные стороны играют существенную роль в предоставлении организации входных данных. Наблюдение за удовлетворенностью заинтересованных сторон требует оценки информации, касающейся восприятия заинтересованными сторонами степени выполнения их потребностей и ожиданий. Модель (рис. 1.16), не показывает процессы на детальном уровне.

Политика и цели в области качества. Политика и цели в области качества устанавливаются, чтобы служить ориентиром для организации. Они определяют желаемые результаты и способствуют использованию организацией ресурсов для достижения этих результатов. Политика в области качества обеспечивает основу для разработки и анализа целей в области

* Примечание. Формулировки, данные в круглых скобках, неприменимы к ГОСТ Р ИСО 9001.

качества. Цели в области качества необходимо согласовывать с политикой в области качества и приверженностью к постоянному улучшению, а результаты должны быть измеримыми. Достижение целей в области качества может оказывать позитивное воздействие на качество продукции, эффективность работы и финансовые показатели и, следовательно, на удовлетворенность и уверенность заинтересованных сторон.

Роль высшего руководства в системе менеджмента качества. С помощью лидерства и реальных действий высшее руководство может создавать обстановку, способствующую полному вовлечению работников и эффективной работе системы менеджмента качества. Принципы менеджмента качества могут использоваться высшим руководством как основа для выполнения своей роли:

а) в разработке и поддержании политики и целей организации в области качества;

б) популяризации политики и целей в области качества во всей организации для повышения осознания, мотивации и вовлечения персонала;

в) обеспечении ориентации на требования потребителей во всей организации;

г) обеспечении внедрения соответствующих процессов, позволяющих выполнять требования потребителей и других заинтересованных сторон и достигать целей в области качества;

д) обеспечении разработки, внедрения и поддержания в рабочем состоянии эффективной системы менеджмента качества для достижения этих целей в области качества;

е) обеспечении необходимыми ресурсами;

ж) проведении периодического анализа системы менеджмента качества;

з) принятии решений в отношении политики и целей в области качества;

и) принятии решений по мерам улучшения системы менеджмента качества.

Документация. Документация дает возможность передать смысл и последовательность действий. Ее применение способствует:

а) достижению соответствия требованиям потребителя и улучшению качества;

б) обеспечению соответствующей подготовки кадров;

в) повторяемости и прослеживаемости;

г) обеспечению объективных свидетельств;

д) оцениванию эффективности и постоянной пригодности системы менеджмента качества.

Разработка документации не должна быть самоцелью, а должна добавлять ценность.

В системах менеджмента качества применяются следующие виды документов:

а) документы, предоставляющие согласованную информацию о системе менеджмента качества организации, предназначенную как для внутреннего, так и внешнего пользования; к таким документам относятся руководства по качеству;

б) документы, описывающие, как система менеджмента качества применяется к конкретной продукции, проекту или контракту; к таким документам относятся планы качества;

в) документы, устанавливающие требования; к ним относятся документы, содержащие технические требования;

г) документы, содержащие рекомендации или предложения; к ним относятся методические документы;

д) документы, содержащие информацию о том, как последовательно выполнять действия и процессы; такие документы могут включать документированные процедуры, рабочие инструкции и чертежи;

е) документы, содержащие объективные свидетельства выполненных действий или достигнутых результатов; к таким документам относятся записи.

Каждая организация определяет объем необходимой документации и ее носители. Это зависит от таких факторов, как вид и размер организации, сложность и взаимодействие процессов, сложность продукции, требования потребителей, соответствующие обязательные требования, продемонстрированные способности персонала, а также от глубины, до которой необходимо подтверждать выполнение требований к системе менеджмента качества.

Оценивание систем менеджмента качества. При оценке систем менеджмента качества следует задавать четыре основных вопроса в отношении каждого оцениваемого процесса:

1. Выявлен и определен ли соответствующим образом процесс?
2. Распределена ли ответственность?
3. Внедрены и поддерживаются ли в рабочем состоянии процедуры?
4. Эффективен ли процесс в достижении требуемых результатов?

Совокупные ответы на приведенные выше вопросы могут определить результаты оценивания. Оценка системы менеджмента качества может различаться по области применения и включать такие виды деятельности, как аудит (проверку) и анализ системы менеджмента качества, а также самооценку.

Аудиты (проверки) применяют для определения степени выполнения требований к системе менеджмента качества. Наблюдения аудитов (проверок) используют для оценки эффективности системы менеджмента качества и определения возможностей для улучшения.

Аудиты (проверки), проводимые первой стороной (самой организацией) или от ее имени для внутренних целей, могут служить основой для декларирования организацией о своем соответствии.

Аудиты (проверки), проводимые второй стороной, могут проводиться потребителями организации или другими лицами от имени потребителей.

Аудиты (проверки), проводимые третьей стороной, осуществляются внешними независимыми организациями. Такие организации, обычно имеющие аккредитацию, проводят сертификацию на соответствие требованиям, например требованиям ГОСТ Р ИСО 9001.

ИСО 19011 содержит методические указания по аудиту (проверке).

Одна из задач высшего руководства – проведение регулярного систематического оценивания пригодности, адекватности, эффективности и результативности системы менеджмента качества с учетом политики и целей в области качества. Этот анализ может включать рассмотрение необходимости адаптации политики и целей в области качества в ответ на изменение потребностей и ожиданий заинтересованных сторон. Анализ включает определение потребности в действиях.

При анализе системы менеджмента качества наряду с другими источниками информации используют отчеты по аудитам (проверкам).

Самооценка организации является всесторонним и систематическим анализом деятельности организации и результатов по отношению к системе менеджмента качества или модели совершенства (модели премии по качеству).

Самооценка может дать общее представление о деятельности организации и степени развития системы менеджмента качества. Она может также помочь определить организации области, нуждающиеся в улучшении, и приоритеты.

Постоянное улучшение. Целью постоянного улучшения системы менеджмента качества является увеличение возможности повышения удовлетворенности потребителей и других заинтересованных сторон. Действия по улучшению включают:

а) анализ и оценку существующего положения для определений областей для улучшения;

б) установление целей улучшения;

в) поиск возможных решений для достижения целей;

г) оценивание и выбор решений;

д) выполнение выбранных решений;

е) измерение, проверку, анализ и оценку результатов выполнения для установления того, достигнуты ли цели;

ж) оформление изменений.

Результаты анализируют с целью установления дальнейших возможностей для улучшения. Таким образом, улучшение является постоянным действием. Аудиты (проверки) и анализ системы менеджмента качества могут также использоваться для определения возможностей улучшения.

Роль статистических методов. Использование статистических методов может помочь в понимании изменчивости и, следовательно, может по-

мочь организациям в решении проблем и повышении результативности и эффективности. Эти методы также способствуют лучшему применению имеющихся в наличии данных для оказания помощи в принятии решений.

Изменчивость можно наблюдать в ходе и результатах многих видов деятельности, даже в условиях очевидной стабильности. Такую изменчивость можно проследить в измеряемых характеристиках продукции и процессов. Ее наличие можно заметить на различных стадиях жизненного цикла продукции от исследования рынка до обслуживания потребителей и утилизации.

Статистические методы могут помочь при измерении, описании, анализе, интерпретации и моделировании такой изменчивости даже при относительно ограниченном количестве данных. Статистический анализ таких данных может помочь лучше понять природу, масштаб и причины изменчивости, способствуя таким образом решению и даже предупреждению проблем, которые могут быть результатом такой изменчивости, а также постоянному улучшению.

Методические указания по применению статистических методов в системе менеджмента качества приведены в ИСО/ТО 10017.

Направленность систем менеджмента качества и других систем менеджмента. Система менеджмента качества является частью системы менеджмента организации, которая направлена на достижение результатов в соответствии с целями в области качества, чтобы удовлетворять потребности, ожидания и требования заинтересованных сторон. Цели в области качества дополняют другие цели организации, связанные с развитием, финансированием, рентабельностью, окружающей средой, охраной труда и безопасностью. Различные части системы менеджмента организации могут быть интегрированы вместе с системой менеджмента качества в единую систему менеджмента, использующую общие элементы. Это может облегчить планирование, выделение ресурсов, определение дополнительных целей и оценку общей эффективности организации. Система менеджмента организации может быть оценена на соответствие собственным требованиям организации. Она может быть также проверена на соответствие требованиям ГОСТ Р ИСО 9001 и ГОСТ Р ИСО 14001. Эти аудиты (проверки) могут проводиться отдельно или совместно.

Взаимосвязь между системами менеджмента качества и моделями совершенства. Подходы систем менеджмента качества, приведенные в семействе стандартов ИСО 9000, и модели совершенства основаны на общих принципах. Оба эти подхода:

а) дают возможность организации выявить свои сильные и слабые стороны;

б) содержат положения по оцениванию в сравнении с общими моделями;

в) обеспечивают основу для постоянного улучшения;

г) включают способы внешнего признания.

Различие между подходами систем менеджмента качества семейства ИСО 9000 и моделями совершенства заключается в их областях применения.

Стандарты семейства ИСО 9000 содержат требования к системам менеджмента качества и рекомендации по улучшению деятельности; оценивание систем менеджмента качества устанавливает выполнение этих требований. Модели совершенства содержат критерии, позволяющие проводить сравнительную оценку деятельности организации, и это применимо ко всем видам деятельности и ко всем заинтересованным сторонам. Критерии оценки в моделях совершенства обеспечивают организации основу для сравнения ее деятельности с деятельностью других организаций.

1.3.2. ГОСТ Р ИСО/МЭК ТО 15504

Основы оценки и аттестации зрелости процессов создания и сопровождения программных средств и информационных систем установлены стандартами ИСО/МЭК ТО 15504, разработанными на базе концепций CMM (Capability Maturity Model for Software).

1.3.2.1. Обзор

ИСО/МЭК ТО 15504 предоставляет основу для аттестации процесса жизненного цикла программных средств. Эта основа может быть использована организациями, занимающимися планированием, управлением, наблюдением, контролем и совершенствованием приобретения, поставки, разработки, эксплуатации, развития и поддержки программных средств.

ИСО/МЭК ТО 15504 предоставляет структурный подход к аттестации процесса жизненного цикла программных средств, проводящейся:

организацией или от ее имени с целью выяснения состояния ее собственных процессов для их усовершенствования;

организацией или от ее имени с целью определения пригодности ее собственных процессов для выполнения определенного требования или класса требований;

организацией или от ее имени с целью определения пригодности процессов другой организации для определенного договора или класса договоров.

Такая основа для аттестации процесса

способствует самоаттестации;

направлена на обеспечение адекватности управления аттестуемыми процессами;

принимает во внимание контекст, в котором выполняются аттестуемые процессы;

выдает набор рейтингов процесса (профиль процесса), а не результат типа зачет/незачет;

подходит ко всем сферам приложений и для организаций любого размера.

Чтобы организация могла улучшить качество своей продукции, она должна иметь проверенный, последовательный и надежный метод для аттестации состояния своих процессов, а также она должна иметь средства использования результатов как часть связной программы усовершенствования.

Использование аттестации процессов внутри организации должно способствовать:

выработке культуры постоянного совершенствования, а также соответствующих механизмов поддержания этой культуры;

разработке процессов, отвечающих бизнес-целям;

оптимизации использования ресурсов.

Это приведет к появлению зрелых организаций, максимально восприимчивых к требованиям потребителя и рынка, имеющих минимальную стоимость полного жизненного цикла своей продукции и, как результат, максимально удовлетворяющих конечного пользователя.

Покупателям также будет выгодно использовать аттестацию процесса. Ее применение при определении зрелости

уменьшит неопределенность при выборе поставщиков программно-насыщенных систем за счет того, что риски, связанные со зрелостью подрядчика, выявляются еще до заключения договора;

позволит заранее предусмотреть необходимые меры на случай возникновения рискового события;

предоставит количественные критерии выбора при сопоставлении потребностей бизнеса, требований и оценочной стоимости проекта со зрелостью конкурирующих поставщиков.

Главные преимущества стандартизированного подхода к аттестации процесса в том, что он

станет открытым, общедоступным подходом к аттестации процесса;

приведет к общему пониманию использования аттестации для усовершенствования процесса и оценки зрелости;

при приобретении облегчит оценку зрелости поставщика;

будет контролироваться и регулярно пересматриваться в свете опыта использования;

будет меняться только международным соглашением;

будет способствовать согласованию существующих схем.

Подход к аттестации процесса, определенный в ИСО/МЭК ТО 15504, разработан с целью предоставить основу для общего подхода к описанию результатов аттестации процесса, позволяющего в какой-то степени сравнивать аттестации, основанные на разных, но совместимых моделях и методах. Изощренность и сложность, требующиеся от процесса, зависят от его контекста. Например, планирование для группы из пяти человек необходимо гораздо в меньшем объеме, чем для группы из пятидесяти человек. Этот контекст влияет на то, как ведущий аттестатор судит о действии, а также на степень сравнимости профилей различных процессов.

1.3.2.2. Область применения

Аттестация процесса применяется в основном в двух контекстах (рис. 1.17).

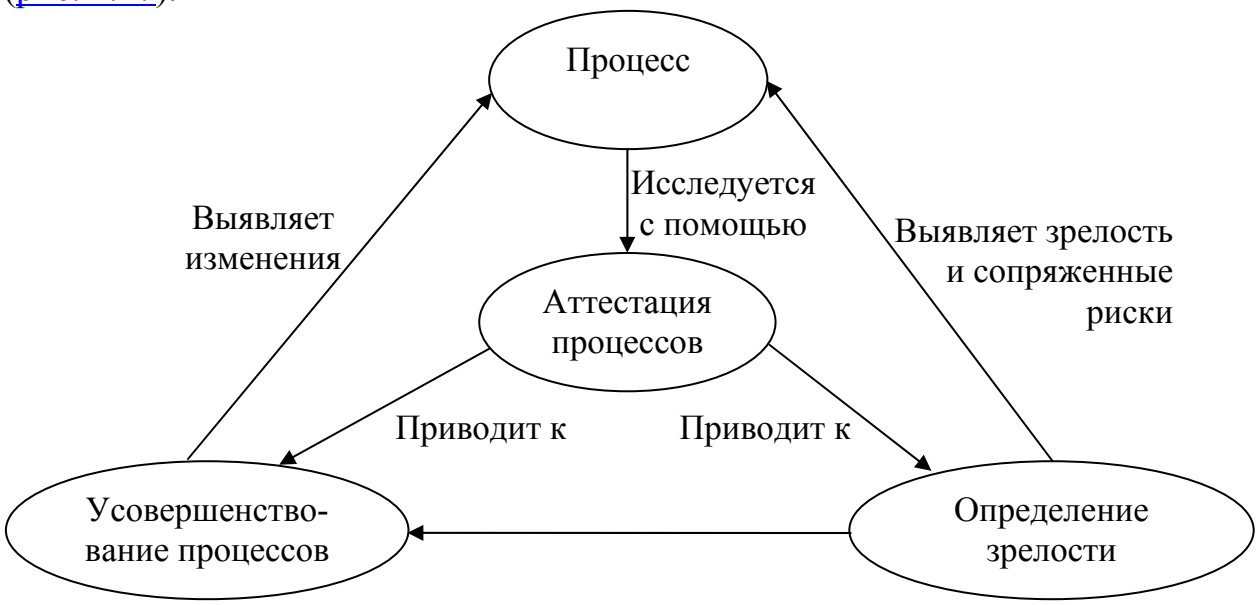


Рис. 1.17. Оценка и аттестация процессов жизненного цикла программных средств и информационных систем

В контексте усовершенствования процесса аттестация процесса предоставляет средства охарактеризовать текущую деятельность организационной единицы в терминах зрелости некоторых выбранных процессов. Анализ результатов в свете бизнес-потребностей организации выявляет сильные и слабые стороны процессов, а также присущие им риски. Это, в свою очередь, позволяет определить, эффективны ли эти процессы в достижении своих целей, а также выявить существенные причины низкого качества или превышения бюджета или сроков. Все вместе это позволяет расставить приоритеты при усовершенствовании процессов.

Определение зрелости процесса связано с анализом соответствия заявленной зрелости выбранных процессов целевому профилю зрелости процессов для того, чтобы выявить риски, связанные с выполнением проекта, использующего выбранные процессы. Заявленная зрелость может быть основана на результатах прежних аналогичных аттестаций процесса или на аттестации, проведенной с целью выработки заявленной зрелости.

Две части ИСО/МЭК ТО 15504 (части 7 и 8) посвящены использованию аттестации для усовершенствования процесса и для определения зрелости процесса. Другие части ИСО/МЭК ТО 15504 посвящены различным вопросам, относящимся к аттестации процесса.

ИСО/МЭК ТО 15504 разработан так, чтобы в рамках единого источника удовлетворить потребности потребителей, поставщиков и аттестаторов, а также их индивидуальные требования.

Преимущества, вытекающие из использования данного комплекта документов, включают:

для приобретателей:

- способность определять текущую и потенциальную зрелости процессов жизненного цикла у поставщика;
- для поставщиков:
- способность определять текущую и потенциальную зрелости своих собственных процессов жизненного цикла;
- способность определять области и приоритеты для усовершенствования процессов;
- основу, задающую схему усовершенствования процессов;
- для аттестаторов:
- основу для проведения аттестаций.

1.3.2.3. Состав ИСО/МЭК ТО 15504

ИСО/МЭК ТО 15504 состоит из девяти частей. В данном разделе описана каждая часть и ее роль в ИСО/МЭК ТО 15504.

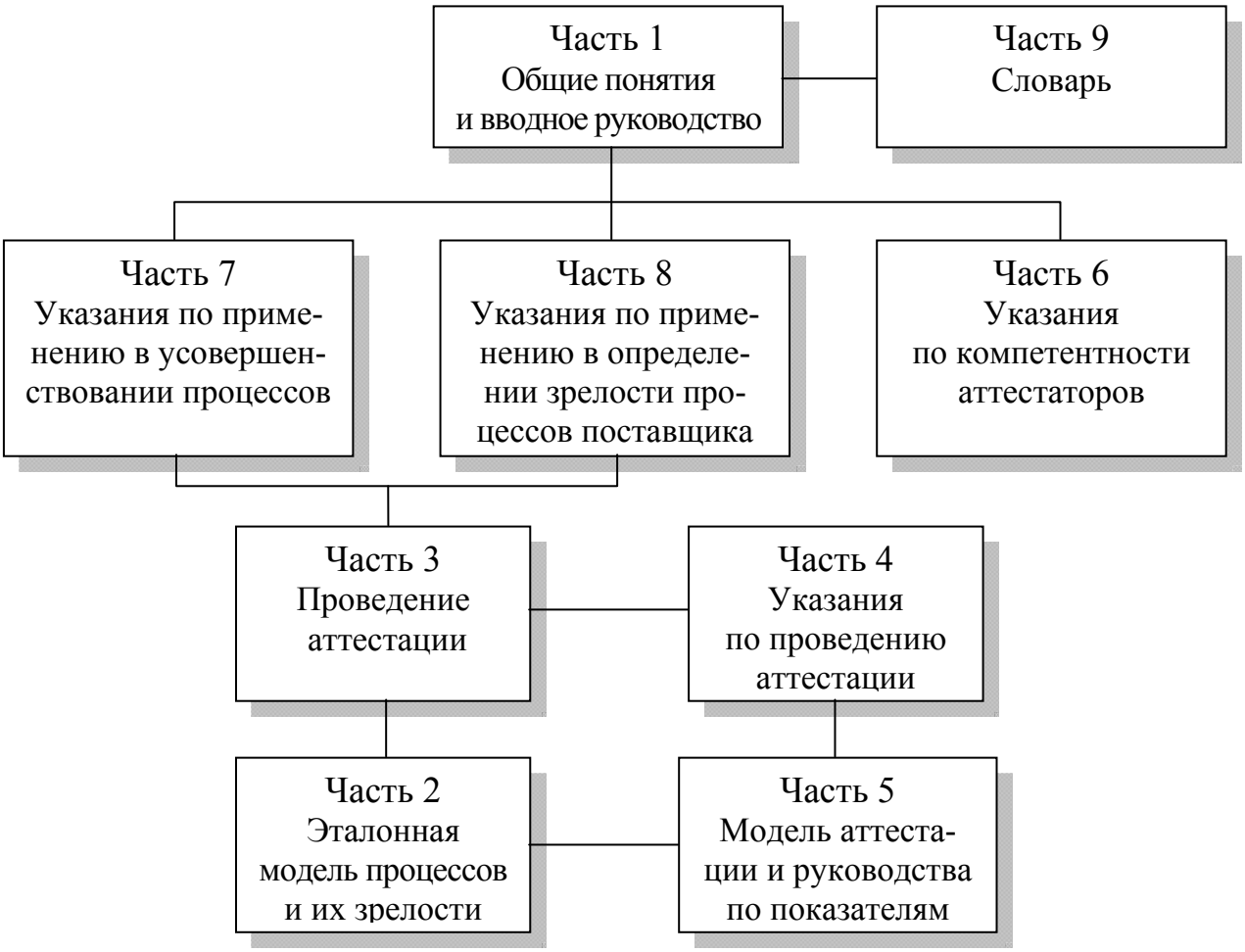


Рис. 1.18. Состав ИСО/МЭК ТО 15504

Рис. 1.18 показывает потенциальную схему использования ИСО/МЭК ТО 15504. Часть 1 (этот документ) служит общей отправной точкой ИСО/МЭК ТО 15504. Читатели, интересующиеся только усовершенствова-

нием процесса или определением зрелости поставщика, должны прочесть части 7 и 8 соответственно, содержащие детальное руководство по этим контекстам использования. Эти части позволят пользователю определить наиболее подходящий для себя способ использования нормативных компонентов ИСО/МЭК ТО 15504 (части 2 и 3). Часть 4 является руководством по применению части 2 и части 3, тогда как часть 5 – это пример аттестационной модели, совместимой с эталонной моделью (часть 2). Пользователи, интересующиеся в основном ролью аттестаторов, должны обратить внимание на часть 6, содержащую руководство по навыкам и компетентности аттестаторов.

В табл. 1.2 перечислены основные классы читателей ИСО/МЭК ТО 15504 и указаны, какие части данного набора документов посвящены первостепенным областям их интересов.

Таблица 1.2

Аудитория ИСО/МЭК ТО 15504

Класс читателей	Интересы	Части, предлагаемые к прочтению
Заказчик (спонсор) аттестации	Как проводится аттестация, какие требуются инструменты и иная поддержка, как инициировать аттестацию	1, 2, 3, 4, 6
Заказчик (спонсор) усовершенствования процесса	Инициирование программы усовершенствования, определение входов для аттестации для целей усовершенствования, использование результатов аттестации для усовершенствования	7
Заказчик (спонсор) определения зрелости процесса	Инициирование программы определения зрелости поставщика, определение целевого профиля зрелости, проверка и использование результатов аттестации при определении зрелости	8
Аттестаторы	Проведение согласованной аттестации, развитие навыков и компетентности, необходимых для проведения аттестации	2, 3, 4, 5, 6
Разработчики аттестационных моделей	Разработка модели для проведения аттестации, совместимой с эталонной моделью	2, 3, 4, 5
Разработчики методов аттестации	Разработка метода, который поддерживал бы проведение аттестации, соответствующей требованиям	2, 3, 4

Класс читателей	Интересы	Части, предлагаемые к прочтению
Разработчики инструментальных средств	Разработка инструментальных средств, которые помогали бы аттестаторам в сборе, записи, классификации данных в ходе аттестации	2, 3, 4, 5

Часть 1 (информационная) является отправной точкой ИСО/МЭК ТО 15504. Она описывает взаимодействие частей набора документов и содержит руководство по их выбору и использованию. Она разъясняет требования, содержащиеся в ИСО/МЭК ТО 15504 и их применимость к проведению аттестаций.

Часть 2 (нормативная) ИСО/МЭК ТО 15504 определяет двумерную эталонную модель для описания процессов и их зрелости, использующуюся при аттестации процессов. Эталонная модель определяет ряд процессов, установленных в терминах их назначения и итогов, а также основу для оценивания зрелости процессов посредством аттестации атрибутов процессов, структурированных по уровням зрелости. Также обозначены требования для установления совместимости различных аттестационных моделей с эталонной моделью.

Часть 3 (нормативная) ИСО/МЭК ТО 15504 определяет требования для проведения аттестации таким образом, чтобы результаты были повторимыми, надежными и согласующимися.

Часть 4 (информационная) ИСО/МЭК ТО 15504 содержит руководство по проведению аттестаций процессов жизненного цикла программных средств, по интерпретации требований ИСО/МЭК ТО 15504-2 и ИСО/МЭК ТО 15504-3 для различных контекстов аттестации. Это руководство охватывает выбор и использование документированного процесса для аттестации, совместимой аттестационной модели (или моделей), а также вспомогательного инструмента или средства аттестации. Это руководство является достаточно общим и применимо для всех организаций для проведения аттестаций с использованием разнообразных методов и технических приемов для того, чтобы поддерживаться целым рядом средств.

Часть 5 (информационная) ИСО/МЭК ТО 15504 содержит пример модели для проведения аттестации процесса, основанную на эталонной модели из ИСО/МЭК ТО 15504-2 и непосредственно с ней совместимую. Аттестационная модель (или модели) расширяет эталонную модель включением в нее всеобъемлющего набора показателей производительности и зрелости процессов.

Часть 6 (информационная) ИСО/МЭК ТО 15504 описывает компетентность, образование, специальную подготовку и опыт, необходимые аттестаторам для проведения аттестации процессов. В ней приведены механизмы,

которые могут быть использованы для демонстрации компетентности и подтверждения образования, специальной подготовки и опыта.

Часть 7 (информационная) ИСО/МЭК ТО 15504 описывает, как определять входы и использовать результаты аттестации, имеющей целью усовершенствование процесса. Это руководство включает примеры применения усовершенствования процесса в различных ситуациях.

Часть 8 (информационная) ИСО/МЭК ТО 15504 описывает, как определять входы и использовать результаты аттестации, имеющей целью определение зрелости процессов. Она охватывает определение зрелости процессов как в простейших ситуациях, так и в более сложных, включающих, например, будущую зрелость. Это руководство по проведению определения зрелости процессов может использоваться либо организацией для определения собственной зрелости, либо потребителем для определения зрелости (потенциального) поставщика.

Часть 9 (нормативная) ИСО/МЭК ТО 15504 является консолидированным словарем терминов, специфически определенных для целей ИСО/МЭК ТО 15504.

1.3.2.4. Связь с другими международными стандартами

ИСО/МЭК ТО 15504 дополняет некоторые международные стандарты и другие модели для оценки зрелости и эффективности организаций и процессов. Данный раздел описывает связь между ИСО/МЭК ТО 15504 и большинством связанных с ним международных стандартов.

ИСО/МЭК ТО 15504 преследует ту же цель, что и серия стандартов ИСО 9000 – обеспечить уверенность в системе управления качеством поставщика, одновременно предоставляя потребителям основу для оценки того, обладают ли потенциальные поставщики производственными возможностями, отвечающими их потребностям. Аттестация процессов дает пользователям возможность оценивать зрелость процессов по непрерывной шкале таким образом, что эти оценки сравнимы и повторимы в отличие от аудитов качества, основанных на ИСО 9001:1994, дающих оценку типа зачет/незачет. Кроме того, основа, описываемая ИСО/МЭК ТО 15504, предоставляет возможность подобрать объем аттестации так, чтобы он охватывал лишь определенные процессы, вызывающие интерес, а не все процессы, используемые организационной единицей.

ИСО/МЭК ТО 15504 в части «Процессного подхода» и «Системного подхода к административному управлению» отвечает концепции ИСО 2000 года в области системы качества. Требования к системам административного управления (менеджмента) качеством установлены в стандартах:

ISO 9000:2000 Quality management systems – Fundamentals and vocabulary (Системы административного управления качеством. Основы и словарь);

ISO 9000:2000 Quality management systems – Requirements (Системы административного управления качеством. Требования);

ISO 9000:2000 Quality management systems – Guidelines for performance improvement (Системы административного управления качеством. Руководящие указания по усовершенствованию);

ISO 19011:2000 Quality management systems – Guidelines for auditing quality and environmental management systems.

Эталонная модель описания, оценки и аттестации зрелости процессов деятельности, используемой при выполнении этапов жизненного цикла продукции, проекта или системы (ИСО/МЭК ТО 15504-2), согласована с ИСО/МЭК 12207:1995 «Информационная технология. Процессы жизненного цикла программных средств». Однако эта эталонная модель может быть использована для разработки модели оценки уровня зрелости процессов других видов деятельности. Изложенные в ИСО/МЭК ТО 15504 рабочие продукты процессов и их характеристики могут быть использованы для оценки и усовершенствования процессов любого вида деятельности. ИСО/МЭК ТО 15504 предоставляет механизм включения в этот перечень дополнительных рабочих продуктов и процессов, необходимых для осуществления конкретного вида деятельности и, следовательно, для оценки уровня зрелости этих дополнительных процессов и вида деятельности в целом.

Концептуальные основы системы административного управления качеством определены в восьми принципах. ИСО определяет принцип административного управления качеством как «всеобъемлющее и фундаментальное правило или убеждение, применяемое при руководстве и управлении организацией, направленное на непрерывное и долгосрочное улучшение ее производительности путем ориентации на потребителей одновременно с удовлетворением потребностей остальных участвующих сторон». Эти принципы административного управления качеством войдут в документ ISO 9000 2000 года. На этих принципах будет базироваться семейство стандартов ISO 9000.

Выполнение этих принципов будет способствовать повышению управленческой культуры, проникновению системы административного управления качеством во все виды деятельности организации (т. е. выполнения концепции всеобщего административного управления качеством, TQM – Total Quality Management) и, как следствие, обеспечению конкурентоспособности создаваемой организацией продукции, проектов, систем и услуг.

Принцип 1. Ориентация организации на потребителя (Customer-Focused Organization)

«Организации зависят от своих потребителей и, таким образом, должны понимать текущие и будущие потребности потребителей, удовлетворять их требования и стремиться превзойти их ожидания».

Применение принципа ориентации организации на потребителя приводит к следующим действиям:

понимание всего спектра потребностей и ожиданий потребителей относительно продуктов, поставок, цен, надежности и т. д.;

обеспечение взвешенного подхода к потребностям и ожиданиям потребителей и других участвующих сторон (владельцев, персонала, поставщиков, местных сообществ и общества в целом);

распространение информации об этих потребностях и ожиданиях во всей организации;

измерение степени удовлетворенности потребителей и влияние на результат;

управление взаимоотношениями с потребителями.

Полезные применения данного принципа включают:

обеспечение того, что потребности потребителей и других участвующих сторон осознаются всей организацией, для формулировки политики и стратегии (policy and strategy formulation);

обеспечение непосредственной связи соответствующих целей и плановых показателей с потребностями и ожиданиями потребителей, для установления целей и плановых показателей (goal and target setting);

повышение производительности организации для удовлетворения потребностей потребителей, для управления операциями (operational management);

обеспечение того, что персонал обладает знаниями и опытом, требующимися для удовлетворения потребителей организации, для управления людскими ресурсами (human resource management).

Принцип 2. Лидерство (Leadership)

«Лидеры организаций обеспечивают единство назначения и направления организации. Они должны создать и поддерживать внутреннюю окружающую среду, в которой люди могут в полной мере участвовать в достижении стратегических целей организации».

Применение принципа лидерства приводит к следующим действиям:

действенность и личный пример;

понимание изменений во внешней окружающей среде и реагирование на них;

внимание к потребностям всех участвующих сторон, включая потребителей, владельцев, персонал, поставщиков, местные сообщества и общество в целом;

выработка ясного видения будущего организации;

выработка общих ценностей и этики на всех уровнях организации;

установление доверия и искоренение страха;

обеспечение персонала требуемыми ресурсами и свободой, необходимыми для того, чтобы действовать ответственно и обоснованно;

воодушевление, поощрение и признание вклада персонала;

содействие открытому и честному общению;

обучение, подготовка и инструктирование персонала;

выработка достойных целей и плановых показателей;

реализация стратегии по достижению этих целей и плановых показателей.

Полезные применения данного принципа включают:

выработку и распространение ясного видения будущего организации, для формулировки политики и стратегии;

преобразование видения организации в измеримые цели и плановые показатели, для установления целей и плановых показателей;

стратегические цели организации достигаются полномочным и вовлеченным персоналом, для управления операциями;

наличие полномочной, мотивированной, хорошо информированной и стабильной рабочей силы, для управления людскими ресурсами.

Принцип 3. Вовлечение персонала (Involvement of People)

«Люди составляют сущность организации на всех уровнях, и их полная вовлеченность способствует применению их способностей на благо организации».

Применение принципа вовлечения персонала приводит к следующим действиям персонала:

принятие на себя задач и ответственность за их решение;

активный поиск возможностей усовершенствования;

активный поиск возможностей повышения собственных квалификации, знаний и опыта;

свободный обмен знаниями и опытом в группах и коллективах;

концентрация на создании ценностей для потребителя;

новаторство и творчество в дальнейшем продвижении стратегических целей организации;

олицетворение организации перед лицом потребителей, местных сообществ и общества в целом;

получение удовольствия от своей работы;

гордость и удовлетворение быть частью организации.

Полезные применения данного принципа включают:

персонал, эффективно участвующий в усовершенствовании политики и стратегии организации, для формулировки политики и стратегии;

персонал, принимающий на себя задачи и разделяющий ответственность за их решение, для установления целей и плановых показателей;

персонал, вовлеченный в соответствующие усовершенствования решений и процессов, для управления операциями;

персонал, в большей степени удовлетворенный своей работой и активно вовлеченный в собственный рост и развитие на благо организации, для управления людскими ресурсами.

Принцип 4. Процессный подход (Process Approach)

«Желаемый результат достигается более эффективно, когда связанные ресурсы и деятельность управляются как процесс».

Применение принципа процессного подхода приводит к следующим действиям персонала:

определение процесса достижения желаемого результата;

выявление и измерение входов и выходов процесса;

выявление интерфейсов процесса с функциями организации;

оценка возможного риска, его последствий и влияния процесса на потребителей, поставщиков и другие участвующие в процессе стороны;

четкое распределение ответственности, полномочий и подотчетности при управлении процессом;

выявление внутренних и внешних потребителей, поставщиков и других участвующих в процессе сторон;

при проектировании процессов уделяется внимание шагам процессов, видам деятельности, потокам, контрольным величинам, потребностям в подготовке персонала, оборудовании, методах, информации, материалах и других ресурсах, необходимых для достижения желаемого результата.

Полезные применения данного принципа включают:

использование определенных процессов во всей организации приведет к более предсказуемым результатам, лучшему использованию ресурсов, сокращению времени цикла и снижению затрат, для формулировки политики и стратегии;

понимание зрелости процессов способствует выработке достойных целей и плановых показателей, для установления целей и плановых показателей;

принятие процессного подхода ко всем операциям приводит к снижению затрат, предотвращению ошибок, контролю за отклонениями, сокращению времени цикла и более предсказуемым выходам, для управления операциями;

установление эффективных по затратам процессов для управления людскими ресурсами, таких как найм, образование и подготовка, способствует приведению этих процессов в соответствие потребностям организации и создает более зрелую рабочую силу, для управления людскими ресурсами.

Принцип 5. Системный подход к административному управлению (System Approach to Management)

«Выявление, понимание и административное управление системой взаимосвязанных процессов для заданной стратегический цели повышает эффективность и результативность организации».

Применение принципа системного подхода к административному управлению приводит к следующим действиям:

определение системы путем выявления или разработки процессов, влияющих на достижение заданной стратегической цели;

структурирование системы так, чтобы достичь заданную стратегическую цель наиболее эффективным способом;

понимание взаимозависимостей между процессами системы;

непрерывное усовершенствование системы посредством измерения и оценки;

предварительное установление ограничений по ресурсам.

Полезные применения данного принципа включают:

создание всеобъемлющих и достойных планов, связывающих входы функций и процессов, для формулировки политики и стратегии;

цели и плановые показатели конкретных процессов приведены в соответствие с ключевыми стратегическими целями организации, для установления целей и плановых показателей;

более широкий взгляд на эффективность процессов, что приводит к пониманию причин проблем и своевременным действиям по усовершенствованию, для управления операциями;

лучшее понимание распределения ролей и ответственности при достижении общих стратегических целей, уменьшая таким образом межфункциональные барьеры и улучшая коллективную работу, для управления людскими ресурсами.

Принцип 6. Непрерывное усовершенствование (Continual Improvement)

«Непрерывное усовершенствование должно быть постоянной стратегической целью организации».

Применение принципа непрерывного усовершенствования приводит к следующим действиям:

превращение непрерывного усовершенствования продуктов, процессов и систем в стратегическую цель каждого сотрудника организации;

применение базовых понятий последовательного (инкрементного) и скачкообразного усовершенствования;

проведение периодических аттестаций степени достижения установленных критериев высшего качества для выявления областей потенциального усовершенствования;

непрерывное повышение эффективности и результативности всех процессов;

поощрение действий, основанных на предотвращении проблем;

предоставление каждому члену организации необходимых образования и подготовки по методам и инструментальным средствам непрерывного усовершенствования, таким, как:

цикл «планирование – исполнение – проверка – корректирующие действия» (Plan – Do – Check – Act);

решение проблем;

реинжиниринг процессов;

нововведение в процессах;

установление мер и целей для направления и отслеживания усовершенствований;

признание усовершенствований.

Полезные применения данного принципа включают:

создание и осуществление более конкурентоспособных бизнес-планов путем объединения непрерывного усовершенствования со стратегическим и бизнес-планированием, для формулировки политики и стратегии;

установление реалистичных и достойных целей усовершенствования и предоставление ресурсов для их достижения, для установления целей и плановых показателей;

вовлечение персонала организации в непрерывное усовершенствование процессов, для управления операциями;

обеспечение всего персонала организации инструментальными средствами, возможностями и мотивацией для совершенствования продуктов, процессов и систем, для управления людскими ресурсами.

Принцип 7. Основанный на фактах подход к принятию решений (Factual Approach to Decision Making)

«Эффективные решения базируются на анализе данных и информации».

Применение данного принципа приводит к следующим действиям:

осуществление измерений и сбор данных и информации, относящихся к стратегической цели;

обеспечение существенной точности, надежности и доступности данных и информации;

анализ данных и информации с применением обоснованных методов;

понимание значения соответствующих статистических методик;

принятие решений и осуществление действий на базе логического анализа, уравновешенного опытом и интуицией.

Полезные применения данного принципа включают:

стратегии, основанные на соответствующих данных и информации более реалистичны и достижимы с большей вероятностью, для формулировки политики и стратегии;

применение соответствующих сравнительных данных и информации для установления реалистичных и достойных целей и плановых показателей, для установления целей и плановых показателей;

данные и информация являются базисом для понимания производительности процессов и систем для направления усовершенствований и предотвращения будущих проблем, для управления операциями;

анализ данных и информации из таких источников, как опросы персонала, предложения сотрудников, целевых групп для направления формулировки политики управления людскими ресурсами, для управления людскими ресурсами.

Принцип 8. Взаимовыгодные отношения с поставщиками (Mutually Beneficial Supplier Relationship)

«Организация и ее поставщики взаимозависимы, и взаимовыгодные отношения повышают способность обоих производить ценности».

Применение принципа взаимовыгодных отношений с поставщиками приводит к следующим действиям:

выявление и выбор ключевых поставщиков;

установление с поставщиками отношений, которые бы уравновешивали краткосрочные выгоды и долгосрочные соображения для организации и общества в целом;

создание ясного и открытого общения;

инициация совместных разработок и усовершенствования продуктов и процессов;

совместное установление ясного понимания потребностей потребителей;

обмен информацией и будущими планами;

признание усовершенствований и достижений поставщиков.

Полезные применения данного принципа включают:

создание конкурентоспособных преимуществ путем организации стратегических союзов или партнерских отношений с поставщиками, для формулировки политики и стратегии;

установление более достойных целей и плановых показателей с помощью вовлечения и участия поставщиков на ранних этапах, для установления целей и плановых показателей;

создание отношений с поставщиками и управление этими отношениями для обеспечения надежных, своевременных и свободных от дефектов поставок, для управления операциями;

выработку и повышение зрелости поставщиков посредством проведения подготовки поставщиков и совместных усилий по усовершенствованию, для управления людскими ресурсами.

Для комплексного обеспечения нормативной базы практической реализации концепции ИСО/МЭК 15504 с учетом принципов системы административного управления качеством необходимо предусмотреть также выполнение требований нижеприведенных международных стандартов:

ISO/TR 10006:1997 Quality management – Guidelines to quality in project management (Административное управление качеством. Руководящие указания по качеству при административном управлении проектами);

ISO 10007:1997 Quality management – Guidelines for configuration management (Административное управление качеством. Руководящие указания административного управления конфигурацией);

ISO/IEC TR 16326:1999 Software engineering – Guide for the application of ISO/IEC 12207 to project management (Программная инженерия. Руково-

дство по приложению ИСО/МЭК 12207 к административному управлению проектами);

ISO/IEC TR 15271:1998 Information technology – Guide for ISO/IEC 12207 (Software Life Cycle Processes) (Информационная технология. Руководство по применению ИСО/МЭК 12207);

ISO/IEC TR 15846:1998 Information technology – Software life cycle processes – Configuration Management (Информационная технология. Процессы жизненного цикла программных средств. Управление конфигурацией);

ISO/IEC 12119:1994 Information technology – Software packages – Quality requirements and testing (Информационная технология. Пакеты программных средств. Требование к качеству и тестирование);

ISO/IEC TR 14759:1999 Software engineering – Mock up and prototype – A categorization of software mock up and prototype models and their use (Программная инженерия. Макетирование и прототипирование);

ISO/IEC 9126:1991 Information technology – Software product evaluation – Quality characteristics and guidelines for their use (Информационная технология. Оценка программного продукта. Характеристики качества и руководящие указания по их применению);

ISO/IEC 14598:1999 Information technology – Software product evaluation – Part 1–5 (Информационная технология. Оценка программной продукции);

ISO/IEC 15026:1998 Information technology – System and software integrity levels (Информационная технология. Уровни целостности программных средств и систем);

ISO/IEC 14764:1999 Information technology – Software maintenance (Информационная технология. Сопровождение программных средств);

ISO/IEC TR 14471:1999 Information technology – Software engineering – Guidelines for the adoption of CASE tools (Информационная технология. Программная инженерия. Руководящие указания по адаптации инструментальных средств CASE);

ISO/IEC TR 9294:1990 Information technology – Guidelines for the management of software documentation (Информационная технология. Руководящие указания по административному управлению программной документацией);

ISO/IEC 6592:2000 Information technology – Guidelines for the documentation of computer-based application systems (Информационная технология. Руководящие указания по документированию прикладных информационных систем);

ISO/IEC 15910:1999 Information technology – Software user documentation process (Информационная технология. Пользовательская документация программных средств);

ISO/IEC 14756:1999 Information technology – Measurement and rating of performance of computer-based software systems (Информационная технология. Измерение и определение рейтинга программных средств информационных систем).

Необходимо рассмотреть возможность применения ИСО/МЭК ТО 15504 в индустрии различных областей промышленности, отвечающих требованиям CALS-технологии (стандарты серии 10303).

Особое внимание необходимо обратить на создание единого и гармонизированного понятийно-терминологического аппарата в области индустрии программной продукции и информационных технологий.

1.3.3. ГОСТ Р ИСО/МЭК 12207–99. Информационная технология. Процессы жизненного цикла программных средств

Разработан Всероссийским научно-исследовательским институтом стандартизации (ВНИИстандарт) Госстандарта России. Настоящий стандарт содержит полный аутентичный текст международного стандарта ИСО/МЭК 12207–95 «Информационная технология. Процессы жизненного цикла программных средств».

1.3.3.1. Введение

Программные средства являются неотъемлемыми частями информационных технологий и традиционных систем, таких как транспортные, военные, финансовые и система здравоохранения. При этом подразумевается усиление роли стандартов, процедур, методов, средств (инструментария) и внешних условий для разработки и сопровождения программных средств (программного обеспечения). Подобная многоплановость подходов создает значительные трудности при управлении программными средствами и в технологиях программирования, особенно при интеграции продуктов и услуг. Требуется определенное упорядочение вопросов создания программных средств при переходе от подобной многоплановости к общей структуре, которая может быть использована профессионалами для «разговора на одном языке» при создании и управлении программными средствами. Настоящий стандарт устанавливает такую общую структуру.

Данная структура охватывает жизненный цикл программных средств от концепции замыслов через определение и объединение процессов до заказа и поставки программных продуктов и услуг. Кроме того, данная структура предназначена для контроля и модернизации данных процессов.

Процессы, определенные в настоящем стандарте, образуют множество общего назначения. Конкретная организация, в зависимости от своих целей, может выбрать соответствующее подмножество процессов для выполнения своих конкретных задач. Поэтому настоящий стандарт следует адаптировать для конкретной организации, проекта или приложения. Настоящий стандарт предназначен для использования как в случае отдельно поставляемых программных средств, так и для программных средств, встраиваемых или интегрируемых в общую систему.

1.3.3.2. Область применения

Назначение. Настоящий стандарт устанавливает, используя четко определенную терминологию, общую структуру процессов жизненного цикла программных средств, на которую можно ориентироваться в программной индустрии. Настоящий стандарт определяет процессы, работы и задачи, которые используются: при приобретении системы, содержащей программные средства, или отдельно поставляемого программного продукта; при оказании программной услуги, а также при поставке, разработке, эксплуатации и сопровождении программных продуктов. Понятие программных средств также охватывает программный компонент программно-аппаратных средств.

Настоящий стандарт также определяет процесс, который может быть использован при определении, контроле и модернизации процессов жизненного цикла программных средств.

Область распространения. Настоящий стандарт применяется при приобретении систем, программных продуктов и оказании соответствующих услуг, а также при поставке, разработке, эксплуатации и сопровождении программных продуктов и программных компонентов программно-аппаратных средств как в самой организации, так и вне ее. Стандарт содержит также те аспекты описания системы, которые необходимы для обеспечения понимания сути программных продуктов и услуг.

Стандарт также применяется при двусторонних отношениях и может в равной степени применяться, если обе стороны принадлежат к одной и той же организации. Диапазон применения может простираться от неформального соглашения о сотрудничестве до официально заключаемого контракта (договора). Стандарт может использоваться одной из сторон для самоконтроля.

Стандарт не распространяется на готовые программные продукты, если они не входят в поставляемый продукт.

Стандарт предназначен: для заказчиков систем, программных продуктов и услуг; поставщиков; разработчиков; операторов; персонала сопровождения; администраторов проектов; администраторов, отвечающих за качество, и пользователей программных продуктов.

Адаптация настоящего стандарта. Настоящий стандарт определяет набор процессов, работ и задач, предназначенных для адаптации к условиям конкретных программных проектов. Процесс адаптации заключается в исключении неприменяемых в условиях конкретного проекта процессов, работ и задач.

Соответствие. Соответствие настоящему стандарту определяется как выполнение всех процессов, работ и задач, выбранных из настоящего стандарта в процессе адаптации, для конкретного программного проекта. Осуществление процесса или работы считается завершенным, когда выполнены все

требуемые для них задачи в соответствии с предварительно установленными в договоре критериями и требованиями.

Любая организация (например, национальная, промышленная ассоциация, компания), применяющая настоящий стандарт в качестве условия обеспечения торговых сделок, обязана определить и опубликовать минимальный набор требуемых процессов, работ и задач, который обеспечивает проверку соответствия поставщика настоящему стандарту.

Ограничения. Настоящий стандарт описывает архитектуру процессов жизненного цикла программных средств, но не определяет детали реализации или выполнения работ и задач, входящих в данные процессы.

Стандарт не предназначен для определения наименований, форматов или подробного содержания выпускаемой документации. Стандарт может требовать разработки документов одного класса или типа, например различных планов, но не предусматривает, чтобы такие документы разрабатывались или комплектовались раздельно или совместно. Решение этих вопросов оставлено на усмотрение пользователей настоящего стандарта.

Стандарт не предопределяет конкретной модели жизненного цикла или метода разработки программного средства. Пользователи, применяющие настоящий стандарт, должны сами выбирать модель жизненного цикла применительно к своему программному проекту и распределять процессы, работы и задачи, выбранные из настоящего стандарта, на данной модели; выбирать и применять методы разработки программных средств и выполнять работы и задачи, соответствующие конкретному программному проекту.

Стандарт не имеет противоречий с существующими в организациях стратегиями, стандартами или процедурами. Однако любые возникающие конфликтные ситуации должны быть разрешены, а любые противоречащие условия и ситуации должны быть упомянуты в примечаниях как исключения при применении настоящего стандарта.

В тексте настоящего стандарта слово «должны» используется для выражения соглашения между двумя или более сторонами; слово «должна» – для выражения объявления цели или намерения одной из сторон; слово «следует» – для выражения рекомендации из имеющихся возможных вариантов; слово «может» – для обозначения образа действий, допускаемого в рамках ограничений настоящего стандарта.

В тексте настоящего стандарта приведен ряд перечней задач, однако ни один из перечней нельзя считать исчерпывающим, и они приведены в качестве примеров.

1.3.3.3. Прикладное применение настоящего стандарта

В настоящем разделе определяются процессы жизненного цикла программных средств, которые могут быть реализованы при заказе, поставке,

разработке, эксплуатации и сопровождении программных продуктов. Целью настоящего раздела является создание «путеводителя» для пользователей, который поможет ориентироваться в стандарте и осмысленно применять его.

Построение стандарта

В настоящем стандарте работы, которые могут выполняться в жизненном цикле программных средств, распределены по пяти основным, восьми вспомогательным и четырем организационным процессам. Каждый процесс жизненного цикла разделен на набор работ; каждая работа разделена на набор задач. Все процессы жизненного цикла описаны ниже (рис. 1.19).

Рис. 1.19. Структура стандарта

Основные процессы жизненного цикла

Раздел 5 состоит из пяти процессов, которые реализуются под управлением основных сторон, вовлеченных в жизненный цикл программных средств. Под основной стороной понимают одну из тех организаций, которые

инициируют или выполняют разработку, эксплуатацию или сопровождение программных продуктов. Основные стороны – заказчик, поставщик, разработчик, оператор и персонал сопровождения программных продуктов. Основными процессами являются:

1. Процесс заказа (подразд. 5.1). Определяет работы заказчика, т. е. организации, которая приобретает систему, программный продукт или программную услугу.

2. Процесс поставки (подразд. 5.2). Определяет работы поставщика, т. е. организации, которая поставляет систему, программный продукт или программную услугу заказчику.

3. Процесс разработки (подразд. 5.3). Определяет работы разработчика, т. е. организации, которая проектирует и разрабатывает программный продукт.

4. Процесс эксплуатации (подразд. 5.4). Определяет работы оператора, т. е. организации, которая обеспечивает эксплуатационное обслуживание вычислительной системы в заданных условиях в интересах пользователей.

5. Процесс сопровождения (подразд. 5.5). Определяет работы персонала сопровождения, т. е. организации, которая предоставляет услуги по сопровождению программного продукта, состоящие в контролируемом изменении программного продукта с целью сохранения его исходного состояния и функциональных возможностей. Данный процесс охватывает перенос и снятие с эксплуатации программного продукта.

Вспомогательные процессы жизненного цикла

Раздел 6 состоит из восьми процессов. Вспомогательный процесс является целенаправленной составной частью другого процесса, обеспечивающей успешную реализацию и качество выполнения программного проекта. Вспомогательный процесс, при необходимости, инициируется и используется другим процессом. Вспомогательными процессами являются:

1. Процесс документирования (подразд. 6.1). Определяет работы по описанию информации, выдаваемой в процессе жизненного цикла.

2. Процесс управления конфигурацией (подразд. 6.2). Определяет работы по управлению конфигурацией.

3. Процесс обеспечения качества (подразд. 6.3). Определяет работы по объективному обеспечению того, чтобы программные продукты и процессы соответствовали требованиям, установленным для них, и реализовывались в рамках утвержденных планов. Совместные анализы, аудиторские проверки, верификация и аттестация могут использоваться в качестве методов обеспечения качества.

4. Процесс верификации (подразд. 6.4). Определяет работы (заказчика, поставщика или независимой стороны) по верификации программных продуктов по мере реализации программного проекта.

5. Процесс аттестации (подразд. 6.5). Определяет работы (заказчика, поставщика или независимой стороны) по аттестации программных продуктов программного проекта.

6. Процесс совместного анализа (подразд. 6.6). Определяет работы по оценке состояния и результатов какой-либо работы. Данный процесс может использоваться двумя любыми сторонами, когда одна из сторон (проверяющая) проверяет другую сторону (проверяемую) на совместном совещании.

7. Процесс аудита (подразд. 6.7). Определяет работы по определению соответствия требованиям, планам и договору. Данный процесс может использоваться двумя сторонами, когда одна из сторон (проверяющая) контролирует программные продукты или работы другой стороны (проверяемой).

8. Процесс решения проблемы (подразд. 6.8). Определяет процесс анализа и устранения проблем (включая несоответствия), независимо от их характера и источника, которые были обнаружены во время осуществления разработки, эксплуатации, сопровождения или других процессов.

Организационные процессы жизненного цикла

Раздел 7 состоит из четырех процессов. Организационные процессы применяются в какой-либо организации для создания и реализации основной структуры, охватывающей взаимосвязанные процессы жизненного цикла и соответствующий персонал, а также для постоянного совершенствования данной структуры и процессов. Эти процессы, как правило, являются типовыми, независимо от области реализации конкретных проектов и договоров; однако уроки, извлеченные из таких проектов и договоров, способствуют совершенствованию организационных вопросов. Организационными процессами являются:

1. Процесс управления (подразд. 7.1). Определяет основные работы по управлению, включая управление проектом, при реализации процессов жизненного цикла.

2. Процесс создания инфраструктуры (подразд. 7.2). Определяет основные работы по созданию основной структуры процесса жизненного цикла.

3. Процесс усовершенствования (подразд. 7.3). Определяет основные работы, которые организация (заказчика, поставщика, разработчика, оператора, персонала сопровождения или администратора другого процесса) выполняет при создании, оценке, контроле и усовершенствовании выбранных процессов жизненного цикла.

4. Процесс обучения (подразд. 7.4). Определяет работы по соответствующему обучению персонала.

2. АНАЛИЗ ПРОБЛЕМЫ И ПОСТАНОВКА ЗАДАЧИ

2.1. Введение в системный анализ

Системный анализ – система понятий, методов (среди которых должен быть метод декомпозиции) и технологий для изучения, описания, реализации систем различной природы и характера, междисциплинарных проблем; это система общих законов, методов, приемов исследования таких систем.

Любую предметную область также можно определить как системную.

Предметная область – раздел науки, изучающий предметные аспекты системных процессов и системные аспекты предметных процессов и явлений. Это определение можно считать системным определением предметной области.

Пример. Информатика – наука, изучающая информационные аспекты системных процессов и системные аспекты информационных процессов. Это определение можно считать системным определением информатики.

Системный анализ тесно связан с синергетикой.

Синергетика – междисциплинарная наука, изучающая общие идеи, методы и закономерности организации (изменения структуры, ее пространственно-временного усложнения) различных объектов и процессов, инварианты этих процессов. «Синергетика» в переводе с греческого означает «совместный», «согласованно действующий».

Системный анализ тесно связан и с философией. Философия дает общие методы содержательного анализа, а системный анализ дает общие методы формального, межпредметного анализа предметных областей, выявления и описания, изучения их системных инвариантов.

Можно дать и философское определение системного анализа: системный анализ – это прикладная диалектика.

Системный анализ предоставляет к использованию в различных науках, системах следующие методы и процедуры:

абстрагирование и конкретизация;
анализ и синтез;
индукция и дедукция;
формализация;
структурирование;
макетирование;
алгоритмизация;
моделирование;
программное управление;
распознавание, классификация и идентификация образов;
экспертное оценивание и тестирование и другие методы и процедуры.

2.1.1. Системные ресурсы

Имеются следующие основные типы ресурсов в природе и в обществе.

Вещество – наиболее хорошо изученный ресурс, который в основном представлен таблицей Д. И. Менделеева и пополняется не так часто. Вещество выступает как отражение постоянства материи в природе, как мера однородности материи.

Энергия – не полностью изученный тип ресурсов, например, мы не владеем управляемой термоядерной реакцией. Энергия выступает как отражение изменчивости материи, переходов из одного вида в другой, как мера необратимости материи.

Информация – мало изученный тип ресурсов. Информация выступает как отражение порядка, структурированности материи, как мера порядка, самоорганизации материи (и социума). Сейчас под этим термином мы будем понимать некоторые сообщения, ниже мы рассмотрим его более детально.

Человек – выступает как носитель интеллекта высшего уровня и является в экономическом, социальном, гуманитарном смысле важнейшим и уникальным ресурсом общества, выступает как мера разума, интеллекта и целенаправленного действия, мера социального начала, высшей формы отражения материи (сознания).

Организация (или организованность) выступает как форма ресурсов в социуме, группе, которая определяет его структуру, включая институты человеческого общества и его надстройки, выступает как мера упорядоченности ресурсов. Организация системы связана с наличием некоторых причинно-следственных связей в этой системе. Организация системы может иметь различные формы, например, биологическую, информационную, экологическую, экономическую, социальную, временную, пространственную и она определяется причинно-следственными связями в материи и социуме.

Пространство – мера протяженности материи (события), распределения ее (его) в окружающей среде.

Время – мера обратимости (необратимости) материи, событий. Время неразрывно связано с изменениями действительности.

Можно говорить о различных полях, в которые «помещен» любой человек: материальном, энергетическом, информационном, социальном, – их пространственных и временных характеристиках.

Пример. Рассмотрим простую задачу – пойти утром на занятия в вуз. Эта часто решаемая студентом задача имеет все аспекты:

материальный, физический аспект – студенту необходимо переместить некоторую массу, например, учебников и тетрадей на нужное расстояние;

энергетический аспект – студенту необходимо иметь и затратить нужное количество энергии на перемещение;

информационный аспект – необходима информация о маршруте движения и месторасположении вуза, кроме того, нужно обрабатывать информацию по пути своего движения;

человеческий аспект – перемещение, в частности переезд на автобусе, невозможен без человека, например, без водителя автобуса;

организационный аспект – необходимы подходящие транспортные сети и маршруты, остановки и т. д.;

пространственный аспект – перемещение на определенное расстояние;

временной аспект – на данное перемещение будет затрачено время (за которое произойдут соответствующие необратимые изменения в среде, в отношениях, в связях).

Все типы ресурсов тесно связаны. Более того, они невозможны друг без друга: актуализация одного из них ведет к актуализации другого.

Пример. При сжигании дров в печке выделяется тепловая энергия, тепловая энергия используется для приготовления пищи, пища используется для получения биологической энергии организма, биологическая энергия используется для получения информации (например, решения некоторой задачи), перемещения во времени и в пространстве. Человек и во время сна расходует свою биологическую энергию на поддержание информационных процессов в организме; более того, сон – продукт таких процессов.

Социальная организация и активность людей совершенствуют информационные ресурсы, процессы в обществе, последние, в свою очередь, совершенствуют производственные отношения.

Если классическое естествознание объясняет мир исходя из движения, взаимопревращений вещества и энергии, то сейчас реальный мир, объективная реальность могут быть объяснены лишь с учетом сопутствующих системных, особенно, системно-информационных процессов.

2.2. Анализ проблемы и моделирование предметной области с использованием системного подхода

2.2.1. Основные положения

Анализ проблем – это процесс осознания реальных проблем и потребностей пользователей и предложения решений, позволяющих удовлетворить эти потребности.

Цель анализа проблемы состоит в том, чтобы добиться лучшего понимания решаемой проблемы до начала разработки.

Чтобы выявить причины (или проблемы, стоящие за проблемой), необходимо опросить людей, которых непосредственно затрагивает данная проблема. Выявление актантов системы является ключевым шагом в анализе проблемы.

При этом необходимо проанализировать и понять область проблемы и исследовать разнообразные области решений. Как правило, возможных решений множество, и нам необходимо найти то, которое наиболее соответствует решаемой проблеме.

Чтобы иметь возможность провести анализ проблемы, полезно определить, что же собой представляет проблема. По определению Гауса и Вайн-

берга (Cause, Weinberg, 1989) проблема – это разница между желаемым и воспринимаемым.

Это определение достаточно разумно, по крайней мере, оно устраняет часто встречающееся среди разработчиков заблуждение, что подлинная проблема заключается в том, что пользователь не понимает, в чем состоит проблема! Согласно данному определению, если пользователь ощущает нечто как проблему, это и есть настоящая проблема, и она достойна внимания.

При анализе проблемы необходимо осуществить следующие пять этапов:

достигнуть соглашения об определении проблемы;

2) выделить основные причины – проблемы, стоящие за проблемой;

3) выявить заинтересованных лиц и пользователей;

4) определить границу системы решения;

5) выявить ограничения, которые необходимо наложить на решение.

Этап 1. Достижение соглашения об определении проблемы

Один из простейших способов заключается в том, чтобы просто записать проблему и выяснить, все ли согласны с такой постановкой (табл. 2.1).

В рамках этого процесса зачастую полезно рассмотреть преимущества предлагаемого решения, причем их следует описывать на языке клиентов/пользователей. Это обеспечивает дополнительную содержательную основу для понимания реальной проблемы. Рассматривая с точки зрения клиента эти преимущества, мы также достигаем лучшего понимания их взгляда на проблему в целом.

Таблица 2.1

Стандартная форма постановки проблемы

Элемент	Описание
Проблема	[Описание проблемы]
Воздействует на	[Указание лиц, на которых оказывает влияние данная проблема]
Результатом чего является	[Описание воздействия данной проблемы на заинтересованных лиц и бизнес-деятельность]
Выигрыш от	[Указание предлагаемого решения]
Может состоять в следующем	[Список основных предоставляемых решением преимуществ]

2.2.3. Этап 2. Выделение основных причин, стоящих за проблемой

Для понимания реальной проблемы и ее причин можно использовать множество методов. Одним из них является метод анализа корневых причин,

представляющий собой семантический способ нахождения причин, лежащих в основе рассматриваемой проблемы или ее проявления.

Рассмотрим реальный пример. Компания GoodsAreUs, занимающаяся торговлей по каталогу, производит и рассылает на дом множество недорогих товаров различных наименований. Решив заняться проблемой недостаточной прибыльности, компания использует рекомендуемую ей программой обеспечения качества методику «качество – во всем» (total quality management, TQM). Применив данный подход, компания практически сразу обратила внимание на ущерб от несоответствия (cost of nonconformance), который представляет собой стоимость всего, что идет не так, как надо, и приводит к бесполезным затратам. Этот ущерб включает в себя переделки, остатки, неудовлетворенность клиента, текучесть кадров и другие негативные факторы. Проанализировав ущерб от несоответствия, компания заподозрила, что наибольший вклад в него вносят «остатки».

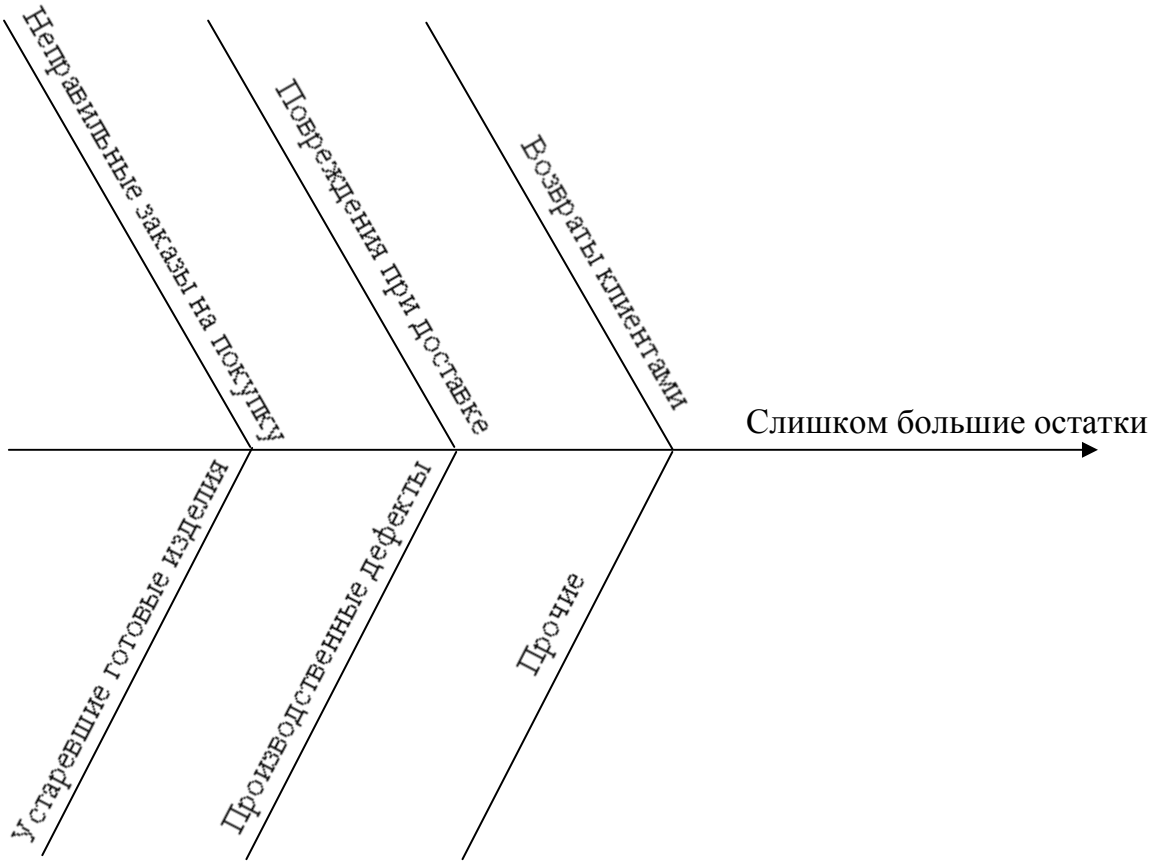


Рис. 2.1. Диаграмма в форме рыбного скелета для отображения корневых причин

Следующим шагом должно стать определение того, какие факторы оказывают влияние на величину остатков. TQM советует для обнаружения проблем, стоящих за проблемой, использовать диаграмму в форме рыбного скелета (рис. 2.1). В нашем случае компания выявила много источников, вносящих свой вклад в остатки. Каждый источник указан как одна из «косточек» на диаграмме.

Способ выявления корневых причин зависит от конкретного случая. Существует несколько способов выявления причин:

опрос сотрудников, непосредственно занимающихся этим делом;
«мозговой штурм» с участием тех, кто знаком с данной областью;
метод упрощенной спецификации приложений;
совместная разработка приложений;
пользовательский сценарий и сеансы разработки схем выбора.

2.2.3.1. Устранение корневых причин

Качественные данные свидетельствуют, что многие корневые причины просто не стоят того, чтобы их устранять, поскольку затраты на их устранение превысят причиняемый проблемой ущерб.

Предложение заменить существующую систему или разработать новую, должно быть хорошо аргументированным. Для этого нужно предоставить стоимостное обоснование такой системы, оценив затраты на разработку и дивиденды от эксплуатации этой системы.

Продолжая анализ, можно применить новую диаграмму в виде рыбного скелета для определения того, какие типы ошибок вносят наибольший вклад в проблему неправильности заказов. Полученные данные можно затем использовать для определения функций новой системы программного обеспечения, которая призвана устранить эти ошибки. Раз мы приняли решение, что проблема неправильных заказов на покупку достойна решения, можно создать для нее формальную постановку (табл. 2.2).

Таблица 2.2

Постановка проблемы ввода заказов на покупку

Элементы	Описание
Проблема воздействует на	Выполнение неправильных заказов на покупку, выполняющий заказы персонал, клиентов, производство, продажи и обслуживание клиентов
Результатом чего является	Увеличение остатков, повышение производственных затрат, неудовлетворенность клиентов, уменьшение прибыли
Выигрыш от	Новой системы, направленной на решение данной проблемы
Может состоять в следующем	Повышение точности заказов на покупку в точке ввода, совершенствование учета данных о покупках, в конечном счете – увеличение прибыли

После постановки проблемы, ее следует передать для ознакомления заказчикам и заинтересованным лицам, чтобы они внесли свои комментарии. Затем постановка проблемы доводится до сведения всех членов команды разработчиков с тем, чтобы все работали в направлении достижения общей цели.

2.2.4. Этап 3. Выявление заинтересованных лиц и пользователей

При решении любой сложной проблемы, как правило, приходится удовлетворять потребности различных групп заинтересованных лиц. Эти группы обычно имеют различные точки зрения на проблему и различные потребности, которые должны быть учтены в решении.

Заинтересованные лица – это все, на кого реализация новой системы или приложения может оказать материальное воздействие.

Понимание потребностей пользователей и других заинтересованных лиц является ключевым фактором в выработке успешного решения.

Первая категория заинтересованных лиц – это пользователи системы. Их потребности легко учесть, поскольку они будут непосредственно привлекаться к определению и использованию системы. Вторую категорию составляют непрямые пользователи, а также те, на кого воздействуют только бизнес-последствия разработки и те, кто воздействует на разработку. Потребности заинтересованных лиц, не являющихся пользователями, также необходимо выявить и учесть.

В зависимости от того, в какой предметной области работает команда, выявление заинтересованных лиц может оказаться как тривиальным, так и нетривиальным этапом анализа проблемы. Часто достаточно провести простой опрос среди тех, кто принимает решения, а также опросить потенциальных пользователей и другие заинтересованные стороны. В этом процессе могут оказаться полезными следующие вопросы.

Кто является пользователями системы?

Кто является заказчиком (экономическим покупателем) системы?

На кого еще окажут влияние результаты работы системы?

Кто будет оценивать и принимать систему, когда она будет представлена и развернута?

Существуют ли другие внутренние или внешние пользователи системы, чьи потребности необходимо учесть?

Кто будет заниматься сопровождением новой системы?

Не забыли ли мы кого-нибудь?

В нашем примере замены системы заказов на покупку основными и наиболее очевидными пользователями являются служащие, занимающиеся вводом заказов на покупку. Они определенно являются заинтересованными лицами, так как их производительность, удобство, комфорт, выполнение работы и ее результаты зависят от системы. Кого еще из заинтересованных лиц можно выделить?

На руководителя отдела приема заказов система также оказывает непосредственное воздействие, но он взаимодействует с системой не напрямую, а посредством различных интерфейсов пользователя и форм отчетов. Главный финансист компании также, очевидно, принадлежит к заинтересованным лицам, так как ожидается, что система повлияет на производительность, качество предоставляемых услуг и прибыльность компании. Наконец, администратор информационной системы и члены команды, разрабатывающей при-

ложение, также являются заинтересованными лицами, так как они будут отвечать за разработку и сопровождение системы. Они также, как и пользователи, будут зависеть от поведения системы. Результаты выявления пользователей и заинтересованных лиц новой системы ввода заказов на покупку представлены в табл. 2.3.

Таблица 2.3

Пользователи и лица, заинтересованные в новой системе

Пользователи	Другие заинтересованные лица
Служащие, занимающиеся вводом заказов Руководитель отдела приема заказов Контроль производства Служащий, выписывающий счета	Администратор информационной системы и команда разработчиков Главный финансист Управляющий производством

2.2.5. Этап 4. Определение границ системы-решения

После того как согласована постановка проблемы и выявлены пользователи и заинтересованные лица, можно перейти к *определению системы,* разрабатываемой для решения данной проблемы. Это важный момент, когда необходимо постоянно помнить как о понимании проблемы, так и о свойствах потенциального решения.

Граница системы описывает оболочку, в которой заключена система (рис. 2.2). Информация в виде ввода и вывода передается от находящихся вне системы пользователей системе и обратно. Все взаимодействия с системой осуществляются посредством интерфейсов между системой и внешним миром.

Рис. 2.2. Отношение ввод/система/вывод

Другими словами, если мы собираемся нечто создать или модифицировать – это часть нашего решения, которая находится внутри границы; если нет – это нечто внешнее по отношению к системе. Таким образом, *мы делим мир на два интересующих нас класса*.

1. Наша система.
2. То, что взаимодействует с нашей системой.

Определим то, что взаимодействует с нашей системой, общим понятием «актанты» (actors). Они выполняют некоторые действия, заставляя систе-

му делать ее работу. Актант изображается простой пиктограммой в виде человечка. Его определение выглядит следующим образом.

Актант – это находящееся вне системы нечто (или некто), взаимодействующее с системой.

С помощью данного понятия мы можем проиллюстрировать границы системы.

Во многих случаях границы системы очевидны. Например, однопользовательский персональный планировщик контактов, работающий на автономной платформе Windows 2000, имеет достаточно хорошо определенные границы. Имеется всего один пользователь и одна платформа. Интерфейсы между пользователем и приложением состоят из диалогов, посредством которых пользователь получает доступ к информации системы, и неких выходных сообщений и коммуникационных путей, которые система использует для документирования или передачи этой информации.

Для системы ввода заказов из нашего примера, которая должна быть объединена с уже существующей информационной системой компании, границы не столь очевидны. Аналитик должен определить, будут ли данные использоваться совместно с другими приложениями, должно ли новое приложение распределяться по разным хостам и клиентам, а также кто будет пользователем. Например, должен ли персонал, занятый в производстве, иметь интерактивный доступ к заказам на покупку? Обеспечивается ли контроль качества или функции аудита? Будет ли система выполняться на компьютере-мэйнфрейме или на новом компьютере-клиенте? Должны ли предоставляться специальные отчеты?

Выявление актантов является ключевым аналитическим этапом в анализе проблемы. Ответы на следующие вопросы помогут их обнаружить.

Кто будет поставлять, использовать или удалять информацию из системы?

Кто будет управлять системой?

Кто будет осуществлять сопровождение системы?

Где будет использоваться система?

Откуда система получает информацию?

Какие внешние системы будут взаимодействовать с системой?

Имея ответы на эти вопросы, аналитик может создать блок-схему, описывающую границы системы, пользователей и другие интерфейсы.

2.2.6. Этап 5. Выявление ограничений, налагаемых на решение

Ограничение уменьшает степень свободы, которой мы располагаем при предложении решения.

Каждое ограничение может значительно сузить нашу возможность создать предполагаемое решение. Следовательно, в процессе планирования необходимо тщательно изучить все ограничения. Многие из них могут даже заставить нас пересмотреть изначально предполагавшийся технологический подход.

Необходимо учитывать, что существуют различные источники ограничений (экономические, технические, политические и т. д.). Ограничения могут быть заданы еще до начала работы («Никакой новой аппаратуры!»), но может получиться, что нам действительно придется их выявлять.

Чтобы их выявить, полезно знать, на что следует обратить внимание. В табл. 2.4 указаны возможные источники системных ограничений. Приведены в таблице вопросы помогут выявить большую часть ограничений. Часто полезно получить объяснение ограничения, как для того, чтобы убедиться, что вы поняли его назначение, так и для того, чтобы можно было обнаружить (если такое произойдет), что данное ограничение больше не применимо к вашему решению.

Таблица 2.4

Возможные источники ограничений системы

Источник	Образцы вопросов
Экономический	Какие финансовые или бюджетные ограничения следует учесть? Существуют ли соображения, касающиеся себестоимости и ценообразования? Существуют ли вопросы лицензирования?
Политический	Существуют ли внешние или внутренние политические вопросы, влияющие на потенциальное решение? Существуют ли проблемы в отношениях между подразделениями?
Технический	Существуют ли ограничения в выборе технологий? Должны ли мы работать в рамках существующих платформ или технологий? Запрещено ли использование любых новых технологий? Должны ли мы использовать какие-либо закупаемые пакеты программного обеспечения?
Системный	Будет ли решение создаваться для наших существующих систем? Должны ли мы обеспечивать совместимость с существующими решениями? Какие операционные системы и среды должны поддерживаться?
Эксплуатационный	Существуют ли ограничения информационной среды или правовые ограничения? Юридические ограничения? Требования безопасности? Какими другими стандартами мы ограничены?
График и ресурсы	Определен ли график? Ограничены ли мы существующими ресурсами? Можем ли мы привлекать работников со стороны? Можем ли мы увеличить штат? Временно? Постоянно?

После того как ограничения выявлены, некоторые из них станут требованиями к новой системе. Другие ограничения будут оказывать влияние на ресурсы и планы реализации. Именно при анализе проблемы необходимо выявить потенциальные источники ограничений и понять, какое влияние каждое ограничение окажет на область возможных решений.

Возвратимся к нашему примеру. Ограничения, которые могут налагаться на новую систему ввода заказов на покупку, представлены в табл. 2.5.

Таблица 2.5

Ограничения, налагаемые на систему ввода заказов на покупку

Источник	Ограничение	Объяснение
Эксплуатационный	Копия данных заказа на покупку должна оставаться в унаследованной базе данных в течение одного года	Риск потери данных слишком велик
Системы и операционные системы	Приложение должно занимать на сервере не более 20 мегабайт	Количество доступной памяти сервера ограничено
Средства, выделенные на оборудование	Система должна быть разработана на существующем сервере	Сокращение издержек и поддержка существующих систем
Средства, выделенные на оплату труда персонала	Фиксированный штат; не привлекать работников со стороны	Фиксированные расходы на зарплату по отношению к текущему бюджету
Технические требования	Должна использоваться новая объектно-ориентирован-ная технология	Мы надеемся, что эта технология повысит производительность и надежность ПО

2.3. Методология ARIS

В теории систем различают структуру системы и ее поведение. Структура описывает статику системы, а поведение описывает динамику. Четыре первых представления в ARIS (организационное, функциональное, представ-

ление выходов и представление данных) описывают структуру системы. В процессном представлении устанавливаются связи перечисленных представлений, и описывается динамическое поведение системы. На рис. 2.3 перечисленные представления сгруппированы в пять частей, образующих здание АРИС.

Функциональное представление содержит описание целей, выполняемых функций, перечень отдельных подфункций, а также общие взаимосвязи и связи подчиненности, которые существуют между функциями, целями и факторами успеха.

Организационное представление описывает взаимодействие пользователей и организационных единиц (ОЕ), а также их связи и имеющие к ним отношение структуры. Организационные модели служат для описания иерархической структуры организации, где группируются субъекты ответственности, ОЕ, должностей и средств, выполняющих работу над объектами, а также для многого другого.

В представлении данных описывают информационную среду предприятия (среду обработки данных), события, сообщения и т. д., где неявно фиксируются также объекты в виде информационных услуг.

Представление процессов (управления) используется для описания связей между тремя предшествующими представлениями. Интеграция этих связей в пределах отдельного представления делает возможным учесть все связи без избыточности. Модели представления процессов описывают отношения между отдельными моделями в рамках всего бизнес-процесса. Это позволяет отслеживать все двусторонние и многосторонние отношения между моделями различных видов, а также полностью описать бизнес-процесс.

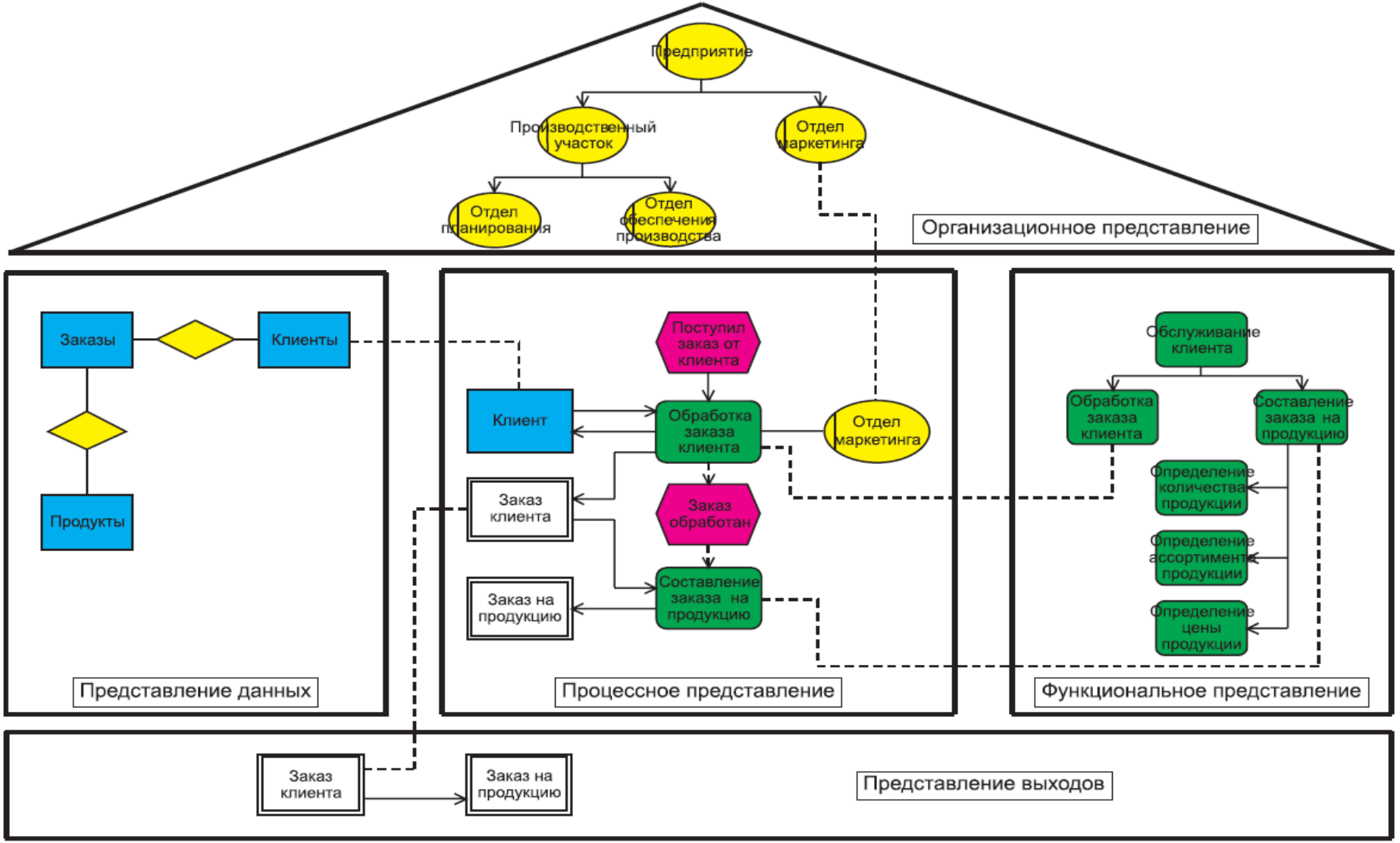


Рис. 2.3. «Здание» ARIS. Представления

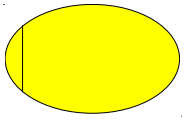
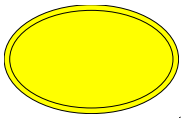
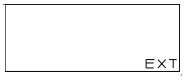
2.3.1. Организационная модель

Организационная диаграмма позволяет всесторонне описать организационно-штатную структуру предприятия. Организационные диаграммы показывают имеющиеся организационные подразделения (как исполнители функций) и их взаимозависимости в соответствии с выбранными критериями структурирования. Отдельные организационные единицы соединяются связями для указания иерархии.

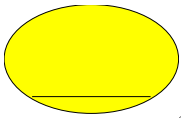
Модель организационной структуры предприятия может содержать следующие структурные элементы: организационная единица, должность или позиция, личность, группа и др. (табл. 2.6).

Таблица 2.6

Элементы организационной диаграммы

№ п/п	Графическая нотация	Наименование	Описание
1		Организационная единица (Organizational unit)	Элемент организационной структуры (структурное подразделение), который отвечает за выполнение определенных задач и преследует определенные цели
2		Позиция или должность (Position)	Наиболее мелкий организационный элемент на предприятии. Обязанности и административные полномочия такого элемента задаются должностными инструкциями
3		Группа (Group)	Несколько сотрудников, которые вместе работают над конкретной задачей в определенный период времени (например, группа проектирования)
4		Внешний сотрудник (External person)	Лицо, выполняющее определенные функции в компании, но не являющееся служащим этой компании. Он может назначаться к организационным единицам или функциям, к выполнению которых он привлекается со стороны или за которые он отвечает
5		Сотрудник (Internal person)	Является служащим компании и может назначаться к организационным единицам или функциям, которые он выполняет или за которые несет ответственность

Окончание табл. 2.6

№ п/п	Графическая нотация	Наименование	Описание
6		Тип сотрудника (Person type)	Обобщает индивидуальных лиц, имеющих одинаковые характеристики. Эти характеристики могут иметь отношение, например, к одним и тем же полномочиям. Начальники отделов или бригадиры, например, должны следовать определенным правилам и выполнять определенные обязанности, которые достаточно описать только один раз
7		Местонахождение (Location)	Определяет физическое положение (местонахождение) организационных единиц, рабочих мест, образцов аппаратных компонентов и технических ресурсов компании. Местонахождение может ссылаться на область, город, предприятие, здание и т. д.

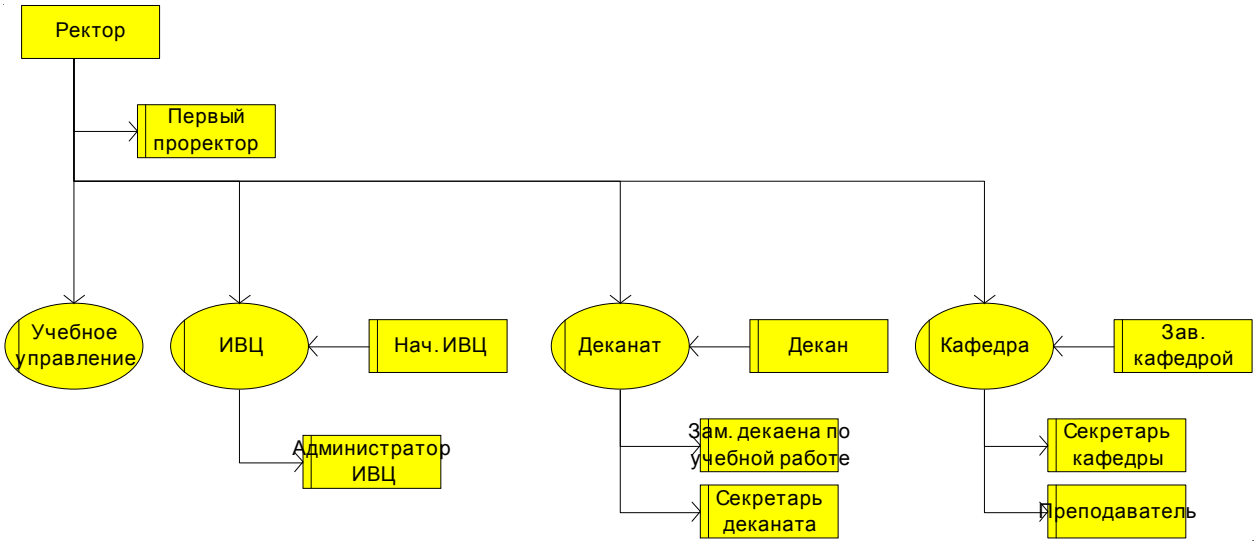


Рис. 2.4. Часть организационной диаграммы КГТУ

При построении модели организационной структуры между структурными элементами могут устанавливаться следующие типы связей: имеет в подчинении (Is superior), связан с (Is assigned to), принадлежит (Belongs to), является организационным управляющим (Is Organization Manager for), имеет

в своем составе (Has member), состоит из (Is composed of), располагает (Is located at), взаимодействует с (Cooperates with), содержит (Subsumes), занимает (Occupies).

Пример организационной диаграммы приведен на рис. 2.4.

2.3.2. Диаграмма цепочки добавленного качества

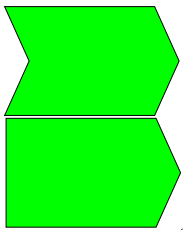
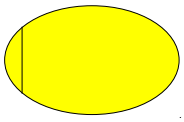
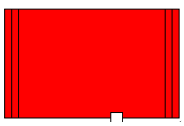
Для уменьшения сложности описания деятельности организации необходимо разработать иерархию моделей бизнес-процессов (БП) организации начиная с самого верхнего уровня и до моделей отдельных БП на нижнем уровне. Для описания модели верхнего уровня используется диаграмма типа *Value-added chain diagram (VAD)*, название которой можно перевести как *Модель цепочки добавленного качества (стоимости)*.

Диаграмма цепочек добавленного качества описывает функции организации, которые непосредственно влияют на реальный выход ее продукции. Эти функции создают последовательность действий, формируя добавленные значения: стоимость, количество, качество и т. д.

Диаграмма цепочек добавленной стоимости может содержать следующие структурные элементы: организационную единицу, должность или позицию, функцию и др. (табл. 2.7).

Таблица 2.7

Элементы диаграммы цепочки добавленного качества

№ п/п	Графическая нотация	Наименование	Описание
1		Функция	Действие или набор действий, выполняемых над исходным объектом (документом, материалом и т. п.) с целью получения заданного результата
2		Технический термин	
3		Организационная единица (Organizational unit)	Элемент организационной структуры (структурное подразделение), который отвечает за выполнение определенных задач и преследует определенные цели
4		Кластер (экземпляр)	Экземпляр объекта данных. Он представляет собой логический взгляд на набор объектов данных или структур

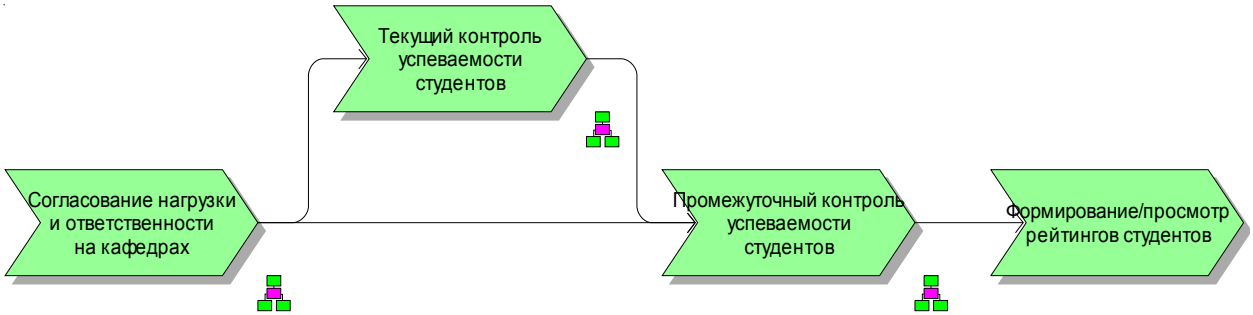


Рис. 2.5. Основные процессы учета успеваемости студентов в системе кредитов

Пример диаграммы цепочек добавленной стоимости представлен на рис. 2.5.

Маленьким значком в нижнем правом углу элемента модели помечаются те функции, для которых существуют модели, раскрывающие эту функцию. Таким механизмом реализуется принцип декомпозиции.

2.3.3. Модели eEPC

Цепочка процесса, управляемая событиями, – Extended event driven process chain (eEPC).

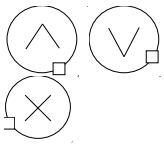
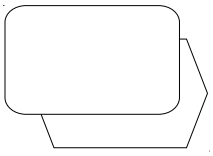
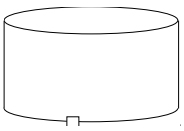
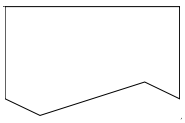
С помощью диаграмм eEPC процедуры БП представляются как логические последовательности событий. События определяют, какое состояние или отношение будет переключать функцию и какое состояние наступит после ее выполнения. Поэтому модель eEPC должна всегда иметь запускающие и завершающие события.

Одно событие может инициировать выполнение одновременно нескольких функций, и, наоборот, функция может быть результатом наступления нескольких событий. Объединения нескольких событий или функций отображаются на диаграмме eEPC с помощью соединителей в виде небольшого кружка. Эти соединители не только отображают графические связи между объектами модели, но и определяют логические связи между ними. Различают два типа связи логических операторов – **связи событий и связи функций**.

Диаграмма цепочек, управляемых событиями, кроме элементов организационной диаграммы и диаграммы данных, может содержать следующие структурные элементы: функцию, событие и др. (табл. 2.8).

Таблица 2.8

Элементы диаграммы цепочек, управляемых событиями

№ п/п	Графическая нотация	Наименование	Описание
1		Функция	Действие или набор действий, выполняемых над исходным объектом (документом, материалом и т. п.) с целью получения заданного результата
2		Событие	Состояние, которое является существенным для управления БП и которое оказывает влияние или контролирует дальнейшее развитие одного или более БП. Изменения состояния отражаются с помощью информационных объектов
3		Правило	Представляет собой правило разветвления и слияния веток БП. Если перейти к рассмотрению каждой отдельной функции БП, то можно сказать, что правило отражает логическое соотношение между несколькими исходными для функции событиями и несколькими результирующими
4		Интерфейс функции	Используется для обозначения функции внешнего БП, который либо не описывается в данной модели, либо является БП другой предметной области
5		Носитель информации	Представляет собой средство хранения информации. Оно может быть реализовано, к примеру, в виде картотеки или компьютерных файлов или БД
6		Документ	Обозначает любой вид документов

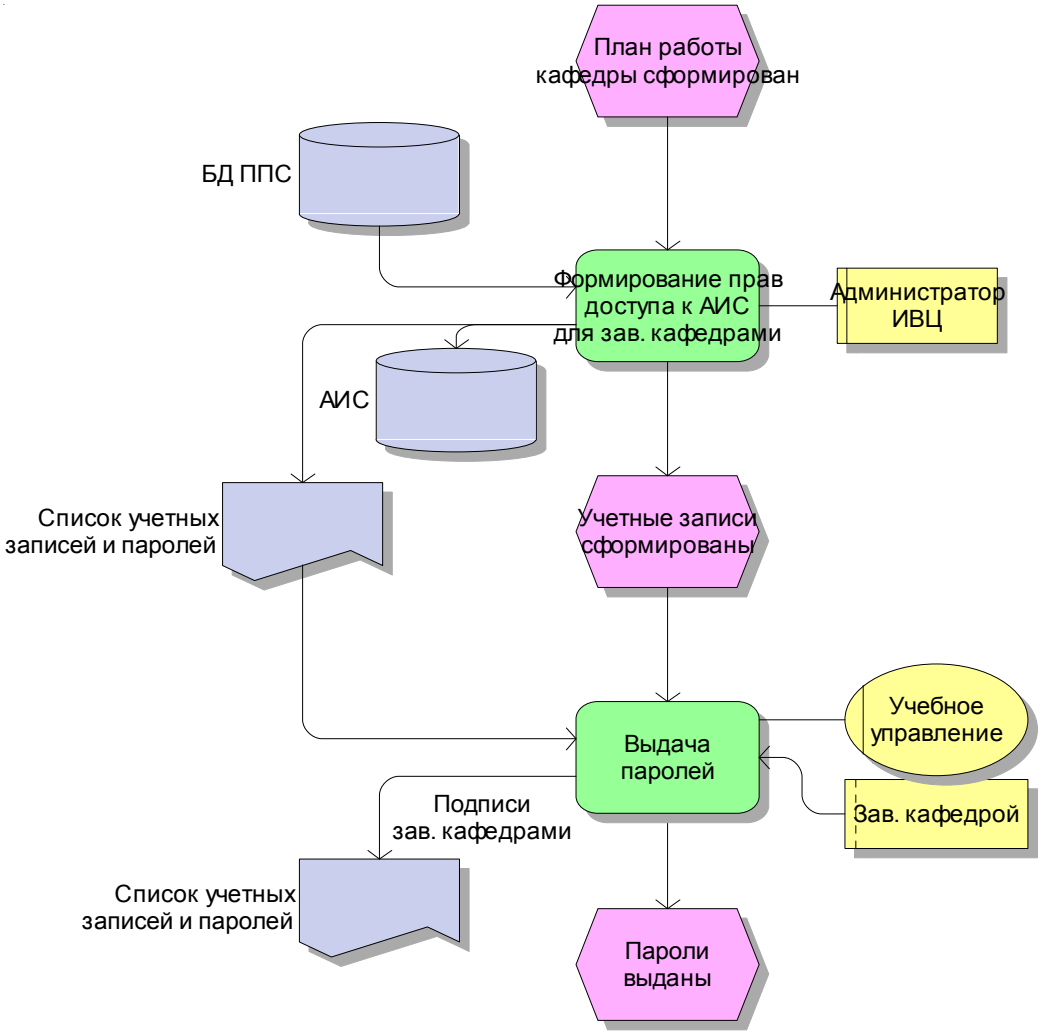


Рис. 2.6. Согласование нагрузки и ответственности на кафедрах

Пример диаграммы цепочек, управляемых событиями, представлен на рис. 2.6.

2.4. Стандарты IDEF0–IDEF3

2.4.1. Методология описания бизнес-процессов IDEF3

IDEF3 – способ описания процессов, основной целью которого является обеспечение структурированного метода, с помощью которого эксперт в предметной области может описать положение вещей как упорядоченную последовательность событий с одновременным описанием объектов, имеющих непосредственное отношение к процессу.

Технология IDEF3 хорошо приспособлена для сбора данных, требующихся для проведения структурного анализа системы. В отличие от большинства технологий моделирования бизнес-процессов, IDEF3 не имеет жестких синтаксических или семантических ограничений, делающих неудобным описание неполных или нецелостных систем. Кроме того, автор модели (системный аналитик) избавлен от необходимости смешивать свои собственные

предположения о функционировании системы с экспертными утверждениями в целях заполнения пробелов в описании предметной области. На рис. 2.7 изображен пример описания процесса с использованием методологии IDEF3.

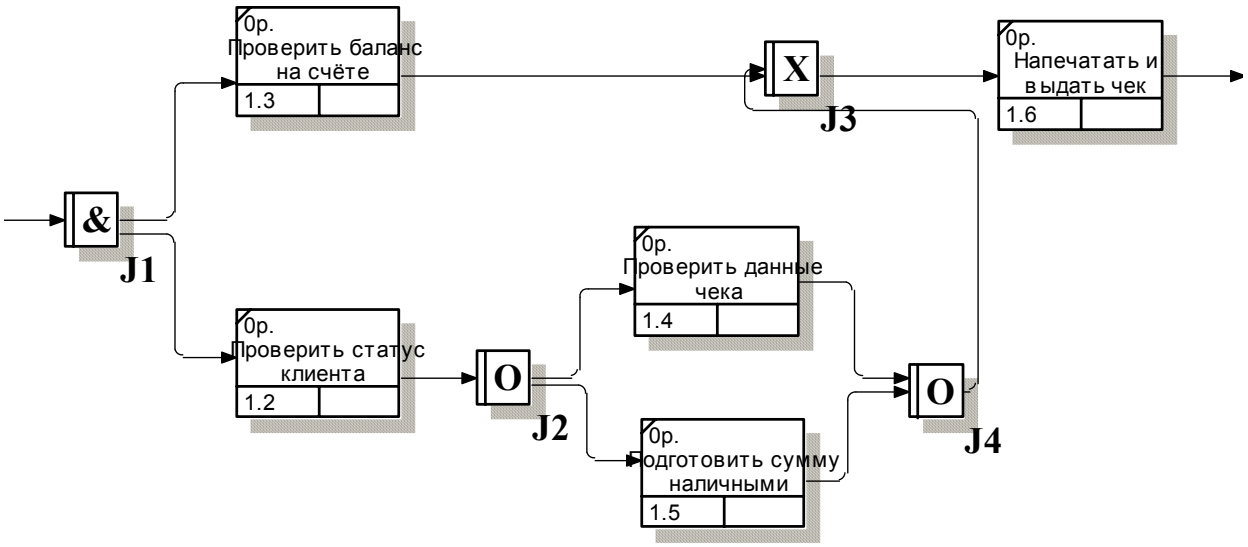


Рис. 2.7. Описание процесса в методологии IDEF3

Технология IDEF3 также может быть использована как метод проектирования бизнес-процессов. IDEF3-моделирование органично дополняет традиционное моделирование с использованием стандарта IDEF0. В настоящее время оно получает все большее распространение как вполне жизнеспособный путь построения моделей проектируемых систем для дальнейшего анализа имитационными методами. Имитационное тестирование часто используют для оценки эксплуатационных качеств разрабатываемой системы, более подробно методы имитационного анализа будут рассмотрены позднее.

2.4.1.1. Синтаксис и семантика моделей IDEF

Модели IDEF3. Основой модели IDEF3 служит так называемый сценарий бизнес-процесса, который выделяет последовательность действий или подпроцессов анализируемой системы. Поскольку сценарий определяет назначение и границы модели, довольно важным является подбор подходящего наименования для обозначения действий. Для подбора необходимого имени применяются стандартные рекомендации по предпочтительному использованию глаголов и отглагольных существительных. Например, «Обработать заказ клиента» или «Применить новый дизайн» – вполне подходящие названия сценариев.

Точка зрения для большинства моделей должна быть явным образом документирована. Обычно это название набора должностных обязанностей человека, являющегося источником информации о моделируемом процессе.

Для системного аналитика также важно понимание цели моделирования – набора вопросов, ответами на которые будет служить модель, границ моделирования (какие части системы войдут в модель, а какие не будут в ней отображены) и целевой аудитории (для кого разрабатывается модель).

Диаграммы. Главной организационной единицей модели IDEF3 является диаграмма. Взаимная организация диаграмм внутри модели IDEF3 особенно важна в случае, когда модель заведомо создается для последующего опубликования или рецензирования, что является вполне обычной практикой при проектировании новых систем. В этом случае системный аналитик должен позаботиться о таком информационном наполнении диаграмм, чтобы каждая из них была самодостаточной и в то же время понятной читателю.

Единица работы. Действие. Аналогично другим технологиям моделирования действие, или в терминах IDEF3 «единица работы» (Unit of Work – UOW), – другой важный компонент модели. Диаграммы IDEF3 отображают действие в виде прямоугольника. Как уже отмечалось, действия именуются с использованием глаголов или отглагольных существительных, каждому из действий присваивается уникальный идентификационный номер. Этот номер не используется вновь даже в том случае, если в процессе построения модели действие удаляется. В диаграммах IDEF3 номер действия обычно предваряется номером его родителя (рис. 2.8).

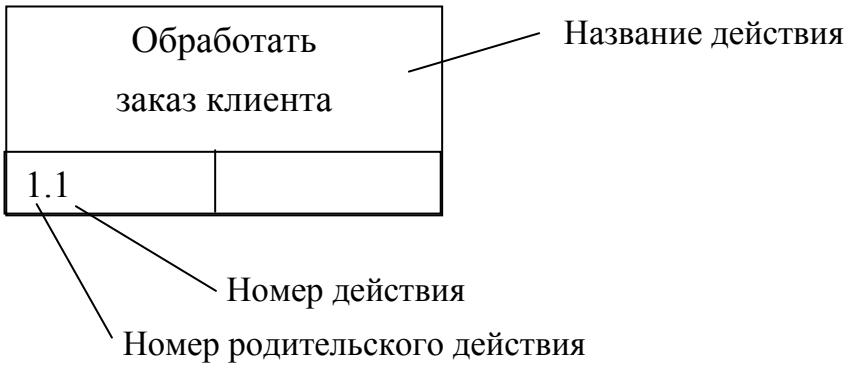


Рис. 2.8. Изображение и нумерация действия в диаграмме IDEF3

Связи. Связи выделяют существенные взаимоотношения между действиями. Все связи в IDEF3 являются однонаправленными, и, хотя стрелка может начинаться или заканчиваться на любой стороне блока, обозначающего действие, диаграммы IDEF3 обычно организовываются слева направо таким образом, что стрелки начинаются на правой и заканчиваются на левой стороне блоков. В табл. 2.9 приведены три возможных типа связей.

Таблица 2.9

Типы связей в модели IDEF3

Изображение	Название	Назначение
→	Временное предшествование (Temporal precedence)	Исходное действие должно завершиться прежде, чем конечное действие сможет начаться
→→	Объектный поток (Object flow)	Выход исходного действия является входом конечного действия. Из этого, в частности, следует, что исходное действие должно завершиться прежде, чем конечное действие сможет начаться
- - - - - - - →	Нечеткое отношение (Relationship)	Вид взаимодействия между исходным и конечным действиями задается аналитиком отдельно для каждого случая использования такого отношения

Связь типа «временное предшествование». Как видно из названия, связи этого типа отражают, что исходное действие должно полностью завершиться, прежде чем начнется выполнение конечного действия. Связь должна быть поименована таким образом, чтобы человеку, просматривающему модель, была понятна причина ее появления. Во многих случаях завершение одного действия инициирует начало выполнения другого (рис. 2.9). В этом примере автор должен принять рекомендации рецензентов, прежде чем начать вносить соответствующие изменения в работу.

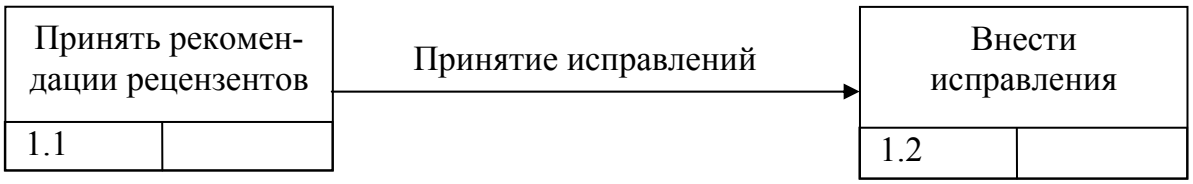


Рис. 2.9. Связь типа предшествование между действиями 1.1 и 1.2

Связь типа «объектный поток». Одной из наиболее часто встречающихся причин использования связи типа «объектный поток» состоит в том, что некоторый объект, являющийся результатом выполнения исходного действия, необходим для выполнения конечного действия. Такая связь отличается от связи временного предшествования двойным концом обозначающей ее стрелки. Наименования потоковых связей должны четко идентифицировать объект, который передается с их помощью. Временная семантика объектных связей аналогична связям предшествования. Это означает, что порождающее объектную связь исходное действие должно завершиться, прежде чем конечное действие начнет выполняться (рис. 2.10). В приведенном примере счет на

оплату услуг является результатом выполнения действия 1.1. Счет необходим для проведения оплаты услуг.

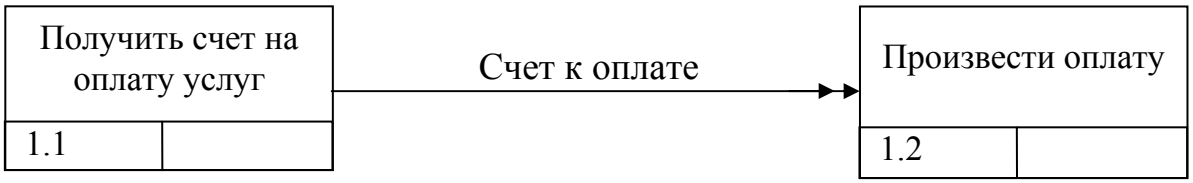


Рис. 2.10. Объектная связь между действиями 1.1 и 1.2

Связь типа «нечеткое отношение». Связи этого типа используются для выделения отношений между действиями, которые невозможно описать с использованием предшественных или объектных связей. Значение каждой такой связи должно быть определено, поскольку связи типа нечеткое отношение сами по себе не предполагают никаких ограничений. Одно из применений нечетких отношений – отображение взаимоотношений между параллельно выполняющимися действиями. Рис. 2.11 иллюстрирует фрагмент процесса запуска бензопилы с водяным охлаждением и нечеткое отношение между действиями запустить двигатель и запустить водяной насос. Название стрелки может быть использовано для описания природы отношения, более подробное объяснение может быть приведено в виде отдельной ссылки.

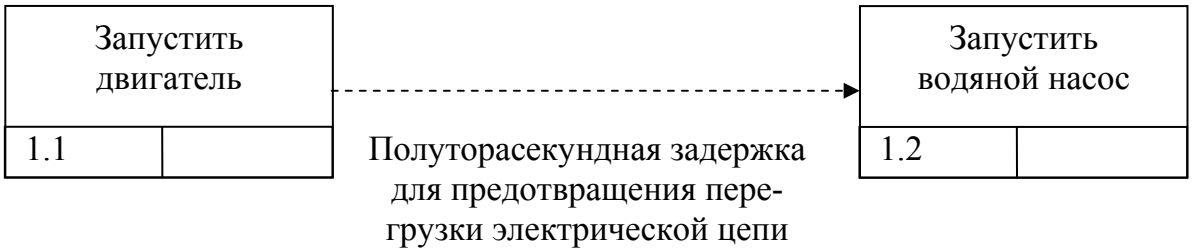


Рис. 2.11. Связь типа «нечеткое отношение»

Наиболее часто нечеткие отношения используются для описания специальных случаев связей предшествования, например для описания альтернативных вариантов временного предшествования. Альтернативная предшественной связи (рис. 2.9) связь нечеткого отношения представлена на рис. 2.12. В этом примере внесение исправлений начинается по мере получения замечаний от рецензентов, т. е. до непосредственного окончания действия по принятию замечаний.

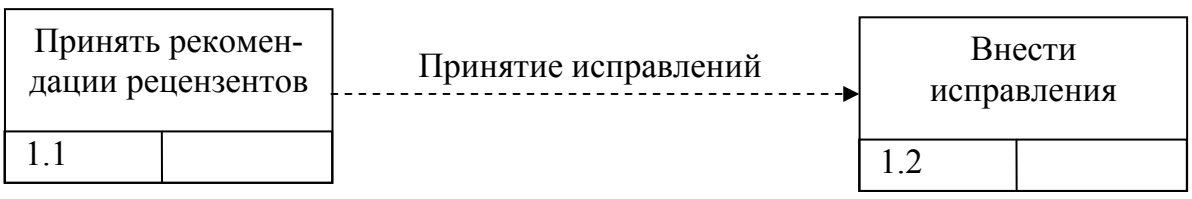


Рис. 2.12. Альтернатива связи предшествования

Соединения. Завершение одного действия может инициировать начало выполнения сразу нескольких других действий, или, наоборот, определенное действие может требовать завершения нескольких других действий для начала своего выполнения. Соединения разбивают или соединяют внутренние потоки и используют для описания ветвления процесса.

Разворачивающие соединения используются для разбиения потока. Завершение одного действия вызывает начало выполнения нескольких других.

Сворачивающие соединения объединяют потоки. Завершение одного или нескольких действий вызывает начало выполнения только одного другого действия.

В табл. 2.10 приведены три типа соединений.

Таблица 2.10

Типы соединений в модели IDEF3

Графическое обозначение	Название	Вид	Правила инициации
&	Соединение «И»	Разворачивающее	Каждое конечное действие обязательно инициируется
		Сворачивающее	Каждое исходное действие обязательно должно завер-
X	Соединение «Исключающее ИЛИ»	Разворачивающее	Одно и только одно конечное действие инициируется
		Сворачивающее	Одно и только одно исходное действие должно за-
O	Соединение «ИЛИ»	Разворачивающее	Одно (или более) конечное действие инициируется
		Сворачивающее	Одно (или более) исходное действие должно завер-

Примеры разворачивающих и сворачивающих соединений приведены на рис. 2.13.

«*И*»-*соединения*. Соединения этого типа инициируют выполнение всех своих конечных действий. Все действия, присоединенные к сворачивающему «И»-соединению, должны завершиться, прежде чем может начать выполняться следующее действие. После обнаружения пожара инициируются включение пожарной сигнализации (рис. 2.14), вызов пожарной охраны и начинается тушение пожара. Запись в журнал производится только тогда, когда все три перечисленных действия завершены.

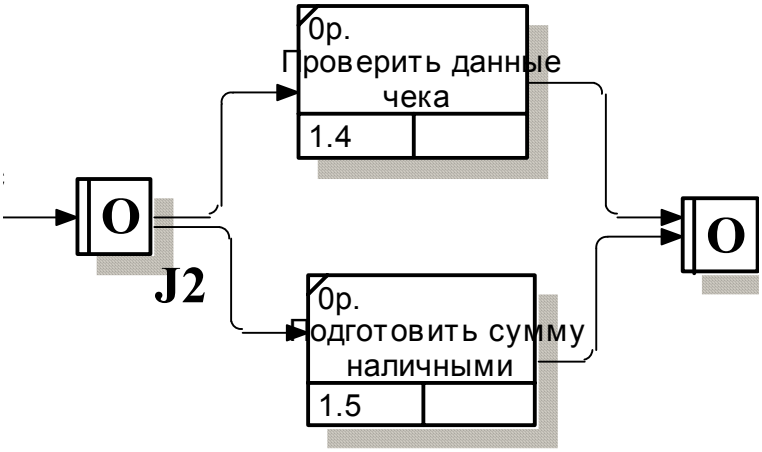


Рис. 2.13. Два вида соединений

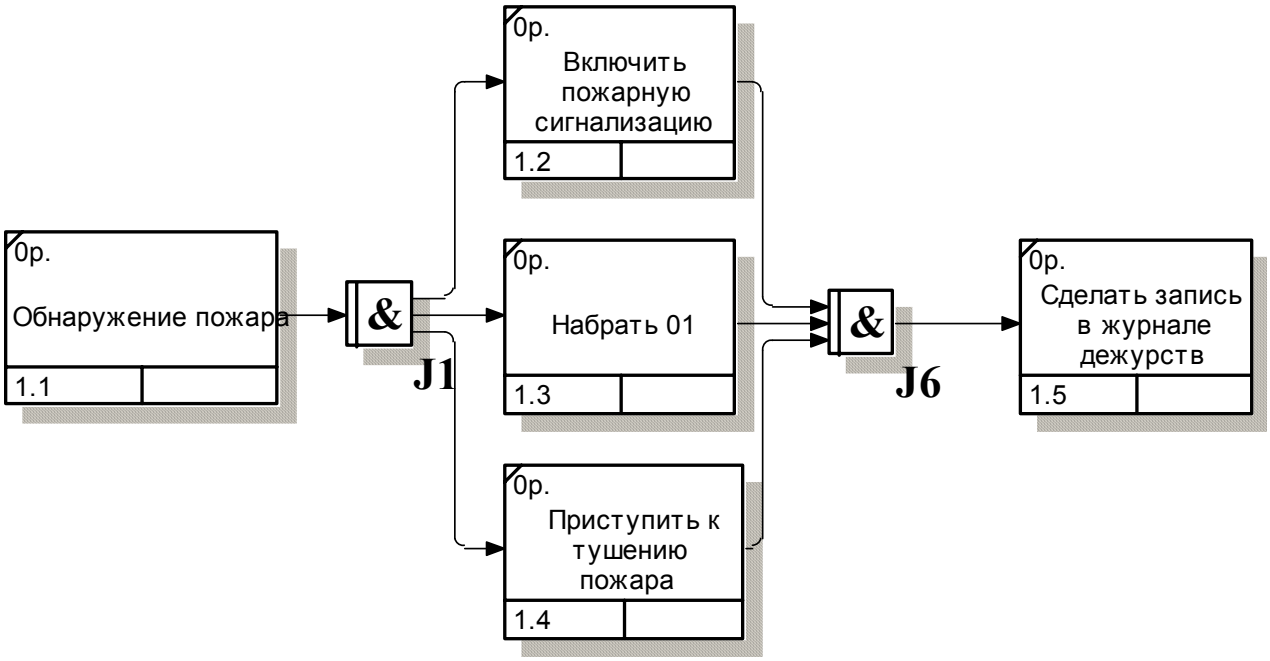


Рис. 2.14. «И»-соединения

Соединение «*Исключающее ИЛИ*». Вне зависимости от количества действий, прицепленных к сворачивающему или разворачивающему соединению «Исключающее ИЛИ», инициировано будет только одно из них, и поэтому только одно из них будет завершено перед тем, как любое действие, следующее за сворачивающим соединением «Исключающее ИЛИ», сможет начаться.

Если правила активации соединения известны, они обязательно должны быть документированы либо в его описании, либо пометкой стрелок, исходящих из разворачивающего соединения (рис. 2.15).

Соединение «Исключающее ИЛИ» используется для отображения того факта, что студент не может одновременно быть направлен на лекции по двум разным курсам.

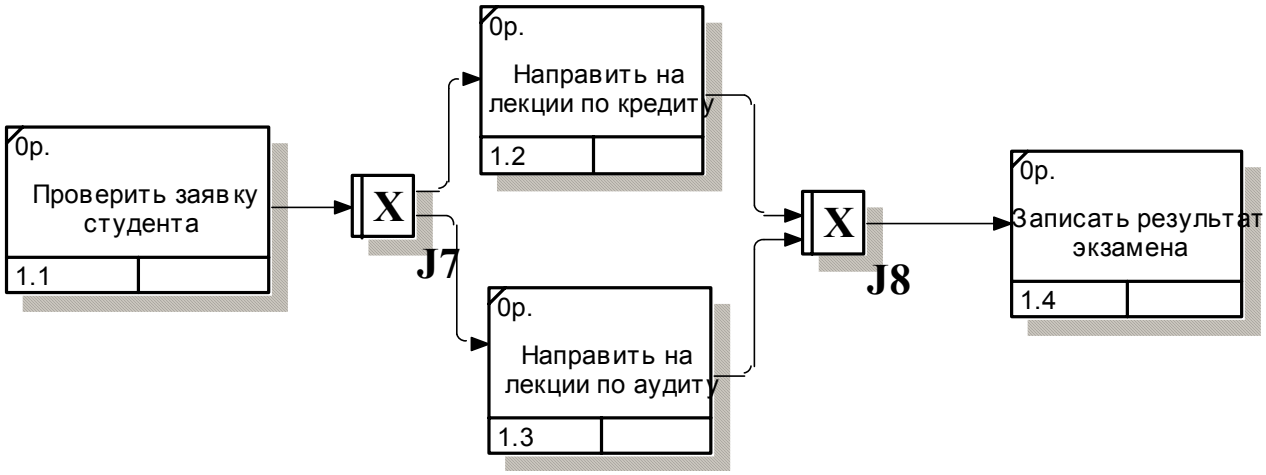


Рис. 2.15. Соединение «Исключающее ИЛИ»

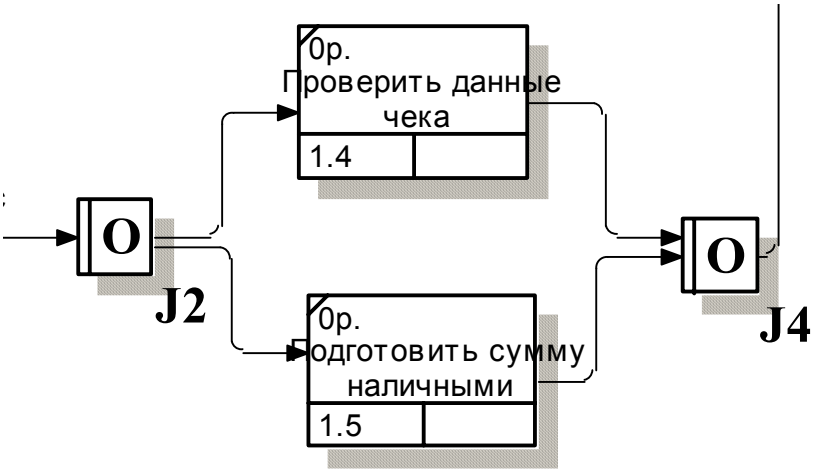


Рис. 2.16. Соединение «ИЛИ»

Соединение «ИЛИ». Соединения этого типа предназначены для описания ситуаций, которые не могут быть описаны двумя предыдущими типами соединений. Аналогично связи нечеткого отношения соединение «ИЛИ» в основном определяется и описывается непосредственно системным аналитиком. Соединение J2 может активировать проверку данных чека и (или) проверку суммы наличных (рис. 2.16). Проверка чека инициируется, если покупатель желает расплатиться чеком, проверка суммы наличных – при оплате наличными. Оба действия инициируются при частичной оплате чеком и частичной – наличными.

Декомпозиция действий. Действия в IDEF3 могут быть декомпозированы или разложены на составляющие, для более детального анализа. Декомпозировать действие можно несколько раз. Это позволяет документировать альтернативные потоки процесса в одной модели.

Для корректной идентификации действий в модели с множественными декомпозициями схема нумерации действий расширяется и наряду с номерами действия и его родителя включает в себя порядковый номер декомпозиции. Например, в номере действия 1.2.5: 1 – номер родительского действия, 2 – номер декомпозиции, 5 – номер действия.

2.4.1.2. Требования IDEF3 к описанию бизнес-процессов

Здесь мы рассмотрим построение IDEF3-диаграммы на основе выраженного в текстовом виде описания процесса. Предполагается, что в построении диаграммы принимают участие ее автор (в основном как системный аналитик) и один или несколько экспертов предметной области, которые будут представлять нам описание процесса.

Определение сценария, границ моделирования, точки зрения. Перед тем как попросить экспертов предметной области подготовить описание моделируемого процесса, должны быть документированы границы моделирования, чтобы экспертам была понятна необходимая глубина и полнота требуемого от них описания. Кроме того, если точка зрения аналитика на процесс отличается от обычной точки зрения для эксперта, это должно быть ясно и аккуратно описано.

Вполне возможно, что эксперты не смогут сделать приемлемое описание без применения формального опроса автором модели. В таком случае автор должен заранее приготовить набор вопросов таким же образом, как журналист заранее подготавливает вопросы для интервью.

Определение действий и объектов. Результатом работы экспертов обычно является текстовый документ, описывающий интересующий аналитика круг вопросов. В дополнение к нему может иметься письменная документация, позволяющая пролить свет на природу изучаемого процесса. Вне зависимости от того, является ли информация текстовой или вербальной, она анализируется и разделяется частями речи для идентификации списка действий (глаголы и отглагольные существительные), составляющих процесс, и объектов (имена существительные), участвующих в процессе.

В некоторых случаях возможно создание графической модели процесса в присутствии экспертов. Такая модель также может быть разработана после сбора всей необходимой информации, что позволяет не отнимать время экспертов на детали форматирования получающихся диаграмм.

Поскольку модели IDEF3 могут одновременно разрабатываться несколькими командами, IDEF3 поддерживает простую схему резервирования

номеров действий в модели. Каждому аналитику выделяется уникальный диапазон номеров действий, что обеспечивает их независимость друг от друга. В табл. 2.11 номера действий выделяются каждому аналитику большими блоками. В этом примере Иван исчерпал данный ему вначале диапазон номеров и дополнительно получил второй.

Таблица 2.11

Распределение диапазонов номеров IDEF3 между аналитиками

Аналитик	Диапазон номеров IDEF3
Иван	1–99
Петр	100–199
Николай	200–299
Иван	300–399

Последовательность и параллельность. Если модель создается после проведения интервью, аналитик должен принять решения по построению иерархии участвующих в модели диаграмм, например, насколько подробно будет детализироваться каждая отдельно взятая диаграмма. Если последовательность или параллельность выполнения действий окончательно не ясна, эксперты могут быть опрошены вторично (возможно, с использованием черновых вариантов незаконченных диаграмм) для получения недостающей информации. Важно, однако, различать предполагаемую (появляющуюся из-за недостатка информации о связях) и явную (ясно указанную в описании эксперта) параллельности.

Итак, IDEF3 – это способ описания бизнес-процессов, который нужен для описания положения вещей как упорядоченной последовательности событий с одновременным описанием объектов, имеющих непосредственное отношение к процессу. IDEF3 хорошо приспособлен для сбора данных, требующихся для проведения структурного анализа системы. Кроме того, IDEF3 применяется при проведении стоимостного анализа поведения моделируемой системы.

2.4.2. Методология функционального моделирования IDEF0

Методология функционального моделирования IDEF0 – это технология описания системы в целом как множества взаимозависимых действий, или функций. Важно отметить функциональную направленность IDEF0 – функции системы исследуются независимо от объектов, которые обеспечивают их выполнение. «Функциональная» точка зрения позволяет четко отделить ас-

пекты назначения системы от аспектов ее физической реализации. На рис. 2.17 приведен пример типовой диаграммы IDEF0.

Наиболее часто IDEF0 применяется как технология исследования и проектирования систем на логическом уровне. По этой причине он, как правило, используется на ранних этапах разработки проекта, до IDEF3-моделирования для сбора данных и моделирования процесса «как есть». Результаты IDEF0-анализа могут применяться при проведении проектирования с использованием моделей IDEF3 и диаграмм потоков данных.

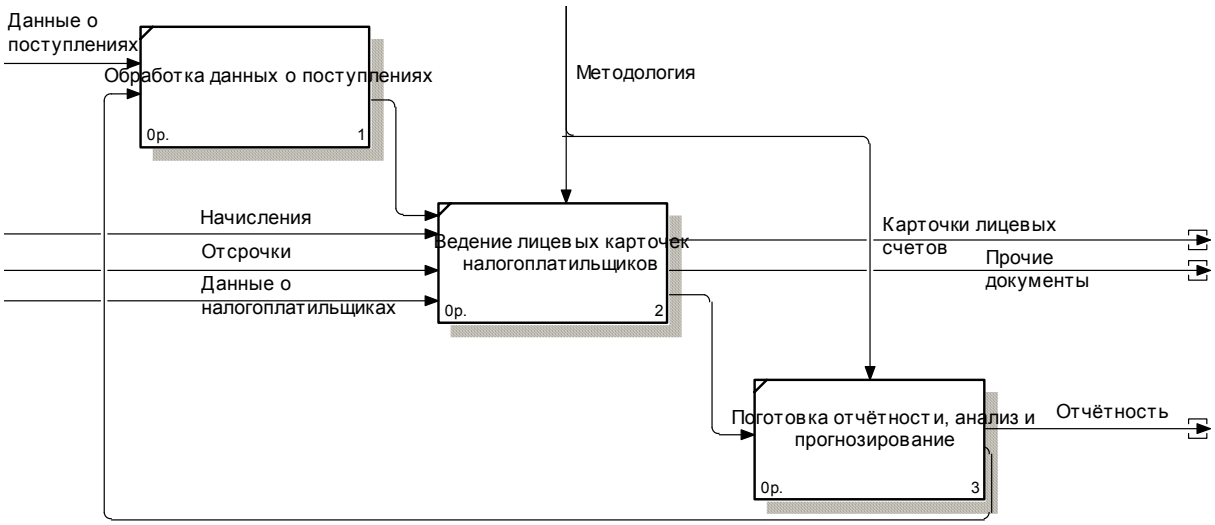


Рис. 2.17. Пример диаграммы IDEF0

2.4.2.1. Синтаксис и семантика моделей IDEF0

Модели IDEF0. IDEF0 сочетает в себе небольшую по объему графическую нотацию (она содержит только два обозначения: блоки и стрелки) со строгими и четко определенными рекомендациями, в совокупности предназначенными для построения качественной и понятной модели системы.

Методология IDEF0 в некоторой степени напоминает рекомендации, существующие в книгоиздательском деле, часто набор напечатанных моделей IDEF0 организуется в брошюру (называемую в терминах IDEF0 комплект), имеющую содержание, глоссарий и другие элементы, характерные для законченной книги.

Первый шаг при построении модели IDEF0 заключается в определении назначения модели – набора вопросов, на которые должна отвечать модель. Набор вопросов можно сравнить с предисловием, в котором раскрывается назначение книги.

Границы моделирования предназначены для обозначения ширины охвата предметной области и глубины детализации и являются логическим продолжением уже определенного назначения модели. Как читающий мо-

дель, так и непосредственно ее автор должны понимать степень детальности ответов на поставленные в назначении модели вопросы.

Следующим шагом указывается предполагаемая целевая аудитория, для нужд которой создается модель. Зачастую от выбора целевой аудитории зависит уровень детализации, с которым должна создаваться модель. Перед построением модели необходимо иметь представление о том, какие сведения о предмете моделирования уже известны, какие дополнительные материалы и (или) техническая документация для понимания модели могут быть необходимы целевой аудитории, какие язык и стиль изложения являются наиболее подходящими.

Под точкой зрения понимается перспектива, с которой наблюдалась система при построении модели. Точка зрения выбирается таким образом, чтобы учесть уже обозначенные границы моделирования и назначение модели. Однажды выбранная точка зрения остается неизменной для всех элементов модели. При необходимости могут быть созданы другие модели, отображающие систему с других точек зрения. Вот несколько примеров точек зрения при построении моделей: клиент, поставщик, владелец, редактор.

Действия. Действие, обычно в IDEF0 называемое функцией, обрабатывает или переводит входные параметры (сырье, информацию и т. п.) в выходные. Поскольку модели IDEF0 представляют систему как множество иерархических (вложенных) функций, в первую очередь должна быть определена функция, описывающая систему в целом – контекстная функция. Функции изображаются на диаграммах как поименованные прямоугольники, или функциональные блоки. Имена функций в IDEF0 подбираются по сходным правилам с именами действий в IDEF3 – с использованием глаголов или отглагольных существительных. Важно подбирать имена таким образом, чтобы они отражали систему так, как если бы она обозревалась с точки зрения, выбранной для моделирования.

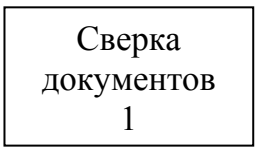


Рис. 2.18. Функциональный блок IDEF0

Пример функционального блока приведен на рис. 2.18.

Выше мы определяли IDEF0-модели как иерархическое множество вложенных блоков. Любой блок может быть декомпозирован на составляющие его блоки. Декомпозицию часто ассоциируют с моделированием «сверху вниз», однако это не совсем верно. Функциональную декомпозицию корректнее определять как моделирование «снаружи вовнутрь», в котором мы рассматриваем систему наподобие луковицы, с которой последовательно снимаются слои.

Границы и связи. Чтобы быть полезным, описание любого блока должно, как минимум, включать в себя описание объектов, которые блок создает в результате своей работы («выхода»), и объектов, которые блок потребляет или преобразует («вход»).

В IDEF0 также моделируются управление и механизмы исполнения. Под управлением понимаются объекты, воздействующие на способ, которым блок преобразует вход в выход. Механизм исполнения – объекты, которые непосредственно выполняют преобразование входа в выход, но не потребляются при этом сами по себе.

Для отображения категорий информации, присутствующих на диаграммах IDEF0, существует аббревиатура ICOM, отображающая четыре возможных типа стрелок:

I (Input) – вход – нечто, что потребляется в ходе выполнения процесса;

C (Control) – управление – ограничения и инструкции, влияющие на ход выполнения процесса;

O (Output) – выход – нечто, являющееся результатом выполнения процесса;

M (Mechanism) – исполняющий механизм – нечто, что используется для выполнения процесса, но не потребляется само по себе. Рис. 2.19 показывает четыре возможных типа стрелок в IDEF0, каждый из типов соединяется со своей стороной функционального блока.

Для названия стрелок, как правило, употребляются имена существительные. Стрелки могут представлять собой людей, места, вещи, идеи или события. Как и в случае с функциональными блоками, присвоение имен всем стрелкам на диаграмме является только необходимым условием для понимания читателем сути изображенного. Отдельное описание каждой стрелки в текстовом виде может оказаться критическим фактором для построения точной и полезной модели.

Рис. 2.19. Соединение стрелок со сторонами функционального блока

Стрелки входа. Вход представляет собой сырье, или информацию, потребляемую или преобразуемую функциональным блоком для производства выхода. Стрелки входа всегда направлены в левую сторону прямоугольника, обозначающего в IDEF0 функциональный блок. Наличие входных стрелок на диаграмме не является обязательным, так как возможно, что некоторые блоки ничего не преобразуют и не изменяют. Примером блока, не имеющего входа, может служить блок «Принятие решения руководством», где для принятия решения анализируется несколько факторов, но ни один из них непосредственно не преобразуется и не потребляется в результате принятия какого-либо решения.

Стрелки управления. Стрелки управления отвечают за регулирование того, как и когда выполняется функциональный блок, и, если он выполняется, какой выход получается в результате его выполнения. Так как управление контролирует поведение функционального блока для обеспечения создания желаемого выхода, каждый функциональный блок должен иметь, как минимум, одну стрелку управления. Стрелки управления всегда входят в функциональный блок сверху.

Управление часто существует в виде правил, инструкций, законов, политики, набора необходимых процедур или стандартов. Влияя на работу блока, оно непосредственно не потребляется и не трансформируется в результате. Может оказаться, что целью функционального блока является как раз изменение того или иного правила, инструкции, стандарта и т. п. В этом случае стрелка, содержащая соответствующую информацию, должна рассматриваться не как управление, а как вход функционального блока.

Управление можно рассматривать как специфический вид входа. В случаях, когда неясно, относить ли стрелку к входу или к управлению, предпочтительно относить ее к управлению до момента, пока неясность не будет разрешена.

Стрелки выхода. Выход – это продукция или информация, получаемая в результате работы функционального блока. Каждый блок должен иметь, как минимум, один выход. Действие, которое не производит никакого четко определяемого выхода, не должно моделироваться вообще (по меньшей мере, должно рассматриваться в качестве одного из первых кандидатов на исключение из модели).

При моделировании непроизводственных предметных областей выходами, как правило, являются данные, в каком-либо виде обрабатываемые функциональным блоком. В этом случае важно, чтобы названия стрелок входа и выхода были достаточно различимы по своему смыслу. Например, блок «Прием пациентов» может иметь стрелку «Данные о пациенте» как на входе, так и на выходе. В такой ситуации входящую стрелку можно назвать «Предварительные данные о пациенте», а исходящую – «Подтвержденные данные о пациенте».

Стрелки механизма исполнения. Механизмы являются ресурсом, который непосредственно исполняет моделируемое действие. С помощью механизмов исполнения могут моделироваться: ключевой персонал, техника и (или) оборудование. Стрелки механизма исполнения могут отсутствовать в случае, если оказывается, что они не являются необходимыми для достижения поставленной цели моделирования.

Комбинированные стрелки. В IDEF0 существует пять основных видов комбинированных стрелок: «Выход-вход», «Выход – управление», «Выход – механизм исполнения», «Выход – обратная связь на управление» и «Выход – обратная связь на вход».

Стрелка «Выход-вход» применяется, когда один из блоков должен полностью завершить работу перед началом работы другого блока. Так, на рис. 2.20 формирование счета должно предшествовать приему заказа.

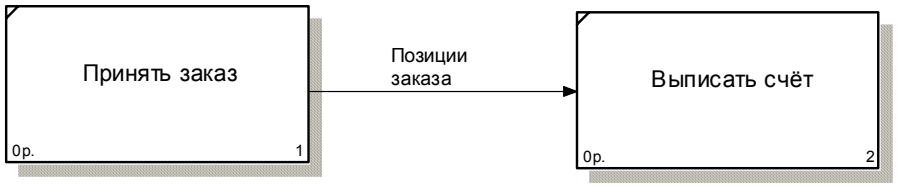


Рис. 2.20. Комбинация стрелок «Выход-вход»

Стрелка «Выход – управление» отражает ситуацию преобладания одного блока над другим, когда один блок управляет работой другого. На рис. 2.21 принципы формирования инвестиционного портфеля управляют поведением брокеров на бирже.

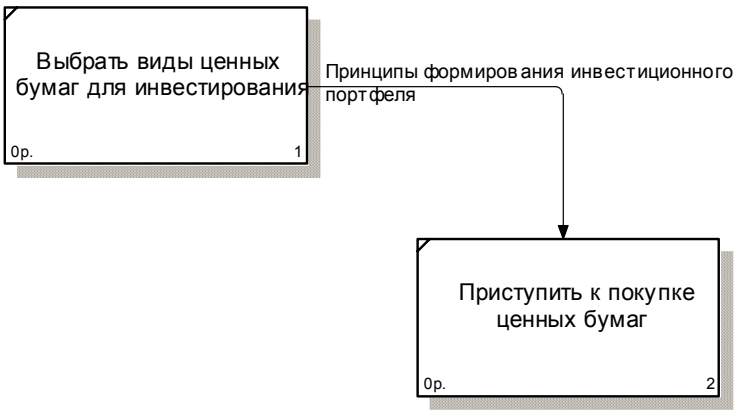


Рис. 2.21. Комбинированная стрелка «Выход – управление»

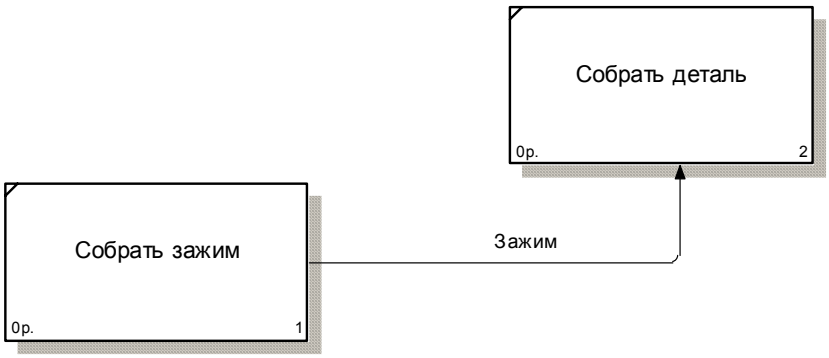


Рис. 2.22. Комбинированная стрелка «Выход – механизм исполнения»

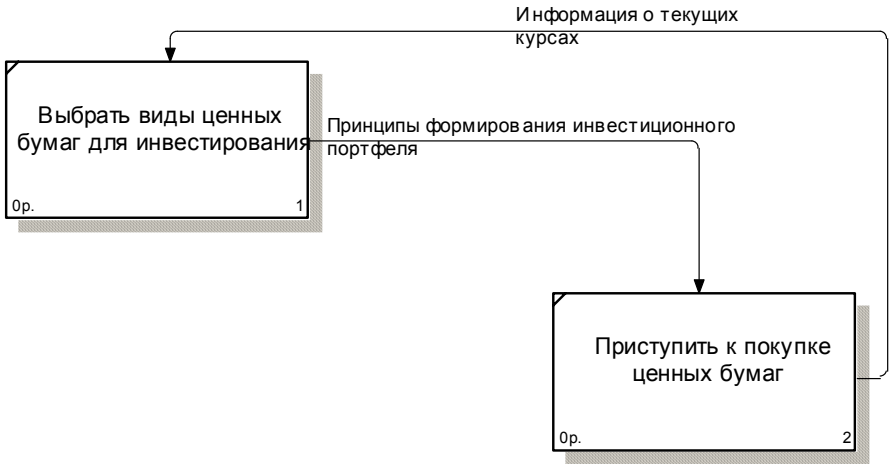


Рис. 2.23. Комбинированная стрелка «Выход – обратная связь на управление»

Стрелки «Выход – механизм исполнения» встречаются реже и отражают ситуацию, когда выход одного функционального блока применяется в качестве оборудования для работы другого блока. На рис. 2.22 зажим, устройство, используемое для закрепления детали во время ее сборки, должно быть собрано для того, чтобы выполнить сборку детали.

Обратные связи на вход и на управление применяются в случаях, когда зависимые блоки формируют обратные связи для управляющих ими блоков. На рис. 2.23 получаемая от брокеров информация о текущих биржевых курсах применяется для корректировки стратегии игры на бирже.

Стрелка «Выход – обратная связь на вход» обычно применяется для описания циклов повторной обработки чего-либо. Рис. 2.24 может служить примером применения стрелки такого типа. Кроме того, связи «Выход – обратная связь на вход» могут применяться в случае, если бракованная продукция может заново использоваться в качестве сырья, как это происходит, например, при производстве оконного стекла, когда разбитое в процессе производства стекло перемалывается и переплавляется заново вместе с обыкновенным сырьем.

Разбиение и соединение стрелок. Выход функционального блока может использоваться в нескольких других блоках. Фактически чуть ли не главная ценность IDEF0 заключается в том, что эта методология помогает выявить

взаимозависимости между блоками системы. Соответственно IDEF0 предусматривает как разбиение, так и соединение стрелок на диаграмме. Разбитые на несколько частей стрелки могут иметь наименования, отличающиеся от наименования исходной стрелки. Исходная и разбитые (или объединенные) стрелки в совокупности называются связанными. Такая техника обычно применяется для того, чтобы отразить использование в процессе только части сырья или информации, обозначаемых исходной стрелкой (рис. 2.25). Аналогичный подход применяется и к объединяемым стрелкам.

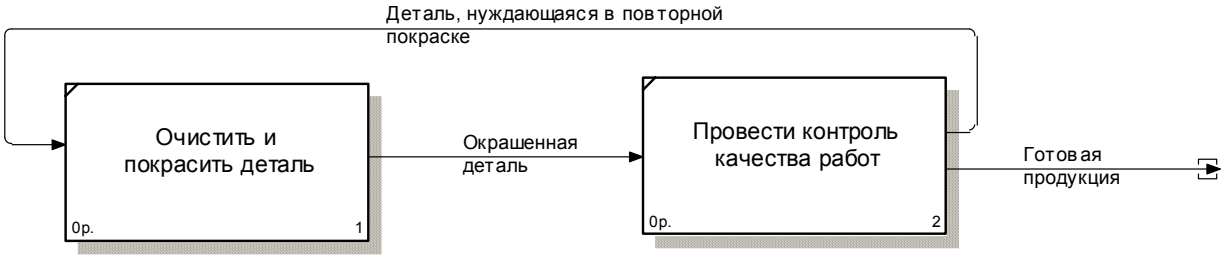


Рис. 2.24. Комбинированная стрелка «Выход – обратная связь на вход»

Рис. 2.25. Разбитая на две части и переименованная стрелка

Туннели. Понятие «связанные стрелки» используется для управления уровнем детализации диаграмм. Если одна из стрелок диаграммы отсутствует на родительской диаграмме (например, ввиду своей несущественности для родительского уровня) и не связана с другими стрелками той же диаграммы, точка входа этой стрелки на диаграмму или выхода с нее обозначается туннелем. На рис. 2.26 , например, стрелка «Корпоративная информационная система» – важный механизм исполнения для данной диаграммы, но, возможно, она более нигде не используется в модели. Туннель в данном случае используется как альтернатива загромождению родительских диаграмм помещением на них несущественных для их уровня стрелок.

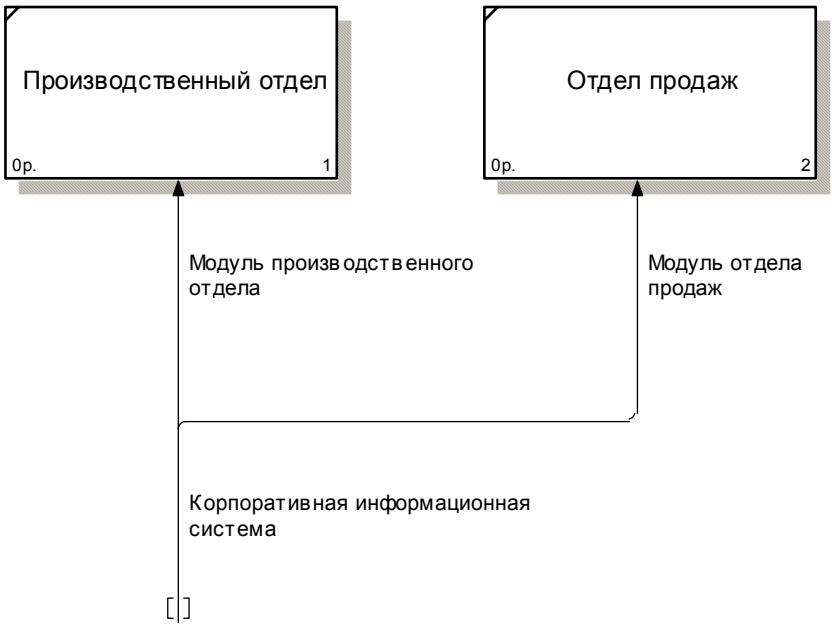


Рис. 2.26. Пример применения туннеля

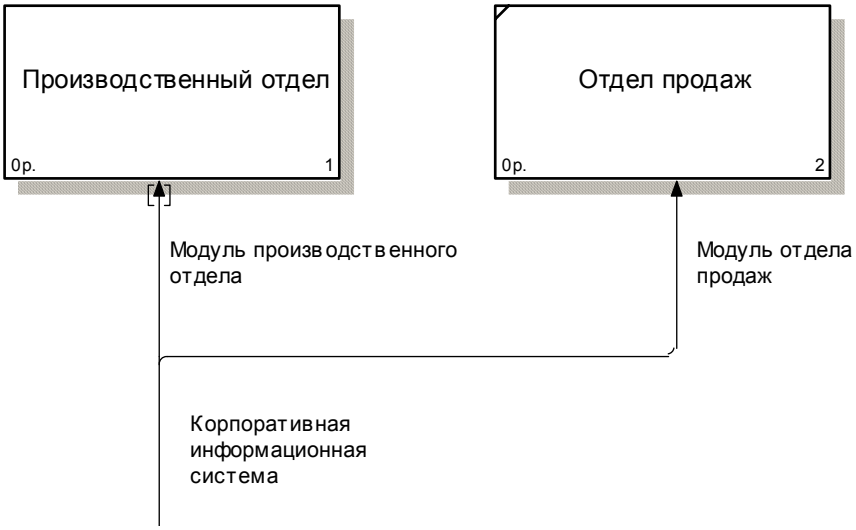


Рис. 2.27. Еще один пример применения туннеля

Кроме того, туннели применяются для отражения ситуации, когда стрелка, присутствующая на родительской диаграмме, отсутствует в диаграмме декомпозиции соответствующего блока. На рис. 2.27 туннель у стрелки «Модуль производственного отдела» обозначает, что на диаграмме декомпозиции производственного отдела отсутствует стрелка механизма управления с соответствующим наименованием.

2.4.2.2. Построение моделей IDEF0

Здесь мы рассмотрим методику построения моделей IDEF0 более подробно.

Диаграммы. На рис. 2.28 типовая диаграмма IDEF0 показана вместе с находящейся на ее полях служебной информацией. Служебная информация состоит из хорошо выделенных верхнего и нижнего колонтитулов (заголовка и «подвала»). Элементы заголовка используются для отслеживания процесса создания модели. Элементы «подвала» отображают наименование модели, к которой относится диаграмма, и показывают ее расположение относительно других диаграмм модели.

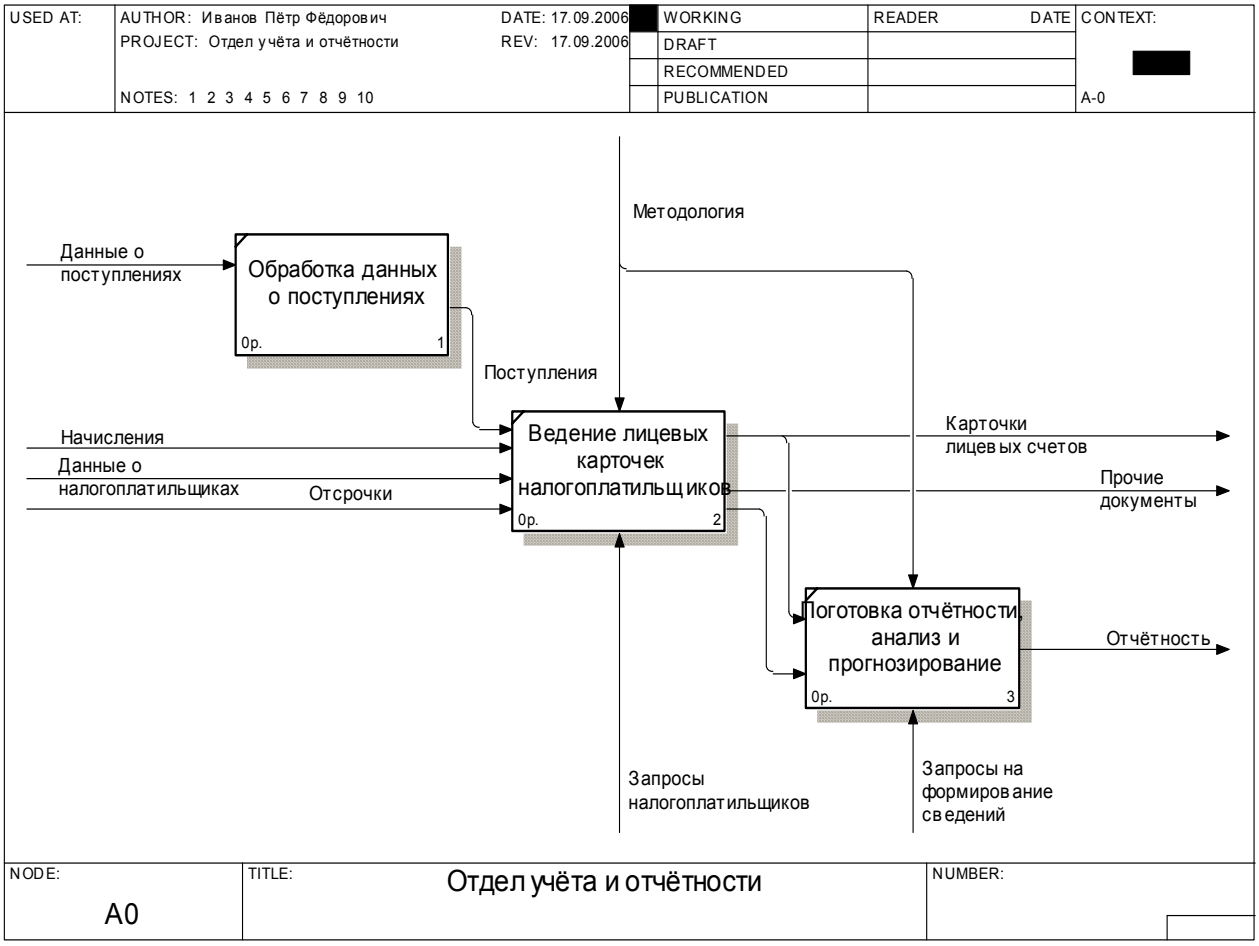


Рис. 2.28. IDEF0-диаграмма со служебной информацией на полях

Все элементы заголовка диаграммы перечислены в табл. 2.12.

Таблица 2.12

Элементы заголовка диаграммы IDEF0

Поле	Назначение
USED AT	Используется для отражения внешних ссылок на данную диаграмму (заполняется на печатном документе вручную)
Author, date, project	Содержит Ф. И. О. автора диаграммы, дату создания, дату последнего внесения изменений, наименование проекта, в рамках которого она создавалась
Notes 1 ... 10	При ручном редактировании диаграмм пользователи могут зачеркивать цифру каждый раз, когда они вносят очередное исправление
Status	Статус отражает состояние разработки или утверждения данной диаграммы. Это поле используется для реализации формального процесса публикации с шагами пересмотра и утверждения
Working	Новая диаграмма, глобальные изменения или новый автор для существующей диаграммы
Draft	Диаграмма достигла некоторого приемлемого для читателей уровня и готова для представления на утверждение
Recommended	Диаграмма одобрена и утверждена, какие-либо изменения не предвидятся
Publication	Диаграмма готова для окончательной печати и публикации
Reader	Ф. И. О. читателя
Date	Дата знакомства читателя с диаграммой
Context	Набросок расположения функциональных блоков на родительской диаграмме, на котором подсвечен декомпозируемый данной диаграммой блок. Для диаграммы самого верхнего уровня (контекстной диаграммы) в поле помещается контекст TOP

Все элементы нижней строки диаграммы перечислены в табл. 2.13.

Таблица 2.13

Элементы нижней строки диаграммы IDEF0

Поле	Назначение
Node	Номер диаграммы, совпадающий с номером родительского функционального блока
Title	Имя родительского функционального блока
Number	Уникальный идентификатор данной версии данной диаграммы. Таким образом, каждая новая версия данной диаграммы будет иметь новое значение в этом поле. Как правило, C-Number состоит из инициалов автора (которые предполагаются уникальными среди всех аналитиков проекта) и последовательного уникального идентификатора, например SDO005. При публикации эти номера могут быть заменены стандартными номерами страниц. Если диаграмма замещает другую диаграмму, номер заменяемой диаграммы может быть заключен в скобки – SDO005 (SDO004). Это обеспечивает хранение истории изменений всех диаграмм модели

Цикл «эксперт – аналитик». Подобно циклу «автор – редактор», применяющемуся в книгоиздательском деле, диаграммы IDEF0 пересматриваются и изменяются для обеспечения точности отражения предметной области и улучшения своего качества.

Для каждого рецензента автором, как правило, подготавливается свой набор диаграмм. Предложения по изменениям и исправлениям возвращаются рецензентами автору для внесения их в модель. При возникновении разногласий между автором и рецензентом спорная диаграмма обычно рассылается всем рецензентам для достижения группового консенсуса.

Формально механизм рецензирования и модификации диаграмм поддерживается полями Status и нумерацией диаграмм, контроль истории изменений – полем Field (см. табл. 2.12).

Построение моделей. Ни одна модель не должна строиться без ясного осознания объекта и целей моделирования. Выбранное определение цели моделирования должно отвечать на следующие вопросы:

Почему моделируется данный процесс?

Что выявит данная модель?

Как ознакомившиеся с этой моделью смогут ее применить?

Следующее предложение может служить примером формулирования цели моделирования. Выявить задачи каждого работника компании и понять в целом взаимосвязь между отдельно взятыми задачами для разработки руководства по обучению новых сотрудников.

Модели строятся для того, чтобы ответить на набор поставленных вопросов. Такие вопросы формулируются на ранних стадиях моделирования и

впоследствии служат основой для четкого и краткого определения цели моделирования. Примерами таких вопросов могут быть:

Каковы задачи менеджера?
Каковы задачи клерка?
Кто контролирует работу?
Какая технология нужна для выполнения каждого шага? и т. п.

Точка зрения. С методической точки зрения при моделировании полезно использовать мнение экспертов, имеющих разные взгляды на предметную область, однако каждая отдельно взятая модель должна разрабатываться исходя из единственной заранее определенной точки зрения. Часто другие точки зрения вкратце документируются в прикрепленных диаграммах FEO (см. ниже) исключительно для наглядности изложения.

Точку зрения нужно подбирать достаточно аккуратно, основой для выбора должна служить поставленная цель моделирования. Наименованием точки зрения может быть наименование должности, подразделения или роли (например, руководитель отдела или менеджер по продажам). Как и в случае с определением цели моделирования, четкое определение точки зрения необходимо для обеспечения внутренней целостности модели и предотвращения постоянного изменения ее структуры. Может оказаться необходимым построение моделей с разных точек зрения для детального отражения всех особенностей, выделенных в системе функциональных блоков.

Границы моделирования. Одним из положительных результатов построения функциональных моделей оказывается прояснение границ моделирования системы, а в целом и ее основных компонентов. Хотя и предполагается, что в процессе работы над моделью будет происходить некоторое изменение границ моделирования, их вербальное (словесное) описание должно поддерживаться с самого начала для обеспечения координации работы участвующих в проекте аналитиков. Как и при определении цели моделирования, отсутствие границ затрудняет оценку степени завершенности модели, поскольку границы моделирования имеют тенденцию к расширению с ростом размеров модели.

Границы моделирования имеют два компонента: ширину охвата и глубину детализации. Ширина охвата обозначает внешние границы моделируемой системы. Глубина детализации определяет степень подробности, с которой нужно проводить декомпозицию функциональных блоков.

Чтобы облегчить правильное определение границ моделирования при разработке моделей IDEF0, существенные усилия затрачиваются на разработку и рецензирование контекстной диаграммы IDEF0 (диаграммы «самого верхнего» уровня). Иногда даже прибегают к построению дополнительной диаграммы для отображения уровня, более высокого, чем контекстный, для данной модели, что позволяет обозначить систему, внутри которой располагается объект для моделирования. Существенные затраты на разработку контекстной диаграммы вполне оправданы, поскольку она является своего рода

«точкой отсчета» для остальных диаграмм модели и вносимые в нее изменения каскадом отражаются на всех лежащих ниже уровнях.

Когда границы моделирования понятны, становятся ясными и причины, по которым некоторые объекты системы не вошли в модель.

Выбор наименования контекстного блока. Рекомендуемой последовательностью действий при построении модели с нуля являются: формулирование цели моделирования, выбор точки зрения, определение границ моделирования. Наименование контекстного блока – функционального блока самого высокого уровня – обобщает определение границ моделирования.

Правила подбора имени для контекстного блока в целом не отличаются от общих правил наименования функциональных блоков, поэтому для них обычно подбирают обобщающие названия, типа «Управление отделом по работе с клиентами», «Обработка заказов» и т. п.

Определение стрелок на контекстной диаграмме. Стрелки диаграмм IDEF0 обычно проще проектировать в следующем порядке: выход, вход, механизм исполнения, управление. Каждый функциональный блок обозначает отдельную функцию, и эта функция часто имеет ясно и кратко описываемые результаты работы. Наличие неясностей при анализе выходов того или иного функционального блока – возможный сигнал необходимости проведения реинжиниринга рассматриваемого бизнес-процесса.

Определение выходов. После идентификации возможных выходов полезно провести анализ модели на предмет покрытия ею всех возможных сценариев поведения процесса. Это означает, что если существует вероятность возникновения той или иной ситуации в ходе процесса, модель отражает возможность возникновения такой ситуации. Многие начинающие аналитики забывают отразить негативные результаты работы функциональных блоков. Например, блок «Провести экзамен по вождению» определенно произведет поток водителей, только что получивших права, но вполне правомерно ожидать и потока лиц, не сдавших экзамен. Негативные результаты часто используются в качестве обратных связей, анализ на их наличие должен проводиться для каждого блока. Важным также является необходимость включения в модель спорных стрелок, принятие решения о наличии которых в модели вполне можно переложить на плечи рецензирующих модель экспертов.

Определение входов. Входы можно рассматривать как особым образом преобразуемые функциональными блоками для производства выхода сырье или информацию. В производственных отраслях определить, как входное сырье преобразуется в готовую продукцию, обычно довольно просто. Однако при моделировании информационных потоков входной поток данных может представляться не потребляемым и не обрабатываемым вообще. Случаи, когда входящие и исходящие стрелки называются в точности одинаково, крайне редки и в основном указывают на бесполезность данного блока для системы в целом или на некорректный выбор имени для исходящей стрелки. Решением может служить применение более подробного описания для входящих и исходящих потоков данных. Например, вход может иметь название

«Предварительный диагноз пациента», а выход – «Уточненный диагноз пациента».

Определение механизмов исполнения. После создания входов и выходов можно приступить к рассмотрению механизмов исполнения, или ресурсов, относящихся к функциональному блоку. В понятие механизма исполнения входят персонал, оборудование, информационные системы и т. п. Например, функциональный блок «Собрать деталь» может потребовать использования какого-либо оборудования, например гаечного ключа. При приеме экзаменов на водительские права механизмом исполнения является инспектор ГИБДД. Как правило, определить механизмы исполнения для функциональных блоков довольно просто.

Определение управления. Должно быть определено управление, контролирующее ход работы функционального блока. Все функциональные блоки в IDEF0 должны иметь хотя бы одно управление. В случаях, когда не ясно, относить ли стрелку к входу или к управлению, следует ее рисовать как управление. Важно помнить, что управление можно рассматривать как особую форму входа функционального блока.

Когда контекстная диаграмма представляется завершенной, попробуйте задать следующие вопросы:

Обобщает ли диаграмма моделируемый бизнес-процесс?

Согласуется ли диаграмма с границами моделирования, точкой зрения и целью моделирования?

Подходит ли выбранный уровень детализации стрелок для контекстного блока? (Обычно на контекстной диаграмме рекомендуется рисовать не более шести стрелок каждого типа.)

Нумерация блоков и диаграмм. Все функциональные блоки IDEF0 нумеруются. В номерах допускается использование префиксов произвольной длины, но в подавляющем большинстве моделей используется префикс А. Номер блока проставляется за префиксом. Контекстный блок всегда имеет номер А0.

Префикс повторяется для каждого блока модели. Номера используются для отражения уровня декомпозиции, на котором находится блок. Блок А0 декомпозируется в блоки А1, А2, А3 и т. д. А1 декомпозируется в А11, А12, А13 и т. д. А11 декомпозируется в А111, А112, А113 и т. д. Для каждого уровня декомпозиции в конец номера добавляется одна цифра.

Связь между диаграммой и ее родительским функциональным блоком. Функциональный блок декомпозируется, если необходимо детально описать его работу. При декомпозиции блока полезно рассмотреть его жизненный цикл, это поможет определить функциональные блоки получающейся детской диаграммы. Например, жизненный цикл 1 блока «Поджарить бифштекс» может выглядеть как следующая последовательность: подготовить продукты – отбить мясо – разогреть масло и т. д.

При моделировании IDEF0 важно иметь в виду, что граница детской диаграммы есть граница родительского функционального блока. Это означа-

ет, что вся работа выполняется блоками самого нижнего уровня. В отличие от иерархии, применяемой в структурном программировании, блоки верхнего уровня не являются субъектами управления для блоков нижнего уровня. Это означает, что в IDEF0 дети – это те же объекты, что и их родители, только показанные с большей детализацией. Действия генерального директора компании на диаграммах IDEF0 могут отражаться рядом с действиями простых рабочих.

На концах граничных стрелок (начинающихся или заканчивающихся за пределами диаграммы) детских диаграмм помещаются коды ICOM, чтобы показать, где находится соответствующая стрелка на родительской диаграмме (рис. 2.29). Они нужны для проверки целостности модели и могут быть полезны, когда порядок расположения стрелок на детской диаграмме отличается от порядка их расположения на родительской диаграмме. Код ICOM состоит из латинской буквы I, C, O или M и числа, показывающего расположение стрелки на родительской диаграмме в порядке сверху вниз или слева направо.

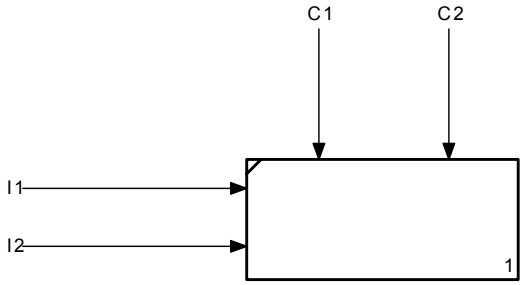


Рис. 2.29. ICOM-коды на граничных стрелках

Два подхода к началу моделирования («в ширину» и «в глубину»). Модели могут проектироваться с использованием подхода «в ширину», когда каждая диаграмма максимально детализируется перед своей декомпозицией, и с подходом «в глубину», когда сначала определяется иерархия блоков, а затем создаются соединяющие их стрелки. Естественно, возможно применение комбинации этих подходов, причем иерархия блоков может иногда немного измениться после того, как нарисованы стрелки. Это происходит из-за того, что создание стрелок может изменить понимание внутренней архитектуры моделируемого объекта.

3. АНАЛИЗ ТРЕБОВАНИЙ И ИХ ФОРМАЛИЗАЦИЯ

3.1. Методы определения требований

3.1.1. Интервьюирование

Одним из наиболее важных и понятных методов получения требований является *интервью с клиентом;* это метод, который можно использовать практически в любой ситуации.

Одна из основных задач интервьюирования – сделать все возможное, чтобы предубеждения и предпочтения интервьюируемых не повлияли на свободный обмен информацией. Это сложная проблема. Социология учит нас, что невозможно воспринимать окружающий мир, не фильтруя его в соответствии со своим происхождением и накопленным опытом.

3.1.1.1. Этапы проведения интервью

Процесс проведения интервью состоит из следующих этапов:

Разработка вопросов (образец вопросов интервью представлен в табл. 3.1).

Выбор опрашиваемых пользователей. Существует несколько групп пользователей: персонал начального уровня, организаторы проекта среднего уровня, менеджеры и другие заказчики особого рода – генеральные директоры и вице-президенты, научные сотрудники, обычные пользователи системы.

Планирование контактов.

Проведение интервью. Интервью можно провести по телефону, персонально или с помощью Интернета (видеоконференция, чат, электронная почта), однако лучшим способом осуществления интервью является непосредственное общение, лицом к лицу.

Завершение встречи и определение последующих действий.

Таблица 3.1

Обобщенное практически контекстно-свободное интервью

Часть I. Определение профиля заказчика или пользователя
Имя Компания Отрасль Должность (Вышеприведенная информация, как правило, может быть внесена заранее.) Каковы ваши основные обязанности?

Продолжение табл. 3.1

Что вы в основном производите? Для кого? Как измеряется успех вашей деятельности? Какие проблемы влияют на успешность вашей деятельности? Какие тенденции, если такие существуют, делают вашу работу проще или сложнее?
Часть II. Оценка проблемы Для каких проблем (прикладного типа) вы ощущаете нехватку хороших решений? Назовите их. (***Замечание***. *Не забывайте спрашивать: «А еще?».*) Для каждой проблемы выясняйте следующее. Почему существует эта проблема? Как она решается в настоящее время? Как заказчик (пользователь) хотел бы ее решать?
Часть III. Понимание пользовательской среды Кто такие пользователи? Какое у них образование? Каковы их навыки в компьютерной области? Имеют ли пользователи опыт работы с данным типом приложений? Какая платформа используется? Каковы ваши планы относительно будущих платформ? Используются ли дополнительные приложения, которые имеют отношение к данному приложению? Если да, то пусть о них немного расскажут. Каковы ожидания заказчика относительно практичности продукта? Сколько времени необходимо для обучения? В каком виде должна быть представлена справочная информация для пользователя (в интерактивном или в виде печатной копии)?
Часть IV. Резюме (перечисляются основные пункты, чтобы проверить, все ли правильно вы поняли) Итак, вы сказали мне (перечислите описанные заказчиком проблемы своими словами) Адекватно ли этот список представляет проблемы, которые имеются при существующем решении? Какие еще проблемы (если такие существуют) вы испытываете?
Часть V. Предположения, аналитика относительно проблемы заказчика (проверенные или непроверенные предположения) *(те проблемы, которые не были упомянуты)* Какие проблемы, если они есть, связаны с *(перечислите все потребности или дополнительные проблемы, которые, по-вашему, может испытывать заказчик или пользователь)*

Продолжение табл. 3.1

Для каждой из указанных проблем выясните следующее. Является ли она реальной? Каковы ее причины? Как она решается в настоящее время? Как бы заказчик (пользователь) хотел ее решать? Насколько важно для заказчика (пользователя) решение этой проблемы в сравнении с другими, упомянутыми им?
Часть VI. Оценка предлагаемого вами решения (если это уместно) (Охарактеризуйте основные возможности предлагаемого вами решения. А потом задайте пользователю следующие вопросы.) Что, если вы сможете? Как вы расцениваете важность этого?
Часть VII. Оценка возможности Кто в организации нуждается в данном приложении? Сколько пользователей указанных типов будет использовать его? Насколько значимо для вас успешное решение?
Часть VIII. Оценка необходимого уровня надежности и производительности, а также потребности в сопровождении Каковы ваши ожидания относительно надежности? Какой, по-вашему, должна быть производительность? Будете ли вы заниматься поддержкой продукта или этим будут заниматься другие? Испытываете ли вы потребности в поддержке? Что вы думаете о доступе для сопровождения и обслуживания? Каковы требования относительно безопасности? Какие требования относительно установки и конфигурации? Существуют ли специальные требования по лицензированию? Как будет распределено программное обеспечение? Есть ли требования на маркировку и упаковку?
Часть IX. Другие требования Существуют ли законодательные требования, требования информационной среды, инструкции или другие стандарты, которых необходимо придерживаться? Нет ли других требований, о которых нам следовало бы знать?
Часть X. Окончание Существуют ли другие вопросы, которые мне следовало бы вам задать? Если мне еще понадобится задать вам несколько вопросов, могу ли я вам позвонить? Будете ли вы принимать участие в обсуждении требований?

Окончание табл. 3.1

Часть XI. Заключение аналитика
После интервью, пока его данные еще свежи в вашей памяти, зафиксируйте три потребности или проблемы с наивысшими приоритетами, выявленные вами в беседе с данным заказчиком (пользователем)

3.1.2. «Мозговой штурм» и отбор идей

«Мозговой штурм» – это метод проведения собрания, при котором группа людей пытается найти решение специфической проблемы посредством накопления всех спонтанно возникающих идей.

Данный процесс имеет ряд очевидных преимуществ.

Поддерживает участие всех присутствующих.

Позволяет участникам развивать идеи друг друга.

Ведущий или секретарь ведет запись всего хода обсуждения.

Его можно применять при различных обстоятельствах.

Как правило, в результате получаем множество возможных решений для любой поставленной проблемы.

Метод способствует свободному мышлению, не ограниченному обычными рамками.

«Мозговой штурм» состоит из двух фаз: генерация идей и их отбор. Основная цель на этапе генерации состоит в том, чтобы описать как можно больше идей, не обязательно глубоких. На этапе отбора главной задачей является анализ всех возникших идей. Отбор идей включает в себя отсечение, организацию, упорядочение, развитие, группировку, уточнение и т. п.

3.1.2.1. Генерация идей

«Мозговой штурм» можно производить различными способами. Ниже описывается один из простых процессов. Все основные участники собираются в одной комнате, и им раздаются материалы для заметок. Это может быть просто стопка бумаги и черный толстый маркер. Листы бумаги должны быть не менее 7×12 см и не более 12×17 см. Каждому участнику нужно выдать не менее 25 листов на каждый сеанс «мозгового штурма».

Каждый участник сеанса «мозгового штурма» исполняет одну из трех ролей: лидер, секретарь или член команды. Лидер отвечает за направление процесса в правильное русло, за порядок и помогает секретарю делать записи. Секретарь записывает все идеи таким образом, чтобы каждый человек, находящийся в комнате, мог видеть эти записи. Участники генерируют идеи.

Ведущий объясняет правила проведения «мозгового штурма» (рис. 3.1) и определяется цель заседания.

Правила проведения «мозгового штурма»
1. Не допускается критика или дебаты.
2. Дайте свободу фантазии.
3. Генерируйте как можно больше идей.
4. Переделывайте и комбинируйте идеи.

Рис. 3.1. Правила проведения «мозгового штурма»

По мере создания идей секретарь просто собирает их и прикрепляет к стене комнаты заседаний. Большинство заседаний по генерации идей длится около часа (иногда 2–3 часа).

3.1.2.2. Отбор идей

После завершения фазы генерации идей начинается процесс отбора, который состоит из нескольких этапов.

1. Отсечение. Заключается в отсечении тех идей, которые не достойны внимания. Если обнаруживается две одинаковые идеи, эти идеи объединяются.

2. Группировка идей. Группы получают названия в зависимости от того, по какому принципу осуществляется группировка. Например, группы могут иметь следующие названия: новые функции, вопросы производительности, предложения по усовершенствованию существующих функций, интерфейс пользователя и вопросы простоты обращения и т. д. Для любой из этих групп можно возобновить генерацию идей, если окажется, что процесс группировки стимулировал возникновение новых идей или некоторая область важных функциональных возможностей осталась неохваченной.

3. Определение функций. Ведущий перечисляет все оставшиеся идеи и просит авторов дать их описание, состоящее из одного предложения (табл. 3.2).

4. Расстановка приоритетов.

Таблица 3.2

Пример определения функций

Область применения приложения	Штурмуемая функция	Описание функции
Автоматизация домашнего освещения	Автоматическое задание освещения	Домовладелец может предварительно задавать основанные на времени последовательности возникновения определенных осветительных событий в зависимости от времени дня

Окончание табл. 3.2

Область применения приложения	Штурмуемая функция	Описание функции
Система ввода заказов на покупку	Быстрота	Достаточно быстрое время ответа, чтобы не мешать проведению обычных операций
Система обнаружения неполадок	Автоматическое уведомление	Все зарегистрированные стороны будут уведомлены посредством электронной почты, когда что-нибудь изменится

3.1.3. Совместная разработка приложений (JAD – Joint Application Design)

Метод совместной разработки приложений – это зарегистрированный товарный знак, относящийся к компании IBM. Он представляет собой групповой подход к определению требований, при реализации которого особое внимание уделяется усовершенствованию группового процесса и правильному подбору людей, вовлеченных в работу над проектом.

Сеансы JAD аналогичны сеансам «мозгового штурма», однако не во всем. Сеансы «мозгового штурма» длятся около двух часов, а сеансы JAD – до трех дней. На сеансах «мозгового штурма» происходит быстрое генерирование идей, а на сеансах JAD – высокоуровневые специфические программные модели функций, данных и линий поведения.

Сеанс JAD имеет определенную структуру, на нем придерживаются определенной дисциплины, и он проходит под руководством арбитра. В его основе лежит обмен информацией с использованием документации, фиксированных требований и правил работы. С момента появления методики JAD на сеансах используются CASE-инструменты и другие программные средства, предназначенные для построения диаграмм потока данных (Data flow diagram, DFD), диаграмм взаимосвязей между сущностями (Entity relationship diagrams, ERD), диаграмм смены состояний и других объектно-ориентированных диаграмм.

3.1.3.1. Роли в сеансах JAD

Разработчики. В задачу разработчика входит оказание помощи организаторам в формулировании их потребностей, которые обычно являются решением существующих проблем. Таким образом, определение требований к ПО происходит совместно с организаторами проекта.

Участники. Для успешного проведения сеанса ключевым требованием является высокий уровень квалификации приглашенных. Правильный подбор людей позволит быстро принимать решения и разрабатывать правильные модели, даже если они не будут завершены.

Арбитр/консультант. Арбитр должен сводить к минимуму проявление непродуктивных человеческих эмоций, наподобие агрессивности или самозащиты. Арбитр не является хозяином ни процесса, ни программного продукта. Он присутствует только для того, чтобы помогать организаторам проекта, разрабатывать программный продукт.

Секретарь. Секретарь сеанса JAD документирует идеи и помогает следить за временем.

3.1.3.2. Сеансы JAD

Согласно Вуду (Wood), сеанс JAD – это своеобразная мастерская, работающая в максимально напряженном режиме, где решения принимаются совместно всеми участниками. При этом участники являются крупными специалистами в рассматриваемом вопросе.

Процесс исследования проекта разбивается на следующие этапы: поиск фактов и сбор информации, подготовка сеанса JAD, проведение самого сеанса JAD и проверка собранной информации.

3.1.3.3. Результаты проведения сеанса JAD

Результатами проведения сеанса могут быть:

диаграмма контекста данных;

диаграмма потока данных первого уровня;

глобальная модель данных – диаграмма взаимосвязей между сущностями;

перечень первичных объектов;

объектная модель высокого уровня;

обязанности кандидатов и сотрудников для каждого объекта;

перечень первичных процессов/сценарии выбора;

другие диаграммы потока данных, диаграммы состояния, деревья альтернатив, таблицы решений;

требования, предъявляемые к данным для каждого процесса;

перечень допущений;

документация по анализу или назначению открытых вопросов.

Результаты сеанса JAD используются в процессе определения требований для организации следующего этапа – создания спецификации SRS.

3.1.3.4. Недостатки метода JAD

Например, участники передают свои идеи арбитру и/или секретарю. Во избежание неверной интерпретации собранных данных, необходимо принять некоторые меры предосторожности. Использование во время сеанса автоматизированных инструментальных средств и проверка результатов всеми участниками уменьшит риск.

На сеансах JAD рассматриваются преимущественно информационные системы, в которых особое внимание уделяется элементам данных и проекту интерфейса. Есть мало информации об использовании метода JAD для определения требований, предъявляемых к системам реального времени.

На проведение трехдневного сеанса JAD с представителями всех групп организаторов проекта, каждая из которых состоит из квалифицированных специалистов, уходит немало средств. Три дня – это средняя продолжительность. Для анализа сложных вложенных систем реального времени и систем, от которых зависит человеческая жизнь, часто требуется больше времени. Если сеанс длится «до тех пор, пока не устанем», то усталость может наступить уже в тот момент, когда будут определены только сценарии выбора.

3.1.4. Раскадровка

Целью раскадровки является получение ранней реакции пользователей на предложенные концепции приложения. С ее помощью можно на самых ранних этапах жизненного цикла наблюдать реакцию пользователей, до того как концепции будут превращены в код, а во многих случаях даже до разработки подробной спецификации.

Раскадровка имеет следующие преимущества:

предельно недорога;

дружественна пользователю, неформальна и интерактивна;

обеспечивает ранний анализ пользовательских интерфейсов системы;

легко создаваема и модифицируема.

Раскадровки можно использовать для ускорения концептуальной разработки различных граней приложения. Их можно применять для понимания визуализации данных, определения и понимания бизнес-правил, которые будут реализованы в новом бизнес-приложении, для определения алгоритмов и других математических конструкций, которые будут выполняться внутри встроенных систем, или для демонстрации отчетов и других результатов на ранних этапах. Раскадровки можно (и нужно!) использовать практически для всех приложений, в которых раннее получение реакции пользователей является ключевым фактором успеха.

3.1.4.1. Типы раскадровок

Раскадровки делятся на три типа в зависимости от режима взаимодействия с пользователем: пассивные, активные и интерактивные.

Пассивные представляют собой историю, рассказываемую пользователю. Они могут состоять из схем, картинок, моментальных копий экрана, презентаций PowerPoint или образцов выходной информации системы. В пассивной раскадровке аналитик играет роль системы и просто проводит пользователя по раскадровке, объясняя следующее: «Когда вы делаете это, происходит вот это».

Активные раскадровки обеспечивают автоматизированное описание *п*оведения системы при типовом использовании или в операционном сценарии. Они создаются с помощью анимации или автоматизации, возможно, посредством автоматического последовательного показа слайдов, анимационных средств или даже фильма.

Интерактивные дают пользователю опыт обращения с системой почти такой же реальный, как на практике. Для своего выполнения они требуют участия пользователя. Интерактивные раскадровки могут быть имитационными, в виде макетов или могут даже представлять собой отбрасываемый впоследствии код. Сложная интерактивная раскадровка, основанная на отбрасываемом коде, может быть весьма похожа на отбрасываемый прототип.

Как видно из рис. 3.2, эти три типа раскадровки предлагают широкий спектр возможностей – от образцов выходной информации до «живых» демонстрационных версий. Различие между сложными раскадровками и ранними прототипами продукта весьма условно.

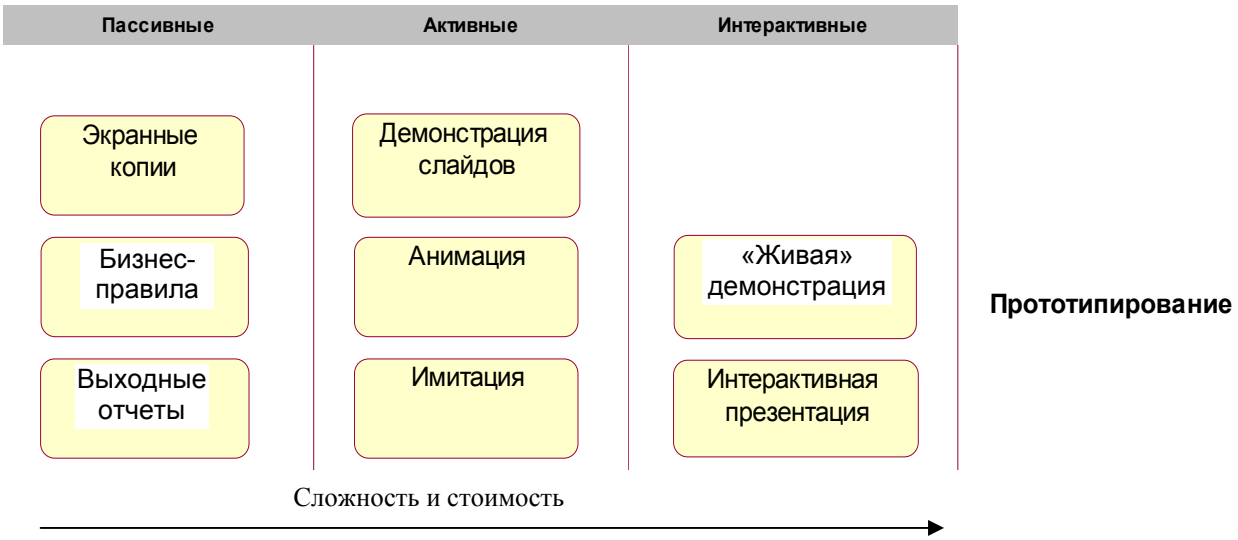


Рис. 3.2. Различные виды раскадровок

Выбор типа раскадровки зависит от сложности системы и того, насколько велик риск, что команда неправильно понимает ее назначение. Для

беспрецедентной новой системы, имеющей расплывчатое определение, может потребоваться несколько раскадровок, от пассивной до интерактивной, по мере того как команда совершенствует свое понимание системы.

3.1.5. Обыгрывание ролей

Метод обыгрывания ролей позволяет команде разработчиков прочувствовать мир пользователя, побывав в его роли. Концепция, лежащая в основе данного метода, достаточно проста: хотя, наблюдая и задавая вопросы, мы повышаем уровень своего понимания, наивно полагать, что посредством одного наблюдения разработчик/аналитик может получить истинно глубокое понимание решаемой проблемы или требований к системе, которая призвана данную проблему решить.

3.1.5.1. Суть метода обыгрывания ролей

Аналитик или любой член команды занимает место пользователя и выполняет его обычные действия. Рассмотрим в качестве примера проблему ввода заказов на покупку.

Разработчик/аналитик может «прочувствовать» проблемы и неточности, присущие существующей системе ввода заказов на покупку, просто заняв место оператора и попытавшись ввести несколько заказов. Полученный в течение часа опыт навсегда изменит понимание командой сути проблемы.

Существует две разновидности метода обыгрывания ролей: сценарный просмотр и CRC-карточки.

3.1.5.2. Сценарный просмотр

Сценарный просмотр – это исполнение роли на бумаге.

При сценарном просмотре каждый участник следует сценарию, который задает конкретную роль в «пьесе». Просмотр будет демонстрировать любые неточности в понимании ролей, недостаток информации, доступной актеру или подсистеме, или недостаток конкретного описания поведения, необходимого актерам для успешного выполнения их роли.

Одним из преимуществ сценарного просмотра является то, что сценарий можно модифицировать и проигрывать снова столько раз, сколько необходимо, пока актеры не сочтут его правильным. Сценарий можно также повторно использовать для обучения новых членов команды. Его можно модифицировать и проигрывать вновь, когда необходимо изменить поведение системы. В определенном смысле данный сценарий становится «живой» раскадровкой для проекта.

3.1.6. CRC-карточки (Class – Responsibility – Collaboration, класс – обязанность – взаимодействие)

Этот метод обыгрывания ролей часто применяется в объектно-ориентированном анализе. В данном случае каждому участнику выдается набор индексных карточек, описывающих класс (объект); обязанности (поведение); а также взаимодействия (с какими из моделируемых сущностей взаимодействует объект). Эти взаимодействия могут просто представлять сущности проблемной области (например, пользователи, нажатые кнопки, лампы и подъемники) или объекты, существующие в области решения (подсветка выключателя в холле, окно многодокументного интерфейса или кабина лифта).

Когда актер-инициатор инициирует определенное поведение, все участники следуют поведению, заданному на их карточках. Если процесс прерывается из-за недостатка информации или если одной сущности необходимо переговорить с другой, а взаимодействие не определено, то карточки модифицируются, и роли разыгрываются снова.

Ниже предлагается образец проигрывания одного из возможных вариантов использования.

1. **Управление включением.** Мой домовладелец только что нажал кнопку, управляющую набором ламп. Он все еще удерживает кнопку в нажатом состоянии. Я послал Центральному блоку управления сообщение, как только кнопка была нажата, и собираюсь посылать ему сообщения каждую секунду, пока кнопка нажата.

2. **Центральный блок управления.** Когда я получил первое сообщение, то изменил состояние выхода с «Выкл.» на «Вкл.». Когда я получил второе сообщение, стало очевидно, что домовладелец хочет изменить яркость освещения, поэтому при получении каждого сообщения я собираюсь изменять яркость на 10 %.

3. **Лампа.** Я аппаратно запрограммирован на изменяемый накал. Я изменяю яркость света при нашем разговоре.

3.1.7. Быстрое прототипирование

Быстрое прототипирование – это частичная реализация системы программного обеспечения, созданная с целью помочь разработчикам, пользователям и клиентам лучше понять требования к системе.

Чем детальнее прототип, тем легче понять требования заказчика. С другой стороны, прототипы сами по себе являются программами, поэтому, чем детальнее прототип, тем он дороже. Первый взгляд на проблему решения, строить прототип или нет, представлен на рис. 3.3. Таблица показывает, например, что относительно недорогой прототип с большой ценностью должен быть создан. Под большой ценностью имеется в виду, что построение прототипа поможет заказчику лучше понять, продукт какого типа будет вы-

пущен, помогает программистам лучше понять, какой продукт они должны выпустить.

	Предполагаемая ценность прототипа	
	Низкая	Высокая
Низкая стоимость прототипа	Возможно	Да
Высокая стоимость прототипа	Нет	Возможно

Рис. 3.3. Выгода от прототипа: грубая оценка

Для случаев, когда однозначно нельзя определить, разрабатывать прототип или нет, нужно оценить затраты на разработку прототипа и возможную прибыль от его реализации. На основе этой оценки определяются оптимальные расходы на прототип и принимается решение о его разработке и размере финансирования в случае положительного решения (рис. 3.4). По мере того как возрастают расходы на прототип, возрастает и его пригодность, но также возрастают и расходы из выделенного бюджета. В результате, вероятно, существует момент, в который затраты оптимальны (точка максимума на кривой), и некоторая точка, за которой деньги уже потрачены зря (где кривая пересекает горизонтальную ось).

Существует много способов разбиения прототипов на категории. Выделяются следующие прототипы: отбрасываемые, эволюционирующие и операционные; вертикальные и горизонтальные; пользовательские интерфейсы и алгоритмические и т. д. Каким должен быть прототип в каждом конкретном случае, зависит от того, какую проблему вы пытаетесь решить путем построения прототипа.

Как пример представьте себе приложение для электронной коммерции, в котором компания-производитель одежды желает продавать товары через Интернет, хранить информацию о клиентах и предоставлять клиентам возможность получать свое фото в одежде из каталога. Финансовая оценка для разных уровней прототипирования для программы, продающей одежду, приведена в табл. 3.3. Для каждой из четырех характеристик, рассмотренных в прототипе, сделано несколько оценок: стоимость работы; процент работы, который будет повторно использоваться в самой программе (т. е. не будет отброшен); и полная прибыль от прототипа. Под полной прибылью здесь понимается оценка того, что будет получено, если характеристика будет включена в прототип, но код не будет использован в программе. Например, мы подсчитали, что если прототип «примерка одежды» будет построен, это сэкономит минимум 20 000 при разработке. Оценка базируется на нижеследующих факторах.

Предотвращение пустой траты времени на предложенные требования, которые, как видно из прототипа, не нужны (т. е. минимум три ненужных требования из 100; на этап требований выделено 300 000, сэкономлено 9000).

Разработка программного проекта «примерка одежды», что уменьшает риски разработки (т. е. оценка того, что это сэкономит минимум одну человеко-неделю времени проектирования = 2000).

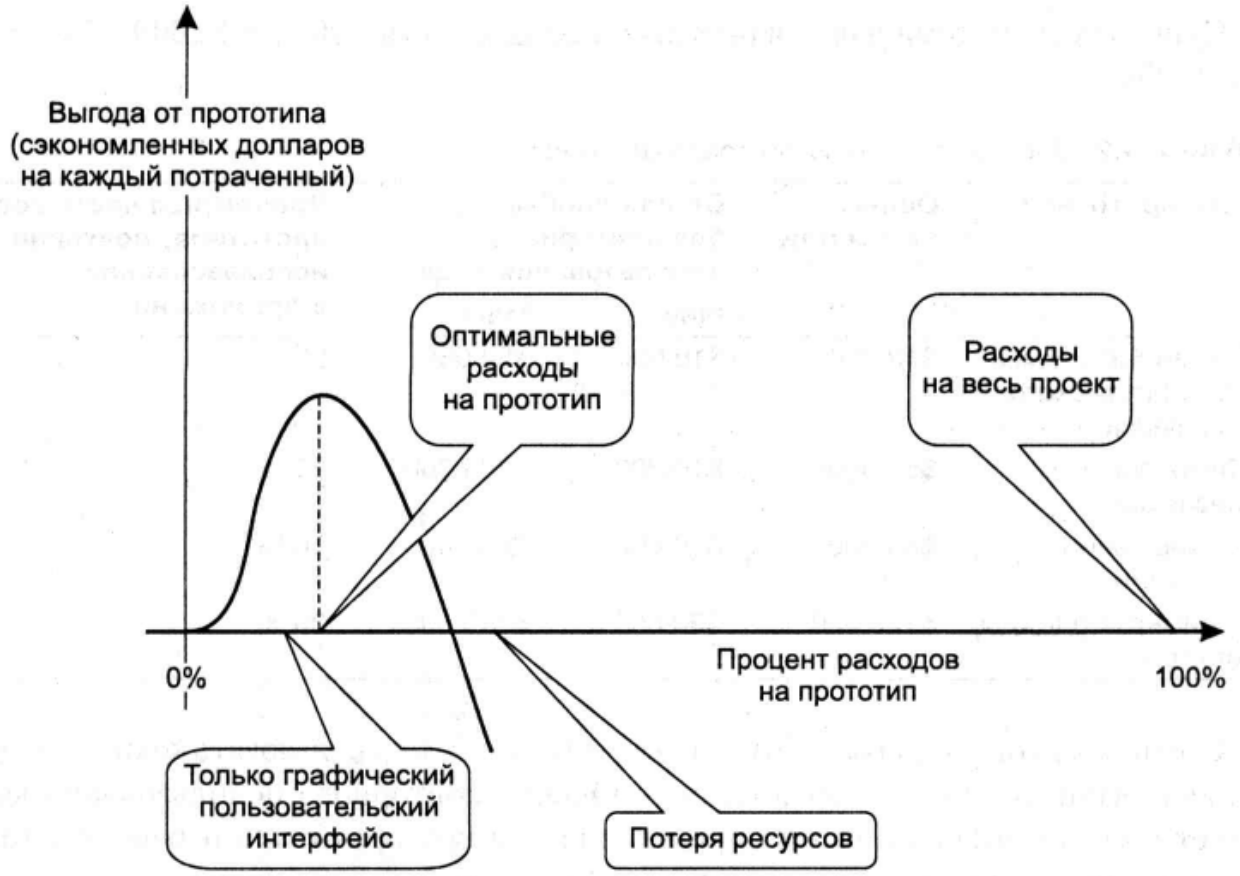


Рис. 3.4. Выгода от прототипа

Переработка, которая может возникнуть из-за изменения требований клиентом после того, как он увидит окончательный продукт (т. е. переработка минимум трех требований по 3000 каждое = 9000).

Существует минимальная экономия, эквивалентная 9000 + 2000 + 9000 = = 20 000.

Оценка повторного использования кода может быть выполнена путем определения классов прототипа и решения того, какие из них будут использованы в самой программе.

Такая оценка состоит из исследования стоимости небольших частей, что все же является сложной задачей. Определение минимума и максимума такой оценки может несколько упростить этот трудный процесс.

Как только оценки сделаны, выполняется несложная часть вычисления лучшего и худшего сценария (табл. 3.3). Минимальное значение выгоды получается из самой пессимистичной комбинации – высоких затрат, низкой общей прибыли и низкого процента повторного использования. Максимальная выгода рассчитывается аналогично.

Усреднение – это один из способов работы с разницей между худшими и лучшими случаями. В результате получаем положительную выгоду для всех предложенных частей прототипа за исключением части «примерка одежды», которая оценена в –4000: общий убыток 4000. Это приведет в дальнейшем к относительно низкой прибыли, высокой стоимости разработки и низкому вторичному использованию.

Таблица 3.3

Оценка программы по продаже одежды

Часть прототипа	Оценка стоимости	Общая прибыль без повторного использования кода		Процентная часть кода прототипа, повторно использованная в приложении	Чистая выгода		
		min	max		min	max	В среднем
	B	D	E	C	D – (1 – – C)B	E – (1 – – C)B	
1. Экранные снимки пользовательского интерфейса	10 000	10 000	80 000	50 %	5 000	75 000	40 000
2. Безопасность транзакций	50 000	10 000	300 000	80 %	0	290 000	145 000
3. Завершение транзакций	80 000	10 000	400 000	50 %	–30 000	200 000	85 000
4. Примерка одежды клиентом	120 000	20 000	140 000	30 %	–64 000	56 000	–4 000

3.2. Формализация требований

Иногда присущая естественному языку неоднозначность просто неприемлема, особенно когда требования касаются жизненно важных вопросов или когда неправильное поведение системы может привести к чрезвычайным экономическим или юридическим последствиям. Если определение требования сложно сформулировать на естественном языке и невозможно предотвратить неправильное понимание спецификации, следует попытаться написать эту часть требований с помощью теоретически обоснованных формальных методов.

3.2.1. Метод вариантов использования и его применение

Метод вариантов использования (прецедентов) является частью методологии объектно-ориентированного проектирования. Это метод анализа и проектирования сложных систем, представляющий собой способ описания поведения системы с точки зрения того, как различные пользователи взаимодействуют с ней для достижения своих целей. Такой ориентированный на пользователя подход предоставляет возможность исследовать различные варианты поведения системы при раннем привлечении пользователя.

Варианты использования можно успешно применять на протяжении всего жизненного цикла программного обеспечения: при анализе, проектировании и в процессе тестирования. В этом случае, процесс разработки программного обеспечения называют «основанный на вариантах использования».

Вариант использования – это функциональный связный блок, выраженный в виде транзакции между актантом и системой. Вариант использования описывает последовательность действий, выполняемых системой с целью предоставить полезный результат конкретному актанту.

3.2.1.1. Построение модели вариантов использования

Модель вариантов использования системы состоит из всех актантов системы и различных вариантов использования, посредством которых актанты взаимодействуют с системой, тем самым, описывая многообразие ее функционального поведения. Она также показывает связи между вариантами использования, что углубляет наше понимание системы.

Первый шаг моделирования вариантов использования состоит в создании системной диаграммы, описывающей границы системы и определяющей ее актанты. Это позволяет параллельно осуществить этапы 3 и 4, в которых требуется выявить заинтересованных лиц системы и определить ее границы.

Второй шаг описывает системное поведение, т. е. то, как пользователи взаимодействуют с системой, осуществляя некоторые последовательности действий, для достижения определенных целей (рис. 3.5).

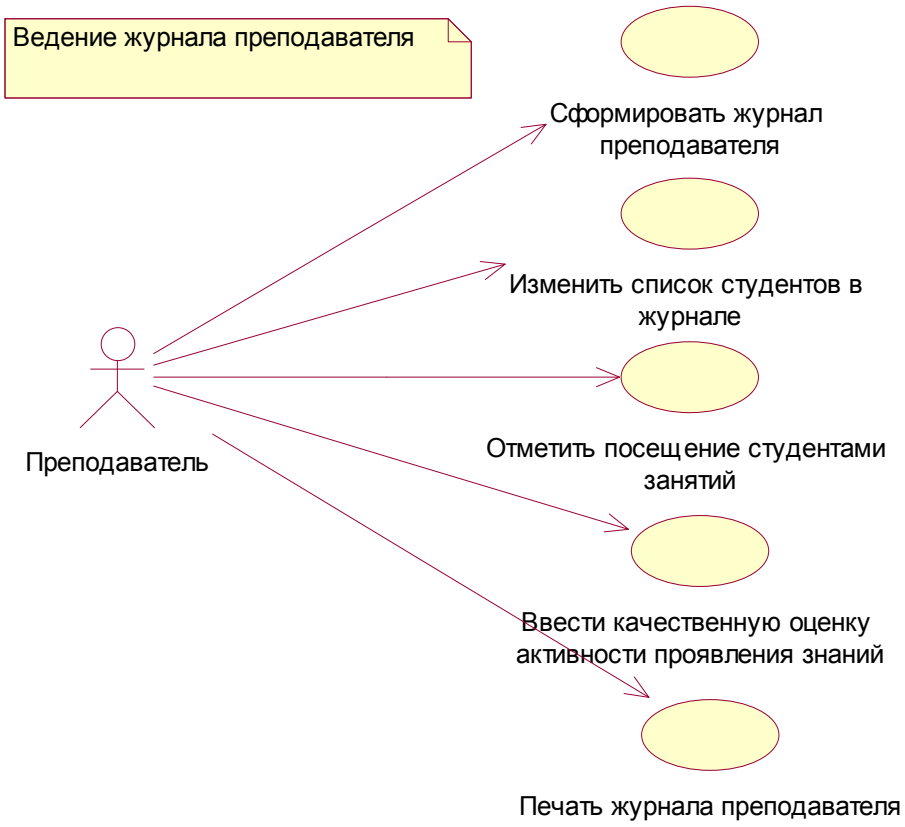


Рис. 3.5. Диаграмма вариантов использования

Следующий шаг состоит в уточнении деталей функционального поведения каждого варианта использования. Спецификации вариантов использования состоят из текстовых и графических описаний каждого варианта использования, написанных с позиции пользователя.

3.2.1.2. Спецификация вариантов использования

Определение потока событий. Сердцевиной варианта использования является поток событий. Это текстовое описание операций актанта и различных ответов системы. Поток событий описывает, что, как предполагается, будет делать система в зависимости от поведения актанта. Не обязательно описывать поток в текстовой форме. Для этого можно использовать диаграммы взаимодействия UML, а также другие формальные методы. Главное – обеспечить понимание, так как единственного подхода на все случаи жизни не существует. Однако, как правило, вполне подходит описание на естественном языке.

Поток событий описывает достижение цели варианта использования и предназначен для рассмотрения следующими заинтересованными лицами:

клиентами, которые одобряют результат;

пользователями, которые будут работать с системой;

разработчиками вариантов использования, которые заинтересованы в точном описании желаемого поведения системы;

рецензентами, которые составляют непредвзятое мнение о системе;

проектировщиками, которые анализируют варианты использования в поисках классов, объектов и т. п.;

тестерами, которым нужно создать тестовые примеры;

менеджером проекта, которому необходимо понимать весь проект в целом;

составителем технической документации, которому нужно документировать функции системы так, чтобы было понятно пользователю;

людьми из отдела маркетинга и продаж, которым необходимо понимать функции продукта и объяснять его достоинства остальным.

Альтернативный поток событий. Вариант использования может иметь различные потоки в зависимости от возникающих условий. Иногда эти потоки связаны с выявленными в процессе обработки ошибочными условиями, в других случаях они могут описывать дополнительные способы обработки конкретных условий.

В нашем примере альтернативный поток событий возникает, когда Жилец удерживает кнопку пульта в нажатом состоянии дольше одной секунды. Нам нужно добавить в вариант использования альтернативный поток.

Выявление пред- и постусловий. Использовать пред- и постусловия нужно только тогда, когда необходимо прояснить поведение, выраженное вариантом использования.

Важно проводить различие между событиями, которые запускают потоки варианта использования, и предусловиями, которые должны быть выполнены до того, как можно будет инициировать вариант использования. Например, предусловием для варианта использования «Управление освещением» является то, что домовладелец (Жилец) должен обеспечить наличие определенного набора осветительных приборов, способных изменять яркость. Еще одним предусловием является то, что выбранная кнопка пульта должна быть запрограммирована для управления этим набором. (Предполагается, что другие варианты использования описывают, как осуществляются эти предусловия.) Итак, нам нужно сформулировать предусловия.

Постусловия позволяют точно указывать состояние, которое должно быть истинным по окончании варианта использования, даже если использовались альтернативные пути.

Чтобы яркость была такой, как нужно, когда Жилец включает свет в следующий раз, система должна «помнить» уровень яркости, который был установлен для выбранной кнопки пульта после действий по изменению яркости. Следовательно, это постусловие, которое мы должны записать для данного варианта использования.

3.2.1.3. Преимущества

Применение вариантов использования имеет ряд преимуществ по сравнению с традиционным подходом, когда определяются отдельные декларативные программные требования:

варианты использования относительно легко писать;

варианты использования пишутся на языке пользователя;

варианты использования предлагают связные сценарии поведения, которые понятны как пользователю, так и разработчику.

Благодаря описанию «сценариев поведения», содержащимся в языке UML специализированным элементам и нотациям моделирования, варианты использования обеспечивают дополнительные возможности связывать деятельность по разработке требований с проектированием и реализацией.

Графическое представление вариантов использования в UML и их поддержка различными инструментальными средствами моделирования обеспечивают визуализацию связей между вариантами использования, что может содействовать пониманию сложной программной системы.

Сценарий, описанный с помощью варианта использования, может практически без изменений применяется в качестве сценария тестирования во время проверки правильности.

3.2.2. Псевдокод

Псевдокод – это квазиязык программирования; попытка соединить неформальный естественный язык со строгими синтаксическими и управляющими структурами языка программирования. В чистом виде псевдокод состоит из комбинации следующих элементов.

Императивных предложений с одним глаголом и одним объектом.

Ограниченного множества (как правило, не более 40–50) «ориентированных на действия» глаголов, из которых должны конструироваться предложения.

Решений, представленных формальной структурой IF-ELSE-ENDIF.

Итеративных действий, представленных структурами DO-WHILE и FOR-NEXT.

На рис. 3.6 представлен пример спецификации с помощью псевдокода алгоритма вычисления отложенного дохода от услуг в течение данного месяца в бизнес-приложении.

```
Set Sum(X)=0
for каждого клиента x
   if клиент оплатил услуги вперед
        and ((Текущий месяц)>=(2 мес. после даты приобретения)) and ((Текущий
        месяц)<=(14 мес. после даты приобретения))
   then Sum(X) = Sum(x) + (сумма, заплаченная клиентом)/12
```

Рис. 3.6. Пример спецификации с использованием псевдокода

Фрагменты текста на псевдокоде расположены уступами; такой формат используется для того, чтобы выделить логические блоки. Сочетание синтаксических ограничений, формата и разбивки существенно уменьшает неоднозначность требования, которое в противном случае было бы очень сложным и расплывчатым. В то же время такое представление требования вполне понятно для человека, не являющегося программистом. Не нужно быть выдающимся ученым, чтобы понимать псевдокод, и нет необходимости знать C++ или Java.

3.2.3. Конечные автоматы

Представление в виде конечного автомата описывает возможные жизненные циклы объекта и состоит из состояний, соединенных переходами.

Каждое состояние – это такой период жизненного цикла объекта, когда он удовлетворяет определенным условиям. Некоторое событие может привести к переходу, в результате которого объект окажется в новом состоянии. При переходе может выполняться действие, предписанное данному переходу.

Представление в виде конечного автомата изображается на диаграммах состояний и переходов.

На рис. 3.7 приведен пример диаграммы состояний и переходов для объекта «Заказ».

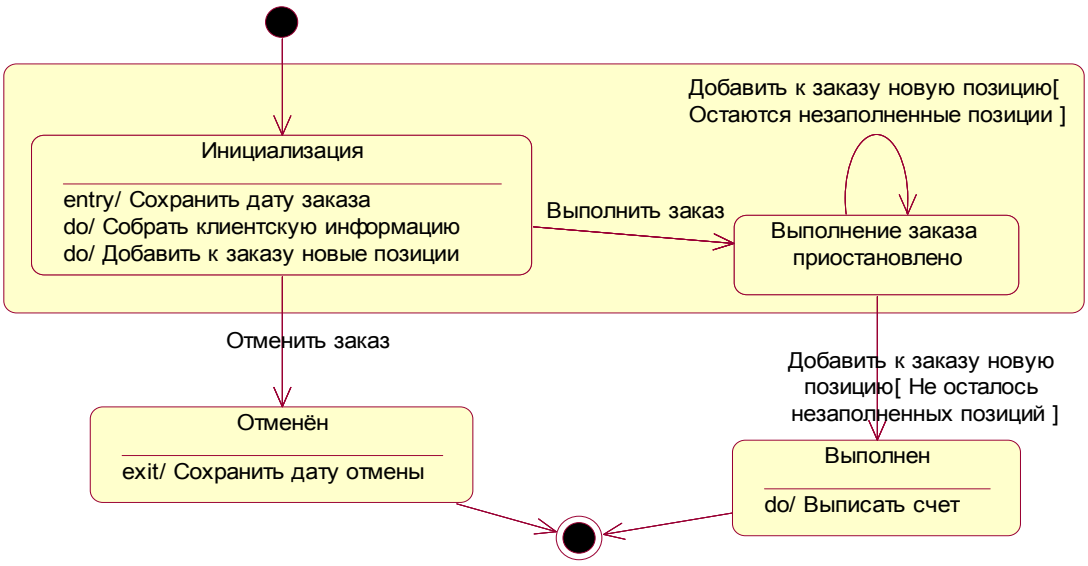


Рис. 3.7. Диаграмма состояний и переходов объекта «Заказ»

3.2.4. Графические деревья решений

Дерево решений применяется для отображения информации в виде графа. На рис. 3.8 показано дерево решений, используемое для описания последовательности действий.

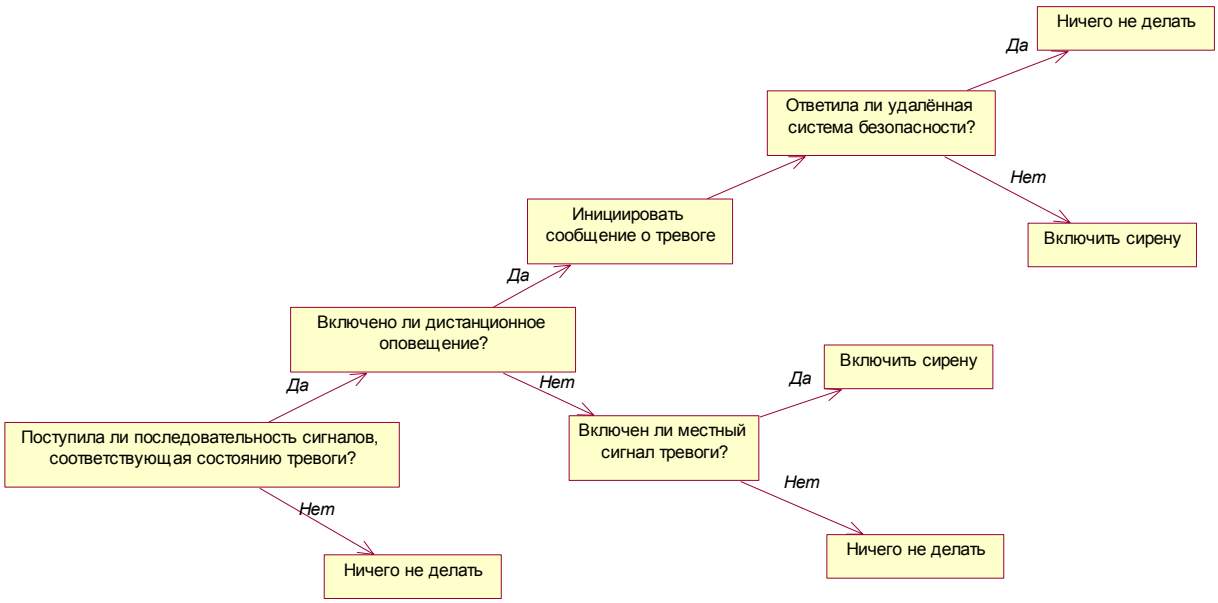


Рис. 3.8. Графическое дерево решений

3.2.5. Диаграммы деятельности

Деятельность представляет собой поток работ или выполнение операции. Представление деятельности отображает как последовательные, так и параллельные виды деятельности. Изображаются такие модели на диаграммах деятельности.

На рис. 3.9 изображена диаграмма, которая описывает деятельности покупателя в Интернет-магазине. Здесь представлены две точки ветвления – для выбора способа поиска товара и для принятия решения о покупке. Присутствуют три линейки синхронизации: верхняя – отражает разделение на два параллельных процесса, средняя отражает и разделение, и слияние процессов, а нижняя – только слияние процессов. Дополнительно на этой диаграмме показаны две дорожки – дорожка покупателя и дорожка магазина, которые разделены вертикальной линией. Каждая дорожка имеет имя и фиксирует область деятельности конкретного лица, обозначая зону его ответственности.

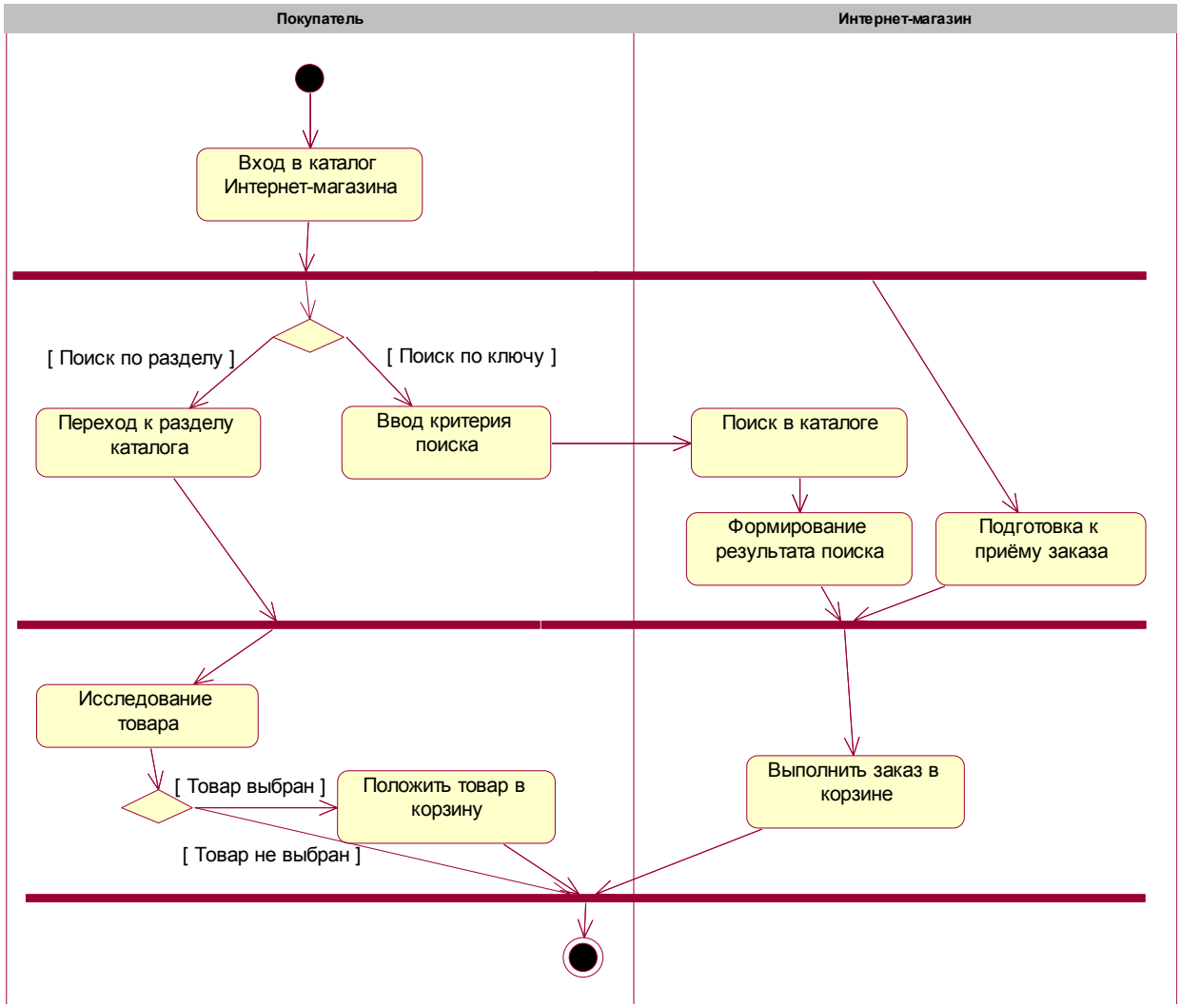


Рис. 3.9. Диаграмма деятельности покупателя в Интернет-магазине

Диаграмма деятельности отражает реальные потоки работ в человеческой организации. Такое бизнес-моделирование и есть основное ее назначение. С таким же успехом ее можно использовать и при моделировании работ программного приложения.

3.3. Техническое задание (ГОСТ 34.602–89)

Настоящий стандарт распространяется на автоматизированные системы (АС) для автоматизации различных видов деятельности (управление, проектирование, исследование и т. п.), включая их сочетания, и устанавливает состав, содержание, правила оформления документа «Техническое задание на создание (развитие или модернизацию) системы».

3.3.1. Общие сведения

В этот раздел включают: полное наименование системы и ее условное обозначение; шифр темы или шифр (номер) договора; наименование предприятий (объединений) разработчика и заказчика (пользователя) системы и их реквизиты; перечень документов, на основании которых создается система, кем и когда утверждены эти документы; плановые сроки начала и окончания работы по созданию системы; сведения об источниках и порядке финансирования работ; порядок оформления и предъявления заказчику результатов работ по созданию системы (ее частей), по изготовлению и наладке отдельных средств (технических, программных, информационных) и программно-технических (программно-методических) комплексов системы.

3.3.2. Назначение и цели создания (развития) системы

3.3.2.1. Назначение системы

Здесь указывают вид автоматизируемой деятельности (управление, проектирование и т. п.) и перечень объектов автоматизации, на которых предполагается ее использовать. Для АСУ дополнительно указывают перечень автоматизируемых органов (пунктов) управления и управляемых объектов.

3.3.2.2. Цели создания системы

Приводят наименования и требуемые значения технических, технологических, производственно-экономических или других показателей объекта автоматизации, которые должны быть достигнуты в результате создания АС, и указывают критерии оценки достижения целей создания системы.

3.3.3. Характеристики объекта автоматизации

В данном разделе приводят: краткие сведения об объекте автоматизации или ссылки на документы, содержащие такую информацию; сведения об условиях эксплуатации объекта автоматизация и характеристиках окружающей среды.

Примечание. Для САПР дополнительно приводят основные параметры и характеристики объектов проектирования.

3.3.4. Требования к системе

Состав требований к системе, включаемых в данный раздел ТЗ на АС, устанавливают в зависимости от вида, назначения, специфических особенностей и условий функционирования конкретной системы. В каждом подразде-

ле приводят ссылки на действующие НТД, определяющие требования к системам соответствующего вида.

3.3.4.1. Требования к системе в целом

Требования к структуре и функционированию системы. Здесь приводят: перечень подсистем, их назначение и основные характеристики, требования к числу уровней иерархии и степени централизации системы; требования к способам и средствам связи для информационного обмена между компонентами системы; требования к характеристикам взаимосвязей создаваемой системы со смежными системами, требования к ее совместимости, в том числе указания о способах обмена информацией (автоматически, пересылкой документов, по телефону и т. п.); требования к режимам функционирования системы; требования по диагностированию системы; перспективы развития, модернизации системы.

Требования к численности и квалификации персонала на АС. Приводят: требования к численности персонала (пользователей) АС; требования к квалификации персонала, порядку его подготовки и контроля знаний и навыков; требуемый режим работы персонала АС.

Требования к показателям назначения АС. Приводят значения параметров, характеризующих степень соответствия системы ее назначению.

Для АСУ указывают: степень приспособляемости системы к изменению процессов, к отклонениям параметров объекта управления; номер методов управления; допустимые пределы модернизации и развития системы; вероятностно-временные характеристики, при которых сохраняется целевое назначение системы.

Требования к надежности. Включают: состав и количественные значения показателей надежности для системы в целом или ее подсистем; перечень аварийных ситуаций, по которым должны быть регламентированы требования к надежности, и значения соответствующих показателей; требования к надежности технических средств и программного обеспечения; требования к методам оценки и контроля показателей надежности на разных стадиях создания системы в соответствии с действующими нормативно-техническими документами.

Требования по безопасности. Включают требования по обеспечению безопасности при монтаже, наладке, эксплуатации, обслуживании и ремонте технических средств системы (защита от воздействий электрического тока,

электромагнитных полей, акустических шумов и т. п.), по допустимым уровням освещенности, вибрационных и шумовых нагрузок.

Требования по эргономике и технической эстетике. Включают показатели АС, задающие необходимое качество взаимодействия человека с машиной и комфортность условий работы персонала.

Требования к эксплуатации, техническому обслуживанию, ремонту и хранению. Включают: условия и регламент (режим) эксплуатации, которые должны обеспечивать использование технических средств (ТС) системы с заданными техническими показателями, в том числе виды и периодичность обслуживания ТС системы или допустимость работы без обслуживания; предварительные требования к допустимым площадям для размещения персонала и ТС системы, к параметрам сетей энергоснабжения и т. п.; требования по количеству, квалификации обслуживающего персонала и режимам его работы; требования к составу, размещению и условиям хранения комплекта запасных изделий и приборов; требования к регламенту обслуживания.

Требования к защите информации от несанкционированного доступа. Включают требования, установленные в НТД, действующей в отрасли (ведомстве) заказчика.

Требования по сохранности информации. Приводят перечень событий: аварий, отказов технических средств (в том числе потеря питания) и т. п., при которых должна быть обеспечена сохранность информации в системе.

Требования к средствам защиты от внешних воздействий. Приводят: требования к радиоэлектронной защите средств АС; требования по стойкости, устойчивости и прочности к внешним воздействиям (среде применения).

Требования к патентной чистоте. Указывают перечень стран, в отношении которых должна быть обеспечена патентная чистота системы и ее частей.

Требования к стандартизации и унификации. Включают: показатели, устанавливающие требуемую степень использования стандартных, унифицированных методов реализации функций (задач) системы, поставляемых программных средств, типовых математических методов и моделей, типовых проектных решений, унифицированных форм управленческих документов, установленных ГОСТ 6.10.1, общесоюзных классификаторов технико-экономической информации и классификаторов других категорий в соответствии с областью их применения, требования к использованию типовых автоматизированных рабочих мест, компонентов и комплексов.

Дополнительные требования. Включают: требования к оснащению системы устройствами для обучения персонала (тренажерами, другими устройствами аналогичного назначения) и документацией на них; требования к сервисной аппаратуре, стендам для проверки элементов системы; требования к системе, связанные с особыми условиями эксплуатации; специальные требования по усмотрению разработчика или заказчика системы.

3.3.4.2. Требования к функциям (задачам)

Здесь приводят: по каждой подсистеме перечень функций, задач или их комплексов (в том числе обеспечивающих взаимодействие частей системы), подлежащих автоматизации; при создании системы в две или более очереди – перечень функциональных подсистем, отдельных функций или задач, вводимых в действие в 1-й и последующих очередях; временной регламент реализации каждой функции, задачи (или комплекса задач); требования к качеству реализации каждой функции (задачи или комплекса задач), к форме представления выходной информации, характеристики необходимой точности и времени выполнения, требования одновременности выполнения группы функций, достоверности выдачи результатов; перечень и критерии отказов для каждой функции, по которой задаются требования по надежности.

3.3.4.3. Требования к видам обеспечения

В зависимости от вида системы приводят требования к математическому, информационному, лингвистическому, программному, техническому, метрологическому, организационному, методическому и другие видам обеспечения системы.

Требования к математическому обеспечению системы. Приводят требования к составу, области применения (ограничения) и способам использования в системе математических методов и моделей, типовых алгоритмов и алгоритмов, подлежащих разработке.

Требования к информационному обеспечению. Приводят требования: к составу, структуре и способам организации данных в системе; к информационному обмену между компонентами системы; к информационной совместимости со смежными системами; по использованию общесоюзных и зарегистрированных республиканских, отраслевых классификаторов, унифицированных документов и классификаторов, действующих на данном предприятии; по применению систем управления базами данных; к структуре процесса сбора, обработки, передачи данных в системе и представлению данных; к защите данных от разрушений при авариях и сбоях в электропита-

нии системы; к контролю, хранению, обновлению и восстановлению данных; к процедуре придания юридической силы документам, продуцируемым техническими средствами АС (в соответствии с ГОСТ 6.10.4).

Требования к лингвистическому обеспечению. Приводят требования к применению в системе языков программирования высокого уровня, языков взаимодействия пользователей и технических средств системы, а также требования к кодированию и декодированию данных, к языкам ввода-вывода данных, языкам манипулирования данными, средствам описания предметной области (объекта автоматизации), к способам организации диалога.

Требования к программному обеспечению. Приводят перечень покупных программных средств. А также требования: к независимости программных средств от используемых комплексов технических средств и операционной среды; к качеству программных средств, а также к способам его обеспечения и контроля; по необходимости согласования вновь разрабатываемых программных средств с фондом алгоритмов и программ.

Требования к техническому обеспечению. Приводят требования: к видам технических средств, в том числе к видам комплексов технических средств, программно-технических комплексов и других комплектующих изделий, допустимых к использованию в системе; к функциональным, конструктивным и эксплуатационным характеристикам средств технического обеспечения системы.

Требования к метрологическому обеспечению. Приводят: предварительный перечень измерительных каналов; требования к точности измерений параметров и (или) к метрологическим характеристикам измерительных каналов; требования к метрологической совместимости технических средств системы; перечень управляющих и вычислительных каналов системы, для которых необходимо оценивать точностные характеристики; требования к метрологическому обеспечению технических и программных средств, входящих в состав измерительных каналов системы, средств, встроенного контроля, метрологической пригодности измерительных каналов и средств измерений, используемых при наладке и испытаниях системы; вид метрологической аттестации (государственная или ведомственная) с указанием порядка ее выполнения и организаций, проводящих аттестацию.

Требования к организационному обеспечению. Приводят: к структуре и функциям подразделений, участвующих в функционировании системы или обеспечивающих эксплуатацию; к организации функционирования системы и порядку взаимодействия персонала АС и персонала объекта автоматизации; к защите от ошибочных действий персонала системы.

Требования к методическому обеспечению САПР. Приводят требования к составу нормативно-технической документации системы (перечень применяемых при ее функционировании стандартов, нормативов, методик и т. п.).

3.3.5. Состав и содержание работ по созданию (развитию) системы

Содержаться перечень стадий и этапов работ по созданию системы в соответствии с ГОСТ 24.601, сроки их выполнения, перечень организаций – исполнителей работ, ссылки на документы, подтверждающие согласие этих организаций на участие в создании системы, или запись, определяющую ответственного (заказчик или разработчик) за проведение этих работ.

Кроме того приводят: перечень документов по ГОСТ 34.201, предъявляемых по окончании соответствующих стадий и этапов работ; вид и порядок проведения экспертизы технической документации (стадия, этап, объем проверяемой документации, организация-эксперт); программу работ, направленных на обеспечение требуемого уровня надежности разрабатываемой системы (при необходимости); перечень работ по метрологическому обеспечению на всех стадиях создания системы с указанием их сроков выполнения и организаций-исполнителей (при необходимости).

3.3.6. Порядок контроля и приемки системы

Указывают: виды, состав, объем и методы испытаний системы и ее составных частей (виды испытаний в соответствии с действующими нормами, распространяющимися на разрабатываемую систему); общие требования к приемке работ по стадиям (перечень участвующих предприятий и организаций, место и сроки проведения), порядок согласования и утверждения приемочной документации; статус приемочной комиссии (государственная, межведомственная, ведомственная).

3.3.7. Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие

Здесь необходимо привести перечень основных мероприятий и их исполнителей, которые следует выполнить при подготовке объекта автоматизации к вводу АС в действие.

В перечень основных мероприятий включают: приведение поступающей в систему информации (в соответствии с требованиями к информационному и лингвистическому обеспечению) к виду, пригодному для обработки с помощью ЭВМ; изменения, которые необходимо осуществить в объекте автоматизации; создание условий функционирования объекта автоматизации, при которых гарантируется соответствие создаваемой системы требованиям,

содержащимся в ТЗ; создание необходимых для функционирования системы подразделений и служб; сроки и порядок комплектования штатов и обучения персонала.

Например, для АСУ приводят: изменения применяемых методов управления; создание условий для работы компонентов АСУ, при которых гарантируется соответствие системы требованиям, содержащимся в ТЗ.

3.3.8. Требования к документированию

Приводят: согласованный разработчиком и заказчиком системы перечень подлежащих разработке комплектов и видов документов, соответствующих требованиям ГОСТ 34.201 и НТД отрасли заказчика; перечень документов, выпускаемых на машинных носителях; требования к микрофильмированию документации; требования по документированию комплектующих элементов межотраслевого применения в соответствии с требованиями ЕСКД и ЕСПД; при отсутствии государственных стандартов, определяющих требования к документированию элементов системы, дополнительно включают требования к составу и содержанию таких документов.

3.3.9. Источники разработки

Должны быть перечислены документы и информационные материалы (технико-экономическое обоснование, отчеты о законченных научно-исследовательских работах, информационные материалы на отечественные, зарубежные системы-аналоги и др.), на основании которых разрабатывалось ТЗ и которые должны быть использованы при создании системы.

4. АРХИТЕКТУРЫ ПРОГРАММНЫХ СИСТЕМ

4.1. Планирование архитектуры

Современные методы разработки программного обеспечения предполагают обратную связь между всеми действующими лицами, от проектировщика до аналитика. Все эти лица являются участниками процесса создания архитектуры программной системы. Под архитектурой системы будем понимать структуру компонентов программной системы, взаимосвязи, а также принципы и нормы их проектирования и развития во времени. Прежде чем начать изучение процесса планирования архитектуры, необходимо познакомиться с понятием архитектурно-экономического цикла (АЭЦ).

4.1.1. Архитектурно-экономический цикл

Взаимоотношения между производственными задачами, требования к продукту, опыт архитектора, архитектуры и созданные системы образуют цикл с цепями обратной связи. Упомянутые цепи обратной связи изображены на рис. 4.1. Частично обратная связь поступает от самой архитектуры, частично – от построенной на ее основе системы.

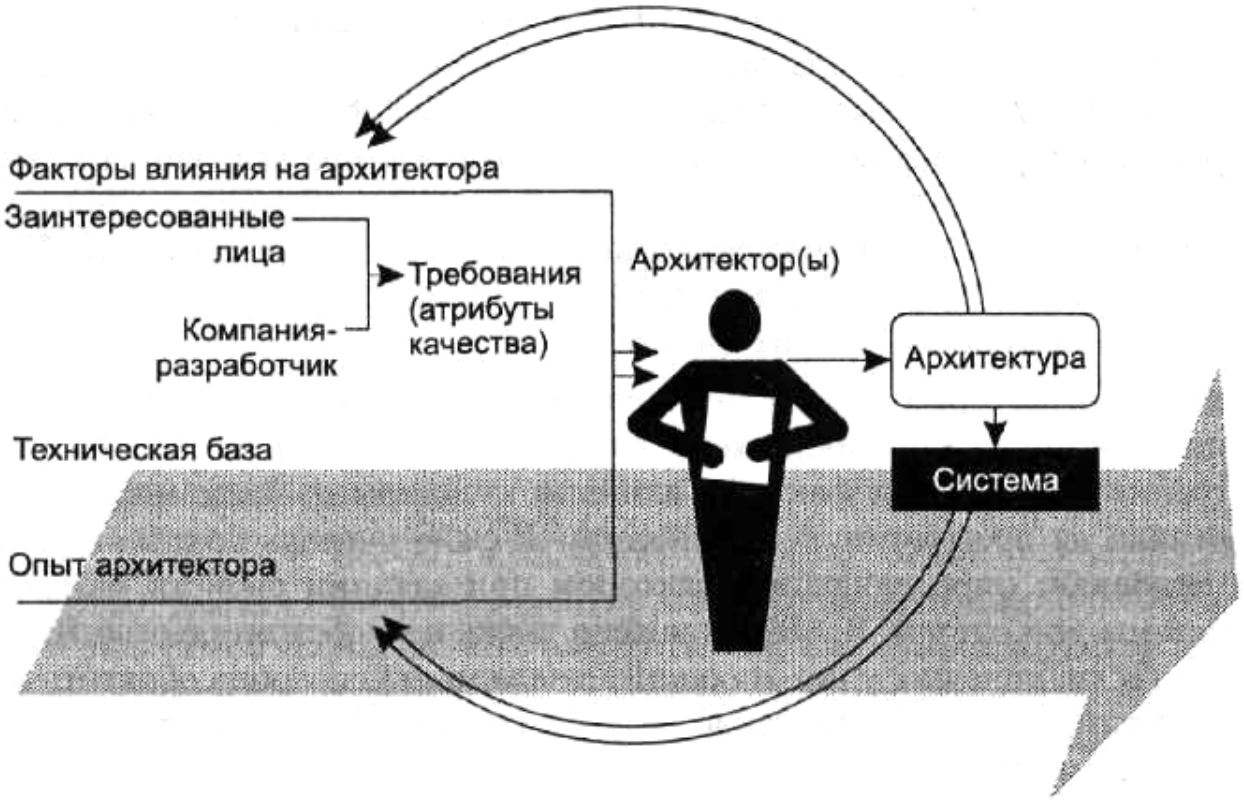


Рис. 4.1. Архитектурно-экономический цикл

Цикл выглядит следующим образом.

1. Архитектура влияет на структуру компании-разработчика. Архитектура обусловливает структуру системы; в частности (в этом мы сможем убедиться), она устанавливает набор блоков программного обеспечения, которое надлежит реализовать (или обеспечить их наличие другим путем), а затем интегрировать в рамках системы. Эти блоки составляют основу разработки структуры проекта. Группы разработчиков укомплектовываются именно по блокам; операции в рамках процессов разработки, тестирования и интеграции также выполняются в отношении блоков. Согласно графикам и бюджетам, ресурсы выделяются частями в расчете на отдельные блоки. Если компания наработала опыт конструирования семейств сходных систем, она будет вкладывать средства в повышение профессионального уровня участников сформированных по блокам групп разработчиков. Следовательно, группы встраиваются в структуру организации. Такой представляется обратная связь от архитектуры к компании-разработчику.

2. Архитектура способна оказывать воздействие на задачи компании-разработчика. Сконструированная на ее основе успешная система предоставляет компании возможность укрепиться в данном сегменте рынка. Такая архитектура предусматривает дальнейшее эффективное производство и размещение сходных систем, вследствие чего компания может откорректировать свои задачи и, воспользовавшись новым преимуществом, занять рыночную нишу. Так выглядит обратная связь от системы к компании-разработчику и конструируемым ею системам.

3. Архитектура может оказывать воздействие на требования, выдвигаемые заказчиком относительно следующей системы, – ее (если она основана на той же архитектуре, что и предыдущая) он может получить в более надеж ном варианте, быстрее и экономичнее, чем в том случае, если бы она конструировалась с чистого листа. Возможно, заказчик откажется от некоторых требований в пользу повышения экономичности. Готовые программные продукты несколько изменили требования, предъявляемые заказчиками, – не предназначенные для удовлетворения индивидуальных потребностей, они недороги и отличаются высоким качеством. На заказчиков, не слишком гибких по части своих требований, аналогичное воздействие оказывают линейки продуктов.

4. Процесс конструирования систем пополняет опыт архитектора, который он может применить при работе над последующими архитектурами, и, соответственно, расширяет базу опыта компании. Успех системы, построенной на основе инструментальных магистралей, .NET или инкапсулированных конечных автоматов, стимулирует построение аналогичных систем в дальнейшем. С другой стороны, неудачные варианты архитектуры редко используются повторно.

5. Иногда оказать сильное воздействие и даже внести изменения в культуру программной инженерии (техническую базу, в рамках которой обучаются и работают конструкторы) способны отдельные системы. Такой эф-

фект на индустрию в 1960-х и начале 1970-х гг. оказали первые реляционные базы данных, генераторы компиляторов и табличные операционные системы; в 1980-х – первые электронные таблицы и системы управления окнами. В 1990-х гг. в качестве такого рода катализатора выступила Всемирная паутина. Подобные инновации всегда находят отражение в последующих системах.

Из этих и некоторых других механизмов обратной связи и образуется архитектурно-экономический цикл. На рис. 4.1 изображены факторы влияния культуры и экономики компании-разработчика на программную архитектуру. В свою очередь архитектура оказывается основным определяющим фактором при задании свойств разрабатываемой системы или систем.

4.1.2. Программный процесс и архитектурно-экономический цикл

Программным процессом (software process) называются действия по организации, нормированию и управлению разработкой программного обеспечения. Ниже приведен перечень операций, направленных на создание программной архитектуры, ее применение для реализации проектного решения, а впоследствии – на реализацию или управление развитием целевой системы или приложения:

создание экономической модели системы;
выявление требований;
создание новой или выбор существующей архитектуры;
документирование и распространение сведений об архитектуре;
анализ или оценка архитектуры;
реализация системы на основе архитектуры;
проверка соответствия реализации архитектуре.

4.1.2.1. Этапы разработки архитектуры

Как следует из структуры АЭЦ, между различными этапами разработки архитектуры существуют развернутые отношения обратной связи

Создание экономической модели системы. Создание экономической модели не ограничивается оценкой потребности в системе на рынке. Этот этап исполняет существенную роль в контексте создания и сужения требований. Сколько должен стоить продукт? Каков целевой сегмент рынка? Насколько быстро продукт должен выйти на рынок? Должен ли он взаимодействовать с другими системами? Есть ли какие-нибудь системные ограничения, в рамках которых он должен существовать?

Все эти вопросы решаются с привлечением архитектора. Одного его, конечно, недостаточно, однако если при создании экономической модели

консультации с архитектором не проводились, то возможность достижения коммерческих задач становится проблематичной.

Выявление требований. Способов узнать, чего же, наконец, хотят заинтересованные лица, множество. В частности, в рамках объектно-ориентированного анализа для фиксации требований используются сценарии, или элементы Use Case. Для системы с повышенными требованиями к безопасности применяются более строгие методики – например, модели конечных автоматов и языки формальных спецификаций.

Применительно к конструируемой системе необходимо принять центральное, основополагающее решение – насколько в ней будут отражены другие, уже сконструированные системы. Поскольку в сегодняшних условиях найти систему, не имеющую сходств с другими системами, весьма непросто, методики выявления требований предполагают знание характеристик предшествующих систем.

Еще один способ выявления требований подразумевает моделирование. Опытные системы помогают моделировать нужное поведение, проектировать пользовательские интерфейсы и проводить анализ потребления ресурсов. Таким образом, в глазах заинтересованных лиц система становится «реальной», а процесс принятия решений по проектированию системы и ее пользовательского интерфейса значительно ускоряется.

Вне зависимости от методики выявления требований, желаемые атрибуты качества конструируемой системы обусловливают ее конечный вид. Для обеспечения отдельных атрибутов качества архитекторами уже давно применяются те или иные тактики. Архитектурные решения компромиссны, однако при специфицировании требований не все эти компромиссы очевидны. Со всей ясностью они проявляются только после создания архитектуры; тогда же принимаются решения относительно сортировки требований в соответствии с приоритетами.

Создание или выбор архитектуры. Фредерик Брукс (Fred Brooks) в своей знаменитой книге «Мифический человеко-месяц» красноречиво и убедительно доказывает, что основным условием стабильного проектирования системы является соблюдение концептуальной целостности, а она может проявиться лишь в узком кругу людей, совместно работающих над проектированием ее архитектуры [10].

Распространение сведений об архитектуре. Для того чтобы архитектура действительно стала основой проекта, ее суть необходимо четко и недвусмысленно донести до всех заинтересованных лиц. Разработчики должны понимать, что от них требуется, тестировщики должны осознавать структуру своих задач, менеджмент должен знать график и т. д. Для того чтобы этой цели можно было добиться, документирование архитектуры должно быть информативным, ясным и понятным людям различных профессий.

Анализ или оценка архитектуры. В процессе проектирования всегда рассматривается множество вариантов проекта. Некоторые из них забраковываются сразу. Из числа остальных в конечном итоге отбирается наиболее подходящий. Одна из глобальных задач, стоящих перед любым архитектором, заключается именно в том, чтобы сделать этот выбор рационально.

Оценить архитектуру на предмет атрибутов качества, которые она обеспечивает, совершенно необходимо – без этого нельзя быть уверенным в том, что конечная система сможет удовлетворить все потребности заинтересованных лиц. Все большее распространение получают методики анализа, ориентированные на оценку сообщаемых системе архитектурой атрибутов качества. Сценарные методики обеспечивают наиболее универсальную и эффективную оценку архитектуры. Самая зрелая методическая база характерна для метода анализа компромиссных архитектурных решений (Architecture Tradeoff Analysis Method, ATAM); метод анализа стоимости и эффективности (Cost Benefit Analysis Method, CBAM), с другой стороны, предусматривает крайне ценную возможность увязки архитектурных решений с их экономическим содержанием.

4.1.3. Суть программной архитектуры

Программная архитектура программы или вычислительной системы – это структура ее структур, т. е. изложение ее программных элементов, их внешних свойств и установленных между ними отношений.

Внешними (externally visible) свойствами называются те предположения, которые сторонние элементы могут выдвигать в отношении данного элемента, – в частности, они касаются предоставляемых элементом услуг, рабочих характеристик, устранения неисправностей, совместного использования ресурсов и т. д. Проанализируем представленное определение более подробно.

Во-первых, архитектура определяет программные элементы. В составе архитектуры приводится информация о взаимоотношениях элементов. Собственно говоря, все, что архитектура сообщает нам об элементах, ограничивается информацией об их взаимодействии. Итак, архитектура – это, в первую очередь, абстракция системы, в которой отсутствует информация об элементах, не имеющая отношения к тому, как они используют, используются, соотносятся или взаимодействуют с другими элементами. Практически во всех современных системах взаимодействие между элементами осуществляется посредством интерфейсов, которые поддерживают деление информации об элементах на публичную и приватную части. Так вот, архитектура имеет дело только с публичной частью; приватные детали элементов, те, что относятся исключительно к их внутренней реализации, в состав архитектуры не входят.

Во-вторых, из вышеприведенного определения ясно, что в состав любой системы может входить и входит целый ряд структур, а следовательно,

ни одной отдельно взятой структуры, которую можно было бы уверенно назвать архитектурой, не существует. В частности, все нетривиальные проекты делятся на блоки реализации; между этими блоками распределяются некоторые обязанности, и они же в большинстве случаев выступают в качестве базы для разделения задач между группами программистов. В таких элементах наряду с программами и данными, доступными для вызова или обращения со стороны других программных средств и блоков реализации, содержатся приватные программы и данные. В крупных проектах эти элементы почти всегда разделяются на мелкие части, а обязанности по работе с ними распределяются между несколькими мелкими группами разработчиков. Довольно часто описания систем составляются при помощи таких структур. Ориентированные на разделение функций системы между различными группами исполнителей реализации, они отличаются чрезмерной статичностью.

Другие структуры в большей степени ориентируются на взаимодействие элементов в период прогона с целью исполнения функции системы. Представим, что систему предполагается сконструировать в виде ряда параллельных процессов. В этом случае часто применяется другая структура, включающая процессы периода прогона, программы из вышеописанных блоков реализации, совместно формирующие отдельные процессы, а также установленные между этими процессами отношения синхронизации.

Можем ли мы взять любую из этих структур в отдельности и назвать ее архитектурой? Нет, не можем – и это несмотря на то, что все они содержат архитектурные сведения. В состав любой архитектуры эти структуры входят наравне со многими другими. В этой связи, очевидно, что, поскольку в составе архитектуры может содержаться несколько структур, в ней должны присутствовать несколько элементов (например, блок реализации и процессы), несколько вариантов взаимодействия между элементами (например, разбиение на составляющие и синхронизация) и даже несколько контекстов (например, время разработки и время прогона). Представленное определение не устанавливает сущность архитектурных элементов и отношений. Является ли программный элемент объектом? процессом? библиотекой? базой данных? коммерческим изделием? Он может быть чем угодно, причем возможности не ограничиваются вышеприведенными вариантами.

В-третьих, согласно нашему определению, программная архитектура есть у любой вычислительной системы с программным обеспечением. Связано это с тем, что любую систему можно представить как совокупность ее элементов и установленных между ними отношений. В простейшем случае система сама по себе является элементом, не представляющим, вероятно, никакого интереса и бесполезным, однако тем не менее согласующимся с понятием «архитектура». То обстоятельство, что архитектура есть у каждой системы, совершенно не означает, что она является общеизвестной. Кто знает, быть может, специалистов, спроектировавших систему, уже не найти, документация тоже куда-то исчезла (или ее никогда не существовало), исходный

код потерян (или не поставлялся), а остался лишь исполняемый двоичный код. Именно в этом случае различие между архитектурой системы и ее представлением становится очевидным. К сожалению, архитектура иногда существует самостоятельно, без описаний и спецификаций; в этой связи существенную важность приобретают документирование и реконструкция архитектуры.

В-четвертых, если поведение отдельно взятого элемента можно зафиксировать или выявить с точки зрения других элементов, то это поведение входит в состав архитектуры. Именно оно позволяет элементам взаимодействовать друг с другом, а взаимодействие, как известно, отражается в архитектуре в обязательном порядке. Этим, помимо прочего, объясняется, почему схемы из прямоугольников и линий, выдаваемые за архитектуры, таковыми не являются. Это не более чем рисунки; самое лучшее, что о них можно сказать, – это то, что они намекают на существование более четкой информации относительно фактических функций обозначенных элементов. Взглянув на наименования изображенных на подобной схеме прямоугольников (например, на них написано «база данных», «пользовательский интерфейс», «исполняемый файл» и т. д.), читатель хотя бы получит представление о функциональности и поведении соответствующих элементов.

Наконец, в представленном определении не отражено качество архитектуры системы, иначе говоря, никаких утверждений относительно перспектив соответствия системы установленным требованиям к поведению, производительности и жизненному циклу в ней нет. Поскольку метод проб и ошибок (предполагающий произвольный выбор архитектуры и последующее конструирование на ее основе системы под лозунгом «Будь что будет») в качестве оптимального способа выбора архитектуры для системы нас совсем не устраивает, то далее рассмотрим вопросы оценки архитектуры и архитектурного проектирования.

4.1.3.1. Архитектурные образцы, эталонные модели и эталонные варианты архитектуры

В промежутке между схемами из прямоугольников и линий – простейшими «набросками» архитектур – и комплексными архитектурами, укомплектованными всей необходимой информацией о системах, существуют многочисленные переходные этапы. Каждый такой этап есть результат принятия ряда архитектурных решений, совокупность архитектурных альтернатив. Некоторые из них сами по себе имеют определенную ценность. Прежде чем переходить к анализу архитектурных структур, рассмотрим три промежуточных этапа.

1. Архитектурный образец – это описание типов элементов и отношений и изложение ряда ограничений на их использование. Образец имеет смысл рассматривать как совокупность ограничений, накладываемых на ар-

хитектуру, – конкретнее, на типы элементов и образцы их взаимодействия; на основе этих ограничений складывается ряд или семейство соответствующих им вариантов архитектуры. К примеру, одним из общеупотребительных архитектурных образцов является клиент-сервер. Клиент и сервер – это два типа элементов; их взаимодействие описывается посредством протокола, при помощи которого сервер взаимодействует со всеми своими клиентами. Термин «клиент-сервер» по смыслу лишь предполагает множественность клиентов; конкретные клиенты не перечисляются, и речи о том, какая функциональность, помимо реализации протоколов, характерна для клиентов или для сервера, не идет. Согласно этому (неформальному) определению, образцу «клиент-сервер» соответствует бесчисленное количество различных вариантов архитектуры, причем все они чем-то отличаются друг от друга. Несмотря на то что архитектурный образец не является архитектурой, он все же содержит весьма полезный образ системы – он накладывает на архитектуру, а следовательно, и на систему, полезные ограничения.

У образцов есть один крайне полезный аспект – дело в том, что они демонстрируют известные атрибуты качества. Именно поэтому архитекторы выбирают образцы не наугад, а исходя из определенных соображений. Некоторые образцы содержат известные решения проблем, связанных с производительностью, другие предназначаются для систем с высокими требованиями к безопасности, третьи успешно реализуются в системах с высокой готовностью. Во многих случаях выбор архитектурного образца оказывается первым существенным решением архитектора.

Синонимичным архитектурному образцу является общеупотребительный термин «архитектурный стиль» (architectural style).

Эталонная модель – это разделение между отдельными блоками функциональных возможностей и потоков данных. Эталонной моделью называется стандартная декомпозиция известной проблемы на части, которые, взаимодействуя, способны ее разрешить. Поскольку эталонные модели имеют происхождение в опыте, их наличие характерно только для сформировавшихся предметных областей. Вы можете назвать стандартные элементы компилятора или системы управления базами данных? А в общих словах объяснить, как эти элементы сообща решают свою общую задачу? Если можете, значит, вы знакомы с эталонной моделью этих приложений.

Эталонная архитектура – это эталонная модель, отображенная на программные элементы (которые сообща реализуют функциональность, определенную в эталонной модели) и потоки данных между ними. В то время как эталонная модель обеспечивает разделение функций, эталонная архитектура отображает эти функции на декомпозицию системы. Соответствие может быть как однозначным, так и неоднозначным. В программном элементе может быть реализована как отдельная часть функции, так и несколько функций сразу.

Эталонные модели, архитектурные образцы и эталонные архитектуры не являются вариантами архитектуры; это не более чем полезные понятия,

способствующие фиксации отдельных элементов архитектуры. Каждый из них появляется как результат проектных решений, принимаемых на самых ранних этапах. Отношение между этими проектными элементами представлено на рис. 4.2.

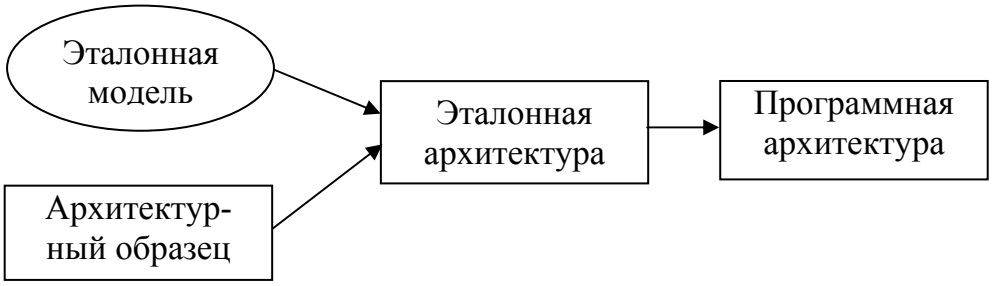


Рис. 4.2. Отношения между эталонными моделями, архитектурными образцами, вариантами эталонной и программной архитектуры

Аналогии архитектуры с другими значениями этого слова проводятся весьма часто. Как правило, архитектура ассоциируется с физической структурой (здания, улицы, аппаратура) и физическим расположением. Архитектор, разрабатывающий план здания, имеет целью обеспечить его доступность, эстетическую привлекательность, освещенность, эксплуатационную надежность и др. Программный архитектор в процессе проектирования системы должен стремиться к обеспечению таких характеристик, как параллелизм, модифицируемость, практичность, безопасность и т. д.; кроме того, он призван установить баланс между требованиями к этим характеристикам.

Аналогии между зданиями и программными системами не очень надежны, они слишком быстро оказываются несостоятельными. Хороши они тем, что помогают осознать важность позиции наблюдателя и прийти к выводу о множественности значений понятия «структура» в зависимости от мотивов ее изучения. Точное определение программной архитектуры значительно менее существенно, чем анализ сущности этого понятия.

4.1.3.2. Архитектурные структуры и представления

Современные программные системы настолько сложны, что разбирать их в комплексе крайне сложно. Приходится концентрировать внимание на одной или нескольких структурах программной системы. Для того чтобы рассуждать об архитектуре осмысленно, мы должны определиться с тем, какая структура или какие структуры в данный момент являются предметом обсуждения, о каком представлении (view) архитектуры мы говорим.

Рассматривая представление архитектуры, мы будем употреблять связанные между собой понятия структуры (structure) и представления (view). Представление – это отображение ряда связанных архитектурных элементов в том виде, в котором ими оперируют заинтересованные в системе лица.

В нем фиксируются отображения совокупности элементов и установленных между ними связей. Структура же – это собственно ряд элементов, существующих в рамках программного или аппаратного обеспечения. В частности, модульная структура представляет собой набор модулей системы с указанием их организации. Модульное представление есть отображение этой структуры, документированное и применяемое теми или иными заинтересованными лицами. Несмотря на то что эти термины иногда используются как синонимы, мы намерены придерживаться приведенных определений.

Архитектурные структуры подразделяются на три общие группы, в каждую из которых включается элементы определенного характера.

Модульные структуры. Элементами таких структур являются модули – блоки реализации. Модули предполагают рассмотрение системы с точки зрения кода. Им, как отдельным областям, выделяются определенные функциональные обязанности. Особого внимания тому, как конечное программное обеспечение заявит себя в период прогона, в данном случае не уделяется. Модульные структуры позволяют отвечать на такие вопросы, как: Какие основные функциональные обязанности несет данный модуль? К каким программным элементам он может обращаться? Какое программное обеспечение он фактически использует? Между какими модулями установлены отношения обобщения или специализации (например, наследования)?

Структуры «компонент и соединитель». В данном случае элементами являются компоненты (основные единицы вычислений) и соединители (инструменты взаимодействия между компонентами) периода прогона. Среди вопросов, на которые отвечают структуры «компонент и соединитель», – такие, например, как: Каковы основные исполняемые компоненты и как происходит их взаимодействие? Каковы основные совместно используемые хранилища данных? Какие части системы воспроизводятся? Каким образом по системе проходят данные? Какие элементы системы способны исполняться параллельно? Какие структурные изменения происходят с системой во время ее исполнения?

Структуры распределения. Структуры распределения демонстрируют связь между программными элементами, с одной стороны, и элементами одной или нескольких внешних сред, в которых данное программное обеспечение создается и исполняется, – с другой. Они отвечают на вопросы: на каком процессоре исполняется данный программный элемент? в каких файлах каждый элемент хранится в ходе разработки, тестирования и конструирования системы? каким образом программные элементы распределяются между группами разработчиков?

Программные структуры. Наиболее распространенные и полезные программные структуры представлены на рис. 4.3.

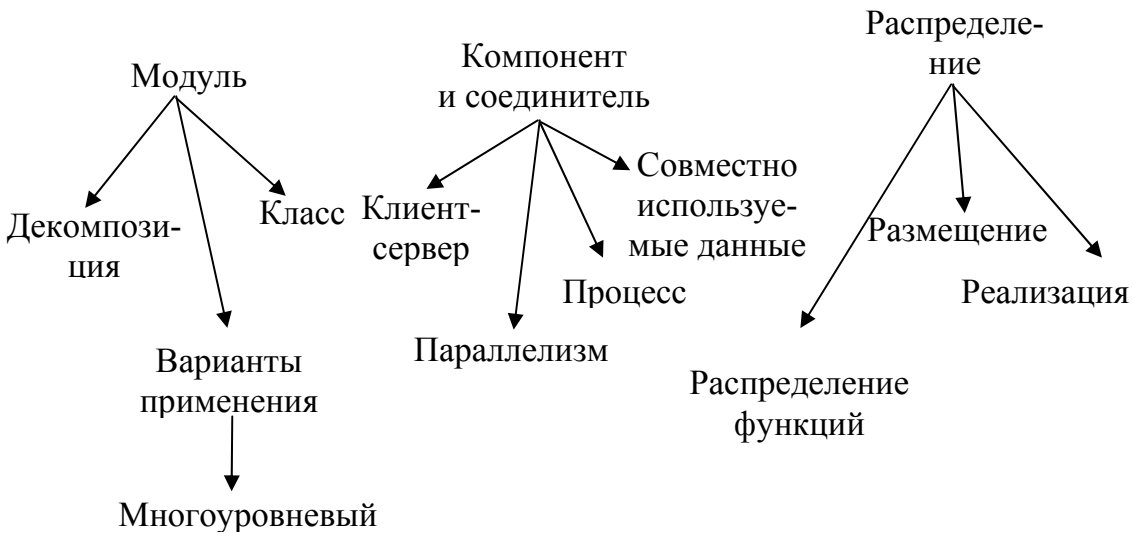


Рис. 4.3. Стандартные структуры программной архитектуры

Модуль. Модульные структуры делятся на следующие разновидности.

Декомпозиция. В качестве блоков выступают модули, между которыми установлены отношения «является подмодулем...»; таким образом, крупные модули в рекурсивном порядке разлагаются на меньшие, и процесс этот завершается только тогда, когда мелкие модули становятся вполне понятными. Модули в рамках этой структуры часто используются в качестве отправной точки для последующего проектирования – архитектор перечисляет блоки, с которыми ему предстоит работать, и в расчете на более подробное проектирование, а также, в конечном итоге, на реализацию распределяет их между модулями. У модулей во многих случаях есть связанные продукты (например, спецификации интерфейсов, код, планы тестирования и т. д.). Структура декомпозиции в значительной степени обеспечивает модифицируемость системы – при этом складывается ситуация, когда наиболее вероятные изменения приходятся на долю нескольких небольших модулей. Довольно часто декомпозиция задействуется как основа для организации проекта, включающая структуру документации, планы интеграции и тестирования. Иногда блоки этой структуры получают оригинальные названия (от пользующихся ими организаций). К примеру, в ряде стандартов Министерства обороны США определяются элементы конфигурации компьютерных программ (Computer Software Configuration Items, CSCI) и компоненты компьютерных программ (Computer Software Components, CSC), которые представляют собой не что иное, как блоки модульной декомпозиции.

Варианты использования. Блоками этой важной, но не слишком распространенной структуры могут быть либо модули, либо (в случаях, когда требуется более мелкая структура) процедуры или ресурсы интерфейсов модулей. Между такими блоками устанавливаются отношения использования (uses relationship). Если для обеспечения правильности первого блока требуется наличие правильной версии (в отличие от заглушки) второго, то последний используется первым. Структура использования полезна при конструировании систем, которые либо легко расширяются дополнительными функ-

циями, либо предполагают возможность быстрого извлечения полезных функциональных подмножеств. Способность без труда разбить рабочую систему на ряд подмножеств подразумевает возможность инкрементной разработки – многофункционального конструкторского приема.

Многоуровневая. Если отношения использования в рамках этой структуры находятся под строгим, особым образом осуществляемым контролем, возникает система уровней – внутренне связанных наборов родственных функций. В рамках строго многоуровневой структуры уровень *n* может обращаться к услугам только в том случае, если они предоставляются уровнем *n*–1. На практике это правило существует в виде многочисленных вариантов (которые частично снимают представленное структурное ограничение). Уровни во многих случаях проектируются в виде абстракций (виртуальных машин), которые, стараясь обеспечить переносимость, скрывают детали реализации нижележащих уровней от вышележащих.

Класс, или обобщение. Блоки модулей в рамках этой структуры называются классами. Отношения между ними строятся по образцам: «наследует от...» и «является экземпляром...». Данное представление способствует анализу коллекций сходного поведения или сходных возможностей (например, классов, которые наследуют от других классов) и параметрических различий, фиксация которых производится путем определения подклассов. Структура классов также позволяет анализировать вопросы повторного использования и инкрементного введения функциональности.

Компонент и соединитель. Среди структур данного вида выделяются следующие.

Процесс или сообщающиеся процессы. Подобно всем прочим структурам «компонент и соединитель», эта является ортогональной по отношению к модульным структурам, а связана она с динамическими аспектами исполняемой системы. В качестве блоков в данном случае выступают процессы или потоки, связь между которыми устанавливается путем передачи данных, синхронизации и/или операций исключения. Здесь, как и во всех остальных структурах «компонент и соединитель», действует отношение прикрепления (attachment), демонстрирующее связь компонентов друг с другом. Структура процессов существенно облегчает конструирование рабочей производительности и готовности системы.

Параллелизм. Данная структура «компонент и соединитель» позволяет архитекторам выявлять перспективы параллелизма и локализовывать возможности состязаний за ресурсы. В качестве блоков выступают компоненты, а соединители играют роль логических потоков. Логическим потоком называется такая последовательность вычислений, которую впоследствии, в ходе процесса проектирования, можно связать с отдельным физическим потоком. Структура параллелизма задействуется на ранних этапах процесса проектирования, способствуя выявлению требований к организации параллельного исполнения.

Совместно используемые данные, или репозиторий. В состав данной структуры входят компоненты и соединители, обеспечивающие создание, хранение и обращение к данным постоянного хранения. Она наилучшим образом приспособлена к таким ситуациям, когда система структурирована на основе одного или нескольких репозиториев совместно используемых данных. Она отражает производство и потребление данных программными элементами периода прогона, а в деле обеспечения высокой производительности и целостности данных эти сведения представляют большую ценность.

Клиент-сервер. Эта структура предназначена для систем, сконструированных в виде группы взаимодействующих клиентов и серверов. В качестве компонентов в данном случае выступают клиенты и серверы, а соединителями являются протоколы и сообщения, которыми они обмениваются в процессе обеспечения работоспособности системы. Такая структура требуется для разделения задач (обеспечивающего модифицируемость), физического распределения и выравнивания нагрузок (обеспечивающего эффективность исполнения).

Распределение. Среди структур распределения выделяются следующие.

Размещение. Структура размещения отражает распределение программного обеспечения между элементами аппаратной обработки и передачи данных. В качестве распределяемых элементов могут выступать программные продукты (как правило, процессы из представления «компонент и соединитель»), аппаратные объекты (процессоры) и каналы передачи данных. Отношения устанавливаются по распределению и демонстрируют физические устройства, на которых размещаются программные элементы; возможны также отношения миграции, однако они устанавливаются только в случае динамического распределения. Настоящее представление позволяет инженерам анализировать производительность, целостность данных, готовность и безопасность. Все эти характеристики чрезвычайно важны в условиях распределенных и параллельных систем.

Реализация. Данная структура демонстрирует отображение программных элементов (обычно модулей) на файловую структуру (структуры) в условиях разработки системы, интеграции и управления конфигурациями. Это крайне важно в контексте организации разработки и процессов конструирования.

Распределение функций. Данная структура обеспечивает разделение обязанностей по реализации и интеграции модулей между соответствующими группами разработчиков. Наличие в составе архитектуры структуры распределения функций делает очевидным, что при принятии соответствующих решений учитывались как архитектурные, так и организационные факторы. Архитектору должно быть известно, какие навыки требуются от разных групп разработчиков. Кроме того, если речь идет о масштабных распределенных проектах с несколькими источниками, на основе структуры распределения функций можно выявлять функциональные сходства и назначать их

одной группе разработчиков, отказываясь, таким образом, от их стихийной многократной реализации.

Общая схема программных структур приводится в табл. 4.1. В ней рассматривается значения элементов, отношения, характерные для каждой из структур, и варианты их практического применения.

Таблица 4.1

Архитектурные структуры системы

Программная структура	Отношения	Варианты практического применения
Декомпозиция	«Является подмодулем...»; «пользуется скрытой информацией совместно с...»	Распределение ресурсов, структурирование и планирование проекта; информационная закрытость, инкапсуляция; управление конфигурациями
Варианты использования	«Требует наличия...»	Конструирование подмножеств; инженерные расширения
Многоуровневая	«Требует наличия...», «обращается к услугам...», «обобщает...»	Инкрементная разработка; реализация систем на основе переносимости «виртуальных машин»
Классы	«Является экземпляром...», «использует метод доступа из...»	В объектно-ориентированных системах проектирования, производящих на основе универсального шаблона быстрые, почти идентичные реализации
Клиент-сервер	«Обменивается данными с...», «зависит от...»	Распределенное функционирование; разделение задач; анализ производительности; выравнивание нагрузки
Процесс	«Исполняется параллельно с...», «может исполняться параллельно с...», «исключает», «предшествует» и т. д.	Анализ сроков; анализ производительности
Параллелизм	«Исполняется в одном логическом потоке»	Выявление местоположений в которых потоки могут разветвляться, объединяться, создаваться и уничтожаться

Окончание табл. 4.1

Программная структура	Отношения	Варианты практического применения
Совместно используемые данные	«Производит данные», «потребляет данные»	Производительность; целостность данных, модифицируемость
Размещение	Распределение, миграция	Анализ производительности, готовности и защиты
Реализация	«Хранится в...»	Управление конфигурациями, интеграция, тестирование
Распределение функций	«Назначается...»	Управление проектом, оптимальное использование интеллектуальных ресурсов, управление общностью

Как правило, структура системы анализируется с точки зрения ее функциональности; при этом мы забываем о других ее свойствах: физическом распределении, взаимодействии процессов и синхронизации – все эти свойства обязательно учитываются на уровне архитектуры. Каждая структура содержит метод анализа тех или иных значимых атрибутов качества. К примеру, для того чтобы создать легко расширяемую или сокращаемую систему, необходимо *сконструировать* (именно сконструировать, а не просто зафиксировать) структуру использования. Структура процессов *конструируется* с целью исключения взаимоблокировки и расширения узких мест. Структура декомпозиции модулей *конструируется* в расчете на производство модифицируемых систем и т. д. Каждая подобная структура снабжает архитектора оригинальным представлением системы и дополнительной базовой точкой проектирования.

4.2. Проектирование архитектуры

4.2.1. Атрибутный метод проектирования

Атрибутный метод проектирования (Attribute-Driven Design, ADD) – это методика определения программной архитектуры, в которой процесс декомпозиции основывается на предполагаемых атрибутах качества продукта. Это рекурсивный процесс декомпозиции, на каждом из этапов которого происходит отбор тактик и архитектурных образцов, удовлетворяющих тем или иным сценариям качества, а также распределение функциональности, направленное на конкретизацию типов модулей данного образца. В контексте жизненного цикла ADD располагается сразу после анализа требований, а начинается он, как мы уже говорили, в тот момент, когда об архитектурных мотивах можно говорить с достаточной степенью уверенности.

Результатом применения ADD являются первые несколько уровней представления декомпозиции модулей архитектуры, а также все другие связанные с ними представления. Не стоит, впрочем, думать, что после ADD становятся известны все детали представлений, – система описывается как набор контейнеров функциональности и существующих между ними взаимосвязей. Во всем процессе проектирования это первый случай сочленения архитектуры, результаты которого, естественно, весьма приблизительны. Тем не менее в контексте реализации предполагаемых атрибутов качества это очень важный момент, поскольку именно здесь закладываются основы достижения функциональности. Различие между архитектурой, являющейся результатом выполнения ADD, и архитектурой, готовой к реализации, лежит в плоскости принятия подробных проектных решений. В частности, это может быть выбор между конкретными объектно-ориентированными образцами проектирования, с одной стороны, и промежуточным программным обеспечением, применение которого сопряжено с многочисленными архитектурными ограничениями, – с другой. Архитектура, спроектированная средствами ADD, предусматривает отсрочку принятия этого решения, благодаря чему достигается большая гибкость.

С участием общих сценариев, а также тактик и образцов можно создать ряд различных процессов проектирования. Отличаются они принятыми принципами деления проектных работ и содержанием процесса проектирования.

4.2.1.1. Этапы ADD

1. *Выбор модуля для декомпозиции*. Как правило, в качестве исходного модуля берется система в целом. Все необходимые входные данные (ограничения, функциональные требования и требования к качеству) для него должны быть известны.

2. Уточнение модуля в несколько этапов:

а) выбор архитектурных мотивов из набора конкретных сценариев реализации качества и функциональных требований. На этом этапе определяются наиболее важные в контексте проведения декомпозиции моменты;

б) выбор архитектурного образца, соответствующего архитектурным мотивам. Создание (или выбор) образца, тактики которого позволяют реализовать эти мотивы. Выявление дочерних модулей, необходимых для реализации этих тактик;

в) конкретизация модулей, распределение функциональности из элементов Use Case, составление нескольких представлений;

г) определение интерфейсов дочерних модулей. Декомпозиция имеет своим результатом новые модули, а также ограничения на типы взаимодействия между ними. Для каждого из модулей эти сведения следует зафиксировать в документации по интерфейсу;

д) проверка и уточнение элементов Use Case в сценариях качества и наложение, исходя из них, ограничений на дочерние узлы. На этом этапе мы проверяем, все ли факторы учли, а также, в расчете на последующую декомпозицию или реализацию, подготавливаем дочерние модули.

3. Вышеперечисленные этапы следует выполнить в отношении всех нуждающихся в дальнейшей декомпозиции модулей.

4.2.2. Создание макета системы

После проектирования архитектуры и формирования рабочих групп можно приступать к конструированию макета системы. Цель этого этапа в том, чтобы обеспечить основополагающую возможность реализации функциональности системы в предпочтительном (с точки зрения задач проекта) порядке.

Согласно классической методике программной инженерии, следует «заглушать» секции кода так, чтобы различные части системы можно было вводить в действие и тестировать по отдельности. О каких именно частях идет речь? Последовательность реализации всегда можно установить по характеристикам архитектуры.

В первую очередь следует реализовать те программы, которые связаны с исполнением и взаимодействием между архитектурными компонентами. В системе реального времени это может быть планировщик, в системе на основе правил – процессор правил (с прототипом набора правил), управляющий их активизацией, в многопроцессной системе – механизмы синхронизации процессов, а в клиент-серверной системе – механизм координации действий клиента и сервера. Базовый механизм взаимодействия во многих случаях выражается в виде промежуточных программ стороннего производства; если это так, реализация заменяется установкой. Помимо базовой инфраструктуры передачи данных или взаимодействия имеет смысл ввести в действие ряд простейших функций, инициирующих механическое поведение. На этом этапе в вашем распоряжении уже появляется исполняемая система. Это та основа, на которую можно начинать насаживать дополнительную функциональность.

Теперь выбирайте – какие функциональные элементы следует ввести в эту систему. Решение в данном случае принимается под влиянием нескольких факторов: во-первых, из побуждения снизить риск, обратившись к наиболее трудным областям в первую очередь, во-вторых, исходя из возможностей и специализации имеющегося персонала, и, в-третьих, из соображений «выбросить» продукт на рынок как можно быстрее.

Приняв решение относительно введения в систему очередной порции функциональности, обратитесь к структуре использования – она сообщит о том, какие еще программные средства системы должны корректно исполняться (другими словами, развиться из состояния банальной заглушки в пол-

ноценные элементы), для того чтобы реализация выбранных функций стала возможной.

Таким вот образом в систему можно вводить все новые и новые элементы, пока они не закончатся. На любом из таких этапов действия по интеграции и тестированию не представляют большой сложности; равным образом легко обнаружить источник недавно появившихся неисправностей. Чем меньше приращение, тем более предсказуемы бюджет и график и тем больше диапазон решений по поставке у управленцев и специалистов по маркетингу.

Заглушенные элементы, несмотря ни на что, приближают момент завершения работы над системой. Поскольку заглушки относятся к тем же интерфейсам, которые будут присутствовать в окончательной версии системы, даже в отсутствии полноценной функциональности они помогают уяснить и протестировать взаимодействие между компонентами. Проверить это взаимодействие с помощью компонентов-заглушек можно двумя способами: путем генерирования жестко закодированных искусственных выходных данных или считывания таких данных из файла. Кроме того, они способны генерировать искусственную нагрузку, по результатам которой можно приблизительно определить, сколько времени уйдет на фактическую обработку данных в законченной рабочей версии системы. Это помогает на ранней стадии проектирования выявить требования по производительности системы, включая рабочие взаимодействия и узкие места.

Согласно Кусумано (Cusumano) и Селби (Selby), компания Microsoft выстраивает свою стратегию на основе эволюционного жизненного цикла поставки (Evolutionary Delivery Life Cycle). По той версии методики, которой пользуется Microsoft, «завершенный» макет системы создается на раннем этапе жизненного цикла продукта, а впоследствии с некоторой регулярностью (во многих случаях ежедневно), строятся «рабочие» версии с сокращенной функциональностью. В результате получается рабочая система, о достаточности характеристик которой можно принять решение в любой момент и которую в любой же момент можно развернуть. Есть, впрочем, одна проблема, которую следует предусмотреть, – дело в том, что та рабочая группа, которая заканчивает свою часть системы первой, задает интерфейс, которому должны соответствовать все последующие системы. Это обстоятельство ставит в невыгодное положение наиболее сложные части системы – их разработка сопряжена со значительно большими объемами аналитических действий, вследствие чего вероятность первоочередного определения их интерфейсов сильно снижается. Таким образом, и без этого сложные подсистемы становятся еще более сложными. По нашему убеждению, все согласования по поводу интерфейсов следует проводить на этапе создания макета системы – при таком условии на последующих стадиях разработки можно будет обратить большее внимание на достижение эффективности.

4.3. Документирование программной архитектуры

4.3.1. Варианты применения архитектурной документации

Характер архитектуры любой системы обусловливается предъявляемыми к ней требованиями; это утверждение справедливо и по отношению к документации архитектуры, другими словами, содержание документации зависит от предполагаемых вариантов ее применения. Документация ни при каких обстоятельствах не может быть универсальной. С одной стороны, она должна быть абстрактной и, следовательно, доступной для понимания новыми сотрудниками, но с другой – весьма детальной – настолько, чтобы ее можно было использовать как план проведения анализа. Между архитектурной документацией, предназначенной, скажем, для проведения анализа безопасности, и архитектурной документацией для изучения конструкторами должны существовать серьезные различия. С другой стороны, у этих вариантов будет мало общего с содержанием руководства для ознакомления, предназначенного новому сотруднику.

Архитектурная документация одновременно инструктивна и описательна. Ограничивая диапазон возможных решений, с одной стороны, и перечисляя принятые ранее проектные решения по системе, – с другой, она обязывает соблюдать определенные принципы.

Из всего этого следует, что заинтересованные в составлении документации лица преследуют разные цели – от представленной в ней информации они требуют разной направленности, разных уровней детализации и разных трактовок. Предполагать, что один и тот же архитектурный документ будет одинаково воспринят всеми без исключения заказчиками, наивно. Стремиться следует к тому, чтобы любое заинтересованное лицо смогло быстро найти необходимую информацию, как можно меньше в процессе ее поиска натыкаясь на незначительные (с его точки зрения) сведения.

В некоторых случаях простейшим решением представляется создание нескольких документов, предназначенных для разных заинтересованных лиц. Впрочем, как правило, задача ограничивается созданием единого комплекта документации с дополнительными облегчающими поиск сведений инструкциями.

Согласно одному из фундаментальных правил технической документации в общем и документации программной архитектуры в частности, составитель должен принимать позицию читателя. Без труда составленная, но трудная для восприятия (с точки зрения читателя – в данном случае заинтересованного лица) документация не найдет себе применения.

4.3.2. Представления

Одним из наиболее важных понятий документирования программной архитектуры является «представление» (view). С понятием представления,

которое можно кратко определить как способ фиксации структуры, связан основной принцип документирования программной архитектуры.

Документирование архитектуры подразумевает документирование всех значимых представлений с последующей фиксацией сведений, относящихся одновременно к нескольким представлениям.

Принцип этот полезен тем, что он помогает разбить проблему документирования архитектуры на ряд менее обширных элементов:

выбор значимых представлений;

документирование представления;

документирование сведений, относящихся к нескольким представлениям.

4.3.2.1. Выбор значимых представлений

Какие представления следует считать значимыми? Для ответа на этот вопрос необходимо знать заинтересованных лиц и предпочтительные для них способы пользования документацией. Лишь располагая этими сведениями, вы сможете составить удобный для них пакет документации. Любой вариант применения архитектуры как средства постановки задач конструкторов, основы для изучения системы, восстановления ее свойств или планирования проекта можно свести к отдельному заинтересованному лицу, предполагающему задействовать документацию архитектуры соответствующим образом. Не менее серьезное влияние на выбор представлений для дальнейшего документирования оказывают наиценнейшие для большинства заинтересованных в разработке системы лиц атрибуты качества. К примеру, многоуровневое представление (layered view) сообщает сведения о переносимости системы. По представлению размещения (deployment view) можно судить о производительности и надежности системы. И так далее. За отражение этих атрибутов качества в документации ратуют аналитики (а быть может, и сам архитектор), в задачу которых входит проверка архитектуры на предмет ее соответствия атрибутам качества.

Различные представления соответствуют отдельным задачам и вариантам использования. Именно по этой причине мы призываем к выбору того или иного представления или набора представлений. Их выбор зависит от задач потребителей документации. Представления ставят акценты на разные элементы системы и/или установленные между ними связи.

Представление отражает произвольный набор элементов системы и установленных между ними отношений. Следовательно, представлением может быть любое сочетание элементов и отношений, которое, по вашему мнению, имеет шансы оказаться интересным части заинтересованных лиц. Ниже приводится состоящая из трех этапов процедура подбора представлений для конкретного проекта.

1. Составьте список возможных представлений. Начните с составления для своего проекта таблицы заинтересованных лиц/представлений. В столб-

цах выразите применимые к вашей системе представления. Некоторые из них (например, модульное представление и представление вариантов использования) носят универсальный характер; иные (многоуровневое представление, а также большинство представлений из группы «компонент и соединитель», в частности клиент-серверное представление и представление совместно используемых данных) подходят только для соответствующим образом спроектированных систем. Разобравшись с содержанием строк и столбцов, заполните все ячейки, отразив в них степень детализации сведений о каждом представлении, необходимую тем или иным заинтересованным лицам: здесь возможны произвольная детализация, обзор, средняя или высокая детализация.

2. Комбинируйте представления. Скорее всего, количество представлений, отраженных в составленном по результатам первого этапа списке, окажется недопустимым. Для того чтобы сократить список, найдите в таблице представления, требующие не более чем обзорной детализации или служащие ограниченному числу заинтересованных лиц. Проверьте, нельзя ли удовлетворить их запросы другим, более универсальным представлением. Затем найдите представления, которые можно комбинировать — выразить в одном сводном представлении информацию из нескольких исходных представлений. В небольших по масштабу проектах информативное содержание представления реализации, как правило, адекватно отражается в представлении декомпозиции на модули. Последнее, в свою очередь, хорошо сочетается с представлениями вариантов использования и многоуровневым представлением. Наконец, представление размещения можно скомбинировать с любым представителем группы «компонент и соединитель», отражающим распределение компонентов между аппаратными элементами, например с представлением процессов.

3. Расставляйте приоритеты. Представления, оставшиеся после выполнения второго этапа, должны соответствовать потребностям сообщества заинтересованных лиц. Теперь необходимо решить, какие из этих представлений имеют первостепенное значение. Конкретное решение зависит от деталей проекта; впрочем, вы в любом случае должны помнить, что полностью завершать работу над одним представлением, прежде чем приступать к следующему, совершенно не обязательно. Информация, детализированная на уровне обзора, представляет некоторую ценность, так что наши предпочтения на стороне метода разработки материала «в ширину». Кроме того, интересы одних заинтересованных лиц в ряде случаев ставятся выше интересов других.

4.3.3. Документирование представления

Стандартного шаблона документирования представлений не существует. Поэтому ниже речь пойдет о методике, доказавшей свою жизнеспособность в практической деятельности – стандартной семичастной структуре (seven-part standard organization). Во-первых, вне зависимости от того, каким

разделам вы отдадите предпочтение, ввести стандартную структуру совершенно необходимо. Распределение информации по отдельным разделам помогает составителю документации уверенно приступить к решению задачи и установить момент ее выполнения; что же касается читателя, то ему становится проще быстро отыскать интересующую информацию и пропустить все ненужное.

1. Первичное отображение (primary presentation) содержит перечень элементов представления и установленные между ними отношения. В первичном представлении должна содержаться (в форме словаря) та информация о системе, которую необходимо донести до читателя в первую очередь. Это, без сомнения, основные элементы и отношения представления, хотя в некоторых случаях они могут быть отражены не полностью. К примеру, в первичном отображении можно показать элементы и отношения, характерные для нормального режима работы, а сведения об обработке ошибок или исключений представить во вспомогательной документации.

Как правило, первичное отображение составляется в графической форме. Дело в том, что большинство графических нотаций годятся только для первичного отображения – для всех остальных элементов документации они не приспособлены. Неизменным спутником графического представления должен быть перечень условных обозначений, содержащий объяснение использованной нотации и символики или хотя бы указывающий на источники получения этих объяснений.

Иногда первичное отображение составляется в виде таблицы, которая по своему характеру прекрасно подходит для компактного представления больших объемов информации. Требование о кратком выражении наиболее значимых сведений о представлении распространяется и на текст.

2. Каталог элементов (element catalog) предназначен для более детального описания элементов и отношений как участвующих, так и не участвующих в первичном отображении. Архитекторы часто совершают одну и ту же ошибку – уделяя излишне серьезное внимание составлению первичного отображения, они забывают, что без дополнительной информации в нем мало толку. К примеру, если на диаграмме показаны элементы A, B и C, необходимо составить относительно детальное описание сущности этих элементов, их назначения и ролей, причем разместить эту информацию лучше всего в словаре представления. Скажем, для представления декомпозиции на модули характерно наличие элементов-модулей, различных вариантов отношения «является частью», а также свойств, определяющих обязанности каждого модуля. В представлении процессов содержатся элементы-процессы, отношения, определяющие синхронизацию и другие механизмы межпроцессного взаимодействия, а также свойства с временными параметрами.

Кроме того, в каталоге обязательно должны разъясняться все элементы и отношения, значимые для данного представления, но по каким-то причинам не показанные в первичном отображении.

3. На контекстной диаграмме (context diagram) должно быть показано отношение изображенной в представлении системы к своему окружению из словаря представления. К примеру, представление «компонент и соединитель» подразумевает демонстрацию взаимодействия отдельных компонентов и соединителей с внешними компонентами и соединителями через посредство интерфейсов и протоколов.

4. Руководство по изменчивости (variability guide) демонстрирует способы применения изменяемых параметров, входящих в состав показанной в данном представлении архитектуры. В некоторых архитектурах принятие решений откладывается до более поздних стадий процесса разработки, что не отменяет необходимость в составлении документации. Примеры изменчивости обнаруживаются во всех линейках программных продуктов, архитектура которых предусматривает возможность создания множества конкретных систем. В руководстве по изменчивости следует документировать все изменяемые параметры архитектуры, включая:

альтернативы, рассматриваемые при принятии того или иного решения. В модульном представлении такими альтернативами являются различные варианты параметризации модулей. В представлении «компонент и соединитель» могут быть отражены ограничения по дублированию, планированию или выбору протоколов. В представлении распределения это условия назначения конкретного программного элемента тому или иному процессору;

время связывания альтернатив. Одни решения принимаются в период проектирования, другие – в период производства, третьи – в период исполнения.

5. Предпосылки архитектурного решения (architecture background) – это раздел, в котором содержится обоснование отраженного в данном представлении проектного решения. Цель его – объяснить читателю, почему проект выглядит так, как он выглядит, и представить убедительные аргументы его превосходства. В состав этого раздела входят следующие элементы:

логическое обоснование, демонстрирующее предпосылки принятия проектных решений, отраженных в данном представлении, и причины отказа от альтернатив;

результаты анализа, оправдывающие проектное решение и объясняющие последствия модификаций;

отраженные в проектном решении допущения.

6. Глоссарий терминов (glossary of terms), применяемых в представлениях, и краткое описание каждого из них.

7. Другая информация. Содержание этого раздела зависит от методов работы конкретной организации. В частности, здесь можно разместить административные сведения: авторство, данные управления конфигурациями и историю изменений. Кроме того, архитектор волен разместить здесь систему ссылок на отдельные разделы сводки требований. Таким образом, речь идет об информации, которая, строго говоря, не имеет прямого отношения к архитектуре, но которую удобнее всего привести вместе с архитектурными сведениями.

4.3.3.1. Документирование поведения

Представления содержат структурную информацию о системе. Но для рассуждений о некоторых свойствах системы ее недостаточно. К примеру, для того чтобы уяснить вопросы взаимоблокировки, необходимо знать последовательность взаимодействий между элементами, которую структурная информация не отражает. Эти сведения, равно как возможности параллелизма и временные зависимости, характерные для взаимодействий (которые происходят в установленные моменты или по истечении установленных временных периодов), раскрывают поведенческие описания. Поведение документируется применительно к отдельному элементу или к множеству взаимодействующих элементов. Выбор в вопросах моделирования зависит от типа проектируемой системы. К примеру, если речь идет о встроенной системе реального времени, на первое место выходят свойства времени и время наступления событий. В системе банковского обслуживания значительно важнее фактического времени наступления событий оказывается их последовательность (например, последовательность элементарных транзакций и процедур отката). В зависимости от вида предполагаемых аналитических действий уместно обращаться к разным методикам моделирования и нотациям. Примерами поведенческих описаний в языке UML являются диаграммы последовательностей и схемы состояний. Подобные нотации весьма широко распространены.

Схемы состояний как формализм появились в 1980-х гг. и изначально предназначались для описания реактивных систем. Ряд реализованных в них полезных расширений дополняет традиционные диаграммы состояний (такие, как вложенность состояний и состояния «и») и придает выразительность абстракции и параллелизму моделей. Схемы состояний позволяют рассуждать о системе в целом. Предполагается отображение всех ее состояний, а методики анализа в отношении системы приобретают универсальный характер. Становятся возможными ответы на вопрос типа: всегда ли время отклика на данный стимул будет меньше 0,5 секунды?

Диаграмма последовательностей помогает документировать последовательность обменов стимулами. Она отражает кооперацию применительно к экземплярам компонентов и их взаимодействию, причем последнее представляется во временной последовательности. Вертикальное измерение при этом выражает время, а горизонтальное – различные компоненты. Диаграммы последовательностей позволяют строить умозаключения на основе конкретных сценариев использования. Они демонстрируют механизм реагирования системы на отдельные стимулы, иллюстрируют выбор путей в системе и отвечают на вопрос типа: какие параллельные операции приходят в момент реагирования системы на определенные стимулы в определенных условиях?

4.3.3.2. Документирование интерфейсов

Интерфейс (interface) – это граница, на которой встречаются, взаимодействуют или передают друг другу информацию две независимые сущности.

Очевидно, что интерфейсы элементов – носителей их внешне видимых другим элементам свойств – являются понятием архитектурным. Поскольку без них невозможны ни анализ, ни проектирование систем, документировать интерфейсы совершенно необходимо.

Под документированием интерфейса подразумевается указание его имени, идентификация, а также отражение всех синтаксических и семантических сведений о нем. Первые два элемента – указание имени и идентификация – обобщенно называются «сигнатура» интерфейса. Если в качестве ресурсов интерфейса выступают вызываемые программы, сигнатура обозначает их и определяет их параметры. Параметры определяются по порядку, типу данных и (иногда) по принципу возможности изменения их значений программой. Та информация о программе, которая содержится в сигнатуре, обычно приводится в заголовочных файлах C или C++ и в интерфейсах Java.

При всей полезности сигнатур (в частности, они делают возможной автоматическую проверку конструкций) ими все не исчерпывается. Соответствие сигнатур обеспечивает успешную компиляцию и/или компоновку системы, но совершенно не гарантирует достижение конечной цели – ее нормальное функционирование. Необходимая для этого информация относится к семантике интерфейса, которая сообщает, что происходит при активизации ресурсов.

Интерфейс документируется в форме спецификации – изложения свойств элемента, которые архитектор желает предать огласке. Это должна быть только та информация, которая необходима для организации взаимодействия с интерфейсом. Другими словами, архитектор должен решить, во-первых, какую информацию об элементе допустимо и уместно сообщить читателю и, во-вторых, какая информация, вероятнее всего, не будет подвержена изменениям. В процессе документирования интерфейса важно, с одной стороны, не раскрыть слишком много сведений, и с другой – не утаить необходимые данные. Разработчики не смогут наладить успешное взаимодействие с элементом, если о нем будет недостаточно информации. Избыток информации, в свою очередь, усложняет будущие изменения в системе и усиливает их влияние, делает интерфейс неудобным для восприятия. Для того чтобы достойно справиться с этой ситуацией, наибольшее внимание следует уделять не реализации элементов, а их взаимодействию с рабочими средами. Документированию подлежат только внешне видимые явления.

Элементы, присутствующие в виде модулей, часто напрямую соответствуют одному или нескольким элементам представления «компонент и соединитель». Как правило, интерфейсы элементов в представлении модулей и представлении «компонент и соединитель» схожи или идентичны и документировать их в обоих этих представлениях излишне. Поэтому спецификацию

интерфейса следует привести в модульном представлении, а в «компоненте и соединителе» поместить ссылку на нее и изложить индивидуальную для данного представления информацию. Кроме того, один модуль может оказаться общим для нескольких модульных представлений – например, декомпозиции на модули и использования. В таком случае, как и в предыдущем, спецификацию интерфейса следует привести в одном из этих представлений, а во всех остальных поставить соответствующую ссылку.

Шаблон для документирования интерфейсов. Ниже изложен один из вариантов стандартной структуры документации интерфейса. Некоторые ее элементы, если они оказываются бесполезными в конкретном применении, можно выкинуть, другие, наоборот, ввести. Значительно более важными, чем сама стандартная структура, представляются нам способы ее применения. Ваша цель должна заключаться в том, чтобы в точности изложить все внешне видимые взаимодействия интерфейсов, предусмотренных в проекте.

Индивидуальность интерфейса. Если у элемента несколько интерфейсов, их нужно как-то отличать друг от друга. Как правило, для этого интерфейсам присваиваются имена, в некоторых случаях сопровождающиеся указанием номера версии.

Предоставляемые ресурсы. Основным предметом документирования любого интерфейса должны быть предоставляемые им ресурсы. Они определяются синтаксисом, семантикой (описывающей следствия их применения) и ограничениями по использованию. Существует ряд нотаций, предназначенных для документирования синтаксиса интерфейса. Одна из них – язык описания интерфейсов (interface definition language, IDL), разработанный рабочей группой OMG, – применяется в сообществе CORBA. С помощью специальных языковых конструкций этого языка описываются типы данных, операции, атрибуты и исключения. Семантическую информацию можно выразить только в комментариях. В большинстве программных языков есть встроенные средства спецификации сигнатур элементов, примером чему – заголовочные файлы (.h) в C и спецификации пакетов в Ada. Наконец, на языке UML синтаксическая информация об интерфейсе выражается через стереотип interface. Для интерфейса необходимо хотя бы ввести имя; кроме того, архитектор волен составить для него сигнатуру.

Синтаксис ресурса. Здесь указывается сигнатура ресурса. Содержащейся в сигнатуре информации должно быть достаточно для того, чтобы любая сторонняя программа смогла корректно с точки зрения синтаксиса обратиться к программе, использующей данный ресурс. Таким образом, в сигнатуру входят имя ресурса, имена и логические типы данных его аргументов (если таковые имеются) и т. д.

Семантика ресурса. Здесь приводится описание результатов вызова ресурса, в частности:

присваивание значений данным, к которым может обращаться актер, вызывающий рассматриваемый ресурс, – от простого задания значения возвращаемого аргумента до обновления центральной базы данных;

события и сообщения, сигнализируемые/отправляемые в результате обращения к ресурсу;

предполагаемое поведение других ресурсов как результат обращения к рассматриваемому ресурсу (к примеру, если данный ресурс уничтожит некий объект, то последующие попытки обращения к этому объекту будут приводить к новому результату – ошибке);

результаты, наблюдаемые пользователем (встречающиеся в основном во встроенных системах: скажем, вызов программы, ответственной за включение бортового индикатора, приводит к наблюдаемому результату).

Кроме того, семантика должна прояснять вопросы, связанные с исполнением ресурса: будет ли он носить элементарный характер, можно ли его приостановить или прервать. Как правило, семантическая информация выражается средствами естественного языка. Для фиксации предусловий и постусловий – сравнительно простого и эффективного метода выражения семантики – специалисты во многих случаях прибегают к булевой алгебре. Еще одним средством передачи семантической информации являются следы – записи последовательностей операций и взаимодействий, описывающих реакцию элемента на тот или иной вариант использования.

Ограничения на использование ресурсов. В каких обстоятельствах возможно обращение к рассматриваемому ресурсу? Существует ли необходимость в предваряющей его считывание инициализации данных? Обусловливается ли вызов одного метода вызовом другого? Не установлены ли какие-то ограничения относительно количества актеров, имеющих возможность одномоментного взаимодействия с ресурсом? Возможно, в силу принадлежности элемента определенному актеру, он единственный, кто обладает полномочиями по модификации элемента, а все остальные актеры ограничиваются лишь считыванием. Не исключено также, что, согласно многоуровневой схеме безопасности, каждый актер может обращаться к строго определенным ресурсам или интерфейсам. Если рассматриваемый ресурс отталкивается от каких-либо допущений в своем окружении (например, требует наличия других ресурсов), их следует задокументировать.

Определения типов данных. Если какой-либо ресурс интерфейса задействует тип данных, отличный от предусмотренного соответствующим языком программирования, архитектор должен привести определение типа данных. Если он определен в другом элементе, достаточно установить ссылку на его документацию. Как бы то ни было, программисты, которые пишут элементы, обращаясь к такому ресурсу, должны знать: а) как объявлять переменные и константы типа данных; б) как в рамках этого типа данных записывать литеральные значения; в) какие операции и сравнения применимы к членам типа данных; г) как преобразовывать значения этого типа данных в данные других типов.

Определения исключений. Здесь предполагаются описания исключений, которые могут порождать ресурсы на данном интерфейсе. Поскольку одни и те же исключения во многих случаях характерны для множества ре-

сурсов, имеет смысл отделять список исключений каждого ресурса от их определений, причем последние размещать в специальном словаре. Настоящий раздел как раз исполняет роль словаря. Здесь же допустимо определять стандартное поведение обработки ошибок.

Характерная для интерфейса изменчивость. Предполагает ли данный интерфейс возможность конфигурирования элемента? Подобные параметры конфигурации (configuration parameters), а также их воздействие на семантику интерфейса подлежат документированию. В качестве примеров изменчивости можно привести емкость видимых структур данных и рабочие характеристики соответствующих алгоритмов. Для каждого параметра конфигурации архитектор должен зафиксировать имя, диапазон значений и время связывания его фактических значений.

Характеристики атрибутов качества интерфейса. Архитектор должен задокументировать характеристики атрибутов качества (например, производительность или надежность), предоставляемых интерфейсом конечным пользователям. Эти сведения можно выразить в форме ограничений на реализацию составляющих интерфейс элементов. Выбор атрибутов качества, на которых предполагается сконцентрироваться и принять обязательства, проводится в зависимости от контекста.

Требования к элементам. Среди требований элемента может значиться наличие конкретных именованных ресурсов других элементов. Документация в данном случае составляется так же, как и в отношении ресурсов, – указываются синтаксис, семантика и ограничения по использованию. В некоторых случаях подобного рода информацию удобнее документировать в форме ряда допущений о системе, принятых проектировщиком элемента. В таком случае требования можно представить на суд экспертов, которые уже на ранних стадиях процесса проектирования смогут подтвердить или опровергнуть принятые допущения.

Логическое обоснование и соображения о проекте. Как и в случае с логическим обоснованием архитектуры в целом (или архитектурных представлений), архитектор должен документировать факторы, обусловившие принятие тех или иных проектных решений относительно интерфейса. В логическом обосновании необходимо указать мотивацию принятия этих решений, ограничения и компромиссы, обрисовать рассмотренные, но впоследствии забракованные решения (не забыв попутно изложить причины отказа от них), и изложить соображения архитектора касательно дальнейших изменений интерфейса.

Руководство по использованию. В некоторых случаях требуется анализ семантики с позиции связей между множеством отдельных взаимодействий. Если это так, в дело вступает протокол (protocol), документировать который следует согласно последовательности взаимодействий. Протоколы отражают комплексное поведение при взаимодействии тех образцов использования, которые, по мысли архитектора, должны встречаться с определенной регулярностью. Сложность взаимодействия с элементом через его интерфейс следует

компенсировать статической поведенческой моделью (например, схемой состояний) или примерами проведения отдельных видов взаимодействия (в форме диаграмм последовательностей).

4.4. Методы анализа архитектуры

4.4.1. Метод анализа компромиссных архитектурных решений – комплексный подход к оценке архитектуры

Метод анализа компромиссных архитектурных решений (Architecture Tradeoff Analysis Method, ATAM) – комплексная универсальная методика оценки программной архитектуры. В соответствии с названием этот метод обнаруживает степень реализации в архитектуре тех или иных задач по качеству, а также (исходя из допущения о том, что любое архитектурное решение влияет сразу на несколько задач по качеству) механизм их взаимодействия – другими словами, их взаимозаменяемость.

Основное назначение ATAM состоит в том, чтобы выявить коммерческие задачи, поставленные в контексте разработки системы и проектирования архитектуры. Вкупе с участием заинтересованных лиц это помогает специалистам по оценке сфокусироваться на тех элементах архитектуры, которые играют первостепенную роль для реализации упомянутых задач.

4.4.1.1. Этапы ATAM

Операции оценки по методу ATAM распадаются на четыре этапа.

На нулевом этапе – «Установление партнерских отношений и подготовка» – руководители группы оценки проводят неофициальные совещания с основными ответственными за проект лицами и прорабатывают подробности предстоящей работы. Представители проекта посвящают специалистов по оценке в суть проекта, тем самым повышая квалификацию некоторых из них. Эти две группы принимают согласованные логистические решения: где и когда встречаться, кто предоставит лекционные плакаты и т. д. Кроме того, они согласовывают предварительный перечень заинтересованных лиц (перечисляя их не по именам, а по ролям), устанавливают сроки и получателей сводного отчета. Они также организуют снабжение специалистов по оценке архитектурной документацией – если, конечно, таковая существует и может оказаться полезной. Наконец, руководитель группы оценки объясняет руководителю проекта и архитектору, какую информацию им следует предоставить на первом этапе, и при необходимости помогает им составить соответствующие презентации.

На первом и втором этапах проводится непосредственно оценка – все погружены в аналитические операции. К началу этих этапов участники группы оценки должны ознакомиться с документацией по архитектуре, получить достаточное представление о системе, знать задействованные архитектурные

методики и ориентироваться в первостепенных атрибутах качества. На первом этапе, намереваясь приступить к сбору и анализу информации, участники группы оценки встречаются с лицами, ответственными за проект (как правило, встреча длится весь день). На втором этапе к специалистам присоединяются заинтересованные в архитектуре лица, и в течение примерно двух дней они проводят аналитические мероприятия совместно.

Третий этап занимает доработка – группа оценки составляет в письменном виде и предоставляет получателям сводный отчет. По сути, участники занимаются самопроверкой и вносят в результаты своей работы разного рода коррективы. На заключительном совещании группы обсуждаются успехи и трудности. Участники изучают отчеты, выданные им на первом и втором этапах, и заслушивают выступление наблюдателя за процессом. По сути, они занимаются поиском путей усовершенствования по части исполнения своих ролей, с тем чтобы проводить последующие оценки с меньшими усилиями и с более высокой эффективностью. Действия, выполненные в период оценки участниками трех групп, тщательно регистрируются. По прошествии нескольких месяцев руководитель группы должен связаться с заказчиком оценки, для того чтобы оценить долгосрочные результаты ее проведения и сравнить издержки с выгодами.

Четыре этапа ATAM, их участники и приблизительный график представлены в табл. 4.2.

Таблица 4.2

Этапы ATAM и их характеристики

Этап	Операции	Участники	Средняя продолжительность
0	Установление партнерских отношений и подготовка	Руководство группы оценки и основные ответственные за проект лица	Проходит в неформальной обстановке, согласно конкретной ситуации; может длиться несколько недель
1	Оценка	Группа оценки и ответственные за проект лица	1 день с последующим перерывом продолжительностью от 2 до 3 недель
2	Оценка (продолжение)	Группа оценки, ответственные за проект лица и заинтересованные лица	2 дня
3	Доработка	Группа оценки и заказчик оценки	1 неделя

4.4.2. Метод анализа стоимости и эффективности – количественный подход к принятию архитектурно-проектных решений

Метод анализа стоимости и эффективности (Cost Benefit Analysis Method, CBAM) базируется на методе ATAM и обеспечивает моделирование затрат и выгод, связанных с принятием архитектурно-проектных решений, и способствует их оптимизации. Методом CBAM оцениваются технологические и экономические факторы, а также сами архитектурные решения.

4.4.2.1. Контекст принятия решений

Все программные архитекторы и ответственные лица стремятся довести до максимума разницу между выгодами, полученными от системы, и стоимостью реализации ее проектного решения. Являясь логическим продолжением метода ATAM, CBAM основывается на его артефактах. Контекст CBAM изображен на рис. 4.4.

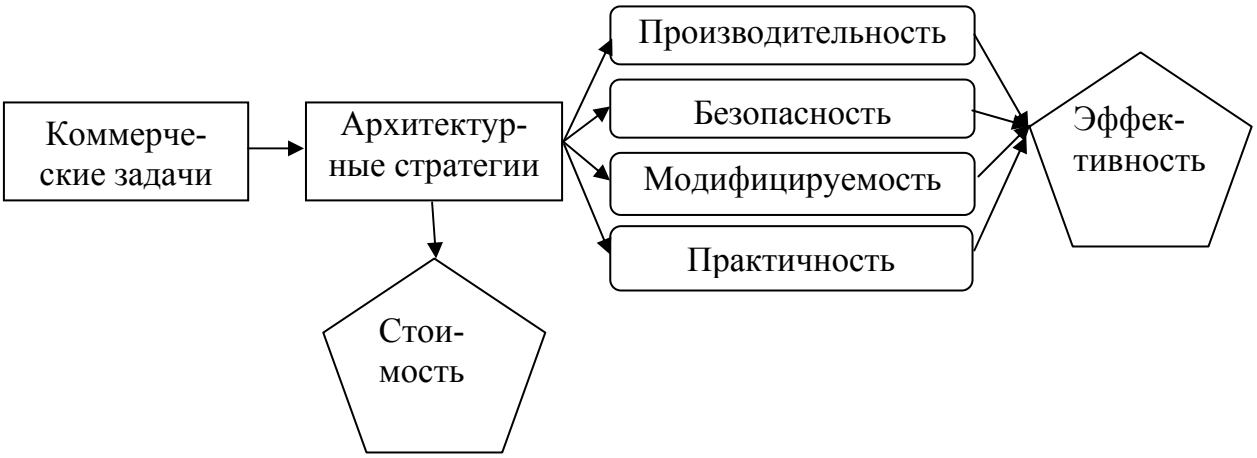


Рис. 4.4. Контекст метода анализа стоимости и эффективности (CBAM)

Поскольку архитектурные стратегии ограничиваются разнообразными техническими и экономическими факторами, стратегии, применяемые программными архитекторами и проектировщиками, должны быть поставлены в зависимость от коммерческих задач программной системы. Прямой экономический фактор – это стоимость реализации системы. Техническими факторами являются характеристики системы – другими словами, атрибуты качества. У атрибутов качества также есть экономический аспект – выгоды, получаемые от их реализации.

Как вы помните, по результатам оценки программной системы по методу ATAM в нашем распоряжении оказался ряд документированных артефактов.

Описание коммерческих задач, определяющих успешность системы.

Набор архитектурных представлений, документирующих существующую или предложенную архитектуру.

Дерево полезности, выражающее декомпозицию задач, которые заинтересованные лица ставят перед архитектурой, — от обобщенных формулировок атрибутов качества до конкретных сценариев.

- Ряд выявленных рисков.
- Ряд точек чувствительности (архитектурных решений, которые оказывают влияние на отдельный показатель атрибута качества).
- Ряд точек компромиссов (архитектурных решений, которые воздействуют сразу на несколько показателей атрибута качества, причем на одни положительно, а на другие отрицательно).

ATAM помогает выявить ряд основных архитектурных решений, значимых в контексте сформулированных заинтересованными лицами сценариев атрибутов качества. Эти решения приводят к реакции со стороны атрибутов качества, точнее говоря, отдельных уровней готовности, производительности, безопасности, практичности, модифицируемости и т. д. С другой стороны, каждое архитектурное решение связано с определенными издержками (стоимостью). К примеру, достижение желаемого уровня готовности путем резервирования аппаратные средств подразумевает один вид издержек, а регистрация в файлах на диске контрольных точек – другой. Эти архитектурные решения приводят к (предположительно разным) измеримым уровням готовности, имеющим определенную ценность для компании-разработчика системы. Возможно, ее руководство полагает, что заинтересованные лица заплатят большую сумму за систему с высокой готовностью (если, к примеру, это телефонный коммутатор или программное обеспечение для медицинского наблюдения), или боится погрязнуть в судебных разбирательствах в случае отказа системы (вполне разумно, если речь идет о программе управления антиблокировочной тормозной системой автомобиля).

ATAM обнаруживает архитектурные решения, принятые относительно рассматриваемой системы, и устанавливает их связь с коммерческими задачами и количественной мерой реакции атрибутов качества. Принимая эти данные на вооружение, CBAM помогает выявить связанные с такими решениями издержку и выгоды. Основываясь на этой информации, заинтересованные лица могут принять окончательные решения относительно резервирования аппаратной части введения контрольных точек и всех прочих тактик, направленных на повышение готовности системы. Вполне возможно, что они предпочтут сконцентрировать ресурсы, которые, как известно, ограниченны, на реализацию какого-то другого атрибута качества – например, на улучше-

ние соотношения выгод и издержек за счет повышения производительности. Из-за ограниченности бюджета разработки и обновления системы каждое архитектурное решение по большому счету соревнуется за право на существование со всеми остальными.

Подобно финансовому консультанту, который никогда напрямую не укажет, во что вкладывать деньги, CBAM не заменяет собой решений, принимаемых заинтересованными лицами. Он лишь помогает им установить и документировать издержки и выгоды архитектурных инвестиций, осознать неопределенность этого «портфеля». На этой основе заинтересованные лица могут принимать рациональные решения, удовлетворяющие их потребности и сводящие к минимуму риски.

Метод CBAM исходит из предположения о том, что архитектурные стратегии (как совокупность архитектурных тактик) оказывают влияние на атрибуты качества системы, а те, в свою очередь, предоставляют заинтересованным лицам некоторые выгоды. Эти выгоды мы называем полезностью (utility). Любая архитектурная стратегия отличается той или иной полезностью для заинтересованных лиц. С другой стороны, есть издержки (стоимость) и время, которые необходимо потратить на реализацию этой стратегии. Отталкиваясь от этой информации, метод CBAM помогает заинтересованным лицам в процессе выбора архитектурных стратегий, характеризующихся максимальной прибылью на инвестированный капитал (return on investment, ROI), – другими словами, наиболее выгодных с точки зрения соотношения выгод и издержек.

4.4.2.2. Реализация CBAM

На рис. 4.5 изображена диаграмма процессов, составляющих основу CBAM. Первые четыре этапа на ней сопровождаются комментариями с указанием относительного числа рассматриваемых сценариев. Постепенно их число уменьшается, таким образом, заинтересованные лица сосредоточиваются на тех сценариях, которые, по их мнению, в контексте ROI возымеют наибольшее значение.

Этап 1. Критический анализ сценариев. Отбор трети от общего количества сценариев путем расстановки приоритетов	N сценариев
Этап 2. Уточнение сценариев. Определение уровней реакции атрибутов качества для наилучшего, наихудшего, текущего и желаемого вариантов сценариев	N/3 сценариев
Этап 3. Упорядочивание сценариев согласно приоритетам. Исключение половины сценариев	N/3 сценариев
Этап 4. Определение полезности для текущего и желаемого уровней каждого из сценариев	N/6 сценариев
Этап 5. Разработка для сценариев архитектурных стратегий и определение уровней реакции атрибутов качества	
Этап 6. Определение ожидаемой полезности архитектурной стратегии путем интерполяции	
Этап 7. Определение общей выгоды, полученной от архитектурной стратегии	
Этап 8. Отбор архитектурных стратегий на основе ROI и с учетом ограничений по затратам	
Этап 9. Наглядное подтверждение полученных результатов	

Рис. 4.5. Диаграмма процессов CBAM

Этап 1. Критический анализ сценариев. Критический анализ сценариев проводится в рамках ATAM; на этом же этапе заинтересованные лица могут формулировать новые сценарии. Приоритеты расставляются в соответствии с потенциалом сценариев в контексте выполнения коммерческих задач системы; по результатам этапа для дальнейшего рассмотрения отбирается треть от общего первоначального числа сценариев.

Этап 2. Уточнение сценариев. Уточнению подвергаются сценарии, отобранные по результатам первого этапа; основное внимание при этом уделяется их значениям стимула-реакции. Для каждого сценария устанавливаются наихудший, текущий, желаемый и наилучший уровни реакции атрибута качества.

Этап 3. Расстановка сценариев согласно приоритетам. Каждому заинтересованному лицу выделяется 100 голосов, которые он берется распределить между сценариями, исходя из их желаемых значений реакции. После подсчета голосов для дальнейшего анализа остается только половина сценариев. Сценарию с наивысшим рангом присваивается вес 1.0, и, отталкиваясь от него, значения веса устанавливаются для всех остальных сценариев.

Именно эти значения впоследствии задействуются при вычислении общей выгоды стратегии. Помимо прочего, на рассматриваемом этапе составляется список атрибутов качества, которые заинтересованные лица считают значимыми.

Этап 4. Установление полезности. Для сценариев, оставшихся после проведения этапа 3, определяются значения полезности всех уровней реакции (наихудшего, текущего, желаемого, наилучшего) атрибута качества.

Этап 5. Разработка для сценариев архитектурных стратегий и установление их желаемых уровней реакции атрибута качества. Разработка (или фиксация разработанных) архитектурных стратегий, ориентированных на реализацию выбранных сценариев, и определение ожидаемых уровней реакции атрибута качества. Учитывая то обстоятельство, что одна архитектурная стратегия иногда оказывает воздействие на несколько сценариев, расчеты необходимо провести для каждого из затронутых сценариев.

Этап 6. Определение полезности ожидаемых реактивных уровней атрибута качества путем интерполяции. Исходя из установленных значений полезности (отраженных на кривой полезности) для рассматриваемой архитектурной стратегии определяется полезность желаемого уровня реакции атрибута качества. Эта операция проводится для каждого из перечисленных на этапе 3 значимых атрибутов качества.

Этап 7. Расчет общей выгоды, полученной от архитектурной стратегии. Значение полезности «текущего» уровня вычитается из желаемого уровня и нормализуется исходя из поданных на третьем этапе голосов. Суммируются выгоды, полученные от конкретной архитектурной стратегии, по всем сценариям и для всех значимых атрибутов качества.

Этап 8. Отбор архитектурных стратегий с учетом ROI, а также ограничений по стоимости и времени. Для каждой архитектурной стратегии определяются стоимостные и временные факторы. Значение ROI для стратегий определяется как отношение выгоды к издержкам. Архитектурные стратегии упорядочиваются по рангу согласно значениям ROI; впоследствии бюджет в первую очередь расходуется на высшие по рангу стратегии.

Этап 9. Интуитивное подтверждение результатов. Проверяется соответствие выбранных архитектурных стратегий коммерческим задачам компании. Если наблюдаются противоречия, ищем недосмотры во время проведения анализа. В случае если противоречия значительны, перечисленные этапы проводятся повторно.

5. ТЕХНОЛОГИЯ MDA

5.1. Использование архитектуры, управляемой моделью

5.1.1. Концепция архитектуры, управляемой моделью

Технология Model Driven Architecture (MDA) – архитектура, управляемая моделью, была разработана независимой некоммерческой организацией Object Management Group (OMG) – консорциум объектного управления. Она объединяет сотни компаний-производителей программного и аппаратного обеспечения.

В технологии MDA основным элементом проектирования считается модель.

Модель – это описание реальной системы, учитывающее определенные характеристики или аспекты моделируемого объекта, процесса или явления.

В конкретном проекте модель MDA представляет собой законченное и формализованное описание создаваемого программного продукта или его смысловой части. Высокая точность и непротиворечивость описания необходима для автоматизации процесса перевода модели в программный (обычно исходный) код.

Понятие «управление моделью» подразумевает прямое использование модели при любых действиях по проектированию, разработке и развертыванию системы. При внесении изменений в архитектуру приложения модель считается первичной. Пусть, например, требуется пополнить описание класса новым нулем или методом. В классическом (немодельном) подходе к разработке модификация описания класса выполнялась бы в исходном коде, ручным кодированием. В технологии MDA происходит модификация визуальной диаграммы, на которой класс представлен в виде графического элемента. На базе такой диаграммы исходный код с измененным описанием класса генерируется автоматически.

Под архитектурой, управляемой моделью, понимается полная и законченная архитектура приложения, целиком формируемая с помощью модели.

В жизненный цикл программы входит этап формализации и анализа требований заказчика. Идея ведения этого этапа с помощью визуальных моделей развивается уже много лет. Она реализована в виде визуальных высокоуровневых средств, понятных людям, слабо знакомым с технологиями программирования. Такие средства задают целостную внутреннюю архитектуру сложной информационной системы. Лучшие подобные решения поддерживают прямую и двустороннюю связь модели и программного кода. Из модели можно автоматически получить исходный код, а из исходного кода – визуальную модель. В результате удается плавно состыковать этап выработки и согласования требований с этапом кодирования и формирования исполнимого приложения.

В крупных проектах с моделью обычно работает не программист, а системный аналитик. Он отвлекается от мелких деталей реализации и перестает мыслить в терминах отдельных операторов языка программирования. Хороший проектировщик способен охватить и продумать структуру больших функциональных логических блоков и основные взаимосвязи между ними. Далее подобная модель детализируется и транслируется в исходный код на выбранном языке.

Архитектура (приложения), управляемая моделью, позволяет быстро создавать объемные программные системы разной функциональной направленности. У заказчика появляется возможность инвестировать основные средства не в конкретные технические решения, а, прежде всего, в реализацию требуемой логики приложения на модельном уровне, которая потом может почти автоматически развертываться на разных платформах. При этом достигается высокий уровень сохранности инвестиций. Ведь систему, созданную с помощью MDA, при появлении новых технологий можно адаптировать к ним с минимальными усилиями модификации модели и на высоком абстрактном уровне.

Технология MDA ориентирована на создание приложений, которые независимы от платформы, операционной системы и языков программирования. Она позволяет строить масштабируемые приложения из компонентов, которые могут использоваться повторно и многократно. Сам процесс разработки выполняется, как явствует из названия, под управлением модели.

Модель приложения – это взаимосвязанный набор визуальных диаграмм, наглядно описывающих внутреннюю структуру системы и принципы ее функционирования. Модель приложения не привязана к конкретному языку или конкретной среде программирования.

Визуальные диаграммы чаще всего строятся с помощью унифицированного языка моделирования UML (Unified Modeling Language). Его стандарт разработан группой OMG для задач объектно-ориентированного проектирования.

5.1.2. Модельные точки зрения и модели MDA

Когда разработчик взаимодействует с создаваемой системой в рамках подхода MDA, ему доступны три модельные точки зрения на систему.

Точка зрения, независимая от вычислительных особенностей Computation Independent Viewpoint (CIV), нацелена на анализ системной среды и системных требований. Логика работы пока что не рассматривается, возможно, она еще не определена. В CIV учитываются программные, аппаратные и другие технологические ограничения, которые не связаны напрямую с внутренней архитектурой разрабатываемой программы.

Точка зрения, независимая от программно-аппаратной платформы Platform Independent Viewpoint (PIV), задает конкретные функциональные

элементы системы, не зависящие от платформы. В ее рамках используются языки моделирования наподобие UML.

Точка зрения, зависящая от платформы Platform Specific Viewpoint (PSV), задает детали реализации, зависящие от используемых платформ и языков программирования. Она сочетает платформенно-независимую логику с дополнительными особенностями, вызванными использованием конкретной платформы или системы.

Каждая из этих точек зрения предлагает разные средства построения соответствующих моделей. Бизнес-аналитики, незнакомые с особенностями программной разработки, вырабатывают свои требования к системе на уровне CIV – с помощью так называемых вычислительно-независимых моделей (Computation Independent Model, CIM). Модели CIM задают основные требования к проекту и включают словари предметной области. Никакие технические характеристики в этих моделях не фиксируются.

Методология MDA описывает не столько моделирование, сколько метамоделирование, предусматривающее большую гибкость в конкретных, прикладных подходах к моделированию.

Метамоделирование – способ описания моделей, определяющий механизмы построения конкретных моделей программных систем с помощью базового словаря и набора ограничений, налагаемых на создаваемые модели. Сегодня вместо термина BMDA корпорация Borland применяет новый термин ECO. Технология ECO (Enterprise Core Objects) – ключевые корпоративные объекты, – реализующая концепцию MDA, стала наиболее важным улучшением последних версий среды Delphi. Она представлена в Delphi 2006 в виде третьей версии ECO III. Каждый компонент ECO представляет собой своеобразную программную «обертку» положений концепции MDA. Он является промежуточным слоем между средствами визуального проектирования программных моделей и их конкретной реализацией на языках программирования Delphi и C+.

Практика перевода требований заказчика напрямую в программный код на языке программирования почти всегда страдает неполноценностью. Заказчик мыслит одними понятиями, программист – совсем другими. Поэтому они не подозревают о множестве потенциально возможных проблем. Так, заказчику какие-то моменты автоматизации выбранных процессов могут казаться очевидными. Он о них и не упоминает, считая, что программа выполнит определенный набор действий сама. Программист, наоборот, подходит к созданию системы с узких технических позиций, программируя только очевидный набор функций, заданный в техническом задании, и редко учитывает все множество взаимосвязей между большим числом требований. В результате в середине проекта выясняется, что архитектуру решения невозможно пополнить новыми требованиями, которые заказчику кажутся простыми, естественными и подразумевавшимися с самого начала. В итоге приходится заново переделывать почти весь проект.

Технология ECO переводит процесс согласования требований на модельный уровень. Диаграммы UML обычно хорошо воспринимаются и заказчиком, и исполнителем. Поэтому, хотя использование визуального языка моделирования в MDA не обязательно, почти 100 % проектов MDA создаются с применением языка UML.

Построение готового приложения происходит в несколько этапов. Сначала строятся модели PIM, затем выполняется их перенос в модели PSM. Доступные па сегодняшний день продукты автоматизируют эти шаги на 50–70 %. Перенос моделей PSM в код конкретного языка программирования автоматизирован почти на 100 %.

Еще один подход, настоятельно рекомендуемый группой OMG при использовании подхода MDA, заключается в выделении логики приложения в отдельную, по возможности платформно-независимую структуру. Достигается это использованием языка объектных ограничений OCL, который сегодня считается частью языка UML. Команды OCL встраиваются непосредственно в модель UML, описывая и уточняя аспекты ее работы. Удобство OCL еще и в том, что его выражения напоминают естественный язык (предложения, записанные на английском). Это позволяет сохранять при построении моделей контакт с людьми (например, представителями заказчика), не являющимися специалистами в области программирования.

При реализации сложной логики далеко не всегда удается обойтись стандартными возможностями языков OCL и UML. Всевозможные манипуляции с наборами объектов, их фильтрация и сортировка решают до 70–90 % потребностей современных проектов. Такие задачи характерны для многих задач автоматизации деятельности организаций. Технология ECO позволяет быстро создавать работающие прототипы, но 10–30 % требований все равно приходится программировать вручную. Для этого используется исходный код на языке Delphi, в который отображается модель UML, сформированная в среде Delphi. Можно выполнить и обратное преобразование: на основе измененного исходного текста получить готовую модель UML.

Физические объекты, структура которых описана с помощью языка UML, а особенности существования – с помощью языка OCL, функционируют в процессе работы приложения в специально выделенной области оперативной памяти, так называемом объектном пространстве. Технология ECO способна автоматически отображать модель не только в программный код, но и в код на языке SQL. Он генерируется для построения схемы базы данных, хранящей копию объектного пространства. Поддержка разных СУБД является в ECO одной из ключевых технологий. С ее помощью удается сохранять текущее состояние объектного пространства приложения в базе данных и затем, после перезапуска приложения, загружать его и продолжать работу с прерванного места.

Среда Delphi на основе подготовленной модели автоматически формирует схему базы данных с учетом версии и специфики конкретной СУБД.

Она автоматически создает в ней все таблицы, нужные для хранения объектного пространства и взаимосвязей между классами, даже если это требует введения дополнительных таблиц. По мере совершенствования модели следует периодически синхронизировать схему базы данных с внесенными в модель модификациями – для этого достаточно нажать одну кнопку. Такой процесс называется в технологии ECO эволюционным развитием базы данных.

Стыковка объектного пространства с пользовательским интерфейсом реализуется в ECO с помощью дескрипторов. Дескриптор ECO – это стыковочная точка (компонент промежуточного уровня), связывающая с помощью выражений OCL элементы модели UML (классы) с программным кодом и внутренней структурой обычного приложения Delphi. Дескрипторы задают способы формирования наборов объектов в объектном пространстве по критериям, заданным выражениями OCL. Они также обеспечивают доступ к ним элементов пользовательского интерфейса, а программному коду предлагают ссылки на объекты ECO.

5.2. Язык объектных ограничений OCL

Один из самых серьезных и справедливо критикуемых недостатков языка моделирования UML – предоставление разработчику только визуальных средств моделирования. Эти средства абстрактны и поэтому далеко не всегда способны точно и формально отразить тот или иной нюанс функционирования проектируемой системы. В результате проектировщик не находит достаточно выразительных средств, позволяющих уточнить ограничения на применение создаваемых структур, специфицировать способы и нюансы их внутреннего функционирования, ввести условия использования и так далее. Конечно, можно представить эти особенности программным кодом конкретной среды программирования. Однако язык UML нацелен, прежде всего, на создание платформно-независимых моделей. Поэтому применяемый для этого текстовый язык, связанный с моделями UML, как минимум, тоже должен быть независим от операционной системы и среды разработки.

В начале 1990-х гг. в корпорации IBM разрабатывалась методика объектно-ориентированного анализа и проектирования Syntropy. Она включала математический язык текстовых описаний элементов визуальных моделей, который оказался сложен для массового практического использования.

Впоследствии на его основе был создан язык объектных ограничений OCL (Object Constraint Language), который применялся тогда как язык моделирования. Сильной стороной языка OCL оказалась независимость от платформы реализации и легкая адаптация к разным средам программирования. Он активно применялся для описания особенностей объектных информационных систем и в 1997 г. вошел в стандарт языка UML 1.1.

До появления языка OCL в системах визуального моделирования при уточнении ограничений на диаграммах UML применялись комментарии или

рекомендации на естественном языке. Они нередко трактовались неоднозначно.

Язык OCL снял множество проблем при проектировании моделей UML. Он предоставил разработчику набор формальных средств взаимодействия с объектами создаваемого приложения. Со временем язык OCL перерос границы стандарта UML. Он нередко задействуется как универсальный и, главное, платформно-независимый язык описания бизнес-логики. Например, сценарии OCL можно выделять и настраивать в проекте независимо от программного кода на конкретном языке.

Язык OCL не является языком программирования. Он предназначен для формального определения выражений, обрабатывающих объекты. Выражение OCL обычно привязано к определенному классу UML и задает множество экземпляров этого класса. Команды OCL выполняют также фильтрацию этого множества или, например, определяют число его элементов.

Язык OCL содержит развитые средства манипуляции над множествами объектов, поэтому он хорошо подходит для решения задач, где обычно применяется язык запросов к реляционным базам данных SQL. Язык OCL позволяет организовывать запросы с большей эффективностью и на более высоком модельном уровне абстракции, нежели язык SQL.

Выражение OCL фактически задает условие, которому должны удовлетворять все экземпляры соответствующего объекта UML. Результатом выполнения выражения OCL является множество объектов, удовлетворяющих этому условию. Условие, которое задается выражением OCL, называется инвариант. Оно представляет собой набор правил или шаблон, накладываемые на описание объекта. Значением инварианта является логическая величина (True или False). Выражения OCL встраиваются в модели UML или задаются в сопроводительной документации.

Язык OCL надежен и безопасен. Стандартом гарантируется, что любое его выражение будет вычислено без побочных эффектов и не окажет влияния на части модели (ни на какие ее состояния, значения или связи). Среда, вычисляющая выражение OCL, просто определяет результирующее значение, которое может впоследствии использоваться (хотя это и не обязательно). Сам язык OCL может применяться для реализации бизнес-логики, управления процессами и других задач только в качестве средства расчета выражений.

Привязка выражения OCL к объекту UML происходит через так называемое объявление контекста. В начале выражения задействуется наименование нужного класса объекта. Может также указываться ключевое слово self (ссылка на текущий объект). Оно допускается, если контекст вычисляемого выражения однозначен, очевиден и может быть связан с выражением автоматически.

5.2.1. Типы данных и операции OCL

В языке OCL используется четыре основных типа данных – значения могут быть целыми, вещественными (с плавающей запятой), логическими и строковыми. Над числами определены стандартные арифметические операции: сложение (знак +), вычитание (–), умножение (*), деление (/). Над значениями всех типов допускаются операции сравнения: меньше (<), больше (>), меньше или равно (<=), больше или равно (>=), не равно (<>), равно (=). Значения логических типов можно обрабатывать с помощью логических операций or (ИЛИ), and (И), not (НЕ), xor (исключающее ИЛИ), а также операции implies. Операция implies представляет собой особую форму логической операции И, результат которой зависит от порядка операндов: X implies Y – это не всегда то же самое, что Y implies X.

5.2.2. Инфиксная форма записи выражений OCL

В язык OCL входит ряд операций, которые являются своеобразными аналогами стандартных функций языка Delphi. Только записываются они не как функции Delphi (имя идентификатора и параметры в круглых скобках), а в так называемой инфиксной записи, которая напоминает форму записи обычных арифметических выражений.

Инфиксная запись выражения подразумевает размещение операндов по обе стороны знака операции и вычисление выражения слева направо без учета приоритетов операций.

Типичное выражение OCL представляет собой цепочку элементов, ссылок и вызовов операций, которые последовательно начиная с левого края выражения и до его правого края обрабатывают значение, полученное к моменту их вызова. Функциональные элементы и операции языка OCL применяются к аргументу, расположенному левее. Они записываются справа от него через точку. Вторым аргументом операции или вторым параметром функции OCL считается элемент, расположенный правее, его заключают в круглые скобки.

Так, для действительных чисел можно использовать операцию abs для определения модуля числа или операцию round для округления. Однако эта операция записывается не слева, как принято в языках программирования, а справа от обрабатываемого значения и через точку. Само значение, соответственно, указывается слева от имени функции и считается ее первым аргументом. Так как второй параметр для данной операции не нужен, результат ее работы можно передать далее, поставив еще одну точку, вслед за которой можно записать очередную функцию.

Пусть, например, требуется округлить число 123,45 с помощью функции round. Соответствующее выражение OCL записывается следующим образом:

```
123.45.round
```

Получить модуль отрицательного числа –123 с помощью функции abs можно так:

–123.abs

Если операция OCL использует два аргумента, второй из них заключается в круглые скобки, как обычный параметр. Например, операция max по выбору максимального из значений x и y запишется следующим образом:

x.max(y)

Для логической обработки язык OCL предлагает несложную форму условного оператора. Такой оператор фактически представляет собой вычисляемое выражение и записывается следующим образом:

if условие then выражение 1 else выражение 2 endif

Если условие истинно, то вычисляется выражение l, и результатом условного оператора считается его значение. В противном случае вычисляется выражение 2 и результатом считается уже его значение.

5.2.3. Последовательности доступа к объектам в языке OCL

Выражение OCL возвращает результат, который может быть: экземпляром одного из базовых типов; экземпляром одного из классов модели; метаинформацией (сведениями об определенном типе данных); набором или коллекцией экземпляров одного класса. Элементы такой коллекции могут быть упорядочены или неупорядочены. Внутри коллекции допускаются одинаковые экземпляры. Если к обычным, скалярным, значениям операции OCL применяются через точку (символ .), то к коллекциям – через комбинацию символов ->.

Для доступа к полям определенного объекта в языке OCL применяется запись, в которой сначала следует имя объекта, затем точка и имя поля объекта. Например, если имеется объект Computer (экземпляр класса Computer), а в этом классе описано поле Mouse, то обращение к полю Mouse записывается следующим образом:

Computer.Mouse

Если у поля Mouse (экземпляра класса Mouse), в свою очередь, имеются внутренние поля, то и к ним можно обращаться, продолжая запись через точку:

Computer.Mouse.Position

Поля объектов в выражении OCL могут быть не скалярными значениями, а коллекциями. Для того чтобы получить (выбрать) все существующие в программе на данный момент экземпляры определенного типа, используется стандартная операция allInstances. Например, все созданные в приложении экземпляры класса Computer можно получить следующим выражением:

Computer.allInstances

Результатом такого выражения будет не один объект, а коллекция, набор экземпляров класса Computer. Соответственно, запись

Computer.allInstances.Mouse

формирует коллекцию значений – экземпляров класса Mouse, входящих, в свою очередь, в состав класса Computer (физически – в текущее множество экземпляров этого класса).

5.2.4. Операции над коллекциями

Рассмотрим основные операции языка OCL, которые можно применять к коллекциям. Такие операции иногда называют итераторами. Они чаще всего задействуются в моделях ECO для описания логики работы программ и формирования наборов записей, отображаемых на экране. Эти операции записываются с использованием операции ->.

5.2.4.1. Стандартные операции

Для выполнения операций над коллекциями язык OCL предлагает ряд стандартных функций:

- функция Size возвращает число элементов коллекции;
- функции MinValue, MaxValue возвращают, соответственно, минимальное или максимальное значение из набора;
- функция Sum подсчитывает сумму всех элементов набора;
- функция Count определяет число элементов коллекции, удовлетворяющих условию, заданному в качестве параметра. Например, выражение Count (Name='Сергей') вычислит число элементов коллекции, у которых в поле Name содержится строка 'Сергей';
- функции isEmpty, NotEmpty возвращают значение True, если, соответственно, в анализируемом наборе нет ни одного элемента или есть хотя бы один элемент.

Функция, применяемая к коллекции, может возвращать скалярный результат (единственное значение) или вектор (набор значений). Результаты работы функций MinValue, MaxValue, Count и некоторых других относятся к последнему типу. Результатом может быть коллекция из нескольких одинаковых значений, каждое из которых соответствует объекту исходной коллекции, отвечающему заданному условию. Если, например, в исходном наборе несколько минимальных величин с одинаковыми значениями, все они попадут в результирующую коллекцию. Чтобы гарантированно получить скалярный результат, надо применить к итоговому набору операцию First (см. 5.2.4).

5.2.4.2. Операция Select

Операция Select позволяет выбрать в коллекции подмножество объектов, свойства которых отвечают заданному условию. Это условие указывается после ключевого слова Select в круглых скобках.

Пусть, например, в классе Computer (компьютер) имеется свойство Memory (объем оперативной памяти). Отбор всех объектов класса Computer,

у которых объем оперативной памяти больше или равен 512 мегабайт (512 единицам, принятым в модели по умолчанию), выполняет следующее выражение OCL:

Computer.allInstances->Select(Memory >= 512)

В дальнейшем по отношению к результирующей коллекции (в инфиксном порядке) можно применять другие допустимые операции OCL, в том числе и саму операцию Select. Например, возможно такое обращение к полям текущего подмножества объектов:

Computer.allInstances->Select(Memory >= 512).Mouse

Это выражение отбирает объекты, представляющие компьютерные мыши (свойство Mouse), которыми оснащены компьютеры с оперативной памятью не менее 512 мегабайт.

5.2.4.3. Операция Reject

Операция Reject отбирает в выходную коллекцию только те объекты, которые не удовлетворяют заданному условию. По смыслу она противоположна операции Select. В примере

Computer.allInstances->Reject(Memory < 512)

будет получен тот же результат, что и при использовании выражения:

Computer.allInstances->Select(Memory >= 512).

5.2.4.4. Выделение элементов коллекции

Для получения первого элемента в коллекции применяется операция First. Для получения последнего элемента в коллекции применяется операция Last. Выделение объекта по его номеру в коллекции выполняет операция At – номер нужного объекта задается в качестве параметра (нумерация начинается с единицы).

Например, следующее выражение получает последний элемент из коллекции компьютеров:

Computer.allInstances->last

Получить название (свойство Name) процессора (свойство Processor) первого элемента текущей коллекции компьютеров можно следующим образом:

Computer.allInstances->first.Processor.Name

Получить объект, представляющий мышь пятого компьютера текущего набора, можно следующим образом:

Computer.allInstances->At(5).Mouse

5.2.4.5. Упорядочение набора

Применение операций выделения первого и последнего элементов не всегда имеет смысл, если исходная коллекция не упорядочена. Упорядочение наборов данных производится операциями OrderBy и OrderDescending. Они

выполняют сортировку по заданному параметру, причем операция OrderDescending выполняет сортировку «в порядке убывания» – способом, противоположным OrderBy. Конкретный способ сортировки коллекций с помощью операции OrderBy определяется реализацией.

Например, отсортируем набор объектов-компьютеров по именам:

Computer.allInstances->OrderBy(Name)

5.2.4.6. Логические итераторы

Проверка выполнения логического условия для всех элементов коллекции осуществляется с помощью итератора ForAll. Пусть, например, в классе Computer имеется свойство Mouse (отдельный класс), а у него, в свою очередь, имеется логическое свойство isMiddleButton (наличие средней кнопки мыши). Тогда выражение:

Computer.allInstances->ForAll(Mouse.IsMiddleButton) вернет значение True, если все без исключения компьютеры коллекции оснащены трехкнопочными мышами.

Итератор Exists, схожий по смыслу с итератором ForAll, возвращает значение True, если хотя бы один из элементов коллекции соответствует заданному условию:

Computer.allInstances->Exists(Mouse.IsMiddleButton)

5.2.4.7. Операции для работы со строками

Для работы со строками в языке OCL применяются следующие операции.

Соединение двух строк в одну (символ +) может также задаваться ключевым словом concat. Строка-параметр присоединяется к исходному операнду справа. Например, следующее выражение к имени первого компьютера в коллекции прибавляет строку ': ноутбук':

Computer.allInstances->first.Name.Concat(': ноутбук')

Результатом выделения подстроки (ключевое слово Substring) являет ся строка, выделенная из исходного операнда. Дополнительные параметры задают номера первого и последнего символов, включаемых в подстроку (нумерация начинается с единицы). Например, выражение

'компьютер'.Substring(2,4)

Вернет подстроку «омп».

Функции ToLower, ToUpper преобразуют все символы строки-операнда, соответственно, к нижнему или верхнему регистру.

Функция strToint преобразует исходную строку в целочисленное значение.

5.2.4.8. Работа с датами

Даты представляются в языке OCL в формате #гггг-мм-дд, где гггг – четырехзначная запись года, мм – номер месяца, а дд – число в месяце. Схожим образом записываются и временные константы – в формате #чч:мм:сс. Здесь чч означает часы, мм – минуты, сс – секунды. Константы, записанные в таком виде, можно использовать в операциях сравнения. Например:

```
Computer.allInstances->Select(MyDate > #2006.01.01)
Users.allInstances->Select(ConnectTime = #23:59:00)
```

Чтобы проверить, лежит ли дата в указанном диапазоне, следует воспользоваться функцией inDateRange. Двумя ее параметрами выступают нижняя и верхняя границы дат.

5.3. Возможности технологии ECO

5.3.1. Введение в технологию ECO

Технология ECO – это реализация концепции архитектуры, управляемой моделью. Она позволяет полностью создать приложение с помощью языка UML.

Технология ECO включает в себя набор объектов моделирования .NET и работает только на платформе .NET. В ней используются диаграммы UML как на фазах начального проектирования приложения, так и на этапе эксплуатации программы. Это достигается за счет встраивания модели в приложение.

Почти весь процесс создания готового приложения в рамках проекта ECO осуществляется манипулированием визуальной моделью, представленной в виде диаграмм UML. При этом удается избежать ручной модификации исходного кода и минимизировать обращения к проектировщику пользовательского интерфейса – весь исходный код приложения автоматически генерируется на основе визуальной модели. Ручная модификация кода нужна обычно для тонкой настройки приложения на дополнительные требования пользователя.

Центральной концепцией программы ECO считается объектное пространство. Всевозможные аспекты создания и поведения объектов в этом пространстве задаются с помощью платформно-независимого языка объектных ограничений OCL.

Объектное (или модельное) пространство ECO – это область памяти, в которой существуют и взаимодействуют объекты ECO текущего проекта. В системе Delphi 2006 поддерживается третья версия технологии ECO III.

Она состыкована со средой моделирования Borland Together, выпускаемой как самостоятельный продукт. Технология ECO III позволяет проектировать не только статическую структуру приложения, иерархию классов, но и его поведение, программную логику. Для этого задействуются так называемые машины состояний (описывающие их диаграммы состояний появились в версии языка UML 2.0). В итоге почти вся разработка сложного приложения выполняется в дизайнере диаграмм UML, встроенном в систему Delphi. На базе созданных таким образом моделей автоматически генерируется полнофункциональный исходный код приложения. В случае ручной модификации исходных текстов структура модели также автоматически подстраивается под внесенные изменения. Такая двунаправленная технология синхронизации модели и кода получила в среде Delphi название LiveSource.

Технология ECO поддерживает не только этап построения приложения и модели ECO, но и этап выполнения программы. При запуске приложения ECO создается объектное пространство ECO, которое называется ECO Space. В этом пространстве хранится вся необходимая метаинформация о модели. В рамках объектного пространства происходит создание и уничтожение экземпляров элементов ECO. Содержимое пространства ECO можно автоматически связать с данными в файлах или базах данных.

Технология ECO хорошо подходит для построения масштабных корпоративных приложений. Она значительно сокращает время разработки приложений, в которых активно задействуются базы данных с множеством таблиц и сложными взаимосвязями и большое число экранных форм, создание которых вручную является трудоемкой рутинной задачей. В технологию ECO встроены средства поддержки реляционной модели баз данных. Они автоматически преобразуют модель UML в описание схемы базы данных, генерируют сценарии SQL, которые создают и модифицируют такие схемы, и выполняют другие необходимые операции.

5.3.2. Модель ECO

Модель может объединять несколько взглядов на объект, каждый из которых уточняет ту или иную особенность его существования. Модель компьютерной программы (в частности, модель ECO) отличается от физических и математических моделей тем, что на ее основе должен быть автоматически сгенерирован безошибочный и работоспособный исходный текст на заданном языке программирования. Это предъявляет повышенные требования к полноте модели ECO, ее точности и непротиворечивости.

Процесс подготовки модели удается упростить с помощью набора готовых объектов ECO. Они прозрачно обеспечивают взаимосвязь между программным кодом и элементами модели. В результате проектирование и развитие архитектуры всего приложения существенно упрощается.

5.3.3. Пространство имен ECO

Познакомимся с организацией пространства имен ECO. В рамках иерархии классов Delphi все классы ECO входят в единое пространство имен Borland.Eco. Оно, в свою очередь, делится на шесть больших категорий.

Категория Borland.Eco.ObjectRepresentation содержит средства стандартного представления объекта ECO во внешних программах. Каждый из прикладных объектов ECO состоит из элементов (внутренние поля, методы). К каждому из этих элементов можно обращаться через стандартный интерфейс Element. Любой элемент объекта, доступный через этот интерфейс, может быть представлен в виде стандартного объекта .NET. Для этого задействуется его свойство AsObject.

Категория Borland.Eco.Handles содержит компоненты поддержки модельного пространства на этапе проектирования. Они связывают объекты ECO, доступные через интерфейсы пространства имен ObjectRepresentation, с элементами пользовательского интерфейса.

Категория Borland.Eco.UmIRt содержит средства доступа к модели UML в процессе работы программы. Компоненты ECO базируются на классах .NET, поэтому к ним можно обращаться универсальным способом, принятым в .NET. В частности, для этого задействуется метод AslObject, предоставляющий для доступа к объекту интерфейс IObject платформы .NET.

Категория Borland.Eco.Subscription содержит средства уведомления об изменении значений объектов ECO по технологии «издатель – подписчики».

Категория Borland.Eco.Persistence содержит средства автоматического сохранения содержимого объектного пространства ECO в файлах и базах данных.

Категория Borland.Eco.Services – это среда поддержки внутренних механизмов технологии ECO. Она занимается вычислением значений выражений OCL, вносит модификации в модель, контролирует ее целостность.

5.4. Разработка приложений на основе ECO

5.4.1. Этапы создания приложения по технологии ECO

Технология создания приложения ECO состоит из следующих этапов.

1. Формируется модель UML будущего приложения. Создаются классы, описываются их атрибуты и методы, настраиваются взаимосвязи.

2. В проект добавляются и настраиваются невизуальные компоненты, связывающие созданную модель UML с прикладной частью проекта.

3. Проектируется пользовательский интерфейс. Задействуются компоненты, обеспечивающие связь интерфейса с моделью UML.

4. Создается переносимая логика приложения на языке OCL. Элементы управления связываются с выражениями OCL для выполнения типичных

стандартных действий (добавление, редактирование и удаление объектов ECO).

5. Пространство ECO связывается с базой данных. В ней будет долговременно храниться его копия: все объекты ECO и связи между ними. Эта последовательность в ходе расширения функциональных возможностей приложения многократно повторяется. Но все вносимые в проект изменения, что принципиально важно, выполняются, начиная с модели, а не с исходного кода или пользовательского интерфейса.

5.4.2. Создание простого MDA-приложения

Рассмотрим пример создания простого MDA-приложения. Имеются две сущности: *Кафедра* и *Преподаватель*. Представим экземпляры этих сущностей в таблицах формы. В этих же таблицах будет возможность модификации значений отдельных полей представленных в них объектов (экземпляров *Кафедра* и *Преподаватель*).

Создается пустая заготовка приложения ECO командой File>New>Other. Выберем значок ECO WinForms Application во вкладке Delphi for .NET Projects (рис. 5.1).

В диалоговом окне введем имя проекта (например, projDeanOffice) и его местоположение (каталог). Нажмем кнопку *OK*.

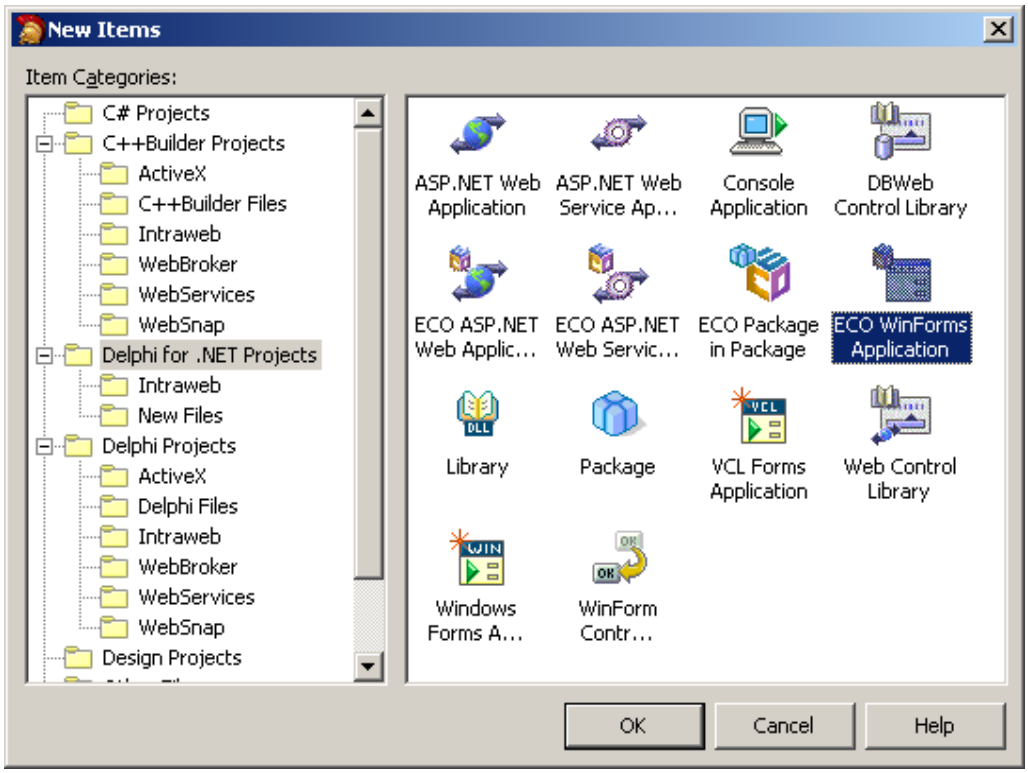


Рис. 5.1. Создание заготовки проекта ECO

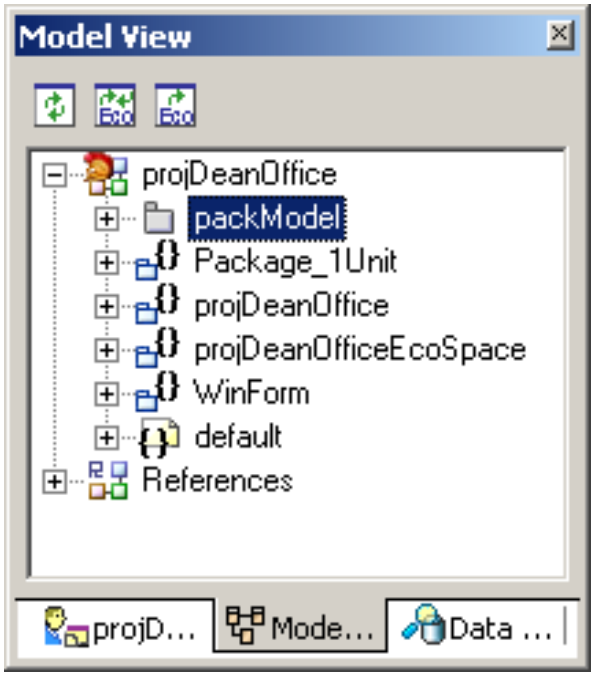


Рис. 5.2. Окно просмотра модели

Среда Delphi сформирует пустую заготовку приложения ECO и откроет окно *Проектировщика*. В нем расположена пустая начальная форма приложения. Структура автоматически созданной модели (пустой заготовки) доступна в окне просмотра модели, открываемом командой View>Model View либо выбором вкладки Model View в правом верхнем окне (рис. 5.2). В этом окне в дополнение к самому проекту (projDeanOffice) и главной форме (WinForm) можно увидеть еще несколько автоматически добавленных элементов. Среди них имеются следующие:

- элемент projDeanOfficeEcoSpace представляет объектное пространство ECO. Это основное хранилище, в котором располагаются экземпляры классов создаваемой модели во время работы программы. Это пространство также ответственно за хранение объектов ECO, например в базе данных или файле XML;
- элемент Package_1 представляет пакет классов UML, которые мы будем создавать.

Переименуем элемент Package_1 в удобное для нас название packModel.

5.4.2.1. Создание модели UML

Для создания модели дважды щелкнем мышью на строке packModel в окне просмотра модели. Откроется окно packModel [diagram] диаграммы классов ECO. Это окно напоминает окно построения обычных диаграмм классов UML. Только при работе с ним применяются элементы, специфичные именно для технологии ECO.

Разместим на диаграмме первый класс, отражающий сущность *Деканат*. Для этого выберем на палитре инструментов Tool Palette инструмент

ECO Class в категории UML ECO Class Diagram и щелкнем в подходящей точке пространства моделирования. В ней появится графический элемент, изображающий класс.

Назовем класс clChair (*Кафедра*). Пробелы в имени класса ECO не допускаются.

Добавим атрибуты класса командой контекстного меню Add>Attribute. Создадим таким образом три атрибута: *Название кафедры*, *Ф. И. О. заведующего кафедрой*, *Ф. И. О. секретаря кафедры* – ChairName, ChairHeadSNP и ChairSecrSNP соответственно (рис. 5.3). Для задания типа атрибута выделим нужный атрибут и в окне Properties установим необходимое значение поля Type в категории General. В данном случае все три атрибута имеют тип String.

Переименуем названия класса и атрибутов в понятные названия на русском языке. Для этого в свойстве Alias (категория General) класса и его атрибутов введем русскоязычные названия.

По аналогии добавим к модели еще один класс clLecturer (*Преподаватель*) с атрибутами LecturerSNP (*Ф. И. О. преподавателя*) и LectAcadDegree (*Ученая степень*) типа String. Переименуем элементы нового класса на диаграмме, изменив значения свойства Alias.

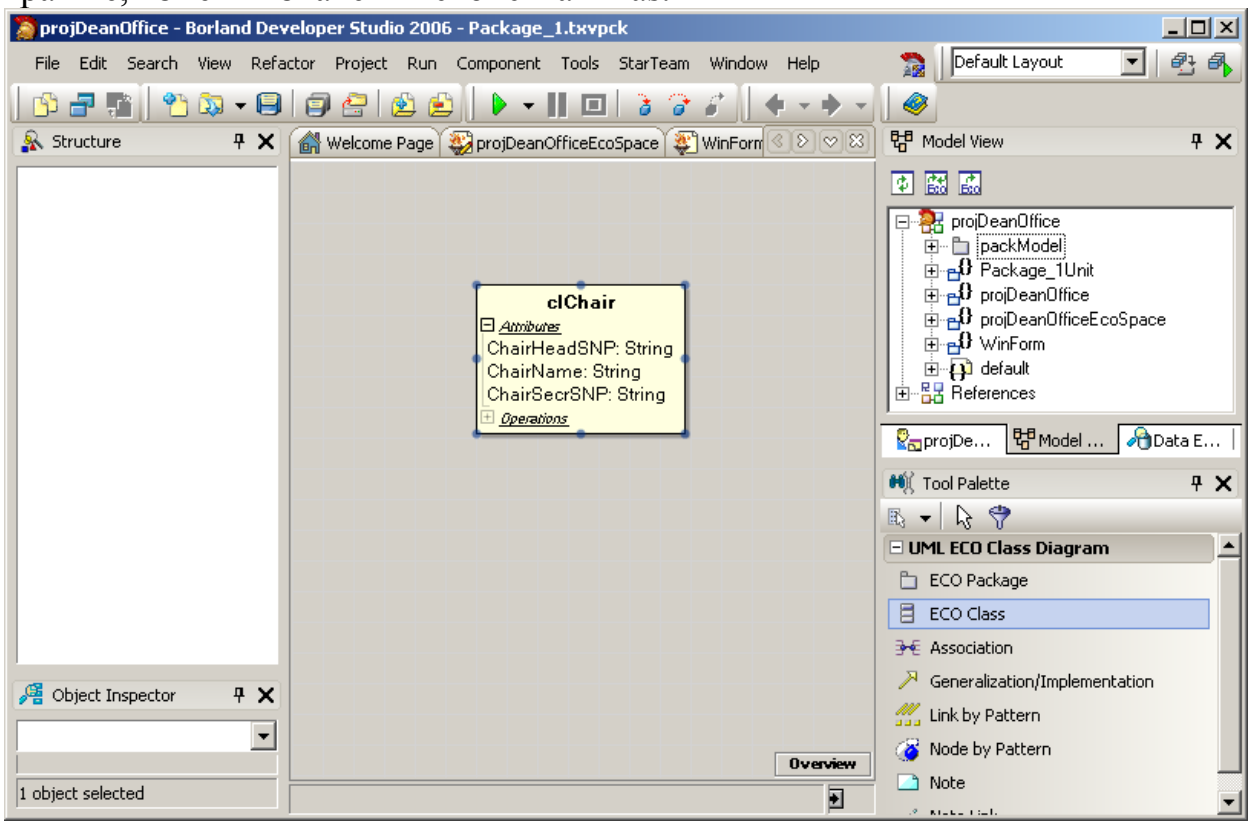


Рис. 5.3. Добавление класса clChair с его атрибутами

Настроим связи между созданными классами. Эта связь будет представлять отношение ассоциации. Выберем на палитре инструментов инструмент Generalization/Implementation. Щелкнем мышью на представлении класса *Ка-*

федра, протянем связь к классу *Преподаватель* и снова щелкнем мышью. В результате между двумя классами сформируется ассоциативное отношение.

Настроим мощность ассоциативного отношения. В нашем случае одному экземпляру класса *Кафедра* соответствует множество экземпляров класса *Преподаватель* (от 1 и более). Для этого в окне Properties выделенной связи выберем категорию End1, соответствующую классу *Кафедра*, и в поле Multiplicity установим значение 1, а в категории End2 (для класса *Преподаватель*) этому же полю – значение 1..*. Теперь дадим сторонам связи названия. В свойство Name для сторон End1 и End2 введем roleChair и roleLecturers соответственно (рис. 5.4). То есть мы назвали роли каждого класса в ассоциативной связи.

На этом этапе создание простейшей модели закончено. Теперь надо организовать связь модели с пользовательским интерфейсом.

5.4.2.2. Создание интерфейса

Перейдем к окну *Проектировщика* для главной формы проекта WinForm (вкладка Design). Переименуем стандартное название главной формы.

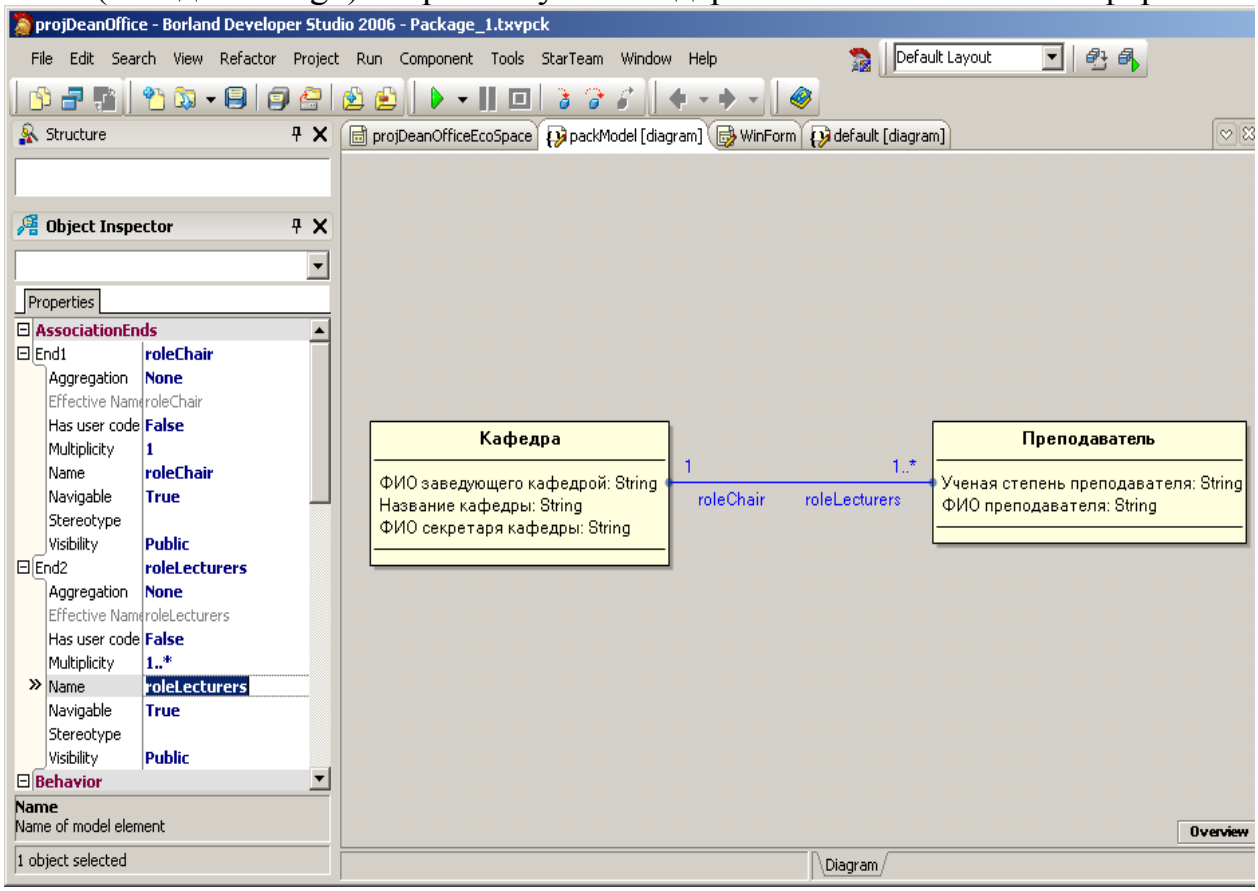


Рис. 5.4. Настройка связи между классами

Назовем ее wfMainForm, а сам класс TWinForm переименуем в TMainForm в свойстве Name категории Design. Свойство Text (категория Appearance) отве-

чает за название заголовка окна, введем в него строку, например *Главная форма*.

Для представления таблицы с объектами ECO на форме воспользуемся готовым компонентом DataGrid из категории палитры инструментов Data Controls. Поместим этот компонент на форму и дадим ему название dgChair (в свойстве Name). Эта таблица будет отвечать за представление экземпляров класса *Кафедра* (clChair). Исходно таблица должна быть пуста. В свойстве CaptionText (заголовок таблицы) введем название *Кафедры*.

Добавим еще одну таблицу dgLecturer, которая в готовом приложении будет отвечать за отображение экземпляров класса *Преподаватель* (clLecturer). В свойстве CaptionText введем название *Преподаватели*.

Для работы с таблицами в простом приложение потребуется пять операций: добавление и удаление данных в двух таблицах и сохранение копии ECO пространства из оперативной памяти в базу данных. Редактирование данных будет осуществляться непосредственно в полях таблиц. Добавим на форму пять кнопок – экземпляров класса Button (рис. 5.5).

5.4.2.3. Связывание интерфейса с моделью

Связь пользовательского интерфейса с моделями ECO обычно происходит через компонент ExpressionHandle. Он доступен в категории палитры инструментов Enterprise Core Objects. Через подобные идентификаторы (дескрипторы) организуется доступ к объектам модели и пространства ECO во время работы программы.

Идентификаторы ECO связываются друг с другом в цепочки. Во главе такой цепочки расположен корневой идентификатор. Он задает общий контекст работы (доступ к объектному пространству) для всех остальных идентификаторов ECO в программе.

Корневой идентификатор добавляется к проекту автоматически (по умолчанию он называется rhRoot). Все остальные идентификаторы ECO добавляются и настраиваются вручную, что связывает пространство ECO с элементами пользовательского интерфейса с помощью типовых механизмов связывания .NET.

Добавим к проекту компонент ExpressionHandle. Он отображается в нижней части окна Проектировщика, под формой, где уже имеется набор компонентов ECO. Назовем его ehChair, введем это имя в свойство Name категории Design.

В свойстве RootHandle этого дескриптора выберем значение rhRoot – ссылку на родительский, автоматически созданный корневой идентификатор. Так задают место данного идентификатора в цепочке доступа к объектному пространству ECO.

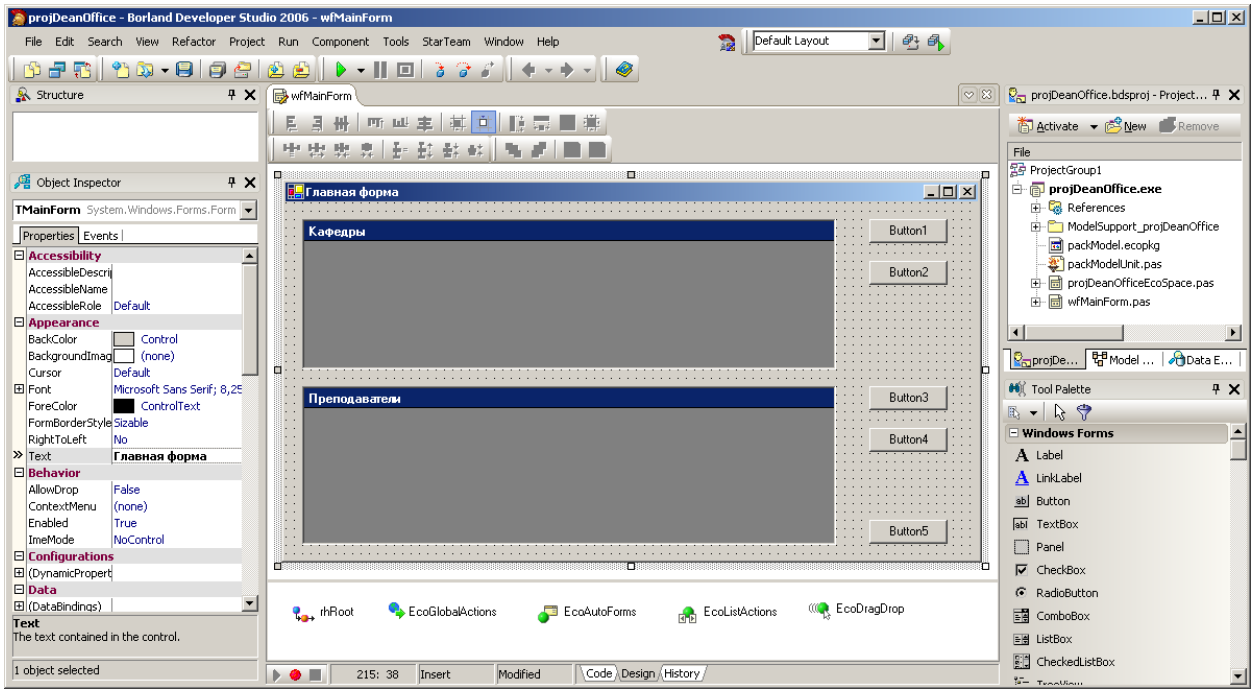


Рис. 5.5. Добавление компонентов пользовательского интерфейса

В свойстве DataSource таблицы dgChair выберем имя ehChair в списке доступных идентификаторов ECO.

Настроим таблицу *Преподаватель* так, чтобы она отражала список преподавателей, принадлежащих выбранной кафедре в первой таблице. Для начала добавим в проект дескриптор CurrencyManagerHandle (из категории Enterprise Core Objects) и дадим ему имя cmhChair, так как этот компонент будет указывать на текущую строку таблицы *Кафедра*. Свяжем дескриптор cmhChair с таблицей dgChair через свойство BindingContext. Теперь он отслеживает выделенную строку этой таблицы.

Зададим ссылку на объект ehChair в свойстве RootHanler, определяющим корневой идентификатор ECO.

Добавим к проекту еще один компонент ExpressionHandle. Назовем его ehLecturer. Дескриптор ehLecturer будет обращаться к экземплярам класса *Преподаватель*.

Зададим ссылку на объект cmhChair в свойстве RootHanler.

Потребителем объектов, предоставляемых дескриптором ehLecturer, будет вторая таблица. В ее свойстве DataSource выберем ссылку на объект ehLecturer.

Приступим к настройке элементов пользовательского интерфейса. Организуем с помощью визуальных средств Delphi создание новых экземпляров класса *Кафедра* и добавление их в таблицу. Выделим в окне *Проектировщика* кнопку Button1. В свойстве EcoListAction (Список стандартных действий ECO) выберем значение Add (*Добавить объект*).

Кнопка Button1 автоматически получила название Add. Переименуем кнопку. Для этого воспользуемся компонентом EcoListActions (он добавляется в проект по умолчанию и находится в нижней части окна *Проектировщика*), в его свойстве CaptionAdd введем *Добавить*.

Объект ECO, который мы хотим добавить к проекту (сделать доступным через элементы управления на форме), задается в свойстве RootHandle. В нем надо выбрать подходящий поставщик объектов ECO, в нашем случае – идентификатор ehChair.

Свяжем результат действия кнопки (созданный экземпляр класса *Кафедра*) с визуальным элементом, отображающим этот экземпляр, в нашем случае – с таблицей dgChair. Для этого в свойстве BindingContext (контекст связывания) выберем имя dgChair (рис. 5.6).

Теперь добавим операцию удаления экземпляров класса *Кафедра*. Реализуем это действие по аналогии с операцией добавления. Выделим в окне *Проектировщика* кнопку Button2. В свойстве EcoListAction выберем значение Delete. Зададим идентификатор ehChair в свойстве RootHandle. Свяжем результат действия кнопки с визуальным элементом – таблицей dgChair. Для этого в свойстве BindingContext у кнопки Delete выберем имя dgChair. Перейдем к компоненту EcoListActions и в его свойстве Caption Delete введем *Удалить*.

Настройка кнопки Button3 (будет отвечать за добавление экземпляра класса *Преподаватель*). Значения свойств кнопки: EcoListAction – Add; RootHandle – ehLecturer; BindingContext – dgLecturer. Перейдем к компоненту EcoListActions и в его свойстве Caption Add введем *Добавить*.

Настройка кнопки Button4 (будет отвечать за удаление экземпляра класса *Преподаватель*). Значения свойств кнопки: EcoListAction – Delete; RootHandle – ehLecturer; BindingContext – dgLecturer. Перейдем к компоненту EcoListActions и в его свойстве Caption Delete введем *Удалить*.

Кнопка Button5 будет отвечать за обновление БД. Назовем ее Сохранить. В свойстве EcoAction (категория ECO|GUI) выберем UpdateDatabase. Таким образом, по нажатию этой кнопки выполняется синхронизация содержимого пространства ECO в оперативной памяти и его копии в базе данных. После каждого нажатия кнопки *Сохранить* содержимое таблицы сохраняется в БД. После нового запуска программы в таблицу пользовательского интерфейса автоматически загружается содержимое модели, сохраненное при последнем выполнении команды UpdateDatabase.

На данном этапе связывание интерфейса с моделью закончено (рис. 5.7).

Действия, которые выполняет дескриптор в программе, определяются значением свойства Expression. В это свойство вводится выражение языка OCL, которое определяет схему создания объектов модели ECO.

Используем дескриптор ehChair для обращения к экземплярам класса *Кафедра*. Введем в свойство Expression объекта ehChair строку clChair.allInstances. Иначе это можно сделать с помощью редактора OCL выражений. Введенное выражение задает доступ ко всем текущим экземплярам класса *Кафедра* в объектном пространстве. В таблице, показанной на форме в окне Проектировщика, сразу же отобразятся поля класса *Кафедра* (ChairHeadSNP, ChairSecrSNP и ChairName). Дескриптор ehChair становится поставщиком данных нашей модели ECO для первой таблицы.

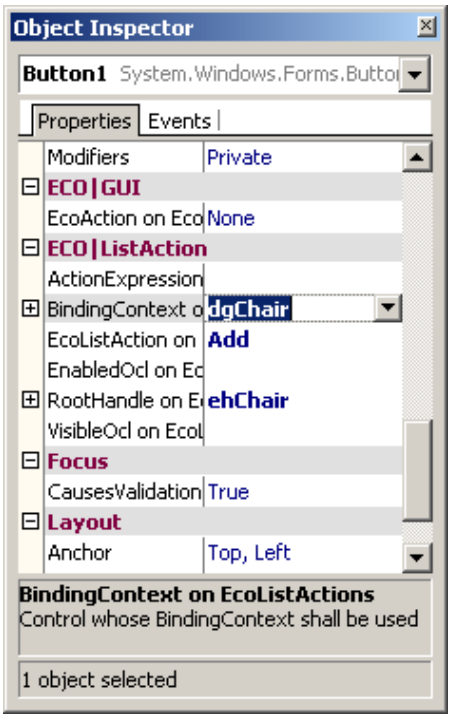


Рис. 5.6. Настройка кнопки добавления экземпляра класса Кафедра

5.4.2.4. Создание логики на OCL

В качестве выражения OCL объекта ehLecturer введем строку self.roleLecturers в свойство Expression. Здесь roleLecturers – это введенное нами при построении модели имя роли класса *Преподаватель* (clLecturer) в его ассоциативной связи с классом *Кафедра* (clChair). Это выражение определяет группу преподавателей, принадлежащих кафедре, которая выбрана в первой таблице. Дескриптор ehLecturer становится поставщиком данных для второй таблицы (рис. 5.8).

Дадим заголовкам полей таблиц русскоязычные названия. Выберем на форме таблицу dgChair. Щелчком на кнопке с многоточием в строке свойства TableStyles откроем окно редактора коллекции стилей таблицы DataGridTableStyle Collection Editor. Создадим новый стиль таблицы, нажав на кнопку Add (рис. 5.9).

Теперь настроим оставшиеся два элемента коллекции. Эти столбцы будут называться: *Ф. И. О. завкафедрой* и *Ф. И. О. секретаря кафедры*. Нажмем кнопку *ОК* в обоих открытых окнах и убедимся, что поля таблицы получили новые названия.

По аналогии настроим стиль таблицы *Преподаватели*. В ней отобразим два поля: LecturerSNP и LectAcadDegree. Назовем их *Ф. И. О. преподавателя* и Ученая степень.

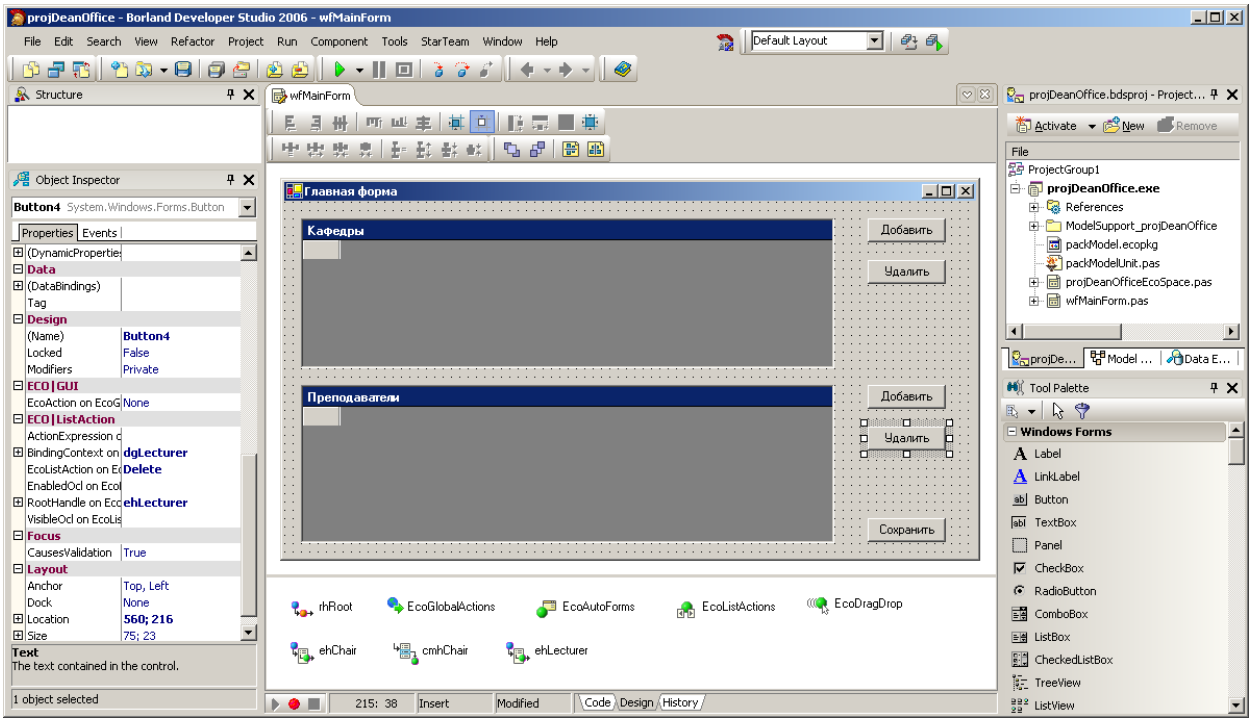


Рис. 5.7. Настроенный пользовательский интерфейс приложения

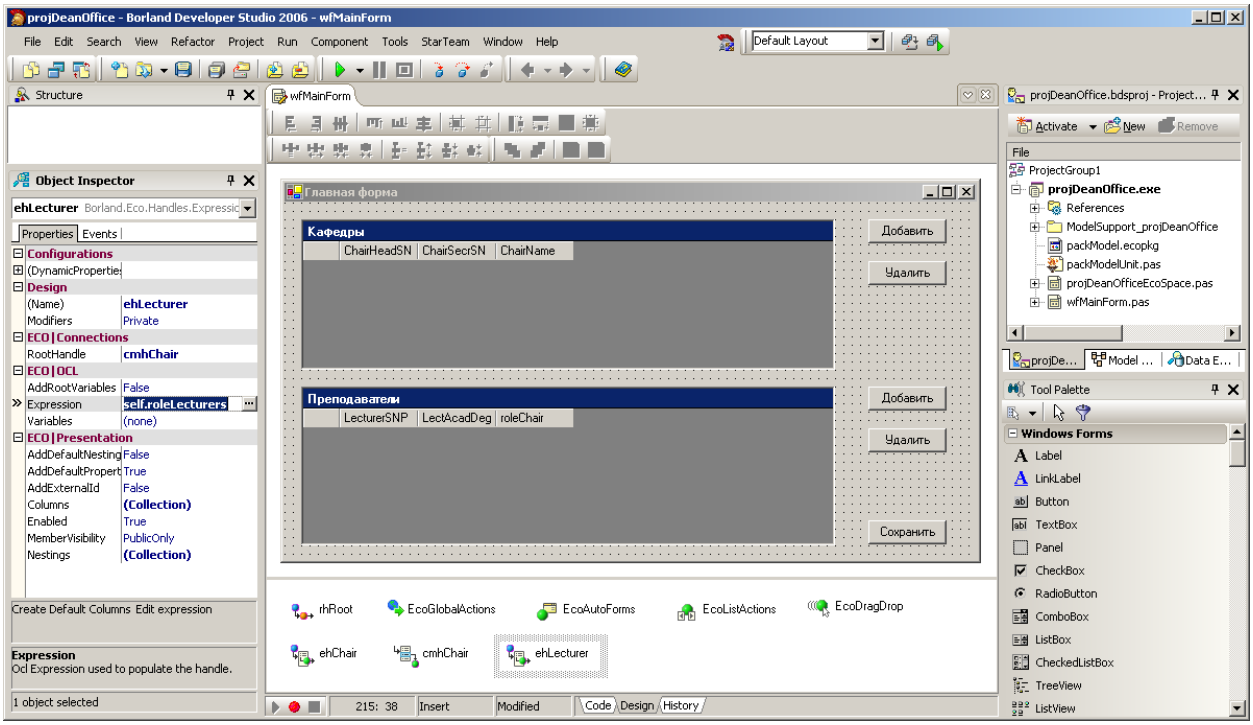


Рис. 5.8. Настройка OCL-выражений для взаимосвязанных таблиц

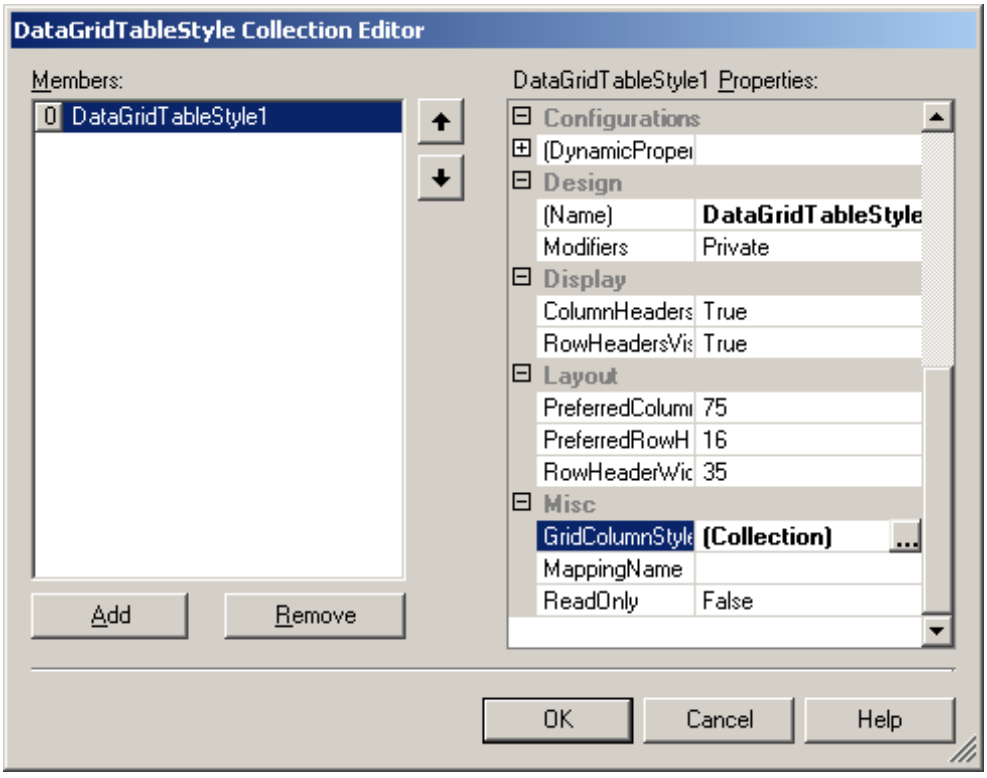


Рис. 5.9. Коллекция таблиц компонента gdChair

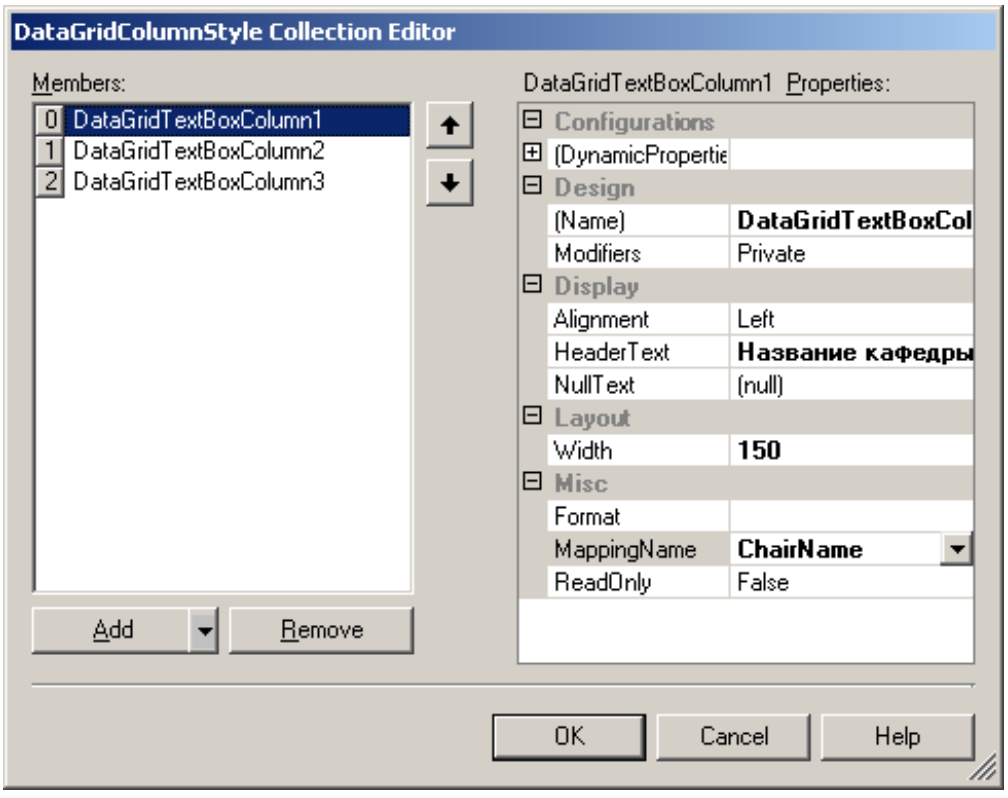


Рис. 5.10. Настройка элемента коллекции столбцов

Запустим программу и убедимся, что при нажатии кнопки *Добавить в таблице* автоматически формируются новые строки, отражающие содержимое новых экземпляров класса из объектного пространства ECO. В них можно ввести нужные значения.

6. ДОКУМЕНТИРОВАНИЕ ПРОГРАММНЫХ СИСТЕМ В СООТВЕТСТВИИ С ГОСТ

6.1. Управление документированием программного обеспечения

Управление документированием программного обеспечения должно выполняться в соответствии с ГОСТ Р ИСО/МЭК ТО 9294–93. Данный ГОСТ разработан техническим комитетом по стандартизации ТК 22 «Информационная технология». Утвержден и введен в действие постановлением Госстандарта России от 20.12.93 № 260. Стандарт подготовлен на основе применения аутентичного текста технических рекомендаций ИСО/МЭК ТО 9294–90 «Информационная технология. Руководство по управлению документированием программного обеспечения».

6.1.1. Область применения

Данный стандарт представляет собой руководство по документированию программного обеспечения для тех руководителей, которые отвечают за производство программного обеспечения или программной продукции. Руководство предназначено для помощи руководителям в обеспечении эффективного проведения документирования в их организациях.

Данный стандарт направлен на определение стратегий, стандартов, процедур, ресурсов и планов, которыми должны заниматься сами руководители для того, чтобы эффективно управлять документированием программного обеспечения.

Руководство предназначено для применения ко всем типам программного обеспечения – от простейших программ до наиболее сложного программного набора или системы программного обеспечения. Охвачены все типы программной документации, относящиеся ко всем стадиям жизненного цикла программного обеспечения.

Принципы управления документированием программного обеспечения одинаковы для любого объема проекта. Для небольших проектов значительную часть положений, приведенных в данном стандарте, можно не применять, но принципы остаются теми же. Руководители могут адаптировать данные рекомендации для своих конкретных потребностей.

Следует подчеркнуть, что руководство дано с точки зрения управления документированием. Подробные советы относительно состава и компоновки программных документов не приведены.

6.1.2. Роль руководителей

Руководители принимают на себя организацию работ по документированию и осуществляют поддержку этих работ в стратегиях, стандартах, процедурах, распределении ресурсов и планах, которыми они определяются.

Эффективность выполнения руководящей роли можно рассматривать как основанную на трех элементах:

1) руководящая обязанность по документированию. Данная обязанность требует признания того, что программная;

2) документация важна и что ее следует планировать, описывать, проверять, утверждать, выпускать, распространять и сопровождать;

3) руководящая поддержка обязанностей персонала по документированию.

Для этого требуется руководство и стимулирование персонала при проведении требуемого документирования и обеспечение его ресурсами для содействия в данной работе.

Для этого требуется обеспечить:

а) опубликованные официальные отчеты о стратегии документирования;

б) стандарты и руководства, определяющие все аспекты документирования программного обеспечения;

в) опубликованные процедуры документирования;

г) выделение соответствующих ресурсов для документирования;

д) планирование документирования, осуществляемое как неотъемлемая часть процесса разработки программного обеспечения;

е) постоянную проверку, осуществляемую для обеспечения соответствия со стратегией, стандартами, процедурами и планами по документированию.

6.1.3. Функции программной документации

Для эффективного управления документированием программного обеспечения важно осознавать различные функции, выполняемые документацией.

Программную документацию можно рассматривать как имеющую шесть основных функций:

информация для управления;

связь между задачами;

обеспечение качества;

инструкции и справки;

сопровождение программного обеспечения;

исторические справки.

6.1.3.1. Информация для управления

Во время разработки программного обеспечения администрации необходимо оценивать ход работы, возникающие проблемы и вероятности развития процесса. Периодические отчеты, согласно которым проверяют ход работ по графику и представляют планы на следующий период, обеспечивают контрольные механизмы и обзор проекта.

6.1.3.2. Связь между задачами

Большинство проектов разработки программного обеспечения разделяется на задачи, зачастую выполняемые различными группами.

В типовом варианте:

специалисты в предметной области начинают проект;

аналитики формулируют требования к системе;

проектировщики разрабатывают системный и программный проекты;

специалисты по изданиям создают пользовательскую документацию в соответствии со стратегией и стандартами по документированию;

специалисты по обеспечению качества и ревизоры оценивают общую полноту и качество функционирования программного обеспечения;

сопровождающие программисты улучшают эксплуатируемое программное обеспечение и разрабатывают его изменения или расширения.

Этим людям необходимы средства общения друг с другом, обеспечивающие информацию, которую можно, при необходимости, воспроизводить, распространять и на которую можно ссылаться.

Большинство методологий разработки программного обеспечения устанавливают официальные документы для связи между задачами. Например, аналитики представляют официальные спецификации требований для проектировщиков, а проектировщики выдают официальные проектные спецификации для программистов.

6.1.3.3. Обеспечение качества

Требуется документация разработки и документация продукции для выполнения задач, связанных с обязанностями по обеспечению качества программного обеспечения.

6.1.3.4. Инструкции и справки

Документация, требующаяся операторам, пользователям, руководителям и другим заинтересованным лицам для того, чтобы понимать и использовать программную продукцию.

6.1.3.5. Сопровождение программного обеспечения

Сопровождающим программистам требуется детальное описание программного обеспечения, такое, чтобы они могли локализовать и корректировать ошибки и модернизировать или изменять программное обеспечение соответствующим образом.

6.1.3.6. Исторические справки

Документация, требуемая в качестве исторической справки по проекту. Данная документация может помочь в переносе и переводе программного обеспечения в новое окружение.

6.1.4. Установление стратегии документирования

Стратегии документирования, подготовленные и отслеживаемые главной администрацией, обеспечивают руководства для ответственных лиц, принимающих решения на всех нижних уровнях. Стратегия обеспечивает главное направление, но не дает рекомендаций, что делать или как это делать.

Из-за существенной роли, которую играет документация на всех стадиях жизненного цикла программного обеспечения, должна быть подготовлена официально утвержденная стратегия. Каждый затронутый стратегией должен быть информирован о ней и должен ее понимать. Официальная, описанная, разрекламированная стратегия устанавливает дисциплину, требуемую для эффективного документирования программного обеспечения.

Стратегия должна поддерживать основные элементы эффективного документирования:

1. Требования документации охватывают весь жизненный цикл программного обеспечения.

Документация требуется на ранних стадиях проекта и должна быть доступна и сопровождаться на всем протяжении процесса разработки программного обеспечения. После завершения процесса разработки документация необходима для использования, сопровождения, модернизации, преобразования или передачи программного обеспечения.

2. Документирование должно быть управляемым.

Управление и контроль требуются для получения и сопровождения документации. Руководители и специалисты по изданиям должны подготовить подробные планы, охватывающие документирование продукции, графиков, обязанностей, ресурсов, обеспечения качества и процедур проверок;

3. Документация должна соответствовать ее читательской аудитории.

Читателями могут быть руководители, аналитики, специалисты по экспертным системам, сопровождающие программисты, канцелярский персонал и т. д. В зависимости от выполняемых задач им требуются различные степени детализации и различное представление материала. Специалисты по изда-

ниям должны быть готовы соответствующим образом спроектировать различные типы документации, предназначенные для различных читателей.

4. Работы по документированию должны быть объединены в общий процесс разработки программного обеспечения.

Процесс разработки должен быть определен.

5. Должны быть определены и использованы стандарты по документированию.

По возможности, должны быть приняты существующие стандарты. Когда подходящие стандарты отсутствуют, должны быть разработаны требуемые стандарты и руководства.

6. Должны быть определены средства поддержки.

Должны быть использованы там, где это экономически целесообразно, средства, помогающие разработке и сопровождению программной продукции, включая документацию.

6.1.5. Определение стандартов и руководств по документированию

Внутри организаций должны быть приняты стандарты и руководства:
для модели жизненного цикла программного обеспечения;
типов и взаимосвязей документов;
содержания документа;
качества документа;
форматов документа;
обозначения документа.

Данные стандарты и руководства будут определять, как следует выполнять задачи документирования, и будут обеспечивать критерии для оценки полноты, полезности и соответствия программной документации, создаваемой в организации.

По возможности должны быть приняты действующие международные и национальные стандарты. Если подходящие стандарты отсутствуют, то организация должна разработать собственные.

Большинство стандартов и руководств выдают рекомендации, которые применимы на общем уровне. Зачастую будут требоваться управленческие решения для адаптации общих рекомендаций к конкретным проектам. Применение стандартов, распространяющихся на организацию документирования, облегчит руководителям проекта определение следующих вопросов:

Какие типы документов требуются?
Каков объем представляемой документации?
Что документы содержат?
Какой уровень качества будет достигнут?
Где документы будут созданы?
Как документы будут хранить, сопровождать и обращать?

Если возможен контракт на программное обеспечение, контракт должен требовать, чтобы документация удовлетворяла принятым стандартам. Он

должен определять типы составляемых документов, уровень качества каждого и процедуры их проверки и утверждения.

6.1.5.1. Выбор модели жизненного цикла программного обеспечения

Существует ряд моделей жизненного цикла программного обеспечения с отличающейся терминологией для различных стадий. С точки зрения программной документации, не имеет значения, какая модель выбрана, до тех пор, пока стадии и соответствующая им документация четко определены, спланированы и выполняемы для любого конкретного программного проекта. Поэтому руководители должны выбрать соответствующую модель жизненного цикла программного обеспечения и гарантировать, чтобы ее применяли в данной организации.

Руководители должны убедиться, что выбранные стадии и соответствующие задачи помогут им в контроле за ходом любого программного проекта. Создание документации, связанной с конкретной стадией, может, например, быть использовано как контрольный пункт для проверки, приемки и завершения стадии до начала следующей.

6.1.5.2. Определение типов и содержания документов

Ниже дана схема основных типов программных документов. Данная схема не является исчерпывающей или окончательной, но будет служить контрольной таблицей основных типов программных документов, которые руководители должны предусмотреть, когда определяют стандартные типы своих документов.

Программные документы можно представить разделенными на три категории:

1) документация разработки;
2) документация продукции;
3) документация управления проектом.

Документация разработки. Документы, описывающие процесс разработки программного обеспечения, определяют требования, которым должно удовлетворять программное обеспечение, проект программного обеспечения, определяют, как его контролируют и как обеспечивают его качество. Документация разработки также включает подробное техническое описание программного обеспечения (программную логику, «программные» взаимосвязи, форматы и хранение данных и т. д.).

Документы, описывающие процесс разработки:

1) являются средством связи между всеми вовлеченными в процесс разработки. Описывают подробности решений, принятых относительно требований к программному обеспечению, проекту, программированию и тестированию;

2) описывают обязанности группы разработки. Определяют, кто, что и когда делает, учитывая роль программного обеспечения, предмета работ, документации, персонала, обеспечивающего качество, и каждого вовлеченного в процесс разработки;

3) выступают как контрольные пункты, которые позволяют руководителям оценивать ход разработки. Если документы разработки отсутствуют, неполны или устарели, руководители теряют важное средство для отслеживания и контроля проекта программного обеспечения;

4) образуют основу документации сопровождения программного обеспечения, требуемой сопровождающими программистами как часть документации продукции;

5) описывают историю разработки программного обеспечения.

Типовыми документами разработки являются:

1) анализы осуществимости и исходные заявки;
2) спецификации требований;
3) спецификации функций;
4) проектные спецификации, включая спецификации программ и данных;
5) планы разработки;
6) планы сборки и тестирования программного обеспечения;
7) планы обеспечения качества, стандарты и графики;
8) защитная и тестовая информация.

Документация продукции. Документация продукции обеспечивает информацию, необходимую для эксплуатации, сопровождения, модернизации, преобразования и передачи программной продукции.

Документация продукции преследует три цели:

обеспечить учебную и справочную информацию для любого использующего или эксплуатирующего программную продукцию;

облегчить программистам, не разрабатывавшим программное обеспечение, его сопровождение и модернизацию;

помочь продаже или приемке программной продукции.

Документация продукции должна включать в себя материалы для следующих типов читателей:

пользователей, которые вводят данные, восстанавливают информацию и решают задачи с помощью программного обеспечения;

операторов, которые «прогоняют» программное обеспечение на вычислительной системе;

сопровождающих программистов, которые сопровождают, модернизируют или изменяют программное обеспечение.

Документация продукции может также включать:

руководства и материалы для руководителей, которые следят за использованием программного обеспечения;

вспомогательные материалы, освещающие возможности программной продукции и уточняющие ее функции, условия эксплуатации и т. д.;

общую информацию, описывающую программную продукцию для всех заинтересованных лиц.

Типовые документы продукции включают:

учебные руководства;

справочные руководства и руководства пользователя;

руководства по сопровождению программного обеспечения;

брошюры и информационные листовки, посвященные продукции.

Документация управления проектом. Документы создают на основе информации управления проектом, такой как:

графики для каждой стадии процесса разработки и отчеты об изменениях графиков;

отчеты о согласованных изменениях программного обеспечения;

отчеты о решениях, связанных с разработкой;

распределение обязанностей.

Данная документация обеспечивает информацию, относящуюся, с точки зрения руководства, к долговечности продукции.

6.1.5.3. Определение качества документов

Руководители должны выбирать стандарты, распространяющиеся на уровень качества, соответственно различным типам документов и различным типам проектов и должны определять, как это качество будет достигнуто и поддержано.

Понятия качества, применимые к содержанию, структуре и представлению документации:

качество содержания можно измерять в элементах точности, полноты и ясности;

качество структуры можно измерять легкостью, с которой читатель имеет возможность определить местоположение информации;

качество представления должно соответствовать типу проекта. Например, руководство пользователя может иметь форму набора машинописных страниц, скрепленных вместе, или может быть типографской книгой с обширными иллюстрациями, созданной специалистом по графике.

6.1.5.4. Определение форматов документов

Стандартизованные форматы документов важны для контроля качества документов, для читаемости документов и для облегчения их сопровождения.

Информация может быть представлена в различных форматах. Проектные спецификации, например, могут быть записаны на установленных бланках. Обучение пользователя можно осуществить посредством учебных программ в классах или учебных конспектов и консультаций.

Форматы документов могут различаться от проекта к проекту. Они зависят от таких факторов, как объем проекта, аудитория,, для которой предназначены документы, количество установленных стадий и бюджет документирования.

В проектируемых форматах должны быть учтены соображения о том, будут ли документы переводить для международного распространения.

Стандарты и руководства данной организации, распространяющиеся на форматы документов, должны быть установлены таким образом, чтобы допускать гибкость для руководителей в выборе форматов, подходящих для их проектов.

6.1.5.5. Определение системы обозначения документов

Стандартные обозначения документов необходимы для эффективного контроля документации. Обозначающая информация может включать:

заглавие документа;
ссылочный номер документа;
номер версии документа;
дату выпуска и пересмотра;
реквизиты автора;
реквизиты утвердившего лица;
обозначение защищенности (авторских прав);
обозначение организации.

Если документы выпускают в виде разрозненных листов, каждая страница должна иметь индивидуальное обозначение (например, со ссылочным номером документа, номером страницы и номером издания).

6.1.6. Установление процедуры документирования

Должны быть установлены процедуры для применяемых в организациях стратегий документирования.

Процедуры определяют последовательность документирования:

планирование;
подготовка;
конфигурационное управление;
проверка;
утверждение;
производство;
хранение; дублирование;
распространение и модернизация; продажа.

Процедуры также должны определять контрольные пункты и методы обеспечения качества.

6.1.7. Распределение ресурсов для документирования

Основными ресурсами, требуемыми для документирования, являются следующие:

персонал;

средства;

финансирование.

6.1.7.1. Персонал

Для процесса разработки программного обеспечения необходимы люди со знанием:

программирования – для разработки программного обеспечения;

сути предмета – для представления информации о применениях программного обеспечения;

документирования – для разработки документации продукции. Важно, чтобы штат был полностью обучен методам документирования и чтобы каждая группа полностью понимала и выполняла свою роль в документировании.

Кроме того, нужны:

проектировщики программного обеспечения и программисты, которые создают документацию разработки, описывающую продукцию или ее задачи; они также обеспечивают документацию сопровождения программной продукции;

специалисты в предметной области, обеспечивающие информацию для (и могут разрабатывать) части стадий изучения, спецификаций требований, планов тестирования и обеспечения качества, планов сборки программного обеспечения в условиях эксплуатации и многих типов графиков;

специалисты по изданию, которые обычно подготавливают документацию по обучению пользователя, а также справочную, информационную документацию о продукции и посредническую.

6.1.7.2. Средства

Важно предусмотреть обеспечение задач документирования соответствующими и подходящими средствами.

Инструментальные программные средства полезны для подготовки и контроля документации. Они могут быть применены для повышения эффективности многих процессов документирования и использования стандартов данной организации, распространяющихся на документирование.

6.1.7.3. Финансирование

Важно, чтобы стоимость документирования определяли как отдельные статьи бюджета, так как она нередко составляет значительную часть стоимости разработки программного обеспечения.

6.1.8. Планирование документирования

План документирования определяет, что должно быть сделано, как это должно быть сделано, когда это должно быть сделано и кто это должен делать.

План документирования может быть частью общего плана проектирования или отдельным документом. Для небольших, неофициальных проектов план может быть всего на одной странице. Для больших проектов это может быть объемный документ, который следует установленным стандартам и является предметом для официальной проверки и процедуры утверждения.

План документирования должен быть доведен до всех участников разрабатывающего коллектива и до всех, кого он касается. Должны быть четко установлены обязанности всех вовлеченных в работу, связанную с документированием.

План должен включать изложение:

общей структуры документации;

типов и содержания документов;

качества и форматов документов;

обозначения документов;

комплектности и хранения документов;

обращения документов;

графика документирования.

График документирования должен распределять время:

для планирования документов;

проверки плана документирования и принципов документирования;

подготовки проектов и проверки их на техническую точность, полноту и соответствие;

редактирования при внесении комментариев, появившихся при проверке;

проведения согласования;

перевода (например, с японского на французский);

распространения.

Планирование следует начинать заранее, и план необходимо проверять на всем протяжении проекта. Подобно любому плану, план документирования отражает намечаемые действия и является объектом для необходимых изменений. В проекте должны быть предусмотрены регулярные проверки результативности изменений в плане.

6.2. Требования к содержанию документов на автоматизированные системы

Требования к содержанию документов изложены в РД 50-34.698–90. Настоящие методические указания распространяются на автоматизированные системы (АС), используемые в различных сферах деятельности (управление, исследование, проектирование и т. п.), включая их сочетание, и устанавливают требования к содержанию документов, разрабатываемых при создании АС.

6.2.1. Общие положения

Требования к содержанию документов, разрабатываемых при создании АС, установлены указаниями РД 50-34.698–90, а также соответствующими государственными стандартами Единой системы программной документации (ЕСПД), Единой системы конструкторской документации (ЕСКД), Системы проектной документации для строительства (СПДС) и ГОСТ 34.602.

Виды и комплектность документов регламентированы ГОСТ 34.201.

Содержание документов является общим для всех видов АС и, при необходимости, может дополняться разработчиком документов, в зависимости от особенностей создаваемой АС. Допускается включать в документы дополнительные разделы и сведения, объединять и исключать разделы.

Содержание каждого документа, разрабатываемого при проектировании АС согласно ГОСТ 34.201, определяет разработчик в зависимости от объекта проектирования (система, подсистема и т. д.).

Содержание документов, разрабатываемых на предпроектных стадиях по ГОСТ 34.601 и организационно-распорядительных, определяют разработчики в зависимости от объема информации, необходимой и достаточной для дальнейшего использования документов.

Документы, при необходимости, сброшюровывают в книги или тома, к которым составляют описи.

6.2.2. Требования к содержанию документов по общесистемным решениям

6.2.2.1. Ведомость эскизного (технического) проекта

Ведомость содержит перечень всех документов, разработанных на соответствующих стадиях создания АС и применяемых из проектов других АС. Ведомость заполняют по разделам-частям проекта АС. Документ выполняется по ГОСТ 2.106. Наименования разделов и подразделов записывают в графах «Обозначение» и «Наименование» в виде заголовков и выделяют подчеркиванием.

6.2.2.2. Пояснительные записки к эскизному, техническому проектам

Документы содержат разделы:

1. Общие положения;
2. Описание процесса деятельности;
3. Основные технические решения;
4. Мероприятия по подготовке объекта автоматизации к вводу системы в действие.

В разделе «Общие положения» приводят:

1) наименование проектируемой АС и наименования документов, их номера и дату утверждения, на основании которых ведут проектирование АС;

2) перечень организаций, участвующих в разработке системы, сроки выполнения стадий;

3) цели, назначение и области использования АС;

4) подтверждение соответствия проектных решений действующим нормам и правилам техники безопасности, пожаро- и взрывобезопасности и т. п.;

5) сведения об использованных при проектировании нормативно-технических документах;

6) сведения о НИР, передовом опыте, изобретениях, использованных при разработке проекта;

7) очередность создания системы и объем каждой очереди.

В разделе «Описание процесса деятельности» отражают состав процедур (операций) с учетом обеспечения взаимосвязи и совместимости процессов автоматизированной и неавтоматизированной деятельности, формируют требования к организации работ в условиях функционирования АС.

В разделе «Основные технические решения» приводят:

1) решения по структуре системы, подсистем, средствам и способам связи для информационного обмена между компонентами системы, подсистем:

2) решения по взаимосвязям АС со смежными системами, обеспечению ее совместимости;

3) решения по режимам функционирования, диагностированию работы системы;

4) решения по численности, квалификации и функциям персонала АС, режимам его работы, порядку взаимодействия;

5) сведения об обеспечении заданных в техническом задании (ТЗ) потребительских характеристик системы (подсистем), определяющих ее качество;

6) состав функций, комплексов задач (задач), реализуемых системой (подсистемой);

7) решения по комплексу технических средств, его размещению на объекте;

8) решения по составу информации, объему, способам ее организации, видам машинных носителей, входным и выходным документам и сообщениям, последовательности обработки информации и другим компонентам;

9) решения по составу программных средств, языкам деятельности, алгоритмам процедур и операций и методам их реализации.

В разделе «Мероприятия по подготовке объекта автоматизации к вводу системы в действие» приводят:

1) мероприятия по приведению информации к виду, пригодному для обработки на ЭВМ;

2) мероприятия по обучению и проверке квалификации персонала;

3) мероприятия по созданию необходимых подразделений и рабочих мест;

4) мероприятия по изменению объекта автоматизации;

5) другие мероприятия, исходящие из специфических особенностей создаваемых АС.

6.2.2.3. Схема функциональной структуры

Документ «Схема функциональной структуры» содержит:

1) элементы функциональной структуры АС (подсистемы АС); автоматизированные функции и (или) задачи (комплексы задач); совокупности действий (операций), выполняемых при реализации автоматизированных функций только техническими средствами (автоматически) или только человеком;

2) информационные связи между элементами и с внешней средой с кратким указанием содержания сообщений и (или) сигналов, передаваемых по связям, и при необходимости, связи других типов (входимости, подчинения и т. д.);

3) детализированные схемы частей функциональной структуры (при необходимости).

6.2.2.4. Описание автоматизируемых функций

Документ «Описание автоматизируемых функций» содержит разделы:

1. Исходные данные;
2. Цели АС и автоматизированные функции;
3. Характеристика функциональной структуры;
4. Типовые решения (при наличии).

В разделе «Исходные данные» приводят:

1) перечень исходных материалов и документов, использованных при разработке функциональной части проекта АС;

2) особенности объекта управления, влияющие на проектные решения по автоматизированным функциям;

3) данные о системах управления, взаимосвязанных с разрабатываемой АС, и сведения об информации, которой она должна обмениваться с абонентами и другими системами;

4) описание информационной модели объекта вместе с его системой управления.

В разделе «Цели АС и автоматизированные функции» приводят описание автоматизированных функций, направленных на достижение установленных целей.

Раздел «Характеристика функциональной структуры» содержит:

1) перечень подсистем АС с указанием функций и (или) задач, реализуемых в каждой подсистеме;

2) описание процесса выполнения функций (при необходимости);

3) необходимые пояснения к разделению автоматизированных функций на действия (операции), выполняемые техническими средствами и человеком;

4) требования к временному регламенту и характеристикам процесса реализации автоматизированных функций (точности, надежности и т. п.) и решения задач.

В разделе «Типовые решения» приводят перечень типовых решений с указанием функций, задач, комплексов задач, для выполнения которых они применены.

6.2.2.5. Описание постановки задачи (комплекса задач)

Документ содержит разделы:

1. Характеристики комплекса задач;
2. Выходная информация;
3. Входная информация.

В разделе «Характеристики комплекса задач» приводят:

1) назначение комплекса задач;

2) перечень объектов (технологических объектов управления, подразделений предприятия и т. п.), при управлении которыми решают комплекс задач;

3) периодичность и продолжительность решения;

4) условия, при которых прекращается решение комплекса задач автоматизированным способом (при необходимости);

5) связи данного комплекса задач с другими комплексами (задачами) АС;

6) должности лиц и (или) наименования подразделений, определяющих условия и временные характеристики конкретного, решения задачи (если они не определены общим алгоритмом функционирования системы);

7) распределение действий между персоналом и техническими средствами при различных ситуациях решения комплекса задач.

Раздел «Выходная информация» содержит:

1) перечень и описание выходных сообщений;

2) перечень и описание имеющих самостоятельное смысловое значение структурных единиц информации выходных сообщений (показателей, реквизитов и их совокупностей, сигналов управления) или ссылку на документы, содержащие эти данные.

В описании по каждому выходному сообщению следует указывать:

1) идентификатор;

2) форму представления сообщения (документ, видеокадр, сигнал управления) и требования к ней;

3) периодичность выдачи;

4) сроки выдачи и допустимое время задержки решения;

5) получателей и назначение выходной информации.

В описании по каждой структурной единице информации следует указывать:

1) наименование;

2) идентификатор выходного сообщения, содержащего структурную единицу информации;

3) требования к точности и надежности вычисления (при необходимости).

Раздел «Входная информация» должен содержать:

1) перечень и описание входных сообщений (идентификатор, форму представления, сроки и частоту поступления);

2) перечень и описание структурных единиц информации входных сообщений или ссылку на документы, содержащие эти данные.

В описании по каждой структурной единице информации входных сообщений следует указывать;

1) наименование;

2) требуемую точность ее числового значения (при необходимости);

3) источник информации (документ, видеокадр, устройство, кодограмма, информационная база на машинных носителях и т. д.);

4) идентификатор источника информации.

Допускается давать в виде приложений иллюстрационный материал, таблицы или текст вспомогательного характера, а также документы, имеющие самостоятельные обозначения (чертежи форм документов, описание массивов информации, схемы и т. д.).

6.2.2.6. Локальная смета и локальный сметный расчет

Локальная смета и локальный сметный расчет содержат сведения о сметной стоимости работ, выполняемых при создании АС, и сметной стоимости объектов, сооружаемых при создании АС, в соответствии с требованиями СНиП 1.02.01 и других документов по определению стоимости АС и ее составных частей.

6.2.2.7. Паспорт

Документ содержит разделы:

1. Общие сведения об АС;
2. Основные характеристики АС;
3. Комплектность;
4. Свидетельство (акт) о приемке;
5. Гарантии изготовителя (поставщика);
6. Сведения о рекламациях.

В разделе «Общие сведения об АС» указывают наименование АС, ее обозначение, присвоенное разработчиком, наименование предприятия-поставщика и другие сведения об АС в целом.

В разделе «Основные характеристики АС» должны быть приведены:

1) сведения о составе функций, реализуемых АС, в том числе измерительных и управляющих;

2) описание принципа функционирования АС;

3) общий регламент и режимы функционирования АС и сведения о возможности изменения режимов ее работы;

4) сведения о совместимости АС с другими системами.

В разделе «Комплектность» указывают все непосредственно входящие в состав АС комплексы технических и программных средств, отдельные средства, в том числе носители данных и эксплуатационные документы.

В разделе «Свидетельство о приемке» приводят дату подписания акта о приемке АС в промышленную эксплуатацию и фамилии лиц, подписавших акт.

В разделе «Гарантии изготовителя» приводят сроки гарантии АС в целом и ее отдельных составных частей, если эти сроки не совпадают со сроками гарантии АС в целом.

В разделе «Сведения о рекламациях» регистрируют все предъявленные рекламации, их краткое содержание и меры, принятые по рекламациям.

6.2.2.8. Формуляр

Документ содержит разделы:

1. Общие сведения;
2. Основные характеристики;
3. Комплектность;
4. Свидетельство о приемке;
5. Гарантийные обязательства;
6. Сведения о состоянии АС;
7. Сведения о рекламациях.

В разделе «Общие сведения» указывают наименование АС, ее обозначение, присвоенное разработчиком, наименование разработчика, дата сдачи АС в эксплуатацию, общие указания персоналу по эксплуатации АС, требования по ведению формуляра и месте его хранения, в том числе перечень технической документации, с которой должен быть ознакомлен персонал.

В разделе «Основные характеристики» указывают:

1) перечень реализуемых функций;

2) количественные и качественные характеристики АС и ее частей;

3) описание принципов функционирования АС, регламент и режимы функционирования;

4) сведения о взаимодействии АС с другими системами.

В разделе «Комплектность» указывают:

1) перечень технических и программных средств, в том числе носителей данных;

2) перечень эксплуатационных документов.

В разделе «Свидетельство о приемке» указывают:

1) даты подписания актов о приемке АС и ее частей в промышленную эксплуатацию;

2) фамилии председателей комиссий, осуществлявших приемку АС.

В разделе «Гарантийные обязательства» указывают:

1) гарантийные обязательства разработчиков АС по системе в целом и частям, имеющим разные гарантийные сроки;

2) перечень технических средств АС, имеющих гарантийные сроки службы меньше гарантийных сроков для системы.

В разделе «Сведения о состоянии АС» указывают:

1) сведения о неисправностях, в том числе дату, время, характер, причину возникновения, и о лицах, устранивших неисправность;

2) замечания по эксплуатации и аварийным ситуациям, принятые меры;

3) сведения о проведении проверок измерительных устройств и точностных характеристик измерительных каналов (для АСУ ТП);

4) сведения о ремонте технических средств и изменениях в программном обеспечении с указанием основания, даты и содержания изменения;

5) сведения о выполнении регламентных (профилактических работ и их результатах).

В разделе «Сведения о рекламациях» указывают сведения о рекламациях с указанием номера, даты, краткого содержания рекламационного акта, а также сведения об устранении замечаний, указанных в акте.

6.2.2.9. Проектная оценка надежности системы

Документ содержит разделы:

1. Введение;
2. Исходные данные;
3. Методика расчета;
4. Расчет показателей надежности;
5. Анализ результатов расчета.

В разделе «Введение» указывают:

1) назначение расчета надежности системы;

2) перечень оцениваемых показателей надежности;

3) состав учитываемых при расчете факторов, а также принятые допущения и ограничения.

В разделе «Исходные данные» приводят:

1) данные о надежности (паспортные и справочные) элементов АС, учитываемые при расчете надежности системы;

2) данные о режимах и условиях функционирования элементов АС;

3) сведения об организационных формах, режимах и параметрах эксплуатации АС.

В разделе «Методика расчета» указывают обоснование выбора методики расчета и нормативно-технический документ, согласно которому проводят расчет, или краткое описание методики расчета и ссылку на источники, где она опубликована.

В разделе «Расчет показателей надежности» указывают:

1) надежностные структуры компонентов АС (комплекса технических средств, программного обеспечения и персонала) по всем оцениваемым функциям (функциональным подсистемам) АС;

2) необходимые вычисления;

3) результаты расчета.

В разделе «Анализ результатов расчета» указывают:

1) итоговые данные расчета по каждой оцениваемой функции (функциональной подсистеме) АС и каждому нормируемому показателю надежности;

2) выводы о достаточности или недостаточности полученного уровня надежности АС по каждой оцениваемой функции (функциональной подсистеме) АС и, при необходимости, рекомендации по повышению надежности.

Если в обоснованных случаях при оценке надежности АС нельзя учесть уровень надежности программного обеспечения АС и уровень надежности действий персонала АС, то в документе «Проектная оценка надежности системы» указывают сведения по оценке надежности АС только с учетом надежности комплекса технических средств, в том числе нестандартных.

6.2.2.10. Общее описание системы

Документ содержит разделы:

1. Назначение системы;
2. Описание системы;
3. Описание взаимосвязей АС с другими системами;
4. Описание подсистем (при необходимости).

В разделе «Назначение системы» указывают:

1) вид деятельности, для автоматизации которой предназначена система;

2) перечень объектов автоматизации, на которых используется система;

3) перечень функций, реализуемых системой.

В разделе «Описание системы» указывают:

1) структуру системы и назначение ее частей;

2) сведения об АС в целом и ее частях, необходимые для обеспечения эксплуатации системы;

3) описание функционирования системы и ее частей.

В разделе «Описание взаимосвязей АС с другими системами» указывают:

1) перечень систем, с которыми связана данная АС;

2) описание связей между системами;

3) описание регламента связей;

4) описание взаимосвязей АС с подразделениями объекта автоматизации.

В разделе «Описание подсистем» указывают:

1) структуру подсистем и назначение ее частей;

2) сведения о подсистемах и их частях, необходимые для обеспечения их функционирования;

3) описание функционирования подсистем и их частей.

6.2.2.11. Программа и методика испытаний (компонентов, комплексов средств автоматизации, подсистем, систем)

Программа и методика испытаний комплекса средств автоматизации проектирования на этапе опытного функционирования предназначена для установления технических данных, подлежащих проверке при испытании компонентов АС и комплекса средств автоматизации проектирования, а также порядок испытаний и методы их контроля.

Программа и методика испытаний системы (подсистемы) на этапе опытного функционирования предназначена для установления данных, обеспечивающих получение и проверку проектных решений, выявление причин сбоев, определение качества работ, показателей качества функционирования системы (подсистемы), проверку соответствия системы требованиям техники безопасности, продолжительность и режим испытаний.

Программы испытаний должны содержать перечни конкретных проверок (решаемых задач), которые следует осуществлять при испытаниях для подтверждения выполнения требований ТЗ, со ссылками на соответствующие методики (разделы методик) испытаний.

Перечень проверок, подлежащих включению в программу испытаний, включает:

1) соответствие системы ТЗ;

2) комплектность системы;

3) комплектность и качество документации;

4) комплектность, достаточность состава и качество программных средств и программной документации;

5) количество и квалификация обслуживающего персонала;

6) степень выполнения требований функционального назначения системы;

7) контролепригодность системы;

8) выполнение требований техники безопасности, противопожарной безопасности, промышленной санитарии, эргономики;

9) функционирование системы с применением программных средств.

Описание методов испытаний системы по отдельным показателям рекомендуется располагать в той же последовательности, в которой эти показатели расположены в технических требованиях.

Программа испытаний содержит разделы:

1. Объект испытаний;
2. Цель испытаний;
3. Общие положения;
4. Объем испытаний;
5. Условия и порядок проведения испытаний;

6. Материально-техническое обеспечение испытаний;

7. Метрологическое обеспечение испытаний;

8. Отчетность.

В документ включают приложения.

В зависимости от особенностей систем допускается объединять или исключать отдельные разделы при условии изложения их содержания в других разделах программы испытаний, а также включать в нее дополнительные разделы (при необходимости).

В разделе «Объект испытаний» указывают:

1) полное наименование системы, обозначение;

2) комплектность испытательной системы.

В разделе «Цель испытаний» указывают конкретные цели и задачи, которые должны быть достигнуты и решены в процессе испытаний.

В разделе «Общие положения» указывают:

1) перечень руководящих документов, на основании которых проводят испытания;

2) место и продолжительность испытаний;

3) организации, участвующие в испытаниях;

4) перечень ранее проведенных испытаний;

5) перечень предъявляемых на испытания документов, откорректированных по результатам ранее проведенных испытаний.

В разделе «Объем испытаний» указывают:

1) перечень этапов испытаний и проверок, а также количественные и качественные характеристики, подлежащие оценке;

2) последовательность проведения и режим испытаний;

3) требования по испытаниям программных средств;

4) перечень работ, проводимых после завершения испытаний, требования к ним, объем и порядок проведения.

В разделе «Условия и порядок проведения испытаний» указывают:

1) условия проведения испытаний;

2) условия начала и завершения отдельных этапов испытаний;

3) имеющиеся ограничения в условиях проведения испытаний;

4) требования к техническому обслуживанию системы;

5) меры, обеспечивающие безопасность и безаварийность проведения испытаний;

6) порядок взаимодействия организаций, участвующих в испытаниях;

7) порядок привлечения экспертов для исследования возможных повреждений в процессе проведения испытаний;

8) требования к персоналу, проводящему испытания, и порядок его допуска к испытаниям.

В разделе «Материально-техническое обеспечение испытаний» указывают конкретные виды материально-технического обеспечения с распределением задач и обязанностей организации, участвующих в испытаниях.

В разделе «Метрологическое обеспечение испытаний» приводят перечень мероприятий по метрологическому обеспечению испытаний с распреде-

лением задач и ответственности организаций, участвующих в испытаниях, за выполнение соответствующих мероприятий.

В разделе «Отчетность» указывают перечень отчетных документов, которые должны оформляться в процессе испытаний и по их завершению, с указанием организаций и предприятий, разрабатывающих, согласующих и утверждающих их, и сроки оформления этих документов.

К отчетным документам относят акт и отчет о результатах испытаний, акт технического состояния системы после испытаний.

В приложения включают перечень методик испытаний, математических и комплексных моделей, применяемых для оценки характеристик системы.

При проведении испытаний в несколько этапов программы испытаний должны быть оформлены в виде единого документа.

Методики испытаний разрабатывают на основе ТЗ и утвержденных программ испытаний с использованием типовых методик испытаний (при наличии). При этом отдельные положения типовых методик испытаний могут уточняться и конкретизироваться в разрабатываемых методиках испытаний в зависимости от особенности системы и условий проведения испытаний. Содержание разделов методик устанавливает разработчик.

6.2.3. Требования к содержанию документов с решениями по организационному обеспечению

6.2.3.1. Описание организационной структуры

Документ содержит разделы:

1. Изменения в организационной структуре управления объектом;
2. Организация подразделений;
3. Реорганизация существующих подразделений управления.

В разделе «Изменения в организационной структуре управления объектом» указывают:

1) проектные решения по изменению организационной структуры управления объектом и их обоснование;

2) описание изменений во взаимосвязях между подразделениями.

В разделе «Организация подразделений» приводят:

1) описание организационной структуры и функций подразделений, создаваемых с целью обеспечения функционирования АС;

2) описание регламента работ;

3) перечень категорий работников и число штатных единиц.

В разделе «Реорганизация существующих подразделений управления» указывают описание изменений, обусловленных созданием АС, которые необходимо осуществить в каждом из действующих подразделений управления объектом: в организационной структуре, функциях подразделений, регламенте работы, составе персонала подразделений.

6.2.3.2. Методика (технология) автоматизированного проектирования

Документ «Методика автоматизированного проектирования» содержит разделы:

1. Общие положения;
2. Постановка задачи;
3. Методика проектирования;
4. Исходные данные;
5. Проектные процедуры;
6. Оценка результатов.

В разделе «Общие положения» указывают класс объектов, на которые распространена методика, состав специалистов-пользователей, требования и ограничения на условия применения методики.

В разделе «Постановка задачи» указывают основные пути и направления решения задачи, требования и ограничения на решение, критерии оценки результатов.

В разделе «Методика проектирования» описывают выбранные математические методы, используемые при проектировании, указывают состав и назначение проектных процедур, порядок взаимодействия проектных процедур в процессе выполнения.

В разделе «Исходные данные» определяют состав, порядок выбора, представления и формирования массивов используемой информации, перечень обозначений элементов, описывающих предметную область, с указанием их наименований, единиц измерений, диапазона изменения значений, критерии оценки исходных данных, выбирают методы и модели решения.

В разделе «Проектные процедуры» указывают по каждой проектной процедуре состав нормативно-справочных входных данных, правила доступа к ним, порядок выполнения процедуры, состав и форму выходных сообщений.

В разделе «Оценка результатов» приводят анализ полученного проектного решения на соответствие заданным критериям.

При проектировании конкретных объектов документ «Методика автоматизированного проектирования» может быть дополнен специфическими разделами, характерными для проектируемых объектов.

6.2.3.3. Технологическая инструкция

Документ «Технологическая инструкция» разрабатывают на операцию или комплекс операций технологического процесса обработки данных.

В документе указывают наименование технологической операции (операций), на которую разработан документ, и приводят сведения о порядке и правилах выполнения операций (операции) технологического процесса об-

работки данных. В инструкции приводят перечень должностей персонала, на которые распространяется данная инструкция.

Номенклатуру технологических инструкций определяют, исходя из принятого процесса обработки данных. Структуру документа устанавливает разработчик в зависимости от содержания.

6.2.3.4. Руководство пользователя

Документ содержит разделы:

1. Введение;
2. Назначение и условия применения;
3. Подготовка к работе;
4. Описание операций;
5. Аварийные ситуации;
6. Рекомендации по освоению.

В разделе «Введение» указывают:

1) область применения;
2) краткое описание возможностей;
3) уровень подготовки пользователя;
4) перечень эксплуатационной документации, с которой необходимо ознакомиться пользователю.

В разделе «Назначение и условия применения» указывают:

1) виды деятельности, функции, для автоматизации которых предназначено данное средство автоматизации;
2) условия, при соблюдении (выполнении, наступлении) которых обеспечивается применение средства автоматизации в соответствии с назначением (например, вид ЭВМ и конфигурация технических средств, операционная среда и общесистемные программные средства, входная информация, носители данных, база данных, требования к подготовке специалистов и т. п.).

В разделе «Подготовка к работе» указывают:

1) состав и содержание дистрибутивного носителя данных;
2) порядок загрузки данных и программ;
3) порядок проверки работоспособности.

В разделе «Описание операций» указывают:

1) описание всех выполняемых функций, задач, комплексов задач, процедур;
2) описание операций технологического процесса обработки данных, необходимых для выполнения функций, комплексов задач (задач), процедур.

Для каждой операции обработки данных указывают:

1) наименование;
2) условия, при соблюдении которых возможно выполнение операции;
3) подготовительные действия;
4) основные действия в требуемой последовательности;
5) заключительные действия;
6) ресурсы, расходуемые на операцию.

В описании действий допускаются ссылки на файлы подсказок, размещенные на магнитных носителях.

В разделе «Аварийные ситуации» указывают:

1) действия в случае несоблюдения условий выполнения технологического процесса, в том числе при длительных отказах технических средств;

2) действия по восстановлению программ и/или данных при отказе магнитных носителей или обнаружении ошибок в данных;

3) действия в случаях обнаружения несанкционированного вмешательства в данные;

4) действия в других аварийных ситуациях.

В разделе «Рекомендации по освоению» указывают рекомендации по освоению и эксплуатации, включая описание контрольного примера, правила его запуска и выполнения.

6.2.3.5. Описание технологического процесса обработки данных

Документ содержит разделы:

1. Технологический процесс сбора и обработки данных на периферийных устройствах при децентрализованной обработке данных;

2. Технологический процесс обработки данных на вычислительном центре.

В разделе «Технологический процесс сбора и обработки данных на периферийных устройствах при децентрализованной обработке данных» указывают:

1) состав и последовательность выполнения технологических операций по сбору, регистрации, подготовке, контролю, передаче, обработке и отображению информации;

2) перечень документации, сопровождающей каждую операцию в данном технологическом процессе.

В разделе «Технологический процесс обработки данных на вычислительном центре» указывают:

1) состав и последовательность выполнения технологических операций по приему, контролю, обработке, хранению, выдаче данных и других операций, выполняемых на вычислительном центре;

2) перечень документации, сопровождающей данный технологический процесс.

6.2.4. Требования к содержанию документов с решениями по программному обеспечению

6.2.4.1. Описание программного обеспечения

Документ содержит вводную часть и разделы:

1. Структура программного обеспечения;

2. Функции частей программного обеспечения;

3. Методы и средства разработки программного обеспечения;
4. Операционная система;
5. Средства, расширяющие возможности операционной системы.

В вводной части приводят основные сведения о техническом, информационном и других видах обеспечения АС, необходимые для разработки программного обеспечения или ссылку на соответствующие документы проекта АС.

В разделе «Структура программного обеспечения» приводят перечень частей программного обеспечения с указанием их взаимосвязей и обоснованием выделения каждой из них.

В разделе «Функции частей программного обеспечения» приводят назначение и описание основных функций для каждой части программного обеспечения.

В разделе «Методы и средства разработки программного обеспечения» приводят перечень методов программирования и средств разработки программного обеспечения АС с указанием частей программного обеспечения, при разработке которых следует использовать соответствующие методы и средства.

В разделе «Операционная система» указывают:

1) наименование, обозначение и краткую характеристику выбранной операционной системы и ее версии, в рамках которой будут выполнять разрабатываемые программы, с обоснованием выбора и указанием источников, где дано подробное описание выбранной версии;

2) наименование руководства, в соответствии с которым должна осуществляться генерация выбранного варианта операционной системы;

3) требования к варианту генерации выбранной версии операционной системы.

Раздел «Средства, расширяющие возможности операционной системы» содержит подразделы, в которых для каждого используемого средства, расширяющего возможности операционной системы, указывают:

1) наименование, обозначение и краткую характеристику средства с обоснованием необходимости его применения и указанием источника, где дано подробное описание выбранного средства;

2) наименование руководства, в соответствии с которым следует настраивать используемое средство на конкретное применение;

3) требования к настройке используемого средства.

6.2.5. Другие разделы

Данный комплекс стандартов и руководящих документов предусматривает описание требований и к другим видам обеспечения:

Требования к содержанию документов с решениями по техническому обеспечению.

Требования к содержанию документов с решениями по информационному обеспечению.

Требования к содержанию документов с решениями по математическому обеспечению.

6.3. Принципы разработки руководства программиста

Требования к содержанию руководства программиста изложены в ГОСТ 19.504–79. Настоящий стандарт устанавливает требования к содержанию и оформлению программного документа «Руководство программиста».

6.3.1. Общие положения

Структуру и оформление документа устанавливают в соответствии с ГОСТ 19.105–78. Составление информационной части (аннотации и содержания) является обязательным. Руководство программиста должно содержать следующие разделы:

1. Назначение и условия применения программ;
2. Характеристика программы;
3. Обращение к программе;
4. Входные и выходные данные;
5. Сообщения.

В зависимости от особенностей в документах допускается объединять отдельные разделы или вводить новые.

6.3.2. Содержание разделов

В разделе «Назначение и условия применения программ» должны быть указаны назначение и функции, выполняемые программой, условия, необходимые для выполнения программы (объем оперативной памяти, требования к составу и параметрам периферийных устройств, требования к программному обеспечению и т. п.).

В разделе «Характеристика программы» должно быть приведено описание основных характеристик и особенностей программы (временные характеристики, режим работы, средства контроля правильности выполнения и самовосстанавливаемости программы и т. п.).

В разделе «Обращение к программе» должно быть приведено описание процедур вызова программы (способы передачи управления и параметров данных и др.).

В разделе «Входные и выходные данные» должно быть приведено описание организации используемой входной и выходной информации и, при необходимости, ее кодирования.

В разделе «Сообщения» должны быть указаны тексты сообщений, выдаваемых программисту или оператору в ходе выполнения программы, опи-

сание их содержания и действий, которые необходимо предпринять по этим сообщениям.

В приложении к руководству программиста могут быть приведены дополнительные материалы (примеры, иллюстрации, таблицы, графики и т. п.).

6.4. Разработка руководства пользователя

6.4.1. Общие замечания

«Руководство пользователя» предназначено для организации эффективной работы пользователя с программным изделием. При изложении материала целесообразно использовать два стиля описания: в виде инструкций (обучающий) и в справочном виде. В то время как стиль инструкций ориентируется на оказание помощи новым пользователям, справочный стиль предназначен для более опытных пользователей, которым требуется информация по более специфическим вопросам.

В секции инструкций материал руководства должен быть упорядочен в соответствии с последовательностью обучения, начиная с простейших и наиболее необходимых операций, появляющихся в первую очередь, и заканчивая более сложными и появляющимися позже. Размер этой секции зависит либо от принятой методики обучения, либо от особенностей коллектива пользователей: некоторые из пользователей могут понять возможности и особенности использования программного изделия после знакомства с небольшим числом примеров, в то время как другим может потребоваться изучение многочисленных примеров.

В справочной секции должны быть представлены основные операции, упорядоченные для удобства использования, например, по алфавиту. Документация, представленная в этой секции, является более формальной, точной и исчерпывающей по сравнению с предыдущей секцией. Например, команда в секции инструкций может быть описана в конкретных понятиях в виде отдельного рабочего примера, а в справочной секции должны быть описаны все возможные параметры, опции и ключевые слова с несколькими примерами, уточняющими их смысл.

Целесообразно дать иллюстрации в виде экранов с описанием особенностей манипуляций на клавиатуре.

Разработка руководства пользователя должна начинаться как можно раньше. Прежде всего должен быть определен круг читателей, так как этот момент в значительной степени определяет стиль изложения. Руководство пользователя может быть создано в качестве он-лайнового средства помощи, т. е. в виде специального программного продукта. В этом случае должны быть разработаны отдельные требования для его проектирования.

Руководство пользователя содержит следующие разделы:

1. Общие сведения;

2. Описание применения;

3. Требования к процедурам.

В первом разделе обычно дается описание прикладной области и описываются основные функции изделия, а также условия его функционирования.

Во втором разделе рассматриваются выполняемые функции, более подробно описываются назначение программного изделия, предоставляемые им возможности для пользователя и отмечаются улучшения, которые появляются в работе пользователя при использовании программного изделия. Значительное внимание уделяется описанию условий эксплуатации, используемого оборудования и программных средств. Структура программного изделия дается с описанием роли каждой компоненты, а рабочие характеристики изделия в виде описания функциональных возможностей с указанием, где это оказывается возможным, количественных параметров входных и выходных потоков, времени реакции и т. п. Отдельно описывается база данных с указанием каждого файла и его назначения. Особое внимание уделяется описанию потоков обрабатываемых данных и результирующих выходов с указанием их взаимосвязей.

В третьем разделе руководства представлена информация о необходимых процедурах запуска системы, подготовки данных и настройки параметров. Кроме этого, здесь же должны быть описаны процедуры обработки ошибок, восстановления информации и требования к этим процедурам.

При описании процедур запуска характеризуется каждый шаг процедур, необходимых для организации работы. Значительное внимание должно быть уделено описанию процедур ввода данных, при этом определяются требования к процедуре подготовки данных: частоте ввода, источникам данных, носителям информации, ограничениям, контролям достоверности. Для организации ввода пользователю должны быть представлены макеты входных форм с подробным описанием назначения каждого реквизита, а для наиболее сложных форм целесообразно представить их образцы с сопутствующим описанием.

Затем должны быть описаны требования, предъявляемые к каждому выходному документу или экрану. При этом должны быть указаны: способ использования и частота выдачи, способ представления (носитель), инструкции по сохранению, распространению и т. д. В этом же разделе должны быть представлены описания всех выходных форм и экранов с объяснением каждого конкретного раздела формы. Для каждого типа результатов работы изделия должны быть приведены образцы с примерами результатов.

В отдельном подразделе руководства должны быть указаны возможные ошибки и процедуры их устранения. Целесообразно перечислить коды возможных ошибок, возникающих при работе программ и необходимые действия пользователя по восстановлению работоспособности программного изделия.

6.4.2. Содержание разделов руководства

Перечень разделов руководства пользователя может быть представлен следующим образом.

6.4.2.1. Общие сведения

Аннотация. Описывается прикладная область, для которой предназначено программное средство и указываются его основные функции.

Условия функционирования. Указывается, где предполагается устанавливать разработанное программное средство.

Используемые материалы. Перечисляются все необходимые документы, справочные материалы, документация по родственным разработкам.

6.4.2.2. Описание применения

Общее описание. Определяются назначение программного комплекса, его функциональные возможности и их совершенствование по сравнению с прежней системой, отмечаются выгоды его использования и особенности его применения.

Условия эксплуатации. Описываются оперативные взаимосвязи основных функций программного комплекса, которые связаны с вводом данных в систему и с выдачей, с использованием результатов функционирования системы в организации. Излагаются сведения об используемой системе защиты и безопасности программ и конфиденциальности информации базы данных. Изложение материала может иллюстрироваться схемами с описанием входных и выходных данных.

Оборудование. Перечисляются технические средства, необходимые для нормального функционирования программного комплекса.

Структура программного комплекса. Приводится общая архитектура программного комплекса и отмечается функциональное назначение каждой программной компоненты в системе.

Рабочие характеристики системы. Характеризуются функциональные возможности системы с указанием количественных параметров системы, включая пропускную способность системы (параметры входного потока транзакций, время их обработки), а также показатели надежности системы. Кроме этого, могут быть представлены качественные показатели системы, характеризующие пригодность к сопровождению, удобство эксплуатации и т. п.

База данных. Приводится общая логическая структура базы данных, и описываются все файлы с указанием их назначения.

6.4.2.3. Требования к процедурам функционирования системы

В этом разделе описываются действия пользователя системы при реализации процедур запуска системы, подготовки данных, настройки параметров системы, а также процедур обработки ошибок и восстановления информации.

Запуск системы. Дается пошаговое описание процедур в процессе инициализации процесса обработки данных в системе.

Ввод данных. Описываются общие требования, характеризующие подготовку данных в различных режимах функционирования системы (ведение базы данных, ввод транзакций, формирование запросов). Указываются источники данных (подразделения организации, занятой эксплуатацией системы), состав и необходимая квалификация персонала, ответственного за ввод данных, способ и режим ввода данных, а также количественные параметры входных потоков данных. В особых случаях оговариваются способы обеспечения и проверки достоверности вводимых данных, права доступа разных категорий пользователей к данным и к операциям над данными. При этом руководство может быть дополнено соответствующими инструкциями.

Форматы ввода. Должны быть представлены форматы всех входных форм и соответствующих экранов, которые используются для ввода данных. Достаточно подробно объясняется назначение каждого реквизита формы и принятые грамматические правила и ограничения при заполнении каждого конкретного реквизита.

Примеры форм для ввода данных (экранов). Приводятся примеры всех входных форм с подробным их описанием и заполненные конкретными данными. Для каждого реквизита указываются на примерах особенности вводимой информации.

Выходы. Описываются подробно требования к каждому выходному результату работы системы и особенности каждого выхода. Отмечаются вид выходного результата (отчет, экран), область использования, способы проверки достоверности, а также режим выдачи информации и количественные характеристики выходного потока.

Форматы вывода. Приводятся макеты всех выходных форм и экранов с объяснениями каждого раздела формы.

Образцы выходных форм. Приводятся примеры результатов каждого типа с определением смысла и способа использования всех переменных.

Возможные ошибки и процедуры их устранения. Перечисляются коды ошибок, возникающих в процессе работы программы, и описываются необходимые действия пользователя по их корректировке. Указываются процедуры восстановления работоспособности системы при ошибках.

Запросы к базе данных. Этот раздел присутствует при наличии запросного режима с выборкой информации из файлов базы данных. Описываются возможные запросы и инструкции по их инициализации, формам и содержанию данных, используемым командам. Характеризуются особенности каждого типа запроса и существующие ограничения.

ЗАКЛЮЧЕНИЕ

В настоящее время программная инженерия – молодая и быстро развивающаяся область. Она ориентирована на решение сложных задач, связанных с организацией, содержанием и управлением процессами разработки программных систем.

Программные системы – это уникальные продукты, не имеющие физической оболочки. Создание таких систем является одной из самых сложных задач и требует большого интеллектуального потенциала. Для решения таких задач делались попытки заимствования опыта из других областей, связанных с производством материальных ценностей, а также из области управления проектами. В результате для процесса разработки создавались совершенно новые подходы и технологии.

Современное общество уже немыслимо без информационных систем, которые пронизывают почти все области деятельности и постоянно развиваются и совершенствуются. Улучшение качества информационных систем неразрывно связано с процессом их производства. Каждый маленький элемент этого, теперь уже огромного процесса играет очень большое значение.

В данном курсе сделана попытка охватить весь процесс производства программного обеспечения с точки зрения содержательной части. Процессы управления и организации – предмет изучения другого курса, который может называться «Организация процесса разработки программного обеспечения».

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Арчибальд, Р. Управление высокотехнологичными программами и проектами / Рассел Д. Арчибальд; ред. А. Д. Баженов, А. О. Арефьева; пер. с. англ. Е. В. Мамонтова. – 3-е изд., перераб. и доп. – М.: ДМК Пресс; Компания АйТи, 2006. – 472 с.
2. Басс, Л. Архитектура программного обеспечения на практике / Л. Басс, П. Клементс, Р. Кацман. – 2-е изд. – СПб.: Питер, 2006. – 575 с.
3. Бек, К. Экстремальное программирование: разработка через тестирование / К. Бек. – СПб.: Питер, 2003. – 224 с. (Биб-ка программиста).
4. Бенькович, Е. С. Практическое моделирование динамических систем / Е. С. Бенькович, Ю. Б. Колесов, Ю. Б. Сениченков. – СПб.: БХВ-Петербург, 2002. – 464 с.
5. Благодатских, В. А. Стандартизация разработки программных средств: учеб. пособие / В. А. Благодатских, В. А. Волнин, К. Ф. Поскакалов; ред. О. С. Разумов. – М.: Финансы и статистика, 2003. – 288 с.
6. Бобровский, С. И. Технологии Delphi. Разработка приложений для бизнеса: учеб. курс / С. И. Бобровский. – СПб.: Питер, 2007. – 720 с.
7. Бобровский, С. И. Технологии Пентагона на службе российских программистов. Программная инженерия / С. И. Бобровский. – СПб.: Питер, 2003. – 222 с.
8. Боггс, Уэнди. UML и Rational Rose / Уэнди Боггс, Майкл Боггс. – М.: Изд-во «Лори», 2001. – 580 с.
9. Брауде, Э. Технология разработки программного обеспечения / Э. Брауде. – СПб.: Питер, 2004. – 655 с.
10. Брукс, Ф. Мифический человеко-месяц, или Как создаются программные системы / Ф. Брукс. – СПб.: Символ-Плюс, 2001. – 304 с.
11. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++ / Г. Буч. – 2-е изд. – М.: Изд-во «Бином», 1999. – 560 с.
12. Вигерс, Карл. Разработка требований к программному обеспечению / Карл Вигерс. – М.: Издательско-торговый дом «Русская редакция», 2004. – 576 с.
13. Войнов, И. В. Моделирование экономических систем и процессов. Опыт построения ARIS-моделей: монография / И. В. Войнов, С. Г. Пудовкина, А. И. Телегин. – Челябинск: Изд-во ЮУрГУ, 2002. – 392 с.
14. ГОСТ 19.001–77. Единая система программной документации. Общие положения.
15. ГОСТ 19.502–78. Единая система программной документации. Общее описание. Требования к содержанию и оформлению.
16. ГОСТ 19.504–79. Единая система программной документации. Руководство программиста. Требования к содержанию и оформлению.

17. ГОСТ 34.602–89. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы.

18. ГОСТ Р ИСО/МЭК 12207–99. Информационная технология. Процессы жизненного цикла программных средств.

19. ГОСТ Р ИСО/МЭК 15910–2002. Информационная технология. Процесс создания документации пользователя программного средства.

20. ГОСТ Р ИСО/МЭК ТО 9294–93. Информационная технология. Руководство по управлению документированием программного обеспечения.

21. ГОСТ Р ИСО/МЭК ТО 15271–2002. Информационная технология. Руководство по применению ГОСТ Р ИСО/МЭК 12207 (Процессы жизненного цикла программных средств).

22. ГОСТ Р ИСО/МЭК ТО 16326–2002. Программная инженерия. Руководство по применению ГОСТ Р ИСО/МЭК 12207 при управлении проектом.

23. ГОСТ Р ИСО/МЭК 12119–2000. Информационная технология. Пакеты программ. Требования к качеству и тестирование.

24. ГОСТ Р ИСО/МЭК 9126–93. Информационная технология. Оценка программной продукции. Характеристики качества и руководства по их применению.

25. ГОСТ Р ИСО/МЭК 8631–94. Информационная технология. Программные конструктивы и условные обозначения для их представления.

26. Грибачев, К. Г. Delphi и Model Driven Architecture. Разработка приложений баз данных / К. Г. Грибачев. – Спб.: Питер, 2004. – 348 с.

27. Иванова, Г. С. Технология программирования: учебник для вузов / Г. С. Иванова. – М.: Изд-во МГТУ им. Н. Э. Баумана, 2002. – 320 с.

28. Камаев, В. А. Технологии программирования / В. А. Камаев, В. В. Костерин. – М.: Высш. шк., 2005. – 359 с.

29. Константайн, Л. Разработка программного обеспечения / Л. Константайн, Л. Локвуд. – СПб.: Питер, 2004. – 592 с.

30. Кратчен, Филипп. Введение в Rational Unified Process / Филипп Кратчен. – 2-е изд. – М.: Издательский дом «Вильямс», 2002 – 240 с.

31. Кролл, П. Rational Unified Process – это легко. Руководство по RUP для практиков / П. Кролл, Ф. Крачтен. – М., КУДИЦ-ОБРАЗ, 2004. – 432 с.

32. Ларман, Крэг. Применение UML и шаблонов проектирования / Крэг Ларман. – 2-е изд. – М.: Издательский дом «Вильямс», 2002. – 624 с.

33. Леонников, А. В. Самоучитель UML / А. В. Леонников. – 2-е изд. – СПб: БХВ-Петербург, 2004. – 432 с.

34. Леффингуэлл, Дин. Принципы работы с требованиями к программному обеспечению. Унифицированный подход / Дин Леффингуэлл, Дон Уидриг. – Издательский дом «Вильямс», 2002. – 448 с.

35. Маклаков, С. В. Создание информационных систем с AllFussion Modelion Suite / С. В. Маклаков. – М.: ДИАЛОГ-МИФИ, 2003 – 432 с.

36. Модель зрелости процессов разработки программного обеспечения / Марк Паулк, Билл Куртис, Мэри Бет Хриссис и др. – М.: Богородский печатник, 2002. – 256 с.

37. Орлов, С. А. Технологии разработки программного обеспечения: учеб. пособие / С. А. Орлов. – 2-е изд. – СПб.: Питер, 2003. – 480 с.

38. Разработка программных проектов на основе Rational Unified Process (RUP) / Г. Полис, Л. Огастин, К. Лоу, Д. Мадхар. – М.: ООО «Бином-Пресс», 2005. – 256 с.

39. Рамбо, Дж. UML: специальный справочник / Дж. Рамбо, А. Якобсон, Г. Буч. – СПб.: Питер, 2002. – 656 с.

40. Рамбо, Дж. UML 2.0. Объектно-ориентированное моделирование и разработка / Дж. Рамбо, М. Блаха. – 2-е изд. – СПб.: Питер, 2007. – 544 с.

41. Садердинов, А. А. Построение комплексных программно-технических проектов интегрированных систем организационного управления (обобщение теории и практики проектирования) / А. А. Садердинов, В. А. Трайнев. – М.: Издательско-книготорговый центр «Маркетинг», 2001. – 287 с.

42. Соммервилл, Иан. Инженерия программного обеспечения / Иан Соммервилл. – 6-е изд. – М.: Издательский дом «Вильямс», 2002. – 624 с.

43. Сухомлин, В. А. Введение в анализ информационных технологий: учебник для вузов / В. А. Сухомлин. – М.: Горячая линия-Телеком, 2003. – 427 с.

44. Федотова, Д. Э. CASE-технологии: Практикум / Д. Э. Федотова, Ю. Д. Семенов, К. Н. Чижик. – М.: Горячая линия-Телеком, 2003. – 160 с.

45. Фаулер, М. UML. Основы / М. Фаулер: пер. с англ. – 3-е изд. – СПб: Символ-Плюс, 2004. – 192 с.

46. Чарнецки, К. Пораждающее программирование: методы, инструменты, применение / К. Чарнецки, У. Айзенкер. – СПб.: Питер, 2005. – 731 с.

47. Черемных, С. В. Моделирование и анализ систем. IDEF-технологии: практикум / С. В. Черемных, И. О. Семенов, В. С. Ручкин. – М.: Финансы и статистика, 2002. – 192 с.

48. Якобсон, А. Унифицированный процесс разработки программного обеспечения / А. Якобсон, Г. Буч, Дж. Рамбо. – СПб.: Питер, 2002. – 496 с.