

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ  
Федеральное государственное бюджетное  
образовательное учреждение  
высшего профессионального образования  
«Пензенский государственный университет» (ПГУ)

---

---

Т. В. Черушева

## ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Учебное пособие

Пенза  
Издательство ПГУ  
2014

УДК 004.415.2

Ч-45

Рецензенты:

кандидат технических наук,  
профессор кафедры «Прикладная информатика»  
Пензенского государственного  
технологического университета  
*В. В. Пикулин;*

кандидат технических наук, доцент кафедры  
«Информационно-вычислительные системы»  
Пензенского государственного университета  
*А. П. Писарев*

## Черушева, Т. В.

Ч-45      Проектирование программного обеспечения : учеб. пособие / Т. В. Черушева. – Пенза : Изд-во ПГУ, 2014. – 172 с.

ISBN 978-5-94170-859-8

Цель пособия – помочь в освоении современных методов и средств проектирования программного обеспечения информационных систем в экономике, технике и в других областях.

Описаны процессы, модели и стадии жизненного цикла программного обеспечения (ПО) экономических информационных систем. Приведены структурный и объектно-ориентированный подходы к проектированию ПО.

Отражено применение стандартного языка объектно-ориентированного моделирования и UML. Рассмотрены функции и компоненты CASE-средств и их практическое воплощение в наиболее развитых программных продуктах.

Издание подготовлено на кафедре «Высшая прикладная математика» Пензенского государственного университета и предназначено для бакалавров, обучающихся по направлению «Прикладная математика»; представляет интерес для научных исследований.

**УДК 004.415.2**

*Рекомендовано к изданию методической комиссией  
физико-математического факультета  
Пензенского государственного университета  
(протокол № 5 от 10 февраля 2014 г.)*

**ISBN 978-5-94170-859-8**

© Пензенский государственный  
университет, 2014

# Содержание

Введение .....	5
Глава 1. Жизненный цикл программного обеспечения .....	7
1.1. Требования к ПО (Software Requirements) .....	7
1.2. Процессы ЖЦ стандарта ISO/IEC 12207 .....	10
1.3. Типы моделей ЖЦ .....	16
1.3.1. Каскадная модель ЖЦ .....	16
1.3.2. Инкрементная модель ЖЦ .....	18
1.3.3. Спиральная модель .....	20
1.3.4. Эволюционная модель ЖЦ .....	22
1.3.5. Стандартизация модели ЖЦ .....	25
Глава 2. Сертификация и оценка процессов создания ПО .....	27
2.1. Понятие зрелости процессов создания ПО. Модель оценки зрелости СММ .....	27
2.2. Методика SPMN .....	35
Глава 3. Методы структурного и объектного анализа и построения моделей предметных областей .....	42
3.1. Визуальное моделирование .....	42
3.2. Структурные методы анализа и проектирования ПО .....	45
3.2.1. Метод функционального моделирования SADT .....	47
3.2.2. Метод моделирования процессов IDEF3 .....	55
3.3. Моделирование потоков данных .....	59
3.4. Основные принципы построения объектной модели .....	63
3.4.1. Основные элементы объектной модели .....	64
3.5. Краткий обзор объектно-ориентированных методов анализа и построения моделей .....	66
3.5.1. Основные понятия методов объектного анализа ПрО .....	68
3.5.2. Объектный метод построения моделей ПрО .....	70
3.6. Методы проектирования архитектуры ПО .....	81
3.6.1. Стандартный подход к проектированию .....	82
3.6.2. Общесистемный подход к проектированию архитектуры .....	84
Глава 4. Методы систематического программирования .....	95
4.1. UML-метод моделирования .....	97
4.3. Компонентный подход .....	101

4.4. Аспектно-ориентированное программирование.....	105
4.5. Генерирующее (порождающее) программирование .....	110
4.6. Агентное программирование .....	114
Глава 5. Промышленные технологии ППО.....	118
5.1. Методология DATARUN .....	118
5.1.2. Инструментальное средство SE Companion .....	124
5.1.3. Примеры ТС ПО различных компаний-поставщиков .....	126
5.1.3.1. Технология Rational Unified Process (IBM Rational Software).....	126
5.1.3.2. Технология Oracle .....	134
5.1.3.3. Технология Borland .....	140
5.1.3.4. Технология Computer Associates.....	143
5.2. Silverrun.....	144
5.3. Vantage Team Builder (Westmount I-CASE) .....	148
5.4. Designer/2000 + Developer/2000 .....	152
5.5. Локальные средства (ERwin, BPwin, S-Designor, CASE.Аналитик) .....	154
5.6. Объектно-ориентированные CASE-средства (Rational Rose).....	156
Заключение .....	159
Варианты заданий на курсовое проектирование .....	160
Варианты структур данных .....	163
Список литературы .....	171

# **Введение**

Проектирование экономических информационных систем (ЭИС) – логически сложная, трудоемкая и длительная работа, требующая высокой квалификации разработчиков. В процессе создания и функционирования ЭИС информационные потребности пользователей меняются, уточняются, что усложняет разработку и сопровождение таких систем.

Основная доля затрат приходится на прикладное программное обеспечение (ПО) и разработку базы данных (БД). На пути проектирования стоят преграды в виде нечеткой и неполной формулировки требований к ПО, недостаточного вовлечения пользователя в работу над проектом, недостатка ресурсов, отсутствия грамотного управления проектом, неудовлетворительного планирования, незнания новых технологий и т.д.

Необходимость контроля процесса разработки программного обеспечения привела к появлению совокупности методов и средств создания ПО, объединенных общим названием «программная инженерия». В основе ее заложена идея: проектирование ПО есть формальный процесс, который можно изучать и совершенствовать.

Для успешной реализации проекта объект проектирования должен быть описан с помощью полных и непротиворечивых моделей архитектуры ПО. Здесь закладываются структурные элементы системы, связи между ними, иерархия подсистем.

Модель – это полное описание системы ПО с некоторой точки зрения. Моделирование является центральным звеном всей работы по созданию качественного ПО. Модели строятся для того, чтобы понять структуру и поведение создаваемой системы, облегчить управление процессом ее создания, уменьшить возможный риск и документировать принимаемые проектные решения.

Язык моделирования должен включать элементы модели (фундаментальные концепции моделирования и их семантику), нотацию (визуальное представление элементов моделирования), руководство по использованию.

Конечная цель разработки ПО – получение работающих приложений (кода).

Проблемы разработки ПО породили потребность в программно-технологических средствах специального класса: CASE-средствах. Термин CASE (Computer Aided Software Engineering) охватывает процесс разработки сложных информационных систем (ИС) в це-

лом. CASE-технология есть совокупность методов проектирования ИС, также набор инструментальных средств, позволяющих в наглядной форме моделировать предметную область, анализировать эту модель и разрабатывать приложения в соответствии с информационными потребностями пользователей.

# **Глава 1. Жизненный цикл программного обеспечения**

## **1.1. Требования к ПО (Software Requirements)**

Методологическую основу проектирования ПО составляет системный подход. Под словом «система» понимается совокупность взаимодействующих компонентов и взаимосвязей между ними. Весь мир можно рассматривать как сложную взаимосвязанную совокупность естественных и искусственных систем.

Системный подход – это методология исследования объектов любой природы как систем, которая ориентирована:

- на раскрытие целостности объекта и обеспечивающих его механизмов;
- выявление многообразных типов связей объекта;
- сведение этих связей в единую картину.

Программное обеспечение в свою очередь определяется как набор компьютерных программ, процедур и, возможно, связанной с ними документации и данных.

По определению Института управления проектами (Project Management Institute, PMI) проект – это временное предприятие, осуществляющееся с целью создания уникального продукта или услуги. В любой инженерной дисциплине под проектированием обычно понимается некий унифицированный подход, с помощью которого мы ищем пути решения определенной проблемы, обеспечивая выполнение поставленной задачи. В контексте инженерного проектирования можно определить цель проектирования как создание системы, которая:

- удовлетворяет заданным (возможно, неформальным) функциональным спецификациям;
- согласована с ограничениями, накладываемыми оборудованием;
- удовлетворяет явным и неявным требованиям по эксплуатационным качествам и потреблению ресурсов;
- удовлетворяет явным и неявным критериям дизайна продукта;
- удовлетворяет требованиям к самому процессу разработки, таким, например, как продолжительность и стоимость, а также привлечение дополнительных инструментальных средств.

В другой формулировке цель проектирования – выявление ясной и относительно простой внутренней структуры, называемой ар-

хитектурой системы. Проект есть окончательный продукт процесса проектирования. Проектирование подразумевает учет противоречивых требований. Его продуктами являются модели, позволяющие понять структуру будущей системы, сбалансировать требования и наметить схему реализации.

Таким образом, под проектом ПО будем понимать совокупность спецификаций ПО (включающих модели и проектную документацию), обеспечивающих создание ПО в конкретной программно-технической среде.

Проектирование ПО представляет собой процесс создания спецификаций ПО на основе исходных требований к нему. Проектирование ПО сводится к последовательному уточнению его спецификаций на различных стадиях процесса создания ПО.

Международным комитетом при американском объединении компьютерных специалистов ACM (Association for Computing Machinery) и институте инженеров по электронике и электротехнике IEEE Computer Society было создано ядро знаний SWEBOK (рис. 1.1, 1.2). В этом ядре были систематизированы разнородные знания в области программирования, планирования и управления, сформулировано понятие программной инженерии и десяти областей, которые соответствуют процессам проектирования ПО и методам их поддержки.



Рис. 1.1. Основные области знаний SWEBOK



Рис. 1.2. Организационные области знаний SWEBOK

Понятие жизненного цикла ПО (ЖЦ ПО) является одним из базовых понятий программной инженерии.

ЖЦ ПО – это период времени, который начинается с момента решения о необходимости создания ПО и заканчивается полным изъятием его из эксплуатации.

Основным нормативным документом, регламентирующим состав процессов ЖЦ ПО, является международный стандарт ISO/IEC 12207: 1995 «Information Technology – Software Life Cycle Processes». Он определяет структуру ЖЦ, содержащую процессы, действия и задачи, которые должны быть выполнены во время создания ПО (его российский аналог ГОСТ Р ИСО/МЭК 12207–99 введен в действие в июле 2000 г.) (рис. 1.3). В данном стандарте процесс определяется как совокупность взаимосвязанных действий, преобразующих некоторые входные данные в выходные. Каждый процесс характеризуется определенными задачами и методами их решения, исходными данными, полученными от других процессов, и результатами.

Каждый процесс разделен на набор действий, каждое действие – на набор задач. Каждый процесс, действие или задача инициируется и выполняется другим процессом по мере необходимости, причем

не существует заранее определенных последовательностей выполнения (естественно, при сохранении связей по входным данным).

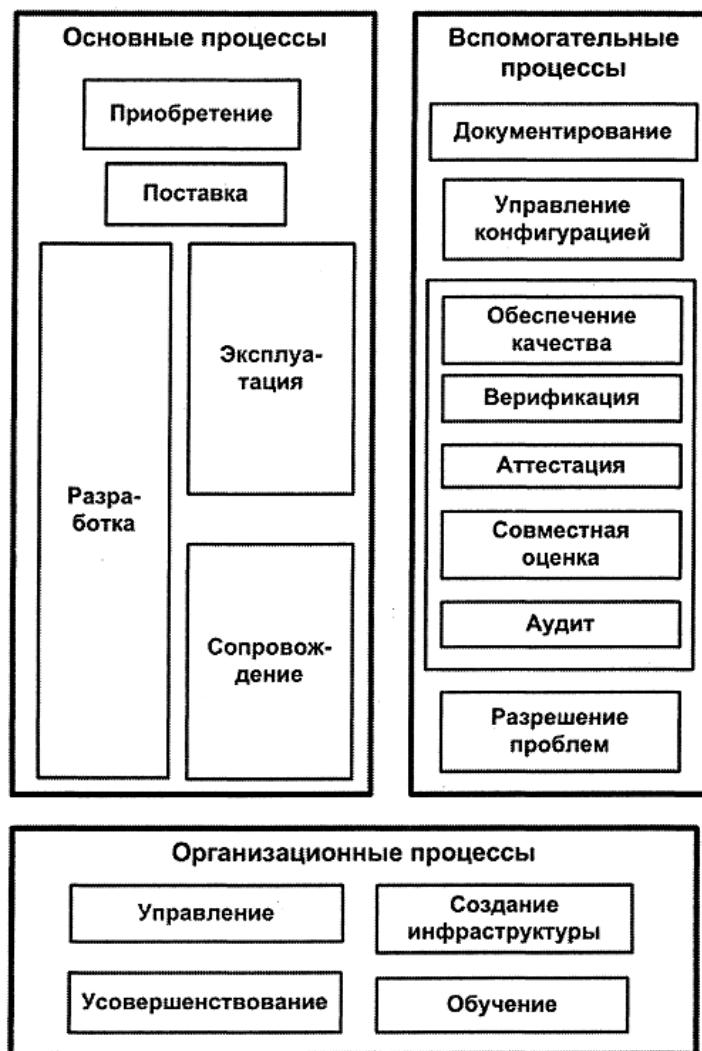


Рис. 1.3. Процессы ГОСТ Р ИСО/МЭК 12207–99

## 1.2. Процессы ЖЦ стандарта ISO/IEC 12207

При выборе схемы модели ЖЦ для конкретной предметной области решаются вопросы включения важных для создаваемого продукта видов работ или не включения несущественных работ. На сегодня основой формирования новой модели ЖЦ для конкретной прикладной системы является стандарт ISO/IEC 12207, который задает полный набор процессов (более 40), охватывающий все возможные виды работ и задач, связанных с построением программных средств (ПС), начиная с анализа предметной области и кончая изготавлением соответствующего продукта. Данный стандарт содержит основные и вспомогательные процессы (рис. 1.4, 1.5).

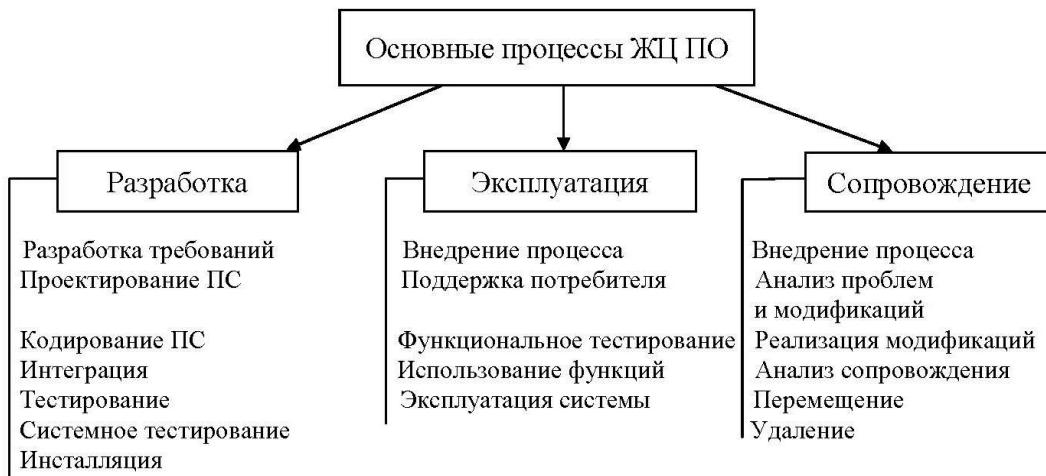


Рис. 1.4. Схема основных процессов ЖЦ ПС

На рис. 1.4 представлены процессы, связанные непосредственно с разработкой ПС. К категории основных процессов относятся также «первичные» процессы, определяющие порядок подготовки договора на разработку ПС, мониторинг деятельности поставщиков ПС заказчику.

Стандарт ISO/IEC 12207 предоставляет структуру процессов ЖЦ, но не обязывает использовать все процессы в модели ЖЦ ПО или в конкретной методологии разработки ПО. Являясь стандартом высокого уровня, он не задает детали того, как надо выполнять действия или задачи, составляющие процессы. Он также не задает требований к формату и содержанию документов, выпускаемых на разных процессах.



Рис. 1.5. Схема вспомогательных процессов ЖЦ ПС

Стандарт ISO/IEC 12207 является основой для принятия других стандартов, связанных с ним. Например, стандартов по управлению ПО, обеспечению качества, верификации и валидации, управлению конфигурацией, метриками ПО и т.д.

**Валидация требований** – это проверка изложенных в спецификации требований, выполняющаяся для того, чтобы путем отслеживания источников требований убедиться, что они определяют именно данную систему. Заказчик и разработчик ПО проводят экспертизу сформированного варианта требований с тем, чтобы разработчик мог далее проводить проектирование ПО. Один из методов валидации – прототипирование, т.е. быстрая отработка отдельных требований на конкретном инструменте и исследование масштабов изменения требований, измерение объема функциональности и стоимости, а также создание моделей оценки зрелости требований.

**Верификация требований** – это процесс проверки правильности спецификаций требований на их соответствие, непротиворечивость, полноту и выполнимость, а также на соответствие стандартам. В результате проверки требований делается согласованный выходной документ, устанавливающий полноту и корректность требований к ПО, а также возможность продолжить проектирование ПО.

Из данного стандарта можно выбрать только те процессы, которые более всего подходят для реализации конкретного ПС. Обязательными являются основные процессы, которые присутствуют во всех известных моделях ЖЦ. В зависимости от целей и задач предметной области они могут быть дополнены дополнительными (документирование, обеспечение качества, верификация и валидация и т.п.) и организационными (планирование, управление и др.) процессами этого стандарта.

Процессы, включенные в модель ЖЦ, предназначены для реализации стандартных задач процессов ЖЦ и могут привлекать другие процессы для реализации специализированных задач системы (например, защиты данных). Интерфейсы (входы и выходы) любых двух процессов ЖЦ должны быть минимальными, и каждый из них должен удовлетворять следующим правилам:

- если процесс А вызывается процессом В и только процессом В, то А принадлежит В;
- если функция вызывается более чем одним процессом, то она становится отдельным процессом;
- проверка любой функции в ЖЦ является обязательной.

Если задача требуется более чем одному процессу, то она может стать процессом, используемым однократно или многократно на протяжении жизни конкретной системы. Каждый процесс должен иметь внутреннюю структуру, установленную в соответствии с тем, что он должен выполнять.

Каждый процесс ЖЦ подкрепляется выбранными для реализации задач ПС средствами, методами программирования и методикой их применения и выполнения.

Важную роль при формировании модели ЖЦ имеют организационные аспекты:

- планирование последовательности работ и сроков их исполнения;
- подбор и подготовка ресурсов (людских, программных и технических) для выполнения работ;
- оценка возможностей реализации проекта в заданные сроки, стоимость и ресурсы.

### **Пример разработки ЖЦ ПС с задачами и действиями для процесса тестирования**

Основное назначение процесса тестирования ЖЦ – выполнение задач процесса на основе входов (входные данные для выполнения задач процесса) и выходов при завершении задач, а также ролей и действий исполнителей этих задач.

В соответствии со стандартом ISO/IEC 12207 были выявлены задачи тестирования и распределены по процессам ЖЦ ПС. В результате был получен единый непрерывный процесс тестирования разных ПС, задачами которого являются *подготовка, проведение и оценивание* результатов тестирования, которые распределились по 20 действиям (шагам) процесса разработки. Данный подход к тщательному тестированию ПС целесообразно применять, например, для систем реального времени.

На шаге *подготовки* осуществляется анализ рабочих продуктов процесса разработки ПС (входных для данного шага процесса тестирования) для определения целей, объектов, сценариев и ресурсов тестирования, адекватных шагу тестирования. Результаты выполнения шагов подготовки тестирования должны фиксироваться в планах тестирования.

На шаге *выполнения* осуществляется фиксация результатов выполнения тестов, их сравнение с ожидаемыми результатами, определение текущего состояния *рабочего продукта* ПС и принятие решения о достаточности тестирования.

Каждый шаг процесса *разработки* состоит из набора решаемых задач, распределения по процессам и подпроцессам ЖЦ (рис. 1.6).

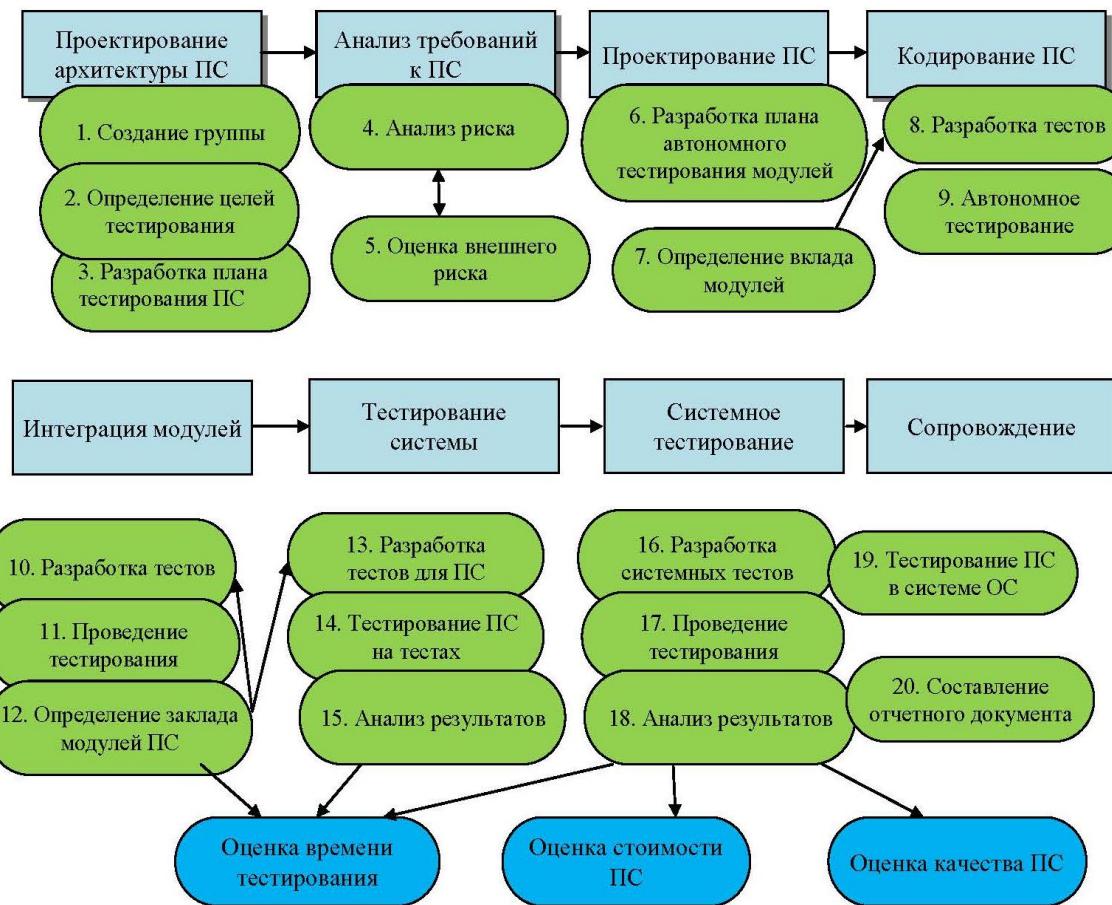


Рис. 1.6. ЖЦ разработки ПС с конкретизированными задачами на подпроцессах тестирования

Шаги процесса и отдельные задачи могут выполняться циклически для разных объектов ПС при их тестировании.

Описание семантики задач и шагов процесса тестирования представлено в табл. 1.1.

Для подключения задач тестирования ко всем процессам ЖЦ проводится:

- распределение обязанностей между участниками процесса с учетом требований относительно их профессиональной подготовки;
- определение стандартов на представление окончательных документов, метрик процесса, критериев начала и завершения задач и перехода к следующему шагу процесса;
- подбор методов тестирования для выбранного класса ПС для проверки правильности выполнения задач тестирования;

– разработка специальных шаблонов документов для документирования процесса тестирования относительно каждого шага процесса тестирования.

При завершении тестирования ПС для определения времени тестирования, стоимости работ учитываются результаты тестирования процесса разработки ПС и оформляется отчетный документ по изготовлению ПС. Оценивание риска отказов проводится на этапе подготовки тестирования и на шагах анализа.

*Таблица 1.1*  
**Состав задач процесса тестирования**

Шаг процесса	Задачи процесса тестирования
1	2
1. Создание группы тестирования	1.1. Определение участников процесса тестирования 1.2. Распределение обязанностей в группе и формирование плана тестирования
2. Анализ риска	2.1. Идентификация рисков 2.2. Упорядочение рисков 2.3. Распределение ресурсов
3. Определение целей тестирования	3.1. Идентификация целей тестирования 3.2. Определение критериев прохождения тестов 3.3. Приведение в порядок целей тестирования по оценкам риска
4. Разработка планов тестирования	4.1. Разработка плана тестирования ПС 4.2. Разработка плана интеграционного тестирования 4.3. Разработка плана автономного тестирования 4.4. Разработка плана комплексного тестирования
5. Разработка тестов	5.1. Проектирование и разработка тестов 5.2. Подготовка тестовых данных 5.3. Проверка тестовых документов
6. Автономное и интеграционное тестирование	6.1. Автономное тестирование модулей и анализ результатов 6.2. Интеграционное тестирование 6.3. Повторное тестирование после устранения дефектов 6.4. Анализ результатов интеграционного тестирования

1	2
7. Тестирование ПС	7.1. Утверждение среды и ресурсов тестирования 7.2. Тестирование ПС 7.3. Повторное тестирование ПС после устранения дефектов 7.4. Анализ результатов завершения тестирования ПС 7.5. Тестирование инсталляции ПС
8. Составление документа по тестированию ПС и подготовка отчета	8.1. Сбор и анализ данных о результатах тестирования 8.2. Подготовка решений и рекомендаций по использованию ПС 8.3. Подготовка итогового документа о результатах тестирования 8.4. Проверка решений и подготовка документа отчета

## **1.3. Типы моделей ЖЦ**

Рассмотренные вопросы послужили источником формирования различных видов моделей ЖЦ, основанных на процессном подходе к разработке программных проектов. К широко используемым типам моделей ЖЦ относятся следующие: каскадная, спиральная, инкрементная, эволюционная, стандартизованная и др.

### **1.3.1. Каскадная модель ЖЦ**

Одной из первых стала применяться *каскадная модель*, в которой каждая работа выполняется один раз и в том порядке, как это представлено в модели (рис. 1.7), т.е. делается предположение, что каждая работа будет выполнена настолько тщательно, что после ее завершения и перехода к следующему этапу возвращения к предыдущему не потребуется.

Разработчик проверяет промежуточный результат разными известными методами верификации и фиксирует его в качестве готового эталона для следующего процесса.

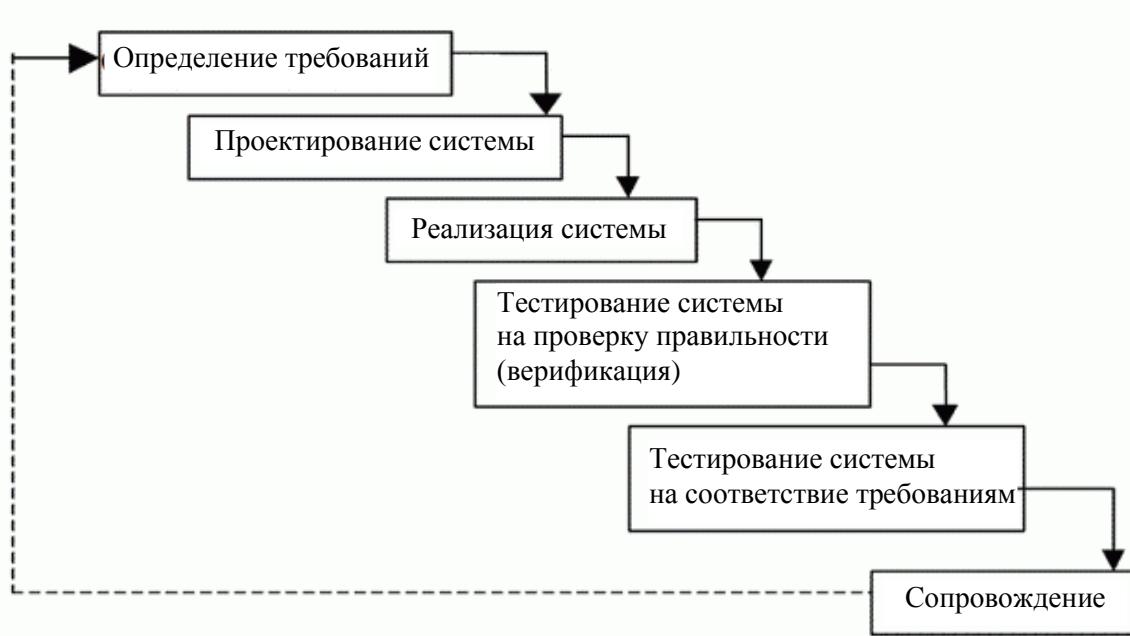


Рис. 1.7. Каскадная модель ЖЦ программных систем

Согласно данной модели ЖЦ работы и задачи процесса разработки обычно выполняются последовательно, как это представлено в схеме. Однако вспомогательные и организационные процессы (контроль требований, управление качеством и др.) обычно выполняются параллельно с процессом разработки. В данной модели возвращение к начальному процессу предусматривается после сопровождения и исправления ошибок.

Особенность такой модели состоит в фиксации последовательных процессов разработки программного продукта. В ее основу положена модель фабрики, где продукт проходит стадии от замысла до производства, затем передается заказчику как готовое изделие, изменение которого не предусмотрено, хотя возможна замена на другое подобное изделие в случае рекламации или некоторых ее деталей, вышедших из строя.

Недостатки этой модели:

- процесс создания ПС не всегда укладывается в такую жесткую форму и последовательность действий;
- не учитываются изменившиеся потребности пользователей, изменения во внешней среде, которые вызовут изменения требований к системе в ходе ее разработки;
- большой разрыв между временем внесения ошибки (например, на этапе проектирования) и временем ее обнаружения (при сопровождении), что приводит к большой переделке ПС.

При применении каскадной модели имеют место следующие факторы риска:

- требования к ПС недостаточно четко сформулированы, либо не учитывают перспективы развития ОС, сред и т.п.;
- большая система, не допускающая компонентной декомпозиции, может вызвать проблемы с размещением ее в памяти или на платформах, не предусмотренных в требованиях;
- внесение быстрых изменений в технологию и в требования может ухудшить процесс разработки отдельных частей системы или системы в целом;
- ограничения на ресурсы (человеческие, программные, технические и др.) в ходе разработки могут сузить отдельные возможности реализации системы.

Полученный продукт может оказаться плохим для применения по причине недопонимания разработчиками требований или функций системы или недостаточно проведенного тестирования.

Преимущества реализации системы с помощью каскадной модели следующие:

- все задачи подсистем и системы реализуются одновременно (ни одна задача не забыта), что способствует установлению стабильных связей и отношений между ними;
- полностью разработанную систему с документацией на нее легче сопровождать, тестировать, фиксировать ошибки и вносить изменения не беспорядочно, а целенаправленно, начиная с требований (например, добавить или заменять некоторые функции), и повторить процесс.

Каскадную модель можно рассматривать как модель ЖЦ, пригодную для создания первой версии ПО с целью проверки реализованных в ней функций. При сопровождении и эксплуатации могут быть обнаружены разного рода ошибки, исправление которых потребует повторного выполнения всех процессов, начиная с уточнения требований.

### **1.3.2. Инкрементная модель ЖЦ**

Первая создаваемая промежуточная версия системы (выпуск 1) реализует часть требований, в последующую версию (выпуск 2) добавляют дополнительные требования, и так до тех пор, пока не будут окончательно выполнены все требования и решены задачи разработки системы (рис. 1.8). Для каждой промежуточной версии на этапах ЖЦ выполняются необходимые процессы, работы и задачи, в

том числе анализ требований и создание новой архитектуры, которые могут быть выполнены одновременно.

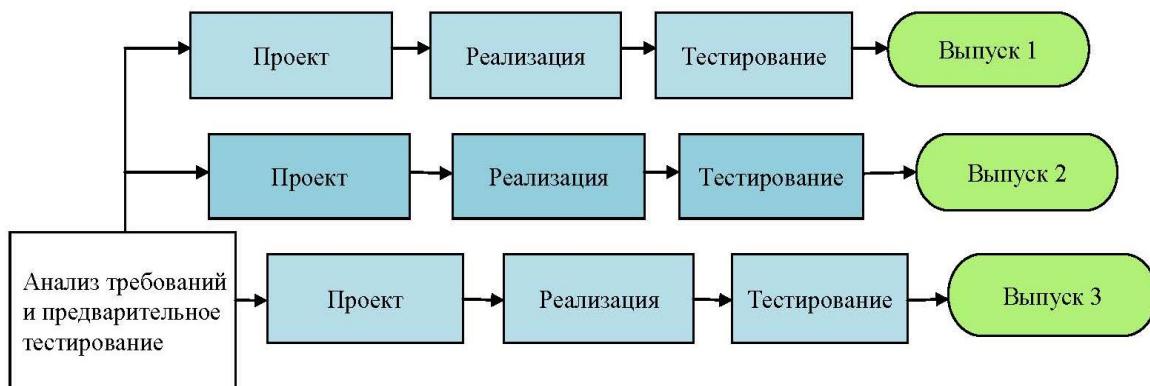


Рис. 1.8. Инкрементная модель ЖЦ

Процессы разработки технического проекта ПС, его программирование и тестирование, сборка и квалификационные испытания ПС выполняются при создании каждой последующей версии.

В соответствии с данной моделью ЖЦ, процессы которой практически такие же, что и в каскадной модели, ориентир делается на разработку некоторой законченной промежуточной версии, а задачи процесса разработки выполняются последовательно или частично параллельно для ряда отдельных промежуточных структур версии.

Работы и задачи процесса разработки следующей версии системы с дополнительными требованиями или функциями могут выполняться неоднократно в той же последовательности для всех промежуточных версий системы. Процессы сопровождения и эксплуатации могут быть реализованы параллельно с процессом разработки версии путем проверки частично реализованных требований в каждой промежуточной версии и так до получения законченного варианта системы. Вспомогательные и организационные процессы ЖЦ обычно выполняются параллельно с процессом разработки версии системы, и к концу разработки будут собраны данные, на основании которых может быть установлен уровень завершенности и качества изготовленной системы.

При применении данной модели необходимо учитывать следующие факторы риска:

– требования составлены с учетом возможности их изменения при реализации продукта;

– все возможности системы требуется реализовать с начала разработки;

– быстрое изменение технологии и требований к системе может привести к нарушению полученной структуры системы;

– ограничения в ресурсном обеспечении (исполнители, финансы) могут привести к затягиванию сроков сдачи системы в эксплуатацию.

Данную модель целесообразно использовать в случаях, когда:

– желательно реализовать некоторые возможности системы быстро за счет создания промежуточной версии продукта;

– система декомпозируется на отдельные составные части, которые можно реализовывать как некоторые самостоятельные промежуточные или готовые продукты;

– возможно увеличение финансирования на разработку отдельных частей системы.

### **1.3.3. Спиральная модель**

Принципиальной особенностью спиральной модели является следующее: прикладное программное обеспечение создается не сразу, как в случае каскадного подхода, а по частям с использованием метода прототипирования (рис. 1.9). Под прототипом понимается действующий программный компонент, реализующий отдельные функции и внешние интерфейсы разрабатываемого программного обеспечения.

Создание прототипов осуществляется в несколько итераций, или витков спирали. Каждая итерация соответствует созданию фрагмента или версии программного обеспечения, на ней уточняются цели и характеристики проекта, оценивается качество полученных результатов, и планируются работы следующей итерации. На каждой итерации производится тщательная оценка риска превышения сроков и стоимости проекта, чтобы определить необходимость выполнения еще одной итерации, степень полноты и точности понимания требований к системе, а также целесообразность прекращения проекта. Спиральная модель избавляет пользователей и разработчиков программного обеспечения от необходимости полного и точного формулирования требований к системе на начальной стадии, поскольку они уточняются на каждой итерации. Таким образом, углубляются и последовательно конкретизируются детали проекта, и в результате выбирается обоснованный вариант, который доводится до реализации.

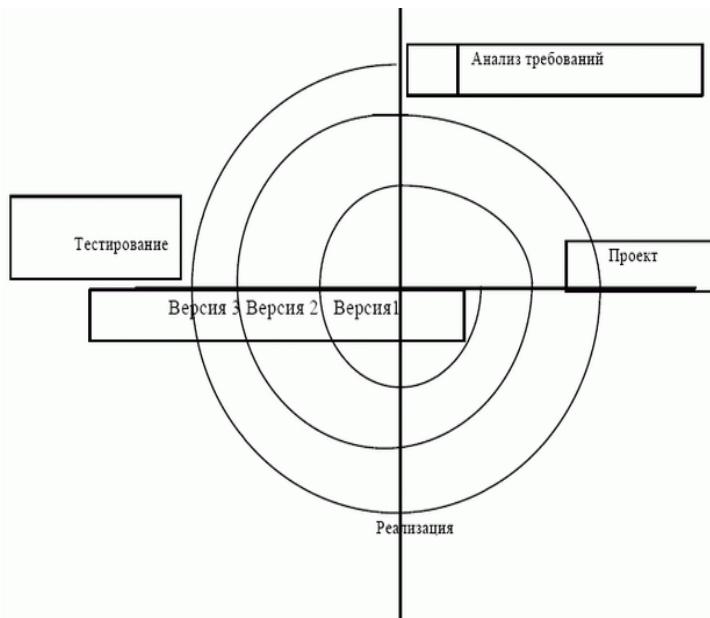


Рис. 1.9. Спиральная модель жизненного цикла программного обеспечения

Спиральная модель не исключает использования каскадного подхода на завершающих стадиях проекта в тех случаях, когда требования к системе оказываются полностью определенными.

Основная проблема спирального цикла – определение момента перехода на следующую стадию. Для ее решения необходимо ввести временные ограничения на каждую из стадий жизненного цикла. Переход осуществляется в соответствии с планом, даже если не

вся запланированная работа закончена. План составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.

Достоинствами спиральной модели являются:

- ускорение разработки (раннее получение результата за счет прототипирования);
- постоянное участие заказчика в процессе разработки;
- разбиение большого объема работы на небольшие части;
- снижение риска (повышение вероятности предсказуемого поведения системы).

Сpirальная модель не исключает использования каскадного подхода на завершающих стадиях проекта в тех случаях, когда требования к системе оказываются полностью определенными.

К недостаткам спиральной модели можно отнести:

- сложность планирования (определения количества и длительности итераций, оценки затрат и рисков);
- сложность применения модели с точки зрения менеджеров и заказчиков (из-за привычки к строгому и детальному планированию);
- напряженный режим работы для разработчиков (при краткосрочных итерациях).

Основная проблема спирального цикла – определение момента перехода на следующую стадию. Для ее решения необходимо ввести временные ограничения на каждую из стадий жизненного цикла.

Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.

При необходимости внесения изменений в систему на каждом витке с целью получения новой версии системы обязательно вносятся изменения в предварительно зафиксированные требования, после чего происходит возврат на предыдущий виток спирали для продолжения реализации новой версии системы с учетом изменений.

#### **1.3.4. Эволюционная модель ЖЦ**

В случае эволюционной модели система разрабатывается в виде последовательности блоков структур (конструкций). В отличие от инкрементной модели ЖЦ подразумевается, что требования устанавливаются частично и уточняются в каждом последующем промежуточном блоке структуры системы.

Использование эволюционной модели предполагает проведение исследования предметной области для изучения потребностей заказчика проекта и анализа возможности применения этой модели для реализации. Модель применяется для разработки несложных и некритических систем, для которых главным требованием является реализация функций системы. При этом требования не могут быть определены сразу и полностью. Тогда разработка системы проводится итерационно путем ее эволюционного развития с получением некоторого варианта системы-прототипа, на котором проверяется реализация требований. Иными словами, такой процесс, по своей сути, является итерационным, с повторяющимися этапами разработки, начиная от измененных требований и до получения готового продукта. В некотором смысле к этому типу модели можно отнести спиральную модель.

Развитием этой модели является модель эволюционного прототипирования в рамках всего ЖЦ разработки (рис. 1.10). В литературе она часто называется моделью быстрой разработки приложений RAD (*Rapid Application Development*). В данной модели приведены действия, которые связаны с анализом ее применимости для конкретного вида системы, а также обследование заказчика для определения потребностей пользователя для разработки плана создания прототипа.

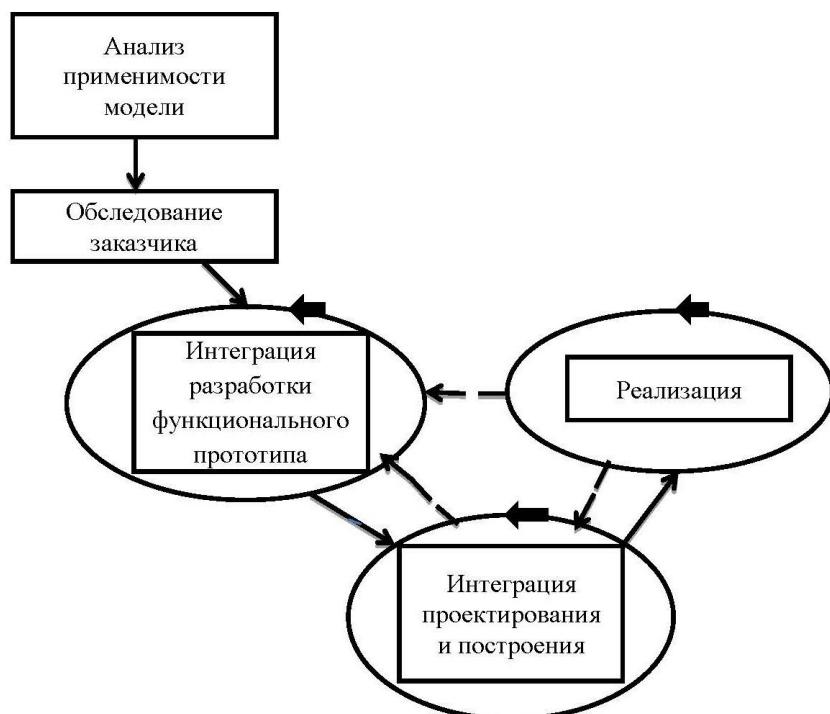


Рис. 1.10. Модель эволюционного прототипирования

В модели есть две главные итерации разработки функционального прототипа, проектирования и реализации системы. Проверяется, удовлетворяет ли она всем функциональным и нефункциональным требованиям. Основной идеей этой модели является моделирование отдельных функций системы в прототипе и постепенное эволюционная его доработка до выполнения всех заданных функциональных требований.

Итераций по получению промежуточных вариантов прототипа может быть несколько, в каждой из которых добавляется функция и повторно моделируется работа прототипа. И так до тех пор, пока не будут промоделированы все функции, заданные в требованиях к системе. Потом выполняется еще итерация – окончательное программирование для получения готовой системы.

Эта модель применяется для систем, в которых наиболее важными являются функциональные возможности и которые необходимо быстро продемонстрировать на CASE-средствах.

Так как промежуточные прототипы системы *соответствуют реализации* некоторых функциональных требований, то их можно проверять и при сопровождении и эксплуатации, т.е. параллельно с процессом разработки очередных прототипов системы. При этом вспомогательные и организационные процессы могут выполняться параллельно с процессом разработки и накапливать сведения по данным количественных и качественных оценок на процессах разработки.

При этом учитываются такие факторы риска:

- реализация всех функций системы одновременно может привести к громоздкости;
- ограниченные человеческие ресурсы заняты разработкой в течение длительного времени.

Преимущества применения данной модели ЖЦ следующие:

- быстрая реализация некоторых функциональных возможностей системы и их апробирование;
- использование промежуточного продукта в следующем прототипе;
- выделение отдельных функциональных частей для реализации их в виде прототипа;
- возможность увеличения финансирования системы;
- обратная связь устанавливается с заказчиком для уточнения функциональных требований;
- упрощение внесения изменений в связи с заменой отдельной функции.

Модель развивается в направлении добавления нефункциональных требований к системе, связанных с защитой и безопасностью данных, несанкционированным доступом к ним и др.

### **1.3.5. Стандартизация модели ЖЦ**

Типичный ЖЦ системы начинается с формулировки идеи или потребности, проходит все процессы разработки, производства, эксплуатации и сопровождения системы. Стандартный ЖЦ состоит из процессов, каждый процесс характеризуется видами деятельности и задачами, которые выполняются на нем. Переход от одного процесса к другому должен быть санкционирован и определены входные и выходные данные.

Модель данного ЖЦ включает в себя процессы:

- определение требований;
- разработка (проектирование, конструирование);
- верификация, валидация, тестирование;
- изготовление;
- эксплуатация;
- сопровождение.

Данной модели соответствуют все виды деятельности, начиная с разработки проекта или концепции программного продукта и заканчивая его изготовлением. Как было сказано выше, стандарт ISO/IEC 12207 объединяет эти виды деятельности в следующие три категории: основные, организационные и вспомогательные процессы, которые и составляют стандартный ЖЦ.

Процессы приобретения, поставки и разработки используются для анализа и определения системных требований и решений верхнего уровня проектирования системы и предварительного определения требований к компонентам системы, включая ПО. Процесс разработки может быть использован для анализа, демонстрации, прототипирования требований и проектных решений.

На этапе проектирования разрабатывается техническое, программное, организационное обеспечение системы, а также проектируются, разрабатываются, интегрируются, тестируются и оцениваются ее компоненты. Результатом этого процесса является система, которая разрабатывалась в соответствии с договором.

Стандарт разработан так, чтобы его можно было применить полностью или частично. Действия и задачи основных процессов отбираются, адаптируются и применяются при разработке или мо-

дификации системы. Процесс разработки может включать одну или более итераций. Результатом являются требования к ПО, проект и реализованный продукт.

Если разрабатываемое ПО – часть системы, то к ней могут применяться все действия процессов разработки, и если эта часть - автономное ПО, то некоторые общие действия на уровне системы могут не использоваться при его разработке.

Во время процесса изготовления система готовится для поставки заказчику и покупателям. Цель процесса – тиражирование (производство) и установка работающей системы у заказчика для сопровождения. Данный процесс заключается в копировании изготовленного продукта и документации на соответствующие носители пользователей. К видам деятельности на процессе относится достижение качества реализации и создания конфигурации (версии) системы. Другие вспомогательные процессы и действия (например, сбор данных о результатах контроля) могут применяться по мере необходимости.

Изготовленная система, начиная с первой ее версии, передается заказчику или продается желающим покупателям. Другие процессы (приобретения, поставки и разработки) могут использоваться при инсталляции и проверке разработанной или модифицированной системы.

Процесс *эксплуатации* включает использование системы ее покупателями. Когда система больше не удовлетворят пользователей, она утилизируется, т.е. удаляется из употребления путем уничтожения кодов, архивов, процедур и т.п.

Во время *сопровождения* система модифицируется вследствие обнаруженных ошибок и недостатков в ее разработке либо по требованиям пользователя, который желает ее адаптировать к новой среде или усовершенствовать отдельные ее функции.

## **Глава 2. Сертификация и оценка процессов создания ПО**

### **2.1. Понятие зрелости процессов создания ПО. Модель оценки зрелости СММ**

Организации, работающие в области разработки, поставки, внедрения и сопровождения ПО и системной интеграции, все больше ощущают, что в основе их конкурентоспособности лежат качество и низкий уровень себестоимости, технологичность производства.

Руководители таких организаций не всегда могут сформировать стратегию совершенствования и развития технологии деятельности своей компании; на рынке труда специалистов необходимой квалификации явно недостаточно. Вместе с тем в области совершенствования технологических процессов разработки и эксплуатации ПО международный опыт долгие годы был недостаточно обобщен и formalизован. Только в начале 1990-х гг. в американский Институт программной инженерии (SEI) сформировал модель технологической зрелости организаций СММ (Capability Maturity Model), определив уровни технологической зрелости и их отличительные черты. В течение десятилетия СММ прошла апробацию в целом ряде организаций, ее эффективность и достоверность проверили заказывающие организации, поставщики ПО, компании, осуществляющие разработку заказного ПО, занимающиеся офшорным программированием.

Сегодня на западе компания-разработчик практически испытывает большие трудности с получением заказов, если она не аттестована по СММ. Заказчики требуют гарантий технологичности компании-исполнителя, гарантий того, что исполнитель не может оказать некачественную услугу.

Оценка технологической зрелости компаний может использоваться:

- заказчиком при отборе лучших исполнителей (например, в тендере);
- компаниями-производителями ПО для систематической оценки состояния своих технологических процессов и выбора направлений их совершенствования;
- компаниями, решившими пройти аттестацию, для оценки «размеров бедствия», т.е. своего текущего состояния;

– аудиторами для определения стандартной процедуры аттестации и проведения необходимых оценок;

– консалтинговыми фирмами, занимающимися реструктуризацией компаний и служб поставщиков информационных технологий и связанных с ними услуг.

По мере повышения технологической зрелости организации процессы создания и сопровождения ПО становятся более стандартизованными и согласованными. При этом формализация процессов позволяет стандартизовать ожидаемые результаты их выполнения и обеспечить предсказуемость результатов выполнения проектов.

Зрелость процессов (software process maturity) – это степень управляемости, контролируемости и эффективности. Повышение технологической зрелости означает потенциальную возможность возрастания устойчивости процессов и указывает на степень эффективности и согласованности использования процессов создания и сопровождения ПО в рамках всей организации.

Реальное использование процессов невозможно без их документирования и доведения до сведения персонала организации, без постоянного контроля и совершенствования их выполнения.

Возможности хорошо продуманных процессов полностью определены. Повышение технологической зрелости процессов означает, что эффективность и качество результатов их выполнения могут постоянно возрастать.

В организациях, достигших технологической зрелости, процессы создания и сопровождения ПО принимают статус стандарта, фиксируются в организационных структурах и определяют производственную тактику и стратегию. Введение их в статус закона влечет за собой необходимость построения необходимой инфраструктуры и создания требуемой корпоративной культуры производства, которые обеспечивают поддержку соответствующих методов, операций и процедур ведения дел даже после того, как из организации уйдут те, кто все это создал.

Модель СММ развивает положения о системе качества организации, формируя критерии ее совершенства – пять уровней технологической зрелости, которые в принципе могут быть достигнуты организацией-разработчиком. Наивысшие – четвертый и пятый уровни – это фактически характеристика организаций, овладевших методами коллективной разработки, в которых процессы создания и сопровождения ПО комплексно автоматизированы и поддерживаются технологически.

Начиная с 1990 г. SEI при поддержке правительственные структур США и организаций-разработчиков ПО постоянно развивает и совершенствует эту модель, учитывая все новейшие достижения в области создания и сопровождения ПО.

СММ представляет собой методический материал, определяющий правила формирования системы управления созданием и сопровождением ПО и методы постепенного и непрерывного повышения культуры производства. Назначение СММ – предоставление организациям-разработчикам необходимых инструкций по выбору стратегии повышения качества процессов путем анализа степени их технологической зрелости и факторов, в наибольшей степени влияющих на качество выпускаемой продукции. Фокусируя внимание на небольшом количестве наиболее критических операций и планомерно повышая эффективность и качество их выполнения, организация таким образом может добиться неуклонного постоянного повышения культуры создания и сопровождения ПО.

СММ – это описательная модель в том смысле, что она описывает существенные (или ключевые) атрибуты, которые определяют, на каком уровне технологической зрелости находится организация. Это нормативная модель в том смысле, что детальное описание методик устанавливает уровень организации, необходимый для выполнения проектов различной сложности и продолжительности по контрактам с правительственными структурами США. СММ не является предписанием, она не предписывает организации, каким образом развиваться. СММ описывает характеристики организации для каждого из уровней технологической зрелости, не давая каких-либо инструкций, как переходить с уровня на уровень. Организации может потребоваться несколько лет для перехода с первого на второй уровень и совсем мало времени для перехода с уровня на уровень далее.

Процесс совершенствования технологии создания ПО отражается в стратегических планах организации, ее структуре, используемых технологиях, общей социальной культуре и системе управления. На каждом уровне устанавливаются требования, при выполнении которых достигается стабилизация наиболее существенных показателей процессов. Выход на каждый уровень технологической зрелости является результатом появления определенного количества компонентов в процессах создания ПО, что в свою очередь приводит к повышению их производительности и качества. На рис. 2.1 показаны пять уровней технологической зрелости СММ.

Надписи на стрелках определяют особенности совершенствования процессов при переходе с уровня на уровень.

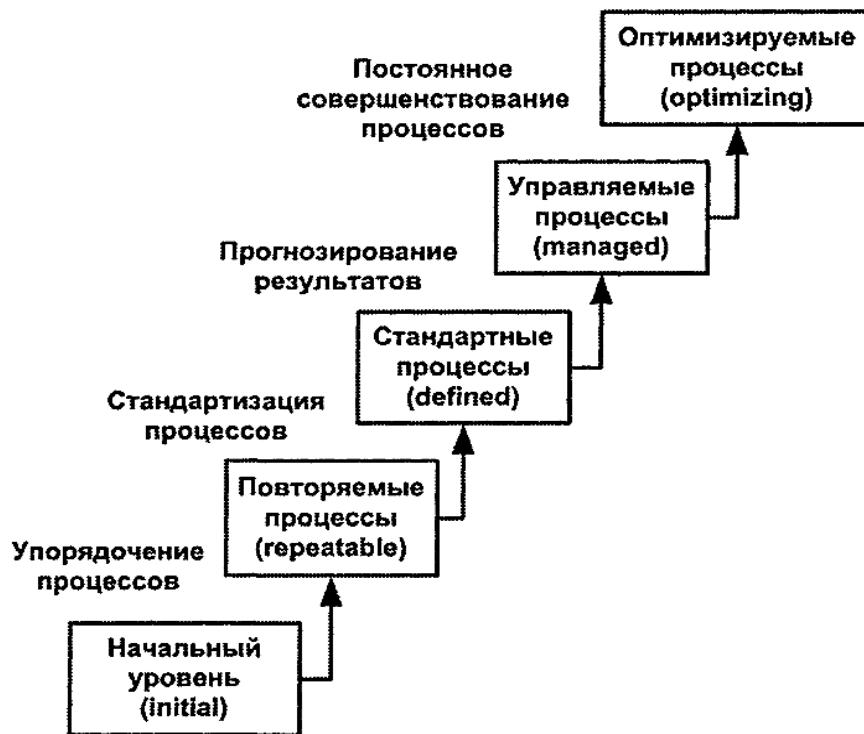


Рис. 2.1. Пять уровней технологической зрелости СММ

Уровни со второго по пятый могут характеризоваться через операции, направленные на стандартизацию и (или) модернизацию процессов создания ПО, и через операции, составляющие сами процессы его создания. При этом первый уровень является как бы базой, фундаментом для сравнительного анализа верхних уровней.

На первом уровне (начальном) основные процессы создания и сопровождения ПО носят случайный характер и выполняются хаотично. Успех выполнения проекта всецело зависит от индивидуальных усилий персонала. На этом уровне, как правило, в организации не существует стабильной среды, необходимой для создания и сопровождения ПО.

На первом уровне процессы являются аморфными («черными ящиками»), и их обозримость весьма ограничена. С самого начала состав и назначение операций практически не определены, что порождает значительные трудности в определении состояния проекта и его продвижения. Требования по выполнению процессов задаются бесконтрольно. Разработка ПО в глазах менеджеров (особенно тех, кто сам не является программистом) иногда выглядит как черная магия.

На втором уровне выполнение требований пользователя и создание ПО контролируемы, поскольку определена база для процессов управления проектом. Процесс создания ПО может рассматриваться как последовательность «черных ящиков», которые можно контролировать в точках перехода из одного «ящика» в другой – зафиксированных этапах. Даже если руководитель не знает, что делается «внутри ящика», точно установлено, что должно получиться в результате выполнения процесса и определены контрольные точки его начала и завершения. Поэтому управление может распознавать проблемы в точках взаимодействия «черных ящиков» и своевременно на них реагировать.

На третьем уровне определена внутренняя структура «черных ящиков», т.е. задачи, из которых состоят процессы. Внутренняя структура представляет собой путь, по которому стандартные процессы в организации применяются в конкретных проектах. Звено управления и исполнители в необходимой степени детализации знают свои роли и ответственность в рамках проекта. Руководство заранее подготовлено к рискам, которые могут возникнуть в процессе выполнения проекта. Так как стандартизованные и документированные процессы становятся «прозрачными» для обозрения, сотрудники, непосредственно не занятые в проекте, могут своевременно получать точные сведения о его текущем состоянии.

На четвертом уровне выполнение процессов жестко привязывается к инструментальным средствам, что дает возможность определения количественных характеристик их трудоемкости и качества выполнения. Руководители, имея объективную базу количественных измерений, получают возможность точного планирования стадий и этапов выполнения проекта, прогнозирования продвижения проекта и могут своевременно и адекватно реагировать на возникающие проблемы. С уменьшением возможных отклонений от заданных сроков, стоимости и качества в процессе выполнения проекта их возможность предвидения результатов постоянно возрастает.

На пятом уровне в целях повышения качества продукции и повышения эффективности ее создания постоянно и планомерно проводится работа по созданию новых усовершенствованных методов и технологий создания ПО. Внимание обращается при этом не только на уже используемые, но и на новые более эффективные процессы и технологии. Руководители могут количественно оценивать влияние и эффективность изменений в технологии создания и сопровождения ПО.

Четвертый и пятый уровни редко встречаются в индустрии ПО. Так, если третьего уровня достигло в мире несколько сотен компаний, то фирм пятого уровня (по информации SEI на 2002 г.) насчитывалось 62, а четвертого – 72. При этом отметим, что заявляют о своем уровне зрелости далеко не все компании. Одни не заинтересованы в афишировании своих организационных технологий, другие выполняют сертификацию просто под давлением заказчика.

Для достижения высших уровней СММ требуется десять и более лет. Но даже третий уровень позволяет смело выходить на международную арену. Для использования СММ компании не надо искать сотрудников с какими-то уникальными способностями, ей достаточно понять общую идею. В описании модели СММ детально указано, что надо делать, чтобы развиваться в соответствии с этой моделью. Следовать регламентированным действиям СММ способен любой менеджер среднего класса.

Последняя версия СММ 1.1 в основном ориентирована на крупные компании, занимающиеся реализацией больших проектов, но она вполне может использоваться и группами из двух-трех человек или отдельными программистами для выполнения небольших проектов (продолжительностью до трех месяцев). В таких случаях модель СММ может сыграть жизненно важную роль, поскольку поступление новых заказов во многом определяется качеством реализации предыдущих проектов. Маленькие группы вполне удовлетворятся вторым уровнем, так как для небольшого проекта отклонение от срока на пару недель непринципиально.

С 2002 г. официально распространяется специальная интеграционная версия СММ1. Это новая разработка SEI, охватывающая все аспекты деятельности компаний: от разработки и выбора подрядчика до обучения, внедрения и сопровождения. Кроме того, модель СММ1 расширена подходами из системной инженерии. В эту модель вошли наработки, сделанные в ходе проектирования версии СММ 2.0 (она не была закончена), основные изменения в которой были направлены на уточнение процессов для компаний четвертого и пятого уровней, что наиболее актуально для крупномасштабных американских проектов.

Модель СММ достаточно весома и важна, однако не стоит применять ее как единственную основу, определяющую весь процесс создания ПО. Она была предназначена в основном для компаний, которые занимаются разработкой ПО для Министерства обороны США.

К недостаткам СММ относятся следующие:

1. Модель сосредоточена исключительно на управлении проектом, а не на процессе создания программного продукта. В модели не учтены такие важные факторы, как использование определенных методов, например прототипирования, формальных и структурных методов, средств статического анализа и т.п.

2. В модели отсутствует анализ рисков и решений, что не позволяет обнаруживать проблемы прежде, чем они окажут воздействие на процесс разработки.

3. Не определена область применения модели, хотя авторы признают, что она является универсальной и подходящей всем организациям. Однако авторы не дают четкого разграничения организаций, которые могут или не могут внедрять СММ в свою деятельность. Небольшие компании находят эту модель слишком бюрократичной. В ответ на эту критику были разработаны стратегии совершенствования технологического процесса для малых организаций.

В качестве альтернативы СММ предлагается обобщенная классификация процессов совершенствования технологической зрелости, которая подходит для большинства организаций и программных проектов. Можно выделить несколько общих типов процессов совершенствования.

1. Неформальный процесс. Не имеет четко выраженной модели совершенствования. Его с успехом может использовать отдельная команда разработчиков. Неформальность процесса не исключает таких формальных действий, как управление конфигурацией, однако при этом сами действия и их взаимосвязи не предопределены заранее.

2. Управляемый процесс. Имеет подготовленную модель, которая управляет процессом совершенствования. Модель определяет действия, их график и взаимосвязи между ними.

3. Методически обоснованный процесс. Подразумевается, что введены в действие определенные методы (например, систематически применяются методы объектно-ориентированного проектирования). Для процессов этого типа будут полезными инструментальные средства поддержки проектирования и анализа процессов (CASE-средства).

4. Процесс непосредственного совершенствования. Имеет поставленную цель совершенствования технологического процесса, для чего существует отдельная строка в бюджете организации и определены нормы и процедуры внедрения нововведений. Частью

такого процесса является количественный анализ процесса совершенствования.

Эту классификацию не назовешь четкой и исчерпывающей – некоторые процессы могут одновременно относиться к нескольким типам. Например, неформальность процесса является выбором команды разработчиков. Эта же команда может выбрать определенную методику разработки, имея при этом все возможности непосредственного совершенствования процесса. Такой процесс подпадает под классификацию неформальный, методически обоснованный, непосредственного совершенствования.

Необходимость приведенной классификации обусловлена тем, что она предоставляет основу для комплексного совершенствования технологии создания ПО и дает возможность организации выбирать разные типы процессов совершенствования. На рис. 2.2 показаны соотношения между разными типами программных систем и процессами совершенствования их разработки.

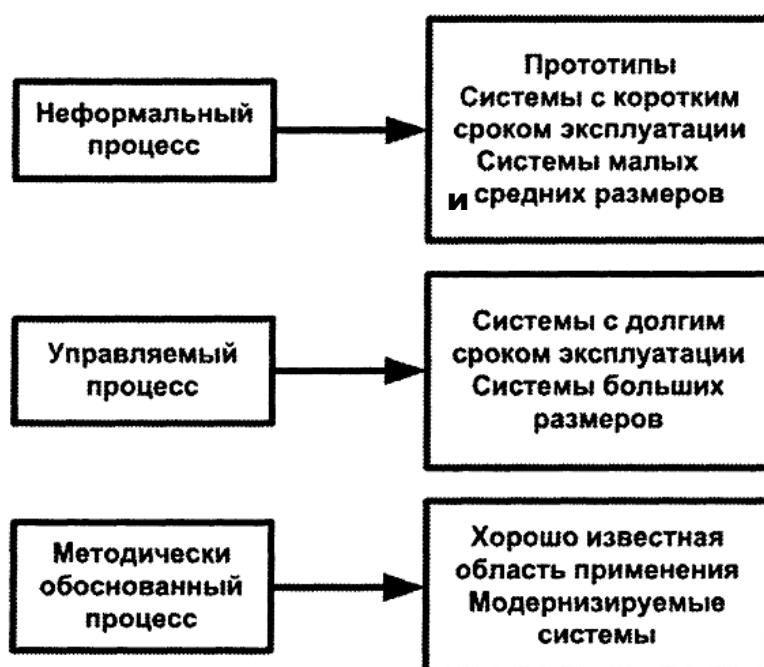


Рис. 2.2. Применимость процессов совершенствования

Многие технологические процессы в настоящее время имеют CASE-средства поддержки, поэтому их можно назвать поддерживающими процессами. Методически обоснованные процессы поддерживаются инструментальными средствами анализа и проектирования.

Главная цель объектного анализа – представить предметную область (ПрО) как множество объектов со свойствами и характери-

стиками, которые достаточны для их определения и идентификации, а также для задания поведения объектов в рамках выбранной системы понятий и абстракций. На произвольном шаге объектного анализа все понятия (сущности) ПрО – суть объекты. Каждый объект – это уникальный элемент, имеет, по крайней мере, одно свойство или характеристику и уникальную идентификацию во множестве объектов.

Предметная область сама является самостоятельным объектом или может быть объектом в составе другой предметной области.

Анализ ПрО проводится с помощью объектно-ориентированных методов и соответствующих стандартов. Конечная цель объектно-ориентированного анализа ПрО – определение объектной модели (ОМ) с помощью выделенных объектов, отношений между ними и их свойствами и характеристиками.

При построении модели ОМ в предметной области также выявляются функциональные задачи, формулируются требования к их проектированию и реализации. Требования, задачи и модель ОМ – необходимые условия построения архитектуры системы для анализируемой ПрО.

## 2.2. Методика SPMN

Признанные мировые лидеры, создающие качественное ПО (их единицы во всем мире), активно используют принцип лучшего практического навыка. Этот принцип ориентирован на улучшение деталей работы и быстрое достижение конечного результата. В нем нет абстрактного «улучшения процесса», а есть конкретные рекомендации, использующие числовые характеристики проекта. Другое преимущество – возможность их немедленного применения в противовес «тяжелой» модели СММ, для сертификации по которой нужны годы труда. Несмотря на определенное число очень успешных результатов внедрения СММ, эта методика не получила массового признания среди небольших фирм в силу сложности и слишком больших усилий, требуемых для ее внедрения. Продолжающиеся неудачи в крупных программных проектах заставили Министерство обороны США сформировать подразделение SPMN (Software Program Managers Network), которое было призвано помочь военным быстро наладить эффективные процессы управления проектами в организациях-разработчиках ПО.

Для SPMN были определены четыре главные цели ее работы:

1. Внедрить в Министерстве обороны лучшие практические навыки создания ПО.

2. Позволить руководителям проектов сфокусировать свои усилия на разработке качественного ПО, а не на следовании должностным инструкциям и формальным методикам, которые только ухудшали состояние проекта.

3. Позволить руководителям проектов использовать лучшие мировые практические навыки с учетом локальной корпоративной культуры.

4. Дать возможность быстро изучить и внедрить эти навыки в свою работу с помощью соответствующих методик обучения и программных систем.

В SPMN было создано три подразделения:

1. Группа оперативных советов определила важнейшие практические навыки (всего их набралось девять). В выявлении лучших навыков участвовали такие общепризнанные эксперты, как Гради Буч, Эдвард Йордон и др. В дополнение к лучшим навыкам они разработали технологию Панели управления ходом программного проекта (Software Project Control Panel), описывающую ключевые индикаторы состояния проекта. Были также определены важнейшие цели типичного проекта, способы их количественной оценки и границы допустимых состояний. Составлен справочник ответов на вопросы, часто задаваемые руководителям проектов, и выделен набор самых плохих практик.

2. Группа периодических обновлений, состоящая из 180 специалистов по программной инженерии, которые обработали 163 методики 56 компаний и выделили 43 лучших практических навыка, расширявших и дополнивших 9 ключевых навыков.

3. Группа управления, контролирующая работу двух предыдущих групп и определяющая способы ее улучшения.

Подход, предложенный отделом SPMN, называется СВР (Critical Best Practices, критически важные практические навыки). Он позволяет тактическими изменениями в работе организации очень быстро (за полтора-два года) примерно на 80 % достичь третьего уровня СММ (на что обычно требуется около десяти лет). При этом подход СВР проверен на сотнях реальных крупных программных проектов.

Рекомендации по применению практических навыков достаточно очевидны. В самом общем виде подход СВР предлагает:

– сфокусироваться на количественных параметрах завершения проекта (дате, бюджете, объеме);

– придумать быстро реализуемую стратегию выполнения проекта;

- измерять продвижение к цели;
- измерять активность разработки.

Результаты работы SPMN показали, что ход выполнения крупных проектов обычно находится на грани хаоса, и существует ряд факторов, от которых зависит, перейдет ли система в неуправляемое состояние. Чтобы правильно управлять проектом, надо придерживаться следующих принципов:

1. Ошибки и логические неувязки надо выявлять как можно раньше и устранять сразу после обнаружения. Между внесением ошибки разработчиком и ее выявлением должно пройти минимальное время (в проектах Министерства обороны США среднее время между внесением ошибки и ее устранением составляло 9 месяцев). Практика почасовой оплаты программистов (имевшая место при выполнении госзаказов в США) совершенно недопустима. Надо также совершенствовать механизмы выявления типичных причин ошибок и способы их устранения.

2. Необходимо планировать работу на основе правильно выбранных показателей. Невозможно реализовать крупный проект, если не подготовить в его рамках максимально подробный план всех видов деятельности с учетом производительности сотрудников, объема проекта, бюджета и других ресурсов.

3. Надо минимизировать неконтролируемые изменения проекта с учетом того, что они вносятся разработчиками на всех этапах, начиная с требований к системе и заканчивая ее пользовательскими интерфейсами.

4. Необходимо эффективно использовать сотрудников. Знания, опыт и мотивация сотрудников – важнейшие факторы успеха. Акцент в управлении проектами должен быть смешен на производительность труда, качество работы, выполнение планов и удовлетворение пользователя. Для этого требуются большие усилия по подготовке профессиональных руководителей проектов и изменения текущих способов их подготовки.

### **Девять лучших навыков, рекомендованных SPMN**

Каждый из описываемых далее навыков полезен сам по себе, но их совместное использование значительно повышает общую эффективность. Немаловажно, что эти навыки могут быть внедрены без дополнительных расходов на оборудование, технологии и персонал.

*Навык 1* – формальное управление рисками. Любой проект по разработке ПО – рискованный. Но отсутствие процедуры управле-

ния рисками в компании – это, пожалуй, самый показательный признак грядущей неудачи проекта. Поэтому необходимо уметь определять риск превышения бюджета и времени выполнения, неверного выбора и возможного отказа оборудования, ошибок программирования и плохого сопровождения. Риск оценивается по вероятности возникновения и его последствиям.

Надо смягчать последствия рисков путем их раннего выявления и максимально ранней ликвидации, профилактической работы и изменения курса проекта «в обход» потенциальных неудач. Неустранимые риски надо отслеживать по параметрам «стоимость последствий риска» и «стоимость устранения риска». Желательно создавать резервные запасы ресурсов для устранения непредвиденных проблем.

В рамках проекта рекомендуется постоянно вести и анализировать списки 10 важнейших рисков; списки неустранимых рисков в наиболее критических точках проекта; отчеты по устранимым, неустранимым и новым рискам; учитывать возможную стоимость последствий рисков в зависимости от имеющегося резерва.

*Навык 2* – соглашения об интерфейсах: пользовательских, внутренних (межмодульных) и внешних (для стыковки с другими компонентами и приложениями).

Интерфейсы программы – это необходимая часть системных требований и ее архитектуры, но руководители проектов часто забывают контролировать соответствие продукта этим соглашениям. Чем позже будут определены соглашения об интерфейсах, тем больше вероятность того, что систему придется заново проектировать, программировать и тестировать.

Для построения пользовательского интерфейса неплохо использовать подход RAD. При этом пользовательский интерфейс (как и все остальные) надо полностью определить, согласовать с заказчиком и утвердить до начала этапов проектирования и разработки. Его описание должно быть включено в системную спецификацию на уровне определения каждого экранного поля, элемента ввода/вывода и средств навигации между формами/окнами/экранами.

Правильность интерфейса проверяет и утверждает только реальный пользователь каждого рабочего места из организаций-заказчика. Для встраиваемой системы готовятся отдельные требования к ее внешнему интерфейсу.

*Навык 3* – формальные проверки проекта. Нередко устранение ошибок начинается, только когда проект переходит к этапу тес-

стирования. Такие этапы были придуманы 30 лет назад для создания небольших по сегодняшним меркам систем, и хотя в тестировании нет ничего плохого, выделять его в отдельный этап методически неверно. Стоимость этапа тестирования может достигать 40–60 % стоимости всего проекта. Эти ненужные усилия можно сократить на порядок, однако немногие руководители знают, как это сделать.

Не менее важны усилия по проверке корректности проекта на этапах формулирования требований, создания архитектуры системы и проектирования. Для выполнения формальных проверок надо использовать небольшие группы сотрудников с четко определенными ролями, привлекая при этом пользователей организации-заказчика. Персонал необходимо постоянно тренировать в умении анализировать код на наличие ошибок. Желательно отслеживать продолжительность усилий по проверкам проекта, число найденных ошибок по отношению к затраченным на их поиск усилиям и среднее время от внесения ошибки до ее устранения.

*Навык 4* – управление проектом на основе метрик. Этот навык нужен для раннего обнаружения потенциальных проблем. Как уже говорилось, стоимость устранения дефекта в проекте увеличивается в геометрической прогрессии по мере роста проекта.

С помощью метрик (числовых характеристик) планируются все задачи проекта. Ход их выполнения надо регулярно отслеживать как минимум по ключевым показателям (стоимость работы и производительность труда). Надо дополнительно контролировать время, затрачиваемое на устранение дефектов, и следить за важнейшими показателями, чтобы по их отклонениям (в любую сторону – например, слишком резкий старт) выявлять потенциальные «подводные камни». Метрики основываются на эмпирических данных, например, на основе результатов анализа схожих по размерам проектов.

*Навык 5* – качество продукта должно контролироваться на глубоком уровне. Проблема реализации мелких деталей программного проекта очень важна при разработке ПО. Иногда мелкое на первый взгляд требование заказчика выливается в глобальную переделку проекта. Если же проект спланирован недостаточно подробно, обсуждать реальное положение дел в ходе его выполнения бессмысленно.

В проекте надо выделить задачи объемом не более 5 % по продолжительности и усилиям, которые могут быть выполнены отдельной группой сотрудников как минимум на 95 %. Каждая подобная задача должна быть ориентирована на выполнение однотипной

работы. Результат выполнения задачи оценивается группой приемки, при этом работа не может быть принята с оговорками: она должна быть выполнена полностью и без ошибок (двоичная система оценки качества «готово/не готово»).

Навык 6 – информация о ходе проекта должна быть общедоступной. Чем больше сотрудников вовлечено в процесс контроля над ходом проекта, тем проще идентифицировать потенциальные проблемы и риски. Надо сделать показатели хода проекта доступными всем сотрудникам и заказчику и организовать канал приема анонимных сообщений о возникающих проблемах. Чаще всего такой канал используется для сведения личных счетов, но лучше получить ложный сигнал, чем не узнать о реальной проблеме. К тому же открытость проекта – это залог снижения числа ложных сообщений.

Навык 7 – чтобы добиться высокого качества, надо отслеживать причины возникновения ошибок. Эффективность работы компании непосредственно зависит от наличия ошибок в проекте. Большинство компаний не контролируют их реальные источники: ошибки программистов, отклонения в графиках выполнения работ, превышение планируемой стоимости, неверно сформулированные требования, неправильно подготовленную документацию и плохо обученный персонал. В каждой фазе проекта ошибки должны отслеживаться формально. Для этого желательно использовать средства конфигурационного управления. Каждый случай обнаружения и устранения ошибки обязательно надо документировать.

Устранять ошибки необходимо по мере их возникновения. При этом учет ошибок удобнее всего вести в нормализованном виде (в расчете на единицу объема, например, на тысячу строк кода). Согласно принципу «снежного кома», с ростом объема проекта норма ошибок в нем увеличивается. Также надо контролировать среднее и максимальное время устранения ошибки и время от внесения до устранения ошибки в течение каждого этапа проекта и на протяжении первого года эксплуатации системы.

Навык 8 – управление конфигурацией. Неконтролируемые изменения в проекте могут быстро ввергнуть его в хаос. Поэтому на практике надо руководствоваться двумя простыми правилами:

– любую информацию, которую использует более чем один сотрудник, надо контролировать с помощью системы управления конфигурацией;

– любую информацию, учитываемую системой качества, надо контролировать с помощью системы управления конфигурацией.

Надо отслеживать все изменения в состоянии создаваемой системы, бюджете и сроках, интерфейсах, контрольных отчетах и т.п. Без систем управления конфигурацией при этом не обойтись, потому что в крупном проекте большие объемы информации меняются очень быстро. Каждый учитываемый объект должен определяться его версией, при этом надо вести архив всех версий всех таких объектов.

Навык 9 – управление персоналом. Главный фактор успеха проекта – качество, опыт и мотивация сотрудников. Не надо забывать, что с помощью различных методик производительность труда программистов можно значительно повысить. К тому же как бы подробно ни документировался проект, некоторые детали его архитектуры всегда хранятся только в головах разработчиков и руководитель проекта должен помогать сотрудникам проявлять индивидуальные творческие способности.

Выявлена высокая степень корреляции между суммами, вкладываемыми в обучение персонала, и общим успехом проекта, поэтому надо постоянно проводить обучение и переподготовку сотрудников. Любые авралы необходимо минимизировать. К авралам (как это на первый взгляд ни парадоксально) обычно приводит работа более 40 часов в неделю, что говорит о неверной организации труда и скрытых ошибках в организационной структуре.

# **Глава 3. Методы структурного и объектного анализа и построения моделей предметных областей**

## **3.1. Визуальное моделирование**

Под моделью ПО в общем случае понимается формализованное описание системы ПО на определенном уровне абстракции. Каждая модель определяет конкретный аспект системы, использует набор диаграмм и документов заданного формата, а также отражает точку зрения и является объектом деятельности различных людей с конкретными интересами, ролями или задачами.

Под термином «моделирование» понимается процесс создания формализованного описания системы в виде совокупности моделей.

*M есть модель системы S, если M может быть использована для получения ответов на вопросы относительно S с точностью A.*

Таким образом, целью модели является получение ответов на некоторую совокупность вопросов. Эти вопросы неявно присутствуют (подразумеваются) в процессе анализа и, следовательно, руководят созданием модели и направляют его. Это означает, что сама модель должна будет дать ответы на эти вопросы с заданной степенью точности. Если модель отвечает не на все вопросы или ее ответы недостаточно точны, то говорят, что модель не достигла своей цели.

Визуальное моделирование – это способ восприятия проблем с помощью зримых абстракций, воспроизводящих понятия и объекты реального мира. Модели служат полезным инструментом анализа проблем, обмена информацией между всеми заинтересованными сторонами (пользователями, специалистами в предметной области, аналитиками, проектировщиками и т.д.), проектирования ПО, а также подготовки документации. Моделирование способствует болееному усвоению требований, улучшению качества системы и повышению степени ее управляемости.

Графические (визуальные) модели представляют собой средства для визуализации, описания, проектирования и документирования архитектуры системы. Под архитектурой понимается набор основных правил, определяющих организацию системы:

- совокупность структурных элементов системы и связей между ними;

- поведение элементов системы в процессе их взаимодействия;
- иерархию подсистем, объединяющих структурные элементы;
- архитектурный стиль (используемые методы и средства описания архитектуры, а также архитектурные образцы).

Архитектура является многомерной, поскольку различные специалисты работают с различными ее аспектами.

Архитектура ПО также предусматривает различные представления, служащие разным целям:

- представлению функциональных возможностей системы;
- отображению логической организации системы;
- описанию физической структуры программных компонентов;
- представлению функциональных возможностей системы;
- отображению логической организации системы;
- описанию физической структуры программных компонентов в среде реализации;
- отображению структуры потоков управления и аспектов параллельной работы;
- описанию физического размещения программных компонентов на базовой платформе.

Архитектурное представление – это упрощенное описание (абстракция) системы с конкретной точки зрения, охватывающее определенный круг интересов и опускающее объекты, несущественные с данной точки зрения. Архитектурные представления концентрируют внимание только на элементах, значимых с точки зрения архитектуры. Архитектурно значимый элемент – это элемент, имеющий значительное влияние на структуру системы и ее производительность, надежность и возможность развития. Это элемент, важный для понимания системы. Например, в состав архитектурно значимых элементов объектно-ориентированной архитектуры входят основные классы предметной области, подсистемы и их интерфейсы, основные процессы или потоки управления.

Поскольку сложность систем повышается, важно располагать хорошими методами моделирования. Хотя имеется много других факторов, от которых зависит успех проекта, но наличие строгого стандарта языка моделирования является весьма существенным.

Язык моделирования включает:

- элементы модели – фундаментальные концепции моделирования и их семантику;
- нотацию (систему обозначений) – визуальное представление элементов моделирования;

- руководство по использованию – правила применения элементов в рамках построения тех или иных типов моделей ПО.

При использовании графических языков моделирования очень важно понимать, чем это поможет, когда дело дойдет до написания кода. Можно привести следующие причины, побуждающие прибегать к их использованию:

– получение общего представления о системе. Графические модели помогают быстро получить общее представление о системе, сказать о том, какого рода абстракции существуют в системе и какие ее части нуждаются в дальнейшем уточнении;

– общение с экспертами организации. Графические модели образуют внешнее представление системы и объясняют, что эта система будет делать;

– изучение методов проектирования. Множество людей отмечает наличие серьезных трудностей, связанных, например, с освоением объектно-ориентированных методов и в первую очередь смену парадигмы. В некоторых случаях переход к объектно-ориентированным методам происходит относительно безболезненно. В других случаях при работе с объектами приходится сталкиваться с рядом препятствий, особенно в части максимального использования их потенциальных возможностей. Графические средства позволяют облегчить решение этой проблемы.

В процессе создания ПО, автоматизирующего деятельность некоторой организации, используются следующие виды моделей:

– модели деятельности организации (или модели бизнес-процессов):

– модели «AS-IS» («как есть»), отражающие существующее на момент обследования положение дел в организации и позволяющие понять, каким образом функционирует данная организация, а также выявить узкие места и сформулировать предложения по улучшению ситуации;

– модели «AS-TO-BE» («как должно быть»), отражающие представление о новых процессах и технологиях работы организации. Переход от модели «AS-IS» к модели «AS-TO-BE» может выполняться двумя способами:

1) совершенствованием существующих технологий на основе оценки их эффективности;

2) радикальным изменением технологий и перепроектированием (реинжинирингом) бизнес-процессов.

– модели проектируемого ПО, которые строятся на основе модели «AS-TO-BE», уточняются и детализируются до необходимого уровня.

### **3.2. Структурные методы анализа и проектирования ПО**

Структурные методы являются строгой дисциплиной системного анализа и проектирования. Методы структурного анализа и проектирования стремятся преодолеть сложность больших систем путем расчленения их на части («черные ящики») и иерархической организации этих «черных ящиков». Выгода в использовании «черных ящиков» заключается в том, что их пользователю не требуется знать, как они работают, необходимо знать лишь их входы и выходы, а также назначение (т.е. функции, которые они выполняет).

Таким образом, первым шагом упрощения сложной системы является ее разбиение на «черные ящики», при этом такое разбиение должно удовлетворять следующим критериям:

- каждый «черный ящик» должен реализовывать единственную функцию системы;
- функция каждого «черного ящика» должна быть легко понимаема независимо от сложности ее реализации;
- связь между «черными ящиками» должна вводиться только при наличии связи между соответствующими функциями системы;
- связи между «черными ящиками» должны быть простыми, насколько это возможно, для обеспечения независимости между ними.

Второй важной идеей, лежащей в основе структурных методов, является идея иерархии. Для понимания сложной системы недостаточно разбиения ее на части, необходимо эти части организовать определенным образом, а именно в виде иерархических структур.

Структурным анализом принято называть метод исследования системы, который начинается с ее общего обзора, затем детализируется, приобретая иерархическую структуру со все большим числом уровней. Для таких методов характерно:

- разбиение системы на уровни абстракции с ограничением числа элементов на каждом из уровней (обычно от 3 до 6–7);
- ограниченный контекст, включающий лишь существенные на каждом уровне детали;
- использование строгих формальных правил записи;
- последовательное приближение к конечному результату.

В структурном анализе основным методом разбиения на уровни абстракции является функциональная декомпозиция, заключающаяся в декомпозиции (разбиении) системы на функциональные подсистемы, которые в свою очередь делятся на подфункции, те – на задачи и так далее до конкретных процедур.

При этом система сохраняет целостное представление, в котором все составляющие компоненты взаимоувязаны. При разработке системы «снизу вверх» отдельных задач ко всей системе целостность теряется, возникают проблемы при описании информационного взаимодействия отдельных компонентов.

Все наиболее распространенные методы структурного подхода базируются на ряде общих принципов. Базовыми принципами являются:

- принцип «разделяй и властвуй» – принцип решения проблем путем разбиения их на множество меньших независимых задач, легких для понимания и решения;
- принцип иерархического упорядочения – принцип организации составных частей системы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне.

Выделение двух базовых принципов не означает, что остальные принципы являются второстепенными, поскольку игнорирование любого из них может привести к нежелательным последствиям (вплоть до неудачного завершения проекта). Основными из этих принципов являются:

- принцип абстрагирования – выделение существенных элементов системы и отвлечение от несущественных;
- принцип непротиворечивости – обоснованность и важность элементов системы;
- принцип структурирования данных – данные должны быть структурированы и иерархически организованы.

В структурном анализе и проектировании используются различные модели, описывающие:

- 1) функциональную структуру системы;
- 2) последовательность выполняемых действий;
- 3) передачу информации между функциональными процессами;
- 4) отношения между данными.

Наиболее распространенными моделями первых трех групп являются:

- функциональная модель SADT (Structured Analysis and Design Technique);

- модель IDEF3;
- DFD (Data Flow Diagrams) – диаграммы потоков данных.

Модель «сущность – связь» (ERM – Entity-Relationship Model), описывающая отношения между данными, традиционно используется в структурном анализе и проектировании, однако, по существу, представляет собой подмножество объектной модели предметной области.

### **3.2.1. Метод функционального моделирования SADT**

Метод SADT представляет собой совокупность правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области. Функциональная модель SADT отображает функциональную структуру объекта, т.е. производимые им действия и связи между этими действиями.

Основные элементы этого метода основываются на следующих концепциях:

- графическое представление блочного моделирования. Графика блоков и дуг SADT-диаграммы отображает функцию в виде блока, а интерфейсы входа/выхода представляются дугами, соответственно входящими в блок и выходящими из него. Взаимодействие блоков друг с другом описывается посредством интерфейсных дуг, выражающих «ограничения», которые в свою очередь определяют, когда и каким образом функции выполняются и управляются;
- строгость и точность.

Правила SADT включают:

- ограничение количества блоков на каждом уровне декомпозиции (правило 3–6 блоков – ограничение мощности краткосрочной памяти человека),
- связность диаграмм (номера блоков), уникальность меток и наименований (отсутствие повторяющихся имен), синтаксические правила для графики (блоков и дуг), разделение входов и управлений (правило определения роли данных);
- отделение организации от функции, т.е. исключение влияния административной структуры организации на функциональную модель.

Метод SADT может использоваться для моделирования самых разнообразных процессов и систем. В существующих системах метод SADT может быть использован для анализа функций, выполняемых системой, и указания механизмов, посредством которых они осуществляются.

Результатом применения метода SADT является модель, которая состоит из диаграмм, фрагментов текстов и гlosсария, имеющих ссылки друг на друга. Диаграммы – главные компоненты модели, все функции организации и интерфейсы на них представлены как блоки и дуги соответственно. Место соединения дуги с блоком определяет тип интерфейса. Управляющая информация входит в блок сверху, в то время как входная информация, которая подвергается обработке, показана с левой стороны блока, а результаты (выход) – с правой стороны. Механизм (человек или автоматизированная система), который осуществляет операцию, представляется дугой, входящей в блок снизу (рис. 3.1).

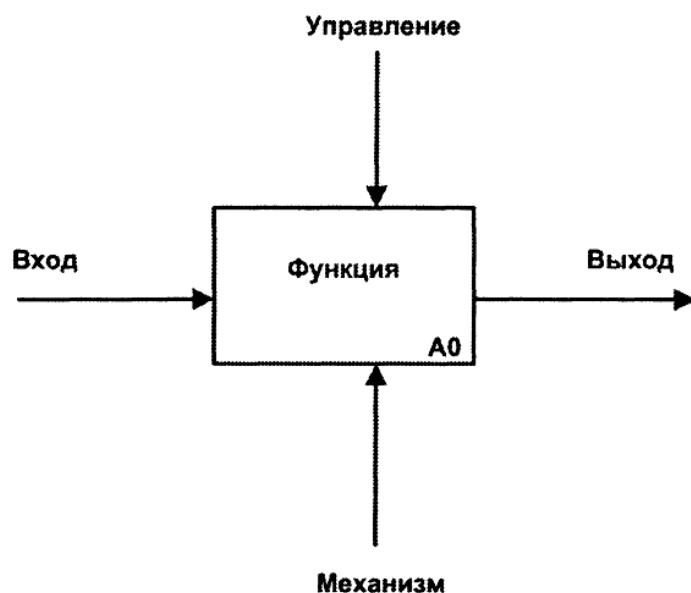


Рис. 3.1. Функциональный блок и интерфейсные дуги

Построение SADT-модели заключается в выполнении следующих действий:

- сбор информации об объекте, определение его границ;
- определение цели и точки зрения модели;
- построение, обобщение и декомпозиция диаграмм;
- критическая оценка, рецензирование и комментирование.

Построение диаграмм начинается с представления всей системы в виде простейшего компонента – одного блока и дуг, изображающих интерфейсы с функциями вне системы (рис. 3.2). Поскольку единственный блок отражает систему как единое целое, имя, указанное в блоке, является общим. Это верно и для интерфейсных дуг – они также соответствуют полному набору внешних интерфейсов системы в целом.

Затем блок, который представляет систему в качестве единого модуля, детализируется на другой диаграмме с помощью нескольких блоков, соединенных интерфейсными дугами. Эти блоки определяют основные подфункции исходной функции. Данная декомпозиция выявляет полный набор подфункций, каждая из которых показана как блок, границы которого определены интерфейсными дугами. Каждая из этих подфункций может быть декомпозирована подобным образом в целях большей детализации.

На SADT-диаграммах не указаны явно ни последовательность (рис. 3.3), ни время. Обратные связи (рис. 3.4), итерации, продолжающиеся процессы и перекрывающиеся (по времени) функции (рис. 3.5) могут быть изображены с помощью дуг. Обратные связи могут выступать в виде комментариев, замечаний, исправлений и т.д.

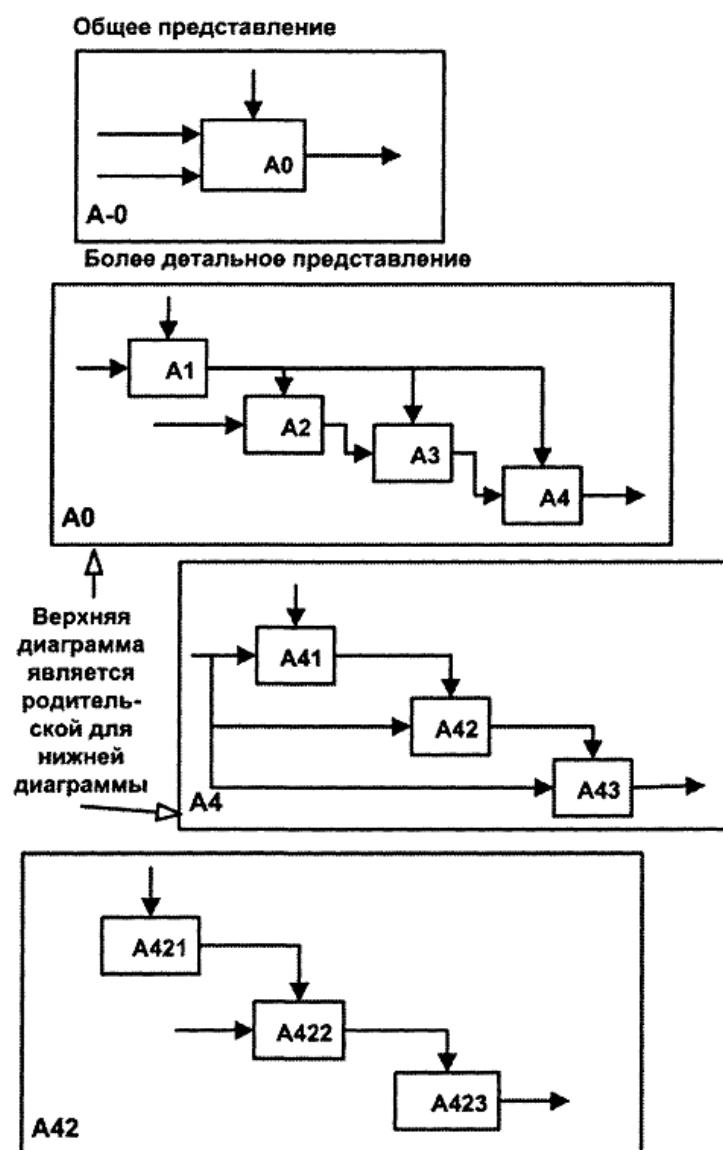


Рис. 3.2. Структура SADT-модели. Декомпозиция диаграмм

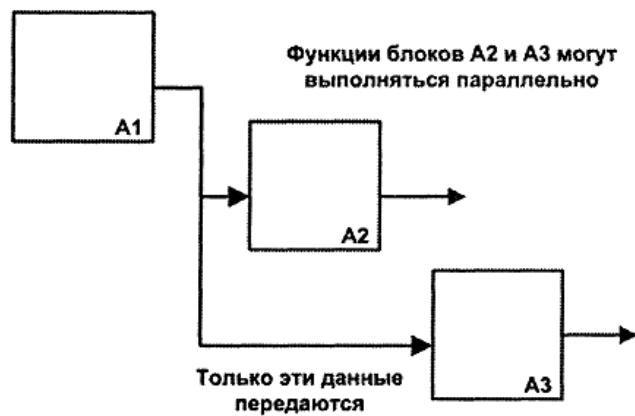


Рис. 3.3. Одновременное выполнение функций

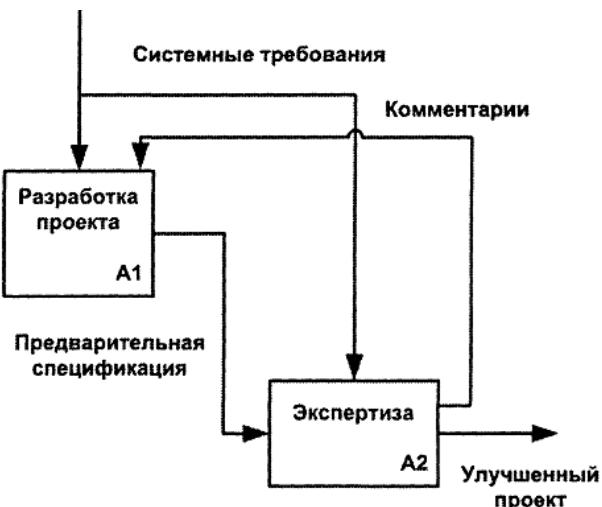


Рис. 3.4. Пример обратной связи

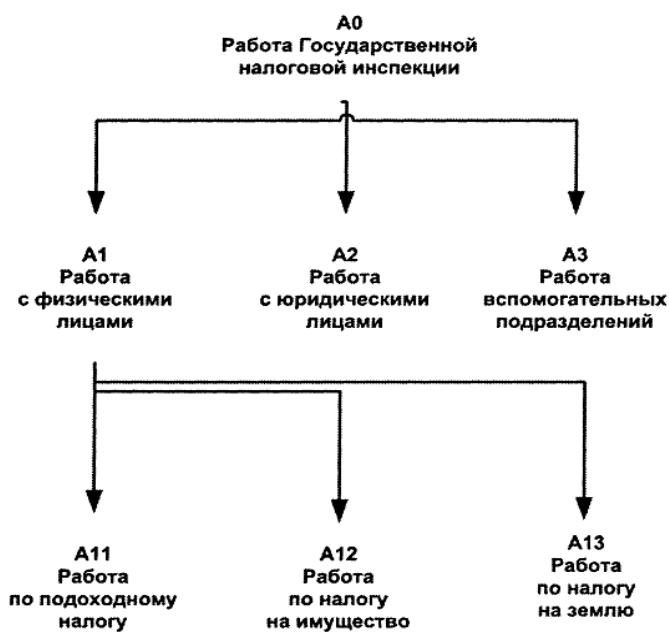


Рис. 3.5. Иерархия диаграмм

Рекомендуется прекращать моделирование, когда уровень детализации модели удовлетворяет ее цели, если:

- блок содержит достаточное количество деталей;
- необходимо изменить уровень абстракции, чтобы достичь большей детализации блока;
- необходимо изменить точку зрения, чтобы детализировать блок;
- блок похож на другой блок этой модели или на блок другой модели;
- блок представляет тривиальную функцию.

Одним из важных моментов при моделировании с помощью метода SADT является точная согласованность типов связей между функциями. Различают, по крайней мере, связи семи типов (в порядке возрастания их относительной значимости):

- случайная;
- логическая;
- временная;
- процедурная;
- коммуникационная;
- последовательная;
- функциональная.

Случайная связь показывает, что конкретная связь между функциями незначительна или полностью отсутствует. Это относится к ситуации, когда имена данных на SADT-дугах в одной диаграмме имеют слабую связь друг с другом

Логическая связь – данные и функции собираются вместе благодаря тому, что они попадают в общий класс или набор элементов, но необходимых функциональных отношений между ними не обнаруживается.

Временная связь представляет функции, связанные во времени, когда данные используются одновременно или функции включаются параллельно, а не последовательно.

Процедурная связь – функции сгруппированы вместе благодаря тому, что они выполняются в течение одной и той же части цикла или процесса.

Коммуникационная связь – функции группируются благодаря тому, что они используют одни и те же входные данные и/или производят одни и те же выходные данные

Последовательная связь – выход одной функции служит входными данными для следующей функции. Связь между элементами

на диаграмме является более тесной, чем в рассмотренных выше случаях, поскольку моделируются причинно-следственные зависимости.

Функциональная связь – все элементы функции влияют на выполнение одной и только одной функции. Диаграмма, являющаяся чисто функциональной, не содержит чужеродных элементов, относящихся к последовательному или более слабому типу связи.

В математических терминах необходимое условие для простейшего типа функциональной связи имеет следующий вид:

$$C = g(B) = g(f(A)).$$

В табл. 3.1 представлены все типы связей, рассмотренные выше.

*Таблица 3.1*

#### **Описание типов связей**

Уровень значимости	Тип связи	Характеристика типа связи	
		Для функций	Для данных
0	Случайная	Случайная	Случайная
1	Логическая	Функции одного и того же типа или множества	Данные одного и того же типа или множества
2	Временная	Функции одного и того же периода времени	Данные, используемые в каком-либо временном интервале
3	Процедурная	Функции, работающие в одной и той же фазе или итерации	Данные, используемые в одной и той же фазе или итерации
4	Коммуникационная	Функции, использующие одни те же данные	Данные, на которые воздействует одна и та же деятельность
5	Последовательная	Функции, выполняющие последовательные преобразования одних и тех же данных	Данные, преобразуемые последовательными функциями
6	Функциональная	Функции, объединяемые для выполнения одной функции	Данные, связанные с одной функцией

**Метод SSADM** базируется на таких структурных диаграммах, как последовательность, выбор и итерация. Моделируемый объект задается последовательностью групп, операторами выбора из группы и циклическим выполнением отдельных элементов (рис. 3.6).



Рис. 3.6. Структура модели

Базовая диаграмма – иерархическая и включает в себя список компонентов описываемого объекта; идентифицированные группы выбранных и повторяемых компонентов, а также последовательно используемых компонентов.

Данный метод представлен моделью ЖЦ со следующими этапами разработки программного проекта (рис. 3.7):

- стратегическое проектирование и изучение возможности выполнения проекта;
- детальное обследование предметной области, включающее в себя анализ и спецификацию требований;
- логическое проектирование и спецификация системы;
- физическое проектирование структур данных в соответствии с выбранной структурой БД (иерархической, сетевой и др.);
- конструирование и тестирование системы.

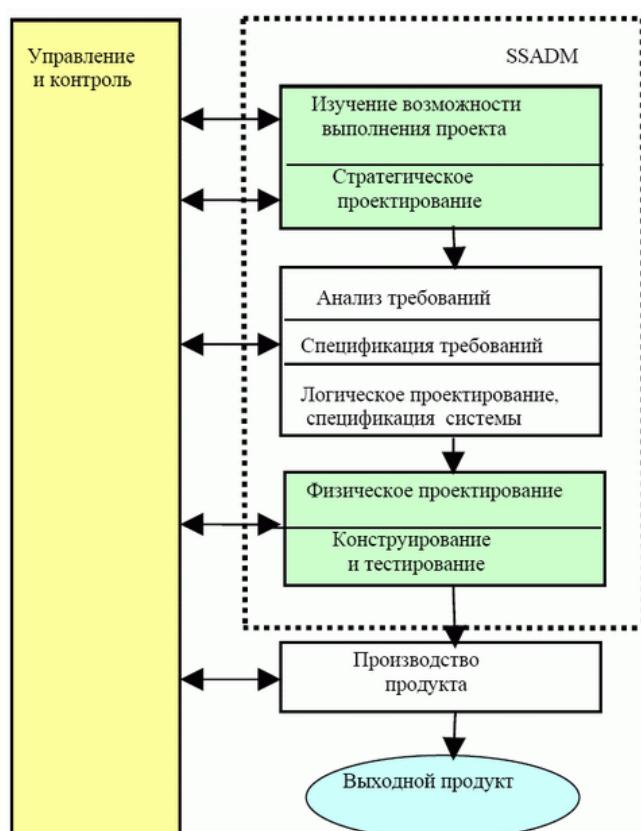


Рис. 3.7. Жизненный цикл SSADM

Детальное обследование предметной области проводится для того, чтобы изучить ее особенности, рассмотреть потребности и предложения заказчика, провести анализ требований из разных документов, обозначить их и согласовать с заказчиком.

Цель стратегического проектирования – определение области действия проекта, анализ информационных потоков, формирование общего представления об архитектуре системы, затратах на разработку и подтверждение возможности дальнейшей реализации проекта. Результат есть спецификация требований, которая применяется при разработке логической структуры системы.

Логическое проектирование – это определение функций, диалога, метода построения и обновления БД. В логической модели отображаются входные и выходных данные, прохождение запросов и установка связей между сущностями и событиями.

Физическое проектирование – это определение типа СУБД и представления данных в ней с учетом спецификации логической модели данных, ограничений на память и времени обработки, а также определение механизмов доступа, размера логической БД, связей между элементами системы. Результат – создание документа, включающего в себя:

- спецификацию функций и способов их реализации, описание процедурных, непроцедурных компонентов и интерфейсов системы;
- определение логических и физических групп данных с учетом структуры БД, ограничений на оборудование и положений стандартов на разработку;
- определение событий, которые обрабатываются как единое целое, и выдача сообщений о завершении обработки и др.

Конструирование – это программирование элементов системы и их тестирование на наборах данных, которые подбираются на ранних этапах ЖЦ разработки системы.

Проектирование системы является управляемым и контролируемым. Создается сетевой график, учитывающий работы по разработке системы, затраты и сроки. Слежение и контроль выполнения плана проводит организационный отдел. Проект системы задается структурной моделью, в которой содержатся работы и взаимосвязи между ними и их исполнителями, а потоки проектных документов между этапами отображаются в сетевом графике. Результаты каждого из этапов ЖЦ контролируются и передаются на следующий этап в виде, удобном для дальнейшей реализации другими исполнителями.

### 3.2.2. Метод моделирования процессов IDEF3

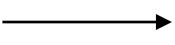
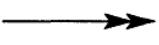
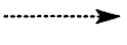
Основой модели IDEF3 служит сценарий процесса, который выделяет последовательность действий и подпроцессов анализируемой системы.

Как и в методе IDEFO (SADT), основной единицей модели IDEF3 является диаграмма. Другой важный компонент модели – действие, или в терминах IDEF3 «единица работы».

Существенные взаимоотношения между действиями изображаются с помощью связей. Все связи в IDEF3 являются односторонними, и хотя стрелка может начинаться или заканчиваться на любой стороне блока, обозначающего действие, диаграммы IDEF3 обычно организуются слева направо таким образом, что стрелки начинаются на правой и заканчиваются на левой стороне блоков. В табл. 3.2 приведены три возможных типа связей.

Таблица 3.2

Типы связей 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 (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» показывает, что исходное действие должно полностью завершиться, прежде чем начнется выполнение конечного действия.

Связь типа «объектный поток» используется в том случае, когда некоторый объект, являющийся результатом выполнения исходного действия, необходим для выполнения конечного действия. Обозначение такой связи отличается от связи временного предш-

ствования двойной стрелкой. Наименования потоковых связей должны четко идентифицировать объект, который передается с их помощью. Временная семантика объектных связей аналогична связям предшествования и означает, что порождающее объектную связь исходное действие должно завершиться, прежде чем конечное действие может начать выполняться.

Связь типа «нечеткое отношение» используется для выделения отношений между действиями, которые невозможно описать с использованием связей предшествования 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.

Завершение одного действия может инициировать начало выполнения сразу нескольких других действий или, наоборот, определенное действие может требовать завершения нескольких других действий до начала своего выполнения. Соединения разбивают или соединяют внутренние потоки и используются для изображения 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 другого действия.

В табл. 3.3 описаны три типа соединений.

*Таблица 3.3*

#### **Типы соединений**

Графическое обозначение	Название	Вид	Правила инициации
1	2	3	4
&	Соединение «и»	Разворачивающее	Каждое конечное действие обязательно инициируется
		Сворачивающее	Каждое исходное действие обязательно должно завершиться
X	Соединение «исключающее «или»»	Разворачивающее	Одно и только одно конечное действие инициируется
		Сворачивающее	Одно и только одно исходное действие должно завершиться

1	2	3	4
0	Соединение «или»	Разворачивающее	Одно или несколько конечных действий инициируются
		Сворачивающее	Одно или несколько исходных действий должны завершиться

Соединения «и» инициируют выполнение конечных действий. Все действия, присоединенные к сворачивающему соединению «и», должны завершиться, прежде чем начнется выполнение следующего действия (рис. 3.8).

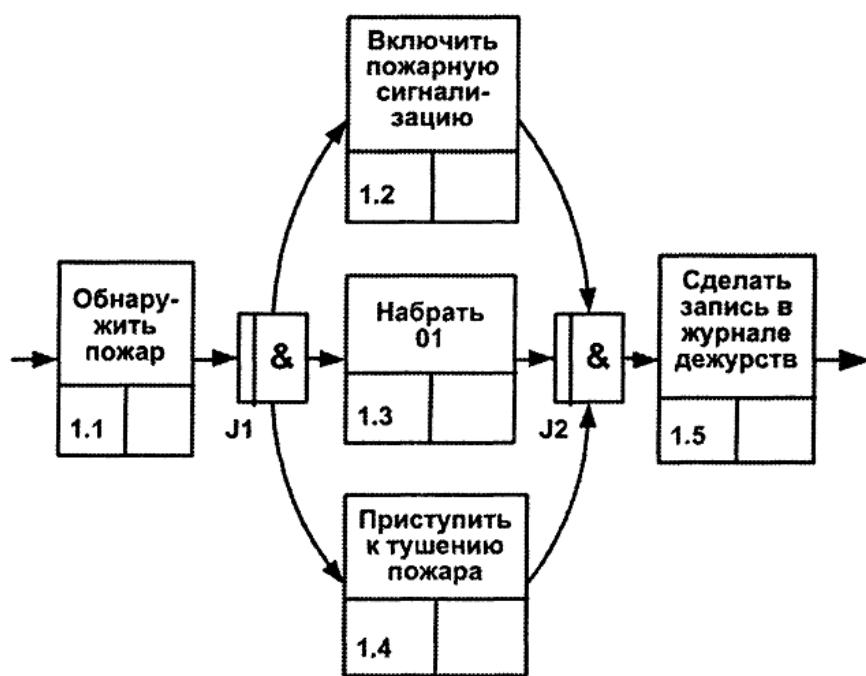


Рис. 3.8. Соединение «и»

Соединение «исключающее или» означает, что вне зависимости от количества действий, связанных со сворачивающим или разворачивающим соединением, инициировано будет только одно из них, и поэтому оно будет завершено перед тем, как любое действие, следующее за сворачивающим соединением, сможет начаться. Если правила активации соединения известны, то они обязательно должны быть документированы либо в его описании, либо пометкой стрелок, исходящих из разворачивающего соединения (рис. 3.9).

Соединение «или» предназначено 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 (рис. 3.10).

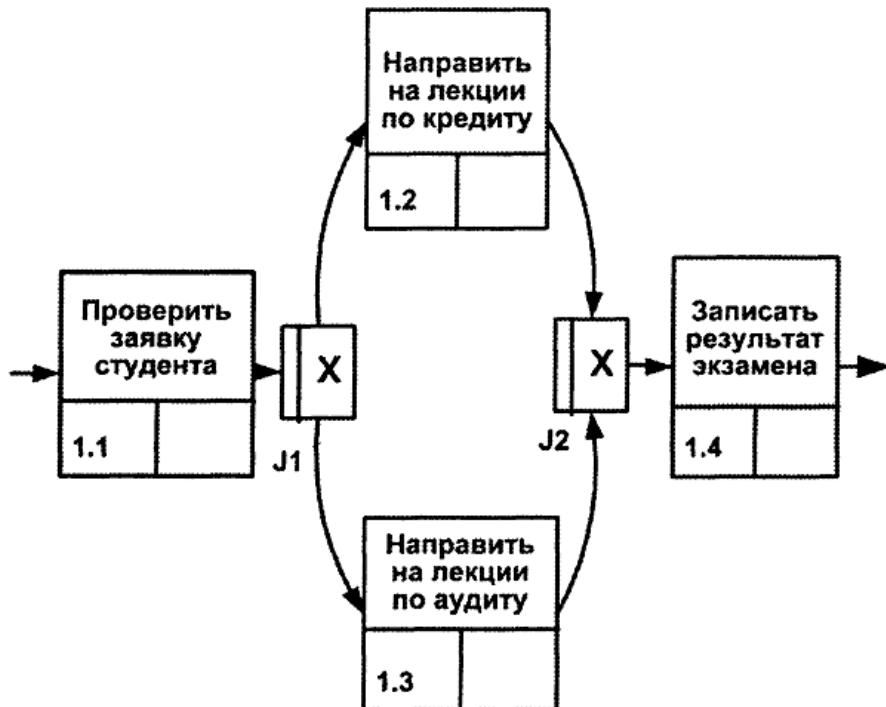


Рис. 3.9. Соединение «исключающее или»

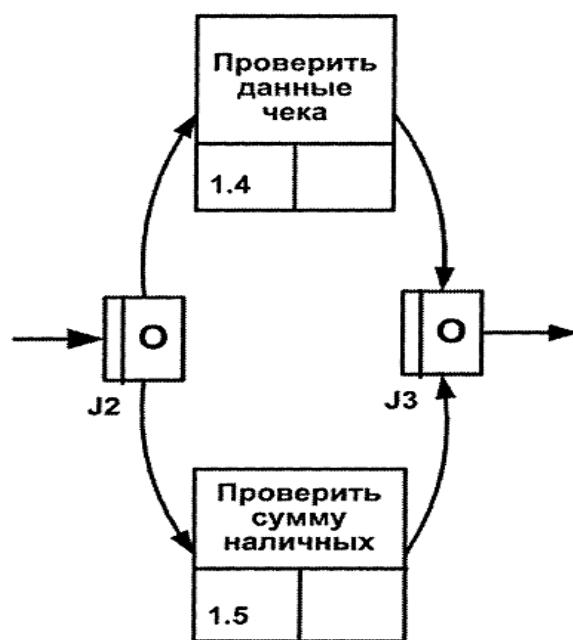


Рис. 3.10. Соединение «или»

### **3.3. Моделирование потоков данных**

Диаграммы потоков данных (Data Flow Diagrams) представляют собой иерархию функциональных процессов, связанных потоками данных. Цель такого представления – продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также выявить отношения между этими процессами. В соответствии с данным методом модель системы определяется как иерархия диаграмм потоков данных, описывающих асинхронный процесс преобразования информации от ее ввода в систему до выдачи потребителю. Источники информации (внешние сущности) порождают информационные потоки (потоки данных), переносящие информацию к подсистемам или процессам. Те в свою очередь преобразуют информацию и порождают новые потоки, которые переносят информацию к другим процессам или подсистемам, накопителям данных или внешним сущностям – потребителям информации.

Диаграммы верхних уровней иерархии (контекстные диаграммы) определяют основные процессы или подсистемы с внешними входами и выходами. Они детализируются с помощью диаграмм нижнего уровня. Такая декомпозиция продолжается, создавая многоуровневую иерархию диаграмм, до тех пор, пока не будет достигнут уровень декомпозиции, на котором детализировать процессы далее не имеет смысла.

Основными компонентами диаграмм потоков данных являются:

- внешние сущности;
- системы и подсистемы;
- процессы;
- накопители данных;
- потоки данных.

Внешняя сущность (рис. 3.11) представляет собой материальный объект или физическое лицо, источник или приемник информации (например, заказчики, персонал, поставщики, клиенты, склад). Определение некоторого объекта или системы в качестве внешней сущности указывает на то, что она находится за пределами границ анализируемой системы. В процессе анализа некоторые внешние сущности могут быть перенесены внутрь диаграммы анализируемой системы, если это необходимо, или наоборот, часть процессов может быть вынесена за пределы диаграммы и представлена как внешняя сущность.

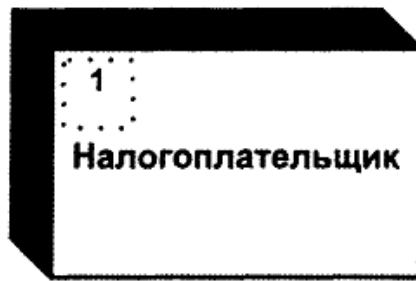


Рис. 3.11. Графическое представление внешней сущности

При построении модели сложной системы она может быть представлена в самом общем виде на так называемой контекстной диаграмме в виде одной системы как единого целого, либо может быть декомпозирована на ряд подсистем.

Процесс представляет собой преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом (рис. 3.12). Физически процесс может быть реализован различными способами: это может быть подразделение организации (отдел), выполняющее обработку входных документов и выпуск отчетов; программа; аппаратно-реализованное логическое устройство и т.д.



Рис. 3.12. Графическое изображение процесса

Информация в поле физической реализации показывает, какое подразделение организации, программа или аппаратное устройство выполняет данный процесс.

Накопитель данных – это абстрактное устройство для хранения информации, которую можно в любой момент поместить в накопитель и через некоторое время извлечь, причем способы помещения и извлечения могут быть любыми.

Главная цель построения иерархии DFD заключается в том, чтобы сделать описание системы ясным и понятным на каждом уровне детализации, а также разбить его на части с точно определенными отношениями между ними. Для достижения этого целесообразно пользоваться следующими рекомендациями:

- размещать на каждой диаграмме от 3 до 6–7 процессов (аналогично SADT). Верхняя граница соответствует человеческим воз-

можностям одновременного восприятия и понимания структуры сложной системы с множеством внутренних связей, нижняя граница выбрана по соображениям здравого смысла: нет необходимости детализировать процесс диаграммой, содержащей всего один или два процесса;

- не загромождать диаграммы несущественными на данном уровне деталями;
- декомпозицию потоков данных осуществлять параллельно с декомпозицией процессов. Эти две работы должны выполняться одновременно, а не одна после завершения другой;
- выбирать ясные, отражающие суть дела, имена процессов и потоков, при этом стараться не использовать аббревиатуры.

Первым шагом при построении иерархии DFD является построение контекстных диаграмм. Обычно при проектировании относительно простых систем строится единственная контекстная диаграмма со звездообразной топологией, в центре которой находится так называемый главный процесс, соединенный с приемниками и источниками информации, посредством которых с системой взаимодействуют пользователи и другие внешние системы. Перед построением контекстной DFD необходимо проанализировать внешние события (внешние сущности), оказывающие влияние на функционирование системы. Количество потоков на контекстной диаграмме должно быть по возможности небольшим, поскольку каждый из них может быть в дальнейшем разбит на несколько потоков на следующих уровнях диаграммы.

Для проверки контекстной диаграммы можно составить список событий, который должен состоять из описаний действий внешних сущностей (событий) и соответствующих реакций системы на них. Каждое событие должно соответствовать одному или более потокам данных: входные потоки интерпретируются как воздействия, а выходные потоки – как реакции системы на входные потоки.

Если для сложной системы ограничиться единственной контекстной диаграммой, то она будет содержать слишком большое количество источников и приемников информации, которые трудно расположить на листе бумаги нормального формата, и, кроме того, единственный главный процесс не раскрывает структуры такой системы. Признаками сложности (в смысле контекста) могут быть: наличие большого количества внешних сущностей (десять и более); распределенная природа системы, многофункциональность системы с уже сложившейся или выявленной группировкой функций в отдельные подсистемы.

Для сложных систем строится иерархия контекстных диаграмм. При этом контекстная диаграмма верхнего уровня содержит не единственный главный процесс, а набор подсистем, соединенных потоками данных. Контекстные диаграммы следующего уровня детализируют контекст и структуру подсистем.

Иерархия контекстных диаграмм определяет взаимодействие основных функциональных подсистем между собой, с внешними входными и выходными потоками данных и внешними объектами (источниками и приемниками информации), с которыми взаимодействует система.

Разработка контекстных диаграмм решает проблему строгого определения функциональной структуры системы на самой ранней стадии ее проектирования.

Для проведения количественного анализа диаграмм IDEF0 и DFD используются следующие показатели:

- количество блоков на диаграмме –  $N$ ;
- уровень декомпозиции диаграммы –  $L$ ;
- число стрелок, соединяющихся с  $i$ -м блоком диаграммы –  $A_i$ .

Данный набор показателей относится к каждой диаграмме модели. Ниже перечислены рекомендации по их желательным значениям.

Необходимо стремиться к тому, чтобы количество блоков на диаграммах нижних уровней было бы ниже количества блоков на родительских диаграммах, т.е. с увеличением уровня декомпозиции убывал бы коэффициент  $N/L$ . По мере декомпозиции модели функции должны упрощаться, следовательно, количество блоков должно убывать.

Диаграммы должны быть сбалансированы. Это означает, например, что у любого блока количество входящих стрелок и стрелок управления не должно быть значительно больше, чем количество выходящих. Следует отметить, что данная рекомендация может не выполняться в моделях, описывающих производственные процессы. Например, при описании процедуры сборки в блок может входить множество стрелок, описывающих компоненты изделия, а выходить одна стрелка – готовое изделие.

Количественная оценка сбалансированности диаграммы может быть выполнена с помощью коэффициента сбалансированности:

$$K_b = \left| \sum A_i / N - \max A_i \right|.$$

Необходимо стремиться к тому, чтобы значение  $K_b$  для диаграммы было минимальным.

### **3.4. Основные принципы построения объектной модели**

Концептуальной основой объектно-ориентированного подхода является объектная модель. Основными принципами ее построения являются:

- абстрагирование (abstraction);
- инкапсуляция (encapsulation);
- модульность (modularity);
- иерархия (hierarchy).

Абстрагирование – это выделение наиболее важных, существенных характеристик некоторого объекта, которые отличают его от всех других видов объектов и, таким образом, четко определяют его концептуальные границы с точки зрения дальнейшего рассмотрения и анализа, и игнорирование менее важных или незначительных деталей. Абстрагирование позволяет управлять сложностью системы, концентрируясь на существенных свойствах объекта. Абстрагирование концентрирует внимание на внешних особенностях объекта и позволяет отделить самые существенные особенности его поведения от деталей их реализации. Выбор правильного набора абстракций для заданной предметной области представляет собой главную задачу объектно-ориентированного проектирования. Абстракция зависит от предметной области и точки зрения – то, что важно в одном контексте, может быть не важно в другом. Объекты и классы – основные абстракции предметной области.

Инкапсуляция – физическая локализация свойств и поведения в рамках единственной абстракции (рассматриваемой как «ящик»), скрывающая их реализацию за общедоступным интерфейсом. Инкапсуляция – это процесс отделения друг от друга отдельных элементов объекта, определяющих его устройство и поведение. Инкапсуляция служит для того, чтобы изолировать интерфейс объекта, отражающий его внешнее поведение, от внутренней реализации объекта. Объектный подход предполагает, что собственные ресурсы, которыми могут манипулировать только операции самого объекта, скрыты от внешней среды. Абстрагирование и инкапсуляция являются взаимодополняющими:

- абстрагирование фокусирует внимание на внешних особенностях объекта, а инкапсуляция (или иначе ограничение доступа) не позволяет объектам-пользователям различать внутреннее устройство объекта.

По-другому инкапсуляцию можно описать, сказав, что приложение разделяется на небольшие фрагменты связанной функциональности. Допустим, в банковской системе имеется информация, касающаяся банковского счета, такая как номер счета, баланс, имя и адрес его владельца, тип счета, начисляемые на него проценты и дата открытия. Со счетом также связаны определенные действия: открыть, закрыть его, положить или снять некоторую сумму денег, а также изменить тип, владельца или адрес. Вся эта информация и действия (поведение) совместно инкапсулируются в объект «счет». В результате все изменения банковской системы, связанные со счетами, могут быть реализованы в одном только объекте «счет».

Еще одним преимуществом инкапсуляции является ограничение последствий изменений, вносимых в систему.

Модульность – это свойство системы, связанное с возможностью ее декомпозиции на ряд внутренне сильно сцепленных, но слабо связанных между собой подсистем (модулей). Модульность снижает сложность системы, позволяя выполнять независимую разработку отдельных модулей. Инкапсуляция и модульность создают барьеры между абстракциями.

Иерархия – это ранжированная или упорядоченная система абстракций, расположение их по уровням. Основными видами иерархических структур применительно к сложным системам являются структура классов (иерархия по номенклатуре) и структура объектов (иерархия по составу). Примерами иерархии классов являются простое и множественное наследование (один класс использует структурную или функциональную часть соответственно одного или нескольких других классов), а иерархии объектов – агрегация.

### **3.4.1. Основные элементы объектной модели**

К основным понятиям объектно-ориентированного подхода (элементам объектной модели) относятся:

- объект;
- класс;
- атрибут;
- операция;
- полиморфизм (интерфейс);
- компонент;
- связи.

Объект определяется как осозаемая сущность, предмет или явление (процесс), имеющие четко определяемое поведение. Объект

может представлять собой абстракцию некоторой сущности предметной области (объект реального мира) или программной системы (архитектурный объект). Любой объект обладает состоянием (state), поведением (behavior) и индивидуальностью (identity).

Состояние объекта – одно из возможных условий, в которых он может существовать, оно изменяется со временем. Состояние объекта характеризуется перечнем всех возможных (статических) свойств данного объекта и текущими значениями (динамическими) каждого из этих свойств. Состояние объекта определяется значениями его свойств (атрибутов) и связями с другими объектами.

Поведение определяет действия объекта и его реакцию на запросы от других объектов. Поведение характеризует воздействие объекта на другие объекты, изменяющее их состояние. Иначе говоря, поведение объекта полностью определяется его действиями. Поведение представляется с помощью набора сообщений, воспринимаемых объектом (операций, которые может выполнять объект).

Каждый объект обладает уникальной индивидуальностью. Индивидуальность – это свойства объекта, отличающие его от всех других объектов.

Структура и поведение схожих объектов определяют общий для них класс. Термины «экземпляр класса» и «объект» являются эквивалентными.

Класс – это множество объектов, связанных общностью свойств, поведения, связей и семантики. Класс инкапсулирует (объединяет) в себе данные (атрибуты) и поведение (операции). Класс является абстрактным определением объекта и служит в качестве шаблона для создания объектов. Графическое представление класса в языке UML показано на рис. 3.13. Класс изображается в виде прямоугольника, разделенного на три части. В первой содержится имя класса, во второй – его атрибуты. В последней части представлены операции класса, отражающие его поведение (действия, выполняемые классом).

Любой объект является экземпляром (instance) класса. Определение классов и объектов – одна из самых сложных задач объектно-ориентированного проектирования.

Атрибут – поименованное свойство класса, определяющее диапазон допустимых значений, которые могут принимать экземпляры данного свойства.

Атрибут – это элемент информации, связанный с классом.

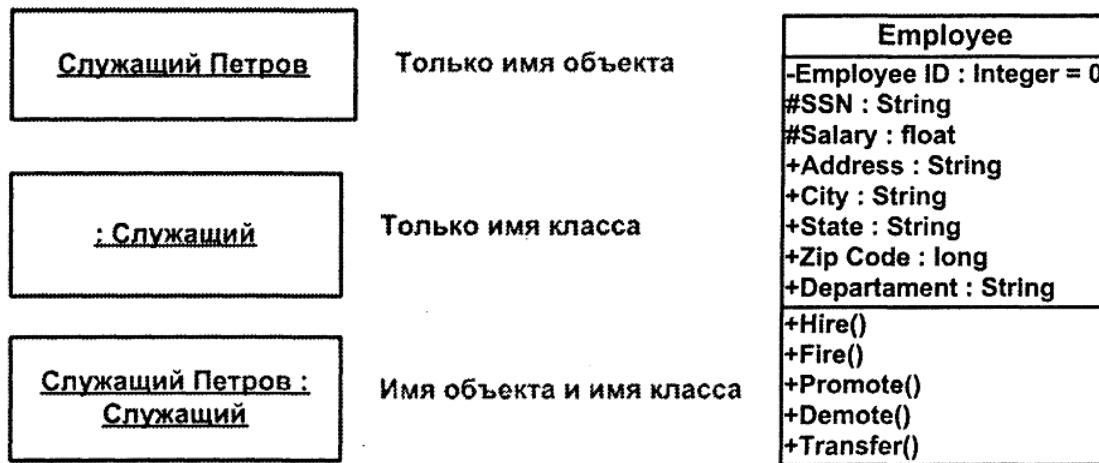


Рис. 3.13. Графическое представление класса

### 3.5. Краткий обзор объектно-ориентированных методов анализа и построения моделей

На данный момент известно более пятидесяти объектно-ориентированных методов анализа ПрО, которые прошли проверку практикой. Приведем некоторые основные из них:

- метод *объектно-ориентированного системного анализа OOAS* (*Object-Oriented system analysis*), позволяющий выделить сущности и объекты ПрО, определить их свойства и отношения, а также построить на их основе информационную модель, модель состояний объектов и процессов представления потоков данных (dataflow);

- метод *объектно-ориентированного анализа OOA* (*Object-Oriented analysis*), обеспечивающий моделирование ОМ и формирование требований к ПрО с помощью понятия «сущность-связь» (entity-relationship ER), спецификацию потоков данных и соответствующих процессов;

- метод *SD (Structured Design)* структурного проектирования системы, данных и программ преобразования входных данных в выходные с помощью структурных карт Джексона;

- методология *объектно-ориентированного анализа и проектирования OOAD* (*Object-oriented analysis and design*), которая основывается на ER-моделировании сущностей и отношений в объектной модели ПрО, обеспечивает определение системы и организацию данных с использованием структурных диаграмм, диаграмм «сущность-связь» и матрицы информационного управления;

– технология объектного моделирования *OMT* (Object Modeling Technique) включает в себя процессы (анализа, проектирования и реализации), набор нотаций для задания четырех моделей (объектной, динамической, функциональной и взаимодействия);

– объединенный метод *UML*, включающий средства и понятия метода Г. Буча (объекты, классы, суперклассы), принципы наследования, полиморфизма и сокрытия информации об объектах, а также варианты использования метода Джекобсона для задания сценариев работы системы при выполнении задач ПрО и диаграммные средства взаимодействия объектов Румбауха;

– метод определения распределенных объектов на основе объектной модели *CORBA* и набора сервисных системных компонентов общего пользования, обеспечивающих их функционирование в среде распределенных приложений;

– метод генерации (*generative*) частей системы из семейства ПрО с помощью готовых объектов, аспектов, компонентов, программ многоразового использования и приложений, а также модели характеристик, в которой представлены функциональные и нефункциональные требования к семейству систем.

Наиболее используемая *объектная модель ПрО* реализована в системе CORBA. Каждый объект модели инкапсулирует некоторую сущность ПрО и определяет один или несколько сервисов (методов) ее реализации. Объекту соответствует одна или несколько операций обращения к методам. Объекты группируются в типы, а их экземпляры – в подтипы/супертипы.

Они инкапсулируют методы реализации, которые невидимы во внешнем интерфейсе, т.е. ОМ не содержит информации о способах реализации типа, а только о наличии его реализации. Во внешнем интерфейсе содержатся операции, которые вызывают методы объектов для их выполнения. Специализация типа определяется постепенно на этапах стратегии, анализа, проектирования и реализации объекта. Взаимодействие объектов осуществляют *брокер объектных запросов* и операций.

Приведенная общая характеристика разновидностей объектно-ориентированных методов показывает, что они имеют много общих черт (например, ER-моделирование, Dataflow), а также свои специфические особенности. Каждый разработчик метода объектно-ориентированного анализа вводил необходимые новые понятия, которые зачастую семантически совпадали с аналогичными понятиями в других методах. Поэтому у авторов UML возникла идея объ-

единить свои индивидуальные методы объектного анализа (Буча, Джекобсона и Рамбауха) для создания единого метода объектного моделирования UML.

### **3.5.1. Основные понятия методов объектного анализа ПрО**

К основным понятиям методов объектного анализа ПрО относем следующие.

*Объект ПрО* – это абстрактный образ с поведением, которое обусловлено его характеристиками и взаимоотношениями с другими объектами ПрО.

Согласно теории Фреге спецификацию объекта можно трактовать как треугольник:

<имя объекта> <денотат> <концепт>,

где <имя объекта> – идентификатор, строка из литер и десятичных чисел; <денотат> – сущность реального мира ПрО, которую обозначает идентификатор; <концепт> – смысл (семантика) денотата в соответствии с интерпретацией сущности моделируемой ПрО.

Объект интерпретируется как понятийная структура, состоит из идентификатора, денотата – образа предмета и концепта, отображающего смысл этого денотата, исходя из цели объектного моделирования ПрО. Денотат можно идентифицировать различными символами алфавита. Одному объекту могут соответствовать несколько концептов в зависимости от избранного уровня абстракции. Объект определяется через его внешнее отличие от других объектов. Внутренняя особенность объекта (его структура, внутренние характеристики) не влияет на внешнее отличие и для объектного моделирования значения не имеет.

*Сущность* – это семантически важный объект или тип объекта, существующий реально в ПрО или являющийся абстрактным понятием, информацию о котором необходимо знать и/или сохранять. Имя сущности должно быть уникальным и может представлять тип или класс объектов. Сущность может иметь синонимы, записываемые через знак "/" (например, аэропорт / аэродром).

*Концепт* – значение некоторой абстрактной сущности ПрО, обозначается уникальным именем или идентификатором. Группа подобных концептов – это родительский концепт, который заведомо определяется некоторым набором общих атрибутов. Концепт вместе со своими атрибутами представляется графически в ОМ или в текстовом виде.

*Атрибут* – это абстракция, которой владеют все абстрагированные концепты сущности. Каждый атрибут обозначается именем, уникальным в границах описания концепта. Множество объединенных в группу атрибутов обозначает идентификатор этой группы. Группа атрибутов может объединяться в класс и иметь идентификатор класса.

*Отношение* – это абстракция набора связей, которые имеют место между разными видами объектов ПрО, абстрагированных как концепты. Каждая связь имеет уникальный идентификатор. Отношения могут быть текстовыми или графическими. Для формализации отношений между концептами добавляются вспомогательные атрибуты и ссылки на идентификаторы этих отношений. Некоторые отношения образуются как следствие существования других отношений.

*Класс* – это множество объектов, обладающих одинаковыми свойствами, операциями, отношениями и семантикой. Любой объект – это экземпляр класса. Класс представляется различными способами (например, списками объектов, операций, состояний). Изменяется класс количеством экземпляров, операций и т.п.

*Предметная область* – это то, что анализируется с целью выделения специфичного множества понятий (сущностей, объектов) и связей между ними. На множестве этих понятий определяются задачи в целях автоматизированного их решения. Пространство ПрО можно разделить на пространство задач (*problem space*) и пространство решений (*solution space*).

*Пространство задач* – это сущности, концепты ПрО, а *пространство решений* – это множество программных реализаций задач, в том числе функциональные компоненты, которые обеспечивают решение задач и функций ПрО, представленных в этом пространстве.

Выделение сущностей ПрО проводится с учетом отличий, определяемых соответствующими понятийными структурами. Объект как абстракции реального мира и понятийная структура обладает поведением, обусловленным свойствами и отношениями данного объекта с другими объектами. Выделенные в ПрО объекты структурно упорядочиваются *теоретико-множественными операциями* (принадлежности, объединения, пересечения и др.).

*Модель ПрО* – это совокупность точных определений понятий, концептов, объектов и их характеристик, а также множества синонимов и классифицированных логических взаимосвязей между этими понятиями.

*Концептуальная модель* – это модель ПрО, которая создается без ориентации на программные и технические средства выполнения задач ПрО в операционной среде.

Для объектов модели устанавливаются отношения или связи. Различаются статические (постоянные) связи, которые не изменяются или изменяются редко, и динамические связи, которые имеют определенные состояния и изменяются во время функционирования системы.

Связи между объектами могут быть следующие:

- *связь один к одному* (1:1) существует тогда, когда один экземпляр объекта некоторого класса связан с единственным экземпляром другого класса, т.е. в связи принимают участие по одному экземпляру из классов;

- *связь один ко многим* (1 : N) существует тогда, когда один экземпляр объекта некоторого класса связан одновременно с одним или более экземплярами другого класса или того же самого класса;

- *связь многие ко многим* (M : N) существует тогда, когда в связях принимают участие несколько экземпляров объектов двух классов, т.е. один или больше экземпляров другого класса связан с одним или более экземплярами первого класса.

Состояние связей между объектами с течением времени может эволюционировать, и они могут существенно влиять на ход решения задачи. Для таких случаев связи строится ассоциативный объект и определяется модель состояний этого объекта путем добавления атрибута, фиксирующего текущее состояние.

Среди действий, которые сопровождают переходы объектов в определенные состояния в модели состояний, должны быть операции создания нового экземпляра ассоциативного объекта, если новая пара экземпляров вступает в связь, или его уничтожения в случае, если объект или связь перестают существовать.

Используя приведенные базовые понятия методов объектного анализа ПрО, далее излагаются объектный метод анализа ПрО и построения моделей, визуальный метод моделирования – UML и проектирование архитектуры системы на основе стандартов.

### **3.5.2. Объектный метод построения моделей ПрО**

Наибольшее распространение среди методов анализа ПрО получил метод ООАС Шлеера и Меллора, предназначенный для отображения ПрО следующими моделями:

- информационная модель системы;

- модель состояний объектов в информационной модели системы, определяемая набором правил поведения, предписаний и физических законов;
- модель процессов, которая отображает процессы и действия, совершающиеся в системе и обеспечивающие прохождение моделей состояний через жизненные циклы – получение, порождение и уничтожение событий в системе.

Согласно этого методу ПрО анализируется в три этапа: информационное моделирование, моделирование состояний, моделирование процессов. В результате их выполнения создается система в виде совокупности этих моделей. Информационная модель отображает ПрО как мир объектов с характеристиками и атрибутами.

При переходе от этого этапа к этапу моделирования состояний для объектов информационной модели определяются связи объектов и их поведение. Создается модель состояний, которая отображает динамику состояния объектов системы и их поведение. На третьем этапе определяются действия и процессы, которые порождают события. Действия имеют функциональную природу. Цель моделирования процессов состоит в том, чтобы расчленить процессы на действия, которые вместе взятые определяют функциональное содержание системы. Рассмотрим модели метода подробнее.

Под **информационной моделью** понимается совокупность объектов (сущностей) ПрО, их характеристик (атрибутов) и связей между ними. Она создается по принципу реляционной модели данных, т.е. представления данных в виде отношений между ними.

Анализ ПрО состоит в выявлении объектов, предоставлении им уникальных и значимых названий, соответствующих смысловым понятиям в этой предметной области. В качестве объектов могут выступать:

- реальные предметы мира ПрО как абстракции фактически существующих физических объектов ПрО;
- роли как абстракции целей или назначения человека, части организации;
- взаимодействия – объекты, получаемые путем установления отношений между другими объектами или частями системы;
- спецификации, используемые для представления правил, критериев и ограничений на применение объектов в системе.

Таким образом, элементами информационной модели могут быть объекты, их атрибуты и идентификаторы, а также связи между объектами.

Для объектов ПрО определяются их характерные признаки или свойства, называемые атрибутами. Каждый атрибут – это абстракция одной характеристики объекта, которая присуща всем представителям класса объектов. Для классов объектов выбираются уникальные имена, устанавливаются атрибуты и связи. Атрибут получает имя, уникальное в рамках класса. Различаются описательные, указывающие и вспомогательные атрибуты.

*Описательный атрибут* устанавливает реальную характеристику, которая может определяться одним из таких возможных способов:

- заданием числового диапазона;
- перечислением возможных значений, которые может принимать атрибут;
- ссылкой на документ, который определяет возможные значения;
- заданием правил генерации допустимых значений.

*Указывающий атрибут* задает форму, назначение, перечисление или ссылку.

*Дополнительный атрибут* задает дополнительные значения, которые может принимать атрибут объекта.

Идентификаторы объекта содержат один или несколько атрибутов, значения которых позволяют однозначно выделить экземпляр объекта в данном классе (например, табельный номер сотрудника, номер паспорта и др.).

Ссылка на некоторый атрибут может уточняться именем класса, задаваемым через точку, а атрибуты – отношениями, которые определяются по следующим правилам:

- каждый объект – экземпляр класса или более чем одного класса, обладает одним значением своего атрибута;
- идентификатор может составляться из нескольких имен атрибутов (через точку), первое имя относится к классу, остальные – к имени объекта.

*Связи объектов* устанавливаются между объектами одного или другого класса и характеризуются количеством экземпляров объектов, которые одновременно могут принимать участие в этих связях.

В информационной модели связи между объектами изображаются стрелками, указывающими направление связи. Возле рамки объекта, принимающего участие в связи, на линии стрелки указывается роль, которую этот объект поддерживает в данной связи. Связь

1:1 обозначается двунаправленной стрелкой, имеющей по одному «наконечнику» с каждой стороны; связь 1 : N представляется стрелкой, имеющей два «наконечника» со стороны объекта, который состоит в связи с несколькими объектами; и, наконец, по два «наконечника» с каждой стороны имеет стрелка, означающая связь N : M.

Над стрелкой может указываться название связи. Связи могут быть безусловными, если каждый экземпляр объекта класса принимает участие в связи, и условными, когда отдельные экземпляры объектов класса не принимают участия в связи. Пример информационной модели с отображением связей приведен на рис. 3.14.

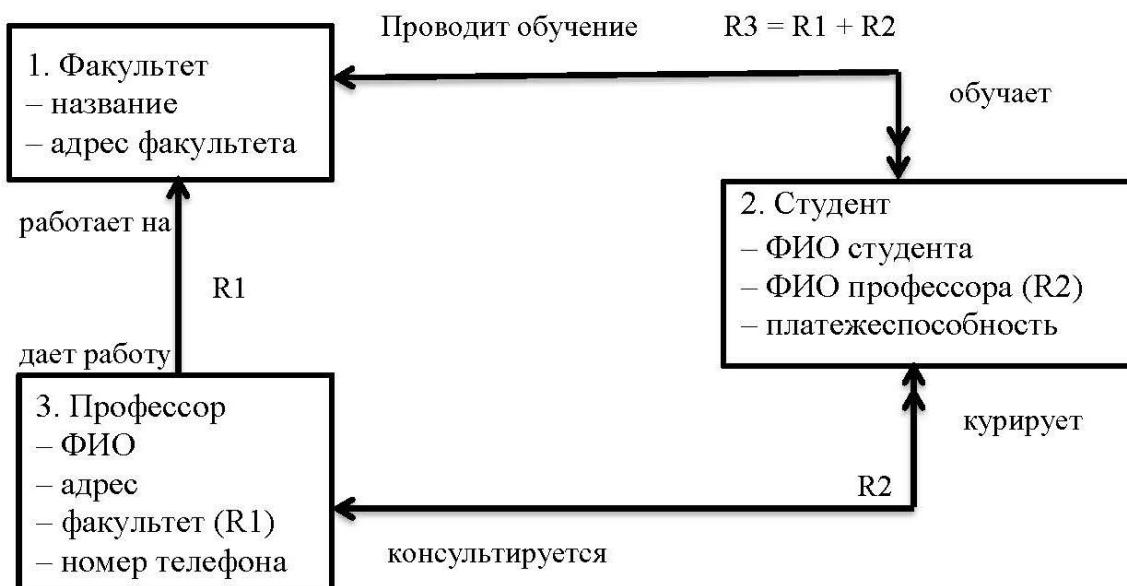


Рис. 3.14. Пример информационной модели

В этом рисунке связь R3 – логическое следствие связей R1 и R2.

Построенная информационная модель сопровождается неформальным описанием всех объектов, их атрибутов и связей, в которых объекты принимают участие.

**Модель состояний** предназначена для отображения динамического поведения и изменения состояний каждого из объектов информационной модели и жизненного цикла поведения объектов. Состояние в модели – это положение или ситуация объекта, определяемая правилами и линией поведения. Событие заставляет объект переходить из одного состояния в другое. Экземпляры класса имеют поведение, которое определяется:

- состоянием, зависящим от текущих значений отдельных его атрибутов;

- состоянием, изменяемым в результате выполненных над объектами действий;
- состоянием ПрО, зависящим от совокупности состояний ее объектов;
- некоторыми процессами и действиями, которые изменяют жизненный цикл состояния объекта.

Построение модели состояний начинается после выделения в информационной модели отдельных объектов, обладающих динамическим поведением, создания экземпляра объекта или его уничтожения после прекращения существования.

В данном методе предусмотрены две нотации для представления динамических аспектов поведения объектов: диаграмма перехода состояний и таблица перехода в состояния.

При построении модели состояний для каждого объекта информационной модели определяются следующие множества и правила:

- множество состояний, в которых объект может находиться;
- множество событий, которые побуждают экземпляры класса изменять свое состояние;
- правила перехода объекта из зафиксированного состояния на новое состояние при условии, что произойдет некоторое событие из множества событий;
- действие на каждое из состояний выполняется при переходе в новое состояние.

Эта информация представляется в диаграмме перехода состояний (рис. 3.15) исходя из следующих условий:

- каждое состояние, определенное для класса объектов, получает номер, уникальный идентификатор (ID) и название;
- состояние обозначается рамкой, содержащей номер и название;
- переход от состояния к состоянию изображается направленной дугой, помеченной меткой и названием события, обусловившего переход;
- начальное состояние обозначается стрелкой, направленной к соответствующей рамке, и является состоянием, которое экземпляр объекта приобретает после своего создания;
- заключительное состояние жизни экземпляра объекта (продолжение или разрушение) обозначается пунктирной рамкой;
- указание на действия, которые должны быть выполнены экземпляром объекта для перехода в другое состояние.

Изменение состояния экземпляра класса объектов осуществляется при выполнении таких действий:

- обработка информации, переданной в систему, что может повлиять на некоторое событие;
- изменение поведения атрибута объекта;
- вычисление атрибута;
- генерация некоторой операции для одного из экземпляров класса объектов;
- генерация события, сообщение о котором передается объекту, внешнему по отношению к данному;
- прием сообщения о событии от внешних объектов;
- взаимодействие с таймером, измеряющим время, истечение которого приводит к созданию некоторого события.

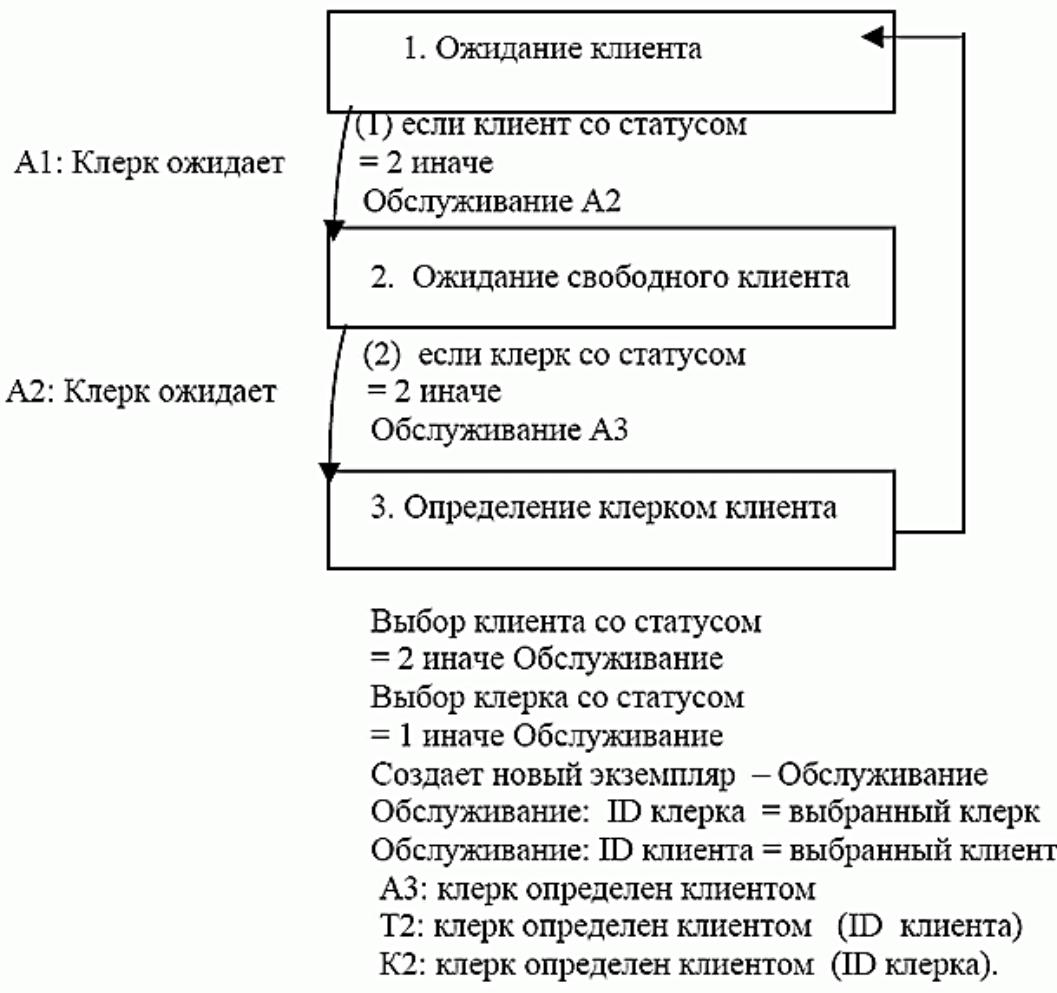


Рис. 3.15. Модель состояний для обслуживания клиентов

Для отдельного экземпляра объекта может быть установлен таймер, который сообщит о наступлении события, соответствующего значению таймера (например, остановка работы прибора).

Альтернативой графической диаграммы перехода состояний является табличная нотация (табл. 3.4 для модели состояний на рис. 3.15).

В таблице каждое состояние представляется строкой, а каждое событие, воздействующее на объект – столбцом. Клетка таблицы перехода состояний – это состояние объекта, если соответствующее столбiku событие произойдет, когда объект находился в состоянии, соответствующем строке. При этом допускается, что некоторые комбинации событие/состояние не приведут к изменению состояния экземпляра объекта, они содержат указание «событие игнорируется». При выборе формы представления – диаграммы или таблицы состояний перехода – преимущество имеет диаграмма из-за наглядности и определенности действий, тогда как табличная форма служит для фиксации всех возможных комбинаций состояния/событие. Этим обеспечиваются полнота и непротиворечивость заданных требований к системе.

*Таблица 3.4*

**Таблица переходов состояний – обслуживание клиентов**

	A1 – клиент ожидает	A2 – клиент ожидает	A3 – известен клиенту
1. Ожидание клиента	2	Событие игнорируется	Не может произойти
2. Ожидание свободного клерка	Событие игнорируется	3	Не может произойти
3. Определение клерка клиентом	Событие игнорируется	Событие игнорируется	1

Важным принципом объединения объектов и компонентов в систему является наличие у них общих событий, причем чаще всего один из них порождает событие, а другие на него реагируют. На этом принципе базируется способ объединения отдельных объектов и компонентов в систему. Взаимодействие (внешнее и внутреннее) объектов рассматривается через обмен сообщениями для задания определенных событий и данных к ним. Внешний объект посыпает сообщение, которое приводит к запуску системы и образованию внешнего события. Ему направляется сообщение о наступлении или отсутствии события.

Поведение отдельного объекта представляется в модели диаграммой перехода в состояния, а поведение системы – в виде схемы взаимодействия отдельных диаграмм, каждая из которых получает

название в соответствующем овале (рис. 3.16). Овалы, отображающие отдельные диаграммы перехода состояний, связаны между собой стрелками, на которых задаются сообщения для возбуждения события. На стрелке указывается метка события (например, C1, C2, ..., C8), а ее направление соответствует направлению передачи сообщения. Внешние объекты обозначаются прямоугольниками с названиями.

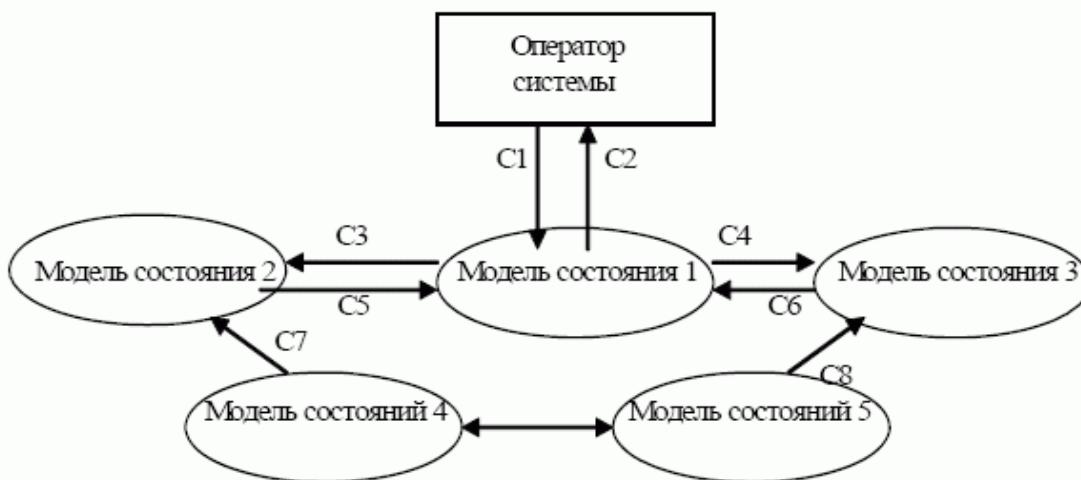


Рис. 3.16. Схема взаимодействия моделей поведения объектов

С помощью событий, указанных на стрелках данной схемы, инициируются модели состояний 1–5, каждая из которых посыпает соответствующее сообщение.

Таким образом, модель состояний представляется диаграммами перехода в состояния, *таблицей состояний*, описанием действий и событий, изменяющих состояния объектов.

**Модель процессов** отражает изменения в моделях состояний. Каждое действие определяется в терминах процессов и архивов данных объектов, т.е. процесс является фундаментальным модулем операции, а архив данных соответствует атрибутам объектов в информационной модели. Процессы действий имеют доступ к данным модели состояний, где они представлены. Для каждого действия модели состояний создается диаграмма процесса. Действия инициируют выполнение событий с помощью функций, которые реализуются в системе и отображаются в соответствующих диаграммах действий. В качестве источников данных для процессов могут выступать:

- атрибуты объектов, которые продолжают существовать после завершения работы системы;
- системные часы;

- таймер;
- данные происходящих событий;
- сообщения от внешних объектов.

Последовательность выполняемых процессов образует поток управления, а каждый процесс образует поток данных.

Для представления потоков данных используют диаграммы действий, правила построения таких диаграмм таковы:

- 1) каждой из диаграмм перехода состояний может отвечать только одна диаграмма действий потоков данных;
- 2) процесс изображается овалом с указанием содержания или названия процесса;
- 3) потоки данных процессов изображаются стрелками, на которых указываются имена данных, передаваемых другому процессу;
- 4) стрелка к овалу процесса указывает на его входные данные, направление от овала – на выходные данные;
- 5) источники данных изображаются прямоугольниками;
- 6) данным, имеющим своим источником архивные объекты, соответствуют потоки с названиями атрибутов объектов, которые передаются потоками;
- 7) потоки данных отмечаются таймером или системными часами (час, минута);
- 8) событие, сообщение о котором получает процесс, изображается стрелкой с названиями данных.

В данном методе различаются следующие процессы общего назначения:

- 1) доступ к архивам;
- 2) подготовка и верификация объектов;
- 3) обработка потоков данных и генерация событий;
- 4) накопление объектов в архиве;
- 5) организация вычислений функций системы.

Потоки обозначаются пунктирными стрелками. Если процесс выполняет проверку определенного условия для передачи управления и входных данных другому процессу, то соответствующий поток изображается пунктирной линией с перечеркиванием. Фрагмент диаграммы действий процесса создания репозитария (типа библиотеки, архива) объектов приведен на рис. 3.17.

К диаграммам действий потоков данных добавляется неформальное описание функций процессов, которые входят в их состав. Для описания подробностей действий процессов нотация не регламентируется.

После завершения описания диаграммы действий потоков данных для всех объектов системы составляется общая *таблица процессов*, состоящая из следующих колонок:

- 1) идентификатор процесса;
- 2) тип процесса;
- 3) название процесса;
- 4) название состояния, для которого определен процесс;
- 5) название действия состояния.

Таблица дает возможность проверить:

- 1) непротиворечивость названий и идентификаторов процессов;
- 2) полноту определенных событий и соответствующих процессов;
- 3) генерацию события или его обработку соответствующим процессом.

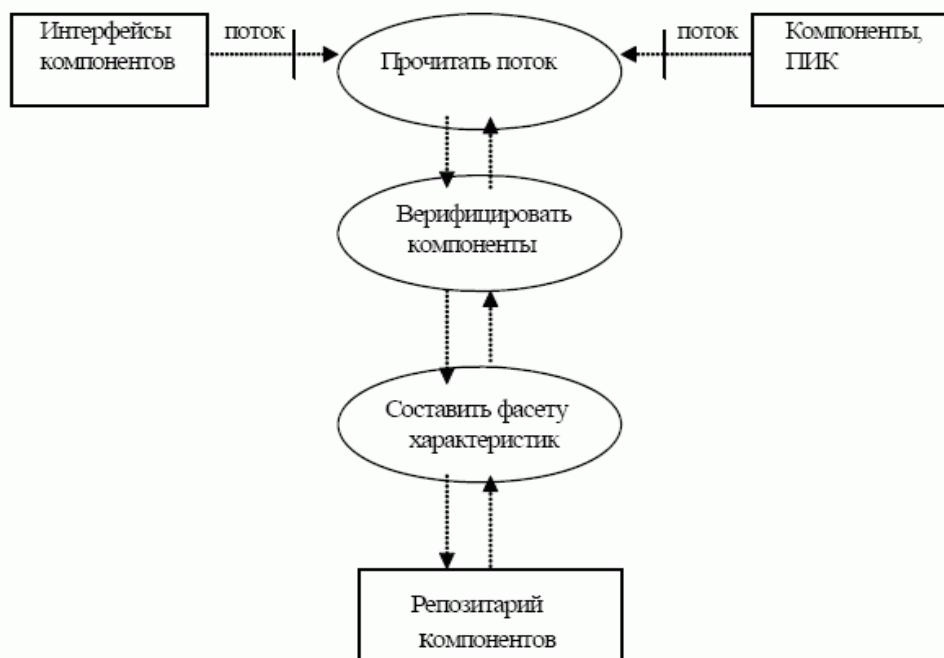


Рис. 3.17. Пример диаграммы действий процессов создания репозитария

К диаграммам действий потоков данных добавляется неформальное описание функций процессов, которые входят в их состав. Для описания подробностей действий процессов нотация не регламентируется.

После завершения описания диаграммы действий потоков данных для всех объектов системы составляется общая *таблица процессов*, состоящая из следующих колонок:

- 1) идентификатор процесса;
- 2) тип процесса;
- 3) название процесса;
- 4) название состояния, для которого определен процесс;
- 5) название действия состояния.

Таблица дает возможность проверить:

- 1) непротиворечивость названий и идентификаторов процессов;
- 2) полноту определенных событий и соответствующих процессов;
- 3) генерацию события или его обработку соответствующим процессом.

На данном этапе создается модель доступа к объектам, которая отображает взаимодействие объектов через модель состояний. Модель состояний обращается к данным экземпляра другого объекта во время выполнения действия. Этот вид взаимодействия считается *синхронным*. Если модель состояний получает событие после того, как действие завершилось, то это взаимодействие – *асинхронное*.

Результатом этапа моделирования процессов являются модель доступа к данным, диаграмма потоков данных действий, *таблица процессов*, содержащая процессы и действия над объектами ПрО, описание процессов путем их упорядочения по ID.

**Определение архитектуры ПрО.** После построения трех моделей выполняется следующий этап метода – проектирование. На нем проводятся разделение ПрО на подсистемы, определение их функций и принципов выполнения этих функций на процессах. В каждую подсистему включаются построенные модели или их фрагменты и соответственно выделенные объекты со всеми характеристиками. Между подсистемами устанавливаются соединительные связи, на которых указываются имена передаваемых данных или участвующих объектов. В результате создается графическое *представление архитектуры* системы. Преобразование результатов объектно-ориентированного анализа по данному осуществляется в языки C++, Ada и др.

Данный метод имеет много общего с методом Буча, реализован в ряде проектов (например, EaseCASE Plus 4.0). Применяется в различных областях (банковские операции, управление летательными аппаратами, оформление кредитных карт и др.) в США, Европе и Японии.

### **3.6. Методы проектирования архитектуры ПО**

*Проектирование ПО* – это процесс разработки, следующий за этапом анализа и формирования требований. Задача проектирования – это преобразование требований к системе в требования к ПО и построение архитектуры системы.

*Архитектура системы* – это структурная схема компонентов системы, взаимодействующих между собой через интерфейсы. Компоненты могут составляться из последовательности более мелких компонентов и интерфейсов. Разработка архитектуры основывается на общем наборе справочников, классификаторов и т.п. В ней идентифицированы общие части, в том числе готовые программные продукты и вновь разработанные компоненты, а также многократно используемые в производстве других приложений.

Основное условие построения архитектуры системы – это декомпозиция системы на компоненты или модули, а также:

- определение целей и проверка их выполнимости;
- определение входных и выходных данных;
- иерархическое представление абстракции системы и скрытие тех деталей, которые будут отработаны на следующих уровнях.

Основные решения по структуре системы принимаются группой архитекторов и аналитиков. Проект разбивается на разделы или отдельные части для их выполнения небольшими группами разработчиков, каждая из которых отвечает за одну или несколько частей системы.

Другой вариант определения архитектуры – это множество представлений, каждое из которых отражает некоторый аспект, интересующий группу участников проекта – аналитиков, проектировщиков, конечных пользователей и др. Представления фиксируют проектные решения по проектированию структуры и отражают аспект разделения приложения на отдельные компоненты и их связи. Эти решения проистекают из требований функциональности и влияют на проектные решения для нижних уровней структуры.

Проектирование архитектуры системы может проводиться структурным, объектно-ориентированным, компонентным и другими методами, каждый из которых предлагает свой путь построения архитектуры, включая концептуальную, объектную и другие модели и соответствующие им конструктивные элементы (блок-схемы, графы объектов и компонентов и др.).

При применении объектно-ориентированного подхода в качестве компонентов выступают отдельные объекты, а процесс кон-

струирования объектной структуры превращается в процесс выявления имеющихся в ПрО объектов и определения их поведения и взаимодействия.

К методам проектирования относятся стандартный подход к проектированию, основанный на сформировавшейся общесистемной технологии традиционного проектирования программных систем.

### **3.6.1. Стандартный подход к проектированию**

Разработка автоматизированных систем (АС) выполнялась на основе стандарта ГОСТ 34.601–90, регламентирующего стадии и этапы процесса разработки АС с учетом особенностей АС и средств объединения подсистем. Основание для разработки АС – это договор между разработчиком системы и заказчиком.

Этапами данного стандарта являются:

- формирование требований;
- разработка концепции системы;
- проектирование эскизного, технического и рабочего проекта.

В эскизном и техническом проекте на основе сформулированных требований и концепций определяются конкретные задачи системы, строится структура системы, а также определяются пути и алгоритмы реализации подсистем. Эти этапы заканчиваются созданием и утверждением отчета о научно-исследовательской работе, в котором дается оценка необходимых для реализации АС ресурсов, вариантов и порядка проведения оценки качества системы.

На этапе разработки *эскизного проекта* используются проектные решения ко всей системе или к ее части, определяются перечень задач, концепция информационной базы, функции и параметры основных компонентов системы, а также основные алгоритмы обработки информации.

Этап технического проектирования предусматривает разработку проектных решений относительно системы и ее частей, разработку документации, комплектацию АС, а также способов реализации технических требований на систему, алгоритмов задач, их распределения по смежным частям проекта и обмена данными между ними.

Проектные решения определяют организационную структуру, функции персонала АС, набор необходимых технических средств, языка и системы программирования, типы СУБД, систему классификации и кодирования, справочники, а также варианты ведения информационной базы системы.

Данный стандарт обеспечивает:

- *концептуальное проектирование*, которое состоит в построении концептуальной модели, уточнении и согласовании требований;
- *архитектурное проектирование*, которое состоит в определении главных структурных особенностей создаваемой системы;
- *техническое проектирование* – это отображение требований, определение задач и принципов их реализации в среде функционирования системы;
- *детальное рабочее проектирование*, которое состоит в спецификации алгоритмов задач, построении БД и программного обеспечения системы.

Рассмотрим каждый вид проектирования более подробно.

При концептуальном проектировании определяются:

- источники поступления данных от заказчика, который несет ответственность за их достоверность;
- объекты системы и их атрибуты;
- способы материализации связей между объектами и виды организации данных;
- интерфейсы с потенциальными пользователями системы для оказания им помощи при формулировке целей и функций системы;
- методы взаимодействия пользователей с системой для обеспечения скорости реакции системы.

Организация интерфейсов базируется на ключевых понятиях, связанных с конкретными экранами и форматами обмена данными, а также включает:

- термины, образы и понятия, которые имеют значение для пользователя и домена;
- модель организации, представления данных, функций и ролей, а также результаты их просмотра;
- визуальные приемы отображения на экране элементов системы, наглядных для пользователя;
- методы взаимодействия подсистем.

При архитектурном проектировании системы может применяться язык UML, который позволяет учитывать аспекты,ственные действующим лицам, а также устанавливать форматы в меню и иконах интерфейсов.

Общая концепция объектного проектирования заключается в построении всех экранных форм и апробации их стиля на их разных вариантах. Выбор вариантов может вступить в противоречие с за-

данными характеристиками нефункциональных требований (например, обеспечение конфиденциальности, быстродействия и др.).

На основе модели *представления требований* и понятий компонентного или объектно-ориентированного проектирования проводится уточнение состава и содержания функций системы, методов их реализации и обеспечение их взаимодействия с помощью диаграмм потоков данных.

*Взаимодействие объектов* – это обмен сообщениями между элементами системы, подготовка ответа при выполнении операций, изменяющих свое состояние, и отправка ответа другим объектам.

Для уточнения поведения объектов используются диаграммы UML, отображающие различные аспекты взаимодействия объектов. Эти уточнения касаются интерфейсов и поведения объектов в сценариях, а также пересмотра моделей требований и состава объектов системы. Изменения начинаются с требований и поиска мест локации для внесения необходимых изменений в модель требований и их трассирование. Наряду с изменением требований к функциям системы могут изменяться нефункциональные требования, касающиеся ограничений на структуру системы и условий среды функционирования системы (отказоустойчивость и др.).

Модели требований для таких систем учитывают назначение и место требований в таких системах. Для этих целей разработаны национальные, корпоративные и ведомственные стандарты, которые фиксируют правила формирования нефункциональных требований, результатом которых могут быть сведения по обеспечению взаимодействия, защиты данных и др.

### **3.6.2. Общесистемный подход к проектированию архитектуры**

Один из путей архитектурного проектирования – традиционный неформальный подход к определению архитектуры системы, составу ее компонентов, способов их представления и объединения во взаимосвязанную систему. Фактически создаваемая архитектура состоит из четырех уровней, которые включают:

- 1) системные компоненты, устанавливающие интерфейс с оборудованием и аппаратурой;
- 2) общесистемные компоненты, устанавливающие интерфейс с универсальными системами компьютеров;
- 3) специфические бизнес-компоненты;
- 4) прикладные программные системы.

**1-й уровень** – системные компоненты. Они осуществляют взаимодействие с периферийными устройствами компьютеров (принтеры, клавиатура, сканеры, манипуляторы и т.п.) и используются при построении операционных систем.

**2-й уровень** – общесистемные компоненты. Они обеспечивают взаимодействие с универсальными сервисными системами среды работы прикладной системы типа операционные системы, СУБД, системы баз знаний, системы управления сетями и т.п. Компоненты данного слоя используются во многих приложениях как необходимые составные компоненты.

**3-й уровень** – специфические компоненты определенной проблемной области, которые являются составляющими компонентами программных систем и предназначены для решения различных задач (например, бизнес-задач).

**4-й уровень** – прикладные программные системы, которые реализуют конкретные задачи отдельных групп потребителей информации из разных предметных областей (офисные системы, системы бухгалтерского учета и др.) и могут использовать компоненты нижних уровней.

Компоненты любого из выделенных уровней используются, как правило, на своем уровне или более верхнем. Каждый уровень отражает соответствующий набор знаний, умений и навыков специалистов, создающих или использующих компоненты. Этот набор определяет соответствующее разделение специалистов программной инженерии (системщики, прикладники, программисты и др.).

При проектировании архитектуры программная система рассматривается как композиция компонент третьего уровня с доступом до компонентов первого и второго уровней. То есть архитектурное проектирование – это разработка компонентов третьего уровня, определение входных и выходных данных, слоев иерархии компонентов и их связей.

Результат проектирования – архитектура и инфраструктура, содержащая набор объектов, из которых можно формировать некоторый конкретный вид архитектурной схемы для конкретной среды выполнения системы. Заканчивается проектирование архитектуры системы описанием, в котором отображены зафиксированные проектные решения, логическая и физическая структура системы, а также способы взаимодействия объектов.

Объектный стиль проектирования заключается в декомпозиции проблемы на отдельные подсистемы (пакеты), определении

функциональных и нефункциональных требований и модели предметной области. Определяются носители интересов (акторов), их возможные действия в пакете для получения результатов. Основу пакета составляет объектная модель, варианты использования, состав объектов и принципы их взаимодействия. Поведение объектов отражается диаграммами, которые задают последовательность взаимодействий объектов, правила перехода от состояния к состоянию (диаграммы состояний) и действия (диаграммы действий), а также поведение объектов кооперации (диаграммы кооперации). Объекты и соответствующие им диаграммы использования задают общую архитектурную схему системы, в рамках которой осуществляется реализация структуры и специфики поведения компонентов.

Архитектурная схема может быть распределенная, клиент-серверная и многоуровневая.

*Распределенная схема* обеспечивает взаимодействие компонентов системы, расположенных на разных компьютерах через стандартные механизмы вызова RPC (Remote Procedure Calls), RMI (Remote Method Invocation), которые реализуются промежуточными средами (COM/DCOM, CORBA, Java и др.). Взаимодействующие компоненты могут быть неоднородными, на разных языках программирования (C, C++, Паскаль, Java, Basic, Smalltalk и др.), которые допускаются в промежуточной среде системы CORBA (рис. 3.18). Для каждой пары языков взаимодействующих компонентов создаются интерфейсы типа  $L_i, L_n$  по количеству пар ЯП системы, допускающих взаимодействие между собой.

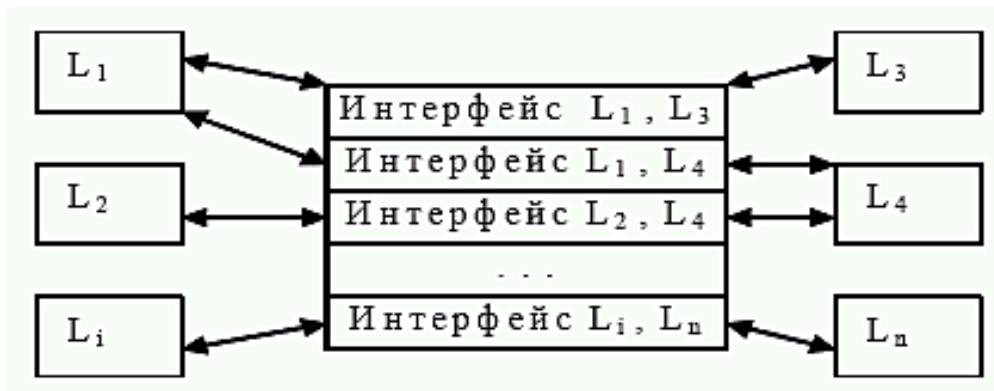


Рис. 3.18. Связь между языками  $L_1, L_2, \dots, L_n$  через интерфейсы

*Схема клиент-сервер – трехуровневая.* Главный вопрос этой схемы – доступ к ресурсам (аппаратуре, ПО и данным) и их разделение. При реализации архитектуры клиент-сервер сервер управляет

ресурсами и предоставляет к ним доступ, а клиент их использует. Архитектура основана на распределенных объектах, которые инкапсулируют ресурс и выдают услуги другим объектам.

Предоставляющие услуги объекты могут пользоваться тоже услугами других объектов. Функцию взаимодействия объектов выполняет *брокер объектных запросов (ORB)* через интерфейс клиент-сервер, он также предоставляет общесистемный сервис, услуги и различные ресурсы. Процесс разработки распределенных объектов начинается с формирования требований, проектирования объектов серверов, которые могут предоставлять услуги объектам клиента.

В качестве инструмента проектирования объектов применяется UML или унифицированный процесс RUP. Связи между объектами и их типами (операции и атрибуты) сервера и клиента задаются диаграммами классов. Взаимодействие объектов моделируется с помощью сценариев взаимодействия или *диаграмм последовательности*. Диаграммы состояний задают ограничения на операции к объектам сервера, преобразуются (генерируются) в интерфейсы (стабы), определяющие структуру и поведение объектов сервера (рис. 3.19).



Рис. 3.19. Процесс разработки распределенных объектов

Стаб клиента используется в классах, экземплярами которых являются объекты клиента. При реализации объектов сервера используется стаб, тип которого наследуется от типа серверного стаба. Интерфейсы описываются в языке IDL и размещаются в промежуточном слое системы CORBA. Стабы предоставляют операции и со-

ответствующие списки формальных параметров. При вызове клиент передает фактические параметры, которые соответствуют формальным параметрам. Объекты клиента и сервера – объекты стандартной модели архитектуры ОМА.

Сущность стиля проектирования в рамках *унифицированного процесса RUP* состоит в предоставлении всех видов деятельности, выполняемых на моделях (анализа, проектирования, разработки и тестирования) процесса ЖЦ.

Модели охватывают все аспекты построения системы, структуру и поведение. В состав архитектуры системы входят модели процессов, содержащие статические и динамические объекты, их связи и интерфейсы между ними. В ней отображаются структура выделенных подсистем, справочников, словарей, а также результаты всех процессов.

Логическая структура проектируемой системы – это композиция объектов и готовых программных продуктов, выполняющих соответствующие функции системы. Композиция основывается на следующих положениях:

- каждая подсистема должна отражать требования и способ их реализации (сценарий, прецедент, актер и т.п.);
- изменяемые функции выделяются в подсистемы так, чтобы для них прогнозировались изменения требований и отдельные объекты, связанные с актером;
- связь объектов осуществляется через интерфейс;
- каждая подсистема должна выполнять минимум услуг или функций и иметь фиксированное множество параметров интерфейса.

Результаты архитектурного проектирования представляются нотациями в виде *диаграмм (сущность-связь, переходы состояний, потоки данных и действий и т.п.)* в *модели анализа* требований. Объекты диаграмм детализируют заданные функциональные требования к разработке системы и отображают процесс решения задач проекта.

Выделенные в *модели анализа* объекты объединяются в систему путем:

- сборки объектов;
- логического объединения объектов;
- объединения по времени, т.е. сборки для заданного промежутка времени;
- коммуникативного объединения объектов через общий источник данных;

- процедурного объединения с помощью операторов вызова;
- функционального объединения объектов.

Если во вновь создаваемой системе используется унаследованная система, то она снимает проблему дублирования и сокращает объем работ при проектировании архитектуры системы.

В сложных программных системах количество выделенных объектов может насчитывать сотни, их композиции не будут иметь выразительного представления, даже с учетом того, что объекты разных сценариев могут совпадать, поэтому в таком случае требуется дополнительный анализ для их отождествления.

Техническое проектирование состоит в отображении архитектуры системы в среду функционирования путем привязки элементов системы к особенностям платформы реализации: СУБД, ОС, оборудование и др. Перенос изготовленной ПС на другую платформу требует изменения параметров, настройки сервисов к новым условиям среды и адаптации используемых БД.

Для реализации таких свойств определяются объекты, которые взаимодействуют с сервисами системы, относительно которых декларируется переносимость. Любой определенный таким образом объект заменяется объектом, который не взаимодействует непосредственно с сервисом, а с некоторым абстрактным объектом-посредником, который осуществляет трансформацию абстрактного интерфейса в интерфейс конкретного сервиса системы. Объект-посредник при этом обладает свойством настраиваться на конкретную среду.

Вместе с тем любой аспект перехода на новую платформу может потребовать построения вспомогательных интерфейсных или управляющих объектов и корректировки существующих. Более того, может возникнуть необходимость использования готовых подсистем, структура которых отличается от тех подсистем, которые были определены на основании анализа требований к системе. В этом случае вносятся соответствующие изменения в модель анализа требований и в архитектуру системы.

UML принят на вооружение почти всеми крупнейшими компаниями – производителями ПО (Microsoft, IBM, Hewlett-Packard, Oracle, Sybase и др.). Кроме того, почти все мировые производители CASE-средств, помимо IBM Rational Software, поддерживают UML в своих продуктах (Together (Borland), Paradigm Plus (Computer Associates), System Architect (Popkin Software), Microsoft Visual Modeler и др.). Полное описание UML можно найти на сайтах – URL: <http://www.omg.org> и <http://www.rational.com>.

Стандарт UML версии L1, принятый OMG в 1997 г., предлагает следующий набор диаграмм:

- Структурные (structural) модели:

– диаграммы классов (class diagrams) – для моделирования статической структуры классов системы и связей между ними (рис. 3.20);

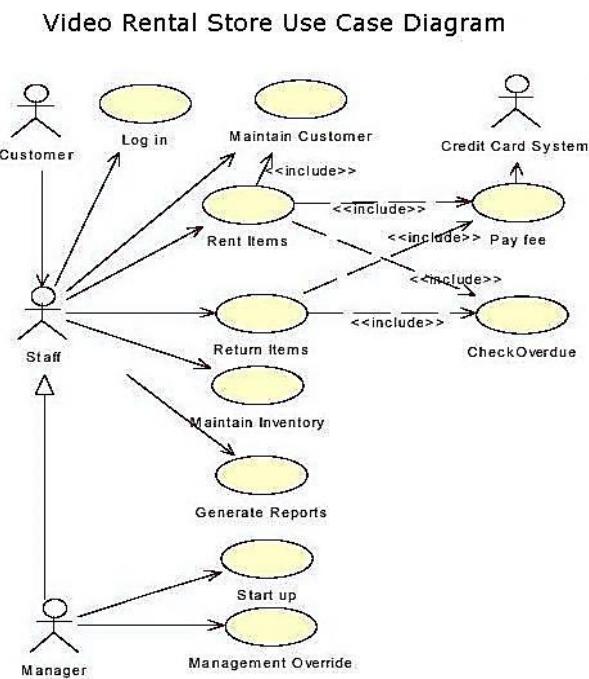


Рис. 3.20. Пример диаграммы классов

– диаграммы компонентов (component diagrams) – для моделирования иерархии компонентов (подсистем) системы (рис. 3.21);

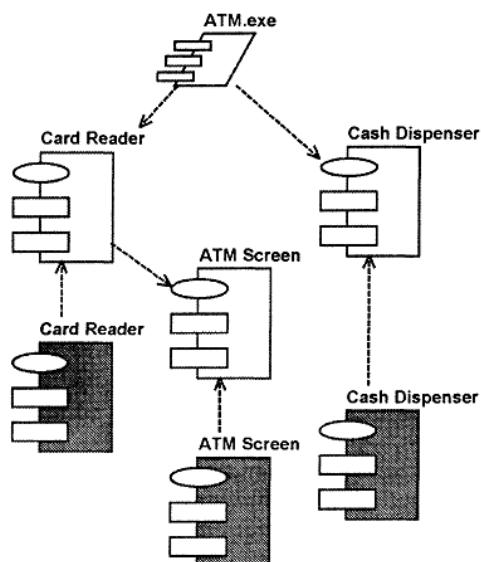


Рис. 3.21. Пример диаграммы компонентов для клиентской части системы

– диаграммы размещения (deployment diagrams) – для моделирования физической архитектуры системы (рис. 3.22).

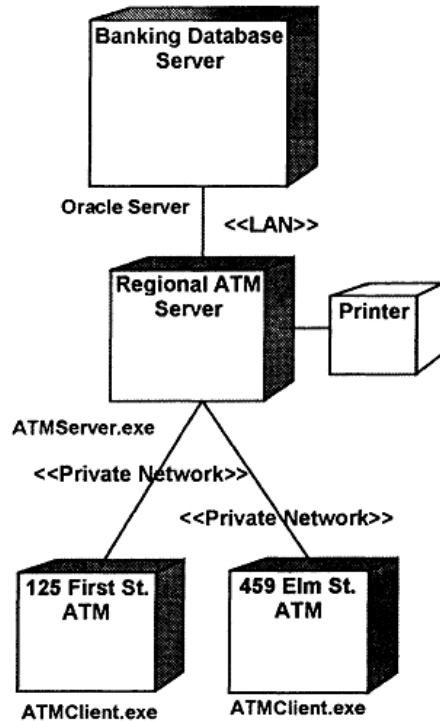


Рис. 3.22. Диаграммы размещения для банковской системы

- Модели поведения (behavioral):
  - диаграммы вариантов использования (use case diagrams) — для моделирования бизнес-процессов и функциональных требований к создаваемой системе (рис. 3.23);

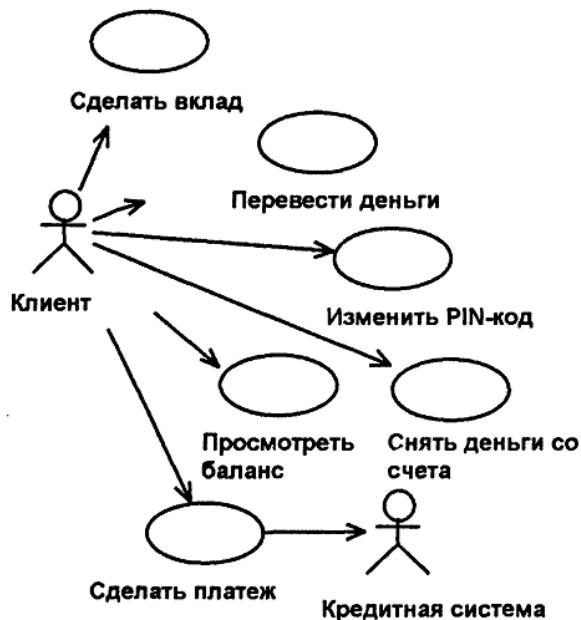


Рис. 3.23. Пример диаграммы вариантов использования

- диаграммы взаимодействия (interaction diagrams);
- диаграммы последовательности (sequence diagrams) (рис. 3.24) и кооперативные диаграммы (рис. 3.25) (collaboration diagrams) – для моделирования процесса обмена сообщениями между объектами;
- диаграммы состояний (statechart diagrams) – для моделирования поведения объектов системы при переходе из одного состояния в другое (рис. 3.26);

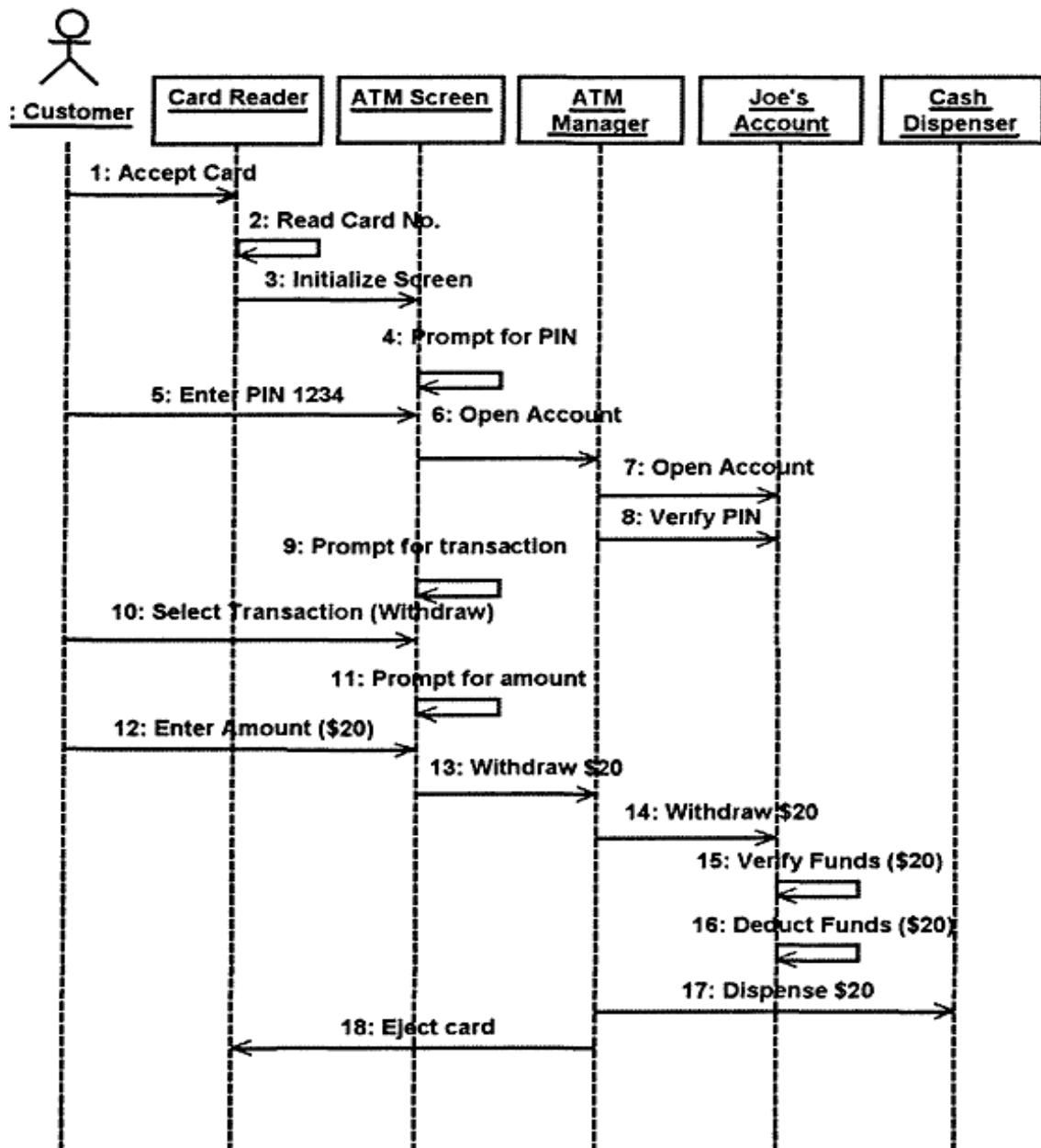


Рис. 3.24. Пример диаграммы последовательности

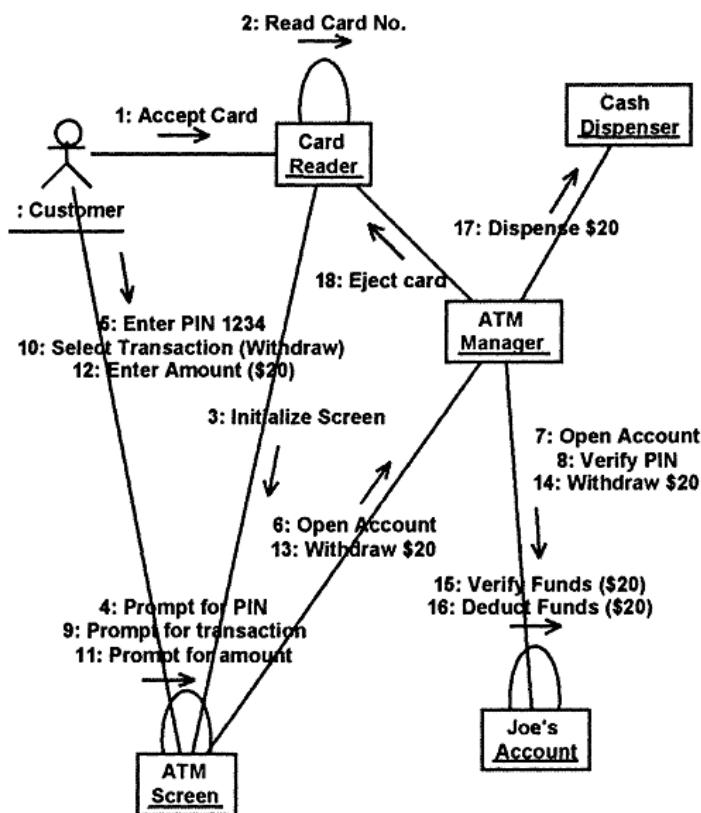


Рис. 3.25. Пример кооперативной диаграммы

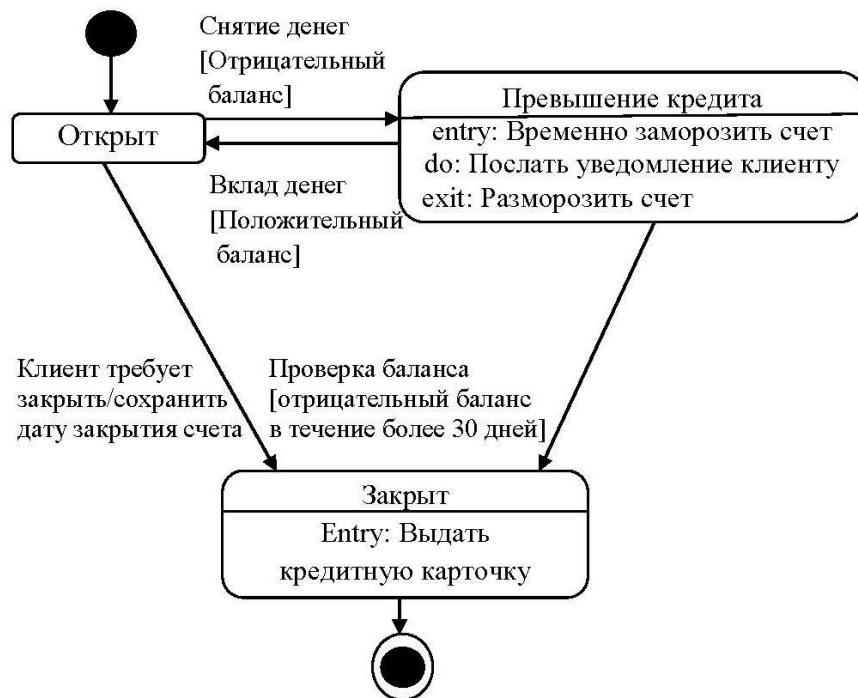


Рис.3.26. Диаграмма состояний для класса Accouont

– диаграммы деятельности (activity diagrams) – для моделирования поведения системы в рамках различных вариантов использования, или потоков управления (рис. 3.27).

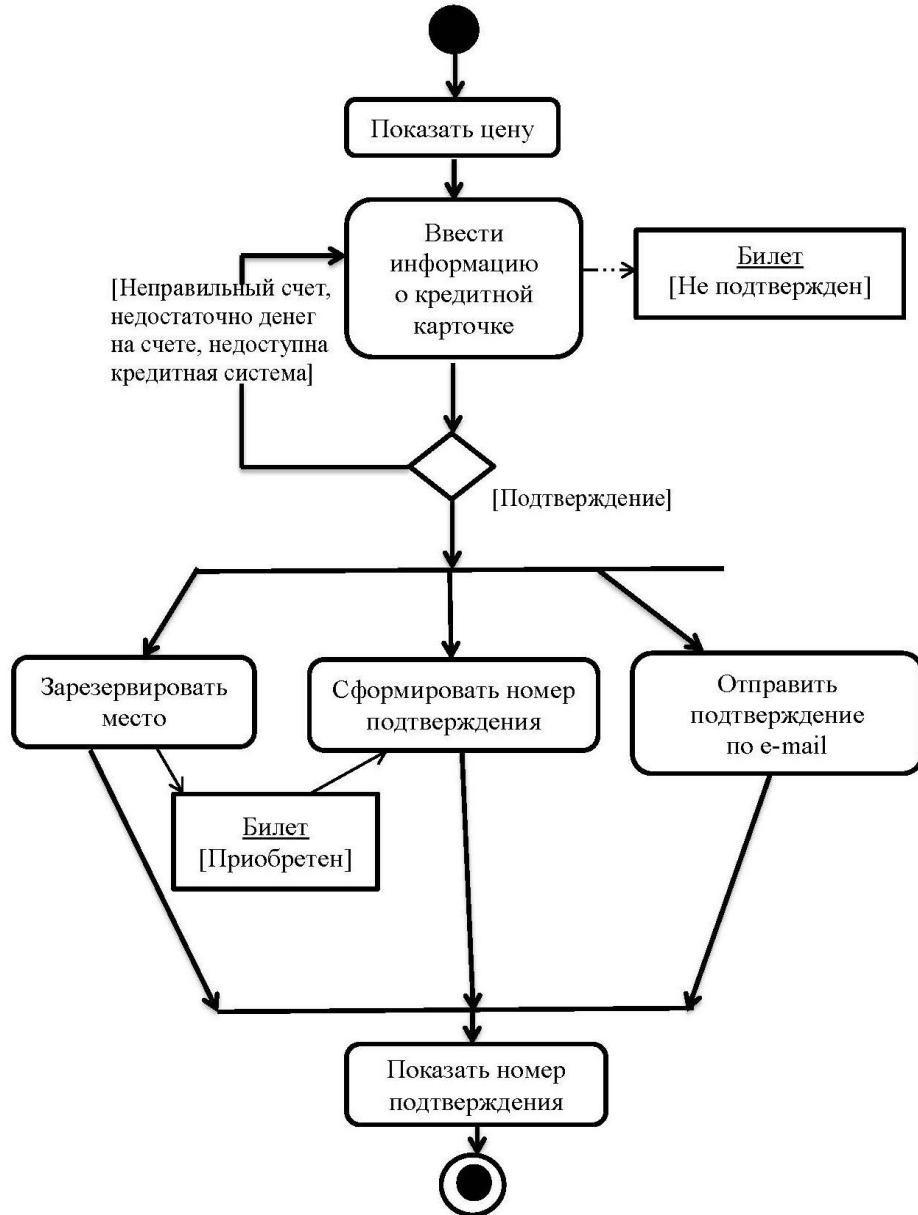


Рис. 3.27. Диаграмма деятельности

## **Глава 4. Методы систематического программирования**

*Объектно-ориентированный подход* (ООП) – стратегия разработки, в рамках которой разработчики системы вместо операций и функций мыслят *объектами*. Объект – это предмет внешнего мира, некоторая сущность, пребывающая в различных состояниях и имеющая множество операций. Операции задают сервисы, предоставляемые объектам для выполнения определенных вычислений, а состояние – это набор атрибутов объекта, поведение которых изменяет это состояние.

Объекты группируются в класс, который служит шаблоном для включения описания всех атрибутов и операций, связанных с объектами данного класса.

Программная система содержит взаимодействующие объекты, имеющие собственное локальное состояние и набор операций для определения состояний других объектов. Объекты скрывают информацию о представлении состояний и ограничивают к ним доступ.

Под *процессом* в ООП понимается проектирование классов объектов и взаимоотношений между ними (рис. 4.1).

Процесс разработки включает в себя следующие этапы:

- *анализ* – создание объектной модели (ОМ) ПрО, в которой объекты отражают реальные ее сущности и операции над ними;
- *проектирование* – уточнение ОМ с учетом описания требований для реализации – реализация ОМ средствами языков программирования C++, Java и др.;
- *сопровождение* конкретных задач системы;
- *программирование* – использование и развитие системы, внесение изменений как в состав объектов, так и в методы их реализации;
- *модификация ПС* – изменение системы в процессе ее сопровождения путем добавления новых функциональных возможностей, интерфейсов и операций.

Приведенные этапы могут выполняться итерационно друг за другом и с возвратом к предыдущему этапу. На каждом этапе может применяться одна и та же система нотаций.

Переход к следующему этапу приводит к усовершенствованию результатов предыдущего этапа путем более детальной реализации ранее определенных классов объектов и добавления новых классов.

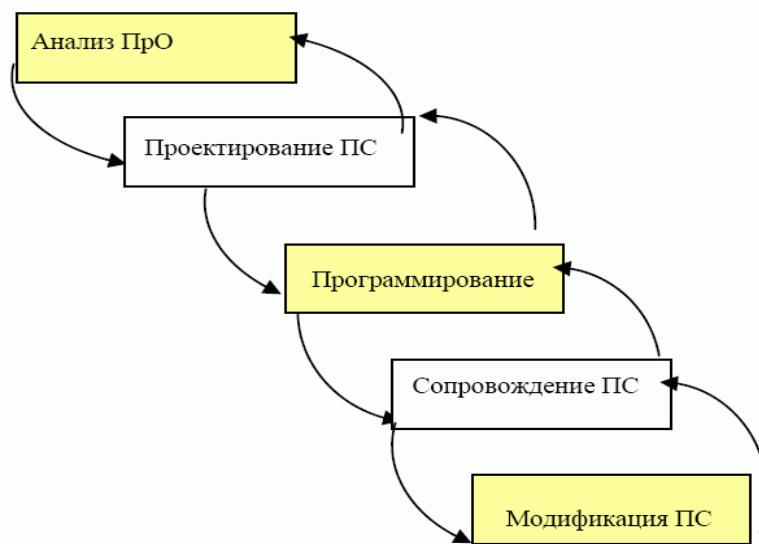


Рис. 4.1. ЖЦ разработки ПС в среде ООП

Результат процесса анализа ЖЦ – модель ПрО и набор других моделей (модель архитектуры, модель окружения и использования), полученных на этапах процесса ЖЦ. Модели отображают связи между объектами, их состояния и набор операций для динамического изменения состояния других объектов, а также взаимоотношения со средой. Объекты инкапсулируют информацию об их состоянии и ограничивают к ним доступ. Модели окружения и использования системы – это две взаимно дополняющие друг друга модели связи системы со средой.

Модель окружения системы – статическая модель, которая описывает другие подсистемы из пространства разрабатываемой ПС, а модель использования системы – динамическая модель, которая определяет взаимодействие системы со своей средой. Это взаимодействие определяется последовательностью запросов к сервисам объектов и ответных реакций системы после выполнения запроса. После определения взаимодействий между объектами проектируемой системы и ее окружением полученные данные используются для разработки архитектуры системы из объектов, созданных в предыдущих подсистемах и проектах.

Существует два типа моделей системной архитектуры:

- *статическая модель* описывает статическую структуру системы в терминах классов объектов и взаимоотношений между ними (обобщение, расширение, использование, структурные отношения);

- *динамическая модель* описывает динамическую структуру системы и взаимодействие между объектами во время выполнения

системы. Результат проектирования – это ПС, в которой определены все необходимые объекты статически или динамически с помощью классов и соответствующих методов реализации объектов. Полученная объектно-ориентированная система проверяется на показатели качества на основе результатов тестирования и сбора данных об ошибках и отказах системы. Такую систему можно рассматривать как совокупность автономных и независимых объектов. Изменение метода реализации объекта или добавление новых функций не влияет на другие объекты системы. Объекты могут быть повторно используемыми.

## 4.1. UML-метод моделирования

UML (United Modeling Language) – унифицированный язык моделирования является результатом совместной разработки специалистов программной инженерии и инженерии требований. Он широко используется ведущими разработчиками ПО как метод моделирования на этапах ЖЦ разработки ПС.

В основу метода положена парадигма объектного подхода, при котором концептуальное моделирование проблемы происходит в терминах взаимодействия объектов и включает:

- онтологию домена, которая определяет состав классов объектов домена, их атрибутов и взаимоотношений, а также услуг (операций), которые могут выполнять объекты классов;
- модель поведения задает возможные состояния объектов, инцидентов, инициирующих переходы с одного состояния к другому, а также сообщения, которыми обмениваются объекты;
- модель процессов определяет действия, которые выполняются при проектировании объектов как компонентов.

Модель требований в UML – это совокупность диаграмм, которые визуализируют основные элементы структуры системы.

Язык моделирования UML поддерживает статические и динамические модели, в том числе модель последовательностей – одну из наиболее полезных и наглядных моделей, в каждом узле которой – взаимодействующие объекты. Все модели представляются диаграммами, краткая характеристика которых дается ниже.

**Диаграмма классов** (*Class diagram*) отображает онтологию домена, эквивалентна структуре информационной модели метода С. Шлеера и С. Меллора, определяет состав классов объектов и их взаимоотношений. Диаграмма задается иконами, как визуальное

изображение понятий и связей между ними. Верхняя часть иконы – обязательная, она определяет имя класса. Вторая и третья части иконы определяют соответственно список атрибутов класса и *список операций* класса.

Атрибутами могут быть типы значений в UML:

- Public (общий) обозначает операцию класса, вызываемую из любой части программы любым объектом системы;
- Protected (защищенный) обозначает операцию, вызванную объектом того класса, в котором она определена или наследована;
- Private (частный) обозначает операцию, вызванную только объектом того класса, в котором она определена.

Пользователь может определять специфические для него атрибуты. Под операцией понимается сервис, который экземпляр класса может выполнять, если к нему будет произведен соответствующий вызов. Операция имеет название и список аргументов.

Классы могут находиться в следующих отношениях или связях.

*Ассоциация* – взаимная зависимость между объектами разных классов, каждый из которых является равноправным ее членом. Она может обозначать количество экземпляров объектов каждого класса, которые принимают участие в связи (0 – если ни одного, 1 – если один,  $N$  – если много).

*Зависимость* между классами, при которой класс-клиент может использовать определенную операцию другого класса; классы могут быть связаны отношением трассирования, если один класс трансформируется в другой в результате выполнения определенного процесса ЖЦ.

*Экземпляризация* – зависимость между параметризованным абстрактным классом-шаблоном (template) и реальным классом, который инициирует параметры шаблона (например, *контейнерные классы* языка C++).

### **Моделирование поведения системы**

*Поведение* системы определяется множеством обменивающихся сообщениями объектов и задается *диаграммами: последовательности, сотрудничества, деятельности и состояния*.

*Диаграмма последовательности* применяется для задания взаимодействия объектов с помощью сценариев, отображающих события, связанные с их созданием и уничтожением. Взаимодействие объектов контролируется событиями, которые происходят в сценарии и поддерживаются сообщениями к другим объектам.

*Диаграммы сотрудничества* задают поведение совокупности объектов, функции которых ориентированы на достижение целей системы, а также взаимосвязи тех ролей, которые обеспечивают сотрудничество.

*Диаграмма деятельности* задает поведение системы в виде определенных работ, которые может выполнять система или актер, виды работ могут зависеть от принятия решений в зависимости от заданных условий или ограничений. В качестве примера использования диаграммы деятельности UML приведена структура программы «Оплатить услуги» (рис. 4.2). Данная диаграмма демонстрирует программу расчета и оплаты услуг. В ней выполняется ряд последовательных действий по расчету стоимости за услуги.

В зависимости от выполнения условия «долга нет» происходит переход в конечное состояние или на разделение потоков на два параллельных. В левой ветви выполняется действие «послать уведомление об оплате» и «получить оплату», а в правой – «получить оплату». Распараллеливание означает, что пользователь может оплатить услуги, не дожидаясь уведомления. Параллельные потоки сливаются в один, затем снова ветвление алгоритма – условие «оплата не получена», «отключить услугу» и переход в конечное состояние.

*Диаграмма состояний* использует *расширенную модель* конечного автомата и определяет условия переходов, действия при входе и выходе из состояния, а также параллельно действующие состояния. Переход по списку данных инициирует некоторое событие. Состояние зависит от условий перехода, подобно тому, как взаимодействуют две параллельно работающие машины.

**Диаграмма реализации** состоит из диаграммы компонента и размещения.

Построение ПС методом UML состоит в выполнении этапов ЖЦ, приведенных на общей схеме реализации ПрО (рис. 4.3).

*Диаграмма компонента* отображает структуру системы как композицию компонентов и связей между ними. Диаграмма размещения задает состав *физических ресурсов* системы (узлов системы) и отношений между ними, к которым относятся необходимые аппаратные устройства, на которых располагаются компоненты, взаимодействующие между собой.

Пакет может быть элементом конфигурации построенной системы, на которую можно ссылаться в разных диаграммах.

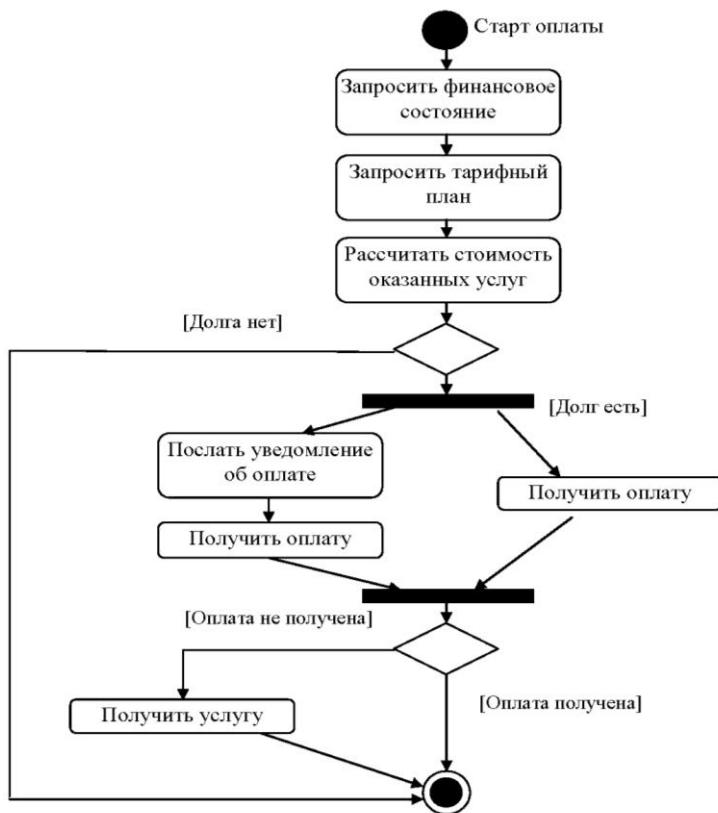


Рис. 4.2. Диаграмма программы расчета и оплаты услуг

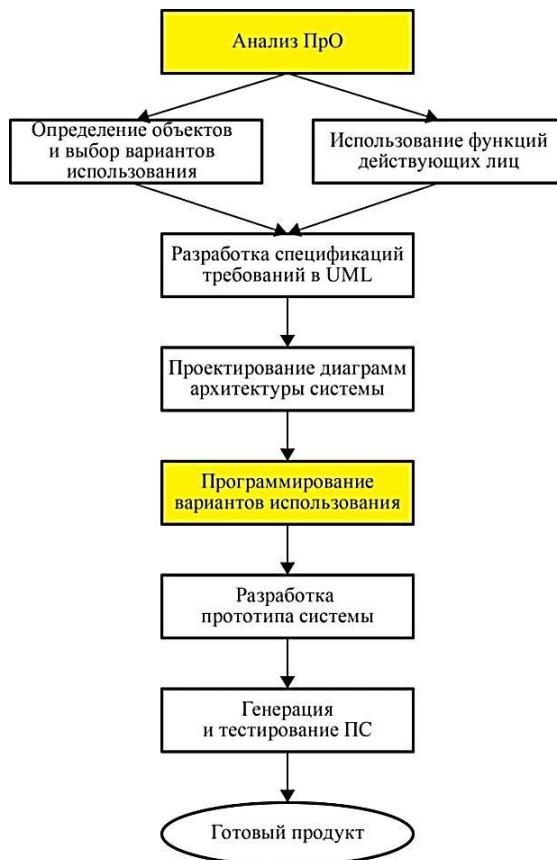


Рис. 4.3. Схема моделирования и проектирования ПС в UML

### 4.3. Компонентный подход

По оценкам экспертов, 75 % работ по программированию в информационном мире дублируются (например, программы складского учета, начисления зарплаты, расчета затрат на производство продукции и т.п.). Большинство из этих программ типовые, но каждый раз находятся особенности, которые влияют на их повторную разработку.

Компонентное проектирование сложных программ из готовых компонентов является наиболее производительным

Переход к компонентам происходил эволюционно: от подпрограмм, модулей, функций. При этом усовершенствовались элементы, методы их композиции и накопления для дальнейшего использования (табл. 4.1).

*Таблица 4.1*  
**Схема эволюции элементов компонентов**

Элемент композиции	Описание элемента	Схема взаимодействия	Представление, хранение	Результат композиции
Процедура, подпрограмма, функция	Идентификатор	Непосредственное обращение, оператор вызова	Библиотеки подпрограмм и функций	Программа
Модуль	Паспорт модуля, связи	Вызов модулей, интеграция модулей	Банк, библиотеки модулей	Программа с модульной структурой
Объект	Описание класса	Создание экземпляров классов, вызов методов	Библиотеки классов	Объектно-ориентированная программа
Компонент	Описание логики (бизнес), интерфейсов (API, IDL), схемы развертывания	Удаленный вызов в компонентных моделях (CQV1 CORBA, OSF,...)	Репозитарий компонентов, серверы и контейнеры компонентов	Распределенное компонентно-ориентированное приложение
Сервис	Описание бизнес-логики интерфейсов сервиса (XML, WSDL, ...)	Удаленный вызов (RPC, HLR, SOAP,...)	Индексация и каталогизация сервисов (XML, UDDI...)	Распределенное сервисо-ориентированное приложение

Компонентный подход дополняет и расширяет существующие подходы в программировании, особенно ООП. Объекты рассматриваются на логическом уровне проектирования ПС, а компоненты – это физическая реализация объектов. Компоненты конструируются

как некоторая абстракция, включающая в себя информационный раздел и артефакты (спецификация, код, контейнер и др.). В этом разделе содержатся сведения: назначение, дата изготовления, условия применения (ОС, среда, платформа и т.п.). Артефакт – это реализация (*implementation*), интерфейс (*interface*) и схема развертывания (*deployment*) компонента.

*Реализация* – это код, который будет выполняться при обращении к операциям, определенным в интерфейсах компонента. Компонент может иметь несколько реализаций в зависимости от операционной среды, модели данных, СУБД и др. Для описания компонентов, как правило, применяются языки объектно-ориентированной ориентации, а также язык JAVA, в котором понятие интерфейса и класса – базовые, используются в инструментах Javabeans и Enterprise Javabeans и в объектной модели CORBA.

*Интерфейс* отображает операции обращения к реализации компонента, описывается в языках IDL или APL, включает в себя описание типов и операции передачи аргументов и результатов для взаимодействия компонентов. Компонент как физическая сущность может иметь множество интерфейсов.

*Развертывание* – это выполнение физического файла в соответствии с конфигурацией (версией), параметрами настройки для запуска на выполнение компонента.

Компоненты наследуются в виде классов и используются в модели, композиции и в каркасе (Фреймворке) интегрированной среды. Управление компонентами проводится на архитектурном, компонентном или интерфейсном уровнях, между которыми существует взаимная связь. Компонент описывается в языке программирования, не зависит от операционной среды (например, от среды виртуальной машины JAVA) и от реальной платформы (например, от платформ в системе CORBA), где он будет функционировать.

**Типы компонентных структур.** Расширением понятия компонента является *шаблон* (паттерн) – абстракция, которая содержит описание взаимодействия совокупности объектов в общей кооперативной деятельности, для которой определены роли участников и их ответственности. Шаблон является повторяемой частью программного элемента как схема или взаимосвязь контекста описания для решения проблемы.

*Компонентная модель* отражает проектные решения по композиции компонентов, определяет типы шаблонов компонентов и допустимые между ними взаимодействия, а также является источ-

ником формирования файла развертывания ПС в среде функционирования.

*Каркас* представляет собой высокоуровневую абстракцию проекта ПС, в которой функции компонентов отделены от задач управления ими. Например, бизнес-логика – это функция компонента, а каркас – управление ими. Каркас объединяет множество взаимодействующих между собою объектов в некоторую интегрированную среду для решения заданной конечной цели. В зависимости от специализации каркас называют «белым или черным ящиком».

Каркас типа «белый ящик» включает абстрактные классы для представления цели объекта и его интерфейса. При реализации эти классы наследуются в конкретные классы с указанием соответствующих методов реализации. Использование такого типа каркаса является характерным для ООП.

Для каркаса типа «черный ящик» в его видимую часть выносятся точки, разрешающие изменять входы и выходы.

**Композиция компонентов** может быть следующих типов:

- *композиция компонент-компонент* обеспечивает непосредственное взаимодействие компонентов через интерфейс на уровне приложения;
- *композиция каркас-компонент* обеспечивает взаимодействие каркаса с компонентами, при котором каркас управляет ресурсами компонентов и их интерфейсами на системном уровне;
- *композиция компонент-каркас* обеспечивает взаимодействие компонента с каркасом по типу «черного ящика», в видимой части которого находится описание файла для развертывания и выполнения определенной функции на сервисном уровне;
- *композиция каркас-каркас* обеспечивает взаимодействие каркасов, каждый из которых может разворачиваться в гетерогенной среде и разрешать компонентам, входящим в каркас, взаимодействовать через их интерфейсы на сетевом уровне.

Компоненты и их композиции, как правило, запоминаются в репозитарии компонентов, а их интерфейсы – в репозитарии интерфейсов.

**Повторное использование** в компонентном программировании – это применение готовых порций formalизованных знаний, добытых во время реализации ПС, в новых разработках.

*Повторно используемые компоненты (ПИК)* – это готовые компоненты, элементы оформленных знаний (проектные решения, функции, шаблоны и др.) в ходе разработки, которые используются не только самими разработчиками, а и другими пользователями пул-

тем адаптации их к условиям новой ПС, что упрощает и сокращает сроки ее разработки. В системе Интернет в данный момент имеется много разных библиотек, репозитариев, содержащих ПИК, и их можно использовать в новых проектах.

При создании компонентов, ориентированных на повторное использование, их интерфейсы должны содержать операции, которые обеспечивают разные способы применения компонентов. Как и любые элементы производства, ПИК должны отвечать определенным требованиям, обладать характерными свойствами и структурой, а также иметь механизмы обращения к ним и др.

Главным преимуществом создания ПС из компонентов является уменьшение затрат на разработку за счет выбора готовых компонентов с подобными функциями, пригодными для практического применения и настройки их к новым условиям, на что тратится меньше усилий, чем на аналогичную разработку.

Поиск готовых компонентов основывается на методах классификации и каталогизации. Метод классификации предназначен для представления информации о компонентах с целью быстрого поиска и отбора, метод каталогизации – для физического их размещения в репозитариях с обеспечением доступа к ним в процессе интеграции.

**Методология компонентной разработки ПС.** Создание компонентной системы начинается с анализа ПрО и построения концептуальной модели, на основе которой создается компонентная модель (рис. 4.4), включающая проектные решения по композиции компонентов, использованию разных типов шаблонов, связей между ними и операции развертывания ПС в среде функционирования.

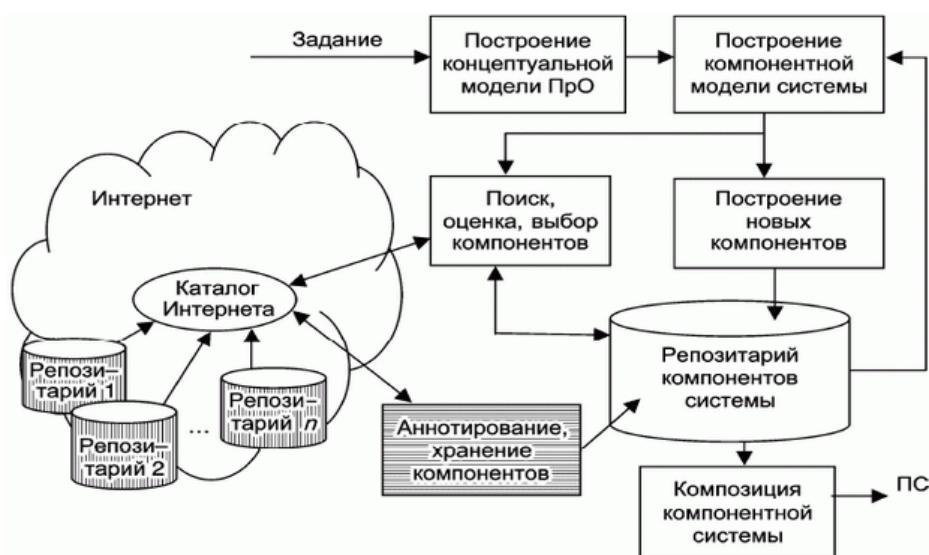


Рис. 4.4. Концептуальная схема построения ПС из компонентов в среде Интернет

Готовые компоненты берутся из репозитариев Интернета и используются при создании ПС, технология построения которых описывается следующими этапами ЖЦ.

1. *Поиск, выбор ПИК* и разработка новых компонентов, исходя из *системы классификации* компонентов и их каталогизации – это формализованное определение спецификаций интерфейсов, поведения и функциональности компонентов, а также их аннотирования и размещения в репозитарии системы или в Интернет.

2. *Разработка требований* (Requirements) к ПС – это формирование и описание функциональных, нефункциональных и других свойств ПС.

3. *Анализ поведения* (Behavioral Analysis) ПС заключается в определении функций системы, деталей проектирования и методов их выполнения.

4. *Спецификация интерфейсов и взаимодействий компонентов* (Interface and Interaction Specification) отражает распределение ролей компонентов, интерфейсов, их идентификацию и взаимодействие компонентов через поток действий (workflow).

5. *Интеграция набора компонентов и ПИК* (Application Assembly and Component Reuse) в единую среду основывается на подборе и адаптации ПИК, определении совокупности правил, условий интеграции и построении конфигурации каркаса системы.

6. *Тестирование компонентов и среды* (Component Testing) основывается на методах верификации и тестирования для проверки правильности как отдельных компонентов и ПИК, так и интегрированной из компонентов ПС.

7. *Развертывание* (System Deployment) включает оптимизацию плана компонентной конфигурации с учетом среды пользователя, развертку отдельных компонентов и создание целевой компонентной конфигурации для функционирования ПС.

8. *Сопровождения ПС* (System Support and Maintenance) состоит из анализа ошибок и отказов при функционировании ПС, поиска и исправления ошибок, повторного ее тестирования и адаптации новых компонентов к требованиям и условиям интегрированной среды.

#### **4.4. Аспектно-ориентированное программирование**

*Аспектно-ориентированное программирование* (АОП) – это парадигма построения гибких к изменению ПС путем добавления новых аспектов (функций), обеспечивающих безопасность, взаимо-

действие компонентов с другой средой, а также синхронизацию одновременного доступа частей ПС к данным и вызов новых общесистемных средств.

Аспектом может быть ПИК, фрагмент программы, реализующий концепцию взаимодействия компонентов в среде, защиту данных и др. ПС, которая создается из ПИК, объектов, небольших методов и аспектов, дополняется необходимыми фрагментами взаимодействия, синхронизации, защиты и т.п. путем встраивания их в точки компонентов ПС, где они необходимы. В результате встроенные фрагменты дополняют компоненты новым содержательным аспектом и тем самым значительно усложняют процесс вычислений.

Практическая реализация аспектов, размещенных в разных частях элементов ПС, обеспечивается механизмом перекрестных ссылок и точками соединения, через которые осуществляется связь с аспектным фрагментом для получения определенной дополнительной функции.

В основе АОП лежит метод разбиения задач ПрО на ряд функциональных компонентов, определения необходимости применения разного рода дополнительных аспектов и установления точек расположения аспектов в отдельных компонентах, где это требуется. Эти работы выполняются на этапах ЖЦ процесса разработки, способствуют реализации ПС с ориентацией на взаимодействие компонентов или их синхронизацию. Такой подход известен при проведении отладки программ, когда фрагменты отладочных программ встраиваются в отдельные точки исходной программы для выдачи промежуточных результатов. Когда отладка завершается успешно, эти участки удаляются. В случае аспектов – их программные фрагменты остаются в программе.

Создание конечного продукта ПС в АОП выполняется по технологии, соответствующей разработке компонентных систем, с той особенностью, что используемые аспекты определяют особые условия выполнения компонентов в среде взаимодействия. Аспекты можно рассматривать как выполнение разных ролей взаимодействующими лицами. Это приближает аспект к роли программного агента, выполняющего дополнительные функции при определении архитектуры системы, и обеспечивает повышение качества компонентов.

Для использования аспектов при выработке проектных решений используется механизм фильтрации входных сообщений, с помощью которых проводится изменение параметров и имен текстов аспектов в конкретно заданном компоненте системы. Код компо-

нента становится «нечистым», когда он пересечен аспектами, и при композиции с другими компонентами общие средства (вызов процедур, RPC, RMI, IDL и др.) становятся недостаточными. Аспекты требуют декларативного сцепления описаний, а фрагменты находятся или берутся из различных объектов. Один из механизмов композиции компонентов и аспектов – фильтр композиции, который обновляет аспекты без изменения функциональных возможностей. Фактически фильтрация касается входных и выходных параметров сообщений, которые переопределяют соответствующие имена объектов. Иными словами, фильтры делегируют внутренним частям компонентов параметры, переадресовывая ранее установленные ссылки, проверяют и размещают в буфере сообщений, локализуют ограничения и готовят компонент для выполнения.

В ОО-программах могут быть методы, выполняющие дополнительно некоторые расчеты с обращением на другие методы внешнего уровня. Деметр сформулировал закон, согласно которому длинные последовательности мелких методов не должны выполняться. В результате создается код алгоритма с именами классов, не задействованных в расчетных операциях, а также дополнительный класс, который расширяет код этими расчетами.

С точки зрения моделирования аспекты можно рассматривать как каркасы декомпозиции системы, в которых отдельные аспекты пересекают ряд многократно используемых ПИК (рис. 4.5).

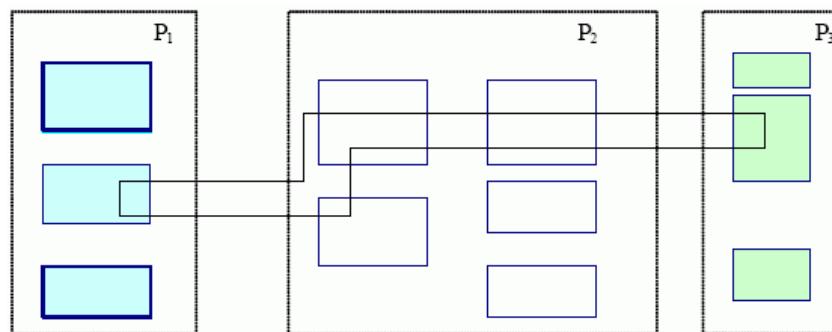


Рис. 4.5. Пример расположения аспектов в программах P1, P2 и P3

Разным аспектам проектируемой системы могут отвечать и разные парадигмы программирования: объектно-ориентированные, структурные и др. Они по отношению к проектируемой ПрО образуют мультипарадигмовую концепцию обработки, такую как синхронизация, взаимодействие, обработка ошибок и др. со значительными доработками процессов их реализации. Кроме того, этот механизм позволяет устанавливать аспектные связи с другими пред-

метными областями в терминах родственных областей. Языки АОП позволяют описывать аспекты для разных ПрО. В процессе компиляции пересекаемые аспекты объединяются, оптимизируются, генерируются и выполняются в динамике.

Существенной особенностью АОП является построение модели, которая пересекает структуру другой модели, для которой первая модель является аспектом. Так как аспект связан с моделью, то ее можно перестроить так, чтобы аспект стал, например, модулем и выполнял функцию посредника, реализуя шаблоны взаимодействия. Один из недостатков – пересечение отдельных компонентов аспектами может привести к понижению эффективности их выполнения.

Переплетение аспектов с компонентами может проявиться на последующих этапах процесса разработки, и поэтому требуется минимизация количества сцеплений между аспектами и компонентами через ссылки в вариантах использования, сопоставление с шаблоном или блоком кода, в котором установлены перекрестные ссылки.

В ходе анализа ПрО и построения ее характеристической модели устанавливается связь с дополнительными аспектами, что приводит к статическому или «жесткому» связыванию компонентов и аспектов модели, учету этого случая при компиляции.

Аспекты с точки зрения моделирования можно рассматривать как каркасы декомпозиции системы с многократным использованием. АОП становится мультипарадигмовой концепцией, сущность которой состоит в том, что разным аспектам проектируемой ПС должны отвечать разные парадигмы программирования. Каждая из парадигм относительно реализации разных аспектов ПС (синхронизации, внедрения, обработки ошибок и др.) требует их усовершенствования и обобщения для каждой новой ПрО.

В АОП используется модель модульных расширений в рамках метамодельного программирования, которая обеспечивает оперативное использование новых механизмов композиции отдельных частей ПС или их семейств с учетом предметно-ориентированных возможностей языков (например, SQL) и каркасов, которые поддерживают аспекты. Технология разработки прикладной системы с использованием АОП включает общие этапы (рис. 4.6):

- декомпозиция функциональных задач с условием многоразового применения модулей и выделенных аспектов, т.е. свойств их выполнения (параллельно, синхронно, безопасно и т.д.);
- анализ языков спецификации аспектов и определение конкретных аспектов для обеспечения взаимодействия, синхронизации и других задач ПрО;

- определение точек встраивания аспектов в компоненты и формирование ссылок и связей с другими элементами;
- разработка фильтров и описание связей аспектов с функциональными компонентами, выделенными в ПрО, отображение фильтров в модели EJB на стороне сервера и управление данными с обеспечением безопасности, защиты доступа к некоторым данным;
- определение механизмов композиции (вызовов процедур, методов, сцеплений) функциональных модулей многоразового применения и аспектов в точках их соединения, как фрагментов свойств управления выполнением этих модулей, или ссылок из этих точек на другие модули;
- создание объектной или компонентной модели, дополнение ее входными и выходными фильтрами сообщений, посылающих объектам ссылки, задания на выполнение методов или аспектов;
- анализ библиотеки расширений для выбора некоторых функциональных модулей, необходимых для реализации задач ПрО;
- компиляция, совместная отладка модулей и аспектов, после чего композиция их в готовый программный продукт.

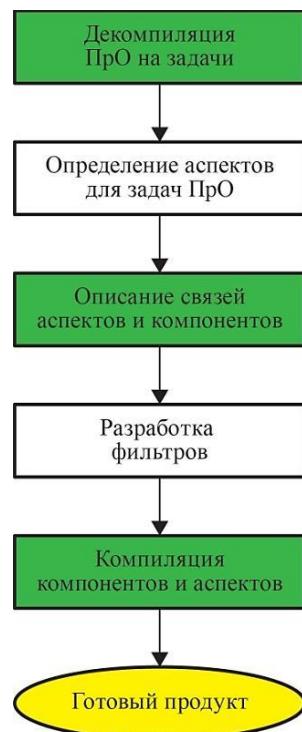


Рис. 4.6. Технологическая схема проектирования ПС средствами АОП

В процессе создания ПС с применением аспектов используются IP-библиотека расширений, активные библиотеки, Smalltalk и ЯП, расширенные средства описания аспектов.

*IP*-библиотека содержит функции компиляторов, средства оптимизации, редактирования, отображения и др. Например, библиотека матриц для вычисления выражений с массивами, предоставляющая память и др., получила название библиотеки генерирующего типа.

Иной вид библиотек АОП – *активные библиотеки*, которые содержат не только базовый код реализации понятий ПрО, но и целевой код обеспечения оптимизации, адаптации, визуализации и редактирования. Активные библиотеки пополняются средствами и инструментами *интеллектуализации* агентов, с помощью которых обеспечивается разработка специализированных агентов для реализации конкретных задач ПрО.

## **4.5. Генерирующее (порождающее) программирование**

Генерирующее программирование (*generate programming*) – генерация семейств приложений из отдельных элементов компонентов, аспектов, сервисов, ПИК, каркасов и т.п. Базис этого программирования – ООП, дополненный механизмами генерации ПИК, другими многоразовыми элементами, а также свойствами их изменчивости, взаимодействия и др.

В нем используются разные методы программирования для поддержки инженерии ПрО как дисциплины инженерного проектирования семейств ПС из разных ранее изготовленных продуктов путем объединения технологии генерации как отдельных ПС, так и их семейств. Эта дисциплина использует методы программирования, соответствующие формализмы и модели для создания более качественных представителей семейства ПС по принципу конвейера.

Главный элемент ПС – это семейство ПС или конкретные его экземпляры, которые генерируются на основе общей генерирующей модели домена (*generative domain model*), включающей в себя средства определения членов (представителей) семейства, методы сборки членов семейства и базу конфигурации с набором правил развертывания в операционной среде.

Каждый член семейства создается путем интеграции отдельных компонентов, планирования, контроля и оценки результатов *интеграционного тестирования*, а также определения затрат на применение многократно используемых ПИК, в том числе из активной библиотеки.

Базовый код элементов активной библиотеки содержит целевой код по обеспечению процедур компиляции, отладки, визуализации и др. Фактически компоненты этих библиотек – это интеллектуальные агенты, генерирующие новых агентов в расширяемой среде программирования для решения конкретных задач ПрО. Эта среда содержит специальные метапрограммы и компоненты библиотек для осуществления отладки, проверки композиции и взаимодействия компонентов. Среда пополняется новыми сгенерированными компонентами для членов семейства, в качестве компонентов многоразового применения.

Реализация целей порождающего программирования по включению ПИК в другие члены семейства проводится по двум сформировавшимся инженерным направлениям:

- *прикладная инженерия* – процесс производства конкретных ПС из ПИК, ранее созданных в среде независимых систем, или как отдельные элементы процесса инженерии некоторой ПрО;
- *инженерия ПрО* – построение членов семейства или самого семейства систем путем сбора, классификации и фиксации ПИК в качестве их конструктивных элементов, а также для частей систем для конкретной ПрО. Поиск, адаптация ПИК и внедрение их в новые члены семейства ПС проводятся с помощью специальных инструментальных средств типа репозитария.

Составная часть инженерии ПрО – инженерия приложений как способ создания отдельных целевых членов семейства для конструирования из этих членов новых ПИК, многократно используемых проектных решений и генерируемых как системы семейства ПрО.

Основные этапы инженерии ПрО приведены на рис. 4.7:

- анализ ПрО и выявление объектов и отношений между ними;
- определение области действий объектов ПрО;
- определение общих функциональных и изменяемых характеристик, построение модели, устанавливающей зависимость между различными членами семейства;
- создание базиса для производства конкретных программных членов семейства с механизмами изменчивости независимо от средств их реализации;
- подбор и подготовка компонентов многоразового применения, описание аспектов выполнения задач ПрО;
- генерация отдельного домена, члена семейства и ПС.

Генерация доменной модели для семейства ПС основывается на модели характеристик, наборе компонентов реализации задач

ПрО, совокупности компонентов и их спецификациях. Результат генерации – готовая подсистема или отдельный член семейства.

К рассмотренной схеме инженерии ПрО также относятся:

– корректировка процессов при включении новых проектных решений или при изменении состава ПИК;

– моделирование изменчивости и зависимостей с помощью механизмов изменения моделей (объектных, взаимодействия и др.), добавления новых требований и понятий, а также фиксации их в модели характеристик и в конфигурации системы;

– разработка инфраструктуры ПИК – описание, хранение, поиск, оценивание и объединение готовых ПИК.

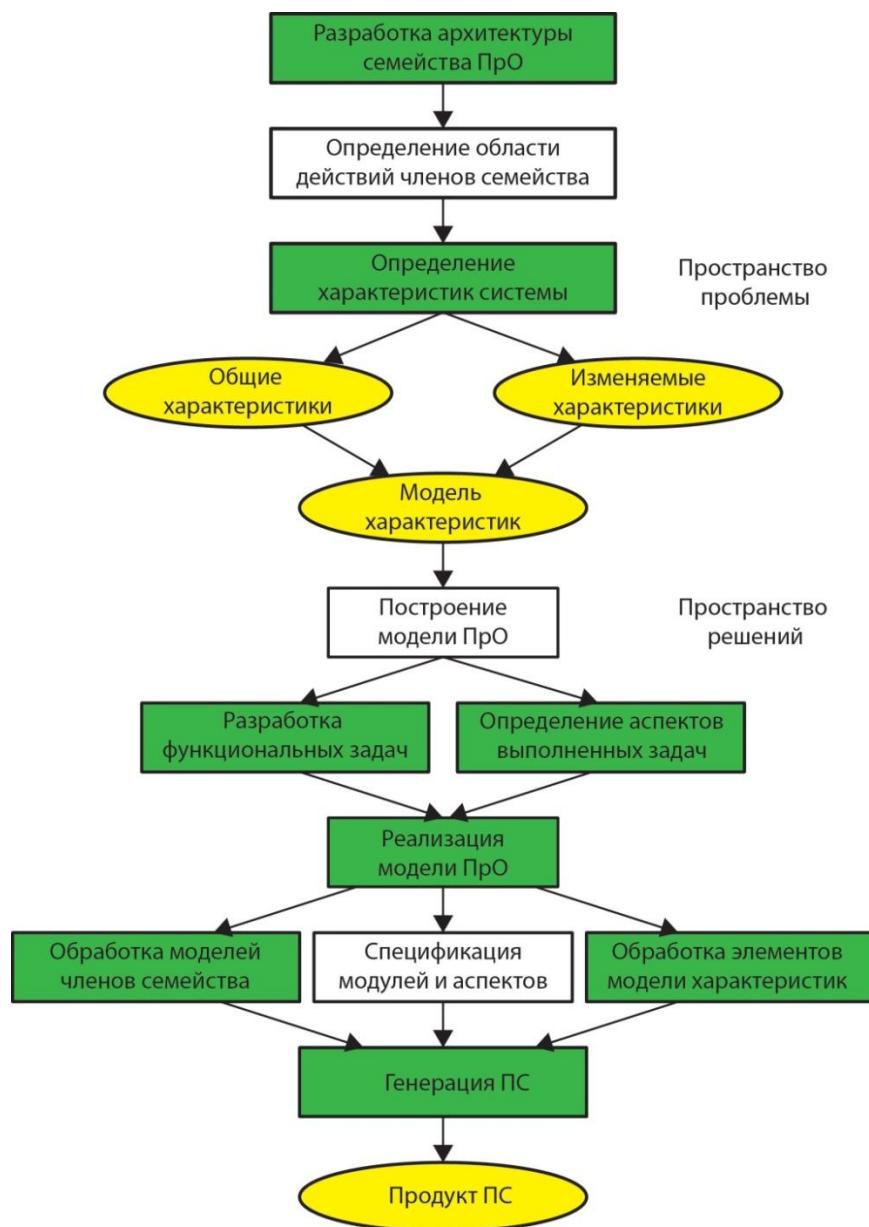


Рис. 4.7. Технологическая схема инженерии ПрО

ЖЦ разработки с повторным или многократным использованием обеспечивает получение семейства систем, определение области их действия, а также определение общих и изменяемых характеристик представителей семейства, заданных в модели характеристик. При их определении используются пространство проблемы и пространство решений.

*Пространство проблемы* (space problem) – компоненты семейства системы, в которых используются ПИК, объекты, аспекты и др., процесс разработки которых включает в себя системные инструменты, а также созданные в ходе разработки ПрО. Инженерия ПрО объединяет в модели характеристик функциональные характеристики, свойства выполнения компонентов, изменяемые параметры разных частей семейства, а также решения, связанные с особенностями взаимодействия групп членов семейства ПС.

Инженерия ПрО обеспечивает не только разработку моделей членов семейства (подсистем), но и моделирование понятий ПрО, модель характеристик для подсистем и набора компонентов, реализующих задачи ПрО. В рамках инженерии ПрО используются горизонтальные и вертикальные типы компонентов в терминологии системы CORBA.

К горизонтальным типам компонентов отнесены общие системные средства, которые нужны разным членам семейства, а именно: графические пользовательские интерфейсы, СУБД, системные программы, библиотеки расчета матриц, контейнеры, каркасы и т.п.

К вертикальным типам компонентов относятся прикладные системы (медицинские, биологические, научные и т.д.), методы инженерии ПрО, а также компоненты горизонтального типа по обслуживанию архитектуры многократного применения компонентов и их интерфейсов и др.

*Пространство решений* (space solution) – компоненты, каркасы, шаблоны проектирования ПрО, а также средства их соединения или встраивания в ПС и оценки избыточности. Элементы пространства реализуют решение задач этой ПрО. Каркас оснащен механизмом изменения параметров модели, которые требуют избыточную фрагментацию «множество мелких методов и классов». Шаблоны проектирования обеспечивают создание многократно используемых решений в различных типах ПС. Для задания и реализации таких аспектов, как синхронизация, удаленное взаимодействие, защита данных и т.п., применяются технологии ActiveX и JavaBeans, а также новые механизмы композиции и др.

Примером систем поддержки инженерии ПрО и реализации горизонтальных методов является система DEMRAL, предназначенная для разработки библиотек: численного анализа, распознавания речи, графовых вычислений и т.д. Основные виды элементов этой библиотеки – абстрактные типы данных (*abstract data types – ADT*) и алгоритмы. DEMRAL позволяет моделировать характеристики ПрО и представлять их в характеристической модели и *предметно-ориентированных языках* описания конфигурации.

Система конструирования RSEB в среде генерирующего программирования использует методы, относящиеся к вертикальным методам, а также ПИК и Use Case при проектировании больших ПС. Методы вертикального типа вызывают различные горизонтальные методы, относящиеся к разным прикладным подсистемам. При работе над отдельной частью семейства могут применяться аспекты взаимодействия, структуры, потоков данных и др. Важную роль при этом выполняют графический пользовательский интерфейс и метод обеспечения взаимодействия компонентов в распределенных средах (например, в CORBA).

## 4.6. Агентное программирование

Понятие интеллектуального и программного агентов появилось более 20 лет назад, их роль в программной инженерии все время возрастает. Так, Джекобсон отметил перспективу использования агентов в качестве менеджеров проектов, разработчиков архитектуры с помощью диаграмм use case и др.

Основной теоретический базис данного программирования – темпоральная, модальная и мультимодельная логики, дедуктивные методы доказательства правильности свойств агентов и др.

С точки зрения программной инженерии агент – это самодостаточная программа, способная управлять своими действиями в информационной среде функционирования для получения результатов выполнения поставленной задачи и изменения текущего состояния среды. Агент обладает следующими свойствами:

- автономность – это способность действовать без внешнего управляющего воздействия;
- реактивность – это способность реагировать на изменения данных и среды и воспринимать их;
- активность – это способность ставить цели и выполнять заданные действия для достижения этой цели;
- способность к взаимодействию с другими агентами (или людьми).

Основными задачами программного агента являются:

- самостоятельная работа и контроль своих действий;
- взаимодействие с другими агентами;
- изменение поведения в зависимости от состояния внешней среды;
- выдача достоверной информации о выполнении заданной функции и т.п.

С интеллектуальным агентом связаны знания типа убеждение, намерение, обязательства и т.п. Эти понятия входят в концептуальную модель и связываются между собой операционными планами реализации целей каждого агента. Для достижения целей интеллектуальные агенты взаимодействуют друг с другом, устанавливают связь между собой через сообщения или запросы и выполняют заданные действия или операции в соответствии с имеющимися знаниями.

Агенты могут быть локальными и распределенными (рис. 4.8). Процессы локальных агентов протекают в клиентских серверах сети, выполняют заданные функции и влияют на общее состояние среды функционирования. Распределенные агенты располагаются в разных узлах сети, выполняют автономно (параллельно, синхронно, асинхронно) предназначенные им функции и могут влиять на общее состояние распределенной среды.

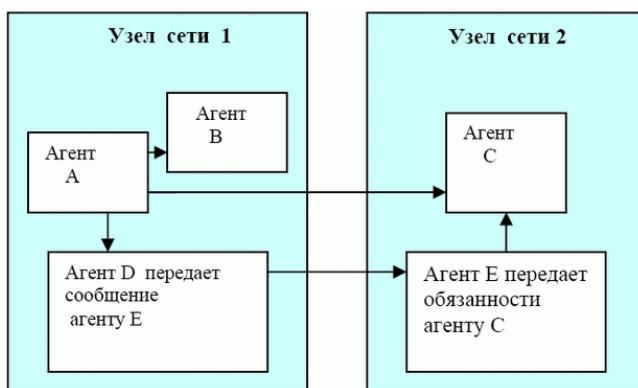


Рис. 4.8. Пример взаимодействия агентов в разных средах

Характер взаимодействия между агентами зависит от совместности целей, компетентности и т.п.

Основу агентно-ориентированного программирования составляют:

- формальный язык описания ментального состояния агентов;
- язык спецификации информационных, временных, мотивационных и функциональных действий агента в среде функционирования;

- язык интерпретации спецификаций агента;
- инструменты конвертирования любых программ в соответствующие агентные программы.

Агенты взаимодействуют между собой с помощью разных механизмов, а именно координации, коммуникации, кооперации или коалиции.

Под *координацией агентов* понимается процесс обеспечения последовательного функционирования при согласованности их поведения и без взаимных конфликтов. Координация агентов определяется:

- взаимозависимостью целей других агентов-членов коалиции, а также от возможного влияния агентов друг на друга;
- ограничениями, которые принимаются для группы агентов коалиции в рамках общего их функционирования;
- компетенцией – знаниями условий среды функционирования и степени их использования.

Главное средство коммуникации агентов – транспортный протокол TCP/IP или протокол агентов ACL (Agent Communication Languages). Управление агентами (Agent Management) выполняется с помощью сервисов: передача сообщений между агентами, доступ агента к серверу и т.п. *Коммуникация* агентов базируется на общем протоколе, языке HTML и декларативном или процедурном (Java, Telescript, ACL и т.п.) языке описания этого протокола.

Примером активной и скоординированной деятельности агентов по поиску необходимой информации является среда Интернет. В нем агенты обеспечивают доступ к информационным ресурсам, а также выполняют ее анализ, интеграцию, фильтрацию и передачу результата запроса пользователю.

Каждый агент выполняет определенную функцию, передает друг другу задание на последующее действие по доступу к информационному ресурсу, извлечению необходимой информации и передачи ее для обработки следующим агентам. При этом могут возникать нерегулярные состояния (тупики, отсутствие ресурса и др.).

Одной из систем построения агентов, основанной на обмене сообщениями в ACL, является JATLite, которая с помощью Java-классов создает новых агентов, вычисляющих определенные функции в распределенной среде. Система Agent Builder предназначена для конструирования программных агентов, которые описываются в языке Java и могут взаимодействовать на языке *KQML* (Knowledge Guery and Manipulation Language).

Построенные агенты выполняют функции менеджера проекта и онтологий, визуализации, отладки и др. Реализацию механизмов

взаимодействий агентов обеспечивают система JAFMAS, ряд других мультиагентных систем.

Современные направления в области проверки правильности программ – формальные спецификации и методы доказательства их правильности. Для доказательства того, что *спецификация программы* задает правильное решение некоторой задачи, для которой она разработана, привлекается математический аппарат.

В формальных методах нет рутинного написания спецификации на ЯП, а есть анализ текста и описание поведения программы в стиле, близком математической нотации, путем рассуждений и доказательств, принятых в математике. Формальные методы в программировании появились одновременно с самим программированием, на которое повлияли работы по теории алгоритмов А. А. Маркова, А. А. Ляпунова, схемы Ю. И. Янова, формальные нотации языка описания взаимодействующих процессов К. А. Хоара и др.

В 70-х гг. прошлого столетия появились формальные спецификации, которые близки ЯП и предоставляют средства, облегчающие проводить рассуждение о свойствах формальных тестов и сближающие их с математической нотацией. Несмотря на это, исследования формальных методов носили в основном академический, теоретический характер, поскольку извлечь из них практическую пользу в программировании не удавалось в силу огромных затрат на формальную спецификацию программ и разработку дополнительных аксиом, утверждений и условий, называемых предварительными условиями (предусловиями) и постусловиями, определяющими заключительные правила получения правильного результата.

Под спецификацией понимается формальное описание функций и данных программы, с которыми эти функции оперируют. Различают видимые данные, т.е. входные и выходные параметры, а также скрытые данные, которые не привязаны к реализации и определяют интерфейс с другими функциями.

Предусловия – это ограничения на совокупность входных параметров; постусловия – ограничения на выходные параметры. Предусловие и постусловие задаются предикатами, т.е. функциями, результатом которых будет булева величина (true/false). Предусловие истинно тогда, когда входные параметры входят в область допустимых значений данной функции. Постусловие истинно тогда, когда совокупность значений удовлетворяет требованиям, задающим формальное определение критерия правильности получения результата.

# **Глава 5. Промышленные технологии ППО**

## **5.1. Методология DATARUN**

Одной из наиболее распространенных в мире электронных методологий является методология DATARUN. В соответствии с методологией DATARUN ЖЦ ПО разбивается на стадии, которые связываются с результатами выполнения основных процессов, определяемых стандартом ISO 12207. Каждую стадию, кроме ее результатов, должен завершать план работ на следующую стадию.

Стадия формирования требований и планирования включает в себя действия по определению начальных оценок объема и стоимости проекта. Должны быть сформулированы требования и экономическое обоснование для разработки ИС, функциональные модели (модели бизнес-процессов организации) и исходная концептуальная модель данных, которые дают основу для оценки технической реализуемости проекта. Основными результатами этой стадии должны быть модели деятельности организации (исходные модели процессов и данных организации), требования к системе, включая требования по сопряжению с существующими ИС, исходный бизнес-план.

Стадия концептуального проектирования начинается с детального анализа первичных данных и уточнения концептуальной модели данных, после чего проектируется архитектура системы. Архитектура включает в себя разделение концептуальной модели на обозримые подмодели. Оценивается возможность использования существующих ИС и выбирается соответствующий метод их преобразования. После построения проекта уточняется исходный бизнес-план. Выходным компонентом этой стадии является концептуальная модель.

На стадии спецификации приложений данных модель архитектуры системы и уточненный бизнес-план совершенствуются, продолжается процесс создания и детализации проекта. Концептуальная модель данных преобразуется в реляционную модель данных. Определяются структура приложения, необходимые интерфейсы приложения в виде экранов, отчетов и пакетных процессов вместе с логикой их вызова. Модель данных уточняется бизнес-правилами и методами для каждой таблицы. В конце этой стадии принимается окончательное решение о способе реализации приложений. По результатам стадии должен быть построен проект ИС, включающий

модели архитектуры ИС, данных, функций, интерфейсов (с внешними системами и с пользователями), требований к разрабатываемым приложениям (модели данных, интерфейсов и функций), требований к доработкам существующих ИС, требований к интеграции приложений, а также сформирован окончательный план создания ИС.

На стадии разработки, интеграции и тестирования должна быть создана тестовая база данных, частные и комплексные тесты. Проводятся разработка, прототипирование и тестирование баз данных и приложений в соответствии с проектом. Отлаживаются интерфейсы с существующими системами. Описывается конфигурация текущей версии ПО. На основе результатов тестирования проводится оптимизация базы данных и приложений. Приложения интегрируются в систему, проводится тестирование приложений в составе системы и испытания системы. Основными результатами стадии являются готовые приложения, проверенные в составе системы на комплексных тестах, текущее описание конфигурации ПО, скорректированная по результатам испытаний версия системы и эксплуатационная документация на систему.

Стадия внедрения включает в себя действия по установке и внедрению баз данных и приложений. Основными результатами стадии должны быть готовая к эксплуатации и перенесенная на программно-аппаратную платформу заказчика версия системы, документация сопровождения и акт приемочных испытаний по результатам опытной эксплуатации.

Стадии сопровождения и развития включают процессы и операции, связанные с регистрацией, диагностикой и локализацией ошибок, внесением изменений и тестированием, проведением доработок, тиражированием и распространением новых версий ПО в места его эксплуатации, переносом приложений на новую платформу и масштабированием системы. Стадия развития фактически является повторной итерацией стадии разработки.

Методология DATARUN опирается на две модели или на два представления:

- модель организации;
- модель ИС.

Методология DATARUN базируется на системном подходе к описанию деятельности организации. Построение моделей начинается с описания процессов, из которых затем извлекаются первичные данные (стабильное подмножество данных, которые организация должна использовать для своей деятельности). Первичные дан-

ные описывают продукты или услуги организации, выполняемые операции (транзакции) и потребляемые ресурсы. К первичным относятся данные, которые описывают внешние и внутренние сущности, такие как служащие, клиенты или агентства, а также данные, полученные в результате принятия решений, как например, графики работ, цены на продукты.

Основной принцип DATARUN заключается в том, что первичные данные, если они должным образом организованы в модель данных, становятся основой для проектирования архитектуры ИС. Архитектура ИС будет более стабильной, если она основана на первичных данных, тесно связанных с основными деловыми операциями, определяющими природу бизнеса, а не на традиционной функциональной модели.

Любая ИС (рис. 5.1) представляет собой набор модулей, исполняемых процессорами и взаимодействующих с базами данных. Базы данных и процессоры могут располагаться централизованно или быть распределенными. События в системе могут инициироваться внешними сущностями, такими как клиенты у банкоматов или временные события (конец месяца или квартала). Все транзакции осуществляются через объекты или модули интерфейса, которые взаимодействуют с одной или более базами данных.

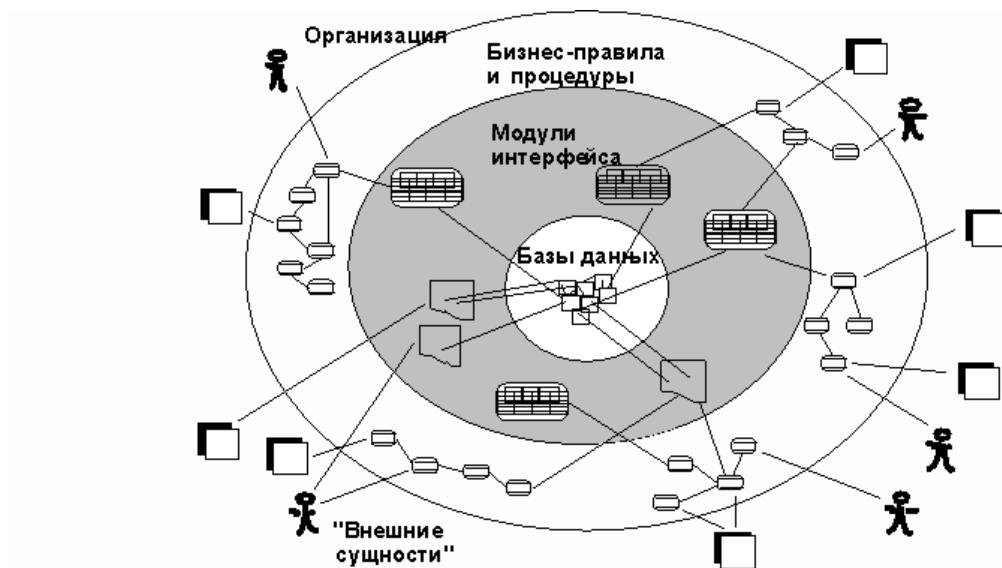


Рис. 5.1. Модель ИС

Подход DATARUN преследует две цели:

– определить стабильную структуру, на основе которой будет строиться ИС. Такой структурой является модель данных, получен-

ная из первичных данных, представляющих фундаментальные процессы организации;

– спроектировать ИС на основании модели данных.

Объекты, формируемые на основании модели данных, являются объектами базы данных, обычно размещаемыми на серверах в среде клиент/сервер. Объекты интерфейса, определенные в архитектуре компьютерной системы, обычно размещаются на клиентской части. Модель данных, являющаяся основой для спецификации совместно используемых объектов базы данных и различных объектов интерфейса, обеспечивает сопровождаемость ИС. На рис. 5.2 представлена последовательность шагов проектирования ИС.

На рис. 5.3 определены модели, создаваемые в процессе разработки ИС. Для их создания используется CASE-средство Silverrun. Silverrun обеспечивает автоматизацию проведения проектных работ в соответствии с методологией DATARUN. Предоставляемая этими средствами среда проектирования дает возможность руководителю проекта контролировать проведение работ, отслеживать выполнение работ, вовремя замечать отклонения от графика. Каждый участник проекта, подключившись к этой среде, может выяснить содержание и сроки выполнения порученной ему работы, детально изучить технику ее выполнения в гипертексте по технологиям, и вызвать инструмент (модуль Silverrun) для реального выполнения работы.

Информационная система создается последовательным построением ряда моделей, начиная с модели бизнес-процессов и заканчивая моделью программы, автоматизирующей эти процессы.

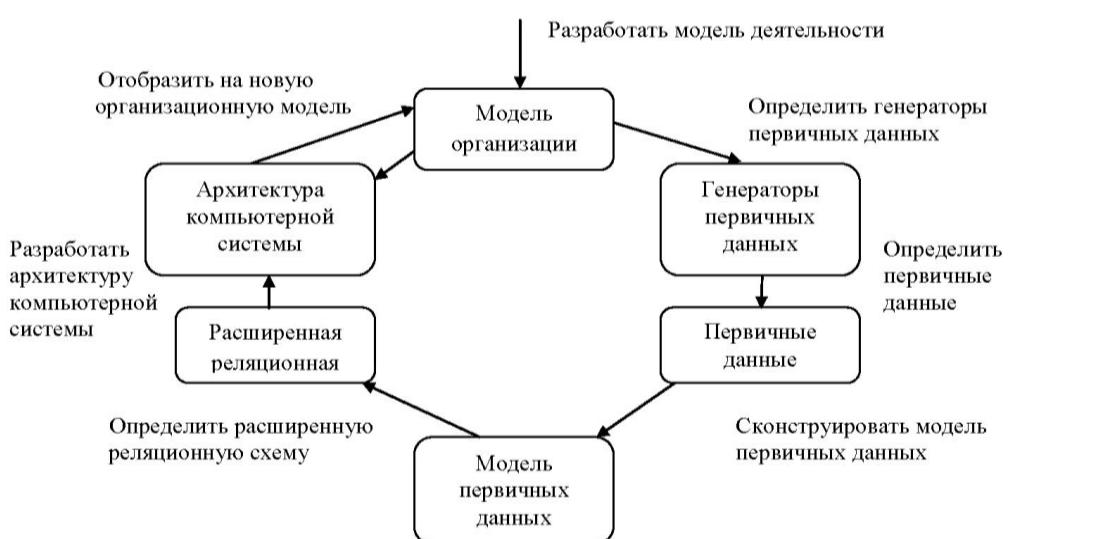
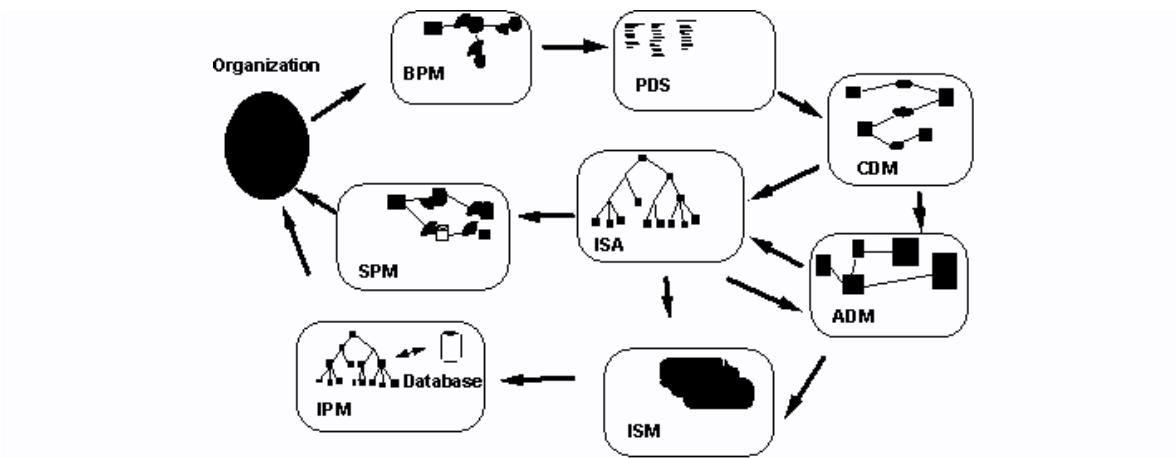


Рис. 5.2. Последовательность шагов проектирования системы



**BPM (Business Process Model)** – модель бизнес-процессов.

**PDS (Primary Data Structure)** – структура первичных данных.

**CDM (Conceptual Data Model)** – концептуальная модель данных.

**SPM (System Process Model)** – модель процессов системы.

**ISA (Information System Architecture)** – архитектура информационной системы.

**ADM (Application Data Model)** – модель данных приложения.

**IPM (Interface Presentation Model)** – модель представления интерфейса.

**ISM (Interface Specification Model)** – модель спецификации интерфейса.

Рис. 5.3. Модели, создаваемые с помощью подхода DATARUN

Создаваемая ИС должна основываться на функциях, выполняемых организацией. Поэтому первая создаваемая модель – это модель бизнес-процессов, построение которой осуществляется в модуле Silverrun BPM. Для этой модели используется специальная нотация BPM. В процессе анализа и спецификации бизнес-функций выявляются основные информационные объекты, которые документируются как структуры данных, связанные с потоками и хранилищами модели. Источниками для создания структур являются используемые в организации документы, должностные инструкции, описания производственных операций. Эти данные вводятся в том виде, как они существуют в деятельности организации. Нормализация и удаление избыточности производится позже при построении концептуальной модели данных в модуле Silverrun ERX. После создания модели бизнес-процессов информация сохраняется в репозитории проекта.

В процессе обследования работы организации выявляются и документируются структуры первичных данных. Эти структуры заносятся в репозиторий модуля BPM при описании циркулирующих в организации документов, сообщений, данных. В модели бизнес-процессов первичные структуры данных связаны с потоками и хранилищами информации.

На основе структур первичных данных в модуле Silverrun ERX создается концептуальная модель данных (ER-модель). От структур первичных данных концептуальная модель отличается удалением избыточности, стандартизацией наименований понятий и нормализацией. Эти операции в модуле ERX выполняются с помощью встроенной экспертной системы. Цель концептуальной модели данных – описать используемую информацию без деталей возможной реализации в базе данных, но в хорошо структурированном нормализованном виде.

На основе модели бизнес-процессов и концептуальной модели данных проектируется архитектура ИС. Определяются входящие в систему приложения, для каждого приложения специфицируются используемые данные и реализуемые функции. Архитектура ИС создается в модуле Silverrun BPM с использованием специальной нотации ISA. Основное содержание этой модели – структурные компоненты системы и навигация между ними. Концептуальная модель данных разбивается на части, соответствующие входящим в состав системы приложениям.

Перед разработкой приложений должна быть спроектирована структура корпоративной базы данных. DATARUN предполагает использование базы данных, основанной на реляционной модели. Концептуальная модель данных после нормализации переносится в модуль реляционного моделирования Silverrun RDM с помощью специального моста ERX-RDM. Преобразование модели из формата ERX в формат RDM происходит автоматически без вмешательства пользователя. После преобразования форматов получается модель реляционной базы данных. Эта модель детализируется в модуле Silverrun RDM определением физической реализации (типов данных СУБД, ключей, индексов, триггеров, ограничений ссылочной целостности). Правила обработки данных можно задавать как непосредственно на языке программирования СУБД, так и в декларативной форме, не привязанной к реализации. Мосты Silverrun к реляционным СУБД переводят эти декларативные правила на язык требуемой системы, что снижает трудоемкость программирования процедур сервера базы данных, а также позволяет из одной спецификации генерировать приложения для разных СУБД.

С помощью модели системных процессов детально документируется поведение каждого приложения. В модуле BPM создается модель системных процессов, определяющая, каким образом реализуются бизнес-процессы. Эта модель создается отдельно для каждого приложения и тесно связана с моделью данных приложения.

Приложение состоит из интерфейсных объектов (экраных форм, отчетов, процедур обработки данных). Каждый интерфейс системы (экранная форма, отчет, процедура обработки данных) имеет дело с подмножеством базы данных. В модели данных приложения (созданной в модуле RDM) создается подсхема базы данных для каждого интерфейса этого приложения. Уточняются также правила обработки данных, специфичные для каждого интерфейса. Интерфейс работает с данными в ненормализованном виде, поэтому спецификация данных, как ее видит интерфейс, оформляется как отдельная подсхема модели данных интерфейса.

Модель представления интерфейса – это описание внешнего вида интерфейса, как его видит конечный пользователь системы. Это может быть как документ, показывающий внешний вид экрана или структуру отчета, так и сам экран (отчет), созданный с помощью одного из средств визуальной разработки приложений, так называемых языков четвертого поколения (4GL – Fourth Generation Languages). Так как большинство языков 4GL позволяют быстро создавать работающие прототипы приложений, пользователь имеет возможность увидеть работающий прототип системы на ранних стадиях проектирования.

После создания подсхем реляционной модели для приложений проектируется детальная структура каждого приложения в виде схемы навигации экранов, отчетов, процедур пакетной обработки. На данном шаге эта структура детализируется до указания конкретных столбцов и таблиц базы данных, правил их обработки, вида экраных форм и отчетов. Полученная модель детально документирует приложение и непосредственно используется для программирования специфицированных интерфейсов.

Далее с помощью средств разработки приложений происходит физическое создание системы: приложения программируются и интегрируются в информационную систему.

### **5.1.2. Инструментальное средство SE Companion**

Инструментальное средство SE Companion является средой, в которой реализован электронный вариант методологии DATARUN. Оно позволяет:

- создать гипертекстовое описание методологии в виде иерархии описания стадий, этапов и операций разработки;
- создать гипертекстовое описание всех методов и методик реализации процессов ЖЦ ПО;

- выделить из гипертекстового описания иерархию процессов ЖЦ ПО для планирования и управления процессом создания ПО (иерархию работ);
- изменять гипертекстовые описания ЖЦ и методов так, как это необходимо разработчику, иными словами, производить авторизацию методологии и отслеживать эти изменения в иерархии работ, предназначеннной для управления проектом;
- привязать к процессам ЖЦ инструментальные средства поддержки этих процессов и обеспечить вызов инструментальных средств из соответствующих экранов гипертекстового справочника;
- обеспечить просмотр гипертекстовых экранов описания используемых методов из инструментальных средств;
- обеспечить поддержку процесса управления разработкой, в частности, за счет взаимодействия со средством планирования работ MS Project, оценивания трудоемкости проекта, отслеживания выполнения работ, создания графиков работ, и др.

Особенно важными являются возможность авторизации методологии и интерактивный доступ любого разработчика к описанию любого метода или процесса в нужный ему момент времени. На современном этапе развития технологии, в условиях быстрого изменения как программных и аппаратных средств, так и задач бизнеса, методология создания, сопровождения и развития ПО не должна быть неизменной; она должна иметь возможность изменяться и настраиваться на новые технологии, методы и инструментальные средства. Современные разработчики больших ИС приобретают одну или несколько методологий поставщика, а затем создают на их основе собственные методологии и технологии, адаптированные к конкретным условиям.

В SE Companion исходным документом, описывающим методологию (как процессы ЖЦ, так и все сопутствующие методы и методики), является файл в формате MS Word. Это обеспечивает возможности для описания методологии с любой степенью детализации, проведения разметки для создания гипертекста и авторизации методологии в принятом стандартном формате.

Гипертекстовое описание методологии и технологии создания ПО строится из описания процессов жизненного цикла, методов и методик и представляет собой единый гипертекстовый документ в формате MS Help. Итоговое гипертекстовое описание получается в результате трансляции исходного документа. Все изменения и дополнения методологии производятся посредством корректировки и, возможно, дополнительной разметки исходного документа.

Описание методологии создания системы обычно состоит из раздела описания процессов ЖЦ и разделов описания методов и методик. В свою очередь раздел описаний процессов состоит из иерархии описаний стадий, этапов и операций жизненного цикла с обязательным описанием выходных компонентов каждого процесса. Компоненты ПО создаются с применением методик и методов, описываемых в соответствующих разделах.

### **5.1.3. Примеры ТС ПО различных компаний-поставщиков**

#### **5.1.3.1. Технология Rational Unified Process (IBM Rational Software)**

На сегодняшний день практически все ведущие компании – разработчики технологий и программных продуктов (IBM, Oracle, Borland, Computer Associates и др.) – располагают развитыми технологиями создания ПО, которые создавались как собственными силами, так и за счет приобретения продуктов и технологий, созданных небольшими специализированными компаниями. Выбор в качестве примера четырех перечисленных компаний объясняется их ведущими позициями на мировом рынке ТС ПО, присутствием на российском рынке и ограниченным объемом настоящего обзора.

Одна из наиболее совершенных технологий, претендующих на роль мирового корпоративного стандарта, – Rational Unified Process (RUP). RUP представляет собой программный продукт, разработанный компанией Rational Software, которая в настоящее время входит в состав IBM.

RUP в значительной степени соответствует стандартам и нормативным документам, связанным с процессами ЖЦ ПО и оценкой технологической зрелости организаций-разработчиков (ISO 12207, ISO 9000, СММ и др.). Ее основными принципами являются:

- итерационный и инкрементный (наращиваемый) подход к созданию ПО;
- планирование и управление проектом на основе функциональных требований к системе – вариантов использования;
- построение системы на базе архитектуры ПО.

Первый принцип является определяющим. В соответствии с ним разработка системы выполняется в виде нескольких краткосрочных мини-проектов фиксированной длительности (от 2 до 6 недель), называемых итерациями. Каждая итерация включает свои

собственные этапы анализа требований, проектирования, реализации, тестирования, интеграции и завершается созданием работающей системы.

Итерационный цикл основывается на постоянном расширении и дополнении системы в процессе нескольких итераций с периодической обратной связью и адаптацией добавляемых модулей к существующему ядру системы. Система постоянно разрастается шаг за шагом, поэтому такой подход называют итерационным и инкрементным.

На рис. 5.4 показано общее представление RUP в двух измерениях. Горизонтальное измерение представляет время, отражает динамические аспекты процессов и оперирует такими понятиями, как стадии, итерации и контрольные точки. Вертикальное измерение отражает статические аспекты процессов и оперирует такими понятиями, как виды деятельности (технологические операции), рабочие продукты, исполнители и дисциплины (технологические процессы).

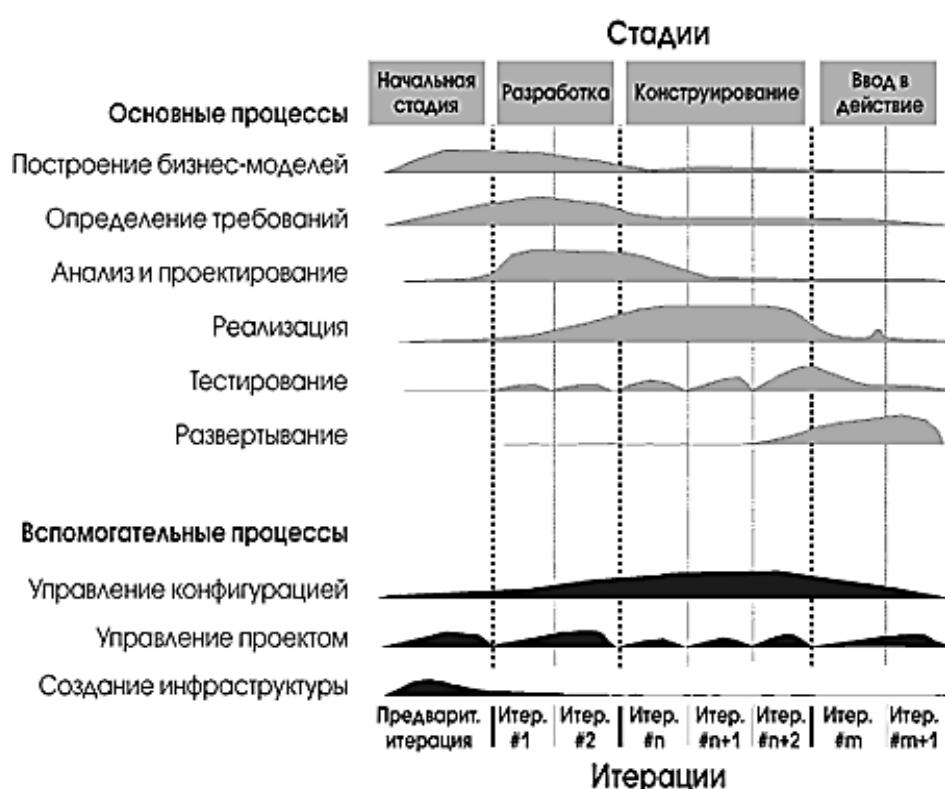


Рис. 5.4. Общее представление RUP

Согласно RUP, ЖЦ ПО разбивается на отдельные циклы, в каждом из которых создается новое поколение продукта. Каждый цикл в свою очередь разбивается на четыре последовательные стадии:

- начальная стадия (inception);

- стадия разработки (elaboration);
- стадия конструирования (construction);
- стадия ввода в действие (transition).

Каждая стадия завершается в четко определенной контрольной точке (milestone). В этот момент времени должны достигаться важные результаты и приниматься критически важные решения о дальнейшей разработке.

Начальная стадия может принимать множество разных форм. Для крупных проектов начальная стадия может вылиться во всестороннее изучение всех возможностей реализации проекта, которое займет месяцы. Во время начальной стадии вырабатывается бизнес-план проекта – определяется, сколько приблизительно он будет стоить и какой доход принесет. Определяются также границы проекта, и выполняется некоторый начальный анализ для оценки размеров проекта.

Результатами начальной стадии являются:

- общее описание системы: основные требования к проекту, его характеристики и ограничения;
- начальная модель вариантов использования (степень готовности – 10–20 %);
- начальный проектный глоссарий (словарь терминов);
- начальный бизнес-план;
- план проекта, отражающий стадии и итерации;
- один или несколько прототипов.

На стадии разработки выявляются более детальные требования к системе, выполняются высокоуровневый анализ предметной области и проектирование для построения базовой архитектуры системы, создается план конструирования, и устраняются наиболее рискованные элементы проекта.

Результатами стадии разработки являются:

- модель вариантов использования (завершенная, по крайней мере, на 80 %), определяющая функциональные требования к системе;
- перечень дополнительных требований, включая требования нефункционального характера и требования, не связанные с конкретными вариантами использования;
- описание базовой архитектуры будущей системы;
- работающий прототип;
- уточненный бизнес-план;
- план разработки всего проекта, отражающий итерации и критерии оценки для каждой итерации.

Самым важным результатом стадии разработки является описание базовой архитектуры будущей системы. Эта архитектура включает:

- модель предметной области, которая отражает понимание бизнеса и служит отправным пунктом для формирования основных классов предметной области;
- технологическую платформу, определяющую основные элементы технологии реализации системы и их взаимодействие.

Эта архитектура является основой всей дальнейшей разработки, она служит своего рода проектом для последующих стадий. В дальнейшем неизбежны незначительные изменения в деталях архитектуры, однако серьезные изменения маловероятны.

Стадия разработки занимает около пятой части общей продолжительности проекта. Основными признаками завершения стадии разработки являются два события:

- разработчики в состоянии оценить с достаточно высокой точностью, сколько времени потребуется на реализацию каждого варианта использования;
- идентифицированы все наиболее серьезные риски, и степень понимания наиболее важных из них такова, что известно, как справиться с ними.

Сущность планирования заключается в определении последовательности итераций конструирования и вариантов использования, реализуемых на каждой итерации. Итерации на стадии конструирования являются одновременно инкрементными и повторяющимися:

- итерации являются инкрементными в соответствии с той функцией, которую они выполняют. Каждая итерация добавляет очередные конструкции к вариантам использования, реализованным во время предыдущих итераций;
- итерации являются повторяющимися по отношению к разрабатываемому коду. На каждой итерации некоторая часть существующего кода переписывается с целью сделать его более гибким.

Результатом стадии конструирования является продукт, готовый к передаче конечным пользователям. Как минимум он содержит следующее:

- ПО, интегрированное на требуемых платформах;
- руководства пользователя;
- описание текущей реализации.

Назначением стадии ввода в действие является передача готового продукта в распоряжение пользователей. Данная стадия включает:

- бета-тестирование, позволяющее убедиться, что новая система соответствует ожиданиям пользователей;
- параллельное функционирование с существующей (legacy) системой, которая подлежит постепенной замене;
- конвертирование баз данных;
- оптимизацию производительности;
- обучение пользователей и специалистов службы сопровождения.

Статический аспект RUP представлен четырьмя основными элементами:

- роли;
- виды деятельности;
- рабочие продукты;
- дисциплины.

Понятие «роль» (role) определяет поведение и ответственность личности или группы личностей, составляющих проектную команду. Одна личность может играть в проекте много различных ролей.

Под видом деятельности конкретного исполнителя понимается единица выполняемой им работы. Вид деятельности (activity) соответствует понятию технологической операции. Он имеет четко определенную цель, обычно выражаемую в терминах получения или модификации некоторых рабочих продуктов (artifacts), таких как модель, элемент модели, документ, исходный код или план. Каждый вид деятельности связан с конкретной ролью. Продолжительность вида деятельности составляет от нескольких часов до нескольких дней, он обычно выполняется одним исполнителем и порождает только один или весьма небольшое количество рабочих продуктов. Любой вид деятельности должен являться элементом процесса планирования итерации, определение вариантов использования и действующих лиц, выполнение теста на производительность. Каждый вид деятельности сопровождается набором руководств (guidelines), представляющих собой методики выполнения технологических операций.

Дисциплина (discipline) соответствует понятию технологического процесса и представляет собой последовательность действий, приводящую к получению значимого результата.

В рамках RUP определены шесть основных дисциплин:

- построение бизнес-моделей;
- определение требований;

- анализ и проектирование;
- реализация;
- тестирование;
- развертывание;

и три вспомогательных:

- управление конфигурацией и изменениями;
- управление проектом;
- создание инфраструктуры.

RUP как продукт входит в состав комплекса Rational Suite, причем каждая из перечисленных выше дисциплин поддерживается определенным инструментальным средством комплекса. Физическая реализация RUP представляет собой Web-сайт, включающий следующие компоненты:

- описание всех элементов динамического и статического аспекта RUP;
- навигатор по всем элементам RUP, глоссарий и средство быстрого обучения технологии;
- руководства для всех участников проектной команды, охватывающие весь жизненный цикл ПО. Руководства представлены в двух видах: для осмыслиения процесса на верхнем уровне и в виде подробных наставлений по повседневной деятельности;
- наставления по использованию инструментальных средств, входящих в состав Rational Suite;
- примеры и шаблоны проектных решений для Rational Rose;
- шаблоны проектной документации для SoDa;
- шаблоны в формате Microsoft Word, предназначенные для поддержки документации по всем процессам и действиям жизненного цикла ПО;
- планы в формате Microsoft Project, отражающие итерационный характер разработки ПО.

Адаптация RUP к потребностям конкретной организации или проекта обеспечивается с помощью Rational Process Workbench (RPW) – специального набора инструментов и шаблонов для настройки и публикации Web-сайтов на основе RUP. RPW поддерживает три основные функции моделирования технологических процессов:

- определение процесса;
- описание процесса;
- представление процесса.

Библиотека элементов процесса содержит текстовую информацию о каждом элементе в модели процесса, все текстовые страницы RUP, а RPW – необходимые шаблоны для создания новых страниц описания. RPW генерирует описание процессов, включающее текст и графику, в виде Web-сайта, соединяя модели процессов и библиотеку описаний в единое целое.

RUP опирается на интегрированный комплекс инструментальных средств Rational Suite. Он существует в следующих вариантах:

- Rational Suite AnalystStudio предназначен для определения и управления полным набором требований к разрабатываемой системе;
- Rational Suite DevelopmentStudio предназначен для проектирования и реализации ПО;
- Rational Suite TestStudio представляет собой набор продуктов, предназначенных для автоматического тестирования приложений;
- Rational Suite Enterprise обеспечивает поддержку полного жизненного цикла ПО и предназначен как для менеджеров проекта, так и отдельных разработчиков, выполняющих несколько функциональных ролей в команде разработчиков.

В состав Rational Suite, кроме самой технологии RUP как продукта, входят следующие компоненты:

- Rational Rose – средство визуального моделирования (анализа и проектирования), использующее язык UML;
- Rational XDE – средство анализа и проектирования, интегрируемое с платформами MS Visual Studio.NET и IBM WebSphere Studio Application Developer;
- Rational Requisite Pro – средство управления требованиями, предназначенное для организации совместной работы группы разработчиков. Оно позволяет команде разработчиков создавать, структурировать, устанавливать приоритеты, отслеживать, контролировать изменения требований, возникающих на любом этапе разработки компонентов приложения;
- Rational Rapid Developer – средство быстрой разработки приложений на платформе Java 2 Enterprise Edition;
- Rational ClearCase – средство управления конфигурацией ПО;
- Rational SoDA – средство автоматической генерации проектной документации;
- Rational ClearQuest – средство для управления изменениями и отслеживания дефектов в проекте на основе средств e-mail и Web;
- Rational Quantify – средство количественного определения узких мест, влияющих на общую эффективность работы программы;

- Rational Purify – средство для локализации трудно обнаруживаемых ошибок времени выполнения программы;
- Rational PureCoverage – средство идентификации участков кода, пропущенных при тестировании;
- Rational TestManager – средство планирования функционального и нагружочного тестирования;
- Rational Robot – средство записи и воспроизведения тестовых сценариев;
- Rational TestFactory – средство тестирования надежности;
- Rational Quality Architect – средство генерации кода для тестирования.

Одно из основных инструментальных средств комплекса Rational Rose представляет собой семейство объектно-ориентированных CASE-средств и предназначено для автоматизации процессов анализа и проектирования ПО, а также для генерации кодов на различных языках и выпуска проектной документации. Rational Rose реализует процесс объектно-ориентированного анализа и проектирования ПО, описанный в RUP. В основе работы Rational Rose лежит построение диаграмм и спецификаций UML, определяющих архитектуру системы, ее статические и динамические аспекты. В составе Rational Rose можно выделить шесть основных структурных компонентов: репозиторий, графический интерфейс пользователя, средства просмотра проекта (браузер), средства контроля проекта, средства сбора статистики и генератор документов. К ним добавляются генераторы кодов для каждого поддерживаемого языка, состав которых меняется от версии к версии.

Репозиторий представляет собой базу данных проекта. Браузер обеспечивает «навигацию» по проекту, в том числе перемещение по иерархиям классов и подсистем, переключение от одного вида диаграмм к другому и т.д. Средства контроля и сбора статистики дают возможность находить и устранять ошибки по мере развития проекта, а не после завершения его описания. Генератор отчетов формирует тексты выходных документов на основе содержащейся в репозитории информации.

Средства автоматической генерации кода, используя информацию, содержащуюся в диаграммах классов и компонентов, формируют файлы описаний классов. Создаваемый таким образом скелет программы может быть уточнен путем прямого программирования на соответствующем языке (основные языки, поддерживаемые Rational Rose, C++ и Java).

В результате разработки проекта с помощью Rational Rose формируются следующие документы:

- диаграммы UML, в совокупности представляющие собой модель разрабатываемой программной системы;
- спецификации классов, объектов, атрибутов и операций;
- заготовки текстов программ.

Тексты программ являются заготовками для последующей работы программистов. Состав информации, включаемой в программные файлы, определяется либо по умолчанию, либо по усмотрению пользователя. В дальнейшем эти исходные тексты развиваются программистами в полноценные программы.

Инструментальное средство Rational XDE представляет собой развитие возможностей Rational Rose в части синхронизации модели и кода (исключающей необходимость прямой и обратной генерации кода). Rational XDE обеспечивает:

- синхронизацию между кодом и моделью;
- отображение элементов кода Java и C# в UML;
- автоматическую синхронизацию с настраиваемым разрешением конфликтов;
- одновременное отображение кода и модели;
- постоянную синхронизацию модели UML как части проекта Java или C#.

### **5.1.3.2. Технология Oracle**

Методическую основу ТС ПО корпорации Oracle составляет метод Oracle (Oracle Method) – комплекс методов, охватывающий большинство процессов ЖЦ ПО. В состав комплекса входят:

- CDM (Custom Development Method) – разработка прикладного ПО;
- PJM (Project Management Method) – управление проектом;
- AIM (Application Implementation Method) – внедрение прикладного ПО;
- BPR (Business Process Reengineering) – реинжиниринг бизнес-процессов;
- OCM (Organizational Change Management) – управление изменениями и др.

Метод CDM оформлен в виде консалтингового продукта CDM Advantage – библиотеки стандартов и руководств (включающего также PJM). Он представляет собой развитие достаточно давно со-

зданного Oracle CASE-Method, известного по использованию CASE-средств фирмы Oracle и книгам Р. Баркера. По существу, CDM является методическим руководством по разработке прикладного ПО с использованием инструментального комплекса Oracle Developer Suite, а сам процесс проектирования и разработки тесно связан с Oracle Designer и Oracle Forms.

В соответствии с CDM ЖЦ ПО формируется из определенных этапов (фаз) проекта и процессов, каждый из которых выполняется в течение нескольких этапов (рис. 5.5):

- стратегия (определение требований);
  - анализ (формулирование детальных требований к системе);
  - проектирование (преобразование требований в детальные спецификации системы);
  - реализация (написание и тестирование приложений);
  - внедрение (установка новой прикладной системы, подготовка к началу эксплуатации);
  - эксплуатация.

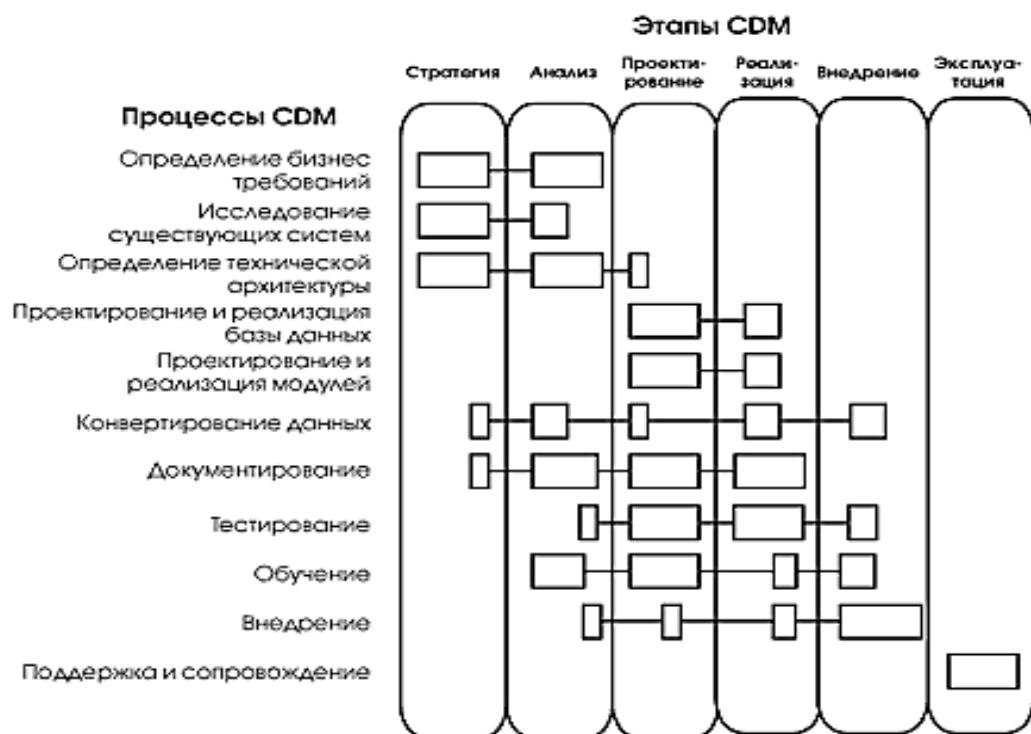


Рис. 5.5. Этапы и процессы CDM

На этапе стратегии определяются цели создания системы, приоритеты и ограничения, разрабатывается системная архитектура, и составляется план разработки. На этапе анализа строятся модель информационных потребностей (диаграмма «сущность-связь»), диа-

грамма функциональной иерархии (на основе функциональной декомпозиции системы), матрица перекрестных ссылок и диаграмма потоков данных.

На этапе проектирования разрабатывается подробная архитектура системы, проектируются схема реляционной БД и программные модули, устанавливаются перекрестные ссылки между компонентами системы для анализа их взаимного влияния и контроля за изменениями.

На этапе реализации создается БД, строятся прикладные системы, производится их тестирование, проверка качества и соответствие требованиям пользователей. Создаются системная документация, материалы для обучения и руководства пользователей.

На этапах внедрения и эксплуатации анализируются производительность и целостность системы, выполняется поддержка и, при необходимости, модификация системы.

#### **Процессы CDM:**

- определение бизнес-требований, или постановка задачи (Business Requirements Definition);
- исследование существующих систем (Existing Systems Examination). Выполнение этого процесса должно обеспечить понимание состояния существующего технического и программного обеспечения для планирования необходимых изменений;
- определение технической архитектуры (Technical Architecture);
- проектирование и реализация базы данных (Database Design and Build). Процесс предусматривает проектирование и реализацию реляционной базы данных, включая создание индексов и других объектов БД;
- проектирование и реализация модулей (Module Design and Build). Этот процесс является основным в проекте. Он включает непосредственное проектирование приложения и создание кода прикладной программы;
- конвертирование данных (Data Conversion). Цель этого процесса – преобразовывать, перенести и проверить согласованность и непротиворечивость данных, оставшихся в наследство от «старой» системы и необходимых для работы в новой системе;
- документирование (Documentation);
- тестирование (Testing);
- обучение (Training);

– внедрение, или переход к новой системе (Transition). Этот процесс включает решение задач установки, ввода новой системы в эксплуатацию, прекращения эксплуатации старых систем;

– поддержка и сопровождение (Post-System Support).

Процессы состоят из последовательностей взаимосвязанных задач.

CDM предоставляет возможность выбрать требуемый подход к разработке. Это возможно, поскольку каждый процесс базируется на известных зависимостях между задачами одного типа и не зависит от того, на какие этапы будет разбит проект.

При определении подхода к разработке оцениваются масштаб, степень сложности и критичность будущей системы. При этом учитываются стабильность требований, сложность и количество бизнес-правил, количество автоматически выполняемых функций, разнообразие и количество пользователей, степень взаимодействия с другими системами, критичность приложения для основного бизнес-процесса компании и целый ряд других.

В соответствии с этими факторами в CDM выделяются два основных подхода к разработке:

– **Классический подход (Classic).** Этапы данного подхода представлены на рис. 5.5. Классический подход применяется для наиболее сложных и масштабных проектов, он предусматривает последовательный и детерминированный порядок выполнения задач. Для таких проектов характерны большое количество реализуемых бизнес-правил, распределенная архитектура, критичность приложения. Применение классического подхода также рекомендуется при нехватке опыта у разработчиков, неподготовленности пользователей, нечетко определенной задаче. Продолжительность таких проектов от 8 до 36 месяцев.

– **Подход быстрой разработки (Fast Track).** Данный подход в отличие от каскадного классического является итерационным и основан на методе DSDM (Dynamic Systems Development Method). В этом подходе четыре этапа – стратегия, моделирование требований, проектирование и генерация системы и внедрение в эксплуатацию. Подход используется для реализации небольших и средних проектов с несложной архитектурой системы, гибкими сроками и четкой постановкой задач. Продолжительность проекта от 4 до 16 месяцев.

PJM – это определенная дисциплина ведения проекта, позволяющая гарантировать, что цели проекта, четко определенные в его начале, остаются в центре внимания на протяжении всего проекта. В основе PJM лежит метод, ориентированный на выполнение само-

стоятельных процессов (под процессом понимается набор связанных задач, выполнением которых достигается определенная цель проекта). Так же, как и CDM, метод руководства проектом представляется в виде четко определенной операционной схемы, в которой выделяются процессы, этапы, задачи, результаты решения задач и зависимости между задачами:

- управление проектом и предоставление отчетности (Control and Reporting). Этот процесс содержит задачи, в результате решения которых определяются границы проекта и подход к разработке, происходит управление изменениями, и контролируется возможный риск;
- управление работой (Work Management). Процесс содержит задачи, помогающие контролировать работы, выполняемые в проекте;
- управление ресурсами (Resource Management). Здесь решаются задачи, связанные с обеспечением каждого этапа исполнителями;
- управление качеством (Quality Management). Процесс управления качеством гарантирует, что проект отвечает требованиям пользователя в течение всего процесса разработки;
- управление конфигурацией (Configuration Management).

Цикл решения задач РМ состоит из отдельных этапов. Количество этапов зависит от выбранного подхода к разработке. Задачи РМ можно распределить внутри каждого процесса по трем группам – задачи планирования, управления и завершения, и по уровням – отнести задачу на уровень проекта или на уровень отдельного этапа.

По аналогии с CDM в РМ предусмотрено широкое использование шаблонов разрабатываемых документов.

Комплекс Oracle Developer Suite содержит набор интегрированных средств разработки для быстрого создания приложений. Он включает средства моделирования, программирования на Java, разработки компонентов, бизнес-анализа и составления отчетов. Все эти средства используют общие ресурсы, что позволяет совместно работать над одним проектом группе разработчиков. Oracle Developer Suite интегрирован с Oracle Database и Oracle Application Server, образуя единую платформу для создания и установки приложений.

Oracle Developer Suite поддерживает стандарты J2EE: Enterprise Java Beans (EJB), сервлеты и страницы JavaServer (JSP). В него также входят анализатор XML, процессор XSLT, процессор схем XML и XSQL-сервлет для разработки XML-приложений.

В Oracle Developer Suite встроена поддержка языка UML для разработки приложений на основе моделей. Модели хранятся в общем репозитории Oracle, который предназначен для поддержки больших коллективов разработчиков.

Oracle Developer Suite включает в себя:

- Oracle Designer – средство моделирования и генерации приложений;
- Oracle Forms – средство быстрой разработки приложений;
- Oracle Reports – визуальное средство разработки отчетов;
- Oracle JDeveloper – средство визуального программирования на языке Java;
- Oracle Discoverer – средство для разработки аналитических приложений;
- Oracle Warehouse Builder – система для построения хранилищ данных;
- Oracle Portal – средство разработки информационного портала организации.

CASE-средство Oracle Designer является интегрированным средством, обеспечивающим в совокупности со средствами разработки приложений поддержку ЖЦ ПО.

Oracle Designer представляет собой семейство методов и поддерживающих их программных продуктов. Базовый метод Oracle Designer (CDM) – структурный метод проектирования систем, охватывающий полностью все стадии ЖЦ ПО. Версия Oracle Designer для объектно-реляционной СУБД Oracle содержит также расширение в виде средств объектного моделирования, базирующихся на стандарте UML.

Oracle Designer обеспечивает графический интерфейс при разработке различных моделей (диаграмм) предметной области. В процессе построения моделей информация о них заносится в репозиторий. В состав Oracle Designer входят следующие компоненты:

- Repository Administrator – средства управления репозиторием (создание и удаление приложений, управление доступом к данным со стороны различных пользователей, экспорт и импорт данных);
- Repository Object Navigator – средство доступа к репозиторию, обеспечивающее многооконный объектно-ориентированный интерфейс доступа ко всем элементам репозитория;
- Process Modeler – средство анализа и моделирования бизнес-процессов;
- Systems Modeler – набор средств построения функциональных и информационных моделей проектируемой системы, включа-

ющий средства для построения диаграмм «сущность-связь» (Entity-Relationship Diagrammer), диаграмм функциональных иерархий (Function Hierarchy Diagrammer), диаграмм потоков данных (Data Flow Diagrammer) и средство анализа и модификации связей объектов репозитория различных типов (Matrix Diagrammer);

– Systems Designer – набор средств проектирования ПО, включающий средство построения структуры реляционной базы данных (Data Diagrammer), а также средства построения диаграмм, отображающих взаимодействие с данными, иерархию, структуру и логику приложений, реализуемую хранимыми процедурами на языке PL/SQL (Module Data Diagrammer, Module Structure Diagrammer и Module Logic Navigator);

– Server Generator – генератор описаний объектов БД Oracle (таблиц, индексов, ключей, последовательностей и т.д.);

– Forms Generator – генератор приложений для Oracle Forms. Генерируемые приложения включают в себя различные экранные формы, средства контроля данных, проверки ограничений целостности и автоматические подсказки;

– Repository Reports – генератор стандартных отчетов, интегрированный с Oracle Reports.

Репозиторий Oracle Designer представляет собой хранилище всех проектных данных и может работать в многопользовательском режиме, обеспечивая параллельное обновление информации несколькими разработчиками. В процессе проектирования автоматически поддерживаются перекрестные ссылки между объектами словаря и могут генерироваться более 70 стандартных отчетов о моделируемой предметной области. Физическая среда хранения репозитория – база данных Oracle.

### **5.1.3.3. Технология Borland**

Компания Borland в результате развития собственных разработок и приобретения целого ряда компаний представила интегрированный комплекс инструментальных средств, реализующих управление полным жизненным циклом приложений (Application Life Cycle Management, ALM). В соответствии с технологией Borland процесс создания ПО включает в себя пять основных этапов:

- определение требований;
- анализ и проектирование;
- разработка;

- тестирование и профилирование;
- развертывание.

Выполнение всех этапов координируется процессом управления конфигурацией и изменениями.

Определение требований реализуется с помощью системы управления требованиями CaliberRM, которая стала частью семейства продуктов Borland в результате покупки компании Starbase. CaliberRM сохраняет требования в базе данных, документы с их описанием создаются с помощью встроенного механизма генерации документов MS Word на базе заданных шаблонов. Система обеспечивает экспорт данных в таблицы MS Access и импорт из MS Word. CaliberRM поддерживает различные методы визуализации зависимостей между требованиями, с помощью которых пользователь может ограничить область анализа, необходимого в случае изменения того или иного требования. Имеется модуль, который использует данные требования для оценки трудозатрат, рисков и расходов, связанных с реализацией требований.

Средство анализа и проектирования Together ControlCenter разработано компанией TogetherSoft. В основе его применения лежит один из вариантов подхода «Быстрой разработки ПО» под названием Feature Driven Development (FDD).

Together ControlCenter – интегрированная среда проектирования и разработки, поддерживающая визуальное моделирование на UML с последующим написанием приложений для платформ J2EE (Java) и .Net (C#, C++ и Visual Basic). Кроме базовой версии, имеются уменьшенный вариант системы для индивидуальных разработчиков и небольших групп (Together Solo), а также редакции для платформы IBM WebSphere и среды разработки Jbuilder.

В системе реализована технология LiveSource, которая обеспечивает синхронизацию между проектом приложения и изменениями – при внесении изменений в исходные тексты меняется модель программы, а при изменении модели надлежащим образом изменяется текст на языке программирования. Это исключает необходимость вручную модифицировать модель или переписывать код. Контроль версий осуществляется благодаря функциональной интеграции Together и системы StarTeam. Поддерживается также интеграция с системой управления конфигурацией Rational ClearCase.

Инструментальные средства тестирования появились в составе комплекса Borland в результате покупки компании Optimizeit. К ним относятся Optimizeit Suite 5, Optimizeit Profiler for .NET и Optimizeit ServerTrace. Первые две системы позволяют выявить потенциаль-

ные проблемы использования аппаратных ресурсов – памяти и процессорных мощностей на платформах J2EE и .Net соответственно. Интеграция Optimizeit Suite 5 в среду разработки Jbuilder, а Optimizeit Profiler – в C#Builder и Visual Basic .Net позволяет проводить контрольные испытания приложений по мере разработки и ликвидировать узкие места производительности. Система Optimizeit ServerTrace предназначена для управления производительностью серверных J2EE-приложений с точки зрения достижения заданного уровня обслуживания и сбора контрольных данных по виртуальным Java-машинам.

Сущность концепции ALM сосредоточена в системе управления конфигурацией и изменениями: именно она объединяет основные фазы ЖЦ ПО. Такой системой является StarTeam, разработанная компанией Starbase. Она выполняет функции контроля версий, управления изменениями, отслеживания дефектов, управления требованиями (в интеграции с CaliberRM), управления потоком задач и управления проектом.

StarTeam совместима с интерфейсом Microsoft Source Code Control и интегрируется с любой системой разработки, которая поддерживает этот API. Кроме того, в системе реализованы средства интеграции со средствами разработки и моделирования Together, JBuilder, Delphi, C++Builder и C#Builder.

В технологии Borland выделяются три уровня интеграции. **Функциональная (touch-point)** интеграция позволяет обратиться из одной системы к функциям другой, выбрав соответствующий пункт меню. Например, интерфейс управления изменениями StarTeam непосредственно отображается в системах Together, C#Builder и Visual Studio .Net. Такая интеграция дает возможность разделять информацию между системами, но не обеспечивает единого рабочего пространства, вынуждает пользователя переключать окна и приводит к дублированию процессов управления структурой проекта. **Встроенная (embedded)** интеграция обеспечивает работу с одной системой непосредственно в среде другой. Например, не выходя из среды разработки Jbuilder, можно просматривать графики производительности, которые создает система Optimizeit. Самый высокий уровень интеграции – **синергетический (synergistic)**, позволяющий сочетать функции двух различных продуктов незаметно для разработчиков. Для большинства продуктов Borland и других поставщиков синергетическая интеграция пока остается делом будущего, однако ее принципы уже реализуются.

#### **5.1.3.4. Технология Computer Associates**

Компания Computer Associates предлагает комплексы инструментальных средств поддержки различных процессов ЖЦ ПО:

1) AllFusion Modeling Suite – интегрированный комплекс CASE-средств, включающий следующие продукты:

- AllFusion Process Modeler (BPwin) – функциональное моделирование;

- AllFusion ERwin Data Modeler (ERwin) – моделирование данных;

- AllFusion Component Modeler (Paradigm Plus) – объектно-ориентированный анализ и проектирование с использованием UML и возможностью генерации кода;

- AllFusion Model Manager (Model Mart) – организация совместной работы команды разработчиков;

- AllFusion Data Model Validator (ERwin Examiner) – проверка структуры и качества моделей данных;

- AllFusion Change Management Suite – комплекс средств управления конфигурацией и изменениями;

- AllFusion Process Management Suite – средства управления процессами и проектами для различных типов приложений;

2) CASE-средства ERwin и BPwin были разработаны фирмой Logic Works, которая в 1998 г. вошла в состав PLATINUM Technology, а затем Computer Associates;

3) BPwin – средство моделирования бизнес-процессов, реализующее метод IDEF0, а также поддерживающее диаграммы потоков данных и IDEF3. В процессе моделирования BPwin позволяет переключиться с нотации IDEF0 на любой ветви модели на нотацию IDEF3 или DFD и создать смешанную модель. BPwin поддерживает функционально-стоимостной анализ (ABC).

Семейство продуктов ERwin представляет собой набор средств концептуального моделирования данных, использующих метод IDEF1X. ERwin реализует проектирование схемы БД, генерацию ее описания на языке целевой СУБД (Oracle, Sybase, DB2, Microsoft SQL Server и др.) и реверсный инжиниринг существующей БД. ERwin выпускается в нескольких конфигурациях, ориентированных на наиболее распространенные средства разработки приложений.

Для управления групповой разработкой используется средство Model Mart, обеспечивающее многопользовательский доступ к мо-

делям, созданным с помощью ERwin и BPwin. Модели хранятся на центральном сервере и доступны для всех участников группы проектирования.

Model Mart удовлетворяет ряду требований, предъявляемых к средствам управления разработкой крупных систем, а именно:

– Совместное моделирование. Каждый участник проекта имеет инструмент поиска и доступа к интересующей его модели в любое время. При совместной работе используются три режима: незащищенный, защищенный и режим просмотра. В режиме просмотра запрещается любое изменение моделей. В защищенном режиме модель, с которой работает один пользователь, не может быть изменена другими пользователями. В незащищенном режиме пользователи могут работать с общими моделями в реальном масштабе времени.

– Создание библиотек решений. Model Mart позволяет формировать библиотеки стандартных решений, включающие наиболее удачные фрагменты реализованных проектов, накапливать и использовать типовые модели, объединяя их при необходимости «сборки» больших систем. На основе существующих баз данных с помощью ERwin возможно восстановление моделей (реверсный инжиниринг), которые в процессе анализа пригодности их для новой системы могут объединяться с типовыми моделями из библиотек моделей.

– Управление доступом. Для каждого участника проекта определяются права доступа, в соответствии с которыми они получают возможность работать только с определенными моделями. Права доступа могут быть определены как для групп, так и для отдельных участников проекта. Роль специалистов, участвующих в различных проектах может меняться, поэтому в Model Mart можно определять и управлять правами доступа участников проекта к библиотекам, моделям и даже к специфическим областям модели.

## 5.2. Silverrun

CASE-средство Silverrun американской фирмы Computer Systems Advisers, Inc. (CSA) используется для анализа и проектирования ИС бизнес-класса и ориентировано в большей степени на спиральную модель ЖЦ. Оно применимо для поддержки любой методологии, основанной на разделном построении функциональной и информационной моделей (диаграмм потоков данных и диаграмм «сущность-связь»).

Настройка на конкретную методологию обеспечивается выбором требуемой графической нотации моделей и набора правил проверки проектных спецификаций. В системе имеются готовые настройки для наиболее распространенных методологий: DATARUN (основная методология, поддерживаемая SilverRun), Gane/Sarson, Yourdon/DeMarco, Merise, Ward/Mellor, Information Engineering. Для каждого понятия, введенного в проекте, имеется возможность добавления собственных описателей. Архитектура SilverRun позволяет наращивать среду разработки по мере необходимости.

### *Структура и функции*

SilverRun имеет модульную структуру и состоит из четырех модулей, каждый из которых является самостоятельным продуктом и может приобретаться и использоваться без связи с остальными модулями.

Модуль построения моделей бизнес-процессов в форме диаграмм потоков данных (BPM – Business Process Modeler) позволяет моделировать функционирование обследуемой организации или создаваемой ИС. В модуле BPM обеспечена возможность работы с моделями большой сложности: автоматическая перенумерация, работа с деревом процессов (включая визуальное перетаскивание ветвей), отсоединение и присоединение частей модели для коллективной разработки. Диаграммы могут изображаться в нескольких предопределенных нотациях, включая Yourdon/DeMarco и Gane/Sarson. Имеется также возможность создавать собственные нотации, в том числе добавлять в число изображаемых на схеме дескрипторов определенные пользователем поля.

Модуль концептуального моделирования данных (ERX – Entity-Relationship eXpert) обеспечивает построение моделей данных «сущность-связь», не привязанных к конкретной реализации. Этот модуль имеетстроенную экспертную систему, позволяющую создать корректную нормализованную модель данных посредством ответов на содержательные вопросы о взаимосвязи данных. Возможно автоматическое построение модели данных из описаний структур данных. Анализ функциональных зависимостей атрибутов дает возможность проверить соответствие модели требованиям третьей нормальной формы и обеспечить их выполнение. Проверенная модель передается в модуль RDM.

Модуль реляционного моделирования (RDM – Relational Data Modeler) позволяет создавать детализированные модели «сущность-

связь», предназначенные для реализации в реляционной базе данных. В этом модуле документируются все конструкции, связанные с построением базы данных: индексы, триггеры, хранимые процедуры и т.д. Гибкая изменяемая нотация и расширяемость репозитория позволяют работать по любой методологии. Возможность создавать подсхемы соответствует подходу ANSI SPARC к представлению схемы базы данных. На языке подсхем моделируются как узлы распределенной обработки, так и пользовательские представления. Этот модуль обеспечивает проектирование и полное документирование реляционных баз данных.

Менеджер репозитория рабочей группы (WRM – Workgroup Repository Manager) применяется как словарь данных для хранения общей для всех моделей информации, а также обеспечивает интеграцию модулей Silverrun в единую среду проектирования.

Платой за высокую гибкость и разнообразие изобразительных средств построения моделей является такой недостаток Silverrun, как отсутствие жесткого взаимного контроля между компонентами различных моделей (например, возможности автоматического распространения изменений между DFD различных уровней декомпозиции). Следует, однако, отметить, что этот недостаток может иметь существенное значение только в случае использования каскадной модели ЖЦ ПО.

#### *Взаимодействие с другими средствами*

Для автоматической генерации схем баз данных у Silverrun существуют мости к наиболее распространенным СУБД: Oracle, Informix, DB2, Ingres, Progress, SQL Server, SQLBase, Sybase. Для передачи данных в средства разработки приложений имеются мости к языкам 4GL: JAM, PowerBuilder, SQL Windows, Uniface, NewEra, Delphi. Все мости позволяют загрузить в Silverrun RDM информацию из каталогов соответствующих СУБД или языков 4GL. Это позволяет документировать, перепроектировать или переносить на новые платформы уже находящиеся в эксплуатации базы данных и прикладные системы. При использовании моста Silverrun расширяет свой внутренний репозиторий специфичными для целевой системы атрибутами. После определения значений этих атрибутов генератор приложений переносит их во внутренний каталог среды разработки или использует при генерации кода на языке SQL. Таким образом можно полностью определить ядро базы данных с использованием всех возможностей конкретной СУБД: триггеров, хранимых процедур, ограничений ссылочной целостности. При создании приложе-

ния на языке 4GL данные, перенесенные из репозитория Silverrun, используются либо для автоматической генерации интерфейсных объектов, либо для быстрого их создания вручную.

Для обмена данными с другими средствами автоматизации проектирования, создания специализированных процедур анализа и проверки проектных спецификаций, составления специализированных отчетов в соответствии с различными стандартами в системе Silverrun имеется три способа выдачи проектной информации во внешние файлы:

– система отчетов. Можно, определив содержимое отчета по репозиторию, выдать отчет в текстовый файл. Этот файл можно затем загрузить в текстовый редактор или включить в другой отчет;

– система экспорта/импорта. Для более полного контроля над структурой файлов в системе экспорта/импорта имеется возможность определять не только содержимое экспортного файла, но и разделители записей, полей в записях, маркеры начала и конца текстовых полей. Файлы с указанной структурой можно не только формировать, но и загружать в репозиторий. Это дает возможность обмениваться данными с различными системами: другими CASE-средствами, СУБД, текстовыми редакторами и электронными таблицами;

– хранение репозитория во внешних файлах через ODBC-драйверы. Для доступа к данным репозитория из наиболее распространенных систем управления базами данных обеспечена возможность хранить всю проектную информацию непосредственно в формате этих СУБД.

### *Групповая работа*

Групповая работа поддерживается в системе Silverrun двумя способами.

В стандартной однопользовательской версии имеется механизм контролируемого разделения и слияния моделей. Разделив модель на части, можно раздать их нескольким разработчикам. После детальной доработки модели объединяются в единые спецификации.

Сетевая версия Silverrun позволяет осуществлять одновременную групповую работу с моделями, хранящимися в сетевом репозитории на базе СУБД Oracle, Sybase или Informix. При этом несколько разработчиков могут работать с одной и той же моделью, так как блокировка объектов происходит на уровне отдельных элементов модели.

### *Среда функционирования*

Имеются реализации Silverrun трех платформ – MS Windows, Macintosh и OS/2 Presentation Manager – с возможностью обмена проектными данными между ними.

Для функционирования в среде Windows необходимо иметь компьютер с процессором модели не ниже i486 и оперативную память объемом не менее 8 Мб (рекомендуется 16 Мб). На диске полная инсталляция Silverrun занимает 20 Мб.

## **5.3. Vantage Team Builder (Westmount I-CASE)**

Vantage Team Builder представляет собой интегрированный программный продукт, ориентированный на реализацию каскадной модели ЖЦ ПО и поддержку полного ЖЦ ПО.

### *Структура и функции*

Vantage Team Builder обеспечивает выполнение следующих функций:

- проектирование диаграмм потоков данных, «сущность-связь», структур данных, структурных схем программ и последовательностей экранных форм;
- проектирование диаграмм архитектуры системы SAD (проектирование состава и связи вычислительных средств, распределения задач системы между вычислительными средствами, моделирование отношений типа «клиент-сервер», анализ использования менеджеров транзакций и особенностей функционирования систем в реальном времени);
- генерация кода программ на языке 4GL целевой СУБД с полным обеспечением программной среды и генерация SQL-кода для создания таблиц БД, индексов, ограничений целостности и хранимых процедур;
- программирование на языке С со встроенным SQL;
- управление версиями и конфигурацией проекта;
- многопользовательский доступ к репозиторию проекта;
- генерация проектной документации по стандартным и индивидуальным шаблонам;
- экспорт и импорт данных проекта в формате CDIF (CASE Data Interchange Format).

Vantage Team Builder поставляется в различных конфигурациях в зависимости от используемых СУБД (ORACLE, Informix, Sybase или Ingres) или средств разработки приложений (Uniface). Конфигу-

рация Vantage Team Builder for Uniface отличается от остальных некоторой степенью ориентации на спиральную модель ЖЦ ПО за счет возможностей быстрого прототипирования, предоставляемых Uniface. Для описания проекта ИС используется достаточно большой набор диаграмм, конкретные варианты которого для наиболее распространенных конфигураций приведены ниже в табл. 5.1.

*Таблица 5.1*

**Диаграммы для описания проекта**

Тип диаграммы	Обозначение	Vantage Team Builder for ORACLE	Vantage Team Builder for Informix	Vantage Team Builder for Uniface
1	2	3	4	5
Сущность-связь	ERD	+	+	+
Потоков данных	DFD	+	+	+
Структур данных	DSD	+	+	+
Архитектуры системы	SAD	+	+	+
Потоков управления	CSD	+	+	+
Типов данных	DTD	+	+	+
Структуры меню	MSD	+		
Последовательности блоков	BSD	+		
Последовательности форм	FSD		+	+
Содержимого форм	FCD		+	+
Переходов состояний	STD	+	+	+
Структурных схем	SCD	+	+	+

При построении всех типов диаграмм обеспечивается контроль соответствия моделей синтаксису используемых методов, а также контроль соответствия одноименных элементов и их типов для различных типов диаграмм.

При построении DFD обеспечивается контроль соответствия диаграмм различным уровням декомпозиции. Контроль за правильностью верхнего уровня DFD осуществляется с помощью матрицы списков событий (ELM). Для контроля за декомпозицией составных потоков данных используется несколько вариантов их описания: в виде диаграмм структур данных (DSD) или в нотации БНФ (форма Бэкуса–Наура).

Для построения SAD используется расширенная нотация DFD, дающая возможность вводить понятия процессоров, задач и периферийных устройств, что обеспечивает наглядность проектных решений.

При построении модели данных в виде ERD выполняется ее нормализация и вводится определение физических имен элементов данных и таблиц, которые будут использоваться в процессе генерации физической схемы данных конкретной СУБД. Обеспечивается возможность определения альтернативных ключей сущностей и полей, составляющих дополнительные точки входа в таблицу (поля индексов), и мощности отношений между сущностями.

Наличие универсальной системы генерации кода, основанной на специфицированных средствах доступа к репозиторию проекта, позволяет поддерживать высокий уровень исполнения проектной дисциплины разработчиками: жесткий порядок формирования моделей; жесткая структура и содержимое документации; автоматическая генерация исходных кодов программ и т.д. – все это обеспечивает повышение качества и надежности разрабатываемых ИС.

Для подготовки проектной документации могут использоваться издательские системы FrameMaker, Interleaf или Word Perfect. Структура и состав проектной документации могут быть настроены в соответствии с заданными стандартами. Настройка выполняется без изменения проектных решений.

При разработке достаточно крупной ИС вся система в целом соответствует одному проекту как категории Vantage Team Builder. Проект может быть декомпозирован на ряд систем, каждая из которых соответствует некоторой относительно автономной подсистеме ИС и разрабатывается независимо от других. В дальнейшем системы проекта могут быть интегрированы.

Процесс проектирования ИС с использованием Vantage Team Builder реализуется в виде четырех последовательных фаз (стадий) – анализа, архитектуры, проектирования и реализации, при этом за конченные результаты каждой стадии полностью или частично переносятся (импортируются) в следующую фазу. Все диаграммы, кроме ERD, преобразуются в другой тип или изменяют вид в соответствии с особенностями текущей фазы. Так, DFD преобразуются в фазе архитектуры в SAD, DSD – в DTD. После завершения импорта логическая связь с предыдущей фазой разрывается, т.е. в диаграммы могут вноситься все необходимые изменения.

#### *Взаимодействие с другими средствами*

Конфигурация Vantage Team Builder for Uniface обеспечивает совместное использование двух систем в рамках единой технологи-

ческой среды проектирования, при этом схемы БД (SQL-модели) переносятся в репозиторий Uniface, и наоборот, прикладные модели, сформированные средствами Uniface, могут быть перенесены в репозиторий Vantage Team Builder. Возможные рассогласования между репозиториями двух систем устраняются с помощью специальной утилиты. Разработка экранных форм в среде Uniface выполняется на базе диаграмм последовательностей форм (FSD) после импорта SQL-модели. Технология разработки ИС на базе данной конфигурации показана на рис. 5.6.

Структура репозитория (хранящегося непосредственно в целевой СУБД) и интерфейсы Vantage Team Builder являются открытыми, что в принципе позволяет интеграцию с любыми другими средствами.

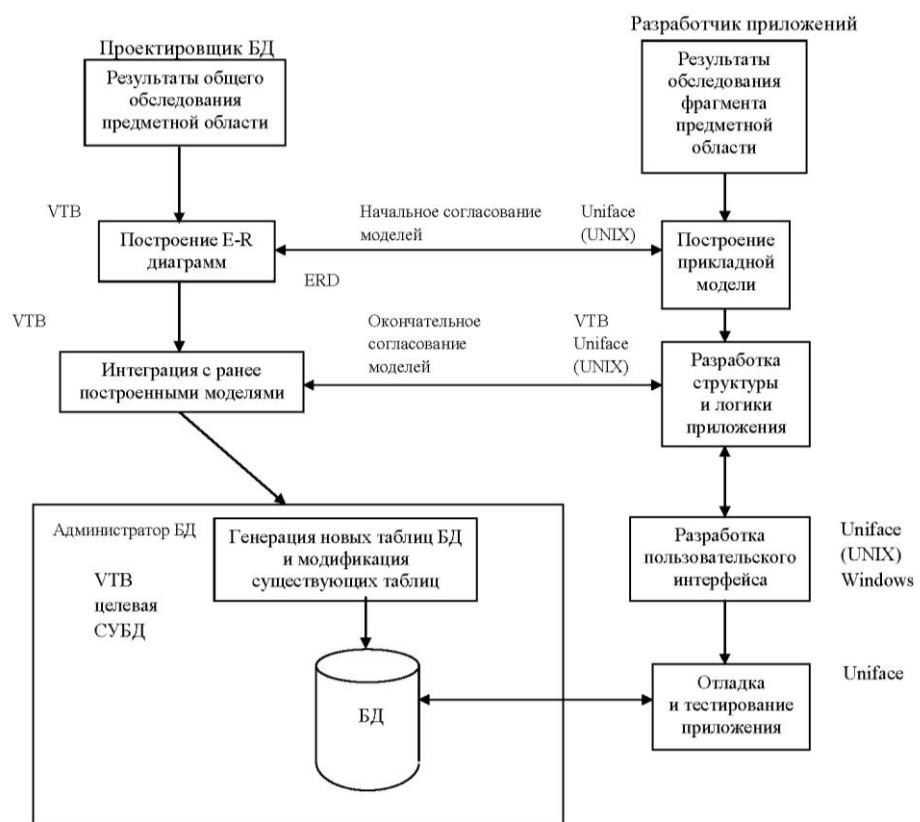


Рис. 5.6. Взаимодействие Vantage Team Builder и Uniface

### *Среда функционирования*

Vantage Team Builder функционирует на всех основных UNIX-платформах (Solaris, SCO UNIX, AIX, HP-UX) и VMS.

Vantage Team Builder можно использовать в конфигурации «клиент-сервер», при этом база проектных данных может располагаться на сервере, а рабочие места разработчиков могут быть клиентами.

## **5.4. Designer/2000 + Developer/2000**

CASE-средство Designer/2000 2.0 фирмы ORACLE является интегрированным CASE-средством, обеспечивающим в совокупности со средствами разработки приложений Developer/2000 поддержку полного ЖЦ ПО для систем, использующих СУБД ORACLE.

### *Структура и функции*

Designer/2000 представляет собой семейство методологий и поддерживающих их программных продуктов. Базовая методология Designer/2000 (CASE\*Method) – структурная методология проектирования систем, полностью охватывающая все этапы жизненного цикла ИС. В соответствии с этой методологией на этапе планирования определяются цели создания системы, приоритеты и ограничения, разрабатываются системная архитектура и план разработки ИС. В процессе анализа строятся модель информационных потребностей (диаграмма «сущность-связь»), диаграмма функциональной иерархии (на основе функциональной декомпозиции ИС), матрица перекрестных ссылок и диаграмма потоков данных.

На этапе проектирования разрабатывается подробная архитектура ИС, проектируются схема реляционной БД и программные модули, устанавливаются перекрестные ссылки между компонентами ИС для анализа их взаимного влияния и контроля за изменениями.

На этапе реализации создается БД, строятся прикладные системы, производится их тестирование, проверка качества и соответствия требованиям пользователей. Создаются системная документация, материалы для обучения и руководства пользователей. На этапах эксплуатации и сопровождения анализируются производительность и целостность системы, выполняется поддержка и при необходимости модификация ИС.

Designer/2000 обеспечивает графический интерфейс при разработке различных моделей (диаграмм) предметной области. В процессе построения моделей информация о них заносится в репозиторий. В состав Designer/2000 входят следующие компоненты:

- Repository Administrator – средства управления репозиторием (создание и удаление приложений, управление доступом к данным со стороны различных пользователей, экспорт и импорт данных);

- Repository Object Navigator – средства доступа к репозиторию, обеспечивающие многооконный объектно-ориентированный интерфейс доступа ко всем элементам репозитория;

– Process Modeller – средство анализа и моделирования деловой деятельности, основывающееся на концепциях реинжиниринга бизнес-процессов (BPR – Business Process Reengineering) и глобальной системы управления качеством (TQM – Total Quality Management);

– Systems Modeller – набор средств построения функциональных и информационных моделей проектируемой ИС, включающий средства для построения диаграмм «сущность-связь» (Entity-Relationship Diagrammer), диаграмм функциональных иерархий (Function Hierarchy Diagrammer), диаграмм потоков данных (Data Flow Diagrammer) и средство анализа и модификации связей объектов репозитория различных типов (Matrix Diagrammer);

– Systems Designer – набор средств проектирования ИС, включающий средство построения структуры реляционной базы данных (Data Diagrammer), а также средства построения диаграмм, отображающих взаимодействие с данными, иерархию, структуру и логику приложений, реализуемую хранимыми процедурами на языке PL/SQL (Module Data Diagrammer, Module Structure Diagrammer и Module Logic Navigator);

– Server Generator – генератор описаний объектов БД ORACLE (таблиц, индексов, ключей, последовательностей и т.д.). Помимо продуктов ORACLE, генерация и реинжиниринг БД могут выполняться для СУБД Informix, DB/2, Microsoft SQL Server, Sybase, а также для стандарта ANSI SQL DDL и баз данных, доступ к которым реализуется посредством ODBC;

– Forms Generator – генератор приложений для ORACLE Forms. Генерируемые приложения включают в себя различные экранные формы, средства контроля данных, проверки ограничений целостности и автоматические подсказки. Дальнейшая работа с приложением выполняется в среде Developer/2000;

Repository Reports – генератор стандартных отчетов, интегрированный с ORACLE Reports и позволяющий русифицировать отчеты, а также изменять структурное представление информации.

Репозиторий Designer/2000 представляет собой хранилище всех проектных данных и может работать в многопользовательском режиме, обеспечивая параллельное обновление информации несколькими разработчиками. В процессе проектирования автоматически поддерживаются перекрестные ссылки между объектами словаря и могут генерироваться более 70 стандартных отчетов о моделируемой предметной области. Физическая среда хранения репозитория – база данных ORACLE.

Генерация приложений, помимо продуктов ORACLE, выполняется также для Visual Basic.

#### *Взаимодействие с другими средствами*

Designer/2000 можно интегрировать с другими средствами, используя открытый интерфейс приложений API (Application Programming Interface). Кроме того, можно использовать средство ORACLE CASE Exchange для экспорта/импорта объектов репозитория с целью обмена информацией с другими CASE-средствами.

Developer/2000 обеспечивает разработку переносимых приложений, работающих в графической среде Windows, Macintosh или Motif. В среде Windows интеграция приложений Developer/2000 с другими средствами реализуется через механизм OLE и управляющие элементы VBX. Взаимодействие приложений с другими СУБД (DB/2, DB2/400, Rdb) реализуется с помощью средств ORACLE Client Adapter для ODBC, ORACLE Open Gateway и API.

### **5.5. Локальные средства (ERwin, BPwin, S-Designor, CASE.Аналитик)**

ERwin – средство концептуального моделирования БД [24], использующее методологию IDEF1X (см. подраздел 2.5). ERwin реализует проектирование схемы БД, генерацию ее описания на языке целевой СУБД (ORACLE, Informix, Ingres, Sybase, DB/2, Microsoft SQL Server, Progress и др.) и реинжиниринг существующей БД. ERwin выпускается в нескольких различных конфигурациях, ориентированных на наиболее распространенные средства разработки приложений 4GL. Версия ERwin/OPEN полностью совместима со средствами разработки приложений PowerBuilder и SQLWindows и позволяет экспортить описание спроектированной БД непосредственно в репозитории данных средств.

Для ряда средств разработки приложений (PowerBuilder, SQLWindows, Delphi, Visual Basic) выполняется генерация форм и прототипов приложений.

Сетевая версия Erwin ModelMart обеспечивает согласованное проектирование БД и приложений в рамках рабочей группы.

BPwin – средство функционального моделирования, реализующее методологию IDEF0.

Возможные конфигурации и ориентировочная стоимость средств (без технической поддержки) приведены в табл. 5.2.

**Возможные конфигурации  
и ориентировочная стоимость средств**

Конфигурация	Стоимость, \$
ERwin/ERX	3,295
Bpwin	2,495
ERwin/ERX for PowerBuilder, Visual Basic, Progress	3,495
ERwin/ERX for Delphi	4,295
ERwin/Desktop for PowerBuilder, Visual Basic	495
ERwin/ERX for SQLWindows / Designer/2000 / Solaris	3,495 / 5,795 / 6,995
ModelMart 5 / 10 user	11,995 / 19,995
Erwin/OPEN for ModelMart	3,995

S-Designor 4.2 представляет собой CASE-средство для проектирования реляционных баз данных. По своим функциональным возможностям и стоимости он близок к CASE-средству ERwin, отличаясь внешне используемой на диаграммах нотацией. S-Designor реализует стандартную методологию моделирования данных и генерирует описание БД для таких СУБД, как ORACLE, Informix, Ingres, Sybase, DB/2, Microsoft SQL Server и др. Для существующих систем выполняется реинжиниринг БД.

S-Designor совместим с рядом средств разработки приложений (PowerBuilder, Uniface, TeamWindows и др.) и позволяет экспортить описание БД в репозитории данных средств. Для PowerBuilder выполняется также прямая генерация шаблонов приложений.

CASE.Аналитик 1.1 является практически единственным в настоящее время конкурентоспособным отечественным CASE-средством функционального моделирования и реализует построение диаграмм потоков данных.

База данных проекта реализована в формате СУБД Paradox и является открытой для доступа.

С помощью отдельного программного продукта (Catherine) выполняется обмен данными с CASE-средством ERwin. При этом из проекта, выполненного в CASE.Аналитике, экспортируется описание структур данных и накопителей данных, которое по определенным правилам формирует описание сущностей и их атрибутов.

## **5.6. Объектно-ориентированные CASE-средства (Rational Rose)**

Rational Rose – CASE-средство фирмы Rational Software Corporation (США) – предназначено для автоматизации этапов анализа и проектирования ПО, а также для генерации кодов на различных языках и выпуска проектной документации. Rational Rose использует синтез-методологию объектно-ориентированного анализа и проектирования, основанную на подходах трех ведущих специалистов в данной области: Буча, Рамбо и Джекобсона. Разработанная ими универсальная нотация для моделирования объектов (UML – Unified Modeling Language) претендует на роль стандарта в области объектно-ориентированного анализа и проектирования. Конкретный вариант Rational Rose определяется языком, на котором генерируются коды программ (C++, Smalltalk, PowerBuilder, Ada, SQLWindows и ObjectPro). Основной вариант – Rational Rose/C++ – позволяет разрабатывать проектную документацию в виде диаграмм и спецификаций, а также генерировать программные коды на C++. Кроме того, Rational Rose содержит средства реинжиниринга программ, обеспечивающие повторное использование программных компонент в новых проектах.

### *Структура и функции*

В основе работы Rational Rose лежит построение различного рода диаграмм и спецификаций, определяющих логическую и физическую структуры модели, ее статические и динамические аспекты. В их число входят диаграммы классов, состояний, сценариев, модулей, процессов.

В составе Rational Rose можно выделить шесть основных структурных компонент: репозиторий, графический интерфейс пользователя, средства просмотра проекта (browser), средства контроля проекта, средства сбора статистики и генератор документов. К ним добавляются генератор кодов (индивидуальный для каждого языка) и анализатор для C++, обеспечивающий реинжиниринг – восстановление модели проекта по исходным текстам программ.

Репозиторий представляет собой объектно-ориентированную базу данных. Средства просмотра обеспечивают «навигацию» по проекту, в том числе перемещение по иерархиям классов и подсистем, переключение от одного вида диаграмм к другому и т.д. Средства контроля и сбора статистики дают возможность находить и устранять ошибки по мере развития проекта, а не после завершения

его описания. Генератор отчетов формирует тексты выходных документов на основе содержащейся в репозитории информации.

Средства автоматической генерации кодов программ на языке C++, используя информацию, содержащуюся в логической и физической моделях проекта, формируют файлы заголовков и файлы описаний классов и объектов. Создаваемый таким образом скелет программы может быть уточнен путем прямого программирования на языке C++. Анализатор кодов C++ реализован в виде отдельного программного модуля. Его назначение состоит в том, чтобы создавать модули проектов в форме Rational Rose на основе информации, содержащейся в определяемых пользователем исходных текстах на C++. В процессе работы анализатор осуществляет контроль правильности исходных текстов и диагностику ошибок. Модель, полученная в результате его работы, может целиком или фрагментарно использоваться в различных проектах. Анализатор обладает широкими возможностями настройки по входу и выходу. Например, можно определить типы исходных файлов, базовый компилятор, задать, какая информация должна быть включена в формируемую модель и какие элементы выходной модели следует выводить на экран. Таким образом, Rational Rose/C++ обеспечивает возможность повторного использования программных компонент.

В результате разработки проекта с помощью CASE-средства Rational Rose формируются следующие документы:

- диаграммы классов;
- диаграммы состояний;
- диаграммы сценариев;
- диаграммы модулей;
- диаграммы процессов;
- спецификации классов, объектов, атрибутов и операций
- заготовки текстов программ;
- модель разрабатываемой программной системы.

Последний из перечисленных документов является текстовым файлом, содержащим всю необходимую информацию о проекте (в том числе необходимую для получения всех диаграмм и спецификаций).

Тексты программ являются заготовками для последующей работы программистов. Они формируются в рабочем каталоге в виде файлов типов .h (заголовки, содержащие описания классов) и .cpp (заготовки программ для методов). Система включает в програм-

мные файлы собственные комментарии, которые начинаются с последовательности символов //##. Состав информации, включаемой в программные файлы, определяется либо по умолчанию, либо по усмотрению пользователя. В дальнейшем эти исходные тексты разбиваются программистами в полноценные программы.

### *Взаимодействие с другими средствами и организация групповой работы*

Rational Rose интегрируется со средством PVCS для организации групповой работы и управления проектом и со средством SoDA – для документирования проектов. Интеграция Rational Rose и SoDA обеспечивается средствами SoDA.

Для организации групповой работы в Rational Rose возможно разбиение модели на управляемые подмодели. Каждая из них независимо сохраняется на диске или загружается в модель. В качестве подмодели может выступать категория классов или подсистема.

Для управляемой подмодели предусмотрены операции:

- загрузка подмодели в память;
- выгрузка подмодели из памяти;
- сохранение подмодели на диске в виде отдельного файла;
- установка защиты от модификации;
- замена подмодели в памяти на новую.

Наиболее эффективно групповая работа организуется при интеграции Rational Rose со специальными средствами управления конфигурацией и контроля версий (PVCS). В этом случае защита от модификации устанавливается на все управляемые подмодели, кроме тех, которые выделены конкретному разработчику. Признак защиты от записи устанавливается для файлов, которые содержат подмодели, поэтому при считывании «чужих» подмоделей защита их от модификации сохраняется и случайные воздействия окажутся невозможными.

## **Заключение**

По прогнозам IDC рынок ТС ПО в ближайшее пятилетие ожидает устойчивый рост в среднем на 6,3 % в год. Определяющим фактором для развития этой тенденции является стремление компаний-разработчиков повысить продуктивность своей работы, сократить сроки вывода новых продуктов на рынок, контролировать расходы и быстро получать отдачу от инвестиций. Достижению этих целей способствует использование сред разработки, позволяющих снизить сложность процессов создания ПО, увеличить их эффективность, уменьшить затраты на разработку и максимально использовать потенциал новых технологий.

Аналитики сходятся на том, что основное направление развития инструментальных средств – их сквозная интеграция, переход от частично интегрированных средств к интегрированным комплексам, объединяющим возможности управления требованиями, моделирования, разработки, тестирования, управления конфигурацией и изменениями и развертывания приложений. В ближайшие годы такие комплексы, помимо перечисленных возможностей, будут включать в себя средства управления потоками работ и проектами. Рынок таких инструментальных средств ожидает глобальная консолидация, обещающая принести значительные выгоды разработчикам. В то же время проблема обоснованного выбора и эффективного применения ТС ПО в крупномасштабных проектах остается актуальной. Невозможно достичь удовлетворительных результатов от применения даже самых совершенных технологий, если они применяются бессистемно, разработчики не обладают необходимой квалификацией для работы с ними и сам проект выполняется и управляемся хаотически.

Систематический, обоснованный подход к выбору и применению ТС ПО может сократить время и повысить качество разработки ПО, обеспечить высокую степень его независимости от конкретных разработчиков, а также снизить затраты на разработку и сопровождение ПО.

# Варианты заданий на курсовое проектирование

Номер варианта	Номер структуры данных	Вариант выбора функций	Вариант диаграммы	Определение места таблиц БД	Дополнительные требования
1	1	1	1	+	5,7
2	2	1	1	-	4,7
3	3	2	2	+	3,8
4	4	2	2	-	6,8
5	5	3	3	+	2,7
6	6	3	3	-	1,8
7	1	4	1	+	1,7
8	2	'4	1	-	2,7
9	3	5	2	+	1,8
10	4	5	2	-	6,8
11	5	1	3	+	2,7
12	6	1	3	-	3,8
13	7	2	1	+	6,8
14	8	2	1	-	1,6
15	9	2	2	+	2,5
16	10	3	2	-	6,7
17	11	3	3	+	4,5
18	7 •	4	3	-	6,8
19	8	4	1	+	1,6
20	9	5	1	-	2,5
21	10	5	2	+	6,7
22	11	2	2	-	4,5
23	1	2	3	+	3,7
24	2	3	3	-	1,7
25	3	3	1	+	3,8
26	4	4	1	-	6,8
27	5	4	2	+	2,7
28	6	5	2	-	1,8
29	1	5	3	+	5,7
30	2	5	3	-	4,7
31	3	2	1	+	3,8
32	4	2	1	-	6,8
33	5	1	2	+	2,7
34	6	1	2	-	1,8
35	7	3	3	+	6,8

*Продолжение табл.*

Номер варианта	Номер структуры данных	Вариант выбора функций	Вариант диаграммы	Определение места таблиц БД	Дополнительные требования
36	8	3	3	—	1,6
37	9	1	1	+	2,5
38	10	4	1	—	6,7
39	11	4	2	+	4,5
40	7	5	2	—	6,8
41	8	5	3	+	1,6
42	9	2	3	—	2,5
43	10	3	1	+	6,7
44	11	3	1	—	4,5
45	12	4	2	+	1,5
46	13	4	2	—	5,7
47	14	1	3	+	1,8
48	15	1	3	—	3,5
49	16	2	1	+	2,8
50	17	2	1	—	3,8
51	12	4	2	+	1,5
52	13	3	2	—	5,7
53	14	3	3	+	1,8
54	15	4	3	—	3,5
55	16	5	1	+	2,8
56	17	5	1	—	3,8
57	18	1	2	+	2,7
58	19	2	2	-	4,5
59	20	2	3	+	1,7
60	21	4	3	—	3,7
61	22	4	1	+	4,5
62	18	3	1	—	2,7
63	19	3	2	+	4,5
64	20	5	2	—	1,7
65	21	5	3	+	3,7
66	22	1	3	—	4,5
67	12	2	1	+	1,5
68	13	2	1	—	5,7
69	14	4	2	+	1,8
70	15	5	2	—	3,5
71	16	3	3	+	2,8
72	17	3	3	—	3,8
73	18	1	1	+	2,7

*Окончание табл.*

Номер варианта	Номер структуры данных	Вариант выбора функций	Вариант диаграммы	Определение места таблиц БД	Дополнительные требования
74	19	4	1	–	4,5
75	20	5	2	+	1,7
76	1	5	1	–	5,7
77	2	4	1	+	4,7
78	3	4	2	–	3,8
79	4	2	2	+	6,8
80	5	2	3	–	2,7

# Варианты структур данных

## Структура данных 1

Абитуриенты по факультетам и специальностям			
Фамилия	Экзаменационные оценки		Сумма баллов
	Математика	Физика	
Специальности			
Код специальности	Наименование		Факультет

Примечание. Выполнить сортировку по убыванию суммы баллов с группировкой по факультетам и специальностям; вычислить количество абитуриентов по факультетам, специальностям, по вузу.

## Структура данных 2

Сотрудники		
Подразделение	ФИО	Оклад (О)
Начисления основной зарплаты сотрудников		
Месяц, количество рабочих дней сотрудника (Д) в месяце (M)	Количество рабочих дней сотрудника (Д)	Начислено = = (О * Д) / M

Примечание. Выполнить группировки по подразделениям, сотрудникам, месяцам с вычислением итоговых сумм.

## Структура данных 3

Материальные ценности предприятия					
Номенклатурный номер	Наименование	Единица измерения			
Журнал учета движения материальных ценностей					
Год	Месяц	Остаток на начало года	Приход за месяц	Расход за месяц	Остаток за месяц

Примечание. Выполнить группировки по наименованиям, по месяцам, по годам, с вычислением итоговых сумм по группам.

### Структура данных 4

Фирма			
Номер/Наименование свидетельства фирмы		Генеральный директор	
Информация о доходах фирм от операций с акциями			
Дата	Курс акций	Количество акций	Доход фирмы
	Покупка/Продажа	Куплено/Продано	

Примечание. Выполнить группировку по фирмам, месяцам с вычислением итоговых сумм.

### Структура данных 5

Туры				
Название	Количество дней	Стоимость на одного человека	Пункт назначения	
Информация о деятельности туристической фирмы				
Месяц	Туры	Группа	Количество участников	Сумма доходов с группы

Примечание. Выполнить группировку по названию тура, месяцам; вычислить итоговые суммы доходов по туром, по месяцам.

### Структура данных 6

Договоры на выполнение работ по заказам												
Номер договора	Название	Заказчик	Сроки работ (месяц)		Стоимость работ по договору	Фонд оплаты труда (ФОТ)	Матери-алы, %	Наклад-ные расхо-ды, %				
			начало	окончание								
Статьи расхода по договорам												
Сумма ФОТ, руб.		Сумма на материалы, руб.		Сумма на накладные расходы, руб.		Сумма на прочие расходы, руб.						

Примечание. Выполнить группировку по месяцам заключения договоров и по заказчикам с вычислением итоговых сумм по каждой статье дохода и расхода.

### **Структура данных 7**

Клиенты банка				
Название	Адрес	Номер счета		
Расчетные счета клиентов				
Номер счета	Дата	Сумма вклада	Сумма, снятая со счета	Остаток на конец месяца

Примечание. Выполнить группировку по клиентам и по датам с вычислением итоговых сумм по каждой статье.

### **Структура данных 8**

Пациент				
Номер полиса	ФИО	Адрес	Место работы	
Медицинские услуги для пациентов в стационаре				
Дата поступления	Дата	Оплата за одного человека	Сумма выписки	Дата

Примечание. Выполнить группировку по предприятиям, пациентам с вычислением итоговых сумм; вычислять количество поступающих пациентов по датам.

### **Структура данных 9**

Договоры на выполнение работ				
Номер договора	Название	Срок окончания	Сумма договора	Фонд оплаты труда, %
Сотрудник				
ФИО	Подразделение	Коэффициент трудового участия		Сумма по договору сотруднику

Примечание. Выполнить группировку по подразделениям, по сотрудникам; рассчитать сумму по договорам для каждого подразделения, сумму по всем договорам для каждого сотрудника.

## Структура данных 10

Карточка товара					
Номер карточки	Наименование товара	Единица измерения	Номер склада		
Данные о приходе товаров					
Номер документа	Товар	Дата	Цена	Количество	Сумма

Примечание. Выполнить группировку по номеру карточки, по номеру склада; рассчитать сумму товаров по каждой карточке, общую сумму товаров на каждом складе.

## Структура данных 11

Рабочие					
ФИО	Номер цеха				
Наряды рабочих					
Номер	Месяц	Технологическая операция	Количество	Расценка	Сумма

Примечание. Выполнить группировку по ФИО рабочего и по цехам с вычислением итогов; вычислить сумму по всем нарядам для каждого рабочего за месяц, количество каждой технологической операции за месяц.

## Структура данных 12

Состав персонального компьютера					
Инвентарный номер	Память, руб.	Системная плата, руб.	Процессор, руб.	Монитор, руб.	Общая стоимость, руб.
Оснащение подразделений компьютерами					
Номер подразделения		Название	Материально ответственное лицо	Инвентарный номер ПК	

Примечание. Выполнить группировку по подразделениям; рассчитать общую стоимость оснащения компьютерами подразделения, количество компьютеров в каждом подразделении.

### Структура данных 13

Сорта картофеля			
Название сорта	Срок хранения		
Статистические данные по урожаям картофеля			
Год	Номер куста	Количество клубней	Масса клубней всего куста

Примечание. Отчет 1 (сорт, срок хранения, средняя масса клубней); отчет 2 (год; сорт, урожайность с 10 кустов); отчет 3 (год, сорт, среднее количество клубней куста, средняя масса клубней).

### Структура данных 14

Цех					
Номер	Название	Начальник			
Продукция цехов по месяцам					
Название	Единица измерения	Себестоимость	Количество	Цена продажи	Прибыль
			изготовлено	реализовано	

Примечание. Продукция весовая, выполнить группировку с вычислением итогов по месяцам, по цехам; вычислить итоговые суммы по отчету.

### Структура данных 15

Водитель АТП					
Табельный номер	ФИО	Категория	Коэффициент надбавки за классность		
Заработка плата по месяцам					
Водитель	Номер автомобиля	Пробег, км	Тариф за пробег, руб./км	Время работы, ч	Повременный тариф, руб./ч
					Всего начислено, руб.

Примечание. Выполнить группировку с вычислением итогов по водителям, по месяцам; вычислить итоговые суммы по отчету.

## Структура данных 16

Автомобиль АТП		Отчет о расходовании горючего по месяцам						
Авто- мобиль	Марка горючего	Норма расхода, л/100 км	Цена, руб / л	Пробег автомо- билия, км	Расход горючего, л		Оплата, руб.	
					по норме	фактиче- ский	план	фактиче- ский

Примечание. Выполнить группировку с вычислением итогов по автомобилям, по месяцам; вычислить разницу в литрах и рублях, итоговые суммы по отчету.

## Структура данных 17

Объект строительства	
Название	Заказчик

Расход строительных материалов (СМ)						
Объект строи- тельства	Строительные материалы	Единица измерения	Цена	Расход СМ (количество)		Сумма, руб.
				по норме	факти- ческий	

Примечание. Выполнить группировку с вычислением итогов по объектам, заказчикам; вычислить разницу и итоговые суммы по отчету.

### **Структура данных 18**

Сотрудник	Стаж работы	Должность	Оклад (О)		
Оплата больничных листов (БЛ)					
Месяц	Количество рабочих дней в месяце (M)	Сотрудник	Количество нерабочих дней сотрудника (H)	% от оклада (П), П = 50, 75 или 100 в зависимости от стажа	Оплата БЛ-О-Н.П/(100-M)

### **Структура данных 19**

Сотрудник		Табельный номер				
Начисления и отчисления при повременной оплате						
Месяц	Со-труд-ник	Виды работ	Расценки, руб/ч	Отработано, ч	Начислено, руб.	Отчисления в пенсионный фонд, % от начислений

Примечание. Вычислить суммы начислений и отчислений по сотрудникам, месяцам, видам работ.

### **Структура данных 20**

Расписание занятий				
День недели	Порядковый номер занятия в день	Группа	Преподаватель	Аудитория

Примечание. Вычислить количество занятий в неделю у каждого преподавателя, у каждой группы, в каждой аудитории; формировать справку о свободных аудиториях в заданный день и на заданный номер занятия.

## Структура данных 21

Поручения преподавателям				
Преподаватели	Дисциплины	Виды занятий	Часов по видам занятий	Часов по дисциплине

Примечание. Вычислить объемы учебной работы по кафедрам, преподавателям, дисциплинам и видам занятий.

## Структура данных 22

Пациент	Год рождения	Рецепты				
Пациент	Год	Месяц	Препарат	Количество	Цена	

Примечание. Вычислить расходы на лекарства в месяц и за год по пациентам, возрасту, препаратам.

Варианты выбора пользовательских функций				Вариант диаграммы		Вариант определения каталога с файлами БД	
1	Кнопки	4	Переключатели	1	Столбиковая	+	Определяется в программе
2	Метки	5	«Закладки»	2	Полигон	-	Не определяется в программе
3	Меню			3	Круговая		

Дополнительные требования к программе:

1. Формирование текста запроса в программе.
2. Использование параметрических запросов.
3. Возможность задания текста запроса пользователем.
4. Обеспечение вывода данных в заданном числовом диапазоне.
5. Обеспечение возможности поиска данных.
6. Использование специальных компонентов для ввода дат.
7. Использование специальных компонентов для ввода целых чисел.
8. Выделение цветом данных о доходах (расходах), прибыли (убытках), поступлении (расходовании), о превышении планового (заданного) значения.

## **Список литературы**

1. Петрухин, В. А. Методы и средства программной инженерии / В. А. Петрухин, Е. М. Лаврищева. – URL: <http://www.intuit.ru/department/se/swebok/12/2.html>
2. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, А. Джакобсон. – М. : ДМК Пресс, 2001.
3. Иванова, Г. С. Технология программирования / Г. С. Иванова. – М. : Изд-во МГТУ им. Баумана, 2002.
4. Иванова, Г. С. Объектно-ориентированное программирование / Г. С. Иванова, Т. Н. Ничушкина, Е. К. Пугачев. – М. : Изд-во МГТУ им. Баумана, 2001.
5. Кватрани, Т. Rational Rose и UML. Визуальное моделирование / Т. Кватрани. – М. : ДМК Пресс, 2001.
6. Ларман, К. Применение UML и шаблонов проектирования / К. Ларман. – М. : Вильямс, 2001.
7. Леоненков, А. Самоучитель UML / А. Леоненков. – СПб. : БХВ-Петербург, 2001.
8. URL: <http://rudocs.exdat.com/docs/index-46035.html>
9. URL: <http://citforum.ru/programming/application/program/2.shtml>
10. URL: <http://www.ca.com>
11. URL: <http://www.rational.com>
12. URL: <http://www.borland.com>
13. URL: <http://www.oracle.com>
14. Вендров, А. М. Проектирование программного обеспечения экономических информационных систем : учеб. / А. М. Вендров. – 2-е изд., перераб. и доп. – М. : Финансы и статистика, 2006.
15. Вендров, А. М. Проектирование программного обеспечения экономических информационных систем : учеб. / А. М. Вендров. – М. : Финансы и статистика, 2000.
16. Коберн, А. Быстрая разработка программного обеспечения / А. Коберн ; пер. с англ. – М. : ЛОРИ, 2002.
17. Коберн, А. Современные методы описания функциональных требований к системам / А. Коберн ; пер. с англ. – М. : ЛОРИ, 2002.
18. Конноли, Т. Базы данных: проектирование, реализация и сопровождение. Теория и практика / Т. Конноли, К. Бегг ; пер. с англ. – 3-е изд. – М. : Вильямс, 2003.
19. Крачтен, Ф. Введение в Rational Unified Process / Ф. Крачтен ; пер. с англ. – М. : Вильямс, 2002.

20. Ларман, К. Применение UML и шаблонов проектирования / К. Ларман ; пер. с англ. – 2-е изд. – М. : Вильямс, 2002.
21. Леффингуэлл, Д. Принципы работы с требованиями к программному обеспечению. Унифицированный подход / Д. Леффингуэлл, Д. Уидриг ; пер. с англ. – М. : Вильямс, 2002.
22. Маклаков, С. В. Создание информационных систем с AllFusion Modeling Suite / С. В. Маклаков. – М. : Диалог-МИФИ, 2003.
23. Worldwide Analysis, Modeling, and Design Tools Forecast and Analysis. – 2002–2006. – IDC, URL: <http://www.idc.com>, 2002.
24. Worldwide Analysis, Modeling, Design and Construction Tools Competitive Analysis. – 2003: 2002 Shares and Current Outlook. – IDC, URL: <http://www.idc.com>, 2003.
25. Worldwide Application Development and Deployment Forecast Summary. – 2003–2007. – IDC, URL: <http://www.idc.com>, 2003.
26. URL: <http://citforum.ru/programming/application/program/index.shtml#v>

*Учебное издание*

**Черушева Татьяна Вячеславовна**

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Редактор *Н. А. Сидельникова*  
Компьютерная верстка *Ю. В. Ануровой*

Подписано в печать 29.10.2014.  
Формат 60×84<sup>1</sup>/<sub>16</sub>. Усл. печ. л. 9,99.  
Заказ № 975. Тираж 50.

---

Издательство ПГУ  
440026, Пенза, Красная, 40.  
Тел./факс: (8412) 56-47-33; e-mail: [iic@pnzgu.ru](mailto:iic@pnzgu.ru)