Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования «Кубанский государственный университет»
(ФГБОУ ВО «КубГУ»)

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению лабораторных работ по дисциплине
«Технологии проектирования программного обеспечения»

для бакалавров направлений подготовки 01.03.02 «Прикладная математика и информатика», 02.03.03 «Математическое обеспечение и администрирование информационных систем» профиль «Технология программирования», 09.03.03 «Прикладная информатика» профиль «Прикладная информатика в экономике»

Краснодар,
2024

УДК 629.113(071):004.01:004.4

Методические указания к выполнению лабораторных работ по дисциплине «Технологии проектирования программного обеспечения», для бакалавров направлений подготовки 01.03.02 «Прикладная математика и информатика», 02.03.03 «Математическое обеспечение и администрирование информационных систем» профиль «Технология программирования», 09.03.03 «Прикладная информатика» профиль «Прикладная информатика в экономике» / Составители: доц. каф. ИТ Полетайкин А.Н., доц. каф. ИТ Добровольская Н.Ю. − Краснодар: КубГУ, 2024. – 107 с.

Методические указания содержат теоретические положения, задания и методические указания к выполнению лабораторных работ по дисциплине «Технологии проектирования программного обеспечения».

Составители: доц. каф. ИТ, к.т.н., доц. Полетайкин А.Н.

доц. каф. ИТ, к.пед.н., доц. Добровольская Н.Ю.

СОДЕРЖАНИЕ

1 Общие положения

Дисциплина «Технологии проектирования программного обеспечения» рассматривает методы и инструментальные средства программной инженерии для решения задач создания качественного программного обеспечения в разных сферах деятельности человека с применением современных информационных и компьютерных технологий.

Изучаются основные понятия, методы и модели программной инженерии, составляющие процесса разработки программного обеспечения, управление требованиями к программной системе, конфигурационное управление программным продуктом, методы, способы и порядок тестирования программного обеспечения, управление версиями и сборками. Рассматриваются и применяются на практике методы, способы и инструментальные средства для анализа предметных областей, постановки задачи, формулирования требований к программе, компилятивной сборки и тестирования специального программного обеспечения для экономики и современного бизнеса.

1.1 Организация занятий по курсу

Дисциплина «Технологии проектирования программного обеспечения» читается студентам 02.03.03 «Математическое обеспечение и администрирование информационных систем» профиль «Технология программирования» и 09.03.03 «Прикладная информатика» профиль «Прикладная информатика в экономике» на третьем курсе обучения, а студентам 01.03.02 «Прикладная математика и информатика» на четвертом курсе обучения.

Целью курса является формирование у студентов знаний, умений и практических навыков в области анализа и системного представления бизнес-процессов, разработки технического задания на создание специального программного обеспечения для решения поставленных задач, организации программного процесса на всех стадиях жизненного цикла программного обеспечения (ПО).

Предметом учебной дисциплины являются методы, подходы и инструментальные средства программной инженерии и технологии проектирования ПО.

Задачами дисциплины является получение представления о жизненном цикле многоверсионного программного продукта, а также приобретения навыков применения знаний и умений, приобретенных в бакалавриате, для создания сложных программных проектов, отвечающих требованиям современного бизнеса.

Изучаемый курс предусматривает выполнение 9 лабораторных работ и предполагает создание программной системы (ПС) и её специального программного обеспечения для реализации обработки информации на ЭВМ в рамках заданного бизнес-процесса (см. приложение А). Рабочая схема выполнения лабораторных занятий представлена в табл. 1.

Таблица 1. Рабочая схема выполнения лабораторных работ

№ л. р.	Тема лабораторной работы: материал для изучения и характер выполняемых работ	Неделя выполнения
1.	Анализ предметной области: выбор и утверждение индивидуальной темы, системный анализ объекта и процесса информатизации, характеристика решения задач и выделение ее недостатков, обоснование необходимости усовершенствования существующего решения задач	1 – 2
2.	Анализ существующих компьютерных разработок: системное описание существующих подобных ПС, сравнительная характеристика описанных систем	3
3.	Техническое задание на создание ПС: назначение и общая цель создания программы, структура программы и состав функциональных задач, функциональные и нефункциональные требования к программе, моделирование требований на языке UML	4 – 5
4.	Проектирование функциональной структуры ПС: построение и документирование функциональной модели разрабатываемого ПО в виде контекстной диаграммы и ее декомпозиции в нотации IDEF0	6 – 7
5.	Объектно-ориентированный подход к проектированию: Проектирование функциональной структуры разрабатываемого ПО в нотации UML	8 – 9
6.	Проектирование базы данных ПС: изучение программных средств для разработки моделей информационной базы ПС, проработка методов нормализации отношений в БД, создание БД ПС при помощи выбранной СУБД	10 – 11
7.	Разработка программного обеспечения ПС: технология программирования прикладных задач, разработка интерфейсной части ПС при помощи современных средств разработки, построение диаграмм UML физической реализации разработанной ПС, анализ производительности программы, оценивание эффективности кода	12 – 13
8.	Тестирование ПС: проверка работоспособности программы, сценарное, регрессионное, нагрузочное, пользовательское тестирование	14 – 15
9.	Разработка баг-репорта ПС: поиск и классификация дефектов программного продукта, составление баг-репортов	16
10.	Документирование и развертывание ПС: освоение методики документирования ПС, разработка функциональной спецификации ПС и руководства пользователя	17

По каждой лабораторной работе оформляется отчет. Отчеты сдаются на проверку руководителю в течение курса по мере их выполнения, и защищаются студентами в установленном порядке.

При защите отчета студенту могут быть заданы вопросы и дополнительные задания по сути лабораторной работы, в том числе из списка контрольных вопросов к данной лабораторной работе. При неудовлетворительной оценке знаний студента по теме данного отчета, студент возвращается к повторному изучению соответствующих материалов, после чего допускается к повторной защите. Неудовлетворительно выполненный отчет также возвращается на доработку. Требования к содержанию отчетов о выполнении лабораторных работ представлены в разделе 1.2.

1.2 Требования к содержанию отчетов о выполнении лабораторных работ

В процессе изучения курса «Технологии проектирования программного обеспечения» должны быть подготовлены и сданы десять отчетов о выполнении лабораторных работ (см. таблицу 1). Лабораторные работы выполняются в соответствии с заданием (раздел 2) в рамках индивидуальной темы.

Отчет должен содержать заголовок, тему лабораторной работы, цель, задание, индивидуальную тему, описание хода выполнения работы, необходимые прикладные материалы (схемы, макеты документов и т.п.), в соответствии с требованиями к содержанию, представленными в соответствующем подразделе раздела 2, и выводы по работе.

1.3 Требования к оформлению отчетной документации

Отчет должен быть выполнен печатным способом с использованием компьютера и принтера на одной стороне листа белой бумаги формата А4 через полтора интервала. Для создания текста работы рекомендуется использовать текстовый редактор MS Word. Шрифт на протяжении всего документа должен быть одинаковый: Times New Roman 14-го размера, за исключением оформления иллюстраций, таблиц и формул, в которых допускается использовать шрифт меньшего размера. Минимально допустимый размер шрифта – 12.

Текст отчета следует печатать, соблюдая следующие размеры полей: слева – 25-30 мм, справа – 10 мм, сверху и снизу – 20 мм.

При оформлении текста работы следует использовать абзацный отступ, который должен составлять 15-17 мм от левого поля документа

Вне зависимости от способа выполнения отчета качество напечатанного текста и оформления иллюстраций, таблиц, распечаток с ПЭВМ должно удовлетворять требованию их четкого воспроизведения.

Нумерация страниц отчета. Страницы отчета следует нумеровать арабскими цифрами, соблюдая сквозную нумерацию по всему тексту отчета. Номер страницы проставляют в центре нижней части листа без точки. Титульный лист включают в общую нумерацию страниц отчета. Номер страницы на титульном листе не проставляют.

Иллюстрации и таблицы, расположенные на отдельных листах, включают в общую нумерацию страниц отчета.

Иллюстрации. Иллюстрации (чертежи, графики, схемы, компьютерные распечатки, диаграммы, фотоснимки) следует располагать в отчете непосредственно после текста, в котором они упоминаются впервые, или на следующей странице. Иллюстрации могут быть в компьютерном исполнении, в том числе и цветные. На все иллюстрации должны быть даны ссылки в отчете.

Иллюстрации следует нумеровать арабскими цифрами сквозной нумерацией. Если рисунок один, то он обозначается “Рисунок 1”. Слово “Рисунок” и его наименование располагают посередине строки. Допускается нумеровать иллюстрации в пределах раздела. В этом случае номер иллюстрации состоит из номера раздела и порядкового номера иллюстрации, разделенных точкой. Например, Рисунок 1.1. Иллюстрации, при необходимости, могут иметь наименование и пояснительные данные (подрисуночный текст). Слово “Рисунок” и наименование помещают после пояснительных данных и располагают следующим образом: Рисунок 1.1 — Функциональная схема.

При ссылках на иллюстрации следует писать “... в соответствии с рисунком 2” при сквозной нумерации и “... в соответствии с рисунком 1.2” при нумерации в пределах раздела.

Таблицы применяют для лучшей наглядности и удобства сравнения показателей. Название таблицы, при его наличии, должно отражать ее содержание, быть точным, кратким. Название таблицы следует помещать над таблицей слева, без абзацного отступа в одну строку с ее номером через тире.

При переносе части таблицы название помещают только над первой частью таблицы, нижнюю горизонтальную черту, ограничивающую таблицу, не проводят.

Таблицу следует располагать в отчете непосредственно после текста, в котором она упоминается впервые, или на следующей странице. На все таблицы должны быть ссылки в отчете. При ссылке следует писать слово “таблица” с указанием ее номера.

Таблицу с большим количеством строк допускается переносить на другой лист (страницу). При переносе части таблицы на другой лист (страницу) слово “Таблица” и номер ее указывают один раз справа над первой частью таблицы, над другими частями пишут слово “Продолжение” и указывают номер таблицы, например: “Продолжение таблицы 1”. При переносе таблицы на другой лист (страницу) заголовок помещают только над ее первой частью.

Таблицу с большим количеством граф допускается делить на части и помещать одну часть под другой в пределах одной страницы. Если строки и графы таблицы выходят за формат страницы, то в первом случае в каждой части таблицы повторяется головка, во втором случае — боковик.

Если повторяющийся в разных строках графы таблицы текст состоит из одного слова, то его после первого написания допускается заменять кавычками; если из двух и более слов, то при первом повторении его заменяют словами “То же”, а далее — кавычками. Ставить кавычки вместо повторяющихся цифр, марок, знаков, математических и химических символов не допускается. Если цифровые или иные данные в какой-либо строке таблицы не приводят, то в ней ставят прочерк.

Таблицы следует нумеровать арабскими цифрами сквозной нумерацией. Допускается нумеровать таблицы в пределах раздела. В этом случае номер таблицы состоит из номера раздела и порядкового номера таблицы, разделенных точкой.

Заголовки граф и строк таблицы следует писать с прописной буквы в единственном числе, а подзаголовки граф – со строчной буквы, если они составляют одно предложение с заголовком, или с прописной буквы, если они имеют самостоятельное значение. В конце заголовков и подзаголовков таблиц точки не ставят. Таблицы слева, справа и снизу, как правило, ограничивают линиями. Допускается применять размер шрифта в таблице меньший, чем в тексте. Горизонтальные и вертикальные линии, разграничивающие строки таблицы, допускается не проводить, если их отсутствие не затрудняет пользование таблицей. Заголовки граф, как правило, записывают параллельно строкам таблицы. При необходимости допускается перпендикулярное расположение заголовков граф. Головка таблицы должна быть отделена линией от остальной части таблицы.

Формулы и уравнения. Уравнения и формулы следует выделять из текста в отдельную строку. Выше и ниже каждой формулы или уравнения должно быть оставлено не менее одной свободной строки. Если уравнение не умещается в одну строку, то оно должно быть перенесено после знака равенства (=) или после знаков плюс (+), минус (-), умножения (х), деления (:), или других математических знаков, причем знак в начале следующей строки повторяют. При переносе формулы на знаке, символизирующем операцию умножения, применяют знак “Х”.

Пояснение значений символов и числовых коэффициентов следует приводить непосредственно под формулой в той же последовательности, в которой они даны в формуле.

Формулы в отчете следует нумеровать порядковой нумерацией в пределах всего отчета арабскими цифрами в круглых скобках в крайнем правом положении на строке. Одну формулу обозначают – (1). Ссылки в тексте на порядковые номера формул дают в скобках. Пример – ... в формуле (1). Допускается нумерация формул в пределах раздела. В этом случае номер формулы состоит из номера раздела и порядкового номера формулы, разделенных точкой, например (3.1).

Ссылки. При ссылках на стандарты и технические условия указывают только их обозначение, при этом допускается не указывать год их утверждения при условии полного описания стандарта в списке использованных источников в соответствии с ГОСТ 7.1. Ссылки на использованные источники следует приводить в квадратных скобках.

2 Методические указания

Лабораторная работа №1

Тема: Анализ предметной области.

Цель: изучение и системное представление бизнес-процессов, подлежащих программированию, приобретение навыков системного анализа объектов и процессов реального мира на предмет организации программного управления ими.

Теоретические положения

1.1. Принцип системного анализа

При системном анализе определяется целевая функция – результат работы изучаемой системы (например, целевая функция работы фабрики – производство продукции, парикмахерская – выполнение стрижек и причесок, и т. д.).

Сущность системного анализа заключается в том, что система разделяется на ряд подсистем (частей), а каждая подсистема в свою очередь делится на задачи.

Понятие «подсистема» подразумевает, что выделяется относительно независимая часть системы, обладающая свойствами системы и, в частности, имеющая подцель, на достижение которой ориентирована подсистема, а также другие свойства – целостности, коммуникативности и т.п., определяемые закономерностями систем.

Система (процесс) может быть разделена на элементы (задачи) не сразу, а последовательным расчленением на подсистемы – совокупности элементов. Такое расчленение, как правило, производится на основе определения независимой функции, выполняемой данной совокупностью элементов совместно для достижения некой частной цели, которая обеспечивает достижение общей цели системы, и называется декомпозицией. Подсистема отличается от простой группы элементов, для которой не выполняется условие целостности.

Последовательное разбиение системы в глубину приводит к получению иерархии подсистем, нижним уровнем которых является элемент. С этой концепцией связано понятие структуры системы.

1.2. Системный подход к описанию объекта и процесса информатизации

Информатизация – это направленный процесс системной интеграции компьютерных средств, информационных и коммуникационных технологий с целью выработки релевантной информации в области ее применения и получения новых общесистемных свойств этой области, позволяющих более эффективно организовать процесс информатизации

Процесс информатизации – продуктивная деятельность человека или группы, целенаправленно осуществляемая в рамках объекта информатизации и имеющая целью достижение конкретного результата, представляющего ценность для потребителя.

Объект информатизации – область деятельности человека или группы, характеризуемая определенной целью и обладающая свойствами системы, элементами которой являются в том числе люди как субъекты деятельности.

Пример:
Объект информатизации: торговое предприятие.
Процесс информатизации: складской учет товаров.
Соответствующая тема: Спроектировать программную систему складского учета товаров.

Объект информатизации описывается лаконично простым повествовательным техническим текстом и содержит такие сведения:

1. Наименование объекта. Если объектом является организация, указывается ее юридическое название. Если объектом является техническое устройство, указывается его название согласно официальной документации на него.
2. Краткая характеристика объекта в 3–5 простых предложениях, раскрывающая его суть. Здесь уместно дать ссылку на источник этой информации.
3. Структурная схема, иллюстрирующая архитектуру объекта. Если объектом является организация, приводится её организационная диаграмма. Если объектом является техническое устройство, приводится его функциональная структура или алгоритм работы. Уместны и другие иллюстрации, если они значимо дополняют текстовую часть описания.
4. Текстовые комментарии компонентов структурной схемы, в том числе задачи (функции) компонентов и особенности их взаимодействия при функционировании объекта.
5. Основные показатели эффективности функционировании объекта (не более трех), на улучшение которых должна быть направлена информатизация.

Пример исчерпывающего описания объекта представлен в приложении Б.

Процесс информатизации, как правило, выбирается среди функционала объекта информатизации. Укрупнено его можно представить в нотации IDEF0 в представить в виде модели «Черный ящик» (рис. 1).

Если объектом является организация, то процесс информатизации представляет собой *бизнес-процесс*. В любом случае процесс информатизации объединяет в себе несколько задач (функций) объекта. Такую структуру можно представить в виде диаграммы декомпозиции.

Разделение бизнес-процесса на подпроцессы (задачи) производится с учетом целевой функции бизнес-процесса, а деление операции – с учетом целевой функции подпроцесса, исходя из целевой функции бизнес-процесса. Такой принцип упрощает изучение сложных бизнес-процессов и является системным подходом. На рис. 2 представлен принцип декомпозиции, использующийся в системном анализе.

В каждом конкретном случае процессу информатизации дается лаконичное и информативное наименование, в полной мере отражающее суть выполняемых действий на объекте. Именование задач и операций также выполняются в соответствии с этим принципом.

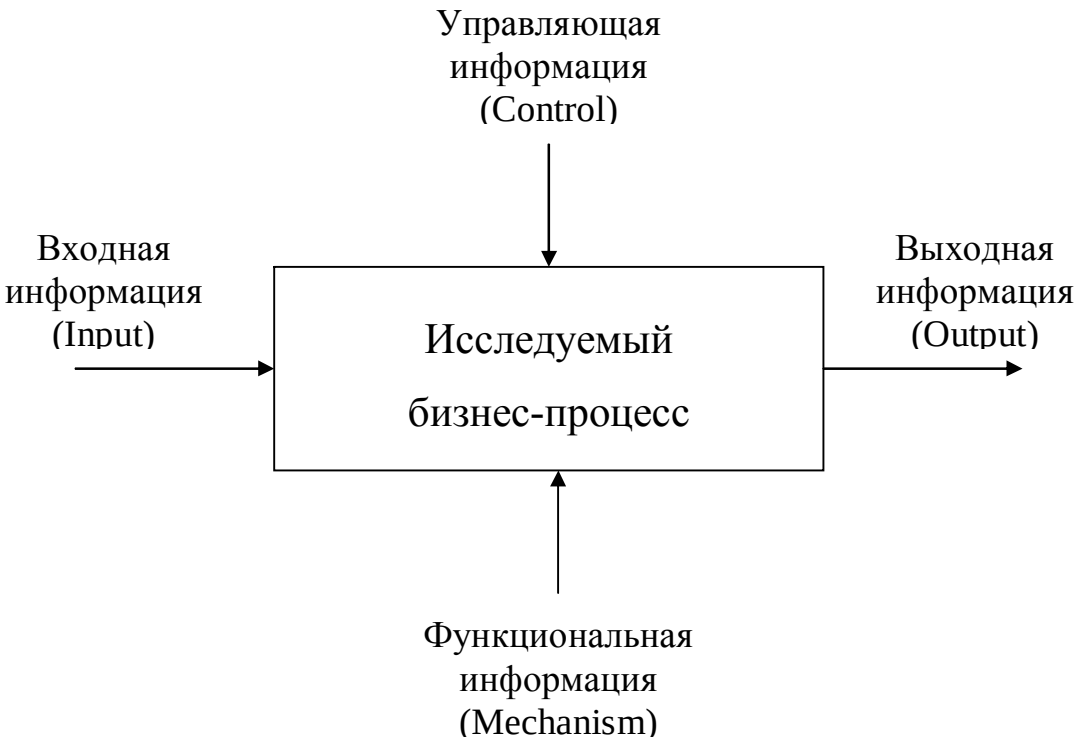


Рис. 1. Обобщенная схема модели «Черный ящик»

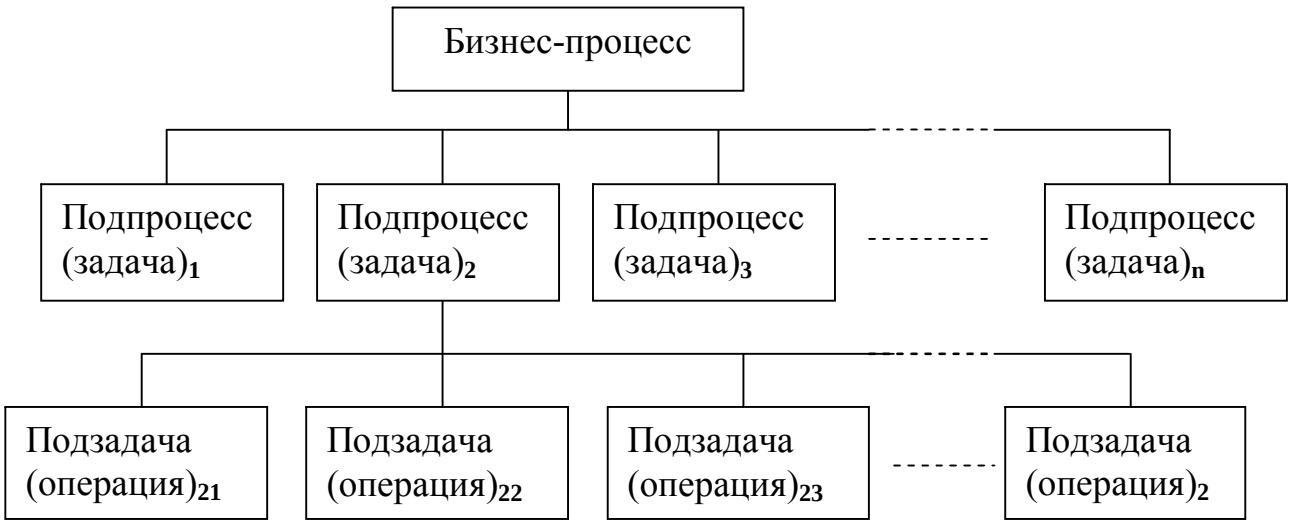


Рис. 2. Принцип декомпозиции бизнес-процесса

При выделении задач следует помнить, что задача программы – это формализованная совокупность действий, выполнение которых приводит к результату заданного вида. Поэтому в качестве задач надо выбирать такие, для которых можно четко сформулировать результат. В данном примере можно выделить такие задачи:

- задача ввода входных данных;
- задача сохранения данных в памяти ЭВМ;
- задача формирования выходных данных;
- задача вычисления некоторого итогового показателя;
- задача статистического анализа данных, и др.

Описание информационных потоков. При выделении задач следует учитывать, что это информационно взаимосвязанный функциональный комплекс, назначение которого – переработка входной информацию в выходную. Любой функциональный элемент, будь то процесс, задача или операция, получает на вход некоторую входную информацию (исходные данные, задание) и генерирует некоторую выходную результатную информацию. Возможно также выделение внутренней информации, циркулирующая внутри процесса. Как правило, это промежуточные данные, являющиеся результатами переработки входных инфопотоков и исходными данными для выработки выходных инфопотоков.

Вся информация циркулирует в рамках процесса информатизации в форме информационных потоков (инфопотоков) и требует описания и систематизации. Такое описание уместно представить форме реестров (табл. 1–3).

Возможные формы представления информации: сигнал, устное сообщение, устное распоряжение, документ, и др. Наименование потока должно адекватно отражать его сущность и соотноситься с его формой представления.

Трудозатраты представляют собой вещественную неотрицательную оценку, выражающую количество часов, которые должен работать 1 человек, чтобы воспринять и обработать входной поток или сформировать и вывести выходной поток. В редких случаях, когда человек не причастен к обработке инфопотока, выставляется оценка 0.

Таблица 1. Реестр входных информационных потоков

№	Наименование и назначение потока	Форма представле-ния	Обработчик (Кто обрабатывает)	Корреспон-дент (Откуда поступает)	Характеристики обработки	
					Трудозатраты, чел·ч	Периодичность, регламент

Таблица 2. Реестр внутренний информационных потоков

№	Наименование и назначение потока	Форма представле-ния	Обработчик (Кто обрабатывает)	Корреспон-дент (Кому передает)	Характеристики обработки	
					Трудозатраты, чел·ч	Периодичность, регламент

Таблица 3. Реестр выходных информационных потоков

№	Наименование и назначение потока	Форма представле-ния	Обработчик (Кто обрабатывает)	Корреспон-дент (Куда отправляется)	Характеристики обработки	
					Трудозатраты, чел·ч	Периодичность, регламент

Периодичность отражает частоту следования инфопотока. Как правило, даются средние оценки. Наилучший способ определения этой характеристики – получить экспертное мнение специалиста из предметной области. Если такого специалиста нет, то периодичность следует оценить исходя из назначения инфопотока и логики его обработки. К примеру, для оперативной информации периодичность может составлять от 10 раз в сутки (например, заявка на ремонт

оборудования) до 100 раз в секунду (например, сигнал от датчика системы реального времени). Для справочной информации периодичность гораздо ниже и может составлять 1 раз в год (обновление штатного расписания) или 10 раз в месяц (сведения о трудоустраиваемом лице). В крайнем случае, если никак не возможно установить приблизительную частоту следования инфопотока, допускается указать «По требованию».

Если формой представления инфопотока является документ, целесообразно представить его макет, либо копию. При этом необходимо кратко описать такие реквизиты документа, смысл которых не является очевидным.

1.3. Модель вариантов использования

Модель прецедентов (модель требований, модель *вариантов использования*) – это базовая диаграмма UML, на которой изображаются отношения между действующими лицами и действиями (прецедентами). Это исходная концептуальная модель системы в процессе ее проектирования и разработки. Создание диаграммы вариантов использования имеет следующие цели:

- определение общих границ и контекста моделируемой предметной области на начальных этапах проектирования ПС;
- формальное представление требований к функциональному поведению (функциональных требований) проектируемой ПС;
- разработка исходной концептуальной модели системы для ее последующей детализации в форме логических и физических моделей в нотации UML;
- подготовка исходной документации для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Система или процесс представляется в форме вариантов использования (прецедентов), с которыми взаимодействуют внешние сущности или актеры (в английской транскрипции – экторы). При этом актером называется любой объект, субъект, система или устройство, взаимодействующие с моделируемой системой извне. Концептуальный характер этой диаграммы проявляется в том, что подробная детализация диаграммы на начальном этапе проектирования имеет отрицательный характер, поскольку предопределяет способы реализации поведения системы – эти аспекты должны быть сознательно скрыты от разработчика на модели прецедентов.

Вариант использования (use case) – внешняя спецификация последовательности действий, которые система или другая сущность могут выполнять в процессе взаимодействия с актерами. Служит для описания сервисов, которые система предоставляет актеру в соответствии с его потребностями без рассмотрения ее внутренней структуры.

Отдельный вариант использования обозначается на диаграмме эллипсом, внутри которого содержится его краткое имя в форме глагольного существительного (рис. 3, а) или глагола (рис. 3, б) с пояснительными словами. Сам текст имени прецедента должен начинаться с заглавной буквы.

а) глагольное существительное б) глагол в) актер

Рис. 3. Графическое обозначение варианта использования и актера

Диаграмма вариантов использования содержит конечное множество вариантов использования, которые в целом должны определять все возможные аспекты ожидаемого поведения системы. Данной модели на всех этапах работы над проектом позволяет не только достичь требуемого уровня унификации обозначений для представления функциональности программы, но и является мощным средством последовательного уточнения требований на основе их итеративного обсуждения со всеми заинтересованными специалистами.

Примерами вариантов использования могут быть следующие действия:

- занесение систему данных о клиенте;
- проверка состояния текущего счета клиента;
- оформление заказа на покупку товара;
- получение информации о кредитоспособности клиента;
- отображение графической формы на экране монитора;
- авторизация пользователя;
- и другие действия, процессы, задачи.

Актер (actor) рассматривается как определенная роль относительно конкретного варианта использования. Стандартным графическим обозначением актера на диаграммах является фигурка человечка, под которой записывается имя актера (рис. 3, в). Имя актера должно быть достаточно информативным с точки зрения семантики. Это может быть:

- наименование должности (продавец, кассир, менеджер, и т.д.)
- технические устройства (датчики, терминалы обслуживания и т.д.);
- другие системы.

Не рекомендуется давать актерам имена собственные или названия моделей конкретных устройств, даже если это с очевидностью следует из контекста проекта, так как одно и то же лицо (устройство) может выступать в нескольких ролях и, соответственно, обращаться к различным сервисам проектируемой системы.

Рассматривая данную диаграмму в качестве модели бизнес-системы, можно ассоциировать ее с "черным ящиком" (рис. 1). При этом справедливы следующие **Правила соответствия моделей вариантов использования и «Черный ящик»:**

1. Для обработки каждого входящего инфопотока на модели «Черный ящик» должен быть хотя бы 1 прецедент на модели вариантов использования.
2. Для формирования каждого исходящего инфопотока на модели «Черный ящик» должен быть хотя бы 1 прецедент на модели вариантов использования.
3. Каждому прецеденту на модели вариантов использования должен быть

поставлен в соответствие хотя бы 1 инфопоток на модели «Черный ящик».

4. Число и названия актеров на модели вариантов использования должны совпадать с составом инфопотоков «Mechanism» на модели «Черный ящик».

! В случае нарушения хотя бы одного из указанных правил следует внести изменения в модели так, чтобы все 4 правила были соблюдены.

Задание

1. Дать характеристику объекта информатизации: наименование, назначение, структура, задачи, действующие лица.
2. Выполнить системное описание заданного бизнес-процесса и выполнить его декомпозицию на подпроцессы (задачи).
3. Построить модель «Черный ящик» и диаграмму вариантов использования UML. Описать построенные модели. Сформировать реестры инфопотоков.
4. Дать характеристику схеме решения задач в ручном режиме и выделить ее недостатки.
5. Обосновать необходимость усовершенствования и развития существующей схемы решения задач за счет создания специального программного обеспечения.

Требования к содержанию отчета

В подразделе 1 дается характеристика заданного объекта информатизации. Приводится его текстовое и графическое описание согласно технологии, рассмотренной в разделе 1.2.

В подразделе 2 выделяется процесс информатизации и ему дается наименование. При этом выполняются следующие обязательные элементы:

- приводится подробное повествовательное описание процесса;
- определяется состав действующих лиц, задействованных в процессе;
- определяется входная и выходная информация, строится структурная схема типа "черный ящик" (рис. 1);
- производится декомпозиция бизнес-процесса на подпроцессы (задачи), строится диаграмма декомпозиции (рис. 2);
- дается общая информация о задачах;
- строится базовая модель вариантов использования (без детализации);
- выполняется описание входных и выходных информационных потоков в виде табл. 1 и 2;
- приводятся правила обработки информации и возможные ограничения;
- определяется нормативно-справочная документация, регламентирующая бизнес-процесс.

На примере объекта информатизации ПАО Банк «ФК Открытие», описание которого представлено в приложении Б, рассмотрим одну из его задач «Проведение операций с иностранной валютой». Тогда процесс информатизации может быть назван так же как и задача, или проще: «Обмен валюты». На рис. 4 и 5 представлены модели «Черный ящик» и вариантов использования.

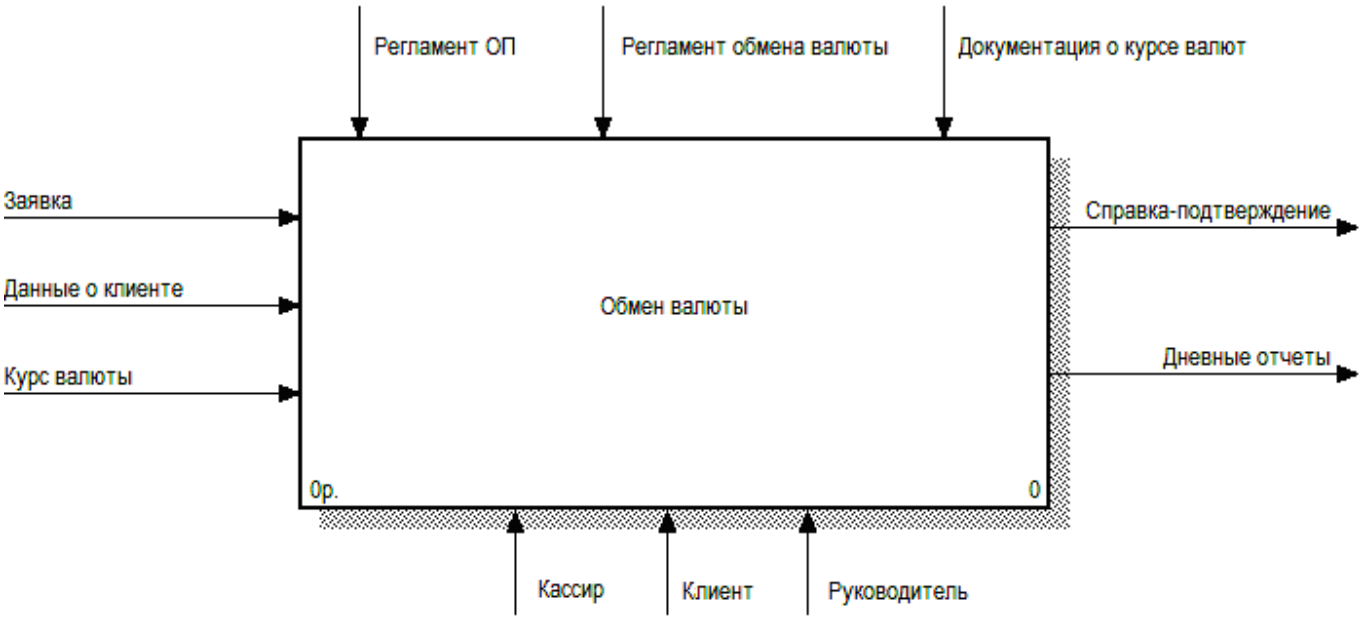


Рис. 4. Пример модели «Черный ящик» для бизнес-процесса обмена валюты

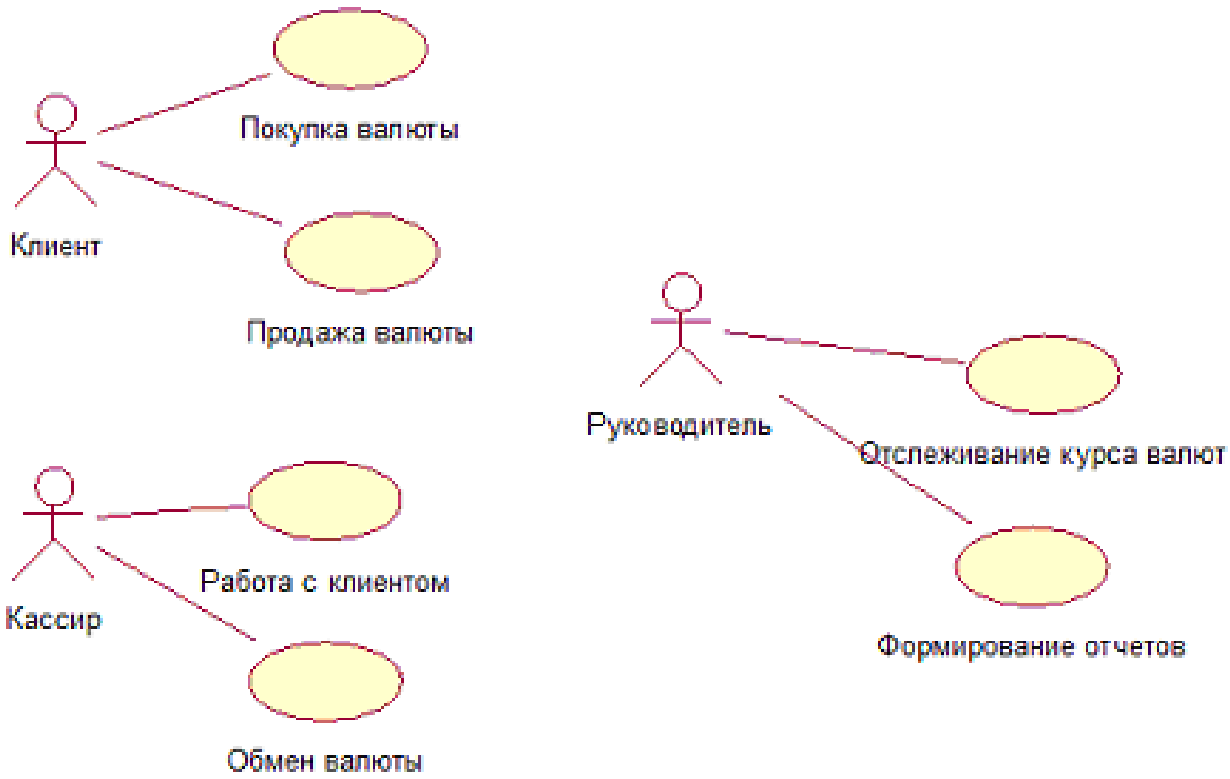


Рис. 5. Пример модели вариантов использования UML для бизнес-процесса обмена валюты

В целом описание бизнес-процесса должно давать представление о вычислительных задачах, предполагаемых заданным бизнес-процессом, величине информационного массива, подлежащего обработке, численности персонала, занятого в процессе создания специального ПО для решения выделенных задач.

Для более строгого структурирования данного материала рекомендуется

использовать табличную форму подачи информации.

В подразделе 3 в повествовательной форме простыми предложениями описываются основные операции, которые выполняются при сборе и обработке информации, а также связанные с этим действия персонала предприятия без использования специального программного обеспечения. Указываются те недостатки этой системы, которые приводят к снижению эффективности решения данных задач и бизнес-процесса в целом.

По результатам анализа недостатков дается обоснование необходимости создания специального ПО для автоматизации бизнес-процесса. Излагаются причины, вследствие которых создание программ для решения задач бизнес-процесса является необходимым.

В целом из материала отчета должно быть видно, что представляет собой заданный бизнес-процесс, какие задачи при этом решаются, какие из них выполняются не достаточно эффективно, почему и на каком уровне необходима программная реализация указанных задач.

Контрольные вопросы и упражнения

1. Что такое декомпозиция бизнес-процесса?
2. Какова типовая структура декомпозиции бизнес-процесса?
3. Что такое схема типа "черный ящик"?
4. Что такое документооборот бизнес-процесса?
5. Что такое нормативно-справочная документация, регламентирующая бизнес-процесс?
6. В чем заключаются цели и назначение выбранного бизнес-процесса и какова его динамика?
7. Что такое программное обеспечение (ПО)?
8. Что такое задача?
9. Чем определяется эффективность решения задач ПО?
10. В чем различия между задачей бизнес-процесса и задачей программы?
11. Каковы могут быть основания для создания специального ПО для автоматизации бизнес-процесса?
12. Какие задачи выбранного бизнес-процесса решаются не достаточно эффективно и почему?
13. Какие критерии эффективности могут быть использованы для оценивания эффективности реализации бизнес-процесса?
14. Перечислите задачи в структуре заданного бизнес-процесса. Дайте краткую характеристику одной из задач, включая задействованные документы.
15. Насколько целесообразным является решение об автоматизации выделенных задач бизнес-процесса?
16. Приведите пример правил обработки информации при описании бизнес-процесса. Выполните описание основных операций, которые выполняются при сборе и обработке информации для конкретной задачи бизнес-процесса.
17. Укажите несколько недостатков, которые приводят к снижению эффективности решения задач бизнес-процесса.

Лабораторная работа №2

Тема: Анализ существующих компьютерных разработок.

Цель: ознакомление с существующими разработками подобных программных решений по выбранной теме, приобретение навыков анализа существующих компьютерных разработок.

Задание

1. Выполнить системное описание существующих подобных программных систем (не менее двух), которые могут быть применены к данному объекту управления; выделить основные преимущества и недостатки представленных систем.
2. Выполнить сравнительную характеристику описанных систем. Результаты сравнительного анализа представить в табличной форме. Набор основных показателей для сравнения (подробнее см. приложение В):
   - назначение системы;
   - эффективность системы;
   - гибкость системы;
   - защищенность системы;
   - живучесть системы;
   - надежность системы;
   - открытость системы;
   - оптимальность использования ресурсов;
   - удобство пользовательского интерфейса системы;
   - стоимость системы (в том числе затраты на тех. поддержку);
   - эргономичность.
3. Сделать вывод о возможности или невозможности использования этих систем на выбранном объекте информатизации.

Требования к содержанию отчета

В подразделе 1 одно за другим приводятся описания ранее разработанных компьютеризированных систем (подсистем), автоматизирующих объекты и процессы, подобные имеющем место в выбранной предметной области. Описание уместно сопровождать следующими иллюстрациями:

- функционально-структурная схема системы;
- обобщенная блок-схема алгоритма функционирования системы;
- экранные формы основных частей пользовательского интерфейса;
- таблицы и графики, отражающие статистические показатели функционирования системы.

Для каждой системы указываются преимущества и недостатки: вообще, а не применительно к выбранному объекту

В подразделе 2 выполняется сравнение представленных систем по нескольким показателям. Сравнение подкреплять количественными показателями (например, сроки внедрения, объем дискового пространства, кол-во единиц техники, стоимость

программного обеспечения и т.п.) и статистическими оценками в виде таблиц, диаграмм, графиков.

В подразделе 3 делается заключение о возможности применения рассмотренных систем к выбранному объекту и процессу информатизации. Оценивается степень этой возможности на предмет удобства и скорости настройки системы на данную предметную область, а также оптимальности их внедрения с учетом затрат на дальнейшее обслуживание и тех. поддержку.

В целом из материала отчета должно быть видно, какие из систем, автоматизирующих подобные выбранным объекты и процессы, существуют на отечественном и зарубежном рынке, и вообще в мире. Какой комплекс задач они позволяют решить, насколько оперативны и эффективны получаемые решения и насколько они соответствуют целям, провозглашенным в лаб. работе №1.

Контрольные вопросы

(ответы должны подкрепляться количественными оценками)

1. Насколько широк спектр существующих автоматизированных систем, которые уместно применить к выбранному объекту информатизации?
2. Какая из рассмотренных систем наиболее близка для применения на выбранном объекте информатизации, и почему?
3. Насколько глубокие изменения необходимо произвести в той или иной из описанных систем для ее использования в другой предметной области?
4. Как быстро произойдет окупаемость той или иной из описанных систем при их внедрении на выбранном объекте?
5. Почему какую-либо из рассмотренных систем целесообразно применить на выбранном объекте, а почему нецелесообразно?

Лабораторная работа №3

Тема: Техническое задание на создание программного продукта.

Цель: Освоение методики предварительного анализа разрабатываемой программы; освоение задач формулирования функциональных и нефункциональных требований к программной реализации отдельных задач и к программе в целом; выработка навыков разработки технического задания.

Теоретические положения

3.1. Требования к ПО

Требование – исходное понимание задачи разработчиками, которому должна соответствовать система или программный продукт и которое является основой всей разработки.

В соответствии со стандартом разработки требований ISO/IEC 29148, требования представляют собой четкие, проверяемые и измеримые утверждения, которые идентифицируют рабочие характеристики, функциональные параметры и конструктивные ограничения продукта или процесса.

Схема многообразия требований к ПО представлена на рис. 6.

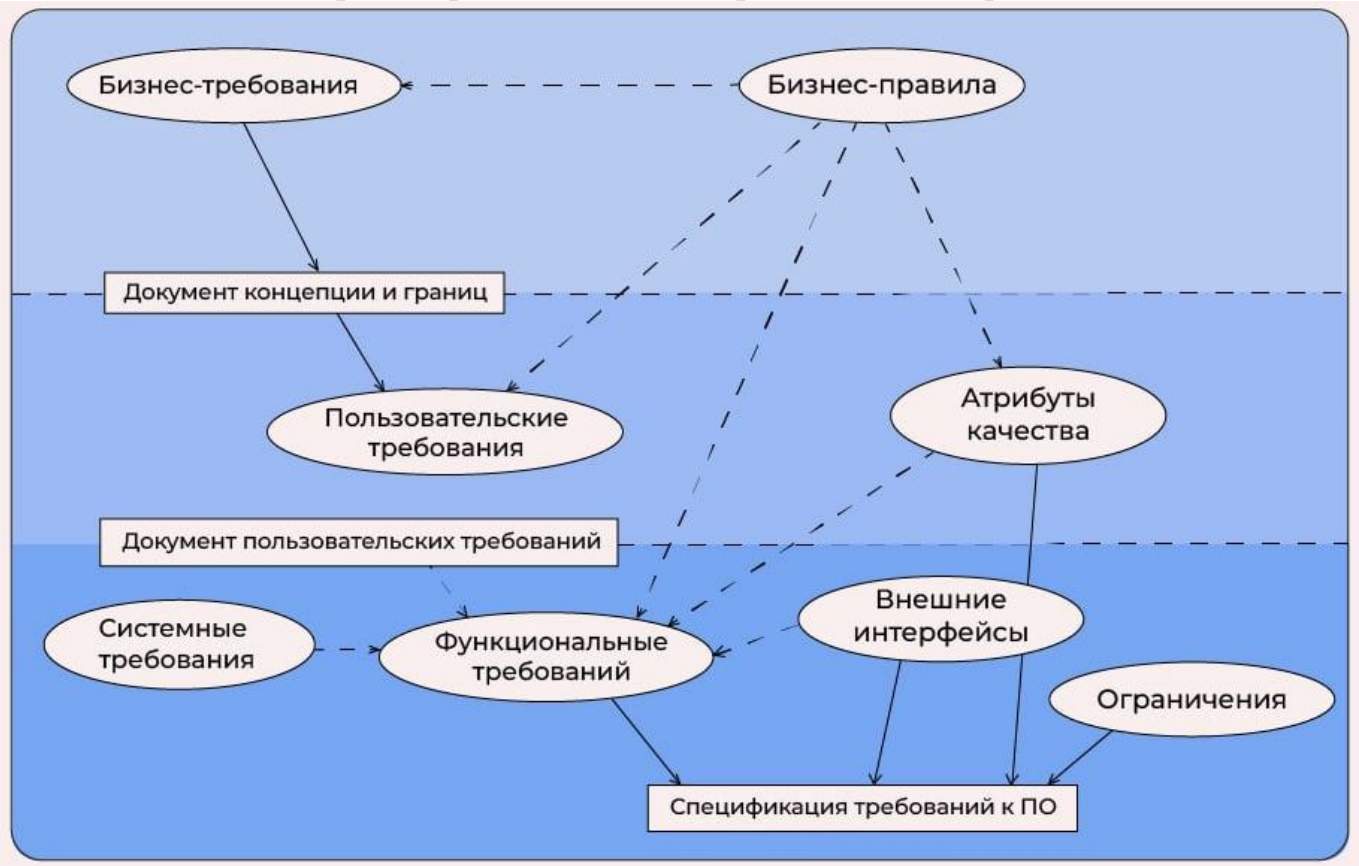


Рис.6. Многоуровневая структура требований к ПО

Бизнес-требования определяют назначение ПО, описываются в документе о видении (vision) и границах проекта (scope). Бизнес-требования описывают, почему

организации нужна такая система, то есть цели, которые организация намерена достичь с ее помощью. Основное их содержание – бизнес-цели организации или клиента, заказывающих систему. Это верхний уровень абстракции требований к системе. Они не относятся напрямую к реализации проекта, а в первую очередь отражают цели бизнеса, абстрагированные от реализации системы. В конечном итоге бизнес-требования формируют документ концепции и границ.

Примеры бизнес-требований:

1. Создать промо-сайт, привлекающий внимание определенной аудитории к определенной продукции компании (идея).

2. Сократить время обработки заказа на 50% (цель) — система должна предоставить интерфейс, упрощающий создание заказа (концепция).

3. Увеличить клиентскую конверсию до 35% (цель) — в системе должны быть представлены механизмы побуждения клиента к заказу (концепция).

Пользовательские требования (user requirements) определяют набор пользовательских задач, которые должна решать программа, а также способы (сценарии) их решения в системе. Пользовательские требования могут выражаться в виде фраз утверждений, в виде способов применения (use case), пользовательских историй (user story), сценариев взаимодействия (scenario). Область пользовательских требований также включает описания атрибутов или характеристик продукта, которые важны для удовлетворения пользователей. Пользовательские требования также часто именуются фичами.

Фича (функциональность) – функционально обобщенная часть системы, решающая отдельную задачу пользователей или интерпретирующая бизнес-процессы (и их артефакты), которые будут реализованы в рамках системы.

Исходя из вышеописанных определений, пользовательские требования содержат:

- цели и задачи пользователей;
- сценарии использования – способ решения задачи пользователей;
- как следствие, описание самих пользователей системы: пользовательские роли и уровни доступа.

В конечном итоге пользовательские требования формируют «Документ пользовательских требований». Пользовательские требования могут быть представлены в виде:

- текстового описания;
- вариантов использования, сценариев использования (Use Case);
- пользовательских историй (User Story);
- диаграммы вариантов использования.

Как правило, пользовательские требования описываются по следующему шаблону: *Пользователь должен иметь возможность + {тезис}*.

Примеры пользовательских требований:

1. Пользователь должен иметь возможность добавить объект в избранное (функциональность).

2. Система должна представлять пользователю диалоговые окна для ввода исчерпывающей информации о заказе и последующей фиксации этой информации в

БД.

Источники пользовательских требований требований (где искать?):

- анализ и декомпозиция бизнес-требований;
- описание бизнес-процессов;
- артефакты бизнес-процессов: документы входные/выходные, стандарты, регламенты, обрабатываемые данные, иная информация, используемая и/или производимая в бизнес-процессе;
- отраслевые стандарты.
- реализованные проекты, проекты конкурентов.

Функциональные требования представляют собой детальное описание поведения и сервисов программной системы, ее функционала. Они определяют, какие функции система должна выполнять.

Функциональные требования определяют, каким должно быть поведение продукта в тех или иных условиях. Это самые низкоуровневые требования и являются результатом декомпозиции верхнеуровневых требований и описывают атомарные функции, которые должны быть реализованы в системе. Они определяют, что разработчики должны создать, чтобы пользователи смогли выполнить свои задачи (пользовательские требования) в рамках бизнес-требований.

Функциональные требования описывают следующие характеристики:

- функция или объект функционирования;
- входные данные и источники;
- выходные данные и пункт их назначения;
- средства для выполнения функции;
- описание предварительных и заключительных условий (предусловий), если функция специфицирована.

Примеры функциональных требований:

1. Система должна по электронной почте отправлять пользователю подтверждение о заказе или заказ может быть создан, отредактирован, удален и перемещен с участка на участок.
2. Операция продажи должна сопровождаться формированием и выводом на печать фискального чека и расходной накладной. Форматы выходных документов представлены в приложении 3.

Источники требований функциональных (где искать?):

- анализ пользовательских требований.
- расширенные описания пользовательских задач (таски);
- прототипы интерфейса (мокапы);
- отраслевые и корпоративные стандарты на производство продукции;
- внешние системы и документация к ним (интеграции).

Функциональные требования являются основными, документируются в спецификации требований к ПО (software requirements specification, SRS). Так же, как и для пользовательских требований, для них строятся диаграммы вариантов использования. Требуемое поведение системы специфицируется одним или несколькими вариантами использования, которые определятся в соответствии с потребностями акторов.

Нефункциональные требования не являются описанием функций системы. Этот вид требований описывает качественные характеристики программной системы. Наиболее очевидные из них следующие:

1. Сопровождаемость (maintainability) – означает, что программа должна быть написана с расчетом на дальнейшее развитие. Это критическое свойство системы, так как изменения ПО неизбежны вследствие изменения бизнеса. Сопровождение программы выполняют, как правило, не те люди, которые ее разрабатывали. Включает такие элементы:
   - наличие и понятность проектной документации;
   - соответствие проектной документации исходному коду;
   - понятность исходного кода;
   - простота изменений исходного кода;
   - простота добавления новых функций.
2. Надежность (dependability). Надежность ПО – комплексное свойство, включает такие элементы как:
   - отказоустойчивость – возможность восстановления программы и данных в случае сбоев в работе;
   - безопасность – сбои в работе программы не должны приводить к опасным последствиям (авариям);
   - защищенность от случайных или преднамеренных внешних воздействий (защита от «дурака», вирусов, спама).
3. Эффективность (efficiency). ПО не должно впустую тратить системные ресурсы, такие как память, процессорное время, каналы связи. Поэтому эффективность ПО оценивается следующими показателями:
   - время выполнения кода;
   - загруженность процессора;
   - объем требуемой памяти;
   - время отклика и т.п.
4. Удобство использования (usability). ПО должно быть легким в использовании, причем именно тем типом пользователей, на которых рассчитано приложение. Это включает в себя интерфейс пользователя и адекватную документацию. Причем, пользовательский интерфейс должен быть не интуитивно, а **профессионально понятным** пользователю.
5. Особенности поставки – наличие инсталлятора, документации.
6. Определенный уровень качества. Например, для новой Java-машины это будет означать, что она удовлетворяет набору тестов, поддерживаемому компанией Sun.
7. Требования к средствам и процессу разработки системы.
8. Требования к переносимости.
9. Требования соответствию стандартам и т.д.

Требования этого вида часто относятся ко всей системе в целом. На практике, особенно начинающие специалисты, часто забывают о некоторых важных нефункциональных требованиях. Более полный перечень нефункциональных требований представлен в приложении Б.

Реализация нефункциональных требований часто требует бóльших затрат, чем функциональных. Так, сопровождаемость требует значительных усилий по поддержанию соответствия проекта исходному коду и применения специальных методов создания модифицируемых программ. Надежность – дополнительных средств восстановления системы при сбоях. Эффективность – поиска специальных архитектурных решений и оптимизации кода. А удобство – проектирования не «интуитивно» понятного, а профессионально понятного интерфейса пользователя. Наиболее полный перечень которых представлен в приложении В.

3.2. Свойства требований к ПО

Сформулируем ряд важных свойств требований.

– Ясность, недвусмысленность – однозначность понимания требований заказчиком и разработчиками. Часто этого трудно достичь, поскольку конечная формализация требований, выполненная с точки зрения потребностей дальнейшей разработки, трудна для восприятия заказчиком или специалистом предметной области, которые должны проинспектировать правильность формализации.

– Полнота и непротиворечивость.

– Необходимый уровень детализации. Требования должны обладать ясно осознаваемым уровнем детализации, стилем описания, способом формализации. Возможные варианты:

- описание свойств предметной области, для которой предназначается ПО;
- техническое задание, которое прилагается к контракту;
- проектная спецификация, которая должна быть уточнена в дальнейшем, при детальном проектировании;
- и т.д.

Важно также ясно видеть и понимать тех, для кого данное описание требований предназначено, иначе не избежать недопонимания и последующих за этим трудностей. Ведь в разработке ПО задействовано много различных специалистов – инженеров, программистов, тестировщиков, представителей заказчика, возможно, будущих пользователей – и все они имеют разное образование, профессиональные навыки и специализацию, часто говорят на разных языках. Здесь также важно, чтобы требования были максимально абстрактны и независимы от реализации.

– Прослеживаемость – важно видеть то или иное требование в различных моделях, документах, наконец, в коде системы. *Например, часто возникают вопросы типа – "Кто знает, почему мы решили, что такой-то модуль должен работать следующим образом ....?".* Прослеживаемость функциональных требований достигается путем их дробления на отдельные, элементарные требования, присвоение им идентификаторов и создания трассировочной модели, которая в идеале должна протягиваться до программного кода. *Хочется, например, знать, где нужно изменить код, если данное требование изменилось.*

На практике полная формальная прослеживаемость труднодостижима, поскольку логика и структура реализации системы могут сильно не совпадать с

таковыми для модели требований. В итоге одно требование оказывается сильно "размазано" по коду, а тот или иной участок кода может влиять на много требований. Но стремиться к прослеживаемости необходимо, разумно совмещая формальные и неформальные подходы.

– Тестируемость и проверяемость – необходимо, чтобы существовали способы оттестировать и проверить данное требование. *Причем, важны оба аспекта, поскольку часто проверить-то заказчик может, а вот тестировать данное требование очень трудно или невозможно в виду ограниченности доступа (например, по соображениям безопасности) к окружению системы для команды разработчика.* Необходимые процедуры проверки требований:

- выполнение тестов,
- проведение инспекций,
- проведение формальной верификации части требований
- определение порога качества (чем выше качество, тем оно дороже стоит!);
- определение критериев полноты проверок, чтобы выполняющие их и руководители проекта четко осознавали, что именно проверено, а что еще нет.
- и пр.

– Модифицируемость. Определяет процедуры внесения изменений в требования.

– Единичность – требование описывает одну и только одну фичу.

– Актуальность – требование не стало устаревшим с течением времени.

– Завершённость – требование полностью определено в одном месте и вся необходимая информация присутствует.

– Атомарность – требование не может быть разбито на ряд более детальных требований без потери завершённости (далее на рис. 8 требования детализированы посредством отношений включения и расширения).

3.3. Ошибки при документировании требований

К сожалению, в процессе составления требований так же могут возникать проблемы или же ошибки. Это связано со многими моментами самого процесса составления, так как он очень объемен и зависит напрямую от пожеланий заказчика, будь то человек или компания. Диапазон причин огромен, и в данном подразделе представлен их систематизированный анализ.

Перечислим ряд ошибок, встречающихся при составлении технических заданий и иных документов с требованиями. Первые три проблемы возникают, когда заказчик не указывает четких требований.

1. Описание возможных решений вместо требований. В данном случае заказчик предлагает множество возможных решений. При этом используются формулировки в настоящем времени, что является нарушением формального описания требований. В таком случае заказчику задается вопрос – «Зачем заказчику

нужен исполнитель?». Этот момент так же создает некую абстракцию для исполнителя ввиду множественной вариативности желаний со стороны заказчика.

Данная проблема разрешается путем разграничение ролей в процессе общения с заказчиком и уточнением требований самого заказчика к готовому продукту.

Неправильно	Правильно
В систему вводятся данные о поставщиках комплектующих. При этом производится контроль корректности данных (отсутствие дублирования записей, корректность банковских реквизитов и т.п.). Данные сохраняются в БД для долгосрочного хранения	Система должна обеспечивать корректный ввод данных о поставщиках комплектующих: наименование, ИНН, юридический и физический адрес, телефоны, банковские реквизиты. Требования к корректности ввода: – отсутствие дублирования записей; – корректность ИНН; – корректность банковских реквизитов. В результате ввода данные должны сохраняться в БД
При включении зажигания компьютер опрашивает состояние датчика присутствия водителя на сидении автомобиля. Если водитель зафиксирован, то фиксируется изображение лица водителя	Фиксация изображения лица водителя должна выполняться при поступлении сигналов с блока зажигания и датчика присутствия водителя на сидении автомобиля

2. Обобщенные требования, отсутствие конкретики по существу их реализации. В данном случае заказчик так и не смог определиться с тем, что он хочет получить в качестве продукта, будь то программное обеспечение или приложение. Таким образом, разработка делается невозможной или максимально абстрактной. Формулировки таких требований, как правило, очень краткие и включают общие фразы и обобщающие эпитеты.

Неправильно	Правильно
При покупке товара клиенту выдается соответствующий документ	Операция продажи должна сопровождаться формированием и выводом на печать фискального чека и расходной накладной. Форматы выходных документов представлены в приложении 3
Необходимо обеспечить ввод в систему исходных данных	Система должна предусматривать ввод и сохранение в БД входных данных: – данные о пациентах (ФИО, дата рождения, паспортные данные, адрес проживания, адрес регистрации, место работы или учебы, контактный телефон); – данные о врачах (ФИО, дата рождения, паспортные данные, специальность, квалификация, адрес, телефон, предыдущая трудовая деятельность); – данные о записи на приём (ФИО пациента, ФИО врача, дата, время). При записи на прием должны быть зафиксированы дата и время внесения записи, а также присвоен номер талона

3. Нечеткие требования оставляют недосказанности, имеют оттенок советов, обсуждений, рекомендаций: "Возможно, что имеет смысл реализовать также…", и т.д. Они не дают однозначного понимания и не допускают однозначную проверку.

Проблему отсутствия чётких требований можно разрешить путем наличия в группе разработчиков человека, имеющего контакт с заказчиком. Постоянный контакт с заказчиком позволит сформировать как сами требования, так и лучше понять моменты разработки, например, какую технологию стоит применить в процессе разработки и, на каком языке программирования это будет лучше сделать.

Неправильно	Правильно
Входные данные должны быть организованы в виде вводимого в базу данных текста или файла, соответствующего определенному шаблону	Входные данные о вакансиях должны быть представлены в формате CSV. Последовательность значений должна определяться порядком расположения полей таблицы вакансий в базе данных ИС
Изображение лица водителя должно быть зафиксировано с достаточным качеством для качественного его распознавания	Разрешение фиксируемого изображения должно быть не менее 300 dpi

4. Игнорирование аудитории, для которой предназначено представление требований. Данная проблема связана со сферой деятельности составителя требований. Например, если спецификацию составляет инженер заказчика, то часто встречается переизбыток информации об оборудовании, с которым должна работать программная система, отсутствует глоссарий терминов и определений основных понятий, используются многочисленные синонимы и т.д. Или допущен слишком большой уклон в сторону программирования, что делает данную спецификацию непонятной всем непрограммистам.

Данная проблема разрешается путем разграничение ролей в процессе общения с заказчиком и уточнением требований самого заказчика к готовому продукту.

Неправильно	Правильно
Мастер для подтверждения приема заказа на выполнение должен посмотреть объект для ремонта, определить какие материалы ему нужны и в каком количестве. После осмотра объекта, статус становится «в работе», при этом клиенту приходит сообщение об этом	Подтверждение приема заказа должно осуществляться мастером на основе результатов обследования объекта. При этом в систему дополнительно должны быть внесены данные о необходимых материалах. После подтверждения приема заказа нажатием на кнопку «Подтвердить» статус заказа должен быть установлен в значение «В работе» и должно быть сформировано сообщение клиенту (требования к формированию сообщения клиенту см. в п. 3.1).

5. Пропуск важных аспектов, связанных с нефункциональными требованиями, в частности:

- требуемых состав БД и специального ПО;
- требуемое взаимодействие подсистем ПС, в том числе посредством телекоммуникаций;
- информации об окружении системы;

– требования к защите информации;

– сроки готовности других систем, с которыми должна взаимодействовать данная.

Это случается, например, когда данная программная система является частью более крупного проекта. Типичны проблемы при создании программно-аппаратных систем, когда аппаратура не успевает вовремя и ПО невозможно тестировать, а в сроках и требованиях это не предусмотрено.

Данная ошибка чревата полной или частичной неработоспособности программного обеспечения, в случае если не были рассмотрены моменты взаимодействия с ПО. Подобная проблема разрешима путем уточнения данных моментов у заказчика, с отсылкой на адаптивность ПО под сферу использования.

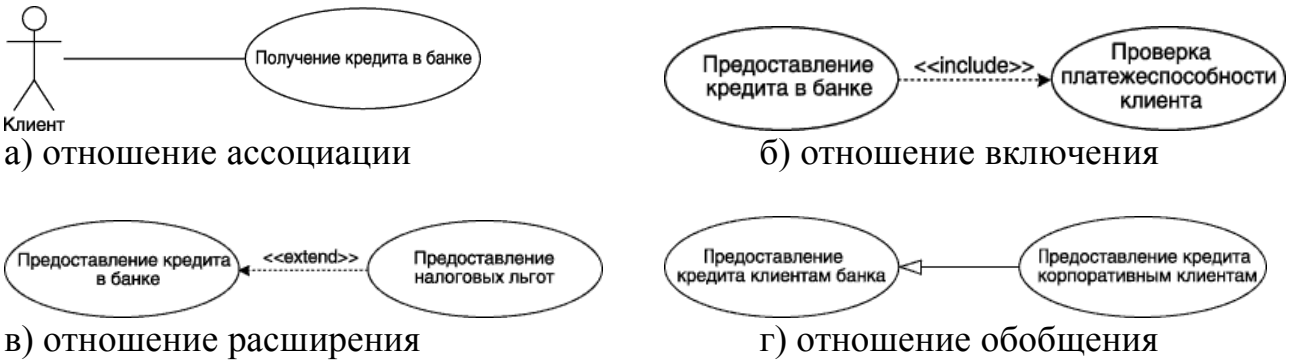
3.4. Объектно-ориентированный подход к моделированию требований на UML

Отношение (*relationship*) – семантическая связь между отдельными элементами модели. Между элементами модели прецедентов могут существовать различные отношения, которые описывают взаимодействие экземпляров одних актеров и вариантов использования с экземплярами других актеров и вариантов использования. Один актер может взаимодействовать с несколькими вариантами использования. Это значит, что этот актер обращается к нескольким сервисам данной системы. В свою очередь один прецедент может взаимодействовать с несколькими актерами, предоставляя свой сервис для всех них. В то же время два варианта использования, определенные в рамках одной моделируемой системы, могут взаимодействовать друг с другом, однако характер этого взаимодействия будет отличаться от взаимодействия с актерами.

В обоих случаях способы взаимодействия элементов модели предполагают обмен сигналами или сообщениями, которые инициируют реализацию функционального поведения моделируемой системы. В языке UML имеется несколько стандартных видов отношений между элементами модели прецедентов: ассоциация, включение, расширение и обобщение.

С помощью отношений включения, расширения и обобщения могут быть представлены общие свойства вариантов использования. Отношение же ассоциации – одно из фундаментальных понятий в языке UML и используется при построении всех графических моделей систем в форме канонических диаграмм. На диаграмме требований устанавливается только между актером и вариантом использования и служит для обозначения специфической роли актера при его взаимодействии с прецедентом. На диаграмме требований, так же как и на других диаграммах, ассоциация обозначается сплошной линией между актером и прецедентом, которая может иметь дополнительные обозначения, например, имя и кратность (рис. 7, а).

В контексте модели прецедентов ассоциация между актером и вариантом использования может указывать на то, что актер инициирует соответствующий прецедент. Такого актера называют главным. В других случаях подобная ассоциация может указывать на актера, которому предоставляется справочная информация о результатах функционирования моделируемой системы. Такого актера называют второстепенным.



а) отношение ассоциации б) отношение включения

в) отношение расширения г) отношение обобщения

Рис. 7. Пример графического представления отно шений на диаграмме требований

Включение (include) в языке UML – это разновидность отношения зависимости между базовым вариантом использования и его специальным случаем. При этом отношением зависимости (dependency) является такое отношение между двумя элементами модели, при котором изменение одного элемента (независимого) неизбежно приводит к изменению другого элемента (зависимого).

Отношение включения устанавливается только между двумя вариантами использования и указывает на то, что заданное поведение для одного варианта использования включается в качестве обязательного составного фрагмента в последовательность поведения другого варианта использования. Данное отношение является направленным бинарным отношением в том смысле, что пара экземпляров вариантов использования всегда упорядочена в этом отношении.

Так, например, отношение включения, направленное от варианта использования "Предоставление кредита в банке" к варианту использования "Проверка платежеспособности клиента", указывает на то, что каждый экземпляр первого (базового) прецедента всегда включает в себя функциональное поведение или выполнение второго (включаемого) прецедента, то есть поведение второго прецедента является частью поведения первого прецедента на данной диаграмме. Графически данное отношение обозначается как отношение зависимости в форме пунктирной линии со стрелкой, направленной от базового прецедента к включаемому, при этом данная линия помечается стереотипом <<include>>, как показано на рис. 7, б. Семантика этого отношения определяется следующим образом. Процесс выполнения базового варианта использования включает в себя как собственное подмножество последовательность действий, которая определена для включаемого варианта использования. Причем выполнение включаемой последовательности действий происходит всегда при инициировании базового прецедента.

Один вариант использования может входить в несколько других прецедентов, а также включать в себя другие прецеденты. Включаемый вариант использования является независимым от базового прецедента в том смысле, что он предоставляет последнему инкапсулированное поведение, детали реализации которого скрыты от последнего и могут быть легко перераспределены между несколькими включаемыми прецедентами. Более того, базовый вариант использования зависит только от результатов выполнения включаемого в него варианта использования, но не от структуры включаемых в него прецедентов.

Расширение (extend) определяет взаимосвязь базового прецедента с другим прецедентом, функциональное поведение которого задействуется базовым не всегда, а только при выполнении дополнительных условий.

В языке UML отношение расширения устанавливается только между вариантами использования, и является зависимостью, направленной к базовому прецеденту и соединенной с ним в так называемой точке расширения. Отношение расширения между вариантами использования обозначается как отношение зависимости в форме пунктирной линии со стрелкой, направленной от прецедента, который является расширением для базового прецедента, и помеченная стереотипом <<extend>>, как показано на рис. 7, в. В изображенном фрагменте имеет место отношение расширения между базовым прецедентом "Предоставление кредита в банке" и прецедентом "Предоставление налоговых льгот". Это означает, что свойства поведения первого прецедента в некоторых случаях могут быть дополнены функциональностью второго прецедента, для чего должно быть выполнено определенное логическое условие данного отношения расширения.

Отношение расширения позволяет моделировать поведение системы таким образом, что один из вариантов использования должен присоединять к своему поведению последовательность действий, определенную для зависимого (расширяющего) прецедента. При этом, в отличие от отношения включения, данное отношение всегда предполагает проверку условия и ссылку на точку расширения в базовом прецеденте. Точка расширения определяет место в базовом прецеденте, в которое должно быть помещено расширение при выполнении соответствующего логического условия. Один вариант использования может быть расширением для нескольких базовых прецедентов, а также иметь в качестве собственных расширений другие прецеденты. Любой базовый вариант использования не зависит от своих расширений.

Семантика отношения расширения определяется следующим образом. Если базовый прецедент выполняет некоторую последовательность действий, которая определяет его поведение, и при этом имеется точка расширения на экземпляр другого прецедента, которая является первой из всех точек расширения у базового прецедента, то проверяется логическое условие данного отношения. Если это условие выполняется, исходная последовательность действий расширяется посредством включения действий расширяющего варианта использования. Следует заметить, что условие отношения расширения проверяется лишь один раз – при первой ссылке на точку расширения, и если оно выполняется, то все зависимые прецеденты вставляются в базовый прецедент.

Обобщение представляет собой реализацию принципа наследования в ООП. Помимо вариантов использования, может также связывать два и более актера, которые могут иметь общие свойства, т.е. взаимодействовать с одним и тем же множеством вариантов использования одинаковым образом. Такая общность свойств и поведения представляется в виде отношения обобщения с другим, возможно, абстрактным прецедентом/актером, который моделирует соответствующую общность действий/ролей.

Графически отношение обобщения обозначается сплошной линией со стрелкой в форме треугольника без заливки, которая указывает на родительский

прецедент или действующее лицо (рис. 7, г). Эта линия со стрелкой имеет специальное название – стрелка-обобщение, – которая используется также и в других канонических диаграммах UML. В данном примере отношение обобщения указывает на то, что вариант использования "Предоставление кредита корпоративным клиентам" есть специальный случай варианта использования "Предоставление кредита клиентам банка". Другими словами, первый прецедент является специализацией второго прецедента. При этом вариант использования "Предоставление кредита клиентам банка" называют предком или родителем по отношению к варианту использования "Предоставление кредита корпоративным клиентам", а последний прецедент называют потомком по отношению к первому прецеденту. Следует отметить, что потомок наследует все свойства поведения своего родителя, а также может обладать дополнительными особенностями поведения.

Отношение обобщения между элементами модели прецедентов применяется в том случае, когда необходимо отметить, что дочерние элементы обладают всеми особенностями поведения родительских. Дочерние актеры/прецеденты участвуют во всех отношениях родительских актеров/прецедентов. По аналогии с принципом наследования ООП, дочерние актеры/прецеденты могут наделяться новыми свойствами поведения, которые отсутствуют у их предков, а также уточнять или модифицировать наследуемые от них свойства поведения.

Задание

1. Установить назначение и общую цель создания программы.
2. Определить структуру программы и состав функциональных задач.
3. Разработать функциональные требования к программе:
   - требования к входным и выходным данным;
   - требования к программной реализации задач;
   - специальные требования к математическому обеспечению программной реализации задач;
4. Разработать модель требований в нотации UML
5. Разработать требования к информационному обеспечению (к базе данных).
6. Разработать требования к инструментальному программному обеспечению (к системе управления базой данных (СУБД), к средству разработки программ (IDE), средствам автоматизированного проектирования ПО)
7. Установить нефункциональные требования к программе. Дать не менее 5 наиболее очевидных для данной системы требований из приложения В:

Требования к содержанию отчета

В подразделе 1 описывается назначение подсистемы и цели ее создания. При описании назначения ПС указывают вид деятельности, которая автоматизируется (учет, расчет, управление, диагностика, проектирование и т.п.) и перечень объектов автоматизации, на которых предполагается ее использовать. При описании цели создания подсистемы приводят наименование и необходимые значения технических, технологических, производственно-экономических или других

показателей объекта автоматизации, которые должны быть достигнуты в результате создания ПС, и указывают критерии оценки достижения указанных целей.

В подразделе 2 указывается перечень задач, программную реализацию которых предполагается осуществить. Разрабатывается статическая объектная модель будущей программы в виде диаграммы вариантов использования и диаграммы классов. Диаграммы реализовать в пакете программ Rational Rose, Visio или Altova UModel.

В подразделе 3 при формулировании функциональных требований к программе необходимо учитывать, что потребуется обеспечить их выполнение при программировании соответствующих задач.

Изложение функциональных требований разбивается на несколько пунктов в зависимости от количества задач, например:

3.1 Требования к задаче "..."

3.2 Требования к задаче "..."

3.3 Требования к задаче "..."

В каждом пункте указывается полное название задачи. Кратко излагается, какие в точности действия программа должна выполнить для реализации данной задачи, в форме словесного или формульно-словесного описания алгоритма. Кроме того, по каждой задаче могут быть указаны следующие требования:

- к временному регламенту реализации каждой задачи;
- к качеству реализации каждой задачи;
- к выходным данным;
- к входным данным;
- к преобразованию входной информации к машиночитаемому виду;
- к возможной одновременности выполнения нескольких задач;
- к достоверности результатов.

При описании требований к входным данным приводится: перечень и описание входных данных (идентификатор, форма представления, сроки и частота поступления), перечень и описание структурных единиц информации входных сообщений или ссылка источники этих данных. В описании для каждой структурной единицы информации входных сообщений следует указывать:

- наименование;
- необходимую точность ее числового значения (при необходимости);
- источник информации (документ, видеокадр, устройство, кодограмма, информационная база на машинных носителях и т.д.);
- идентификатор источника информации.

Требования к программной реализации задач должны содержать:

- требования к организации хранения данных в виде базы данных;
- предварительное описание (структуру) интерфейса для доступа к реализации отдельных задач и подзадач;
- требования к методу программирования, к структуре кода, к именованию программных и информационных объектов.

При формулировке специальных требований к математическому обеспечению программной реализации задач приводят требования к составу, области применения (ограничение) и средств использования в системе математических методов и

моделей, типичных алгоритмов и алгоритмов, которые подлежат разработке.

В подразделе 4 выполняется построение модели требований в нотации UML. Предусмотреть расширение и включение вариантов использования. Пример модели требований в нотации UML показан на рис. 8.

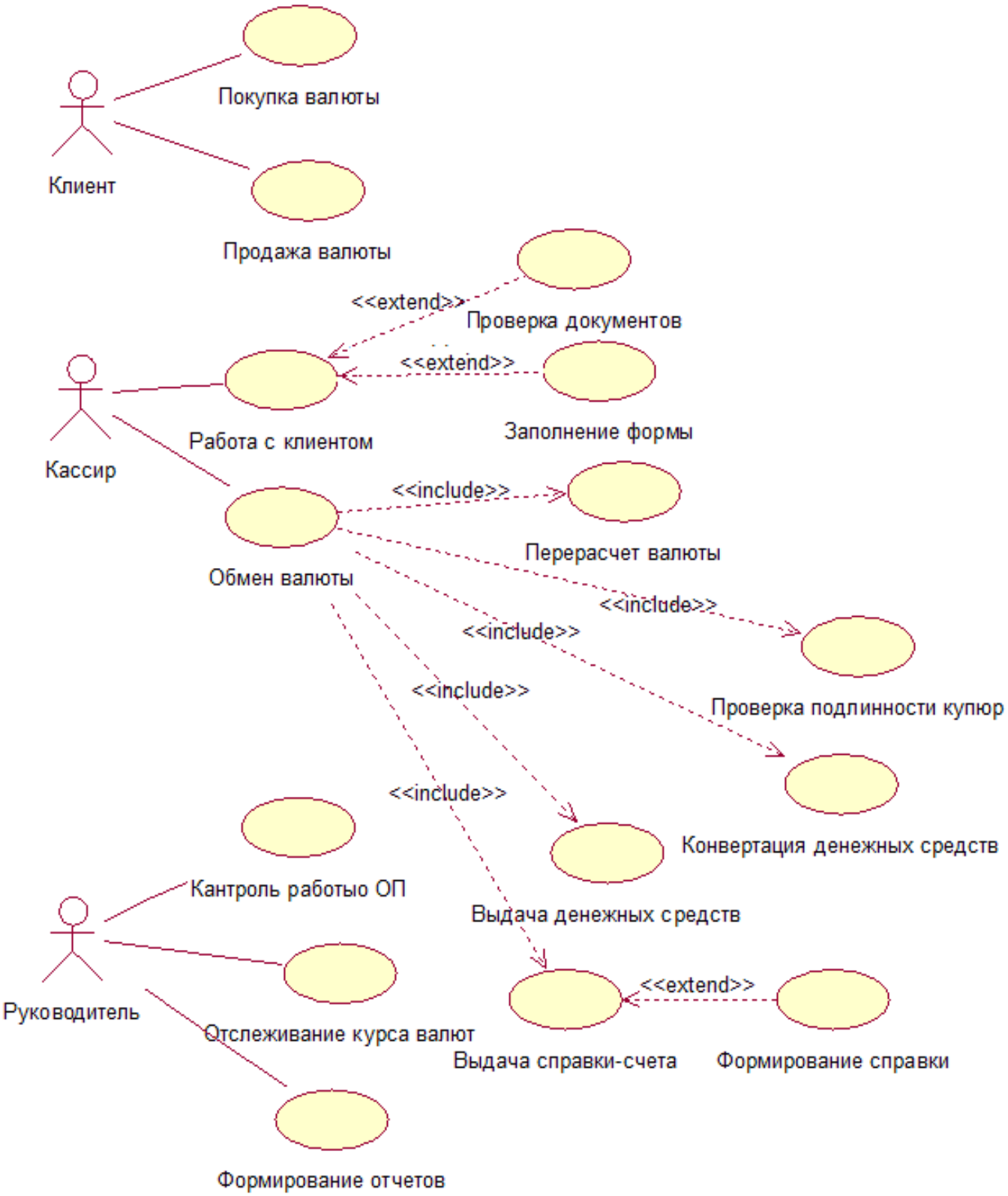


Рис. 8. Модель требований UML для ПС выполнения операции обмена валюты

В подразделе 5 указываются требования к базе данных ПС, которые могут включать следующие пункты:

– требования к организации данных, которые должны сохраняться в ПС;

– отсутствие дублирования информации и сокращение чрезмерности данных, поддержка целостности, достоверности и актуальности данных;

– низкая стоимость хранения и использования данных;

– сетевой режим доступа к общим данным, распределение информационных ресурсов в подсистеме;

– возможность получения данных с помощью языка запросов высокого уровня без использования прикладных программ;

– защита от несанкционированного доступа к данным, их порчи и уничтожения;

– обеспечение конфиденциальности секретной информации;

– возможность ведения архивов и восстановление данных в случае разрушения баз данных (БД) после сбоев.

В подразделе 6 указываются:

– требования к операционной среде;

– требования к инструментальным средствам программной инженерии, обеспечивающих разработку ПО (CASE-средства и средства объектно-ориентированного моделирования);

– требования к инструментальным средствам разработки ПО;

– требования к использованию готовых программных пакетов;

– требования к вспомогательным программным средствам (сервисные программы и утилиты).

В подразделе 7 указывают нефункциональные требования к разрабатываемой программе. Рекомендуется выбрать не менее 5 показателей, наиболее соответствующих специфике разрабатываемой ПС. Формулировки нефункциональных требований должны быть семантически и терминологически связаны с функциональными требованиями. Требования к их формулировкам такие же, как и к формулировкам функциональных требований (см. подраздел 3.3).

Все требования к подсистеме согласовываются с руководителем. ГОСТ на содержание технического задания на создание программ приведен в приложении Г.

Контрольные вопросы и упражнения

1. Что такое техническое задание и какова его структура?
2. Для чего и зачем разрабатываются компьютерные программы?
3. Какие положительные результаты могут быть получены в процессе использования компьютерной программы?
4. Что подразумевает программная реализация задач бизнес-процесса?
5. Каково назначение разрабатываемой компьютерной программы?
6. Назовите цели, в соответствии с которыми разрабатывается программа.

7. Дайте определение функционально-структурной и объектной модели компьютерной программы. Укажите принципиальные различия между этими моделями.
8. Перечислите базовые объектные модели компьютерной программы и компьютерные средства их реализации.
9. Что такое функциональные и нефункциональные требования к компьютерной программе?
10. Принципы формулировки требования к компьютерной программе при разработке технического задания.
11. Укажите основные аспекты формулирования функциональных требований к программной реализации задач бизнес-процессов.
12. Какие виды обеспечения компьютерной программы разрабатываются при ее создании?
13. Что такое прикладное обеспечение компьютерной программы?
14. Какие требования предъявляются к входным и выходным данным при программной реализации задач бизнес-процессов?
15. Какие требования предъявляются собственно к программной реализации задач бизнес-процессов?
16. Какие требования предъявляются к прикладному математическому обеспечению при программной реализации задач бизнес-процессов?
17. Улучшение каких технических, технологических, производственно-экономических или других показателей бизнес-процесса может быть достигнуто в результате создания и использования компьютерной программы?
18. Какова численность оперативного персонала, задействованного в автоматизируемых задачах до и после предполагаемого внедрения компьютерной программы?
19. Сформулировать функциональные и нефункциональные требования к программному приложению, реализующему поиск данных в односвязном списке тремя разными способами. Разработать объектные модели, описывающие работу программы.

Лабораторная работа №4

Тема: Проектирование функциональной структуры программного продукта: функционально-ориентированный подход.

Цель: изучение методики функционально-ориентированного подхода программной инженерии для разработки и описания функциональности разрабатываемого программного обеспечения.

Теоретические положения

4.1 Функционально-ориентированный подход к проектированию ПС

Функционально-ориентированный подход рассматривает объект исследования, как набор функций, преобразующий поступающий поток информации в выходной поток. Процесс преобразования информации потребляет определенные ресурсы. Принципиальное отличие данного подхода от других подходов заключается в четком отделении функций (методов обработки данных) от самих данных.

Распространенной методикой реализации функционально-ориентированного подхода является IDEF0, являющаяся следующим этапом развития графического языка описания функциональных систем SADT (Structured Analysis and Design Technique). Целью методики является построение функциональной схемы исследуемой системы, описывающей все необходимые процессы с точностью, достаточной для однозначного моделирования деятельности системы.

В основе методологии лежат четыре основных понятия: функциональный блок, интерфейсная дуга, декомпозиция, глоссарий.

Функциональный блок (Activity Box) представляет собой некоторую конкретную функцию в рамках рассматриваемой системы. По требованиям стандарта название каждого функционального блока должно быть сформулировано в глагольном наклонении (например, "производить услуги"). На диаграмме функциональный блок изображается прямоугольником с текстом названия внутри.

Интерфейсная дуга (Arrow) отображает элемент системы, который либо обрабатывается функциональным блоком, либо является результатом этой обработки, либо оказывает иное влияние на *функцию*, представленную данным функциональным блоком. С их помощью отображают объекты, в той или иной степени определяющие процессы, происходящие в системе. Это могут быть элементы реального мира (детали, вагоны, сотрудники и т.д.) или инфопотоки (документы, данные, инструкции и т.д.). В зависимости от того, к какой из сторон функционального блока подходит дуга, она носит название "входящей", "исходящей" или "управляющей".

Любой функциональный блок по требованиям стандарта должен иметь, по крайней мере, одну управляющую интерфейсную дугу и одну исходящую.

Декомпозиция (Decomposition) является основным понятием стандарта IDEF0. Принцип декомпозиции применяется при разбиении сложного процесса на составляющие его функции. При этом уровень детализации процесса определяется непосредственно разработчиком.

Декомпозиция позволяет постепенно и структурированно представлять модель системы в виде иерархической структуры отдельных диаграмм, что делает ее менее перегруженной и легко усваиваемой.

Модель IDEF0 всегда начинается с представления системы как единого целого – одного функционального блока с интерфейсными дугами, простирающимися за пределы рассматриваемой области. Такая диаграмма с одним функциональным блоком называется контекстной диаграммой.

В процессе декомпозиции базовый функциональный блок подвергается детализации на другой диаграмме. Получившаяся диаграмма второго уровня содержит функциональные блоки, отображающие главные подфункции функционального блока контекстной диаграммы, и называется дочерней (Child Diagram) по отношению к нему, а каждый функциональный блок дочерней диаграммы – дочерним блоком (Child Box). В свою очередь, функциональный блок-предок называется родительским блоком по отношению к дочерней диаграмме (Parent Box), а диаграмма, к которой он принадлежит – родительской диаграммой (Parent Diagram). Каждая из подфункций дочерней диаграммы может быть далее детализирована путем аналогичной декомпозиции соответствующего ей функционального блока. В каждом случае декомпозиции функционального блока все интерфейсные дуги, входящие в данный блок или исходящие из него, фиксируются на дочерней диаграмме. Этим достигается структурная целостность IDEF0-модели.

Обычно IDEF0-модели несут в себе сложную и концентрированную информацию. Для того чтобы ограничить их перегруженность и сделать удобочитаемыми, в стандарте приняты соответствующие ограничения сложности:

- количество функциональных блоков на диаграмме – от 3 до 6;
- количество интерфейсных дуг на один функциональный блок не более 5;
- любой функциональный блок должен иметь не менее одной управляющей (Control, Mechanism) и одной исходящей (Output) дуги;
- компоновка блоков и интерфейсных дуг с подписями должна обеспечивать максимальную читабельность диаграммы.

Преимущества ФОП выражаются в наглядности графического языка, структурной целостности модели и преемственности при организации новых проектов. Недостатком его является сложность проектирования, возрастающая вместе со сложностью проектируемой системы.

Основной недостаток функционального моделирования связан с отсутствием явных средств для объектно-ориентированного представления моделей сложных систем. Некоторые аналитики отмечают важность знания и применения нотации IDEF0, однако отсутствие возможности реализации соответствующих графических моделей в объектно-ориентированном программном коде существенно сужают диапазон задач, решаемых с ее помощью.

4.2. Пример функционального моделирования ПС

Начинать функциональное моделирование ПС следует с перечисления её автоматизированных функций. Перечисление автоматизированных функций выполняется по пунктам простыми повествовательными предложениями. Объем текста описания для несложной информационной функции ПС обычно составляет 30–50 слов. В случае описания сложной вычислительной или управляющей функции ПС слов может быть больше в разы. При этом в описание при необходимости могут быть включены формулы, таблицы и дополнительные иллюстрации. Так, например, в описание функции операции обмена валюты может быть представлена формула для расчета суммы к выдаче.

При построении контекстной диаграммы ПС за основу принимается разработанная при выполнении лабораторной работы №1 структурная схема типа «черный ящик» с обозначенными входными (Input) выходными (Output) данными, а также списков нормативно-справочной документации (Control) и экторов, задействованных в бизнес-процессе (Mechanism) (подробнее см. рис. 1). Пример контекстной диаграммы, выполненной при помощи CASE-средства BPWin, представлен на рис. 4 в требованиях к содержанию отчета о выполнении лабораторной работы №1.

Следует иметь в виду, что при выполнении лабораторной работы №1 моделью черного ящика был описан процесс информатизации как есть. На этапе постановки задачи происходит переосмысление процесса информатизации и некоторые его задачи проектируемой ПС реализуются полностью автоматически. В таком случае внешнее дополнение ПС на модели черного ящика может отличаться от исходной модели процесса информатизации. Например, взяв за основу модель, показанную на рис. 4, можно заметить, что эктор «Клиент» является лишним, так как не является пользователем ПС. К тому же очевидно, что эти данные о курсах валюты не вводятся в ПС как входные ни одним из действующих лиц. Вместо этого данные о курсах валют по логике работы программы извлекаются из корпоративной базы данных непосредственно при расчете суммы к выплате. Поэтому контекстная диаграмма для ПС обмена валют будет выглядеть так, как показано на рис. 9.

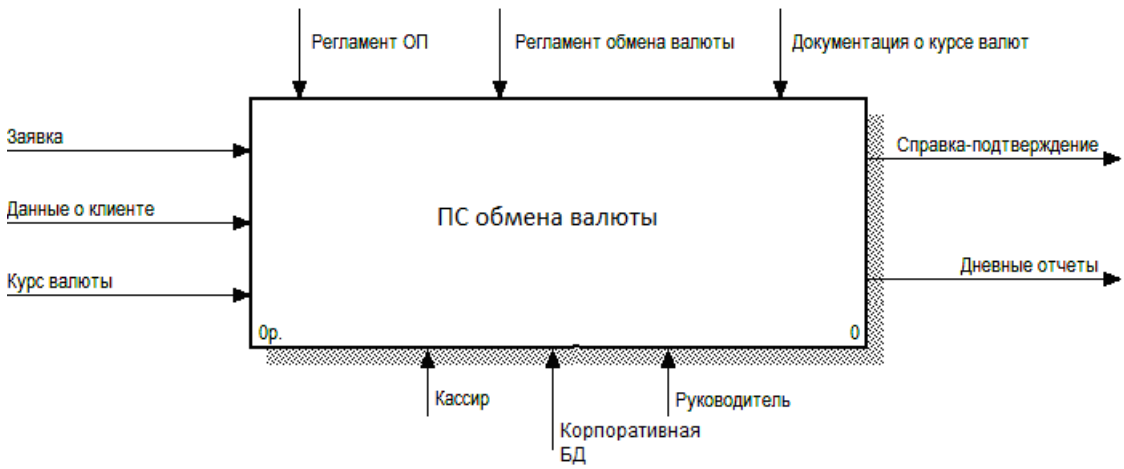


Рис. 9. Контекстная диаграмма ПС выполнения операции обмена валюты, выполненная в BPWin

Диаграмма декомпозиции такой контекстной диаграммы показана на рис. 10. Стрелка эктора «Корпоративная БД» отражает механизм автоматического запроса курса валюты при выполнении функции «Операция обмена валюты» (см. блок 2 на рис. 10).

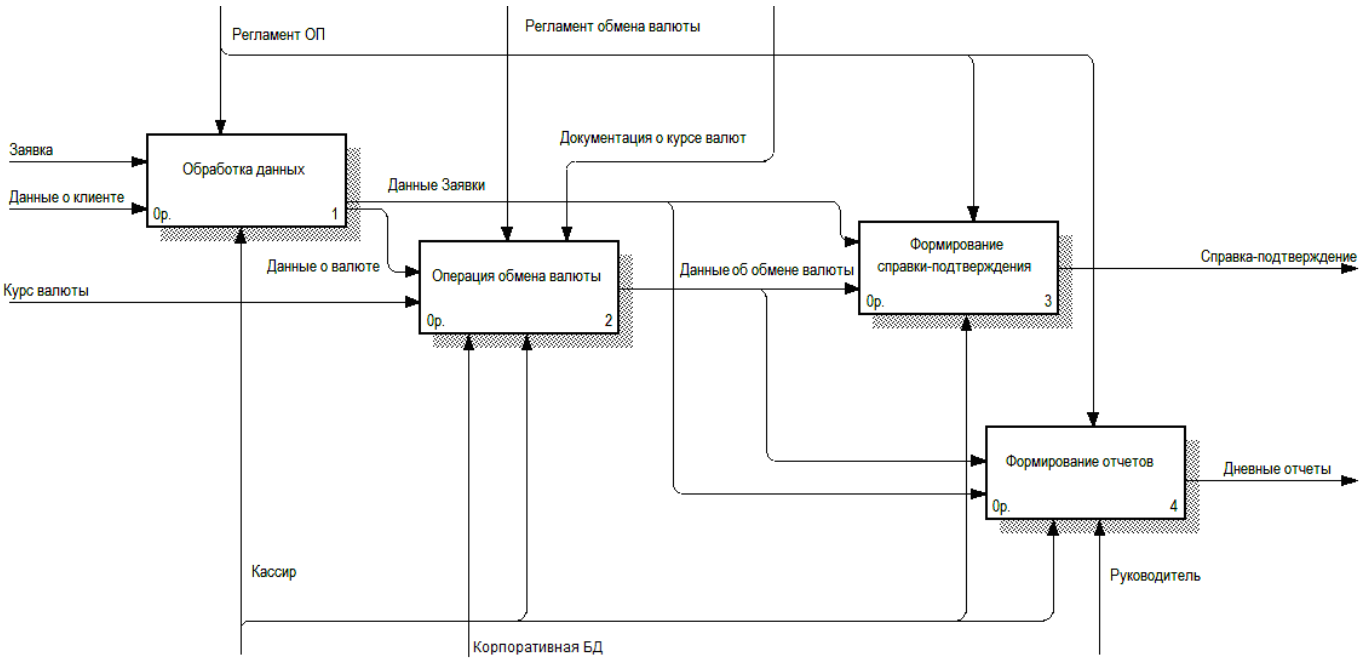


Рис. 10. Диаграмма декомпозиции ПС обмена валюты, выполненная в BPWin

Дальнейшая декомпозиция до уровня операций производится в отдельности для каждого из четырех подпроцессов – блоков диаграммы, показанной на рис. 10. При это следует руководствоваться следующими правилами:

- любая дуга может иметь ответвление, как показано на рис. 10;
- слияние дуг не допускается;
- все дуги должны быть именованы;
- во всей иерархии диаграмм не должно быть дуг с одинаковыми именами.

Задание

1. Построить функциональную модель разрабатываемого ПО в виде контекстной диаграммы в нотации IDEF0 при помощи пакета BPWin.
2. На основе контекстной диаграммы создать диаграмму декомпозиции A0 на дочерние подпроцессы (задачи).
3. Для всех функциональных блоков диаграммы A0 построить диаграммы декомпозиции A2 на подзадачи. По согласованию с преподавателем некоторые блоки могут не декомпозироваться в виду тривиальности их функционала.

Требования к содержанию отчета

В подразделе 1 выполняется перечисление автоматизированных функций ПС, на основе которого строится контекстная диаграмма в нотации IDEF0. Построение выполняется при помощи CASE-средства, например, пакета программ BPWin 4.1 или Ramus Educational (http://softrare.ru/windows/ramus).

Для перечисления автоматизированных функций ПС за основу может быть взят материал анализа предметной области из отчета о выполнении лабораторной работы №1. При этом описание автоматизированных функций ПС должно соответствовать функциональным требованиям к ПС, сформулированным в техническом задании при выполнении лабораторной работы №3. Если перечисление функций ПС не соответствует функциональным требованиям, следует модифицировать либо описание автоматизированных функций, либо формулировки функциональных требований.

В подразделе 2 также при помощи CASE-средства BPwin 4.1 строится диаграмма декомпозиции A0. Дочерние процессы диаграммы должны в точности соответствовать задачам, заявленным при формулировании функциональных требований к ПО (лабораторная работа №3). Связи (интерфейсные дуги) диаграммы должны учитывать функциональные требования к задачам, в частности в входной и выходной информации, к преобразованию данный, и др. При этом следует придерживаться требований ГОСТ.

Диаграмма декомпозиции должна быть документирована данными словарей Activity Dictionary и Arrow Dictionary. Стрелки диаграммы A0 описать в виде таблицы 3:

Таблица 3. Описание элементов функциональной модели

Наименование стрелки	Источник стрелки	Тип стрелки источника	Приемник стрелки	Тип стрелки приемника

В графах 2 и 4 указываются наименования блоков или «Внешняя граница».

В графах 3 и 5 указываются типы стрелок: Input, Output, Control, Mechanism.

В подразделе 3 выполняется дальнейшая декомпозиция задач бизнес-процесса на поздадачи – операции. При этом следует опираться на материал подраздела 2 лабораторной работы №1. В качестве операций следует определять такие, для которых можно четко сформулировать результат (Output). Отдельно рассматриваются действия, которые выполняются программными или техническими средствами, и человеком (Mechanism).

Каждая диаграмма A2 должна отражать все действия, направленные на программную обработку и хранение входной информации, а также удовлетворять вышеуказанным требованиям ГОСТ.

Документировать диаграммы A2 в формате табл. 3 не нужно.

Контрольные вопросы и упражнения

1. В чем заключается функционально-ориентированный подход к разработке и описанию функциональности компьютерной программы?
2. Каково назначение и структура функциональной модели компьютерной программы?
3. Что такое перечисление автоматизированных функций компьютерной программы?
4. Чем отличаются модели черного ящика, построенные при выполнении лабораторных работ 1 и 4?
5. Что такое нотация IDEF0?
6. Какова структура функциональной модели в нотации IDEF0?
7. В чем заключается принцип «черного ящика» и принцип декомпозиции при построении функциональной модели в нотации IDEF0?
8. Перечислите основные правила построения диаграмм в нотации IDEF0.
9. Как выражаются функциональные требования к компьютерной программе на диаграммах в нотации IDEF0?
10. Какие существуют ограничения на декомпозицию в нотации IDEF0?
11. Как документируются диаграммы в нотации IDEF0?
12. Выполните функциональное моделирование работы компьютерной программы в нотации IDEF0.

Лабораторная работа №5

Тема: Проектирование функциональной структуры программного продукта: объектно-ориентированный подход.

Цель: изучение методики объектно-ориентированного подхода программной инженерии для разработки и описания функциональности разрабатываемого программного обеспечения.

Методические указания к построению диаграмм UML прилагаются в виде отдельного документа. Использовать CASE-средство ROSE ENTERPRISE либо STAR UML.

Задание

1. Проанализировать описание функционирования программной системы, разработанного при выполнении лабораторной работы №4, на предмет выявления набора абстракций предметной области проектируемой ПС. В качестве предварительных кандидатов в абстракции принять подлежащие, выделенные из текста анализируемого потока событий.
2. Разделить выделенные абстракции на три типа: абстракции сущности, абстракции поведения, абстракции интерфейсы. Результат представить в виде таблицы 4. Для каждой абстракции указать ее класс согласно следующей классификации:
   - Люди
   - Места
   - Предметы
   - Инструменты
   - Организации
   - Концепции
   - События
   - Показатели

Таблица 4. Абстракции подсистемы

№	Абстракция	Тип	Класс	Описание

3. Проанализировать поведение выделенных абстракций. Выделить возможное поведение каждой абстракции в пределах функциональности проектируемой ПС, представленной моделью требований UML (рис. 8). Заполнить таблицу 5.

Таблица 5. Абстракции подсистемы и их поведение

№	Абстракция	Требование согласно модели UML	Описание поведения

4. Построить диаграмму классов UML (class diagram), указывая при этом лишь имена классов без указания свойств класса. Пример диаграммы приведен на рис. 11.

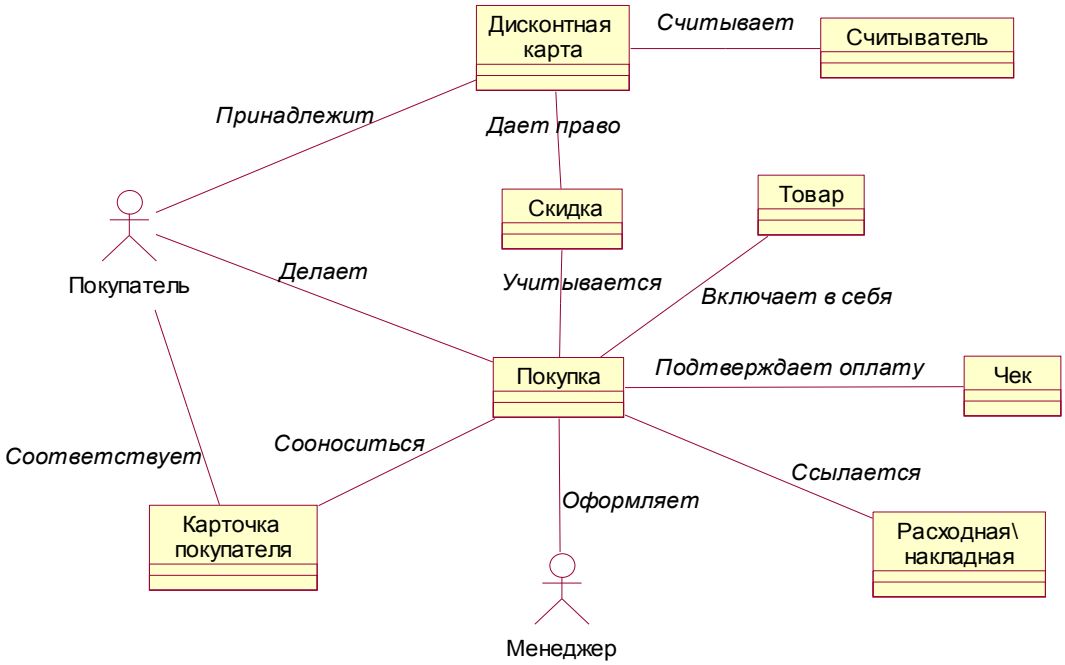


Рис. 11. Пример диаграммы классов UML

5. На основе анализа описания предметной области, разработанного при выполнении лабораторной работы №1, выявить атрибуты и операции классов. Заполнить секции атрибутов и операций классов.

6. Выбрать в модели классов такой класс, которых характеризуется наиболее частой сменой состояний, и построить для него диаграмму состояния (statechart diagram).

7. На основе анализа функциональных моделей, разработанных при выполнении лабораторной работы №4, для каждого из базовых вариантов использования построить диаграмму деятельности (activity diagram). Для вариантов использования, с которыми связаны несколько действующих лиц, диаграмму деятельности построить в виде дорожек с привязкой к исполнителям конкретных операций алгоритма

8. Для каждого варианта использования выделить список объектов участвующих во взаимодействии в этом прецеденте, заполнить таблицу 6.

Таблица 6. Список объектов для каждого потока событий

№ п.п.	Прецедент	Объект	Описание объекта

9. Создать диаграммы последовательности (sequence diagram) для перечисленных прецедентов (**одну диаграмму для всех объектов из табл. 6**).

10. Для наиболее сложных диаграмм последовательности создать кооперативные диаграммы (collaboration diagram) и доработать их, если это необходимо.

Теоретические положения

5.1. Построение модели классов

Диаграмма классов (class diagram) – диаграмма языка UML, на которой представлена совокупность декларативных или статических элементов модели, таких как классы с атрибутами и операциями, а также связывающие их отношения. Предназначена для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Когда говорят о данной диаграмме, имеют в виду статическую структурную модель проектируемой системы, т.е. графическое представление таких структурных взаимосвязей логической модели системы, которые не зависят от времени.

Класс (class) – абстрактное описание множества однородных объектов, имеющих одинаковые атрибуты, операции и отношения с объектами других классов. Графически класс в нотации UML изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции (рис. 12). В этих секциях указываются имя класса, атрибуты и операции класса.

На начальных этапах разработки диаграммы отдельные классы могут обозначаться простым прямоугольником, в котором должно быть указано его имя (рис. 12, а). По мере проработки отдельных компонентов диаграммы описания классов дополняются атрибутами (рис. 12, б) и операциями (рис. 12, в). Четвертая секция (рис. 12, г) не обязательна и служит для размещения дополнительной информации справочного характера, например, об исключениях или ограничениях класса, сведения о разработчике или языке реализации класса. Предполагается, что окончательный вариант диаграммы содержит наиболее полные описания классов, которые состоят из трех или четырех секций.

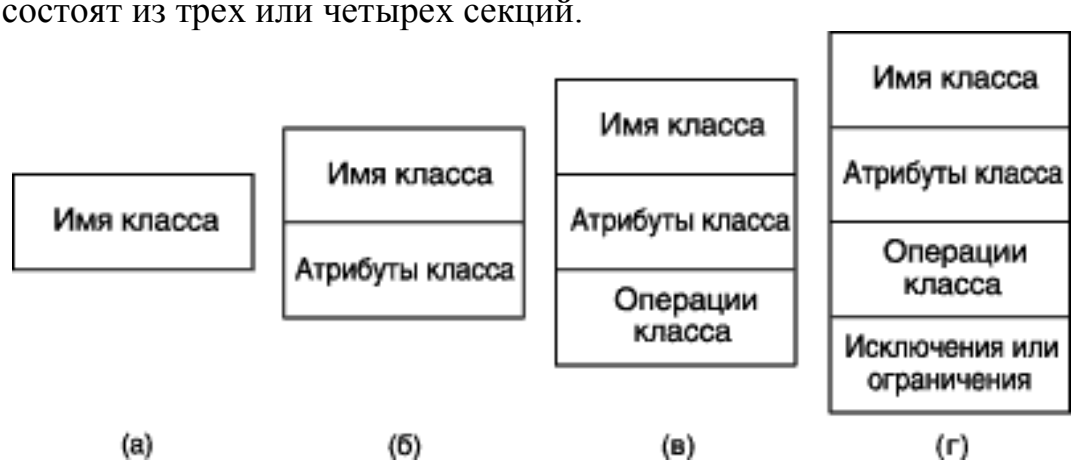


Рис. 12. Варианты графического изображения класса на диаграмме

Имя класса должно быть уникальным в пределах пакета, который может содержать одну или несколько диаграмм классов. Имя указывается в самой верхней секции прямоугольника, поэтому она называется секцией имени класса. В дополнение к общему правилу именования элементов языка UML, имя класса записывается по центру секции полужирным шрифтом и должно начинаться с заглавной буквы. Рекомендуется в качестве имен классов использовать имена

существительные, записанные по практическим соображениям без пробелов. Необходимо помнить, что имена классов образуют словарь предметной области при проектировании системы.

В секции имени класса могут также находиться стереотипы или ссылки на стандартные шаблоны, от которых образован данный класс и, соответственно, от которых он наследует свои свойства: атрибуты и операции. В этой секции может также приводиться информация о разработчике класса и статус состояния разработки.

Атрибут (attribute) – содержательная характеристика класса, описывающая множество значений, которые могут принимать отдельные объекты этого класса. Атрибут класса служит для представления отдельного свойства или признака, который является общим для всех объектов данного класса. Атрибуты класса записываются в секции атрибутов (рис. 12, б).

Операция (operation) – это сервис, предоставляемый каждым экземпляром или объектом класса по требованию своих клиентов, в качестве которых могут выступать другие объекты, в том числе и экземпляры данного класса. Операции класса записываются в секции операций (рис. 12, в). Совокупность операций характеризует функциональный аспект поведения всех объектов данного класса, который описывается другими каноническими диаграммами UML. Имя операции представляет собой строку текста, которая используется в качестве идентификатора соответствующей операции и должна быть уникальной в пределах данного класса. Имя операции должно начинаться со строчной буквы, и, как правило, записываться без пробелов.

Идентификация классов анализа заключается в предварительном определении набора классов системы на основе описания предметной области. Прежде всего, выделяются основные абстракции предметной области. Способы идентификации основных абстракций аналогичны способам идентификации сущностей в модели "сущность-связь" – поиск абстракций, описывающих физические или материальные объекты, процессы и события, роли людей, организации и другие понятия предметной области. Единственным формальным способом идентификации сущностей является анализ текстовых описаний предметной области, выделение из описаний имен существительных и выбор их в качестве "кандидатов" на роль абстракций. Каждая сущность должна иметь наименование, выраженное существительным в единственном числе. Далее абстракции идентифицируются по трем типам, которые используются для уточнения семантики отдельных классов:

1. Управляющий класс (control class) – класс, отвечающий за координацию действий других классов на диаграмме. На каждой диаграмме классов должен быть хотя бы один управляющий класс, который контролирует последовательность выполнения действий данного варианта использования. Данный класс является активным и инициирует рассылку множества сообщений другим классам модели. Управляющий класс изображается в форме прямоугольника класса со стереотипом <<control>> в секции имени. Примеры управляющих классов: менеджер транзакций, журнал операций, координатор ресурсов, обработчик событий, обработчик ошибок, и т.п.

2. Класс-сущность (entity class) – пассивный класс, информация о котором должна храниться постоянно и не уничтожаться с выключением системы или завершением работы программы. Как правило, этот класс соответствует отдельной сущности логической модели данных системы. В этом случае его атрибуты являются потенциальными полями таблицы, а операции – присоединенными или хранимыми процедурами. Этот класс пассивный и лишь принимает сообщения от других классов модели, реагируя на них посредством своих операций.

Источники выявления классов-сущностей:

- ключевые абстракции, созданные в процессе архитектурного анализа,
- глоссарий проекта,
- описание потоков событий вариантов использования.

Класс-сущность может быть изображен стандартным образом в форме прямоугольника класса со стереотипом <<entity>> с секции имени класса.

3. Граничный класс (boundary class) – класс, который располагается на границе системы с внешней средой и непосредственно взаимодействует с актерами, но, тем не менее, является составной частью системы.

Как правило, для каждой пары "действующее лицо – вариант использования" определяется один граничный класс. Типы граничных классов:

- пользовательский интерфейс, выражающий обмен информацией с пользователем без деталей интерфейса: кнопок, списков, окон, и т.п.
- системный интерфейс с надсистемой или другой системой (используемые протоколы обмена без деталей их реализации);
- аппаратный интерфейс для связи с внешними техническими устройствами (аппаратные средства, протоколы, драйверы).

Граничный класс может быть изображен также стандартным образом в форме прямоугольника класса со стереотипом <<boundary>> в секции имени.

5.2. Отношения между классами

Классы вступают друг с другом в отношения. На диаграмме классов выделяют такие виды отношений: ассоциация, наследование, агрегирование и использование. При изображении конкретной связи ей можно сопоставить текстовую метку, документирующую имя этой связи и/или ее роль. Имя отношения должно быть уникально в своем контексте. Также связь может иметь по ее концам обозначения кратности (множественности) отношений между классами, некоторые из которых показаны ни рис. 13.

*	неопределенное множество
1..*	один или много
0..1	ноль или 1
1..8	от 1 до 8
12	точно 12
{2,3,5,7,11}	определенное множество

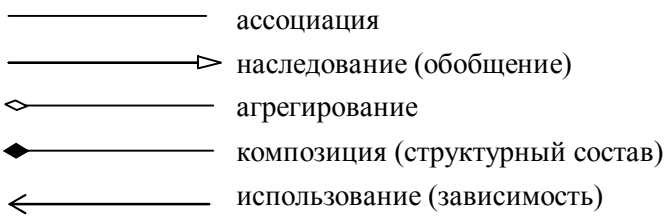


Рис. 13. Обозначения множественности и значки отношений между классами

Значок ассоциации соединяет два класса и означает наличие семантической связи между ними. Ассоциации часто отмечаются существительными (например, Выбор), описывающими природу связи. Одна пара классов может иметь одну или несколько ассоциативных связей.

Наследование представляет отношение типа "общее/частное", выглядит как значок ассоциации со стрелкой, которая указывает от подкласса к суперклассу. При этом подкласс наследует структуру и поведение своего суперкласса. Класс может иметь один (одиночное наследование), или несколько (множественное наследование) суперклассов.

Агрегирование обозначает отношение "целое/часть" и получается из значка ассоциации добавлением закрашенного кружка на конце, обозначающем агрегат. Экземпляры класса на другом конце стрелки будут в определенном смысле частями экземпляров класса-агрегата. Разрешается рефлексивная и циклическая агрегация. Агрегация не требует обязательного физического включения части в целое.

Композиция – это разновидность агрегирования с четко выраженным отношением владения, причем время жизни частей и целого совпадают. Части с нефиксированной кратностью могут быть созданы уже после создания самого композита, живут и уничтожаются вместе с ним. Также части могут быть удалены явным образом еще до уничтожения композита. Кроме того, в композитном агрегировании целое отвечает за диспозицию своих частей, то есть композит должен управлять их созданием и уничтожением.

Использование обозначает отношение типа "клиент/сервер" и изображается как ассоциация с пустым кружком на конце, соответствующем клиенту. Эта связь выражает зависимость клиента от сервера и означает, что клиент нуждается в услугах сервера, то есть операции класса-клиента вызывают операции класса-сервера или имеют сигнатуру, в которой возвращаемое значение и/или аргументы принадлежат классу сервера. Фактически отношение использования показывает, что один элемент использует другой.

Общие рекомендации по построению диаграмм классов:

- использовать зависимость, только в случае, если моделируемое отношение не является структурным;
- использовать обобщение, только если имеет место отношение типа "является";
- располагать элементы так, чтобы свести к минимуму число пересекающиеся линий отношений;
- пространственно организовывать элементы так, чтобы семантически близкие сущности располагались рядом;
- чтобы привлечь внимание к важным особенностям диаграммы, использовать примечания (комментарии) и выделение цветом;

Однако доскональное описание динамики моделируемой системы представляется диаграммами поведения UML. Для моделирования поведения на логическом уровне в языке UML могут использоваться сразу несколько канонических диаграмм: состояний, деятельности, последовательности и

кооперации, каждая из которых фиксирует внимание на отдельном аспекте функционирования системы.

5.3. Диаграммы состояний

В отличие от других диаграмм диаграмма состояний описывает процесс изменения состояний одного экземпляра определенного класса, т. е. моделирует все возможные изменения в состоянии конкретного объекта. Они определяют все возможные состояния, в которых может находиться объект, а также процесс смены состояний объекта в результате некоторых событий. Эти диаграммы обычно используются для описания поведения одного объекта в нескольких прецедентах. При этом изменение состояния объекта может быть вызвано внешними воздействиями со стороны других объектов или извне. Именно для описания реакции объекта на подобные внешние воздействия и используются диаграммы состояний.

Основными понятиями, входящими в формализм автомата, являются состояние и переход. Главное различие между ними заключается в том, что длительность нахождения системы в отдельном состоянии существенно превышает время, которое затрачивается на переход из одного состояния в другое. Предполагается, что в пределе время перехода из одного состояния в другое равно нулю (если дополнительно ничего не сказано). Другими словами, переход объекта из состояния в состояние происходит мгновенно.

В языке UML под состоянием понимается абстрактный метакласс, используемый для моделирования отдельной ситуации, в течение которой имеет место выполнение некоторого условия. Состояние может быть задано в виде набора конкретных значений атрибутов объекта, при этом изменение их отдельных значений будет отражать изменение состояния моделируемого класса или объекта. Представляются прямоугольниками со скругленными углами (рис. 14).

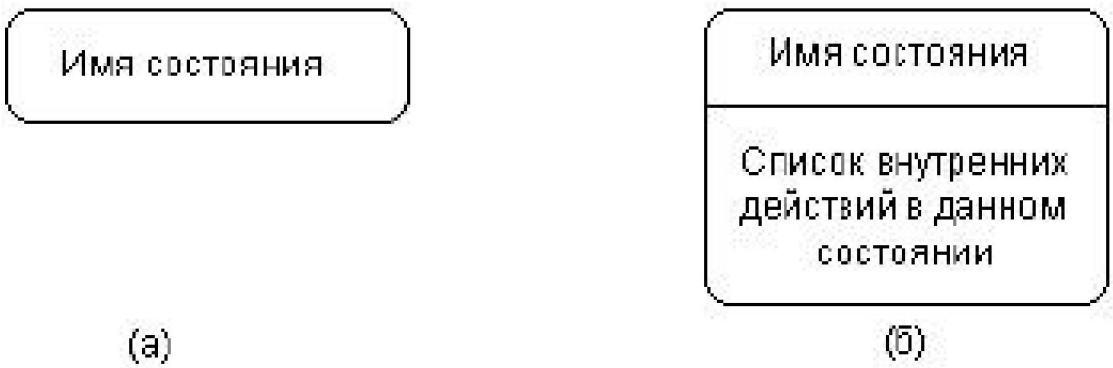


Рис. 14. Графическое изображение состояний на диаграмме состояний

При этом под действием в языке UML понимают некоторую атомарную операцию, выполнение которой приводит к изменению состояния или возврату некоторого значения (например, "истина" или "ложь").

Имя состояния представляет собой строку текста, которая раскрывает содержательный смысл данного состояния. Имя всегда записывается с заглавной буквы. Поскольку состояние системы является составной частью процесса ее

функционирования, рекомендуется в качестве имени использовать глаголы в настоящем времени (звенит, печатает, ожидает) или соответствующие причастия (занят, свободен, передано, получено). Имя у состояния может отсутствовать, т. е. оно является необязательным для некоторых состояний. В этом случае состояние является анонимным, и если на диаграмме состояний их несколько, то все они должны различаться между собой.

Секция внутренних действий содержит перечень внутренних действий или деятельностей, которые выполняются в процессе нахождения моделируемого элемента в данном состоянии. Каждое из действий записывается в виде отдельной строки и имеет следующий формат:

<метка-действия '/' выражение-действия>

Метка действия указывает на обстоятельства или условия, при которых будет выполняться деятельность, определенная выражением действия. При этом выражение действия может использовать любые атрибуты и связи, которые принадлежат области имен или контексту моделируемого объекта. Если список выражений действия пустой, то разделитель в виде наклонной черты '/' может не указываться.

Перечень меток действия имеет фиксированные значения в языке UML, которые не могут быть использованы в качестве имен событий. Эти значения следующие:

entry — эта метка указывает на действие, специфицированное следующим за ней выражением действия, которое выполняется в момент входа в данное состояние (входное действие);

exit — эта метка указывает на действие, специфицированное следующим за ней выражением действия, которое выполняется в момент выхода из данного состояния (выходное действие);

do — эта метка специфицирует выполняющуюся деятельность ("do activity"), которая выполняется в течение всего времени, пока объект находится в данном состоянии, или до тех пор, пока не закончится вычисление, специфицированное следующим за ней выражением действия. В последнем случае при завершении события генерируется соответствующий результат.

include — эта метка используется для обращения к подавтомату, при этом следующее за ней выражение действия содержит имя этого подавтомата.

Во всех остальных случаях метка действия идентифицирует событие, которое запускает соответствующее выражение действия. Эти события называются внутренними переходами и семантически эквивалентны переходам в само это состояние, за исключением той особенности, что выход из этого состояния или повторный вход в него не происходит. Это означает, что действия входа и выхода не выполняются.

В качестве примера состояния рассмотрим ситуацию ввода пароля пользователя при аутентификации входа в некоторую программную систему (рис. 15). В этом случае список внутренних действий в данном состоянии не пуст и

включает 4 отдельных действия, первые два из которых стандартные и описаны выше, а два последних определяются своей спецификацией.

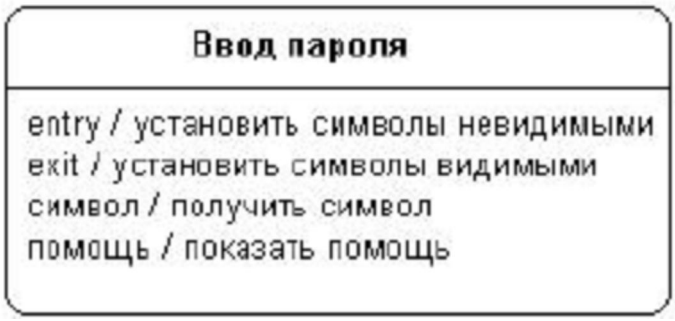


Рис. 15. Пример состояния с непустой секцией внутренних действий

Начальное состояние

Начальное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий (псевдосостояния). В этом состоянии находится объект по умолчанию в начальный момент времени. Оно служит для указания на диаграмме состояний графической области, от которой начинается процесс изменения состояний. Графически начальное состояние в языке UML обозначается в виде закрашенного кружка (рис. 16, а), из которого может только выходить стрелка, соответствующая переходу.

Рис. 16. Графическое изображение начального и конечного состояний на диаграмме состояний

На самом верхнем уровне представления объекта переход из начального состояния может быть помечен событием создания (инициализации) данного объекта. В противном случае переход никак не помечается. Если этот переход не помечен, то он является первым переходом в следующее за ним состояние.

Конечное состояние

Конечное (финальное) состояние представляет собой частный случай состояния, которое также не содержит никаких внутренних действий (псевдосостояния). В этом состоянии будет находиться объект по умолчанию после завершения работы автомата в конечный момент времени. Оно служит для указания на диаграмме состояний графической области, в которой завершается процесс изменения состояний или жизненный цикл данного объекта. Графически конечное

состояние в языке UML обозначается в виде закрашенного кружка, помещенного в окружность (рис. 16, б), в которую может только входить стрелка, соответствующая переходу.

Переход

Простой переход (simple transition) представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния другим. Пребывание моделируемого объекта в первом состоянии может сопровождаться выполнением некоторых действий, а переход во второе состояние будет возможен после завершения этих действий, а также после удовлетворения некоторых дополнительных условий. В этом случае говорят, что переход срабатывает, Или происходит срабатывание перехода. До срабатывания перехода объект находится в предыдущем от него состоянии, называемым исходным состоянием, или в источнике (не путать с начальным состоянием — это разные понятия), а после его срабатывания объект находится в последующем от него состоянии (целевом состоянии).

Переход осуществляется при наступлении некоторого события: окончания выполнения деятельности (do activity), получении объектом сообщения или приемом сигнала. На переходе указывается имя события. Кроме того, на переходе могут указываться действия, производимые объектом в ответ на внешние события при переходе из одного состояния в другое. Срабатывание перехода может зависеть не только от наступления некоторого события, но и от выполнения определенного условия, называемого сторожевым условием. Объект перейдет из одного состояния в другое в том случае, если произошло указанное событие и сторожевое условие приняло значение "истина".

Переходы представляются стрелками от одного состояния к другому, которые вызываются выполнением некоторых функций объекта. Переходы имеют метки, которые синтаксически состоят из трех необязательных частей (см. рис. 17): <Событие> <[Условие]> < / Действие>.

Переходы между составными состояниями

Переход, стрелка которого соединена с границей некоторого составного состояния, обозначает переход в составное состояние. Он эквивалентен переходу в начальное состояние каждого из подавтоматов (возможно, единственному), входящих в состав данного суперсостояния.

Переходы между параллельными состояниями

В отдельных случаях переход может иметь несколько состояний-источников и несколько целевых состояний. Такой переход получил специальное название — параллельный переход. Введение в рассмотрение параллельного перехода обусловлено необходимостью синхронизировать и/или разделить отдельные подпроцессы на параллельные нити без спецификации дополнительной синхронизации в параллельных подавтоматах.

Рис. 17. Диаграмма состояний объекта «заказ»

Графически такой переход изображается вертикальной черточкой, аналогично обозначению перехода в известном формализме сетей Петри. Если параллельный переход имеет две или более входящих дуг (рис. 18, а), то его называют соединением (join). Если же он имеет две или более исходящих из него дуг (рис. 18 б), то его называют ветвлением (fork).

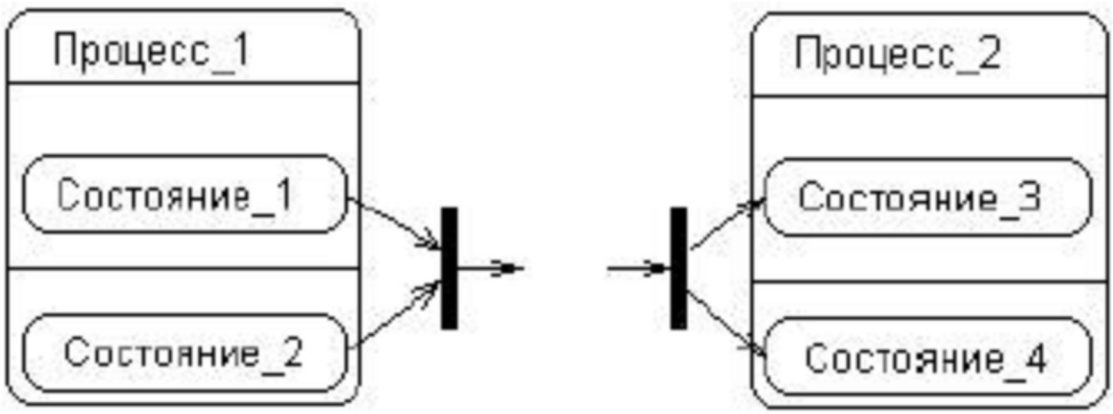


Рис. 18. Различные варианты переходов в (из) составное состояние

Синхронизирующие состояния

Как уже было отмечено, поведение параллельных подавтоматов независимо друг от друга, что позволяет реализовать многозадачность в программных системах. Однако в отдельных случаях может возникнуть необходимость учесть в модели синхронизацию наступления отдельных событий. Для этой цели в языке UML имеется специальное псевдосостояние, которое называется синхронизирующим состоянием.

Синхронизирующее состояние (synch state) обозначается небольшой окружностью, внутри которой помещен символ звездочки "*". Оно используется совместно с переходом-соединением или переходом-ветвлением для того, чтобы явно указать события в других подавтоматах, оказывающие непосредственное влияние на поведение данного подавтомата.

Заключительные рекомендации по построению диаграмм состояний

Основные особенности построения диаграмм состояний были рассмотрены при описании соответствующих модельных элементов, входящих в пакет Автоматы. Однако некоторые моменты не нашли отражения, о чем необходимо сказать в заключение этой главы.

По своему назначению диаграмма состояний не является обязательным представлением в модели и как бы "присоединяется" к тому элементу, который, по замыслу разработчиков, имеет нетривиальное поведение в течение своего жизненного цикла. Наличие у системы нескольких состояний, отличающихся от простой дихотомии "исправен — неисправен", "активен — неактивен", "ожидание — реакция на внешние действия", уже служит признаком необходимости построения диаграммы состояний. В качестве начального варианта диаграммы состояний, если нет очевидных соображений по поводу состояний объекта, можно воспользоваться этими суперсостояниями, рассматривая их как составные и уточняя их (детализируя их внутреннюю структуру) по мере рассмотрения логики поведения объекта.

При выделении состояний и переходов следует помнить, что длительность срабатывания отдельных переходов должна быть существенно меньшей, чем нахождение моделируемого объекта в соответствующих состояниях. Каждое из состояний должно характеризоваться определенной устойчивостью во времени. Другими словами, из каждого состояния на диаграмме не может быть самопроизвольного перехода в какое бы то ни было другое состояние. Все переходы должны быть явно специфицированы, в противном случае построенная диаграмма состояний является либо неполной, либо ошибочной.

При разработке диаграммы состояний нужно постоянно следить, чтобы объект в каждый момент мог находиться только в единственном состоянии. Если это не так, то данное обстоятельство может быть как следствием ошибки, так и неявным признаком наличия параллельности у поведения моделируемого объекта. В последнем случае следует явно специфицировать необходимое число подавтоматов,

вложив их в то составное состояние, которое характеризуется нарушением условия одновременности.

Следует обязательно проверять, что никакие два перехода из одного состояния не могут сработать одновременно (требование отсутствия конфликтов у переходов). Наличие такого конфликта может служить признаком ошибки либо неявной параллельности типа ветвления рассматриваемого процесса на два и более подавтомата. Если параллельность по замыслу разработчика отсутствует, то необходимо ввести дополнительные сторожевые условия либо изменить существующие, чтобы исключить конфликт переходов. При наличии параллельности следует заменить конфликтующие переходы одним параллельным переходом типа ветвления.

Использование исторических состояний оправдано в том случае, когда необходимо организовать обработку исключительных ситуаций (прерываний) без потери данных или выполненной работы. При этом применять исторические состояния, особенно глубокие, надо с известной долей осторожности. Нужно помнить, что каждый из подавтоматов может иметь только одно историческое состояние. В противном случае возможны ошибки, особенно когда подавтоматы изображаются на отдельных диаграммах состояний.

И, наконец, следует отметить, что некоторые дополнительные конструкции автоматов, такие как точки динамического выбора (dynamic choice points) или точки соединения (junction points), в книге не нашли отражения. Это сделано по той причине, что данные модельные элементы хотя и позволяют моделировать более сложные аспекты динамического управления поведением объекта, не являются базовыми. Соответствующая информация содержится в оригинальной документации по языку UML.

5.4. Диаграммы деятельности

Диаграмма деятельности — это частный случай *диаграммы состояний*. На диаграмме деятельности представлены переходы потока управления от одной деятельности к другой внутри системы. Этот вид диаграмм обычно используется для описания поведения, включающего в себя множество параллельных процессов.

Основными элементами диаграмм деятельности являются (рис. 19):

- овалы, изображающие действия объекта;
- линейки синхронизации, указывающие на необходимость завершить или начать несколько действий (модель логического условия "И");
- ромбы, отражающие принятие решений по выбору одного из маршрутов выполнения процесса (модель логического условия "ИЛИ");
- стрелки — отражают последовательность действий, могут иметь метки условий.

На диаграмме деятельности могут быть представлены действия, соответствующие нескольким вариантам использования. На таких диаграммах появляется множество начальных точек, поскольку они отражают теперь реакцию системы на множество внешних событий. Таким образом, диаграммы деятельности позволяют получить полную картину поведения системы и легко оценивать влияние изменений в отдельных вариантах использования на конечное поведение системы.

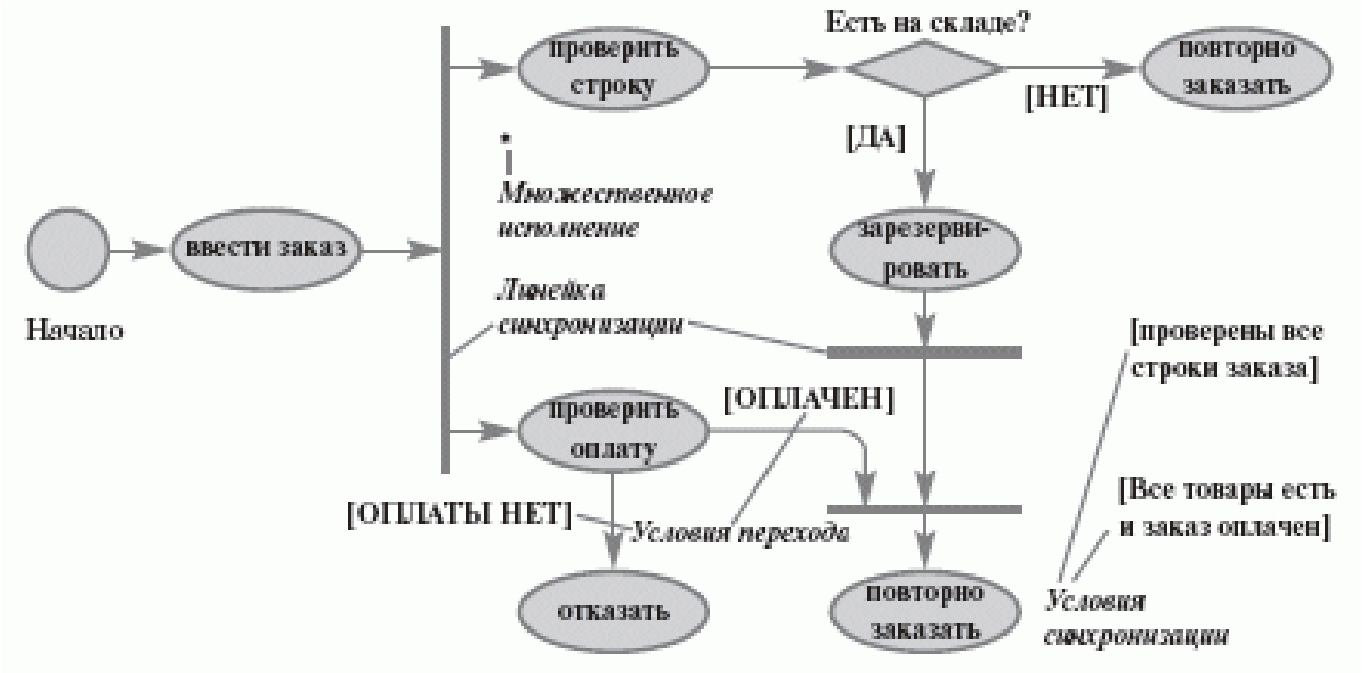


Рис. 19. Диаграмма деятельности — обработка заказа

Любая деятельность может быть подвергнута дальнейшей декомпозиции и представлена в виде отдельной диаграммы деятельности или спецификации (словесного описания).

Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой операции в предыдущем состоянии. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами — переходы от одного состояния действия к другому.

Достоинством диаграммы деятельности является возможность развёртывания её в виде дорожек, т.е. с привязкой к исполнителям конкретных операций алгоритма.

Пример: Вариант использования «Складирование товара»:

Цель процесса – подготовить склад для работы с торговыми представителями. В рамках данного процесса выполняется размещение поступившего товара на складе, контроль качества товара, учет товара и ввод результатов учета в базу данных программы «Кладовщик-оператор».

Участники: В процессе участвуют: грузчик, лаборант, диспетчер, кладовщик, бирковщик, фасовщик, упаковщик.

Схема диаграммы деятельности для задачи складирования товара с дорожками показана на рис. 20.

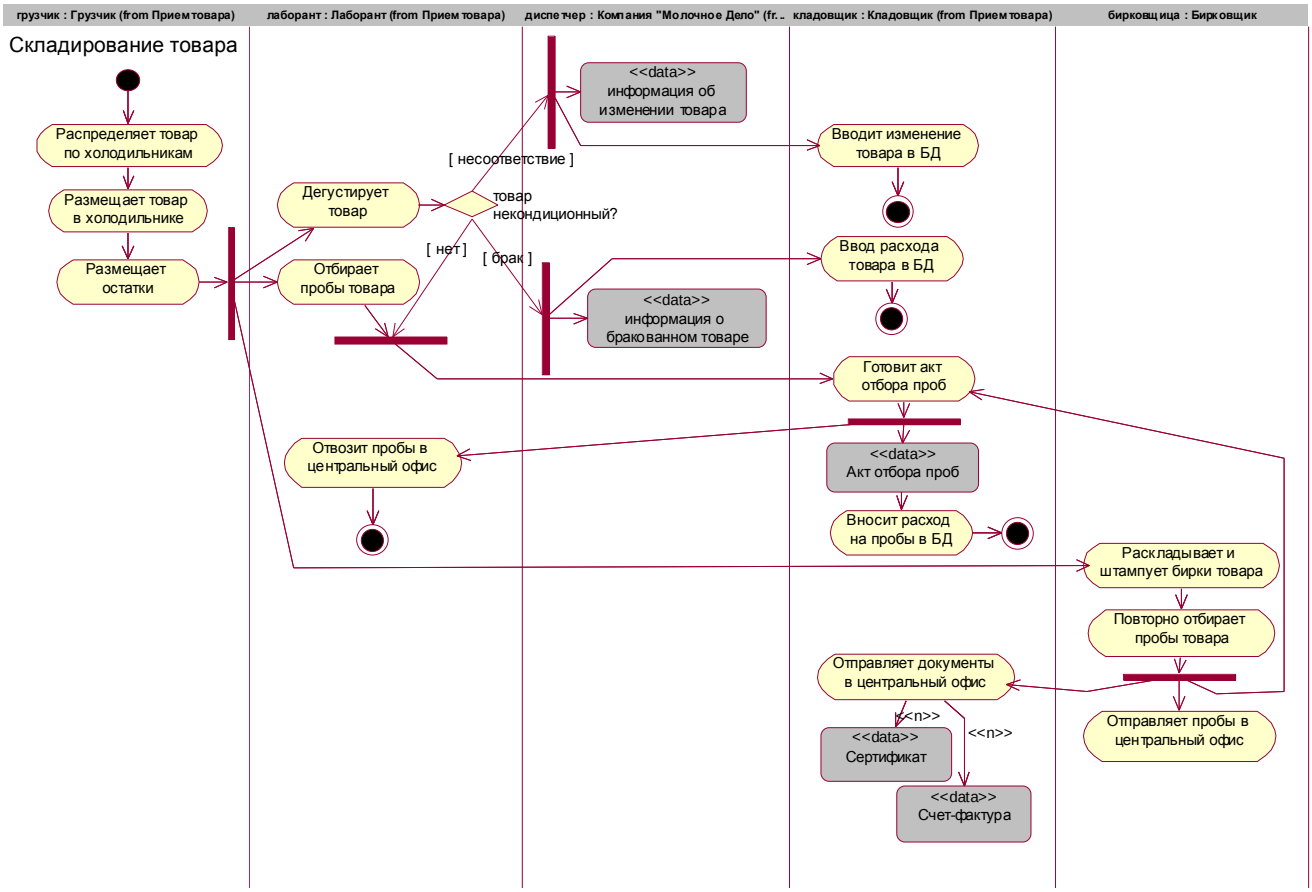


Рис. 20. Пример фрагмента диаграммы деятельности для задачи складирования товара с дорожками

Описание модели:

a) **Распределяет товар по холодильникам**. Выполняется грузчиком, кладовщиком; ответственный исполнитель кладовщик. Грузчик развозит товар по холодильникам склада в указанные кладовщиком места (фактически места распределяются кладовщиками 1 раз, далее грузчики самостоятельно развозят товар). Схема размещения видов товаров по холодильникам определяется кладовщиками в оперативном порядке. Такая сортировка нужна для того, чтобы при погрузке было проще работать, не нужно было бы тратить время на поиск нужного товара и т.д.

b) **Размещает товар в холодильнике**. Выполняется грузчиком, кладовщиком; ответственный исполнитель кладовщик. Развозит товар на определенные места в холодильнике. Схема размещения товара внутри холодильника также определяется кладовщиком, с той же целью, что и распределение товара по холодильникам.

c) **Размещает остатки**. Выполняется грузчиком. Остатки размещают так, чтобы отгрузить их потом в первую очередь. Под остатками понимается товар, поступивший на склад из других офисов (не с производственных предприятий компании), или товар, который не был отгружен в предыдущую смену.

d) **Дегустирует товар**. Выполняется лаборантом, диспетчером; ответственный исполнитель лаборант. После того, как товар развезли по местам в

холодильнике, лаборант проверяет качество товара. Приезжает ежедневно (обычно с 22:00 и до разгрузки последней машины). На месте лаборант проверяет качество продукции, дегустируя ее. Если обнаруживает какие-нибудь отклонения – бракует, сообщает об этом кладовщику. Затем кладовщик или лаборант сообщает диспетчеру о наличии некондиционного товара («товар такой-то в таком-то количестве забракован»), потом указывают в программе, что этот товар в соответствующем количестве «приостановлен к продаже» (ушел в «расход», «порча»). Кроме того, по заключению лаборанта и решению менеджера по качеству кладовщики могут сменить тип товара (например, сметана 30% поступает жидкая – ее записывают как сметану 15%). В этом случае тоже уведомляется диспетчер.

e) **Отбирает пробы товара.** Выполняется лаборантом. Лаборант берет пробы всех товаров для проверки их в лаборатории.
f) **Вводит изменения товара в БД**. Выполняется кладовщиком. Информация об изменении типа товара вносится в БД программы «Кладовщик-оператор».
g) **Вносит расход на пробы в БД**. Выполняется кладовщиком. Кладовщик вносит информацию о количестве отобранного на пробы товара в БД программы «Кладовщик-оператор». Информация для внесения в БД берется из акта отбора проб.
h) **Готовит акт отбора проб.** Выполняется лаборантом, кладовщиком; ответственный исполнитель кладовщик. Пробы взвешиваются, по результатам взвешивания составляется расходная накладная (на «дегустацию»); данные из расходной накладной кладовщик вносит в базу данных программы «Кладовщик-Оператор».
i) **Отвозит пробы в центральный офис**. Выполняется лаборантом. После отбора проб и составления акта лаборант отвозит пробы в лабораторию в ВНИМИ.
j) **Раскладывает и штампует бирки товара**. Выполняется бирковщиком. Делает бирки для всего привезенного товара: после привоза и складирования товара бирковщик обходит склад и для каждого товара указывает (штампует на соответствующих бирках) дату производства. Указывается дата, время. Для товаров, являющихся остатками, указывается дата их поступления на склад. Сами бирки заказываются в типографии: завскладом по телефону заказывает бирки для весовых товаров (по заявке бирковщика) в типографии, параллельно уведомляет об этом по телефону службу снабжения компании.
k) **Повторно отбирает пробы товара**. Выполняется бирковщиком. По запросу от менеджера по качеству независимо от лаборанта отбирает пробы продукции для дегустациив центральном офисе Компании.
l) **Отправляет пробы в центральный офис**. Выполняется бирковщиком, курьером; ответственный исполнитель бирковщик. Бирковщик утром передает отобранные пробы товара курьеру (водителю) из отдела логистики. Курьер отвозит пробы вместе с документами и сертификатами в центральный офис.
m) **Отправляет документы в центральный офис**. Выполняется кладовщиком, диспетчером; ответственный исполнитель кладовщик. Диспетчеру по электронной почте передаются данные о недостатках/излишках, остатках,

утренний отчет по внешним поставщикам, отчет по внутренним поставщикам, отчет сдачи склада, счет-фактуры и накладные поставщиков.

особенности

В некоторых случаях фасовщиками на складе выполняется перефасовка товара:

- Если товар прибыл в упаковках большего размера, чем принимают торговые точки (коробки по 150 плавленных сырков, весовой товар в 10-литровых ведрах, прочее), фасовщики заменяют тару на подходящую для торговых точек.
- Если товар прибыл в поврежденной упаковке, он либо отправляется обратно производителю (например, если тара - стаканчики), либо упаковка заменяется на другую. Решение принимается менеджером по качеству.
- Если на складе есть товар, срок годности которого истекает, для того чтобы оперативно его продать он также может быть перефасован (например, творог в пачках перефасовывается в ведра 3,5 литра). При такой перефасовке тип товара изменяется. Решение принимается начальником транспортно-диспетчерского отдела и менеджером по качеству.

Основные правила замены тары при перефасовке: ведра 10 литров (весовой товар) на ведра 3,5 литра (весовой товар); стаканчики на пакеты (весовой товар); пачки (штучный товар) на ведра 3,5 литра (весовой товар).

После того, как перефасовка выполнена, кладовщик проверяет результат, производит взвешивание полученного товара, и вносит эти данные в БД программного средства «Кладовщик-оператор». При этом производятся спецпроводки, отмечающие перевод товара из одной тары (и/или типа товара) в другую: в расходной накладной на фасовку отмечается количество старого товара в таре и новой тары, в приходной накладной – количество товара в новой таре и количество старой тары.

В случае если товар от поставщика прибыл в поврежденной упаковке (например, верда упаковываются в полиэтиленовые пакеты, такой пакет может оказаться порванным), упаковщики заменяют упаковку на новую. Для этого на склад днем поступают упаковочные материалы. Упаковка также выполняется после перефасовки для некоторых типов тары (например, для ведер).

2.5. Диаграммы взаимодействия

Диаграммы взаимодействия предназначены для моделирования динамических аспектов системы. Они показывают взаимодействие объектов, включающее набор объектов и их отношений, а также пересылаемые между объектами сообщения. К диаграммам взаимодействия относятся диаграммы последовательности (Sequence Diagram) и диаграммы сотрудничества (Collaboration Diagram). Эти диаграммы позволяют с разных точек зрения рассмотреть взаимодействие объектов в создаваемой системе.

Порядок построения диаграмм последовательности

1. Построение диаграммы последовательности целесообразно начинать с выделения из всей совокупности тех и только тех классов, объекты которых участвуют в моделируемом взаимодействии. После этого все объекты нанести на диаграмму с соблюдением некоторого порядка инициализации сообщений. Здесь необходимо установить какие объекты будут существовать постоянно, а какие временно (только на период выполнения требуемых действий).
2. После визуализации на диаграмме объектов можно приступить к спецификации сообщений. При этом следует учитывать те роли, которые играют сообщения в проектируемой системе. При необходимости уточнения этих ролей надо использовать их разновидности и стереотипы. Для уничтожения объектов, которые создаются на время выполнения своих действий, нужно предусмотреть явное сообщение.
3. Наиболее простые случаи ветвления процесса взаимодействия можно изобразить на одной диаграмме с использованием соответствующих графических примитивов. Однако следует помнить, что каждый альтернативный поток управления может существенно затруднить понимание построенной модели. Поэтому общим правилом является визуализация каждого потока управления на отдельной диаграмме последовательности. В этой ситуации такие отдельные диаграммы должны рассматриваться совместно как одна модель взаимодействия.
4. Дальнейшая детализация диаграммы последовательности связана с введением временных ограничений на выполнение отдельных действий в системе. Для простых асинхронных сообщений временные ограничения могут отсутствовать. Однако необходимость синхронизировать сложные потоки управления, как правило, требуют введения в модель таких ограничений.

Разработка диаграммы последовательности

На диаграмме последовательности изображаются объекты, которые непосредственно участвуют во взаимодействии и не показываются возможные статические ассоциации с другими объектами. Для диаграммы последовательности ключевым моментом является динамика взаимодействия объектов во времени.

Диаграмма последовательности имеет два измерения:

1) Одно – слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта, участвующего во взаимодействии. Графически каждый объект изображается прямоугольником и располагается в верхней части своей линии жизни.

2) Второе измерение – вертикальная временная ось, направленная сверху вниз. Начальному моменту времени соответствует самая верхняя часть диаграммы.

Взаимодействия объектов реализуются посредством сообщений, которые посылаются одними объектами другим. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения и также образуют порядок по времени своего возникновения. Масштаб на оси времени не указывается, поскольку

диаграмма последовательности моделирует лишь временную упорядоченность взаимодействий типа «раньше-позже».

Пример диаграммы последовательности для моделирования телефонного разговора представлен на рис. 21. В рамках данного примера диаграмма последовательности строится для вычислительного модуля задачи, показывая на каких этапах и какие объекты выполняют необходимые в проекте действия.

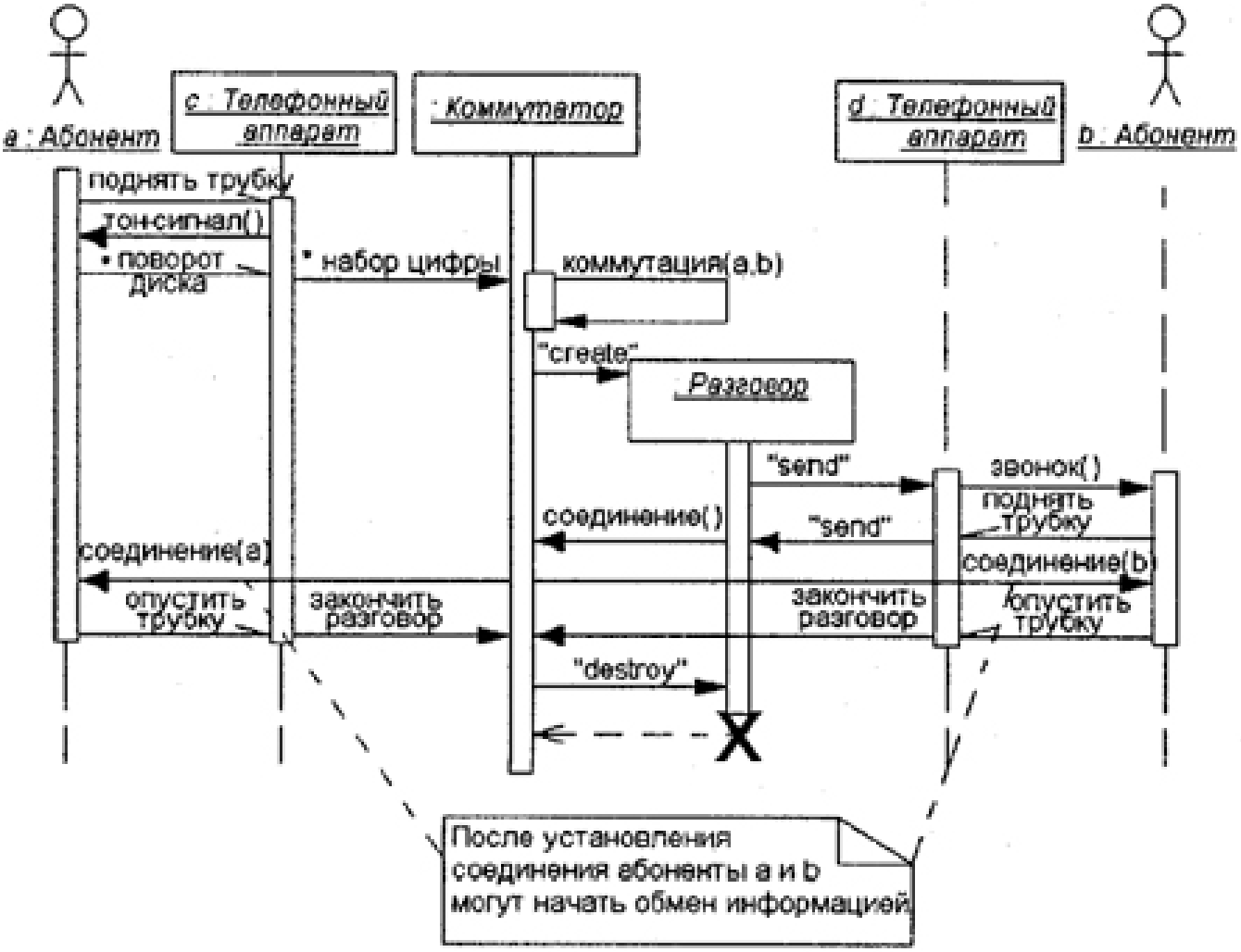


Рис. 21. Пример диаграммы последовательности для моделирования телефонного разговора

Требования к содержанию раздела

В подразделе 1 приводится описание функционирования программы в виде сценария или потока событий. При этом следует опираться на статическую функциональную структуру ПО в виде диаграммы требований UML, построенной в при выполнении лабораторной работы №3. Необходимо выделить основные абстракции, их атрибуты и поведение, и описать их в виде табл. 4 и 5. Также представить диаграмму классов с указанием лишь их имен (рис. 12, а).

В подразделе 2 отражается построение полной модели классов (рис. 12, в) с указанием атрибутов (с указанием типов данных) и операций (с указанием полных

сигнатур).

В подразделе 3 определяется класс, которых характеризуется наиболее нетривиальным поведением (наиболее частой сменой состояний). Как правило, это класс-сущность, отражающий одну из центральных сущностей предметной области, с которой выполняются действия по ходу автоматизируемого бизнес-процесса. Для данного класса строится диаграмма состояний UML и выполняется её описание по тексту. Диаграмма состояний должна иметь начальное и конечное состояние (конечных состояний может быть несколько).

В подразделе 4 отражается построение одной или нескольких диаграмм деятельности, которые должны покрывать все функциональные требования и, соответственно, все варианты использования, представленные на модели требований UML, построенной при выполнении лабораторной работы №3. При этом следует строго придерживаться принципов разветвления и слияния потоков функционирования, как это показано на рис. 19 и 20. Все диаграммы должны быть описаны по тексту подраздела.

В подразделе 5 отражается построение диаграмм взаимодействия UML. При этом необходимо построенными моделями последовательности покрыть все классы и особенности их взаимодействия с другими классами и действующими лицами, которые представлены на диаграмме требований UML, построенной при выполнении лабораторной работы №3. Для не менее чем одной диаграммы последовательности дополнительно построить диаграмму кооперации UML. Все диаграммы должны быть описаны по тексту подраздела.

Лабораторная работа №6

Тема: Проектирование базы данных программной системы.

Цель: изучение программных средств для разработки моделей информационной базы ПС, проработка методов нормализации отношений в БД, приобретение навыков применения CASE-средста ERwin для моделирования базы данных ПС.

Теоретические положения

6.1. CASE-средство ERwin для моделирования баз данных

Выбор средств проектирования программного обеспечения является одной из самых важных задач при разработке любого программного обеспечения. В настоящее время для проектирования различных систем повсеместно используют CASE-средства.

Термин CASE (Computer Aided Software Engineering) используется в настоящее время в весьма широком смысле. Первоначальное значение термина CASE, ограниченное вопросами автоматизации разработки только лишь ПО, в настоящее время приобрело новый смысл, охватывающий процесс разработки сложных ПС в целом. Теперь под термином CASE-средства понимаются программные средства, поддерживающие процессы создания и сопровождения ПС, включая анализ и формулировку требований, проектирование специального ПО (приложений) и баз данных, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом, а также другие процессы. Таким образом, CASE-средства вместе с системным ПО и техническими средствами образуют полную среду разработки ПС.

Потребности разработчиков и проектировщиков нередко выходят за границы возможностей обычных средств разработки. Поэтому CASE-средства необходимы, например, для осуществления логического моделирования данных, объектного моделирования, а иногда даже для обратного проектирования бизнес-процессов.

Таким образом, разработка ПС с использованием CASE-технологий является наиболее эффективным методом и имеет массу преимуществ, как для пользователей, так и для разработчиков.

Компания CA Technologies является разработчиком программных систем и решений управления ИТ-средой, который обладает опытом работы со всеми ИТ-средами, от больших ЭВМ и распределенных сред до виртуальных и облачных сред. Данная компания занимается разработкой CASE-средства ERwin Data Modeler, которое мы будем использовать при создании базы данных ПС.

ERwin Data Modeler (или просто ERwin) – CASE-средство для проектирования и документирования баз данных, которое позволяет создавать, документировать и сопровождать базы данных, хранилища и витрины данных. Модели данных помогают визуализировать структуру данных, обеспечивая эффективный процесс организации, управления и администрирования таких

аспектов деятельности предприятия, как уровень сложности данных, технологий баз данных и среды развертывания.

ERwin предназначен для всех компаний, разрабатывающих и использующих базы данных, для администраторов баз данных, системных аналитиков, проектировщиков баз данных, разработчиков, руководителей проектов. ERwin Data Modeler позволяет управлять данными в процессе корпоративных изменений, а также в условиях стремительно изменяющихся технологий.

ERwin позволяет наглядно отображать сложные структуры данных. Удобная в использовании графическая среда ERwin Data Modeler упрощает разработку базы данных и автоматизирует множество трудоемких задач, уменьшая сроки создания высококачественных и высокопроизводительных транзакционных баз данных и хранилищ данных.

ERwin имеет мощные средства визуализации модели, такие, как использование различных шрифтов, цветов и отображение модели на различных уровнях, например, на уровне описания сущности, на уровне первичных ключей сущности и т.д. Эти средства ERwin значительно помогают при презентации модели в кругу разработчиков системы или сторонним лицам.

Обычно разработка модели базы данных состоит из двух этапов: составление логической модели и создание на ее основе физической модели. ERwin полностью поддерживает такой процесс, он имеет два представления модели: логическое (logical) и физическое (physical). Таким образом, разработчик может строить логическую модель базы данных, не задумываясь над деталями физической реализации, т.е. уделяя основное внимание требованиям к информации и бизнес-процессам, которые будет поддерживать будущая база данных.

Далее при помощи ERwin разработаем логическую модель данных, и создадим на ее основе физическую модель.

6.2. Разработка логической модели данных

Логическая модель описывает понятия предметной области, их взаимосвязь, а также ограничения на данные, налагаемые предметной областью. Логическая модель данных является визуальным графическим представлением структур данных, их атрибутов и связей. Логическая модель представляет данные таким образом, чтобы они легко воспринимались бизнес-пользователями.

Логическая модель данных является начальным прототипом будущей базы данных. Она строится в терминах информационных единиц, но без привязки к конкретной СУБД.

Основными компонентами логической модели являются:

- сущности;
- атрибуты сущности;
- связи между сущностями.

Сущность моделирует структуру однотипных информационных объектов (документов, хранилищ данных, таблиц базы данных). В пределах модели данных сущность имеет уникальное имя, выраженное существительным.

Сущность является шаблоном, на основании которого создаются конкретные экземпляры сущности.

Сущность обладает следующими свойствами:

- каждая сущность имеет уникальное имя, и к одному и тому же имени должна применятся одинаковая интерпретация;
- сущность обладает одним или несколькими атрибутами, которые либо принадлежат сущности либо наследуются через связь;
- сущность обладает одним или несколькими атрибутами, которые однозначно идентифицируют каждый экземпляр сущности;
- каждая сущность может обладать любым количеством связей с другими сущностями модели.

Атрибут – любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности. Атрибут представляет тип характеристик или свойств, ассоциированных со множеством реальных или абстрактных объектов (людей, мест, событий, состояний, идей, пар предметов и т.д.). Экземпляр атрибута – это определенная характеристика отдельного элемента множества. Экземпляр атрибута определяется типом характеристики и ее значением, называемым значением атрибута. В ER-модели атрибуты ассоциируются с конкретными сущностями. Таким образом, экземпляр сущности должен обладать единственным определенным значением для ассоциированного атрибута.

Связь – поименованная ассоциация между двумя сущностями, значимая для рассматриваемой предметной области. Связь – это ассоциация между сущностями, при которой, как правило, каждый экземпляр одной сущности, называемой родительской сущностью, ассоциирован с произвольным (в том числе нулевым) количеством экземпляров второй сущности, называемой сущностью-потомком, а каждый экземпляр сущности-потомка ассоциирован в точности с одним экземпляром сущности-родителя.

На примере бизнес-процесса управления продажами книг произведем идентификацию сущностей ПС и связей между ними, приведенных в табл. 7.

Таблица 7. Идентификация сущностей (пример заполнения)

Документ	Сущность
Накладные	Накладные
Счет-фактура	Счета
Книги	Товары
Клиенты	Клиенты
Заказы	Заказы

Исходя из данных табл. 7, каждой сущности в соответствие ставится документ. На данном этапе получаем пять основных сущностей, которые далее, при разработке физической модели данных, разобьем на подсущности.

Далее в табл. 8 приведем имена, атрибуты и назначения сущностей. Сущности в табл. 8 разделим на оперативные и справочные. Справочные сущности – это сущности, информация о которых хранится в справочниках. Как правило, это неизменная и многократно используемая в течение длительного периода времени информация. Информация об оперативных сущностях может меняться для каждого случая, как по назначению, так и по количеству. При разработке модели определим сущности, их первичные и внешние ключи и атрибуты, а также связи между сущностями.

Таблица 8. Сущности логической модели данных

Сущность	Атрибуты	Описание
Справочные		
1 Товары	Код товара, название книги, жанр, страницы, год издания, код автора, фамилия, имя, отчество, код издательства, название издательства, город сокращенно, город полностью	Информация о книгах авторах и издательствах
2 Виды накладных	Код вида, вид накладной	Информация о видах накладных
3 Клиенты	Код клиента, Фамилия, Имя, Отчество, телефон	Информация о клиентах
Оперативные		
4 Заказы	Код заказа, *Код товара,* количество, сумма, код поставщика, имя поставщика, номер телефона, адрес	Информация по заказам и поставщикам
5 Накладные	Код накладной, номер накладной, дата создания, вид накладной, цена, сумма, *код счета, код заказа, код клиента, код товара,* код сотрудника, фамилия, имя, отчество	Информация о накладных и сотрудниках
6 Счета	Код счет-фактуры, дата создания, *код заказа, код товара, код клиента,* количество, цена	Информация о счетах

При помощи CASE-средства ERWin разработаем ER-диаграмму логической модели данных на основе табл. 8, представленную на рисунке 21.

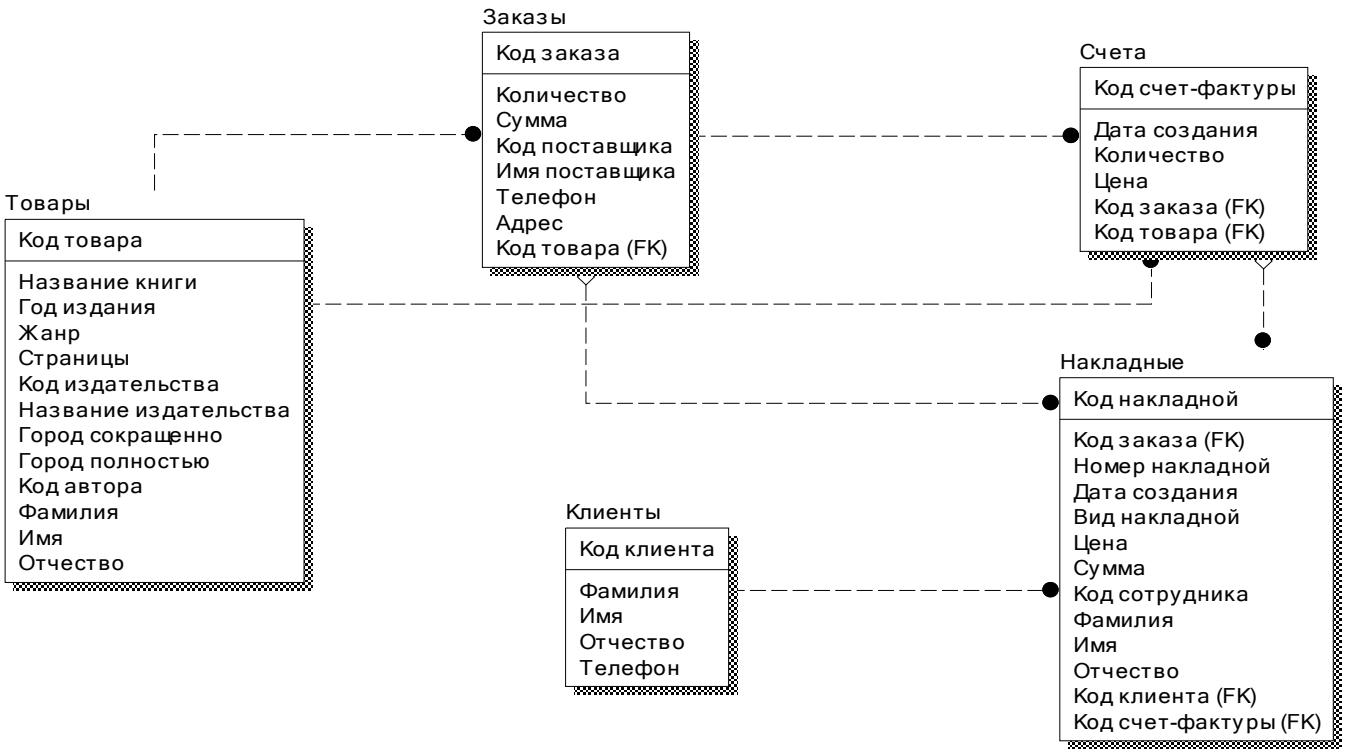
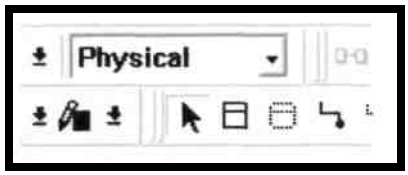


Рис. 22. ER-диаграмма логической модели данных

6.3. Разработка физической модели данных

Физическая модель баз данных определяет способы размещения данных в среде хранения и способы доступа к этим данным, которые поддерживаются на физическом уровне, является графическим представлением реально реализованной базы данных. В физической модели содержится информация обо всех объектах БД. Поскольку стандартов на объекты БД не существует (например, нет стандарта на типы данных), физическая модель зависит от конкретной реализации СУБД. Физическая база данных будет состоять из таблиц, столбцов и связей. Физическая модель зависит от платформы, выбранной для реализации, и требований к использованию данных. Компонентами физической модели данных являются таблицы, столбцы и отношения. Сущности логической модели, вероятно, станут таблицами в физической модели. Логические атрибуты станут столбцами. Логические отношения станут ограничениями целостности связей.

На основе логической модели данных разработаем при помощи CASE-средства ERWin ER-диаграмму физической модели данных. В ERWin необходимо выбрать тип модели:



Для этого выделим из сущностей подсущности для нормализации отношений. Отношения, разработанные на стадии формирования логической модели данных, преобразуются в таблицы, атрибуты становятся столбцами таблиц,

для ключевых атрибутов создаются уникальные индексы, домены преображаются в типы данных, принятые в конкретной СУБД.

Ограничения, имеющиеся в логической модели данных, реализуются различными средствами СУБД, например, при помощи индексов, декларативных ограничений целостности, триггеров, хранимых процедур. При этом опять-таки решения, принятые на уровне логического моделирования определяют некоторые границы, в пределах которых можно развивать физическую модель данных. Точно также, в пределах этих границ можно принимать различные решения. Например, отношения, содержащиеся в логической модели данных, должны быть преобразованы в таблицы, но для каждой таблицы можно дополнительно объявить различные индексы, повышающие скорость обращения к данным. Многое тут зависит от конкретной СУБД.

Из сущности Товары выделяем подсущности Авторы и Издательства; из сущности Заказы выделяем Поставщиков и Позиции заказа; из сущности Накладные выделяем Позиции, Виды накладных и Сотрудников; из сущности Счета выделяем Позиции счета; сущность Клиенты оставляем неизменной (**рис. 24**).

Средствами ERWin выполним генерацию SQL-кода для создания реляционной базы данных ПС. Алгоритм действий следующий:

- При помощи CASE-средства ER-Win открыть физическую модель и вызвать диалог генерации SQL-кода (меню Tools → Forward Engineer → Schema Generation…).
- Выбрать формат будущей БД (SQL Server, Interbase, Oracle, и т.д.). Лучше отключить генерацию триггеров, для чего в группе триггер поснимать все птички.
- Нажать на кнопку Preview….
- В появившемся диалоговом окне скопировать сгенерированный код в буфер обмена.

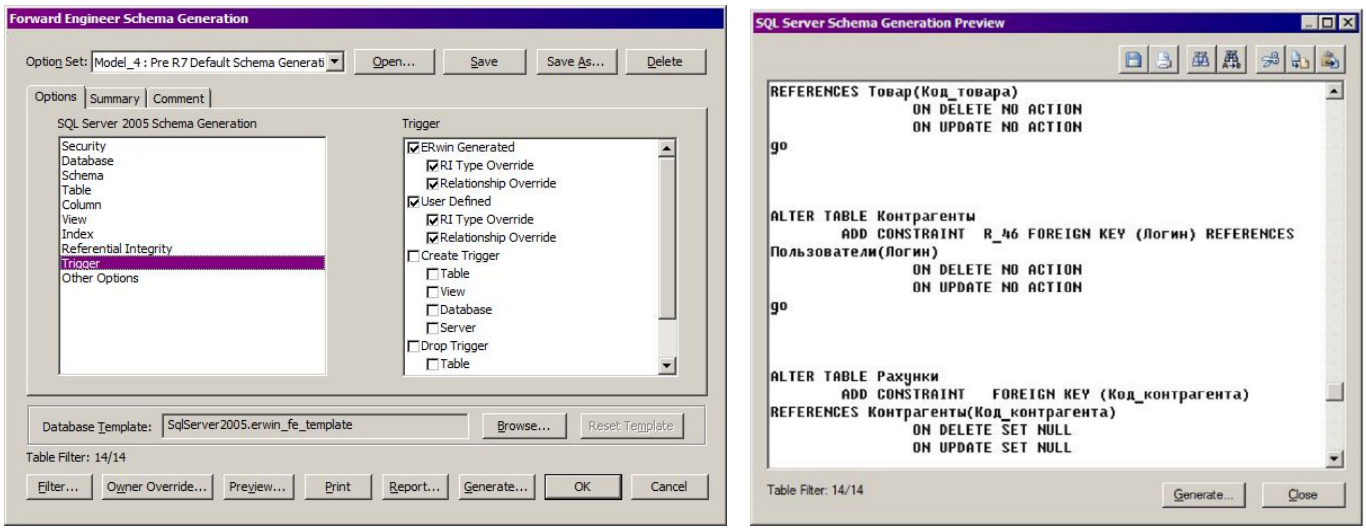


Рис. 23. Настройка и результаты процедуры генерации кода SQL

Формируется список разработанных таблиц и связей между таблицами БД, описание сущностей физической модели в формате табл. 8.

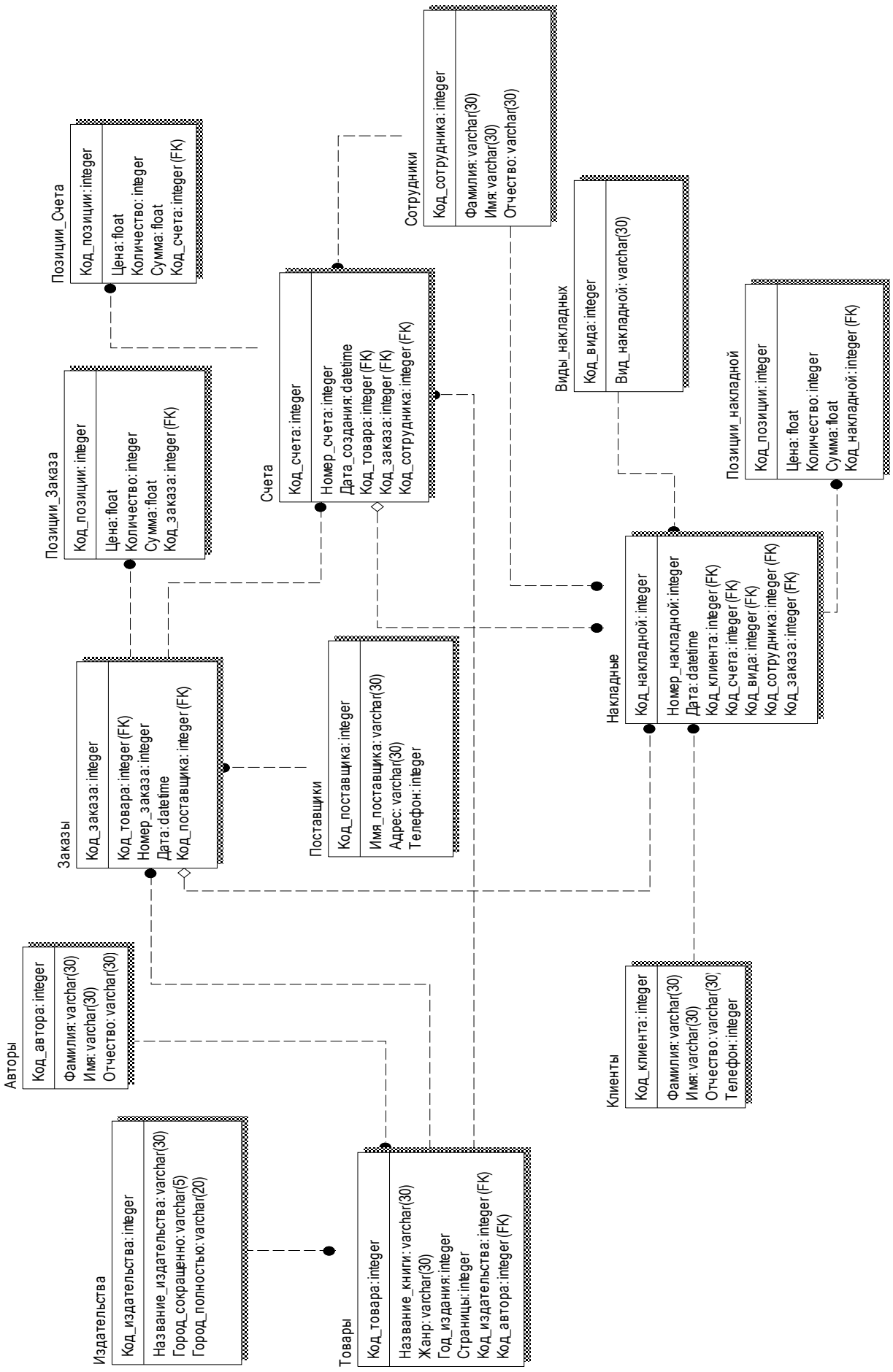


Рис. 24. ER-диаграмма физической модели данных

Задание

1. На основе модели классов UML, разработанной при выполнении лабораторной работы №5, произвести идентификацию сущностей информационной базы ПС и связей между ними.
2. При помощи CASE-средства ERWin разработать ER-диаграмму логической модели данных.
3. Провести нормализацию сущностей логической модели данных и разработать ER-диаграмму физической модели данных.
4. Средствами ERWin на основе физической модели данных выполнить генерацию SQL-кода для создания реляционной базы данных ПС.
5. В соответствии с требованиями технического задания, разработанного при выполнении лабораторной работы №3, провести обоснованный выбор СУБД.
6. В выбранной СУБД развернуть БД, доработать её структуру с учетом возможной нормализации отношений, а также доработать структуру таблиц с учетом ограничений на значения полей.
7. Выполнить описание таблиц БД в формате табл. 9 с указанием для всех полей типов данных выбранной СУБД. Столбцы «Условие на значение», «Значение по умолчанию» и «Примечание» заполнять при необходимости. Выполнить краткое описание таблиц БД и связей между ними.

Таблица 9. Структура таблицы _____

Имя поля	Тип данных	Размер, байт	Условие на значение*	Значение по умолчанию*	Примечание*

* Указанные столбцы не являются обязательными для заполнения

Требования к содержанию раздела

В подразделе 1 приводится анализ классов-сущностей из модели классов UML на предмет целесообразности хранения соответствующих данных в ПС и достаточности атрибутов этих классов для реализации автоматизированных функций ПС. Количество сущностей в модели зависит от предметной области. Классифицировать сущности на такие, которые отражают:

а) справочные, отражающие нормативно-справочную (условно-постоянную) информацию о предметной области;
б) оперативные, отражающие входную (текущую) информацию для решения текущих задач процесса информатизации.

В подразделе 2 строится логическая модель данных. Сущности логической модели данных следует разделить на оперативные и справочные. Имена, атрибуты и назначение сущностей логической модели данных привести в формате табл. 8. При разработке модели определить сущности, их первичные и внешние ключи и атрибуты, а также связи между сущностями.

Для моделирования данных использовать CASE-средство ERWin либо побочное ему инструментальное средство.

В подразделе 3 сущности логической модели данных рассматриваются как реляционные отношения и выполняется нормализация этих отношений. Таким образом осуществляется физическое моделирование данных, цель которого – таблицы в нормальных формах высшего, минимум, третьего (3НФ) порядка. При проведении нормализации следует стремиться к исключению аномалий модификации данных, к атомарности атрибутов, в первую очередь первичных ключей, и к минимизации дублирования данных.

По результатам физического моделирования данных осуществляется генерацию SQL-кода для создания реляционной базы данных ПС. Полученный SQL-код приводится в приложении к отчету.

В подразделе 3 отражается обоснованный выбор СУБД посредством сравнения двух-трех современных СУБД примерно равных по своим функциональным возможностям. Выбор СУБД должен быть обоснован масштабом проекта и особенностями решаемых в нем задач.

В этом же подразделе отражается развертывание БД на базе выбранной СУБД. **Необходимо представить доказательный результат развертывания, полученный средствами СУБД**, и выполнить его краткое описание. Также следует представить краткое описание таблиц БД в формате табл. 10 и описание всех связей между таблицами в формате табл. 11. Структуру всех таблиц БД в формате табл. 9, вынести в приложение к отчету.

Таблица 10. Список разработанных таблиц

№ п/п	Имя таблицы	Описание

Таблица 11. Связи между таблицами БД

Родительская таблица		Дочерняя таблица		Тип связи*
Название	Атрибут PK	Название	Атрибут FK	

* в качестве типа связи указывать «1:1», «1:M», «M:M»

Контрольные вопросы и упражнения

1. Что такое информационное обеспечение ПС?
2. Что такое информационная база ПС?
3. Опишите алгоритм создания информационной базы ПС?
4. Какие современные СУБД используются для создания информационной базы ПС?
5. Что такое нормализация отношений БД?
6. Чем отличается физическая модель данных от логической?
7. В чем различия между справочными таблицами и таблицами оперативных данных?

Лабораторная работа №7

Тема: Разработка программного обеспечения ПС

Цель: Приобретение навыков проектирования и разработки программной системы при помощи современных инструментальных средств.

Теоретические положения

7.1. Классификация ПО

Программное обеспечение (ПО) ПС – это комплекс компьютерных программ, обеспечивающих обработку и передачу данных, а также документация по их конфигурированию и применению. Включает в себя системное, служебное, инструментальное, прикладное и специальное ПО.

Системное ПО – это операционные системы, управляющие функционированием средств вычислительной техники, сетевого оборудования и прикладного ПО.

Служебное ПО – это программные средства, выполняющие функции по обслуживанию компьютерной техники и программ других классов: различные утилиты, средства диагностики, антивирусные программы, и т.п.

Инструментальное ПО – это совокупность программных средств, необходимых для проектирования и разработки ПС: CASE, СУБД, интегрированные среды разработки (IDE), интерпретаторы и трансляторы программ, и т.п.

Прикладное ПО – это совокупность программ, необходимых для обеспечения функционирования программ решения задач управления объектом: внешние библиотеки для прикладных программ, средства защиты данных, офисное ПО, графические пакеты, браузеры, а также иные программы необходимые для полноценного использования специального ПО.

При проектировании ПО проводится анализ и обоснованный выбор инструментального ПО (СУБД, IDE, CASE-средства и др.). Для анализа выбирается не менее 2-х современных систем (языков) программирования или инструментальных средств разработки (IDE), которые пригодны для разработки соответствующих программ. Для каждого средства указываются основные возможности и характерные особенности без подробного описания языка программирования, интегрированной среды и т.п. В результате анализа делается вывод, на основании чего выбирается средство разработки специального ПО.

По результатам разработки общесистемного ПО строится его структурная схема в виде древовидной блок-схемы либо организационной диаграммы (рис. 25).

К программному обеспечению, подлежащему разработке в процессе рабочего проектирования, относится категория специального ПО (СПО). СПО – это совокупность программ, непосредственно реализующих алгоритмы решения функциональных задач. СПО разрабатывается при проектировании ПС на базе математического обеспечения и представляет собой программную реализацию моделей, разработанных в при выполнении лабораторной работы №4 и 4.1. Зачастую включает в себя интерфейсные средства: web-страницы, формы, отчеты,

модули кода для реализации серверной части СПО, а также запросы, представления, курсоры, хранимые процедуры, макросы и другие модули программного кода. По результатам разработки специального ПО строится его структурная схема в виде диаграммы компонентов UML, либо в произвольной нотации.

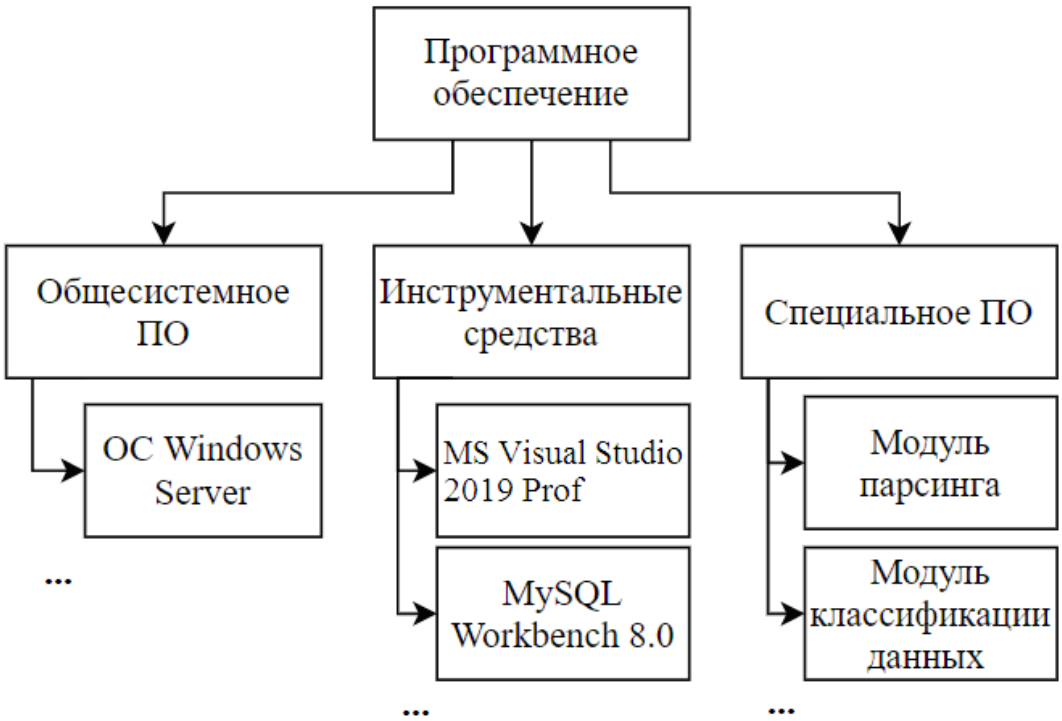


Рис. 25. Структурная схема общесистемного программного обеспечения (пример)

7.2. Диаграммы компонентов UML

Для визуализации статического аспекта физических компонентов ПО и их отношений, а также для специфицирования деталей этих компонентов в UML используются диаграммы компонентов (component diagram). Диаграмма компонентов позволяет определить состав программных компонентов, в роли которых может выступать исходный, бинарный и исполняемый код, и др., а также установить зависимости между ними.

При разработке диаграмм компонентов преследуются такие цели:

– спецификация общей структуры исходного кода системы;
– спецификация исполнимого варианта системы.

Для графического представления компонентов используются прямоугольники. Внутри прямоугольника записывается имя компонента и стереотип. Возможные типы компонентов показаны в табл. 12. Компоненты-файлы на схеме приводятся с указанием расширения.

При спецификации общей структуры исходного кода ПС необходимо учитывать специфику используемого языка программирования. В частности в Java рекомендуется отдельный класс описывать в отдельном файле, несмотря на то, что язык позволяет описывать несколько классов в одном файле и использовать механизм внутренних классов. Диаграмма компонентов, применяемая для рассматриваемой цели, изображена на рис. 26.

Таблица 12. Типы компонентов приложения

Стереотип	Тип компонента
«source»	файл исходного текста программы
«form»	файл интерфейсной формы или web-страницы
«executable»	исполняемый файл
«library»	статическая или динамическая библиотека
«document»	файл документации
«data»	файл данных
«table»	таблица базы данных
«view»	представление как объект базы данных
«file»	любой другой файл, кроме указанных выше в это таблице

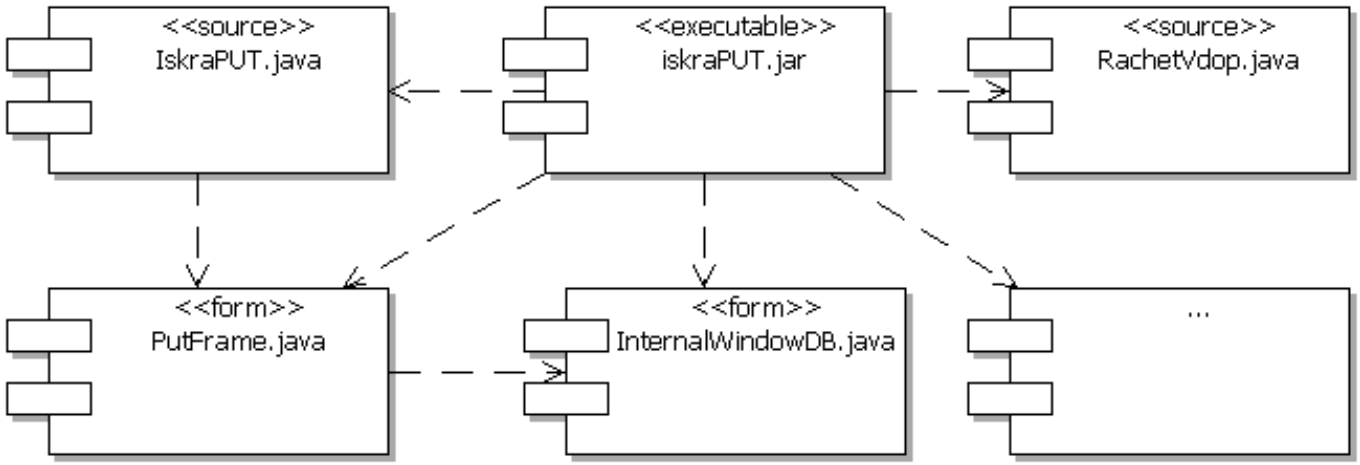


Рис. 26. Фрагмент диаграммы компонентов, специфицирующей структуру исходного кода

На диаграмме компонентов могут быть показаны детали структурной реализации. На рис. 27 представлены варианты структурной организации модуля исходного кода.

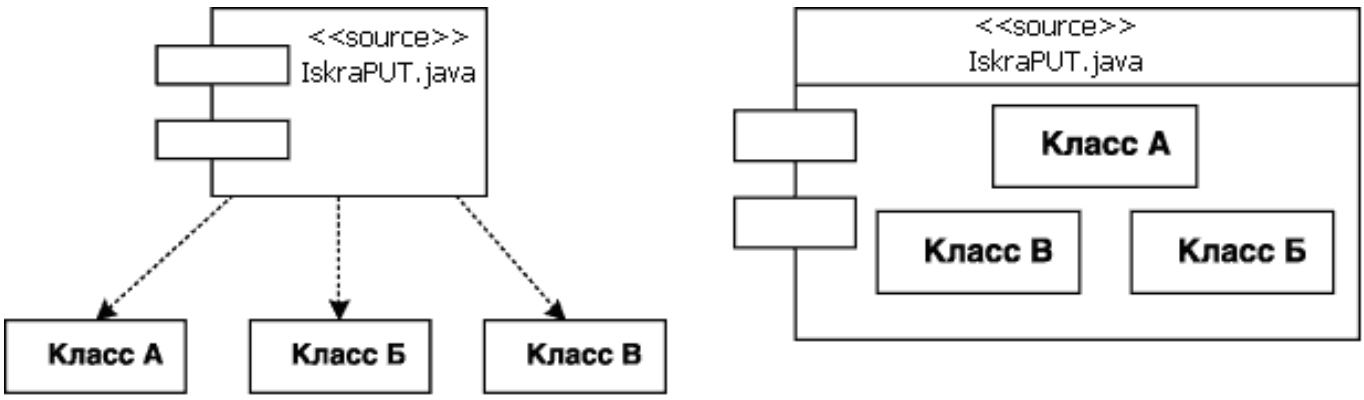


Рис. 27. Способы реализация классов компонентом

Зависимость между программными компонентами может отражать наличие в независимом компоненте описаний классов, которые используются в зависимом компоненте для создания соответствующих объектов. Зависимости могут связывать

компоненты и импортируемые этим компонентом интерфейсы, а также различные виды компонентов между собой. Так. например, изображенный на рис. 28 фрагмент диаграммы компонентов отражает отношение зависимости модуля main от импортируемого интерфейса IDialog, который, в свою очередь, реализуется компонентом с именем Library. При этом для второго компонента этот интерфейс является экспортируемым.

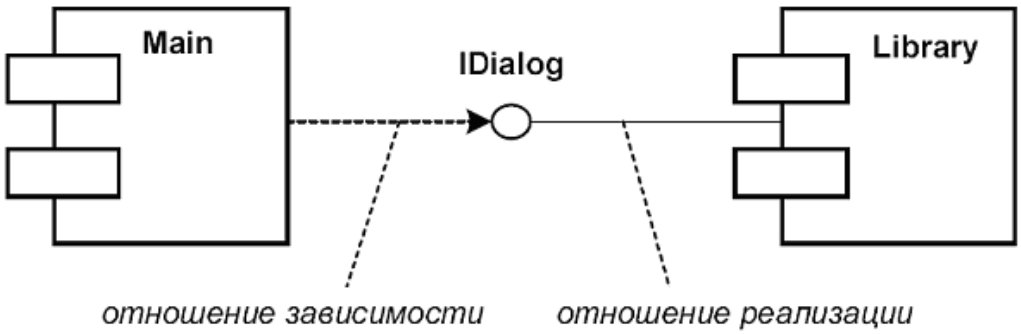


Рис. 28. Фрагмент диаграммы компонентов с отношениями зависимости и реализации

Диаграмма компонентов, отражающая взаимодействие компонентов разных типов нагляднее представляются с использованием *стереотипных* элементов (пиктограмм). На рис. 29. показан пример такой модели со стереотипными элементами типа «source», «library», «form», «document», «table».

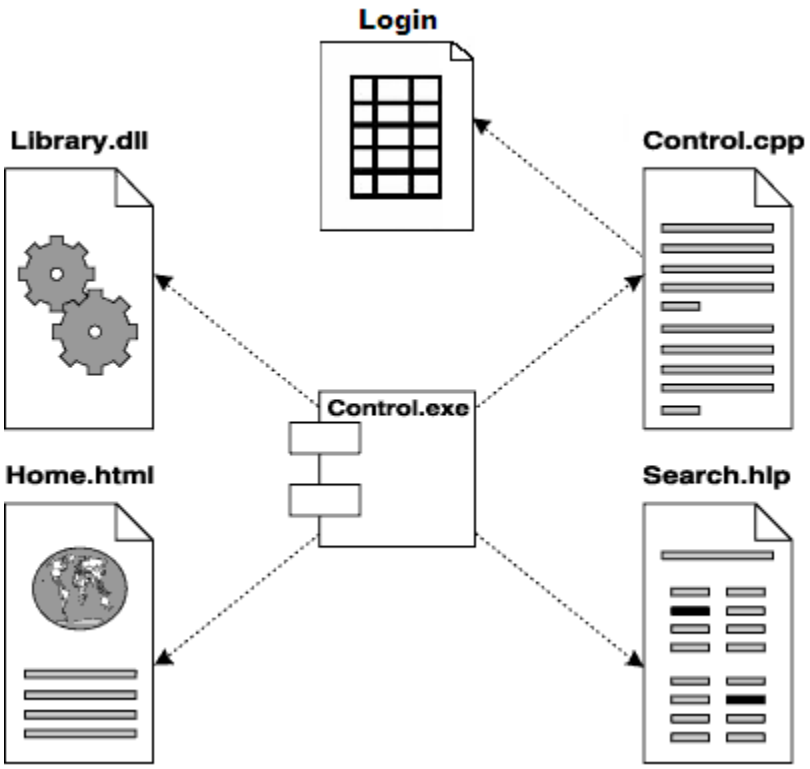


Рис. 29. Зависимость между компонентами приложения с использованием стереотипных элементов

В приложении Д на рис. Д.1 показан пример диаграммы компонентов ПС управления банкоматом.

7.3. Диаграммы развертывания UML

Диаграмма развертывания, или применения (deployment diagram), – это еще один из двух видов диаграмм, используемых при моделировании физических аспектов объектно-ориентированной системы. Такая диаграмма показывает конфигурацию узлов, где производится обработка информации, и то, какие компоненты размещены на каждом узле. Диаграммы развертывания используются для моделирования статического вида системы с точки зрения развертывания. В основном под этим понимается моделирование топологии аппаратных средств, на которых выполняется система. Диаграммы развертывания важны для визуализации, специфицирования и документирования встроенных, клиент-серверных и распределенных систем.

Для графического представления элементов диаграмм развертывания используются параллелепипеды. Внутри параллелепипеда записывается имя и стереотип. Возможные типы компонентов показаны в табл. 13.

Таблица 13. Типы элементов диаграмм развертывания UML

Стереотип	Тип элемента
«processor»	Ресурсоемкий узел
«device»	Нересурсоемкий узел (устройство)
«sensor»	Измерительное устройство (датчик)
«printer»	Печатающее устройство (принтер)
«net»	Сетевая передающая среда
«database»	База данных
«node»	Узел, не попадающий под тип, указанный выше в этой таблице

На рис. 30 показан фрагмент соединения типа «Клиент-Сервер» посредством локальной вычислительной сети.

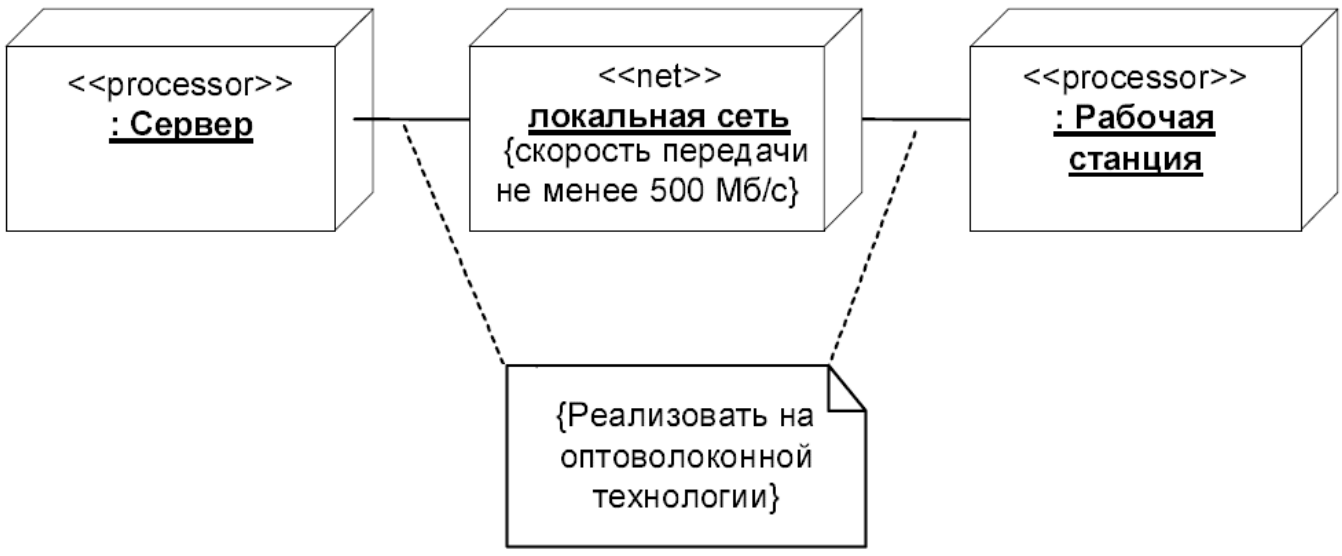


Рис. 30. Фрагмент диаграммы развертывания с соединениями между узлами

Кроме соединений на диаграмме развертывания могут присутствовать отношения зависимости между узлом и размещаемыми на нем компонентами. Подобный способ является альтернативой вложенному изображению компонентов внутри символа узла, что не всегда удобно, поскольку делает этот символ излишне объемным (в качестве примера см. удаленный терминал на рис. Г.2). Поэтому при большом количестве развернутых на узле компонентов соответствующую информацию можно представить в форме отношения зависимости, которая обозначается пунктирной линией (рис. 31).

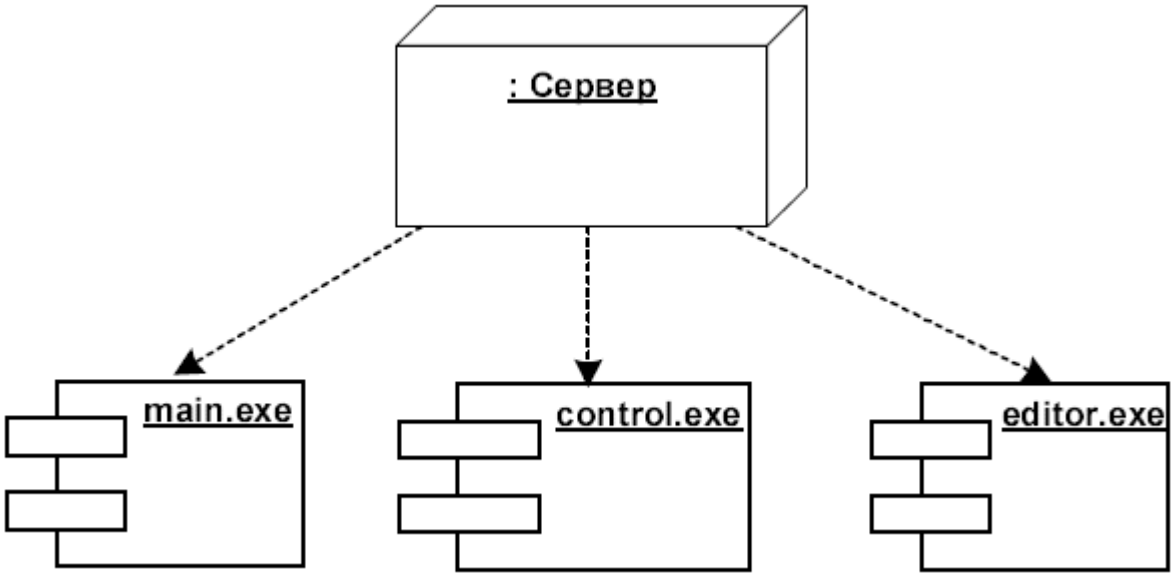


Рис. 31. Диаграмма развертывания с отношением зависимости между узлом и развернутыми на нем компонентами

В приложении Д на рис. Д.2 показана пример диаграммы развертывания ПС управления банкоматом.

Задание

1. В соответствии с требованиями технического задания, разработанного при выполнении лабораторной работы №3, провести обоснованный выбор средства разработки специального ПО. Разработать схему общесистемного ПО на подобие схемы, показанной на рис. 15.
2. В соответствии с требованиями технического задания, разработанного при выполнении лабораторной работы №3, а также проектными решениями, разработанными при выполнении лабораторных работ №4 и №6, разработать специальное программное обеспечение ПС.
3. Выполнить описание разработанных компонентов приложения в виде табл. 14. Типы компонентов указать согласно табл. 12. Имена компонентов-файлов привести с указанием расширения.

Таблица 14. Перечень разработанных компонентов приложения

№	Имя	Тип	Описание

4. Построить структурную схему разработанного приложения в виде диаграммы компонентов UML, выражающую взаимодействие его компонентов с компонентами БД в процессе функционирования приложения.
5. Запустить приложение на выполнение. Убедиться в соответствии результатов выполнения приложения требованиям, установленным в техническом задании. При обнаружении логических ошибок задокументировать их и устранить.
6. Представить экранные формы компонентов приложения, в том числе отчетов.
7. Проанализировать код приложения по критерию сложности. В качестве критерия сложности использовать:
   - число модулей (классов) приложения;
   - суммарное число переменных подпрограмм (методов классов), включая их формальные параметры;
   - суммарное количество операторов подпрограмм (методов классов);
   - глубину вложенности структурных операторов ветвления и повторения;
   - глубину наследования классов.
8. Выполнить описание физических элементов ПС в виде табл. 15. Типы элементов указать согласно табл. 13.

Таблица 15. Перечень узлов программной системы

№	Имя	Тип	Описание

9. Построить диаграмму развертывания UML, выражающую зависимости между узлами ПС и развернутыми на них компонентами из табл. 14.

Требования к содержанию отчета

В подразделе 1 кратко документируется разработанное специальное ПО. Выполняется описание разработанных модулей и компонентов, как по тексту, так и в виде таблиц 14 и 15. Листинг кода может быть представлен в приложении.

В подразделе 2 представляется диаграмма компонентов и выполняется её краткое описание. В описании должны раскрываться существенные детали реализации и взаимодействия компонентов (например, технологии реализации или обмен данными и др.). При этом следует избегать использования стандартного стереотипа «file», а максимально дифференцировать компоненты по типам согласно табл. 12.

В подразделе 3 представляется диаграмма развертывания и выполняется её краткое описание. В описании должны раскрываться существенные детали развертывания и взаимодействия узлов (например, важные с точки зрения понимания схемы технические характеристики, обмен данными между узлами, отношения узлов с компонентами и др.). При этом следует избегать использования стандартного стереотипа «node», а максимально дифференцировать компоненты по типам согласно табл. 13. Также следует различать на диаграмме узлы и реализованные на их базе компоненты, как это показано на рис. 31.

Контрольные вопросы и упражнения

1. Что такое программное обеспечение ПС?
2. Каковы отличия между сервисными программами и утилитами?
3. Какие типы программных средств относятся к специальному ПО.
4. Как происходит разработка программного кода ПС?
5. Что такое компиляция и построение программы?
6. Как в интегрированной среде программирования осуществляется обнаружение синтаксических ошибок?
7. Какова связь компонентов ПО с компонентами ИО разрабатываемой ПС?
8. Что такое специальное ПО?
9. Какие бывают типы компонентов специального ПО?
10. Каково назначение диаграммы компонентов UML?
11. Какие бывают типы физических элементов ПС?
12. Каково назначение диаграммы развертывания UML?
13. Определить сложность заданного фрагмента программного кода.
14. Предложить способы оценивания производительности и эффективности программного обеспечения на базе данного кода.

Лабораторная работа №8

Тема: Тестирование ПС

Цель: Освоение методики сценарного и интеграционного тестирования ПО, в том числе разработанной программной системы в ручном режиме.

Теоретические положения

8.1. Тестирование, как способ контроля качества ПО.

Тестирование – это проверка соответствия между реальным и ожидаемым поведением программы в специально заданных условиях. Исходной информацией для тестирования является сведения о том, как программная система должна себя вести, то есть требования к ней или к ее отдельной части. Наиболее распространенным способом тестирования программ является метод *черного ящика*. При этом реализация системы недоступна тестировщикам, и тестируется только ее интерфейс посредством пользовательского воздействия. Противоположностью данному методу является тестирование методом *белого ящика*, когда исходный код доступен тестировщикам и используется в качестве источника информации о тестируемой системе. В этом случае на основе требований к системе помимо ее реализации создается (программируется) ее тестовая модель.

По способу реализации тесты могут быть ручными и автоматическими.

Ручной тест – это последовательность действий тестировщика (или разработчика), которую он может воспроизвести и при этом ошибка произойдет. Как правило, в средствах контроля ошибок такие последовательности действий содержатся в некоторой форме описания ошибки.

Автоматический тест – это некоторая программа, которая воздействует на систему и проверяет то или иное ее свойство (функциональное требование).

8.2. Критерии тестирования

В любом случае качество тестирования может оказаться неудовлетворительным – ошибки находятся редко или вообще не находятся ввиду того, что количество возможных состояний программы очень велико, и тестирование не в состоянии покрыть их все.

На практике в реальных проектах, определяют критерии тестирования, которые определяют тот уровень качества, который необходимо обеспечить в данном проекте. Хорошее качество стоит дорого и очевидно, что разное ПО имеет разное качество, например, система управления химическим реактором и текстовый редактор. В силу ограниченности ресурсов на тестирование также целесообразно определить те аспекты ПО, которые наиболее важны как для общей работоспособности системы, так и для удовлетворения заказчика.

При тестировании на некоторых (но не на всех возможных) входных данных применяют принцип факторизации – множество всех возможных входных значений разбивается на значимые с точки зрения тестирования классы и "прогоняются" тесты не на всех возможных входных значениях, но берут по одному набору значений из каждого класса. Например, тестируют некоторую функцию системы на ее граничные значения – очень большие значения параметров, очень маленькие и т.п. Часто факторизацию удобно делать, исходя из требований к данной функции. Также бывает полезно посмотреть на ее реализацию и протестировать ее по разным логическим веткам (порождаемым, например, условными операторами).

8.3. Виды тестирования

Модульное тестирование – тестируется отдельный модуль, в отрыве от остальной системы. Самый распространенный случай применения – тестирования модуля самим разработчиком, проверка того, что отдельные модули, классы, методы выполняют действительно то, что от них ожидается. Различные среды разработки широко поддерживают средства модульного тестирования. Созданные разработчиком модульные тесты часто включаются в пакет регрессионных тестов и таким образом, могут запускаться многократно.

Интеграционное тестирование – два и более компонентов тестируются на совместимость. Это очень важный вид тестирования, поскольку разные компоненты могут создаваться разными людьми, в разное время, на разных технологиях. Этот вид тестирования должен применяться самими программистами, чтобы, по крайней мере, удостовериться, что все компоненты программы совместимы в первом приближении. Далее тонкости интеграции могут исследовать тестировщики.

Необходимо отметить, что такого рода "ошибки на стыках" непросто обнаруживать и устранять. Во время разработки все компоненты все вместе не готовы, интеграция откладывается, а в конце обнаруживаются трудные ошибки (в том смысле, что их устранение требует существенной работы: реструктуризации, рефакторинга). Здесь выходом является ранняя интеграция системы и в дальнейшем использование практики постоянной интеграции.

Сценарное тестирование – системный подход к тестированию ПО по предварительно написанным и задокументированным тест-кейсам (сценариям) – последовательностей действий пользователя в приложении.

Системное тестирование – это тестирование всей системы в целом, как правило, через ее пользовательский интерфейс. При этом могут использоваться как ручные, так и автоматические тесты с акцентом на том, как ПО выглядит и работает в целом, удобно ли оно, удовлетворяет ли она ожиданиям заказчика. При этом могут открываться различные дефекты, такие как неудобство в использовании тех или иных функций, забытые или "скудно" понятые требования.

Регрессионное тестирование – тестирование системы в процессе ее разработки и сопровождение на не регресс. То есть проверяется, что изменения системы не ухудшили уже существующей функциональности. Для этого создаются пакеты регрессионных тестов, которые запускаются с определенной

периодичностью – например, в пакетном режиме, связанные с процедурой постоянной интеграции.

Нагрузочное тестирование – тестирование системы на корректную работу с большими объемами данных. Например, проверка баз данных на корректную обработку большого (предельного) объема записей, исследование поведение серверного ПО при большом количестве клиентских соединений, эксперименты с предельным трафиком для сетевых и телекоммуникационных систем, одновременное открытие большого числа файлов, проектов и т.д.

Стрессовое тестирование – тестирование системы на устойчивость к непредвиденным ситуациям. Этот вид тестирования нужен далеко не для каждой системы, так как подразумевает высокую планку качества.

Приемочное тестирование – тестирование, выполняемое при приемке системы заказчиком. Более того, различные стандарты часто включают в себя наборы приемочных тестов.

8.4. Сценарное тестирование

Сценарное тестирование предполагает разделение задачи на этапы (подготовка, выполнение, завершение и т.д.) и последовательное выполнение этих этапов.

Особенности этого подхода:

- четкое понимание того, какие функции покрыты тестами;
- уверенность, что все задокументированные тест-кейсы будут пройдены в срок;
- быстрое и легкое подключение нового специалиста на проект благодаря наличию подробных сценариев.

Пример. При тестировании почты, сценарное тестирование может выглядеть так:

- перейти на стартовую страницу;
- нажать на кнопку «Новый»;
- в поле «Кому» написать j.cannegieter@squerist.nl;
- нажать на кнопку «Файл»;
- выбрать документ «Тест1»;
- в поле «Тема» написать «Тестовое вложение <дата>»;
- открыть почтовый ящик j.cannegieter;
- проверить, доставлено ли сообщение и вложение;

Для этого же примера сценарий общего (исследовательского) тестирования может выглядеть так:

- создать и отправить письмо с вложением одному получателю.
- проверить, доставлено ли сообщение и вложение.

8.5. Интеграционное тестирование

Интеграционное тестирование предназначено для проверки связи между компонентами, а также взаимодействия с различными частями системы (операционной системой, оборудованием либо связи между разными системами).

Уровни интеграционного тестирования:

1. Компонентный интеграционный уровень (*Component Integration testing*). Проверяется взаимодействие между компонентами системы после проведения компонентного (модульного) тестирования.
2. Системный интеграционный уровень (*System Integration Testing*), также известное как Тестирование взаимодействия (*Interoperability Testing*). Проверяется взаимодействие системы с другими системами после проведения системного тестирования.

Подходы к интеграционному тестированию:

1. Снизу вверх (*Bottom Up Integration*). Все низкоуровневые модули, процедуры или функции собираются воедино и затем тестируются. После чего собирается следующий уровень модулей для проведения интеграционного тестирования.
2. Сверху вниз (*Top Down Integration*). Вначале тестируются все высокоуровневые модули, и постепенно один за другим добавляются низкоуровневые. Все модули более низкого уровня симулируются заглушками с аналогичной функциональностью, затем по мере готовности они заменяются реальными активными компонентами.
3. Большой взрыв (*"Big Bang" Integration*). Все или практически все разработанные модули собираются вместе в виде законченной системы или ее основной части, и затем проводится интеграционное тестирование. Такой подход очень хорош для сохранения времени. Однако если тест кейсы и их результаты записаны не верно, то сам процесс интеграции сильно осложнится, что станет преградой для команды тестирования при достижении основной цели интеграционного тестирования

Например. Установлено, что данные по продажам из портала передаются в аналитическую систему. Тогда результатом интеграционного тестирования является решение о том, что необходимо дополнительно проверить, что:

а) продажи отправлены в корректном формате;

б) данные по продажам получены аналитической системой в корректном формате;

в) потери данных не произошло.

Задание

Выполнить системное пользовательское тестирование работоспособности ПС, разработанную при выполнении лабораторной работы №7, посредством воздействия на её интерфейсную часть. Для этого необходимо:

1. Согласно функциональным требованиям к ПС, предъявленным при выполнении лабораторной работы №3, разработать несколько сценариев тестирования ПС.

2. Реализовать сценарии и задокументировать полученные результаты тестирования.

3. По согласованию с преподавателем разработать сценарий тестирования стороннего программного продукта согласно данным, представленным в приложении Е. Оформить отчет по выполненному сценарному тестированию.

4. Выполнить интеграционное тестирование на компонентном уровне разработанных при выполнении лабораторной работы №7 автоматизированных функций программного продукта, используя подход *Bottom Up Integration.*

5. Выполнить нагрузочное тестирование программы и оценить эффективность разработанных при выполнении лабораторной работы №7 автоматизированных функций ПС, запросов к БД. В качестве критерия эффективности использовать время выполнения функции. Предварительно подготовить 5 массивов исходных данных на 10, 50, 100, 500 и 1000 записей. Построить графики зависимости времени вычислений от объема исходных данных.

Требования к содержанию отчета.

В подразделе 1 приводятся все разработанные сценарии для тестирования ПС, а также результаты сценарного тестирования посредством проверки работоспособности программы. В идеале разработанные сценарии должны соответствовать автоматизированным функциям ПС и инициировать срабатывание всех команд меню, ссылок и других пользовательских инструментов, предусмотренных требованиями.

В подразделе 2 документируются обнаруженные логические ошибки (при наличии таковых). Для каждой логической ошибки:

- описать ее сущность;
- представить тестовую последовательность манипуляций для ее обнаружения (как правило, это ссылка на сценарий и подмножество шагов полной тестовой последовательности в сценарии);
- представить форму ее проявления в результате применения теста;
- определить форму ее выражения в коде;
- описать способ ее устранения.

Подраздел 3 формируется в случае выдачи преподавателем дополнительного задания из приложения Е. Необходимо отразить сценарное тестирование некоторого интернет-магазина. Для заданных задач для сценарного тестирования представить разработанные сценарии и результаты их реализации аналогично содержанию подразделов 1 и 2. В силу специфики тестируемого продукта тестирование проводить методом черного ящика. Поэтому форму выражения ошибки в коде и способ её устранения в данном случае приводить не нужно.

В подразделе 4 необходимо описать сценарии тестирования работоспособности отдельных модулей программного продукта. Выполнить эти сценарии. Затем выполнить аналогичное тестирование на более высоком уровне, объединяющем работу нескольких модулей.

Наивысший уровень – полный цикл работы всего программного продукта, проверка которого осуществляется посредством системного тестирования.

Результаты интеграционного тестирования на всех уровнях представить аналогично подразделу 2.

В подразделе 5 необходимо представить технические характеристики машины, используемой для анализа производительности программы. Привести определенную последовательность действий пользователя (наподобие последовательности проверочных действий в сценарии, как в подразделе 1). Привести временн***ы***е оценки проверок для каждой конфигурации и кратко их прокомментировать. Привести временн***ы***е оценки нагрузочных проверок в виде сводной таблицы и графика на каждом файле для каждого метода. Прокомментировать результаты выполненного анализа.

Контрольные вопросы и упражнения

1. Перечислите и охарактеризуйте методы обеспечения качества программного обеспечения.
2. Что такое тестирование компьютерной программы?
3. Что такое ожидаемое поведение программы?
4. Какие виды тестов и тестирования программного обеспечения применимы к разработанному программному приложению?
5. Что такое производительность программного обеспечения?
6. Укажите методы и способы оценивания производительности программного обеспечения.
7. В чем заключается идентификация ошибок в программе?
8. Какими способами можно обнаружить логически ошибки в коде?
9. Перечислите показатели, по которым выполняется анализ кода.
10. Что понимается под эффективностью программного обеспечения?
11. Что понимается под сложностью программного обеспечения?
12. Каковы могут быть формы представления результатов оценивания эффективности программного обеспечения?
13. От чего зависит эффективность программного обеспечения?
14. Что такое модульное (компонентное) тестирование?
15. Укажите цели разработки сценариев тестирования.
16. В чем разница между тест кейсом и тестовым сценарием?
17. Перечислите преимущества использования сценариев при тестировании программных продуктов.

Лабораторная работа №9

Тема: Разработка баг-репорта ПС

Цель: Приобретение навыков выявления дефектов (багов), определения их характеристик, построение баг-репорта.

Теоретические положения

9.1. Структура баг-репорта

Баг-репорт (*bug report*) – это технический документ, описывающий ситуацию или последовательность действий, приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата. Шаблон баг-репорта приведен на рис. 32. Его составляет тестировщик, чтобы разработчикам было понятно, что работает неправильно, насколько дефект критичен и что нужно исправить. Баг-репорты – часть программного процесса.

Шапка	
Короткое описание (Summary)	Короткое описание проблемы, явно указывающее на причину и тип ошибочной ситуации.
Проект (Project)	Название тестируемого проекта
Компонент приложения (Component)	Название части или функции тестируемого продукта
Номер версии (Version)	Версия на которой была найдена ошибка
Серьезность (Severity)	Наиболее распространена пятиуровневая система градации серьезности дефекта: • S_1 Блокирующий (Blocker) • S_2 Критический (Critical) • S_3 Значительный (Major) • S_4 Незначительный (Minor) • S_5 Тривиальный (Trivial)
Приоритет (Priority)	Приоритет дефекта: • P_1 Высокий (High) • P_2 Средний (Medium) • P_3 Низкий (Low)
Статус (Status)	Статус бага. Зависит от используемой процедуры и **жизненного цикла бага** (bug workflow and life cycle)
Автор (Author)	Создатель баг репорта
Назначен на (Assigned To)	Имя сотрудника, назначенного на решение проблемы
Окружение	
ОС / Сервис Пак и т.д. / Браузера + версия / ...	Информация об окружении, на котором был найден баг: операционная система, сервис пак, для WEB-тестирования – имя и версия браузера и т.д.
...	
Описание	
Шаги воспроизведения (Steps to Reproduce)	Шаги, по которым можно легко воспроизвести ситуацию, приведшую к ошибке.
Фактический Результат (Result)	Результат, полученный после прохождения шагов к воспроизведению
Ожидаемый результат (Expected Result)	Ожидаемый правильный результат
Дополнения	
Прикрепленный файл (Attachment)	Файл с логами, скриншот или любой другой документ, который может помочь прояснить причину ошибки или указать на способ решения проблемы

Рис. 32. Шаблон баг-репорта

9.2. Основные характеристики и виды дефектов тестируемого ПО

Градация Серьезности дефекта (Severity)

Рассмотрим градации дефектов (багов) ПО.

S1 Блокирующая (Blocker). Блокирующая ошибка, приводящая приложение в нерабочее состояние, в результате которого дальнейшая работа с тестируемой системой или ее ключевыми функциями становится невозможна. Решение проблемы необходимо для дальнейшего функционирования системы.

S2 Критическая (Critical). Критическая ошибка, неправильно работающая ключевая бизнес-логика, дыра в системе безопасности, проблема, приведшая к временному падению сервера или приводящая в нерабочее состояние некоторую часть системы, без возможности решения проблемы, используя другие входные точки. Решение проблемы необходимо для дальнейшей работы с ключевыми функциями тестируемой системой.

S3 Значительная (Major). Значительная ошибка, часть основной бизнес-логики работает некорректно. Ошибка не критична или есть возможность для работы с тестируемой функцией, используя другие входные точки.

S4 Незначительная (Minor). Незначительная ошибка, не нарушающая бизнес-логику тестируемой части приложения, очевидная проблема пользовательского интерфейса.

S5 Тривиальная (Trivial). Тривиальная ошибка, не касающаяся бизнес-логики приложения, плохо воспроизводимая проблема, малозаметная посредствам пользовательского интерфейса, проблема сторонних библиотек или сервисов, проблема, не оказывающая никакого влияния на общее качество продукта.

Градация Приоритета дефекта (Priority)

P1 Высокий (High). Ошибка должна быть исправлена как можно быстрее, т.к. ее наличие является критической для проекта.

P2 Средний (Medium). Ошибка должна быть исправлена, ее наличие не является критичной, но требует обязательного решения.

P3 Низкий (Low). Ошибка должна быть исправлена, ее наличие не является критичной, и не требует срочного решения.

Порядок исправления ошибок по их приоритетам: High -> Medium -> Low.

9.3. Требования к количеству открытых дефектов

Наличие открытых дефектов P1, P2 и S1, S2, считается неприемлемым для проекта. Все подобные ситуации требуют срочного решения и идут под контроль к менеджерам проекта. Наличие строго ограниченного количества открытых ошибок P3 и S3, S4, S5 не является критичным для проекта и допускается в выдаваемом приложении. Количество же открытых ошибок зависит от размера проекта и установленных критериев качества.

Все требования к открытым ошибкам оговариваются и документируются на этапе принятия решения о качестве разрабатываемого продукта.

Баг-репорт – это технический документ. Поэтому язык описания проблемы должен быть техническим. Должна использоваться правильная терминология при использовании названий элементов пользовательского интерфейса (editbox, listbox, combobox, link, text area, button, menu, popup menu, title bar, system tray и т.д.), действий пользователя (click link, press the button, select menu item и т.д.) и полученных результатах (window is opened, error message is displayed, system crashed и т.д.).

9.4. Требования к обязательным полям баг-репорта

Обязательными полями баг-репорта являются: короткое описание (Bug Summary), серьезность (Severity), шаги к воспроизведению (Steps to reproduce), результат (Actual Result), ожидаемый результат (Expected Result). Ниже приведены требования и примеры по заполнению этих полей.

Короткое описание. Название говорит само за себя. В одном предложении надо уместить смысл всего баг-репорта, а именно: коротко и ясно, используя правильную терминологию сказать, что и где не работает. Например:

1. Приложение зависает, при попытке сохранения текстового файла размером больше 50Мб.
2. Данные на форме "Профайл" не сохраняются после нажатия кнопки "Сохранить".

Также применяется Принцип "Где? Что? Когда?", который заключается в составлении предложения, в котором факты дефекта изложены в следующей последовательности:

- Где? В каком месте интерфейса пользователя или архитектуры программного продукта находится проблема. Начинать предложение следует с существительного, а не предлога.
- Что? Что происходит или не происходит согласно спецификации или представлению о нормальной работе программного продукта. При этом следует указывать на наличие или отсутствие объекта проблемы, а не на его содержание (его указывают в описании). Если содержание проблемы варьируется, все известные варианты указываются в описании.
- Когда? В какой момент работы программного продукта, по наступлению какого события или при каких условиях проблема проявляется.

В такой последовательности незнакомые дефекты удобнее сортировать по summary (ведь, скорее всего, именно среди дефектов других инженеров будет производиться поиск дубликатов).

Серьезность. Если проблема найдена в ключевой функциональности приложения и после ее возникновения приложение становится полностью недоступно, и дальнейшая работа с ним невозможна, то она блокирующая. Обычно все блокирующие проблемы находятся во время первичной проверки новой версии продукта (Build Verification Test, Smoke Test), т.к. их наличие не позволяет полноценно проводить тестирование. Если же тестирование может быть продолжено, то серьезность данного дефекта будет критическая.

Шаги к воспроизведению / Результат / Ожидаемый результат. Важно четко описать все шаги, с упоминаем всех вводимых данных (имени пользователя, данных для заполнения формы) и промежуточных результатов. Пример такого описания показан на рис. 33.

Шаги к воспроизведению:
1. Войдите в системы: Пользователь Тестер1, пароль xxxXXX
--> Вход в систему осуществлен
2. Кликните линк «Профайл»
--> Страница «Профайл» открылась
3. Введите Новое имя пользователя: Тестер2
4. Нажмите кнопку «Сохранить»

Результат:
На экране появилась ошибка. Новое имя пользователя не было сохранено

Ожидаемый результат:
Страница профайл перегрузилась. Новое значение имени пользователя сохранено.

Рис. 33. Описание дефекта

9.5. Основные ошибки при написании баг-репортов

1. Недостаточность предоставленных данных. Не всегда одна и таже проблема проявляется при всех вводимых значениях и под любым вошедшим в систему пользователем, поэтому настоятельно рекомендуется вносить все необходимые данные в баг-репорт.
2. Определение серьезности. Часто происходит либо завышение, либо занижение серьезности дефекта, что может привести к неправильной очередности при решении проблемы.
3. Язык описания. Часто при описании проблемы используются неправильная терминология или сложные речевые обороты, которые могут ввести в заблуждение человека, ответственного за решение проблемы.
4. Отсутствие ожидаемого результата. В случаях, если вы не указали, что же должно быть требуемым поведением системы, вы тратите время разработчика, на поиск данной информации, тем самым замедляете исправления дефекта. Вы должны указать пункт в требованиях, написанный тест-кейс или же ваше личное мнение, если эта ситуация не была документирована.

Задание

1. Выполнить поиск дефектов разработанного при выполнении лабораторной работы №7 программного продукта, указать их характеристики и в соответствии с предложенной выше структурой баг-репорт. Баг-репорт включается в отчет о выполнении лабораторной работы №10.

2. По согласованию с преподавателем рассмотреть сайты с целью поиска дефектов: https://www.orange-elephant.ru/, https://flowwow.com/krasnodar/, https://sadovod-opt.com/ и др. Выбрать сайт некоторого интернет-магазина или аналогичного ресурса. Оформить соответствующий баг-репорт.

Контрольные вопросы и упражнения

1. Перечислите основные три цели тестирования программного обеспечения.
2. В чем отличия методик тестирования белого и черного ящиков?
3. Укажите уровни тестирования ПО.
4. Перечислите варианты серьезности дефекта.
5. Укажите обязательные поля баг-репортов.
6. Перечислите стадии жизненного цикла дефекта.

Лабораторная работа №10

Тема: Документирование и развертывание ПС.
Цель: Освоение методики документирования ПС.

Теоретические положения

10.1. Документирование ПО

Документирование относится к категории вспомогательных процессов жизненного цикла информационной системы, обеспечивающих выполнение основных процессов жизненного цикла ПС.

При осуществлении рабочего проектирования создаются следующие виды документации:

– функциональные спецификации – описывают функционирование отдельных компонентов ПС и системы в целом в отношении реализации функциональных требований к системе;
– проектные спецификации – раскрывают проектные решения, которые необходимы заказчику для выполнения функций сопровождения ПС;
– руководство пользователя – документ, регламентирующий действия пользователя при работе с ПС.

Документирование часто сопровождается диаграммами UML, которые позволяют визуально описывать приложение, наглядно представлять архитектуру и требования к приложению.

Основное требование к документации – она должна быть понятна и соответствовать функциональной и обеспечивающей структуре ПС. Основные ошибки при документировании ПС изложены в подразделе 3.3 при рассмотрении задачи документирования требований. Это нечеткие формулировки, игнорирование аудитории, для которой предназначена данная документация, а также пропуск важных аспектов, связанных с нефункциональными требованиями.

10.2. Руководство пользователя

Руководство пользователя – это своего рода инструкция по эксплуатации приложения конечными пользователями. Предназначено для всех людей, принимающих участие в настройке и эксплуатации программного приложения.

Примерная структура руководства пользователя:

1. «Назначение приложения». Здесь задаются общие понятия, необходимые навыки для работы с системой:

- название приложения и номер версии;
- класс ПС, для которых предназначено приложение;

- степень универсальности приложения для решения поставленных задач;
- характеристики пользовательского интерфейса;
- уровень подготовки пользователей для эффективного взаимодействия с приложением;

2. «Установка приложения». В этой части описывается процесс установки и базовой настройки системы.

3. «Концепция приложения». Приводится общая структура системы, приводится справочник терминов и понятий, которые используются в интерфейсах управления системой и данном руководстве, описывается процесс функционирования в общем виде.

4. «Функциональный состав приложения». Рассматривается процесс решения задач на объекте в рамках функционального состава ПС. По ходу описания приводятся экранные формы всех интерфейсных окон / страниц, посредством которых пользователи взаимодействуют с приложением и устанавливаются ссылки на них.

5. «Инструменты и настройки приложения». Описывается назначение и применение настроек приложения, работа с модулями, обновлениями и полезными инструментами системы.

6. «Получение помощи». Приводятся способы решения возникших проблем и получения помощи (Help).

Задание

1. Собрать документированные материалы, разработанные при выполнении лабораторных работ №1–9 в единый документ с разбивкой на разделы. Темы лабораторных работ обозначить названиями разделов. В разделы не включать цель, задание и выводы из отчетов о выполнении лабораторных работ.
2. Дополнительно отдельным разделом 10 описать назначение, технические характеристики, принцип работы и меры безопасности при эксплуатации ПС.
3. Дополнительно отдельным разделом 11 составить руководство пользователя.

Контрольные вопросы и упражнения

1. Что такое документирование ПС и каково его назначение?
2. Какие элементы включает в себя структура документации к ПС?
3. Как документируются диаграммы в нотации IDEF0?
4. Что такое руководство пользователя?
5. Перечислите основные структурные элементы руководства пользователя.
6. Каково назначение руководства пользователя?
7. Какие задачи решаются на этапе эксплуатации ПС?

Список рекомендуемой литературы

ГОСТы:

1. ГОСТ Р ИСО/МЭК 12207–99. ГОСУДАРСТВЕННЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ. Информационная технология. ПРОЦЕССЫ ЖИЗНЕННОГО ЦИКЛА ПРОГРАММНЫХ СРЕДСТВ.
2. ГОСТ Р ИСО/МЭК 15504-1-2009. Информационные технологии. Оценка процессов. Часть 1. Концепция и словарь Текст. Введ. 14.09.2009. – М. : Стандартинформ, 2010. – 19 с.
3. ГОСТ 34.601–90. Автоматизированные системы. Стадии создания. В кн.: Информационная технология. Комплекс стандартов и руководящих документов на автоматизированные системы. М.: Комитет стандартизации и метрологии СССР,1991. – с.45-52.
4. ГОСТ 34.602–89. Техническое задание на создание автоматизированной системы. В кн.: Информационная технология. Комплекс стандартов и руководящих документов на АС. М.: Комитет стандартизации и метрологии СССР,1991. – с.15-29
5. РД 50–34.698–90. Автоматизированные системы. Требования к содержанию документов. В кн.: Информационная технология. Комплекс стандартов и руководящих документов на АС. М.: Комитет стандартизации и метрологии СССР,1991. – с.66-104.
6. ГОСТ 19.701–90 (ИСО 5807-85). Схемы алгоритмов, программ, данных и систем. Госстандарт СССР, 1990. – 20 с.

Основная литература:

7. Полетайкин, А. Н. Социальные и экономические информационные системы: законы функционирования и принципы построения : учеб. пособие / А. Н. Полетайкин ; Сиб. гос. ун-т телекоммуникаций и информатики. - Новосибирск : СибГУТИ, 2016. - 240 с. : ил.
8. Гагарина Л.Г., Кокорева Е.В., Виснадул Б.Д. Технологии разработки программного обеспечения : учеб. пособие; под ред. Л.Г. Гагариной. – М.: ИД "ФОРУМ" : ИНФРА-М, 2012. – 399 с.
9. Штерн В. Основы С++. Методы программной инженерии: Пер. с англ. : монография. – М. : Лори, 2003. – 860с.
10. Мартин Р. Чистый код. Создание, анализ и рефакторинг : монография. – СПб.: ПИТЕР, 2012. – 464 с.
11. Мартишин С.А., Симонов В.Л., Храпченко М.В. Основы теории надёжности информационных систем : учеб. пособие. – М. : ФОРУМ: ИНФРА-М, 2013. – 254 с. : ил.
12. Орлов С.А. Технологии разработки программного обеспечения : Учебник для вузов. – СПб. : ПИТЕР, 2002. – 463 с.
13. Лавров С.С. Программирование. Математические основы, средства, теория : монография. – СПб. : БХВ-Петербург, 2001. – 317 с.
14. Кнут Д.Э. Искусство программирования: В 3 т. : Пер. с англ. Т.3:Сортировка и поиск : монография / Под ред. Ю.В. Козаченко. – 2-е изд. – М. :

Издат.дом "Вильямс", 2003. – 822 с.

15. Вернер М. Основы кодирования : учебник / пер. с нем. Д.К. Зигангирова. – М. : Техносфера, 2004. – 286 с.

16. Иванова Г.С. Технология программирования : учебник – 3-е изд., перераб. и доп. – М. : Изд-во МГТУ им. Н. Э. Баумана, 2006. – 335 с.

17. Липаев В.В. Технологии разработки программного обеспечения. Методологические основы: учебник. – М.: «ТЕИС», 2006 – 608 с.

18. Никитин В.А. Управление качеством на базе стандартов ИСО 9000:2000 : монография. – СПб. : ПИТЕР, 2002. – 262 с.

Дополнительная литература:

19. Губарев А.В. Информационное обеспечение системы менеджмента качества : моногр. – Москва : Горячая линия-Телеком, 2013. – 132 с. : ил.

20. Лодон Дж. Управление информационными системами : учебник. – 7-е изд. – СПб. : ПИТЕР, 2005. – 910 с.

21. Кронрод А.С. Беседы о программировании : научно-популярная литература / предисл. Л.А. Кронрод; послеслов. В.Л. Арлазарова. – 2-е изд., стереотип. – М. : УРСС, 2004. – 246 с.

22. Гвоздева В.А. Введение в специальность программиста : учебник. – 2-е изд., испр. и доп. – М. : ИД "ФОРУМ"-ИНФРА-М, 2007. – 207 с.

23. Вирт Н. Алгоритмы и структуры данных. Новая версия для Оберона + CD : монография / пер. с англ. под ред. Ф.В. Ткачева. – 2-е изд., испр. – М. : ДМК Пресс, 2012. – 272 с.

24. Босуэлл Д., Фаучер Т. Читаемый код или Программирование как искусство : монография. – СПб. : ПИТЕР, 2012. – 203 с.

25. Зыков С.В. Основы современного программирования. Разработка гетерогенных систем в Интернет-ориентированной среде : учеб.пособие. – 2-е изд.,стер. – М. : Горячая линия-Телеком, 2012. – 444 с.

26. Гуриков С.Р. Введение в программирование на языке Visual C# : учеб. пособие. – М. : ФОРУМ-ИНФРА-М, 2013. – 444 с.

27. Кузнецов С.М., Малозёмов Б.В. Программирование и основы алгоритмизации : учеб. пособие. – Новосиб. гос. техн. ун-т. – Новосибирск : Изд-во НГТУ, 2006. – 199 с. : ил.

28. Кнут Д.Э. Искусство программирования: В 3 т. : Пер. с англ. Т.1:Основные алгоритмы : монография / Под общ. ред. Ю.В.Козаченко. – М. : Издат.дом "Вильямс", 2002. – 712 с.

29. Кнут Д.Э. Искусство программирования: В 3 т. : Пер. с англ. Т.2:Получисленные алгоритмы : монография / Под общ. ред. Ю.В.Козаченко. – 3-е изд. – М. : Издат.дом "Вильямс", 2003. – 828 с.

30. Макконелл Дж. Основы современных алгоритмов : учеб. пособия / пер. с англ. под ред. С.К. Ландо; доп. М.В. Ульянова. – М. : Техносфера, 2004. – 366 с.

Приложение А – Варианты индивидуальных заданий

1. Обменный пункт: сотрудники пункта, виды валют, курсы валют, операции обмена.
2. Ювелирный магазин: названия изделий, комитенты (кто сдал изделия на комиссию), журнал сдачи изделий на продажу, журнал покупки изделий.
3. Поликлиника: врачи, пациенты, виды болезней, журнал учета прихода пациентов.
4. Кондитерский магазин: виды конфет, поставщики, торговые точки, журнал поступления и отпуска товара.
5. Автобаза: автомашины, водители, рейсы, журнал выезда машин на рейсы.
6. Парикмахерская: клиенты, прайс услуг, сотрудники, кассовый журнал.
7. Школа: учителя, предметы, ученики, журнал успеваемости.
8. Оплата услуг на дачных участках: виды услуг, список владельцев, сотрудники управления, журнал регистрации оплат.
9. Гостиница: проживающие, сотрудники гостиницы, номера, журнал регистрации проживающих.
10. Книжный магазин: авторы, книги, продавцы, покупатели, регистрация поступлений и продаж.
11. Ремонтная мастерская: виды работ, исполнители, заказы на ремонт, заказчики.
12. Аптечный киоск: номенклатура лекарств, работники аптеки, покупатели, журнал регистрации продаж.
13. Выставка: стенды, стендисты, экскурсии, посетители.
14. Охранная служба: список постов охраны, список охранников, журнал выхода на дежурство, журнал учета замечаний.
15. Столовая: продукты, блюда, меню, журнал заказов
16. Фото мастерская: заказчики работ, прайс работ, журнал поступления заказов, исполнители.
17. Ветеринарная лечебница: список животных, список болезней, список хозяев, журнал посещений.
18. Сельское хозяйство: список растений, список угодий, список работников, журнал посевной.
19. Холдинг: список регионов, список предприятий, список показателей, журнал учета отчетных данных.
20. Фонды предприятия: список основных средств, список категорий основных средств, список материально ответственных лиц, журнал учета состояния основных средств.
21. Учет расхода материалов в компании: список статей затрат, список сотрудников, журнал учета расхода канцтоваров, список департаментов.
22. Фильмотека: список фильмов, список клиентов, список библиотекарей, журнал выдачи фильмов.
23. Цирк: список категорий артистов, список артистов, журнал выхода артистов на работу, список цирковых площадок.

24. Спортивные мероприятия: список спортсменов, список видов спорта, список стадионов, журнал учета выступлений спортсменов.
25. Компьютерные занятия: список слушателей курсов, список предметов, список преподавателей, журнал учета успеваемости.
26. Сбор урожая: список видов продукции, список сборщиков, список бригад, журнал учета сбора урожая.
27. Фирма по обслуживанию населения: список заказчиков, список товаров, список разносчиков, журнал заказов.
28. Партийная работа: список членов партии, список мероприятий, журнал учета выхода на мероприятие, список городов
29. Экономическая база данных: список регионов, список показателей, список отраслей, отчетные статистические данные.
30. Журнальные статьи: список тем, список авторов, список названия статей, список журналов.
31. Анализ причин заболеваемости: список больных, список болезней, список районов, журнал учета заболевших.
32. Отдел кадров: список сотрудников, штатное расписание, список отделов, журнал перемещения сотрудников по службе.
33. Расчет нагрузки на преподавателя: список преподавателей, список кафедр, предметов, журнал нагрузки.
34. Проектные работы: список проектов, список специалистов, список должностей, журнал учета работ.
35. Учет компьютерного оборудования: список типов оборудования, список материально ответственных лиц, список департаментов, журнал регистрации выдачи оборудования.
36. Прививки детям: список прививок, список детей, список родителей, журнал учета сделанных прививок.
37. Начисление налогов в бюджет: виды налогов, список отраслей, список предприятий, журнал учета поступления налогов.
38. Экспертная система: список оцениваемых объектов, список экспертов, список регионов, журнал учета оценок.
39. Ремонтная мастерская электронного оборудования: список работ, список мастеров, список запасных частей, журнал учета выполненных работ, список поступившего оборудования.
40. Магазин по продаже автомобилей: список фирм производителей, список автомобилей, журнал поступления автомобиля, список водителя пригнавшего машину.
41. Автомобильный гараж: список владельцев, список автомобилей, список сторожей, журнал прихода и ухода автомобилей.
42. Учет криминогенной ситуации в городе: список районов, список типов преступлений, список дежурных, журнал регистрации преступлений.
43. Система здравоохранения: список регионов, список санаториев, список пенсионеров, журнал регистрации выдачи путевок в санатории.
44. Туристические агентства: список туров, список стран, список клиентов, журнал регистрации продаж туров.

45. Продажа билетов на рейсы: список рейсов, прайс билетов, список компаний, журнал продаж билетов.
46. Продажа пиломатериалов: виды пиломатериалов, регионы поставщики, список заказчиков, журнал учета продаж пиломатериалов.
47. Склад металлоконструкций: прайс товара металлоконструкций, список поставщиков, список продавцов, журнал учета продаж.
48. Система поддержки решений: список экспертов, список тем обсуждений, список департаментов, журнал учета предложений.
49. Детский сад: список родителей, список детей, список групп, журнал посещения детского сада.
50. Дом творчества молодежи: список кружков, список руководителей, список детей, журнал регистрации посещения кружков.

Приложение Б – Примерное описание объекта информатизации

Характеристика объекта информатизации

Объектом информатизации является ПАО Банк «ФК Открытие». Банк «Открытие» − крупнейший частный банк в России и четвертый по размеру активов среди всех российских банковских групп. Работает на финансовом рынке с 1993 года [1].

Активы банка и его дочерних компаний по международным стандартам финансовой отчётности (МСФО) на 30 июня 2016 года составили 3 087,8 млрд рублей, собственный капитал − 217,8 млрд рублей.

«Открытие» − универсальный коммерческий банк с устойчивой диверсифицированной структурой бизнеса и качественным управлением капиталом.

Банк развивает следующие ключевые направления бизнеса:

- корпоративный;
- инвестиционный;
- розничный;
- малый и средний бизнес;
- приват банкинг.

Особое внимание «Открытие» уделяет высокотехнологичным сервисам. Так, в рамках проекта «Рокетбанк» предлагается полностью дистанционный сервис для физических лиц, а в рамках проекта «Точка» − полностью дистанционное обслуживание для предпринимателей.

Банк «Открытие» был сформирован в результате интеграции более чем десяти банков различного масштаба, в том числе таких крупных федеральных, как НОМОС-БАНК, Ханты-Мансийский банк и банк «Петрокоммерц».

Банк «Открытие» входит в список десяти системообразующих кредитных организаций, утвержденный Центральным банком, а также в список крупнейших компаний мира FORBES Global 2000.

Надежность банка подтверждена рейтингами международных агентств S&P Global (BB-) и Moody’s Investors Service (Ba3).

Клиентская база объединенного банка насчитывает свыше 30 000 корпоративных клиентов, 165 000 клиентов малого бизнеса и около 3,2 млн физических лиц, в том числе премиальных клиентов. 470 отделений банка различного формата расположены в 53 экономически значимых регионах России. Значительная часть бизнеса сосредоточена в Москве, Санкт-Петербурге, Тюменской области, Екатеринбурге, Новосибирской области, Хабаровском крае, Волгоградской области.

Ключевым акционером банка «Открытие» является «Открытие Холдинг», который владеет 64,7% голосующих акций (см. рис.). Бенефициарами «Открытие Холдинг» являются: Вадим Беляев, Группа «ИФД Капиталъ», Банк «ВТБ»,

Группа «ИСТ», Рубен Аганбегян, Александр Мамут. Бумаги банка также находятся в свободном обращении на Московской бирже.

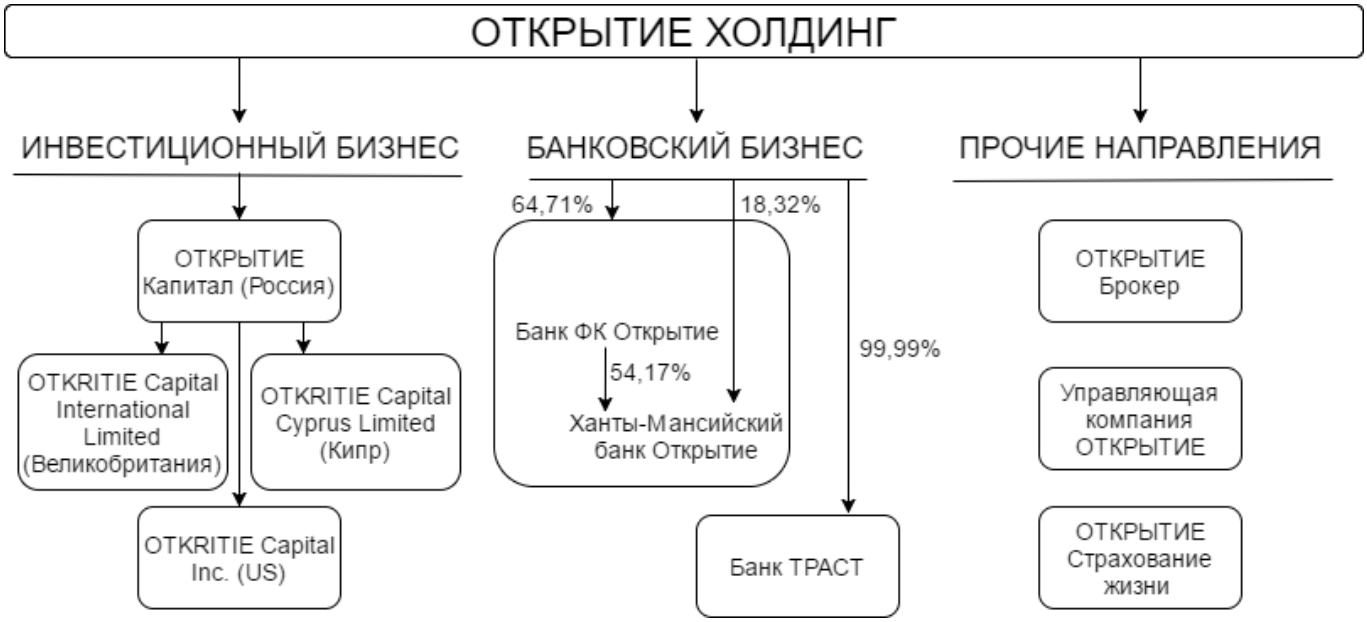


Рисунок − Структура ОАО «Открытие Холдинг»

Главой Наблюдательного совета банка «Открытие» является Дмитрий Ромаев, председателем Правления − Евгений Данкевич.

Банк «Открытие» активно участвует в реализации социально значимых проектов, поддерживая такие направления, как наука (Политехнический музей), образование (Московский государственный университет им. Ломоносова, Европейский университет в Санкт-Петербурге), культура (Малый театр, сотрудничество с автором мультфильма «Ёжик в тумане» Юрием Норштейном), здравоохранение (Фонд помощи хосписам «Вера»), спорт (стратегическое партнерство с ФК «Спартак-Москва», Югорский лыжный марафон).

Для информатизации было выбрано отделение банка в г. Новосибирске, располагающееся по адресу: 630102, г. Новосибирск, ул. Кирова, д. 44.

Контактный телефон: (383) 325-10-85.

Управляющий филиалом: Ермилов Валерий Николаевич.

Главный бухгалтер: Жданов Алексей Владимирович.

Филиал обслуживает как частные, так и юридические лица.

Услуги частным лицам включают в себя:

- срочные переводы по системе «Western Union» в рублях (по России) и в валюте в любую точку мира;
- прием коммунальных платежей;
- оплата услуг операторов мобильной связи;
- оплата кредитов, полученных в других банках;
- автокредитование;
- кредиты по пластиковым картам;
- открытие и ведение текущих рублёвых и валютных счетов;
- предоставление индивидуальных банковских сейфов;

– оформление и обслуживание пластиковых карт;
– денежные вклады;
– ипотечное кредитование;
– инвестиции;
– страхование;
– пенсионный калькулятор;
– сейфовые ячейки;
– операции с иностранной валютой и чеками.

Услуги корпоративным клиентам включают в себя:

– расчётно-кассовое обслуживание, в том числе по системе «Банк-Клиент»;
– широкий выбор кредитных продуктов (кредитные линии, овердрафт, вексельные кредиты, лизинг);
– оперативное проведение валютно-финансовых операций;
– вклады и депозиты для корпоративных клиентов;
– организация процедуры выплаты зарплаты с помощью пластиковых карт;
– оформление пластиковых корпоративных карт международных платёжных систем;
– консалтинговые услуги;
– дистанционное обслуживание;
– инвестиции.

Процессами информатизации являются: расчёт ежемесячного платежа для физических лиц и оценка их кредитоспособности при ипотечном кредитовании.

Ипотечное кредитование − долгосрочный кредит, предоставляемый юридическому или физическому лицу банками под залог недвижимости (земли, производственных и жилых зданий, помещений, сооружений). Самый распространенный вариант использования ипотеки в России − это покупка физическим лицом квартиры в кредит. Закладывается при этом, как правило, вновь покупаемое жилье, хотя можно заложить и уже имеющуюся в собственности квартиру [2].

Цель информатизируемых процессов: рассчитать ежемесячный платёж и проверить кредитоспособность заёмщика.

Ограничения на операции: возраст клиента должен быть не менее 18 лет и не более 65 лет на момент выплаты последнего платежа.

Перечень ссылок

1 Банк «Финансовая корпорация Открытие» [Офиц. сайт]. URL: https://www.open.ru (дата обращения: 17.05.2017).

2 IPOhelp [Офиц. сайт]. URL: http://www.ipohelp.ru/manual.html (дата обращения: 10.02.2017).

Приложение В – Показатели для оценивания программной системы

(согласно ГОСТ Р ИСО/МЭК 9126-93)

1. Целостность – способность системы противостоять (сохранять информационное содержание и однозначность интерпретации смысла) изменениям, искажениям или порче при возникновении сбоев или ошибок.
2. Рациональность – подразумевает, что при функционировании оптимальным образом используются имеющиеся в распоряжении системы ресурсы: время, оборудование (память), люди.
3. Численность задействованного персонала – количество лиц, задействованных в эксплуатации и обслуживании системы.
4. Работоспособность – способность системы выполнять свои функции с эксплуатационными показателями не ниже заданных.
5. Производительность – характеристика системы, отражающая ее способность производить определенный объем работ в единицу времени, например, пропускная способность, время ответа, доступность, число продуктов, полученная прибыль, быстродействие.
6. Гибкость – Возможность модификации обеспечивающей части системы, обычно возникает по двум причинам: чтобы отразить в системе изменение требований или чтобы исправить ошибки, внесённые ранее в процессе разработки. Гибкость заключается в возможности адаптации, наращивания, изменения средств.
7. Открытость – прозрачность функциональной части системы и возможность ее модификации без нарушения процесса функционирования.
8. Защищенность (Безопасность) – способность обеспечения защиты данных от разрушения, искажения или преднамеренных фальсификаций злоумышленником. Характеризует возможное отсутствие риска, связанного с нанесением некоторого ущерба
9. Потенциальная управляемость – возможность компенсировать возмущение быстрее, чем успеют измениться эти возмущения.
10. Наблюдаемость основных параметров управляемого процесса – характеризует степень и глубину отслеживания параметров функционирования системы и основных характеристик объекта управления на всех циклах и этапах работы системы/объекта.
11. Надёжность – свойство системы сохранять во времени в установленных пределах значения всех параметров, характеризующих способность системы выполнять требуемые функции в заданных режимах и эксплуатации.
12. Степень оперативности и надежности управления – характеризует способность системы и субъекта управления выполнять поставленные задачи в сроки, обеспечивающие эффективное управление объектом. Управление является оперативным, если время, затрачиваемое на решение системой задач, в совокупности со временем, необходимым персоналу на подготовку к реализации управленческих решений, не будет превышать критического времени, диктуемого условиями сложившейся ситуации на объекте. О.у.

достигается ускорением процесса сбора и обработки данных о состоянии объекта, принятия решений, планирования прогнозирования социально-экономических показателей.

13. Безотказность – свойство системы сохранять работоспособность в течение требуемого интервала времени непрерывно без вынужденных перерывов. Безотказность является наиболее важной компонентой надёжности, так как она отражает способность длительное время функционировать без отказов. Один из показателей надежности системы.
14. Живучесть – свойство системы сохранять работоспособность в условиях возмущающих воздействий внешней среды и отказов компонентов системы с минимальной частотой отказов, а в случае их возникновения эффективно восстанавливать утраченные функции и ресурсы.
15. Прогрессивность компьютерных и информационных технологий, задействованных в системе
16. Наглядность интерфейса – должен быть удобным, интуитивно понятным и продуманным с точки зрения инженерной психологии, эргономики и методов технической эстетики
17. Долговечность — свойство системы сохранять работоспособность до наступления предельного состояния. Зависит от долговечности технических средств и подверженности системы моральному старению.
18. Сопровождаемость – отражает возможность проведения конкретных изменений (модификаций) системы. Изменение может включать исправления, усовершенствования или адаптацию компонентов системы к изменениям в окружающей обстановке, требованиях и условиях функционирования.
19. Стоимость – овеществленный в товаре труд разработчиков. Определяется количеством труда, материала, энергии и информации, затраченных на производство товаров, и измеряется количеством, например эквивалента рабочего времени.
20. Эффективность – получение от функционирования системы существенного технико-экономического, социального или другого эффекта.
21. Мобильность – возможность перенесения системы из одного окружения в другое. Окружающая обстановка может включать организационное, техническое или программное окружение, а также условия эксплуатации.
22. Наличие тех. поддержки. Введенная в эксплуатацию готовая система требует определенной технической поддержки. Это обусловлено, прежде всего, динамичностью информационных процессов: совершенствованием документооборота, появлением дополнительных структур данных и автоматизированных функций, что является обычным явлением для любых развивающихся систем.

Приложение Г – Состав и содержание технического задания на создание программ (ГОСТ 34.602- 89)

№ п/п	Раздел	Содержание
1	Общие сведения	• полное наименование программы и ее условное обозначение • шифр темы или шифр (номер) договора; • наименование предприятий разработчика и заказчика системы, их реквизиты • перечень документов, на основании которых создается программа • плановые сроки начала и окончания работ • сведения об источниках и порядке финансирования работ • порядок оформления и предъявления заказчику результатов работ по созданию системы, ее частей и отдельных средств
2	Назначение и цели создания (развития) программы	• вид автоматизируемой деятельности • перечень объектов, на которых предполагается использование программы • наименования и требуемые значения технических, технологических, производственно-экономических и др. показателей объекта, которые должны быть достигнуты при внедрении программы
3	Характеристика бизнес-процесса	• краткие сведения об объекте автоматизации • сведения об условиях эксплуатации и характеристиках окружающей среды
4	Состав и содержание работ по созданию программы	• перечень стадий и этапов работ • сроки исполнения • состав организаций — исполнителей работ • вид и порядок экспертизы технической документации • программа обеспечения надежности • программа математического обеспечения
5	Порядок контроля и приемки программы	• виды, состав, объем и методы испытаний программы • общие требования к приемке работ по стадиям • статус приемной комиссии
6	Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу программы в действие	• преобразование входной информации к машиночитаемому виду • изменения в объекте автоматизации • сроки и порядок комплектования и обучения персонала
7	Требования к документированию	• перечень подлежащих разработке документов • перечень документов на машинных носителях
8	Источники разработки	• документы и информационные материалы, на основании которых разрабатывается ТЗ и программы

Приложение Д – Примеры построения диаграмм физической реализации программной системы управления банкоматом

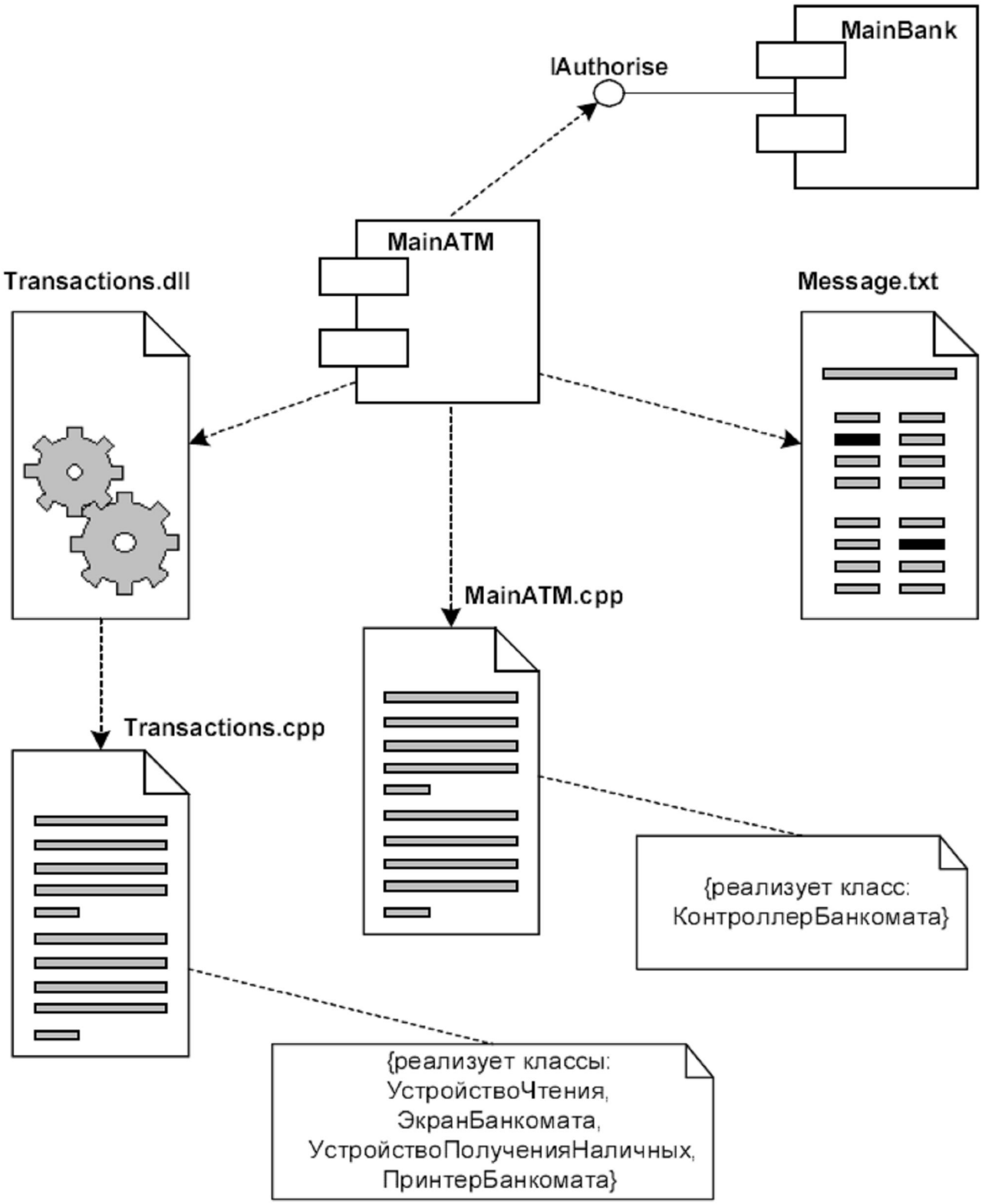


Рис. Д.1. Диаграмма компонентов

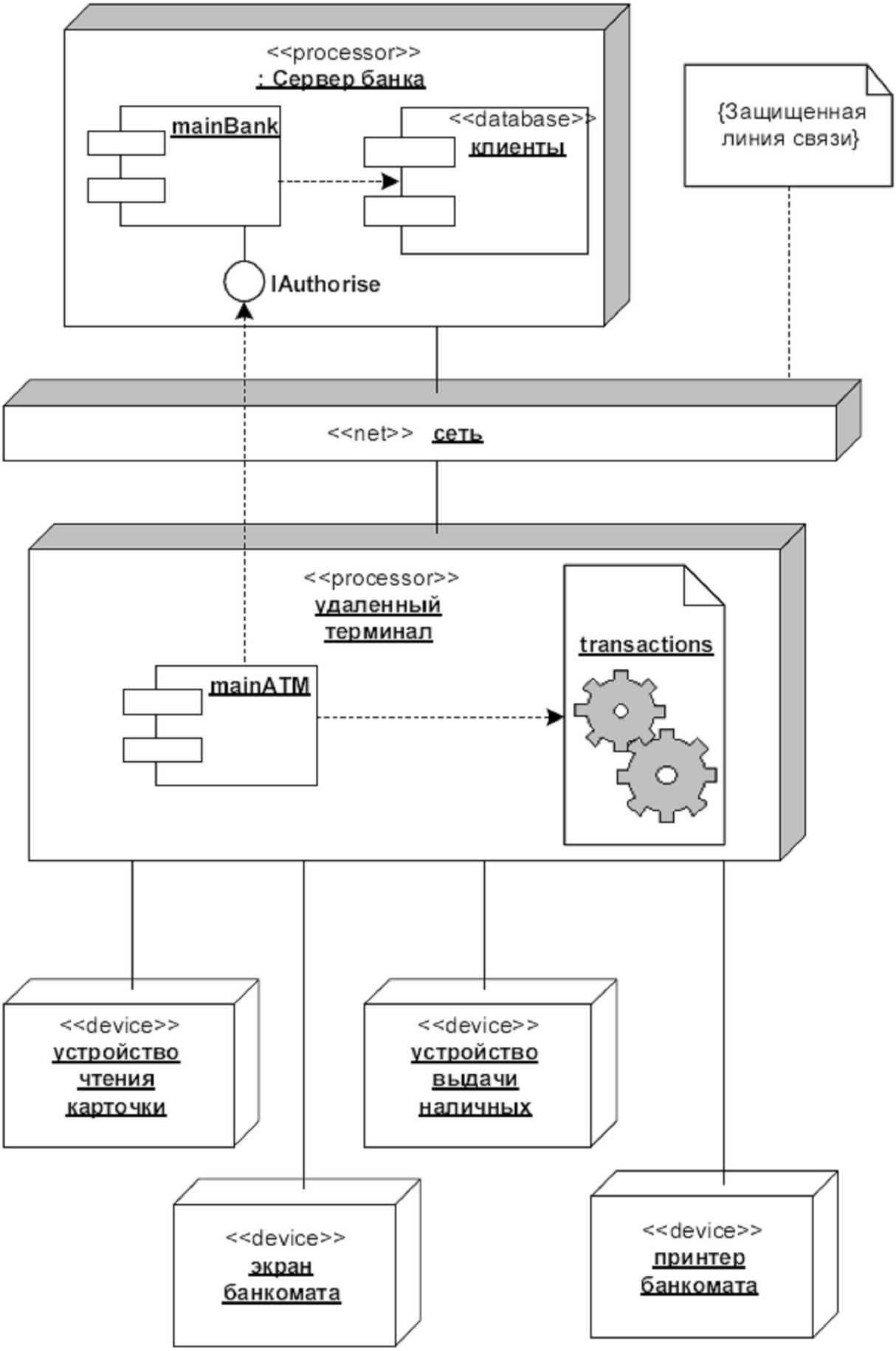


Рис. Д.2. Диаграмма развертывания

Приложение Е – Дополнительное задание на проведение сценарного тестирования

В качестве тестируемого программного продукта может быть выбран некоторый интернет-магазин, например:

- https://www.orange-elephant.ru/;
- https://flowwow.com/krasnodar/;
- https://sadovod-opt.com/ и др.).

Для разработки сценария необходимо выбрать три задачи из списка ниже. Например, в соответствии с задачей 8 из списка ниже один из сценариев может быть следующим: *поиск горнолыжной мужской куртки 52 размера, с ограничением стоимости*.

Перечень задач для сценарного тестирования.

1. Ориентирование в категориях каталога. Вариант 1
2. Ориентирование в категориях каталога. Вариант 2
3. Фильтрация по заданным параметрам
4. Сброс фильтров
5. Сортировка по заданному параметру
6. Изучение карточки товара в каталоге
7. Изучение страницы с подробным описанием товара
8. Поиск товара по заданным параметрам
9. Сравнение товаров
10. Ориентирование в подборках товаров
11. Добавление товара в избранное
12. Работа с изображением товара
13. Ориентирование в стоимости товара
14. Взаимодействие с чужими отзывами
15. Считываемость доступного кол-ва товара
16. Обращение в поддержку
17. Заказ в один клик
18. Добавление товаров в корзину
19. Удаление товара из корзины
20. Очистка корзины
21. Применение промокодов
22. Поиск ближайшей точки по карте
23. Понимание условий возврата товаров
24. Понимание условий программы лояльности
25. Понимание Push/SMS о статусе заказа
26. Считываемость баннеров
27. Перелистывание карусели
28. Отмена заказа
29. Треккинг заказа
30. Ориентирование в стоимости опций и заказа
31. Считываемость рекомендуемых товаров
32. Поиск заданной точки по карте

33. Оформление доставки
34. Смена города/региона
35. Работа с кликабельными тэгами
36. Оставление фидбека о товаре
37. Взаимодействие с видео о товаре
38. Передача ссылок и шеринг информации о товаре
39. Использование списка избранного
40. Повторение прошлых заказов
41. Редактирование профиля
42. Редактирование состава заказа в корзине
43. Изменений опций заказа
44. Выбор способа оплаты
45. Выбор способа получения товара
46. Считываемость экранов/сообщений об ошибке
47. Взаимодействие с всплывающими подсказками
48. Регистрация
49. Поиск товаров, похожих на просматриваемый
50. Считываемость продвигаемых товаров
51. Оформление самовывоза
52. Свободный поиск товара
53. Работа с поисковой строкой
54. Оплата заказа
55. Поиск товара через каталог