Среднее профессиональное образование

ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

А. В. Рудаков
Г. Н. Федорова

ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНЫХ ПРОДУКТОВ

ПРАКТИКУМ

Учебное пособие

ACADEMIA

соответствует ФГОС

СРЕДНЕЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАНИЕ

А. В. РУДАКОВ, Г. Н. ФЕДОРОВА

ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНЫХ ПРОДУКТОВ

Практикум

Рекомендовано
Федеральным государственным учреждением
«Федеральный институт развития образования» в качестве
учебного пособия для использования в учебном процессе
образовательных учреждений, реализующих программы
среднего профессионального образования

Регистрационный номер рецензии 325 от 16 июня 2009 г. ФГУ «ФИРО»

4-е издание, стереотипное

ACADEMA

Москва
Издательский центр «Академия»
2014

УДК 681.3.06(075.32)
ББК 32.973-018я723
Р83

Рецензенты:

ст. преподаватель кафедры «Инжиниринг и менеджмент качества» *Г. А. Акимов* (БГТУ «ВОЕНМЕХ» им. Д. Ф. Устинова); зав. лабораторией Центра компьютерного обучения Московского автомобилестроительного колледжа *А. А. Соломашкин*

Рудаков А. В.

Р83 Технология разработки программных продуктов. Практикум: учеб. пособие для студ. учреждений сред. проф. образования / А. В. Рудаков, Г. Н. Федорова. — 4-е изд., стер. — М. : Издательский центр «Академия»; 2014. — 192 с.

ISBN 978-5-4468-0465-8

В учебном пособии в систематизированном виде приведены необходимые теоретические сведения, практические задания и примеры их выполнения; представлены задания на построение моделей программных продуктов с использованием как структурного, так и объектно-ориентированного подхода (с применением стандартного языка моделирования UML и современных CASE-средств), задания на разработку тестов, справочной системы, а также на создание инсталляционных пакетов программных продуктов.

Учебное пособие может быть использовано при изучении профессионального модуля ПМ.03 «Участие в интеграции программных модулей» (МДК.03.01) в соответствии с требованиями ФГОС СПО для специальности 230115 «Программирование в компьютерных системах» и является частью учебно-методического комплекта.

Для студентов учреждений среднего профессионального образования.

УДК 681.3.06(075.32)
ББК 32.973-018я723

ISBN 978-5-4468-0465-8

Уважаемый читатель!

Данное учебное издание является частью учебно-методического комплекта по специальности «Программирование в компьютерных системах».

Практикум предназначен для изучения профессионального модуля ПМ.03 «Участие в интеграции программных модулей» (МДК 03.01).

Учебно-методические комплекты нового поколения включают в себя традиционные и инновационные учебные материалы, позволяющие обеспечить изучение общеобразовательных и общепрофессиональных дисциплин и профессиональных модулей. Каждый комплект содержит учебники и учебные пособия, средства обучения и контроля, необходимые для освоения общих и профессиональных компетенций, в том числе и с учетом требований работодателя.

Учебные издания дополняются электронными образовательными ресурсами. Электронные ресурсы содержат теоретические и практические модули с интерактивными упражнениями и тренажерами, мультимедийные объекты, ссылки на дополнительные материалы и ресурсы в Интернете. В них включен терминологический словарь и электронный журнал, в котором фиксируются основные параметры учебного процесса: время работы, результат выполнения контрольных и практических заданий. Электронные ресурсы легко встраиваются в учебный процесс и могут быть адаптированы к различным учебным программам.

Учебно-методический комплект разработан на основании Федерального государственного образовательного стандарта среднего профессионального образования с учетом его профиля.

Предисловие

Практикум подготовлен в соответствии с Государственным образовательным стандартом по специальности «Программное обеспечение вычислительной техники и автоматизированных систем» и по своему содержанию дополняет выпущенное ранее учебное пособие «Технология разработки программного продукта» (2005 г.).

Цель практикума — формирование у студентов практических навыков грамотной разработки программных продуктов (ПП) с использованием современных методов и средств. Основная идея этих методов заключается в применении инженерного подхода к созданию ПП.

Практикум включает в себя основные аспекты учебного курса «Технология разработки программного продукта». В практикуме более детально рассмотрены теоретические и практические вопросы, изложенные в одноименном учебном пособии. Практикум включает в себя перечень заданий и методические указания по их выполнению. Структура учебного пособия отражает последовательность изложения материала, принятую в учебной программе. Практикум состоит из 13 глав, списка литературы и приложения. Каждая глава содержит теоретическую часть, практические задания, методические указания для их выполнения и контрольные вопросы. Изложение материала строится на примере выполнения сквозного задания — от разработки технического задания до получения готового программного продукта.

В последней главе дана примерная структура курсового проекта по дисциплине «Технология разработки программного продукта», состав документации, а также предлагаются примеры выполнения курсовых проектов.

В Приложении приведен перечень индивидуальных заданий, которые можно использовать и как задания на практических занятиях, и как темы для курсового проектирования.

Глава 1

РАЗРАБОТКА ТЕХНИЧЕСКОГО ЗАДАНИЯ

1.1. ОСНОВНЫЕ СВЕДЕНИЯ

Программный документ «Техническое задание» разрабатывается в соответствии с ГОСТ 19.201—78. Техническое задание содержит совокупность требований к программному средству и может использоваться как критерий проверки и приемки разработанной программы, поэтому достаточно полно составленное (с учетом возможности внесения дополнительных разделов) и принятое заказчиком и разработчиком техническое задание является одним из основополагающих документов проекта. Умение грамотно создавать техническое задание на разработку программного продукта определяет профессиональный уровень программиста и избавляет его от претензий со стороны заказчика.

Техническое задание представляет собой документ, в котором формулируются основные цели разработки, требования к программному продукту, определяются сроки и этапы разработки и регламентируется процесс приемно-сдаточных испытаний. В формулировании технического задания участвуют представители как заказчика, так и исполнителя. В основе этого документа лежат исходные требования заказчика, результаты выполнения предпроектных исследований и т. п.

Разработка технического задания выполняется в такой последовательности:

1) устанавливают набор выполняемых функций, а также перечень и характеристики исходных данных;

2) определяют перечень результатов, их характеристики и способы их представления;

3) уточняют среду функционирования программного обеспечения: конкретную комплектацию и параметры технических средств, версию используемой операционной системы и, возможно, версии и параметры другого установленного программного

обеспечения, с которым предстоит взаимодействовать будущему программному продукту.

В случаях, когда разрабатываемое программное обеспечение собирает и хранит некоторую информацию или включается в управление каким-либо техническим процессом, необходимо также четко регламентировать действия программы при сбое оборудования и энергоснабжения.

Основные факторы, определяющие характеристики разрабатываемого программного обеспечения:

- исходные данные и требуемые результаты, которые определяют функции программы или системы;
- среда (программная и аппаратная), в которой разрабатываемое программное обеспечение будет функционировать, может быть задана, а может выбираться для обеспечения параметров, указанных в техническом задании;
- возможное взаимодействие с другим программным обеспечением и (или) конкретными техническими средствами также может быть определено, а может выбираться исходя из набора выполняемых функций.

В соответствии с ГОСТ 19.201—78 программный документ «Техническое задание» содержит следующие разделы.

Введение.

1. Основание для разработки.
2. Назначение разработки.
3. Требования к программе или программному изделию.
4. Требования к программной документации.
5. Технико-экономическое обоснование.
6. Стадии и этапы разработки.
7. Порядок контроля и приемки.
8. Приложения.

В зависимости от особенностей программы или программного изделия допускается уточнять содержание разделов, вводить новые разделы или объединять некоторые из них. Рассмотрим подробнее содержание разделов технического задания.

Во введении указываются цель разработки программного продукта, краткая характеристика области применения и описание объекта, в котором он используется, т.е. описание предметной области.

1. В разделе «Основание для разработки» должны быть указаны:

- документы, на основании которых ведется разработка;
- организация, утвердившая этот документ, и дата его утверждения;

- наименование и (или) условное обозначение темы разработки.

2. Раздел «Назначение разработки» содержит определение функциональных и эксплуатационных задач, которые должна решить разрабатываемая система для достижения поставленной цели. Назначением программы может быть управление техническим комплексом, различные калькуляции, совершенствование производства и т.д. При необходимости программного обеспечения информационных систем целью разработки может быть получение своевременной и точной информации для принятия обоснованных, объективных решений, избавление пользователя от рутинного труда в делопроизводстве и перевод учреждения на безбумажную технологию и т.д. В этом же разделе должна быть представлена начальная контекстная диаграмма задачи.

3. В раздел «Требования к программе или программному изделию» входят следующие подразделы:

- требования к функциональным характеристикам;
- требования к надежности и безопасности;
- условия эксплуатации;
- требования к составу и параметрам технических средств;
- требования к информационной и программной совместимости;
- требования к маркировке и упаковке;
- требования к хранению и транспортированию;
- специальные требования.

Требования к *функциональным характеристикам* включают в себя описание состава выполняемых функций, требования к входной и выходной информации, а также к сервисным функциям программы. Для определения функций программы необходимо тщательно изучить работу ее будущих пользователей, составить список всех операций, выполняемых вручную или с использованием других программ, выделить среди них те, которые подлежат автоматизации. Например, к основным функциональным характеристикам программного обеспечения информационной системы относятся:

- возможность поиска и отбора необходимой информации из базы данных с использованием поисковой системы;
- формирование требуемых форм отчетности на основе отобранных данных;

- необходимые калькуляции и расчеты с использованием баз данных;
- возможность предоставления существующей базы данных другим приложениям;
- возможность работы пользователя с системой через Интернет и т.д.

В дальнейшем, исходя из функциональных характеристик, определяется структура и назначение файлов данных, используемых в данной системе (электронные справочники, журналы документов, электронные личные дела, архивы и т.п.). На этом этапе уже можно определить, какая архитектура информационной системы (клиент—сервер, файл—сервер) является необходимой и достаточной для успешного решения поставленных задач.

При описании требований к входным данным должны быть указаны их характер, организация и предварительная подготовка, формат, описание и способ кодирования. *Входной* информацией программы могут быть первичные документы (накладные, отчеты и т.д.), нормативно-справочная информация (справочники, классификаторы, кодификаторы и т.д.), электронные документы, входные сигналы и т.п. *Выходной* информацией программы могут быть документы (электронные или бумажные), файлы данных, выходные сигналы и т.д. При описании требований к выходным данным указывается их характер, организация, формат, описание и способ кодирования.

Помимо основных функций в техническом задании описываются требования к сервисным функциям программы, такие как возможность корректировки настроек (конфигурирования) системы, возможность резервного сохранения данных, изменения пароля входа в систему, вызова без выхода из программы календаря, калькулятора, редактора и т.д. Если разработанное программное обеспечение не будет выполнять указанных в техническом задании функций, то оно считается не соответствующим техническому заданию, т.е. неправильным с точки зрения критериев качества. Универсальность будущего продукта также обычно специально не оговаривается, но подразумевается.

Требования к *надежности и безопасности* содержат описание требований к обеспечению надежного и устойчивого функционирования программного продукта, к контролю входной и выходной информации, ко времени восстановления после отказа и т.п. *Надежность* — способность программы безотказно выполнять определенные функции при заданных условиях в течение заданного периода времени с достаточно большой вероятностью. Надежный

программный продукт не исключает наличия в нем ошибок, но важно, чтобы ошибки при практическом применении в заданных условиях проявлялись редко. Степень надежности характеризуется вероятностью работы программного продукта без отказа в течение определенного периода времени. Существует множество подходов к обеспечению надежности системы (предупреждение ошибок, исправление ошибок, самовосстановление системы после сбоев, проверка вводимых данных в рамках допустимых значений и т.д.). Самый простой способ — ограничение доступа. Контроль доступа к программному продукту и базе данных строится путем парольной защиты программ при их запуске, использования ключевой дискеты для запуска программ, ограничения программ или данных, функции обработки, доступных пользователям и т.д.

Требования к *составу и параметрам технических средств* включают указания на состав технических средств и их основные характеристики, а именно: минимальные системные требования, необходимые для работы программы; указываются мощность процессора (Гц), на базе которого должен работать ПК, объем оперативной памяти (Мб), необходимый объем свободного дискового пространства, разрешение монитора, наличие устройства чтения компакт-дисков и т.п., а также возможность переноса программы с одной аппаратной платформы на другую.

Требования к *информационной и программной совместимости* содержат требования к информационным структурам, языкам программирования и программным средствам, используемым программой, а именно:

- требования к операционным системам и средам, в которых может функционировать разрабатываемый программный продукт;
- возможность адаптации программы к различным операционным системам;
- необходимость установки на компьютер пакетов программ — средств разработки приложений (для доработки, модернизации или эксплуатации данного программного продукта);
- необходимость инсталляции различных графических компонентов и т.д.

4. «Требования к программной документации». Основными документами, регламентирующими разработку будущих программ, должны быть документы Единой системы программной документации: руководство пользователя, руководство администратора, описание применения.

Эффективность системы определяется удобством ее использования и экономической выгодой, полученной от внедрения программно-аппаратного комплекса.

5. В разделе «Технико-экономическое обоснование» представлены ориентировочная экономическая эффективность разрабатываемого программного продукта, экономические преимущества разработки по сравнению с имеющимися на предприятии образцами или аналогами (или в сравнении с ручными операциями).

6. Стадии и этапы разработки описаны в учебном пособии А. В. Рудакова «Технология разработки программных продуктов».

7. «Порядок контроля и приемки» предполагает указание на виды испытаний и общие требования к приему работы.

В программный документ «Техническое задание» допускается включать приложения, где при необходимости приводят:

- образцы входных и выходных документов и отчетов, описания файлов данных и т. д.
- перечень научно-исследовательских и других работ, обосновывающих разработку;
- схемы алгоритмов, таблицы, описания, обоснования, расчеты и т. д.;
- другие источники разработки.

1.2. ПРИМЕРЫ РАЗРАБОТКИ ТЕХНИЧЕСКОГО ЗАДАНИЯ

1.2.1. Техническое задание на разработку программного обеспечения АИС «Склад оптовой торговли»

Введение

Работа выполняется в рамках проекта автоматизации управления торговым предприятием.

1. Основание для разработки

Основанием для разработки является договор № _____ от _____.

Организация, утвердившая договор: _____

_____.

Наименование работы: Автоматизированная информационная система «Склад оптовой торговли».

2. Назначение разработки

Автоматизированная информационная система «Склад оптовой торговли» предназначена для обобщения информации о движении и наличии товаров, приобретенных для оптовой торговли. Пользователями программы выступают менеджеры склада, отдел учета, отдел приема и оформления заказов. Приобретение товаров от поставщиков осуществляется на основании договоров купли-продажи, в которых оговариваются условия поставки. Данные первичных документов по приходу товаров обобщаются в журнале поступления товаров, содержащем название приходного документа, его дату и номер, краткую характеристику документа, дату регистрации документа, сведения о поступивших товарах. Оформление и учет реализации товаров зависят от способа расчета за приобретаемые товары между покупателем и продавцом. Товары реализуются за наличный и безналичный расчет. Менеджер ведет журнал учета отпуска товаров, где указывается: номер по порядку, дата отпуска, наименование товаров, количество единиц и сумма отпуска, фамилия, инициалы и подпись лица, выдавшего товар. Данные первичных документов фиксируются в карточках учета, которые выполняют роль регистров складского и бухгалтерского учета. Бухгалтер не реже одного раза в неделю осуществляет проверку записей в книгах или карточках. Основанием для отражения в бухгалтерском учете операций по поступлению и выбытию товаров являются товарные отчеты с приложенными к ним документами.

3. Требования к программе

3.1. Требования к функциональным характеристикам

Автоматизированная информационная система «Склад оптовой торговли» должна обеспечивать выполнение функций:

- ввод, хранение, поиск и обработку информации по приходу и реализации товаров на складе;
- ведение журнала регистрации приходных и расходных документов;
- своевременное получение информации о наличии товаров на складе;
- формирование отчетов, необходимых менеджеру и бухгалтеру, содержащих все данные о поступлении и реализации товаров.

Нормативно-справочная информация автоматизированной информационной системы «Склад оптовой торговли» представ-

лена справочниками контрагентов, номенклатуры, единиц измерения.

Первичные документы для учета товаров в оптовой торговле:

- расходные накладные, кассовые и товарные чеки, содержащие дату реализации, перечень реализуемых товаров, их количество, цену и общую сумму продажи. Расходные накладные заполняются на основании заказов на товар;
- документы на поступление товаров от поставщика, содержащие следующую информацию: дата поступления товаров на склад, сведения о поставщике, перечень поступивших товаров, количество, цена и общая сумма.

Выходными данными являются следующие виды отчетов:

- отчет о поступлении товаров за определенный период, содержащий сведения о поставщиках, перечень поступивших товаров, их количество, цену, сумму поступления по каждому наименованию товара и общую сумму поступления;
- отчет о продажах за определенный период, содержащий перечень реализованных товаров, их цену, количество, сумму продаж по каждому наименованию проданных товаров и общую сумму реализации;
- инвентарная карточка, которая содержит данные о поступлении и реализации каждого товара.

В программе необходимо предусмотреть возможность корректировки настроек системы; резервное сохранение данных; возможность изменения пароля входа в систему; наличие встроенной справочной системы; быстрый поиск необходимых документов и справочной информации и т. д.;

3.2. Требования к надежности

Разрабатываемое программное обеспечение должно иметь:

- возможность самовосстановления после сбоев (отключения электропитания, сбои в операционной системе и т. д.);
- парольную защиту при запуске программы;
- ограничение несанкционированного доступа к данным;
- возможность резервного копирования информационной базы;
- разграничение пользовательских прав;

- исключение несанкционированного копирования (тиражирования) программы.

Предусмотреть контроль вводимой информации и блокировку некорректных действий пользователя при работе с системой.

3.3. Требования к составу и параметрам технических средств

Системные требования для работы программного продукта должны быть следующими: тактовая частота процессора ~1 200 Гц; объем оперативной памяти 64 Мб; объем свободного дискового пространства 50 Мб; разрешение монитора 1 024×768; наличие устройства чтения компакт-дисков.

3.4. Требования к информационной и программной совместимости

Программа должна работать в операционных системах Windows 2000/XP. Все формируемые отчеты должны иметь возможность экспортирования в редактор электронных таблиц MS Office Excel 2003/2007.

3.5. Требования к транспортированию и хранению

Программа поставляется на лазерном носителе информации. Программная документация поставляется в электронном и печатном виде.

3.6. Специальные требования

Программное обеспечение должно иметь дружественный интерфейс, рассчитанный на пользователя средней квалификации (с точки зрения компьютерной грамотности).

Ввиду объемности проекта задачи предполагается решать поэтапно. При этом модули программного обеспечения (ПО), созданные в разное время, должны предполагать возможность наращивания системы и быть совместимы друг с другом; поэтому документация на принятое эксплуатационное ПО должна содержать полную информацию, необходимую для работы с ним программистов. Язык программирования определяется выбором исполнителя, при этом он должен обеспечивать возможность интеграции программного обеспечения с пакетом MS Office 2003/2007.

4. Требования к программной документации

В ходе разработки программы должны быть подготовлены: текст программы, описание программы, программа и методика испытаний, руководство пользователя, технико-экономическое обоснование.

При выполнении операций по регистрации поступления товаров на склад и их отгрузки со склада используется ручной труд, а именно ведутся книги прихода и расхода товаров. Очевидно, что использование программы значительно сократит время, затрачива-

емое на регистрацию товара. Кроме того, на складе для каждой единицы товара существует инвентарная карточка; в нее тоже заносятся сведения о поступлении, расходовании и остатке данного товара на настоящий момент времени. Для получения этих сведений по конкретному товару требуется не менее 8—10 мин. С использованием программы затраты времени сокращаются до 1—2 мин. В конце каждого месяца ответственный работник склада составляет отчеты об оборотах товара на складе и выводит остаток товаров по каждой позиции. На эту операцию уходит 1—2 дня, т.е. 6—12 ч. Формирование оборотной ведомости в компьютере займет 3—4 мин. Кроме того, предполагается возможность получения отчетов за любой период времени. При ручном создании отчетов человеком могут быть допущены ошибки; правильно составленный алгоритм разрабатываемой программы ошибки исключает.

Экономический эффект от внедрения автоматизированной информационной системы «Склад оптовой торговли» ожидается за счет сокращения времени на выполняемые менеджерами операции, исключения ошибок при формировании отчетов, увеличения времени на анализ хозяйственной деятельности и т.д.

1.2.2. Техническое задание на разработку системы решения комбинаторных задач

Введение

Настоящее техническое задание распространяется на разработку системы решения комбинаторно-оптимизационных задач, предназначенной для ввода и хранения данных указанных задач, а также для их решения, хранения полученных результатов и использования разработчиками программных и аппаратных средств вычислительной техники.

Широкий круг задач проектирования различного рода технических объектов, в том числе компьютеров, относится к классу комбинаторно-оптимизационных задач, точные методы решения которых, как правило, имеют экспоненциальную вычислительную сложность и нереализуемы даже на современных компьютерах. В настоящее время для решения таких задач широко используются приближенные методы и алгоритмы, требующие различных вычислительных ресурсов и дающие неодинаковую точность решения.

При этом данные методы и алгоритмы не систематизированы, оценки их вычислительной и емкостной сложности и сведения о возможной точности получаемых решений не полны и разбросаны по многим источникам. В рамках единой системы не суще-

ствует программной реализации даже для ограниченного круга алгоритмов решения основных комбинаторно-оптимизационных задач проектирования.

Создание системы, в рамках которой были бы реализованы наиболее часто упоминаемые методы и алгоритмы решения указанных задач, позволит как оценивать и исследовать отдельные методы и алгоритмы, так и сравнивать их с точки зрения затрат вычислительных ресурсов и точности получаемых решений.

1. Основание для разработки

Система разрабатывается на основании приказа заместителя директора по учебной работе ... № ... от .... и в соответствии с учебным планом на 200___—200___ г.

2. Назначение разработки

Система призвана решить небольшой круг комбинаторно-оптимизационных задач на графах:

поиск цикла минимальной длины (задача коммивояжера);

поиск кратчайшего пути;

поиск минимального связывающего дерева.

Пользователями могут выступать научные работники и инженеры, занимающиеся проектированием компьютеров, студенты соответствующих специальностей, а также специалисты других предметных областей, которым приходится решать подобные задачи.

3. Требования к программе или программному изделию

3.1. Требования к функциональным характеристикам

Система должна представлять совокупность методических и программных средств решения следующих задач:

- построение минимального покрывающего дерева;
- поиск покрывающего цикла минимальной длины (задача коммивояжера);
- поиск кратчайшего пути.

Для этих задач должны быть реализованы:

- алгоритм, обеспечивающий получение точного решения;
- в случае если точное решение дает алгоритм, имеющий не полиномиальную вычислительную сложность, то необходимо дополнительно разработать алгоритм, обеспечивающий получение приближенных решений с полиномиальной вычислительной сложностью.

Методическое обеспечение должно быть реализовано в пользовательском интерфейсе системы, который должен предполагать:

- выбор задачи, метода и алгоритма ее решения;
- ввод данных;
- решение проектной задачи и сохранение исходных данных, промежуточных и окончательных результатов во встроенной базе данных для последующего анализа.

3.2. Требования к надежности

Программный продукт должен соответствовать современному уровню требований к надежности программного обеспечения:

- предусматривать контроль вводимой информации и блокировку некорректных действий пользователя при работе с системой;
- обеспечивать корректное завершение вычислений с соответствующей диагностикой при превышении имеющихся вычислительных ресурсов;
- обеспечивать целостность информации, хранящейся в базе данных.

3.3. Требования к составу и параметрам технических средств

Системные требования для работы программного продукта должны быть следующими: тактовая частота процессора ~1 000 Гц; объем оперативной памяти 64 Мб; объем свободного дискового пространства 20 Мб; разрешение монитора 1 024×768; наличие устройства чтения компакт-дисков; принтер.

3.4. Требования к информационной и программной совместимости

Программа должна работать в операционных системах Windows 2000/XP.

4. Требования к программной документации

Разрабатываемая система должна включать справочную информацию о работе системы и подсказки пользователю. В состав сопровождающей документации должны входить: расчетно-пояснительная записка, содержащая описание системы; руководство пользователя; руководство системного программиста.

5. Этапы разработки (табл. 1.1)

После утверждения технического задания организация-разработчик непосредственно приступает к созданию программного обеспечения.

Задание

Разработать документ «Техническое задание» на программный продукт в соответствии с одним из вариантов, представленных в приложении. Оформить работу в соответствии с ГОСТ 19.106—78.

Таблица 1.1. Этапы разработки

Номер этапа	Название этапа	Срок	Отчетность
1	Разработка ядра системы	1.1.2008—31.3.2008	Описание внутренних форматов, интерфейса и форматов данных базы. Реализация системы на уровне интерфейса
2	Разработка методов и алгоритмов и их реализация для задачи коммивояжера	1.4.2008—30.6.2008	Описание методов и алгоритмов. Программные модули, реализующие методы
3	Разработка методов и алгоритмов и их реализация для задачи построения минимального связывающего дерева и задачи поиска кратчайшего пути в графе	1.7.2008—30.9.2008	Описание методов и алгоритмов. Программные модули, реализующие методы
4	Тестирование программного продукта и составление программной документации	1.10.2008—31.12.2008	Тесты. Документация. Программный продукт

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Перечислите основные этапы разработки программных продуктов.
2. Что включают в себя предпроектные исследования?
3. Назовите основные разделы технического задания.
4. Перечислите функциональные и эксплуатационные требования к программному продукту. В чем их принципиальное различие?
5. Приведите основные разделы документа «Техническое задание».

Глава 2

ПРИМЕНЕНИЕ СТРУКТУРНОГО ПОДХОДА В АНАЛИЗЕ ТРЕБОВАНИЙ И ОПРЕДЕЛЕНИИ СПЕЦИФИКАЦИЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

2.1. ОСНОВНЫЕ СВЕДЕНИЯ

Собственно разработка любого программного обеспечения начинается с анализа требований к будущему программному продукту. В результате анализа получают спецификации разрабатываемого программного обеспечения: выполняют декомпозицию и содержательную постановку решаемых задач, уточняют их взаимодействие и определяют эксплуатационные ограничения. В процессе определения спецификаций строят общую модель предметной области как некоторой части реального мира, с которой будет тем или иным способом взаимодействовать разрабатываемое программное обеспечение, и конкретизируют его основные функции.

Спецификации представляют собой полное и точное описание функций и ограничений разрабатываемого программного обеспечения. При этом функциональные спецификации описывают функции разрабатываемого программного обеспечения, а эксплуатационные определяют требования к техническим средствам, надежности, безопасности и т.д. Применительно к функциональным спецификациям требование *полноты* означает, что спецификации должны содержать всю существенную информацию, чтобы ничто важное не было упущено, и не должны содержать несущественной информации, например деталей реализации, чтобы не препятствовать разработчику в выборе наиболее эффективных решений. Требование *точности* означает, что спецификации должны однозначно восприниматься как заказчиком, так и разработчиком.

Последнее требование выполнить достаточно сложно, так как естественный язык для описания спецификаций не подходит: подробные спецификации на естественном языке не обеспечивают необходимой точности. Точные спецификации разрабатывае-

мого программного обеспечения можно определить, только разработав некоторую *формальную модель* этого программного обеспечения.

Все функциональные спецификации разрабатываемого программного обеспечения описывают перечень функций и состав обрабатываемых данных. Они различаются только системой приоритетов (акцентов), которая используется разработчиком в процессе анализа требований и определения спецификаций. Так, диаграммы переходов состояний определяют некоторые аспекты поведения программного обеспечения во времени, диаграммы потоков данных — направление и структуру потоков данных, а концептуальные диаграммы классов — отношение между основными понятиями предметной области. На рис. 2.1 показана классификация моделей, используемых в качестве спецификаций разрабатываемого программного обеспечения.

В рамках структурного подхода на этапе анализа и определения спецификаций используют три типа моделей: ориентированные на функции, ориентированные на данные и ориентированные на потоки данных.

Каждый тип модели целесообразно использовать для своего специфического класса программных разработок. Разные модели описывают проектируемое программное обеспечение с разных сторон. Методологии структурного анализа и проектирования, основанные на моделировании потоков данных, обычно использу-

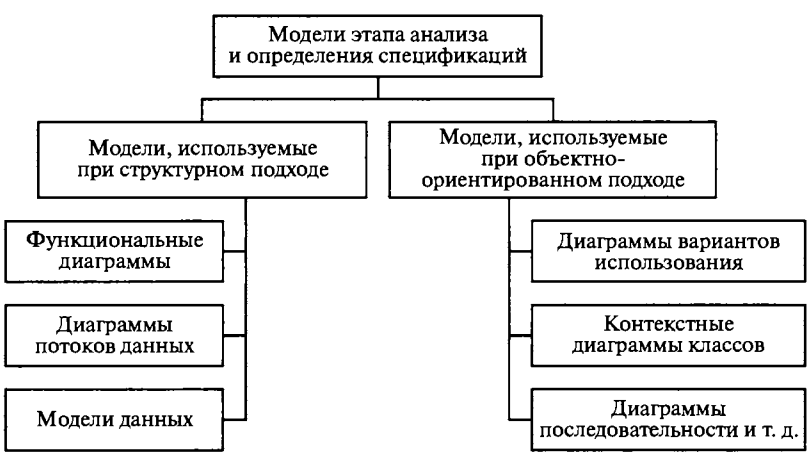


Рис. 2.1. Классификация моделей программного обеспечения, используемых на этапе определения спецификаций

Рис. 2.2. Компоненты полной спецификации методологий структурного анализа и проектирования программного обеспечения, основанных на потоках данных

ют комплексное представление проектируемого программного обеспечения в виде совокупности следующих моделей:

- диаграммы переходов состояний (SDT — State Transition Diagrams), характеризующие поведение системы во времени;
- функциональные диаграммы (SADT — Structured Analisis and Design Technique);
- диаграммы потоков данных (DFD — Data Flow Diagrams), описывающие взаимодействие источников и потребителей информации через процессы, которые должны быть реализованы в системе;
- диаграммы «сущность — связь» (ERD — Entity — Relationship Diagrams), описывающие базы данных разрабатываемой системы.

Компоненты спецификации методологий структурного анализа приведены на рис. 2.2.

Спецификацию процесса обычно представляют в виде краткого текстового описания, схем алгоритмов, псевдокодов.

Словарь данных — это краткое описание основных понятий, используемых при составлении спецификаций. Словарь должен включать в себя определение основных понятий предметной области, описание структур элементов данных, их типов и форматов, а также всех сокращений и условных обозначений. Словарь предназначен для повышения степени понимания предметной

области и исключения риска возникновения разногласий при обсуждении моделей между заказчиками и разработчиками.

2.2. ДИАГРАММЫ ПЕРЕХОДОВ СОСТОЯНИЙ

Характеристика диаграмм переходов состояний (SDT). Диаграммы SDT демонстрируют поведение разрабатываемой программной системы при получении управляющих воздействий. Под *управляющими воздействиями*, или *сигналами*, в данном случае понимают управляющую информацию, получаемую системой извне; например, управляющими воздействиями считают команды пользователя и сигналы датчиков, подключенных к компьютерной системе. Получив такое управляющее воздействие, разрабатываемая система должна выполнить определенные действия, а затем или остаться в том же состоянии, или перейти в другое состояние, зафиксировав некоторые изменения в системе.

Главное предназначение этой диаграммы — описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла. Диаграмма переходов состояний представляет динамическое поведение сущностей на основе спецификации их реакции на восприятие некоторых конкретных событий. Системы, которые реагируют на внешние воздействия других систем или пользователей, иногда называют реактивными. Если такие воздействия инициируются в произвольные случайные моменты времени, то говорят об асинхронном поведении модели.

Несмотря на то, что диаграммы чаще всего используются для описания поведения отдельных экземпляров классов (объектов), они также могут применяться для спецификации функциональности других компонентов моделей, таких как варианты использования, действующие лица, подсистемы, операции и методы.

Для построения диаграммы переходов состояний необходимо в соответствии с теорией конечных автоматов определить основные состояния, управляющие воздействия (или условия перехода), выполняемые действия и возможные переходы разрабатываемого программного обеспечения. Условные обозначения, используемые при построении диаграмм переходов состояний, показаны на рис. 2.3.

Диаграмма переходов состояний, по существу, представляет собой граф специального вида, который обозначает некоторый

Рис. 2.3. Условные обозначения диаграмм переходов состояний:
а — начальное и конечное состояния; *б* — промежуточное состояние; *в* — переход

автомат. Понятие *автомат* в контексте UML обладает довольно специфической семантикой, основанной на теории автоматов. Вершинами этого графа являются состояния и некоторые другие типы элементов автомата (псевдосостояния), которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние. Простейшим примером визуального представления состояний и переходов на основе формализма автоматов может служить ситуация с исправностью технического устройства, такого как компьютер. В этом случае вводятся в рассмотрение два самых общих состояния: «исправен» и «неисправен» и два перехода: «выход из строя» и «ремонт». Графически эта информация может быть представлена в виде диаграммы состояний компьютера (рис. 2.4).

Основными понятиями, входящими в формализм автомата, являются *состояние* и *переход*. Главное различие между ними в том, что время 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 допол-

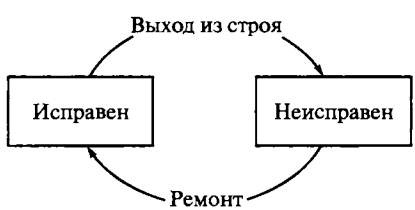


Рис. 2.4. Простейший пример диаграммы переходов состояний для технического устройства типа компьютер

нительно ничего не сказано). Другими словами, переход объекта из состояния в состояние происходит мгновенно.

Для понимания семантики конкретной диаграммы состояний необходимо представлять не только особенности поведения моделируемой сущности, но и знать общие сведения по теории автоматов.

Сторожевое условие (guard condition), если оно есть, всегда записывается в прямых скобках после события-триггера и представляет собой некоторое булевское выражение. Напомним, что булевское выражение должно принимать одно их двух взаимно исключающих значений: «истина» или «ложь». Из контекста диаграммы состояний должна явно следовать семантика этого выражения. Если сторожевое условие принимает значение «истина», то соответствующий переход может сработать, в результате чего объект перейдет в целевое состояние. Если же сторожевое условие принимает значение «ложь», то переход не может сработать и при отсутствии других переходов объект не может перейти в целевое состояние по этому переходу. Однако вычисление истинности сторожевого условия происходит только после возникновения связанного с ним события.

Для интерактивного программного обеспечения с развитым пользовательским интерфейсом основными управляющими воздействиями выступают команды пользователя, для программного обеспечения реального времени — сигналы от датчиков и (или) оператора производственного процесса. Общим для этих типов программного обеспечения является наличие состояния ожидания, когда система приостанавливает работу до получения очередного управляющего воздействия. Для интерактивного программного обеспечения наиболее характерно получение команд различных типов, а для программного обеспечения реального времени — однотипных сигналов либо от многих датчиков, либо требующих продолжительной обработки.

В отличие от интерактивных систем для систем реального времени обычно установлено более жесткое ограничение на время обработки полученного сигнала. Такое ограничение часто требует выполнения дополнительных исследований поведения системы во времени.

К программному обеспечению, при разработке которого требуется уточнение особенностей поведения посредством построения диаграммы переходов состояний, относится и программное обеспечение, ориентированное на работу в сети. При этом обычно отдельно строят модели поведения сервера и клиента, представляя сообщения, передаваемые между ними в виде управляющих воздействий.

Рис. 2.5. Диаграмма переходов состояний объекта «Заказ»

Пример построения диаграммы переходов состояний. На предлагаемой диаграмме (рис. 2.5) описываются возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение объекта «Заказ» автоматизированной информационной системы «Склад оптовой торговли» в течение его существования (поступление, обработка, формирование поставки). На ней отображаются функции, которые выполняются объектом «Заказ» в определенном состоянии. Метки деятельности имеют следующий синтаксис: *выполнить/< деятельность >* (например, выполнить/проверить строку). Переходы имеют метки, которые синтаксически состоят из трех необязательных частей: *<Событие> <[Условие]> </Действие>* (например, [не все строки проведены]/получить следующую строку).

2.3. ФУНКЦИОНАЛЬНЫЕ ДИАГРАММЫ

Характеристика функциональных диаграмм (SADT) — диаграммы SADT отражают взаимные связи функций разрабатывае-

мого программного обеспечения. Они создаются на ранних стадиях проектирования систем, для того чтобы помочь проектировщику выявить основные функции и составные части проектируемой программной системы и, по возможности, обнаружить и устранить существенные ошибки. Для создания функциональных диаграмм предлагается использовать методологию SADT, предложенную Д.Россом. На основе методологии SADT была построена известная методология описания сложных систем IDEFO (Icam DEFinition), являющаяся основной частью программы ICAM (интегрированная компьютеризация производства), проводимой по инициативе ВВС США. Применение методологии SADT позволяет построить модель, состоящую из диаграмм, фрагментов текстов и глоссария, имеющих ссылки друг на друга. Диаграммы — главные компоненты модели. Функции системы и интерфейсы представлены на диаграммах в виде блоков и дуг. Место соединения дуги с блоком определяет тип интерфейса. Управляющая информация входит в блок сверху, в то время как информация, которая подвергается обработке, показана с левой стороны блока, а результаты выхода даны с правой стороны. Механизм (человек или автоматизированная система), осуществляющий операцию, представлен дугой, входящей в блок снизу (рис. 2.6).

Одна из наиболее важных особенностей методологии SADT — постепенное введение все больших уровней детализации по мере создания диаграмм, отображающих модель. Построение SADT-модели начинается с представления всей системы в виде простейшей компоненты — одного блока и дуг, изображающих интерфейсы с функциями вне системы. Поскольку единственный блок представляет всю систему как единое целое, имя, указанное в блоке, является общим. Это верно и для интерфейсных дуг — они также представляют собой полный набор внешних интерфейсов системы в целом.

Затем блок, в котором система дана в качестве единого модуля, детализируется на другой диаграмме с помощью нескольких

Рис. 2.6. Функциональный блок и интерфейсные дуги

блоков, соединенных интерфейсными дугами. Эти блоки представляют основные подфункции исходной функции. Данная декомпозиция выявляет полный набор подфункций, каждая из которых представлена как блок, границы которого определены интерфейсными дугами. Каждая из этих подфункций может быть декомпозирована подобным образом для более детального представления. Во всех случаях каждая подфункция может содержать только те элементы, которые входят в исходную функцию. Кроме того, модель не может опустить какие-либо элементы, т.е., как уже отмечалось, родительский блок и его интерфейсы обеспечивают контекст. К нему нельзя ничего добавить, и из него не может быть ничего удалено.

Модель SADT представляет собой серию диаграмм с сопроводительной документацией, разбивающих сложный объект на составные части, представленные в виде блоков. Детали каждого из основных блоков показаны в виде блоков на других диаграммах. Каждая детальная диаграмма является декомпозицией блока из более общей диаграммы. На каждом шаге декомпозиции более общая диаграмма называется *родительской* для более детальной диаграммы.

Дуги, входящие в блок и выходящие из него на диаграмме верхнего уровня, являются точно теми же самыми, что и дуги, входящие в диаграмму нижнего уровня и выходящие из нее, потому что блок и диаграмма представляют одну и ту же часть системы.

Стрелки, приходящие с родительской диаграммы или уходящие на нее, нумеруют, используя символы и числа. Символ обозначает тип связи: *I* — входные, *C* — управляющие, *M* — механизмы, *R* — результаты. Число — номер связи по соответствующей стороне родительского блока, считая сверху вниз и слева направо. Все диаграммы связывают друг с другом иерархической нумерацией блоков: начальный уровень — *A*0, следующий — *A*1, *A*2 и т.п., следующий — *A*11, *A*12, *A*13 и т.д., где первая цифра — номер родительского блока, а последняя — номер конкретного субблока родительского блока. Детализацию завершают при получении функций, назначение которых хорошо понятно как заказчику, так и разработчику. Эти функции описывают, используя естественный язык или псевдокоды. В процессе построения иерархии диаграмм фиксируют всю уточняющую информацию и строят словарь данных, в котором определяют структуры и элементы данных, показанных на диаграммах. Таким образом, в результате получают спецификацию, которая состоит из иерархии функци-

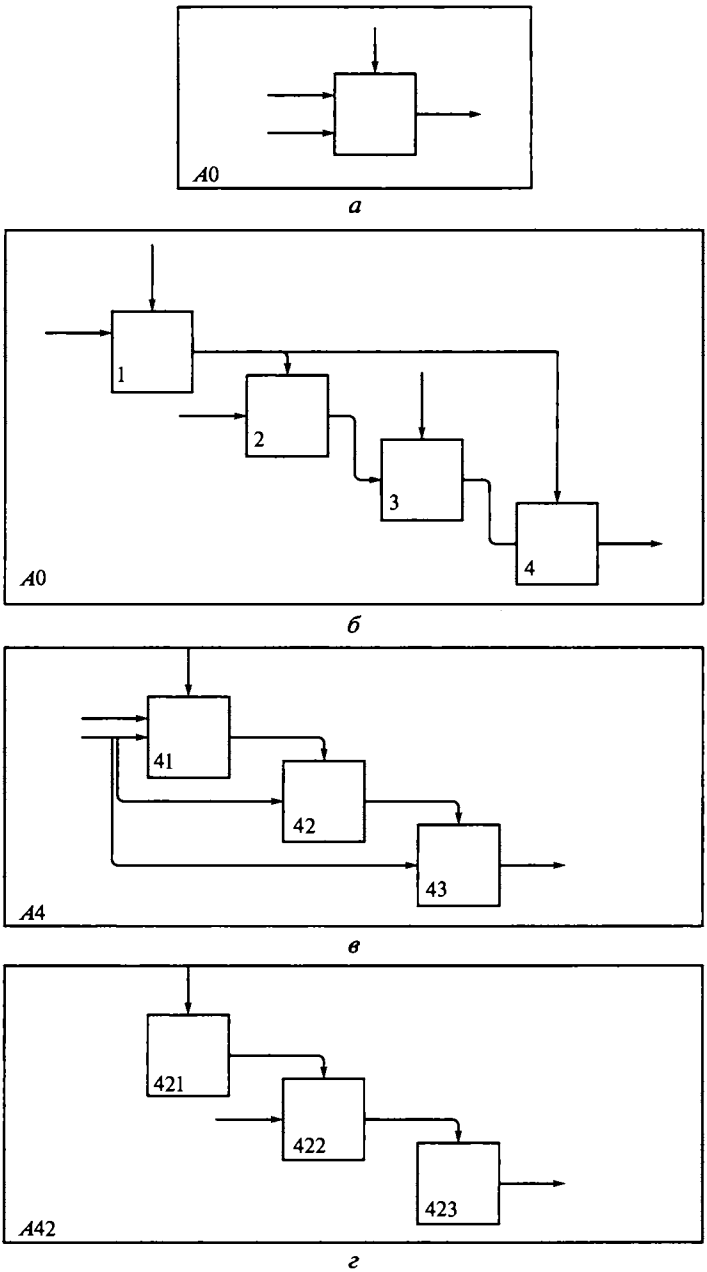


Рис. 2.7. Структура SADT-модели. Декомпозиция диаграмм

ональных диаграмм, описаний функций нижнего уровня и словаря, имеющих ссылки друг на друга.

На рис. 2.7 представлена декомпозиция четырех диаграмм, показывающая структуру SADT-модели, общее (рис. 2.7, *а*) и детальное представление блока *A*0 (рис. 2.7, *б*), декомпозицию блоков *A*4 (рис. 2.7, *в*) и *A*42 (рис. 2.7, *г*). Не присоединяемые дуги соответствуют входам, управлениям и выходам родительского блока. Источник или получатель этих дуг может быть обнаружен только на родительской диаграмме. Не присоединяемые концы должны соответствовать дугам на исходной диаграмме. Все граничные дуги должны продолжаться на родительской диаграмме, чтобы она была полной и непротиворечивой.

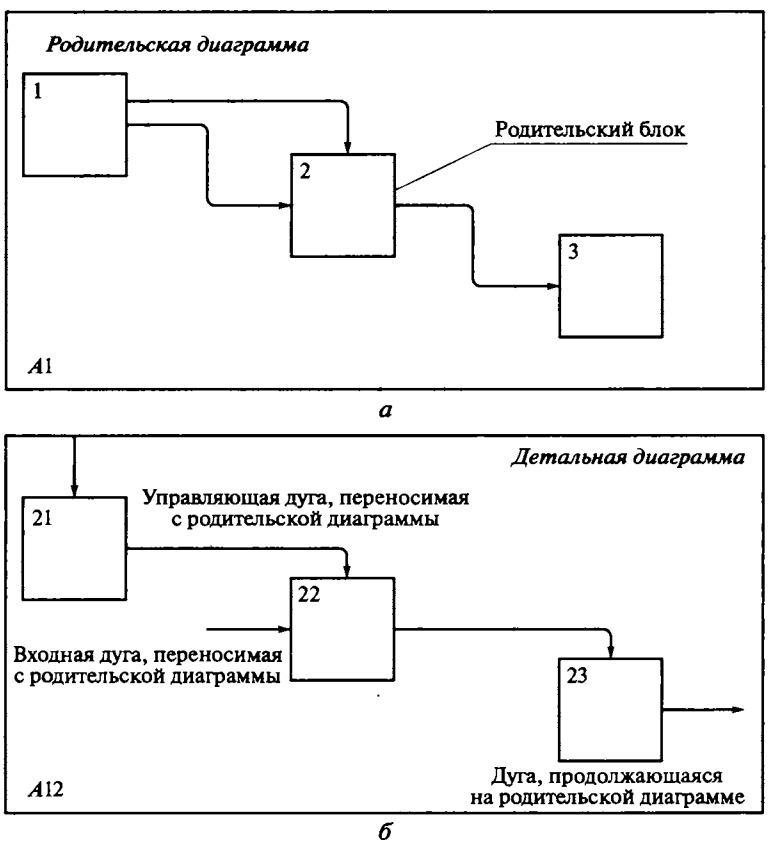


Рис. 2.8. Соответствие интерфейсных дуг родительской (*а*) и детальной (*б*) диаграмм

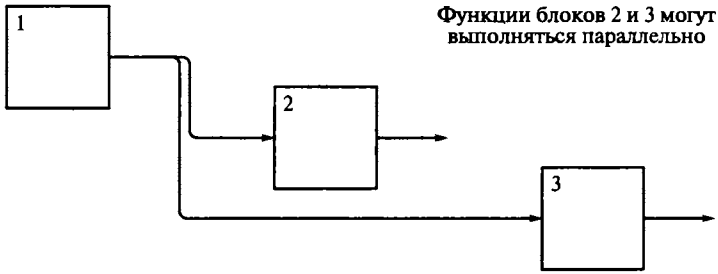


Рис. 2.9. Пример одновременного выполнения функций

Каждый компонент модели может быть декомпозирован на другой диаграмме, т.е. каждая диаграмма иллюстрирует «внутреннее строение» блока на родительской диаграмме.

На рис. 2.8, *а*, *б* и рис. 2.9 представлены различные варианты выполнения функций и соединения дуг с блоками.

На SADT-диаграммах не указаны явно ни последовательность, ни время. Обратные связи, итерации, продолжающиеся процессы и перекрывающиеся (по времени) функции могут быть изображены также с помощью дуг. Обратные связи могут выступать в виде комментариев, замечаний, исправлений и т.д. (рис. 2.10).

Одним из важных моментов при моделировании системы с помощью методологии SADT является точная согласованность типов связей между функциями. Различают, по крайней мере, следующие типы связей: случайная; логическая; временна́я; процедурная; коммуникационная; последовательная; функциональная.

Случайная связь возникает, когда конкретная связь между функциями мала или полностью отсутствует. Это относится к си-

Рис. 2.10. Пример обратной связи

туации, когда имена данных на SADT-дугах в одной диаграмме имеют малую связь друг с другом.

Логическая связь — данные и функции собираются вместе, поскольку попадают в общий класс или набор элементов, но необходимых функциональных отношений между ними при этом не обнаруживается.

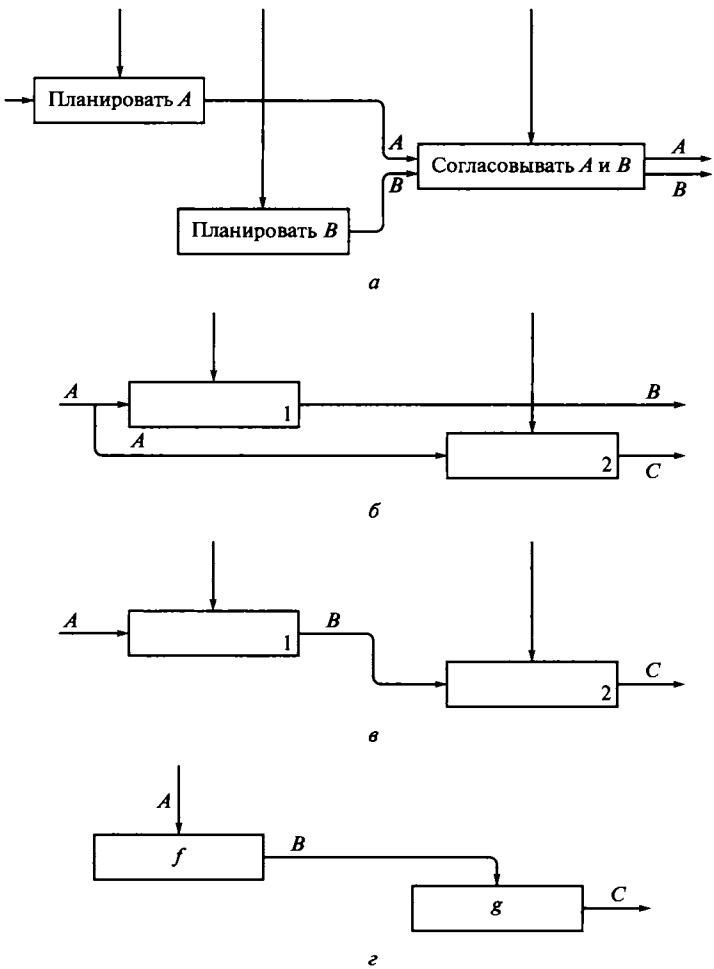


Рис. 2.11. Типы связей:

а — процедурная; *б* — коммуникационная; *в* — последовательная; *г* — функциональная

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

При *процедурной* связи функции сгруппированы вместе, поскольку они выполняются в течение одной и той же части цикла или процесса (рис. 2.11, *а*).

Диаграммы демонстрируют *коммуникационную* связь, когда блоки группируются вследствие того, что они используют одни и те же входные данные и (или) производят одни и те же выходные данные (рис. 2.11, *б*).

При *последовательной* связи выходные данные одной функции служат входными данными для другой функции. Связь между элементами на диаграмме является более тесной, чем на рассмотренных выше уровнях, поскольку моделируются причинно-следственные зависимости (рис. 2.11, *в*).

Диаграмма отражает полную *функциональную* связь при наличии полной зависимости одной функции от другой. Диаграмма, которая является чисто функциональной, не содержит чужеродных элементов, относящихся к последовательному или более слабому типу связности. Одним из способов определения функционально связанных диаграмм является рассмотрение двух блоков, связанных через управляющие дуги, как показано на рис. 2.11, *г*.

В математических терминах необходимое условие для простейшего типа функциональной связности (рис. 2.11, *г*) имеет следующий вид: $C = g(B) = g(f(A))$.

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

Пример разработки функциональной диаграммы программы построения графиков. Разработку функциональных диаграмм продемонстрируем на примере уточнения спецификаций программы построения графиков и таблиц функций одной переменной.

На рис. 2.12, *а* показана диаграмма верхнего уровня, на которой хорошо видно, что́ является исходными данными для программы и получения каких результатов мы ожидаем.

Диаграмма на рис. 2.12, *б* уточняет функции программы. На ней показаны четыре блока: ввод — выбор и ее разбор, добавление функции в список, построение таблицы значений и построе-

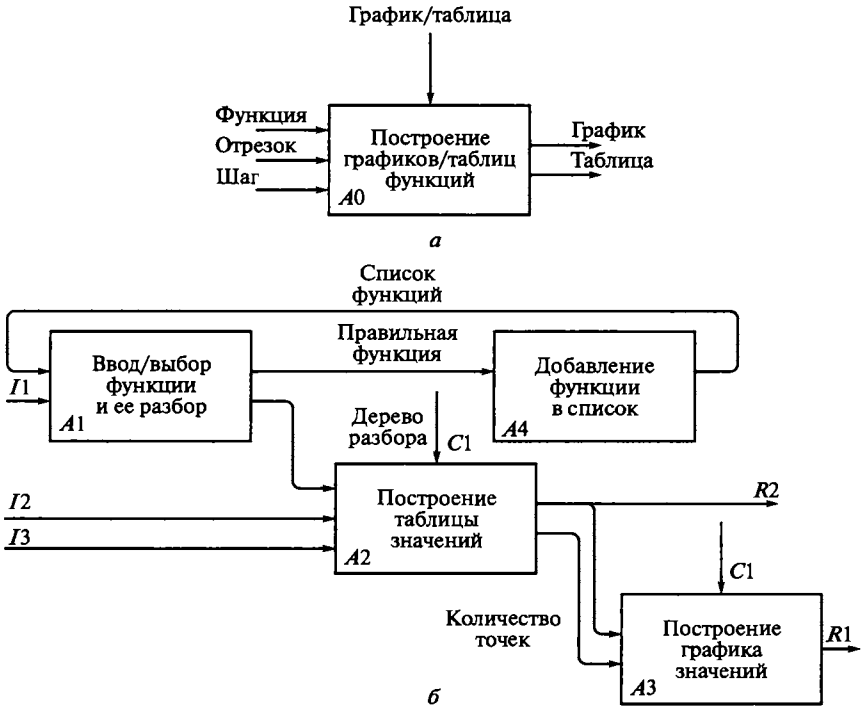


Рис. 2.12. Функциональные диаграммы для системы исследования функций:

а — диаграмма верхнего уровня; *б* — уточняющая диаграмма

ние графика функции. Для каждого блока определены исходные данные, управляющие воздействия и результаты. Согласно правилам обозначения входов — выходов, имеющих продолжение на родительской диаграмме, на данной диаграмме использованы следующие обозначения: *I*1 — функция; *I*2 — отрезок; *I*3 — шаг; *C*1 — вид график — таблица; *R*1 — график функции на отрезке; *R*2 — таблица значений функции на отрезке.

Словарь в этом случае должен содержать описание всех данных, используемых в системе.

Функциональную модель целесообразно применять для определения спецификаций программного обеспечения, не предусматривающего работу со сложными структурами данных, так как она ориентирована на декомпозицию функций.

2.4. ДИАГРАММЫ ПОТОКОВ ДАННЫХ

Характеристика диаграмм потоков данных (DFD). Диаграмма DFD состоит из узлов обработки данных, средств их хранения и внешних по отношению к используемой диаграмме источников или потребителей данных.

Диаграмма потоков данных — основное средство моделирования функциональных требований к системе, проектируемой или реально существующей. В основе модели лежат понятия внешней сущности, процесса, хранилища (накопителя) данных потока данных. Источники информации (внешние сущности) порождают информационные потоки (потоки данных), переносящие информацию к подсистемам или процессам; те, в свою очередь, преобразуют информацию и порождают новые потоки, которые переносят информацию к другим процессам или подсистемам, накопителям данных или внешним сущностям — потребителям информации.

Для изображения диаграмм потоков данных традиционно используют два вида нотаций — Йордана и Гейна—Сарсона, которые представлены в табл. 2.1.

Внешняя сущность — это материальный предмет или физическое лицо, представляющее собой источник или приемник информации, например, заказчики, персонал, поставщики, клиенты,

Таблица 2.1. Виды нотаций

Понятие	Нотация Йордана	Нотация Гейна — Сарсона
Внешняя сущность	Наименование	Номер Наименование
Процесс, система, подсистема	Наименование Номер	Номер Наименование Механизм
Накопитель данных	Наименование	№ Наименование
Поток данных	Наименование →	Наименование →

склад. Определение некоторого объекта или системы в качестве внешней сущности указывает на то, что этот объект или эта система находятся за пределами границ анализируемой информационной системы. В процессе анализа некоторые внешние сущности могут быть перенесены внутрь диаграммы анализируемой информационной системы, если это необходимо, или, наоборот, часть процессов информационной системы может быть вынесена за пределы диаграммы и представлена как внешняя сущность.

При построении модели сложной информационной системы она может быть представлена в самом общем виде на так называемой *контекстной диаграмме*. На такой диаграмме показывают, как разрабатываемая система будет взаимодействовать с потребителями и источниками информации, т.е. описывают интерфейс системы с внешним миром. Система может быть представлена как единое целое либо может быть декомпозирована на ряд подсистем. Номер подсистемы служит для ее идентификации. В поле имени вводится наименование подсистемы в виде предложения с подлежащим и соответствующими определениями и дополнениями.

Процесс представляет собой преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом. Физически процесс может быть реализован различными способами: это может быть подразделение организации (отдел), выполняющее обработку входных документов и выпуск отчетов, программа, аппаратно реализованное логическое устройство и т.д. Номер процесса служит для его идентификации. В поле имени вводится наименование процесса, при этом использование таких глаголов, как обработать, модернизировать или отредактировать означает, как правило, недостаточно глубокое понимание данного процесса и требует дальнейшего анализа. Информация в поле физической реализации показывает, какое подразделение организации, программа или аппаратное устройство выполняют данный процесс.

Накопитель данных представляет собой абстрактное устройство для хранения информации, которую можно в любой момент поместить в накопитель и через некоторое время извлечь; причем способы помещения и извлечения могут быть любыми. Накопитель данных может быть реализован физически в виде ящика в картотеке, таблицы в оперативной памяти, файла на магнитном носителе и т.д.

В нотации Гейна—Сарсона накопитель данных идентифицируется буквой *D* и произвольным числом. Имя накопителя выбирается из соображения наибольшей информативности для проектировщика.

Накопитель данных в общем случае выступает прообразом будущей базы данных, и описание хранящихся в нем данных должно быть увязано с информационной моделью.

Поток данных определяет информацию, передаваемую через некоторое соединение от источника к приемнику. Реальный поток данных может быть информацией, передаваемой по кабелю между двумя устройствами, пересылаемыми по почте письмами, магнитными лентами или дискетами, переносимыми с одного компьютера на другой, и т.д. Каждый поток данных имеет имя, отражающее его содержание.

Построение иерархии диаграмм потоков данных начинают с контекстной диаграммы, которая определяет наиболее общий вид системы. Обычно начальная контекстная диаграмма имеет форму звезды. Если проектируемая система содержит большое количество внешних сущностей (более 10), имеет распределенную природу или включает уже существующие подсистемы, то строят иерархии контекстных диаграмм. В процессе детализации соблюдают правило балансировки — при детализации подсистемы могут использоваться компоненты только тех подсистем, с которыми у разрабатываемой подсистемы существует информационная связь (т.е. с которыми она связана потоками данных). На недетализируемые процессы составляют спецификации, которые должны содержать описание функций данного процесса. Такое описание может выполняться на естественном языке, с применением структурированного естественного языка (псевдокодов), с применением таблиц и деревьев решений, в виде схем алгоритмов.

Пример построения диаграммы потоков данных АИС «Склад оптовой торговли». Построение иерархии диаграмм потоков данных начнем с контекстной диаграммы, которая определяет наиболее общий вид системы. Таким образом, определим, как разрабатываемая система будет взаимодействовать с приемниками и источниками информации (рис. 2.13).

Автоматизированная информационная система «Склад оптовой торговли» предназначена для получения данных о движении и наличии товаров, приобретенных для оптовой продажи. Первичные документы по приходу товаров фиксируются в журнале поступления товаров. Оформление и учет реализации товаров зависят от способа расчета за приобретаемые товары между покупателем и продавцом. Менеджер склада ведет журнал учета закупок и отпуска товаров. Данные первичных документов сохраняются в соответствующих накопителях.

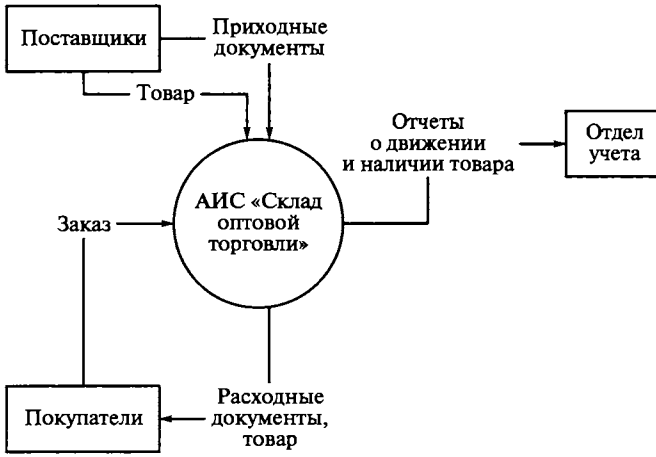


Рис. 2.13. Начальная контекстная диаграмма (диаграмма нулевого уровня) в нотации Йордана для АИС «Склад оптовой торговли»

Внешними сущностями для разрабатываемой системы являются поставщики, покупатели, менеджер склада, отдел учета и контроля, отдел приема и оформления заказов. Сведения о них сохраняются в соответствующих таблицах (справочниках). Поставщик передает товар на склад, документы на поставку товара (накладные, счета-фактуры) вводятся в базу данных. Покупатель подает заказ на приобретение товаров, в отделе приема и оформления заказов проверяется каждая строка поступившего заказа. При отсутствии на складе какой-либо позиции оформляется заявка поставщику, т.е. инициируется поставка необходимого товара. На основании заказа заполняются документы на реализацию товара, которые сохраняются в базе данных, распечатываются и выдаются покупателю. В конце каждого дня все первичные документы передаются менеджеру отдела учета (бухгалтеру). На основании сведений о приходе и реализации товара менеджеры склада и работники отдела учета формируют отчеты по оборотам и остаткам товара на складе.

Для построения диаграммы потоков данных нужно запустить MS Visio. Появится окно, в котором необходимо выбрать папку Shapes/Business Process. В открывшемся списке форм (Shapes) для построения диаграммы потоков данных следует выбрать пункт Data Flow Diagram Shapes. В результате проделанных действий на

экране появится окно, в левой части которого будет отображен набор графических символов, а в правой части — лист для рисования диаграммы. Выбрать нужный компонент можно щелчком мыши по соответствующей пиктограмме, после чего графический символ переносится на рабочее поле мышкой при нажатой правой клавише. Диаграмма потоков данных АИС «Склад оптовой торговли» приведена на рис. 2.14.

Пример построения диаграммы потоков данных подсистемы распределения студентов. Разработаем диаграмму потоков данных автоматизированной системы, предназначенной для сбора и хранения информации о студентах, обучающихся на старших курсах учебного заведения. Система предназначена для использования в образовательных учреждениях, с целью предоставления через Интернет потенциальным работодателям информации о выпускниках. Начальная контекстная диаграмма потоков данных в нотации Гейна—Сарсона приведена на рис. 2.15. Внешними сущностями в ней являются работодатель, администратор и студент.

Работодатель регистрируется в системе, вводит данные для поиска кандидатов и получает от системы результаты поиска. Администратор вводит информацию о студентах, которые сохраняются в базе данных. Студент может изменить свою контактную информацию, изменения также сохраняются в базе данных. Выбранным студентам работодатель посылает сообщение с предложением о работе. Диаграмма потоков данных системы распределения студентов приведена на рис. 2.16.

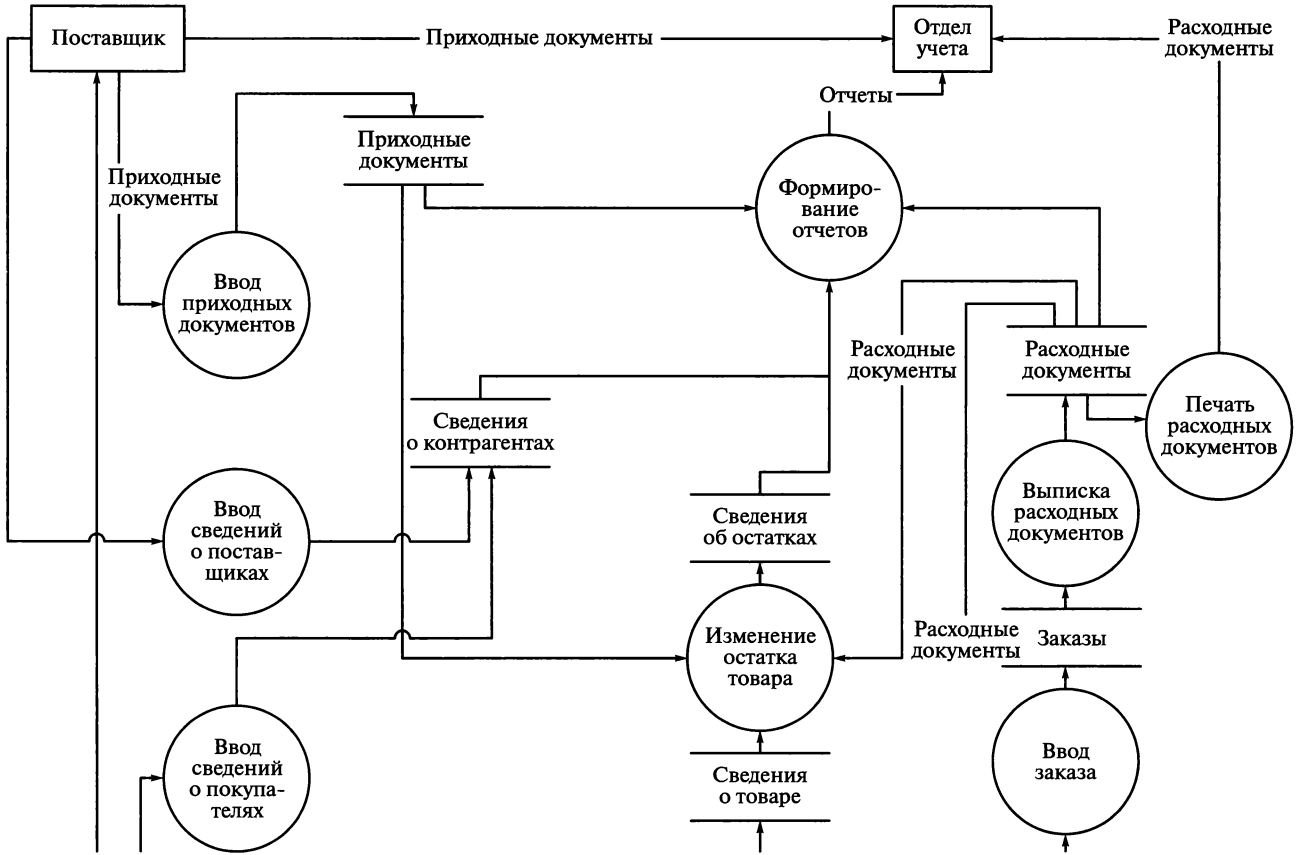
Пример построения диаграммы потоков данных для системы учета успеваемости студентов. Разработаем диаграммы потоков данных системы учета успеваемости студентов. В качестве внешних сущностей для системы выступают декан, заместитель декана по курсу и сотрудник деканата. Определим потоки данных между этими сущностями и системой.

Декан должен получать:

- сводку успеваемости по факультету (процент успеваемости групп, курсов и в целом по факультету) на текущий или указанный момент времени;
- полные сведения об учебе конкретного студента (успеваемость по всем изученным предметам всех завершенных семестров обучения с учетом пересдач).

Заместитель декана по курсу должен получать:

- сводку успеваемости по курсу (процент успеваемости по группам) на текущий или указанный момент;


Поставщик
Приходные документы
Отдел учета
Расходные документы
Отчеты
Приходные документы
Приходные документы
Формирование отчетов
Ввод приходных документов
Расходные документы
Расходные документы
Печать расходных документов
Сведения о контрагентах
Выписка расходных документов
Ввод сведений о поставщиках
Сведения об остатках
Изменение остатка товара
Расходные документы
Заказы
Ввод сведений о покупателях
Сведения о товаре
Ввод заказа

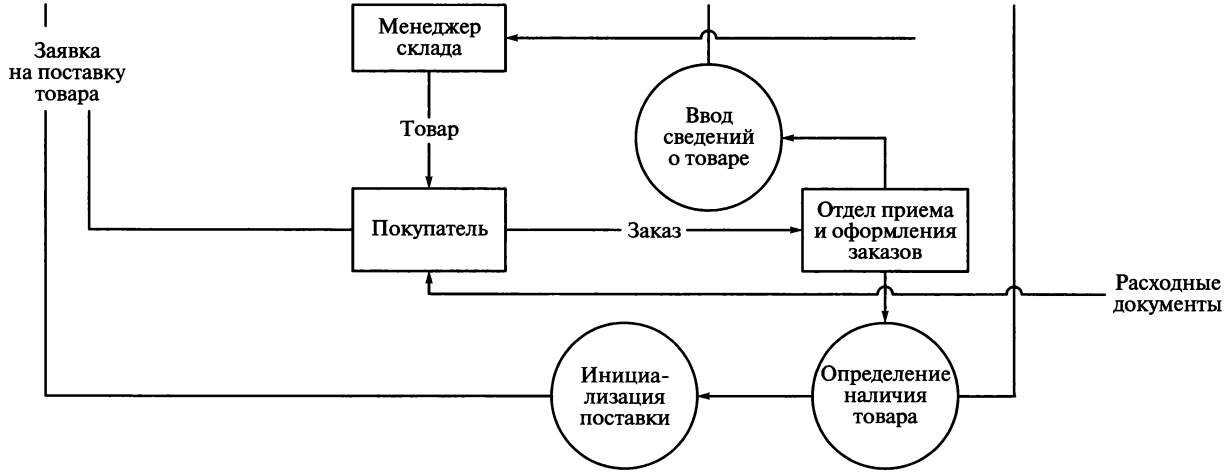


Рис. 2.14. Диаграмма потоков данных АИС «Склад оптовой торговли»

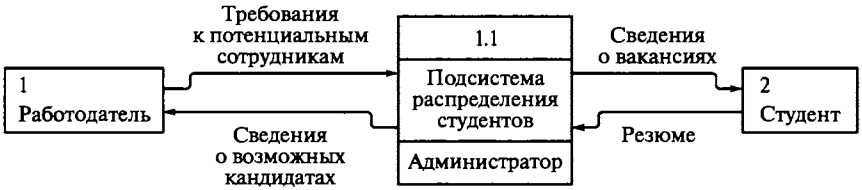


Рис. 2.15. Начальная контекстная диаграмма в нотации Гейна—Сарсона для системы распределения студентов

- сведения о сдаче экзаменов и зачетов указанной группой;
- текущие сведения об успеваемости конкретного студента;

Рис. 2.16. Диаграмма потоков данных системы распределения студентов

- полные сведения об учебе конкретного студента (успеваемость по всем изученным предметам всех завершенных семестров обучения с учетом пересдач);
- список задолжников по факультету и группам с указанием несданных предметов.

Сотрудник деканата должен обеспечивать:

- ввод списков студентов, зачисленных на первый курс;
- корректировку списков студентов в соответствии с приказами о зачислении, отчислении, переводе и т. п.;
- ввод учебных планов кафедр;
- ввод расписания сессии;
- ввод результатов сдачи зачетов и экзаменов на основании ведомостей и направлений.

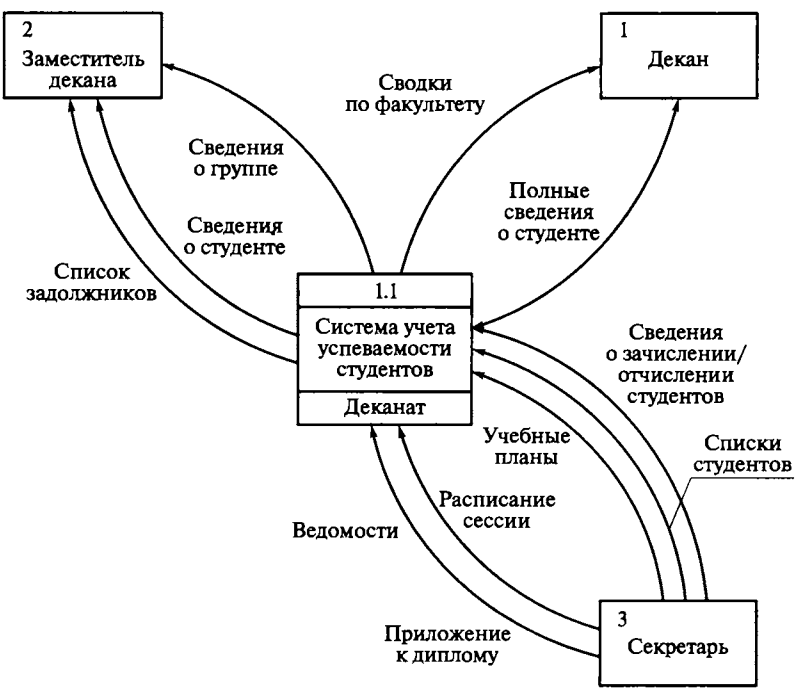


Рис. 2.17. Контекстная диаграмма системы учета успеваемости студентов

Рис. 2.18. Детализирующая диаграмма потоков данных второго уровня

Кроме того, сотрудник деканата должен иметь возможность получать:

- справку о прослушанных студентом предметах с указанием часов и итоговых оценок;
- приложение к диплому выпускника также с указанием часов и итоговых оценок.

В результате получаем контекстную диаграмму в нотации Гейна — Сарсона (рис. 2.17). Далее детализируем процессы в системе (рис. 2.18). На детализирующей диаграмме потоков данных выделены две подсистемы: подсистема наполнения базы и подсистема формирования отчетов, а также хранилище данных, которое может быть реализовано как с помощью средств СУБД, так и без них. Дальнейшая детализация процессов очевидна, однако становится ясно, что полная спецификация данной разработки должна включать описание базы данных в виде диаграммы «сущность — связь».

2.5. ДИАГРАММА «СУЩНОСТЬ — СВЯЗЬ»

Характеристика диаграмм «сущность — связь». Данная диаграмма — (ER-модель данных) обеспечивает стандартный способ определения данных и отношений между ними. Она включает сущности и взаимосвязи, отражающие основные бизнес-правила предметной области. Диаграммы «сущность — связь» в отличие от функциональных диаграмм определяют спецификации структур данных программного обеспечения. Первый вариант модели «сущность — связь» был предложен П.Ченом. В дальнейшем многими авторами были разработаны свои варианты подобных моделей. Все варианты диаграмм «сущность — связь» исходят из одной идеи — графическое изображение нагляднее текстового описания. Все такие диаграммы используют графическое изображение сущностей предметной области, их свойств (атрибутов) и взаимо-

связей между сущностями. Наиболее распространенной является нотация Баркера, ее и будем придерживаться.

Базовыми понятиями ER-модели данных (ER — Entity — Relationship) являются сущность, атрибут и связь.

Сущность — это класс однотипных реальных или абстрактных объектов (людей, событий, состояний, предметов и т.п.), информация о которых имеет существенное значение для рассматриваемой предметной области. Структурой данных называют совокупность правил и ограничений, которые отражают связи, существующие между отдельными частями (элементами) данных.

Каждая сущность должна иметь:

- уникальное имя;
- один или несколько атрибутов, которые либо принадлежат сущности, либо наследуются через связь;
- один или несколько атрибутов, которые однозначно идентифицируют каждый экземпляр сущности.

Экземпляр сущности — это конкретный представитель данной сущности. Имя сущности должно отражать тип или класс объекта, а не его конкретный экземпляр (студент, а не Иванов).

На диаграмме в нотации Баркера сущность изображается прямоугольником, иногда с закругленными углами (рис. 2.19, *а*). Каждая сущность обладает одним или несколькими атрибутами.

Атрибут — любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности (рис. 2.19, *б*). Атрибут, таким образом, представляет собой некоторый тип характеристик или свойств, ассоциированных с множеством реальных или абстрактных объектов. Экземпляр атрибута — определенная характеристика конкретного экземпляра сущности.

Рис. 2.19. Обозначения сущности в нотации Баркера:

а — без атрибутов; *б* — с указанием атрибутов; *в* — с уточнением атрибутов и их типов (# — ключевой, * — обязательный, ∘ — необязательный)

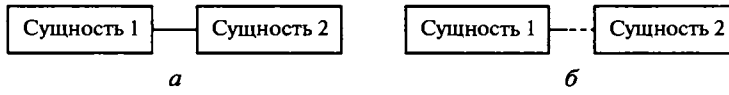


Рис. 2.20. Модальность связи:

а — обязательная; *б* — необязательная (пунктир до середины линии)

Атрибуты делятся на ключевые, т.е. входящие в состав уникального идентификатора ключа, и описательные — прочие.

Первичный ключ — это атрибут или совокупность атрибутов и (или) связей, предназначенная для уникальной идентификации каждого экземпляра сущности (совокупность признаков, позволяющих идентифицировать объект). Ключевые атрибуты помещают в начало списка и помечают символом «#» (рис. 2.19, *в*).

Описательные атрибуты могут быть обязательными или необязательными. Обязательные атрибуты для каждой сущности всегда имеют конкретное значение, необязательные могут быть не определены. Обязательные и необязательные описательные атрибуты помечают символами «*» и «○» соответственно.

Связь — это отношение одной сущности к другой или к самой себе. Каждая связь может иметь одну из двух модальностей связей. Если любой экземпляр одной сущности связан хотя бы с одним экземпляром другой сущности, то связь является обязательной (рис. 2.20, *а*). Необязательная связь представляет собой условное отношение между сущностями (рис. 2.20, *б*). Связь может иметь разную модальность с разных концов. Каждая связь может быть прочитана как слева направо, так и справа налево. Каждая сущность может обладать любым количеством связей с другими сущностями модели. Различают три типа отношений (рис. 2.21): «один-к-одному»; «один-ко-многим»; «многие-ко-многим».

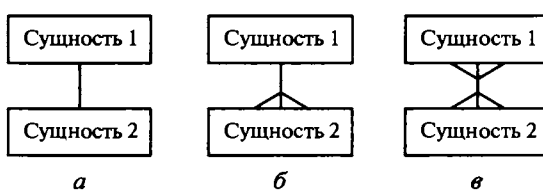


Рис. 2.21. Обозначение отношений в нотации Баркера:

а — «один-к-одному»; *б* — «один-ко-многим»; *в* — «многие-ко-многим»

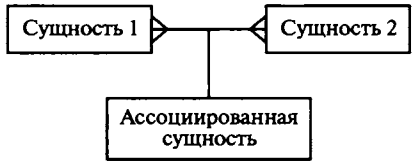


Рис. 2.22. Обозначение ассоциированной сущности в нотации Баркера

Связь «один-к-одному» означает, что один экземпляр первой сущности связан только с одним экземпляром второй сущности. Такая связь, скорее всего, свидетельствует о том, что была неверно разделена одна сущность на две (хотя иногда к такому типу связи прибегают в случае если есть необходимость «засекретить» часть данных). При связи «один-ко-многим» каждый экземпляр первой сущности связан с несколькими экземплярами второй сущности. Связь «многие-ко-многим» показывает, что каждый экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, и наоборот. Такой тип связи является временным. Он допустим на ранних этапах разработки модели. В дальнейшем такую связь необходимо заменить двумя связями «один-ко-многим» путем создания промежуточной сущности.

Независимая сущность представляет независимые данные, которые всегда присутствуют в системе. Они могут быть как связаны с другими сущностями, так и нет.

Зависимая сущность представляет данные, зависящие от других сущностей системы, поэтому она всегда должна быть связана с другими сущностями.

Ассоциированная сущность представляет данные, которые ассоциируются с отношениями между двумя и более сущностями. Обычно данный вид сущностей появляется в модели для разрешения отношения «многие-ко-многим» (рис. 2.22).

Если экземпляр сущности полностью идентифицируется своими ключевыми атрибутами, то говорят о полной идентификации сущности. В противном случае идентификация сущности осуще-

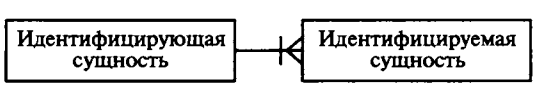


Рис. 2.23. Обозначение идентификации посредством другой сущности в нотации Баркера

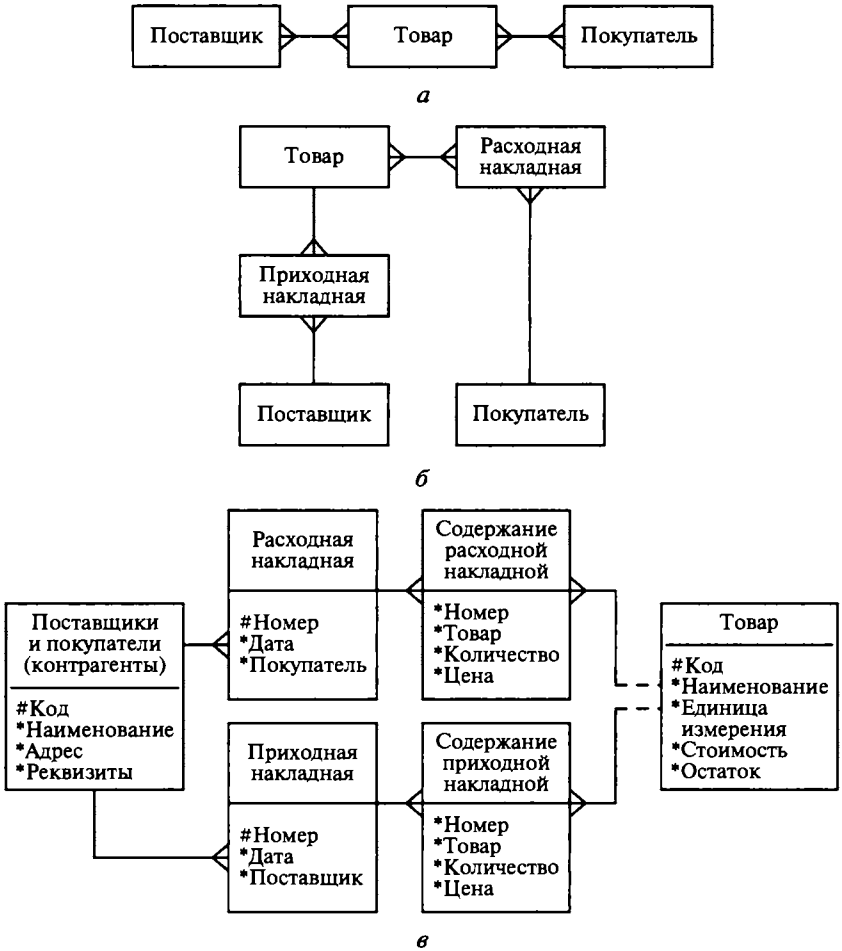


Рис. 2.24. Варианты ER-диаграммы:

а — первый; *б* — промежуточный; *в* — окончательный

ствляется с использованием атрибутов связанной сущности, что указывается черточкой на линии связи (рис. 2.23).

Пример разработки диаграммы «сущность — связь» АИС «Склад оптовой торговли». Основными сущностями для решения указанной задачи являются: поставщик, покупатель, товар.

Сразу возникает очевидная связь между сущностями — «покупатели могут приобретать много товаров», «товары могут приобретаться многими покупателями». Отношение между ними относится к типу «многие-ко-многим» (рис. 2.24, *а*). Для разрешения

этого отношения введем ассоциированную сущность «Накладная», которая отражает приобретение (продажу) товара покупателем (поставщиком) (рис. 2.24, *б*).

Проанализируем атрибуты сущностей. Каждый поставщик и покупатель является юридическим лицом и имеет наименование, адрес, банковские реквизиты. Каждый товар имеет наименование, цену, характеризуется единицей измерения. Каждая накладная имеет уникальный номер, дату выписи, список товаров с количествами и ценами, а также общую сумму накладной. Покупатели покупают товары, получая при этом расходные накладные, в которые внесены данные о количестве и цене приобретенного товара. Каждый покупатель может получить несколько накладных. Каждая накладная должна выписываться на одного покупателя. Каждая накладная должна содержать не менее одного товара (не может быть «пустой» накладной). Каждый товар, в свою очередь, может быть продан нескольким покупателям по нескольким накладным. Аналогичную цепь рассуждений можно выстроить для определения связей между сущностями «Товар» и «Поставщик». Покупатель может быть одновременно и поставщиком, поэтому эти две сущности объединены в одну — «Контрагент». Теперь можно все это внести в диаграмму, как показано на рис. 2.24, *в*.

Уточненная диаграмма должна быть проверена с точки зрения возможности получения всех выходных данных (отчетов), указанных в техническом задании или показанных на диаграмме потоков данных разрабатываемой системы.

Задание

На основании технического задания из практической работы 1 (см. Приложение) выполнить структурный анализ функциональных требований к программному обеспечению и разработать необходимые диаграммы. Оформить результаты, используя MS Visio.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Перечислите принципы создания диаграмм. Объясните назначение каждого вида диаграммы.
2. В чем заключаются основные принципы структурного подхода?
3. Что общего и в чем различия между функциональной моделью SADT и диаграммой потоков данных?
4. Назовите достоинства и недостатки структурного подхода.

Глава 3

ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПРИ СТРУКТУРНОМ ПОДХОДЕ

3.1. СТРУКТУРНАЯ СХЕМА

Процесс проектирования программного обеспечения включает в себя определение структурных компонентов программной системы и связей между ними. Результат уточнения структуры может быть представлен в виде структурной схемы, которая дает достаточно полное представление о проектируемом программном обеспечении.

На рис. 3.1 приведена структурная схема программного обеспечения автоматизированной информационной системы «Склад оптовой торговли».

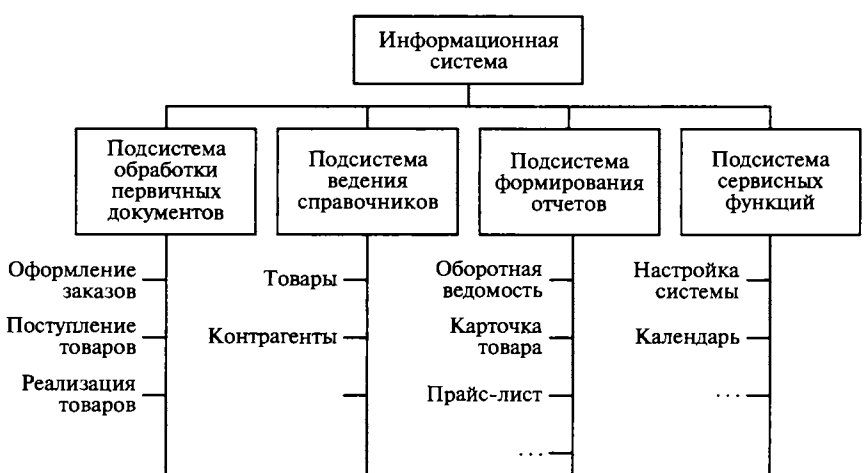


Рис. 3.1. Структурная схема программного обеспечения АИС «Склад оптовой торговли»

3.2. ФУНКЦИОНАЛЬНАЯ СХЕМА

Функциональная схема — это схема взаимодействия компонентов программного обеспечения с описанием информационных потоков, состава данных в потоках и указанием используемых файлов и устройств.

Схемы могут использоваться на различных уровнях детализации, при этом число уровней зависит от размеров и сложности задачи обработки данных. Уровень детализации должен быть таким, чтобы различные части и взаимосвязь между ними были понятны в целом.

Схемы данных отображают путь данных при решении задач и определяют этапы обработки, а также различные применяемые носители данных. Схема данных состоит из следующих символов:

- символы данных (символы данных могут также указывать вид носителя данных);
- символы процесса, производимого с данными (символы процесса могут также указывать функции, выполняемые вычислительной машиной);
- символов линий, указывающих потоки данных между процессами и (или) носителями данных;
- специальные символы, используемые для облегчения написания и чтения схемы.

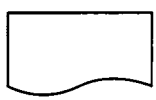
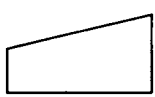
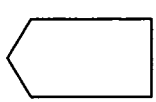
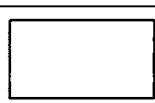
Схемы программ отображают последовательность операций в программе. Схема программы состоит из следующих символов:

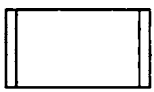
- символы процесса, указывающие фактические операции обработки данных (включая символы, определяющие путь, которого следует придерживаться с учетом логических условий);
- линейные символы, указывающие поток управления;
- специальные символы, используемые для облегчения написания и чтения схемы.

Схемы работы системы отображают управление операциями и поток данных в системе. Схема работы системы состоит из следующих символов:

- символы данных, указывающие на наличие данных (символы данных могут также указывать вид носителя данных);

Таблица 3.1. Графические обозначения основных блоков алгоритмов

Название	Графическое обозначение	Назначение
Терминатор		Символ отображает выход во внешнюю среду и вход из внешней среды (начало или конец схемы программы, внешнее использование и источник или пункт назначения данных)
Данные		Символ отображает данные, носитель данных не определен
Запоминающее устройство с прямым доступом		Символ отображает данные, хранящиеся в запоминающем устройстве с прямым доступом (магнитный диск, магнитный барабан, гибкий магнитный диск)
Документ		Символ отображает данные, представленные на носителе в удобочитаемой форме (машинограмма, документ для оптического или магнитного считывания, микрофильм, рулон ленты с итоговыми данными, бланки ввода данных)
Ручной ввод		Символ отображает данные, вводимые вручную во время обработки с устройств любого типа (клавиатура, переключатели, кнопки, световое перо, полоски со штриховым кодом)
Дисплей		Символ отображает данные, представленные в человекочитаемой форме на носителе в виде отображающего устройства (экран для визуального наблюдения, индикаторы ввода информации)
Процесс		Символ отображает функцию обработки данных любого вида (выполнение определенной опе-

Название	Графическое обозначение	Назначение
		рации или группы операций, приводящее к изменению значения, формы или размещения информации или к определению, по которому из нескольких направлений потока следует двигаться)
Предопределенный процесс		Символ отображает предопределенный процесс, состоящий из одной или нескольких операций или шагов программы, которые определены в другом месте (в подпрограмме, модуле)

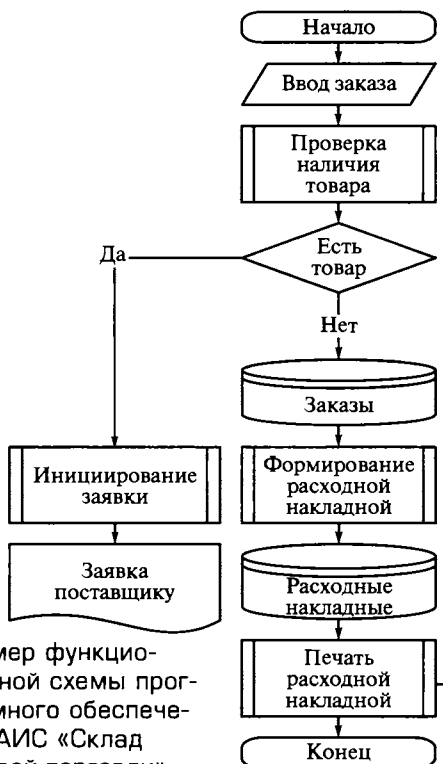


Рис. 3.2. Пример функциональной схемы программного обеспечения АИС «Склад оптовой торговли»

- символы процесса, указывающие операции, которые следует выполнить над данными, а также определяющие логический путь, которого следует придерживаться;
- линейные символы, указывающие потоки данных между процессами и (или) носителями данных, а также поток управления между процессами;
- специальные символы, используемые для облегчения написания и чтения блок-схемы.

Для изображения функциональных схем используют специальные обозначения, установленные ГОСТ 19.701—90 (табл. 3.1).

Функциональные схемы более информативны, чем структурные. На рис. 3.2 приведена функциональная схема программной системы, реализующей операцию продажи товара автоматизированной информационной системы «Склад оптовой торговли».

Задание

На основании технического задания и анализа требований к программному обеспечению в предыдущих работах разработать структурные и функциональные схемы программного обеспечения по своему варианту задания.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Приведите пример структурной схемы ПО.
2. Для чего используют функциональные схемы?
3. Опишите основные элементы функциональных схем ПО.
4. В чем заключаются достоинства и недостатки использования функциональных схем?

Глава 4

ПРИМЕНЕНИЕ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПОДХОДА В АНАЛИЗЕ И ПРОЕКТИРОВАНИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

4.1. ДИАГРАММЫ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

В настоящее время существуют десятки приемов, методик, визуальных представлений, позволяющих моделировать требования к программному обеспечению. Необходимо определить целесообразность использования тех или иных приемов. Анализ требований должен соответствовать тому, что́ делает система, абстрагируясь от деталей реализации, т.е. от того, как она это делает.

Язык UML предоставляет в распоряжение пользователей легко воспринимаемый и выразительный язык визуального моделирования, предназначенный для разработки и документирования сложных моделей систем различного целевого назначения.

Определение вариантов использования. Разработку спецификаций программного обеспечения начинают с анализа требований к функциональности, указанных в техническом задании. В процессе анализа выявляют внешних пользователей разрабатываемого программного обеспечения и перечень отдельных аспектов его поведения в процессе взаимодействия с конкретными пользователями. Аспекты поведения программного обеспечения были названы вариантами использования, или прецедентами (use cases).

Не следует путать вариант использования с конкретными операциями будущей системы. Каждый вариант использования связан с некоторой целью, имеющей самостоятельное значение; например, для текстового редактора Формирование оглавления — это вариант использования, а Связывание заголовков со специальными стилями — операция, которую необходимо выполнить, чтобы стало возможно автоматическое построение оглавления.

В зависимости от цели выполнения процедуры различают следующие варианты использования:

- основные (базовые) — обеспечивают требуемую функциональность разрабатываемого ПО;

- вспомогательные — обеспечивают выполнение необходимых настроек системы и ее обслуживание (например, архивирование информации и т. п.);
- дополнительные — обеспечивают дополнительные удобства для пользователя (как правило, реализуются, если не требуют серьезных затрат каких-либо ресурсов ни при разработке, ни при эксплуатации).

Графических средств языка UML на практике оказывается недостаточно для спецификации функциональных требований к программной системе. Следует отметить, что одним из требований языка UML является самодостаточность диаграмм для представления информации о моделях проектируемых систем. Однако изобразительных средств языка UML явно не хватает для того, чтобы учесть на диаграммах вариантов использования особенности функционального поведения сложной системы. С этой целью рекомендуется дополнять этот тип диаграмм текстовыми сценариями, которые уточняют или детализируют последовательность действий, совершаемых системой при выполнении ее вариантов использования.

В контексте языка UML сценарий используется для дополнительной иллюстрации взаимодействия актеров и вариантов использования. Предлагаются различные способы представления или написания подобных сценариев. Один из таких шаблонов представлен в табл. 4.1 и может быть рекомендован для применения на начальных этапах концептуального моделирования. В зависимости от уровня абстракции вариант использования может описываться кратко или более подробно. Краткая форма описания содержит: имя варианта использования, перечень действующих лиц (актеров) и тип варианта использования (основной, вспомогательный или дополнительный) и его краткое описание.

При написании *сценариев* вариантов использования важно понимать, что текст *сценария* должен дополнять или уточнять диаграмму вариантов использования, но не заменять ее полностью. В противном случае будут потеряны достоинства визуального представления моделей.

Основные понятия. Диаграммы вариантов использования. Диаграммы вариантов использования позволяют наглядно представить ожидаемое поведение программной системы. Основными понятиями диаграмм вариантов использования являются: действующее лицо, вариант использования и связь.

На рис. 4.1 приведены условные обозначения, которые применяют при изображении диаграмм вариантов использования.

Таблица 4.1. Шаблон для написания сценария отдельного варианта использования

Имя	Типичный ход событий, приводящий к успешному выполнению варианта использования	Исключение 1	Примечание 1
Действующие лица		Исключение 2	Примечание 2
Цель		Исключение 3	Примечание 3
Тип		...	...
Краткое описание			
Ссылки на другие варианты использования		Исключение *n*	Примечание *n*

Действующее лицо — внешняя по отношению к разрабатываемому программному обеспечению сущность, которая взаимодействует с ним в целях получения или предоставления какой-либо информации. Как уже упоминалось выше, действующими лицами могут быть пользователи, другое программное обеспечение или какие-либо технические средства.

Вариант использования — некоторая очевидная для действующего лица процедура, решающая его конкретную задачу. Все варианты использования так или иначе связаны с требованиями к функциональности разрабатываемой системы и могут сильно различаться по объему выполняемой работы (рис. 4.2).

Связь — взаимодействие действующих лиц и соответствующих вариантов использования.

Варианты использования также могут быть связаны между собой. При этом фиксируют связи использования и расширения.

Рис. 4.1. Основные условные обозначения диаграмм вариантов использования:

а — действующее лицо; *б* — вариант использования; *в* — связь

Использование (*uses* (*include*)) подразумевает, что существует некоторый фрагмент поведения разрабатываемого ПО, который повторяется в нескольких вариантах использования. Этот фрагмент оформляют как отдельный вариант и указывают связь с ним типа «использование».

Расширение (*extends*) применяют, если имеется два подобных варианта использования, различающиеся наличием в одном из них некоторых дополнительных действий. В этом случае дополнительные действия определяют как отдельный вариант использования, который связан с основным вариантом связью типа «расширение».

Главное назначение диаграммы вариантов использования заключается в формализации функциональных требований к системе и возможности согласования полученной модели с заказчиком на ранней стадии проектирования. Любой из вариантов использования может быть подвергнут дальнейшей декомпозиции на множество подвариантов использования отдельных элементов, которые образуют исходную сущность.

Пример разработки диаграммы вариантов использования. Для иллюстрации особенностей спецификации функциональных требований на диаграмме вариантов использования можно рассмотреть модель системы «Склад оптовой торговли» (рис. 4.3). Для первоначального понимания структуры программной системы выявляются действующие лица (люди-актеры или системы, между которыми происходит взаимодействие). Рассматриваемая

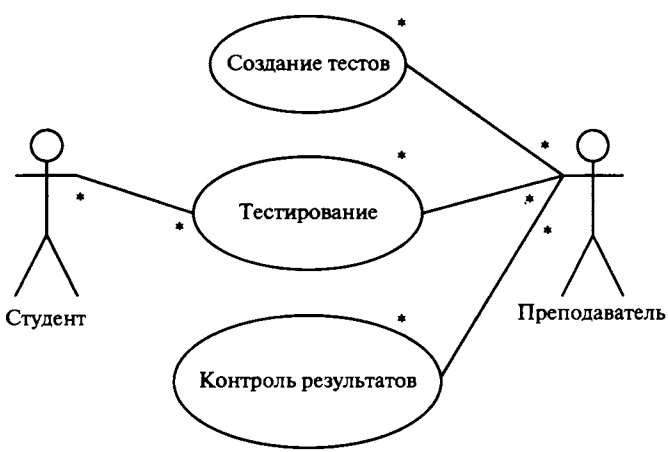


Рис. 4.2. Пример диаграммы вариантов использования для тестовой системы

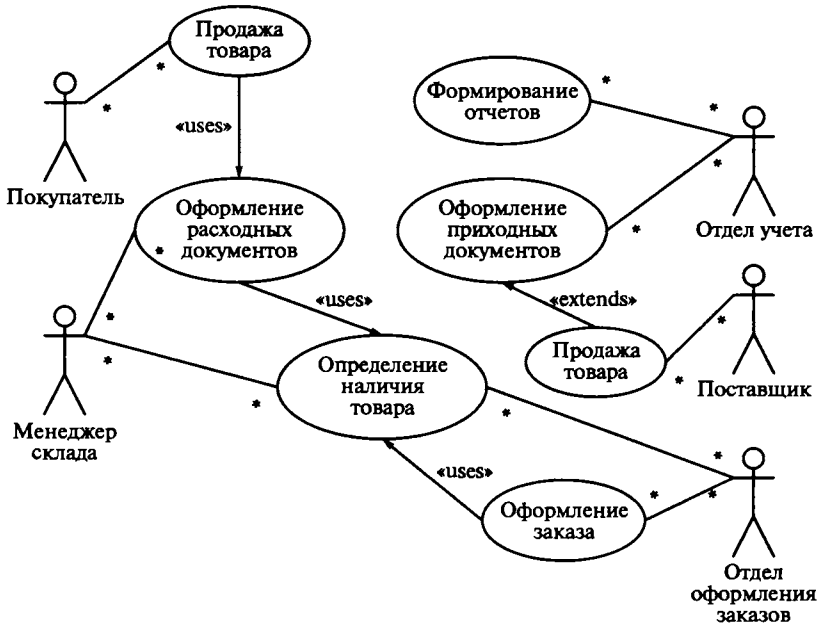


Рис. 4.3. Диаграмма вариантов использования для проектирования программного обеспечения АИС «Склад оптовой торговли»

система имеет пять актеров, двое из которых выступают контрагентами, а другие — менеджерами склада, осуществляющими выполнение всех операций. Каждый из этих актеров взаимодействует с системой, хотя главными актерами, пожалуй, являются поставщики и покупатели (контрагенты), поскольку именно они инициируют функциональность системы. Далее формулируются варианты использования, т.е. действия, выполняемые системой для реализации общения действующих лиц (актеров).

Каждый из действующих лиц преследует определенные цели по отношению к системе: поставщик — сдать товар на склад, покупатель — приобрести товар, менеджер склада — принять и отпустить товар, менеджер учетного отдела — определить объемы поступления и продаж и проанализировать товарный запас. На основании этих целей можно сформулировать базовые варианты использования и проанализировать взаимосвязи между ними. На самом деле вариантов использования может быть гораздо больше. Например, проверить платежеспособность клиента, получить информацию о товаре, оценить запасы товара на складе, получить

оплату и т.д. Однако эта диаграмма дает понять, что будет делать система, как она будет функционировать.

На следующем этапе разработки модели вариантов использования для рассматриваемой системы следует дополнить данную диаграмму текстовым сценарием, написанным на основе предложенного ранее шаблона. Этот сценарий будет дополнять диаграмму, раскрывая содержание и логическую последовательность отдельных действий, которые выполняются системой и актерами в процессе поступления и реализации товаров. В этом случае сценарий удобно представить в виде таблиц, каждая из которых описывает отдельный раздел шаблона.

В главном разделе сценария указывается имя рассматриваемого варианта использования, имена взаимосвязанных с ним актеров, цель выполнения варианта, условный тип и ссылки на другие варианты использования (табл. 4.2).

В следующем разделе сценария описывается последовательность действий, приводящая к успешному выполнению рассматриваемого варианта использования. При этом инициатором действий должен выступать актер Покупатель. Для удобства последующих ссылок каждое действие актеров помечается порядковым номером в последовательности действий (табл. 4.3).

Таблица 4.2. Сценарий варианта использования

Вариант использования	Продажа товара
Актеры	Покупатель, Менеджер отдела оформления заказов, Менеджер склада
Краткое описание	Покупатель запрашивает товар. Менеджер отдела оформления заказов резервирует товар, оформляет заказ, передает заказ Менеджеру склада. Покупатель оплачивает товар, получает товар на складе
Цель	Получение необходимого товара
Тип	Базовый
Ссылки на другие варианты использования	Включает в себя варианты использования: определить наличие товара; оформить заказ

Таблица 4.3. Последовательность действий актеров

Действия актеров	Отклик системы
1. Покупатель запрашивает товар Исключение 1. На складе нет необходимого количества запрашиваемого товара	2. Менеджер отдела оформления заказов проверяет наличие необходимого товара на складе 3. Менеджер отдела оформления заказов резервирует нужный товар
4. Покупатель оплачивает товар Исключение 2. Покупатель не оплатил товар	5. Менеджер отдела оформления заказов выдает разрешение на получение товара 6. Менеджер отдела оформления заказов передает заказ на склад 7. Менеджер склада выдает товар и расходную накладную покупателю 8. Менеджер оформления заказов блокирует получение товара покупателем

В третьем разделе сценария описывается последовательность действий, выполняемых при возникновении исключительных ситуаций, или исключений (табл. 4.4).

Можно дополнить данный сценарий, аналогичным образом описав не только варианты использования «Оформление заказа» и «Определение наличия товара», но и рассмотрев другие исключения, например оформление скидок постоянным покупателям и т. п. При этом полнота сценариев и модели вариантов использования будут определяться теми функциональными требованиями, которые сформулированы в рамках конкретного проекта.

Отдельные небольшие по своему объему сценарии могут быть размещены на диаграмме в форме примечаний note, предназначенных для включения в модель произвольной текстовой информации, которая имеет непосредственное отношение к контексту разрабатываемого проекта. В качестве такой информации могут быть комментарии разработчика (например, дата и версия разработки диаграммы или ее отдельных компонентов), ограничения (например, на значения отдельных связей или экземпляры сущностей) и помеченные значения. Применительно к диаграммам вариантов использования примечание может иметь уточняющую

Таблица 4.4. Последовательность действий актеров при возникновении исключительных ситуаций

Действия актеров	Отклик системы
Исключение 1. На складе нет необходимого количества запрашиваемого товара	
4. Покупатель оплачивает товар	3. Менеджер отдела оформления заказов инициирует поставку нужного товара
Исключение 2. Покупатель не оплатил товар	
	8. Менеджер оформления заказов блокирует получение товара покупателем

информацию, относящуюся к контексту тех или иных вариантов использования.

Графически примечания на всех типах диаграмм обозначаются прямоугольником с «загнутым» верхним правым уголком (рис. 4.4). Собственно текст примечания размещается внутри этого прямоугольника. Примечание может относиться к любому элементу диаграммы, в этом случае их соединяет пунктирная линия. Если примечание относится к нескольким элементам, то от него проводятся, соответственно, несколько линий. Как уже отмечалось, примечания могут присутствовать не только на диаграмме вариантов использования, но и на других канонических диаграммах.

Рекомендации к разработке диаграмм вариантов использования. Как было отмечено ранее, одно из главных назначений диаграммы вариантов использования заключается в формализации функциональных требований к системе. Диаграмма вариантов использования может служить основой для согласования с заказчи-

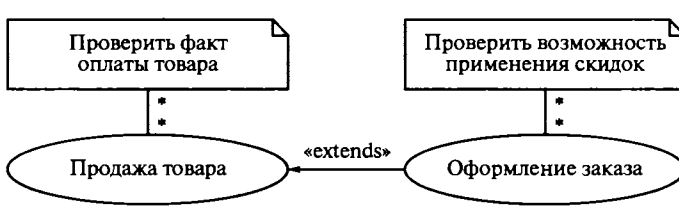


Рис. 4.4. Пример примечаний на диаграммах вариантов использования

ком функциональных требований к системе на ранней стадии проектирования. Любой из базовых вариантов использования в последующем может быть подвергнут декомпозиции на частные варианты использования. При этом рекомендуется, чтобы общее количество актеров в модели не превышало 20, а вариантов использования — 50. В противном случае модель теряет свою наглядность и, возможно, заменяет собой одну из некоторых других диаграмм.

Для разработки диаграммы вариантов использования рекомендуется соблюдать некоторую последовательность действий:

- определить главных, или первичных, и второстепенных актеров;
- определить цели главных актеров по отношению к системе;
- сформулировать основные варианты использования, которые специфицируют функциональные требования к системе;
- упорядочить варианты использования по степени убывания риска их реализации;
- рассмотреть все базовые варианты использования в порядке убывания их степени риска;
- выделить участников, интересы, предусловия и постусловия выполнения выбранного варианта использования;
- написать успешный сценарий реализации выбранного варианта использования;
- определить исключения или неуспех в выполнении сценария варианта использования;
- написать сценарии для всех исключений;
- выделить общие варианты использования и изобразить их взаимосвязи с базовыми со стереотипом uses (include);
- выделить варианты использования для исключений и изобразить их взаимосвязи с базовыми со стереотипом extend;
- проверить диаграмму на отсутствие дублирования вариантов использования и актеров.

Семантика построения диаграммы вариантов использования должна определяться следующими особенностями рассмотренных выше элементов модели. Отдельный экземпляр варианта использования по своему содержанию является выполнением последовательности действий, которая инициализируется посредством экземпляра сообщения от экземпляра актера. В качестве отклика или ответной реакции на сообщение актера выполняется после-

довательность действий, установленная для данного варианта использования. При этом актеры могут генерировать новые сообщения для инициирования вариантов использования.

Подобное взаимодействие будет продолжаться до тех пор, пока не закончится выполнение требуемой последовательности действий экземпляром варианта использования и указанный в модели экземпляр актера не получит требуемый экземпляр сервиса. Окончание взаимодействия означает отсутствие инициализации сообщений от актеров для базовых вариантов использования.

Варианты использования могут быть дополнительно специфицированы примечаниями с текстом, которые в последующем могут стать прототипами операций и методов совместно с атрибутами. Дальнейшая разработка моделей связана с реализацией вариантов использования в виде графа деятельности посредством конечного автомата или любого другого механизма логического представления поведения, включающего предусловия и постусловия. Взаимодействие между вариантами использования и актерами может уточняться на диаграмме кооперации, когда описываются взаимосвязи между системой, содержащей эти варианты использования, и окружением или внешней средой этой системы.

4.2. ДИАГРАММЫ ДЕЯТЕЛЬНОСТИ

Характеристика диаграмм деятельности. Если диаграмма вариантов использования дает «вид сверху» на функциональность системы, диаграмма деятельности, напротив, позволяет подробно иллюстрировать отдельный вариант использования и его сценарии. В зависимости от степени детализации диаграммы деятельностей могут использоваться на разных этапах разработки. На этапе анализа требований и уточнения спецификаций диаграммы деятельностей позволяют конкретизировать основные функции разрабатываемого программного обеспечения.

Под деятельностью в данном случае понимают задачу (операцию), которую необходимо выполнить вручную или с помощью средств автоматизации. Каждому варианту использования соответствует своя последовательность задач. В теоретическом плане диаграмма деятельности — это обобщенное представление алгоритма, реализующего анализируемый вариант использования. На диаграмме деятельность обозначается прямоугольником с закругленными углами (рис. 4.5, *а*). Диаграммы деятельности позволяют описывать альтернативные и параллельные процессы.

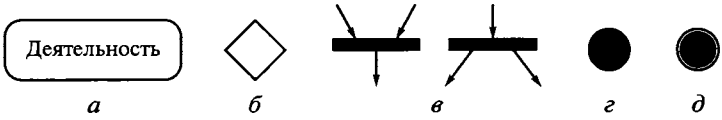


Рис. 4.5. Условные обозначения диаграммы деятельностей:
а — деятельность; *б* — выбор; *в* — линейки синхронизации; *г* — начало; *д* — конец

Для обозначения *альтернативных процессов* используют ромб (рис. 4.5, *б*), условие указывают рядом, а альтернативы «да», «нет» — рядом с соответствующими выходами. С помощью этого же блока можно построить циклический процесс. Множественность активации деятельности обозначают символом «*», помещенным рядом со стрелкой активации деятельности, и при необходимости уточняют надписью вида «для каждой строки».

Для обозначения *параллельных процессов* используют линейки синхронизации (рис. 4.5, *в*). Условные обозначения начала и окончания диаграммы деятельности даны на рис. 4.5, *г* и *д*. Пример диаграммы деятельности с указанием параллельности процессов приведен на рис. 4.6. Условие синхронизации можно уточнить, указав его на диаграмме (рис. 4.7).

Пример построения диаграммы деятельности. В предыдущем примере для конкретизации описания вариантов использования в автоматизированной информационной системе «Склад оптовой торговли» был разработан текстовый сценарий. Этот сценарий дополняет диаграмму, раскрывая содержание отдельных действий, выполняемых системой и актерами. Однако вместо описания вариантов использования или в дополнение к ним можно использовать диаграммы деятельностей. Диаграмма деятельности

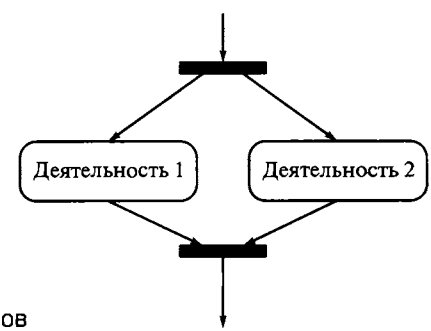


Рис. 4.6. Пример диаграммы деятельности с указанием параллельности процессов

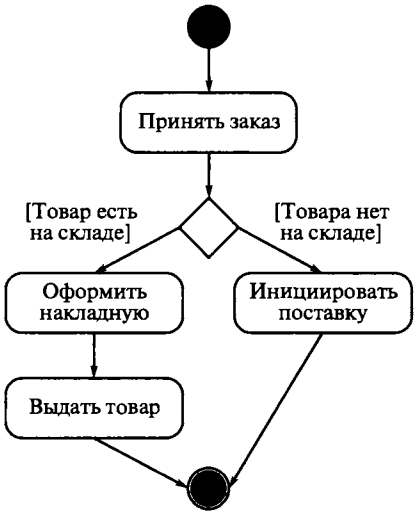


Рис. 4.7. Пример диаграммы деятельности

позволяет проиллюстрировать вариант использования с требуемой степенью подробности. Имеет смысл уточнять только те варианты использования, краткое описание которых недостаточно для понимания сущности решаемых проблем.

В рамках разрабатываемой модели построим диаграммы деятельности для реализации вариантов использования «Поставка товара» (рис. 4.8) и «Продажа товара» (рис. 4.9). Диаграмма деятельности позволяет проиллюстрировать вариант использования с различной степенью подробности. Полная модель системы может содержать несколько диаграмм деятельности, каждая из которых описывает последовательность реализации либо наиболее важных вариантов использования (типичный ход событий и все исключения), либо нетривиальных операций классов.

Помимо стандартного формата описания, UML предлагает вариант с «плавательными дорожками». Этот формат удобен для описания случая, когда в варианте использования участвуют несколько действующих лиц. При этом все состояния действия на диаграмме деятельности делятся на отдельные группы, которые отделяются друг от друга вертикальными линиями (рис. 4.10).

Применение дорожек открывает дополнительные возможности для наглядного представления бизнес-процессов, позволяя специфицировать деятельность подразделений предприятий (рис. 4.11). В случае типового проекта большинство деталей реализации действий могут быть известны заранее на основе анализа существующих систем или предшествующего опыта разработки систем-

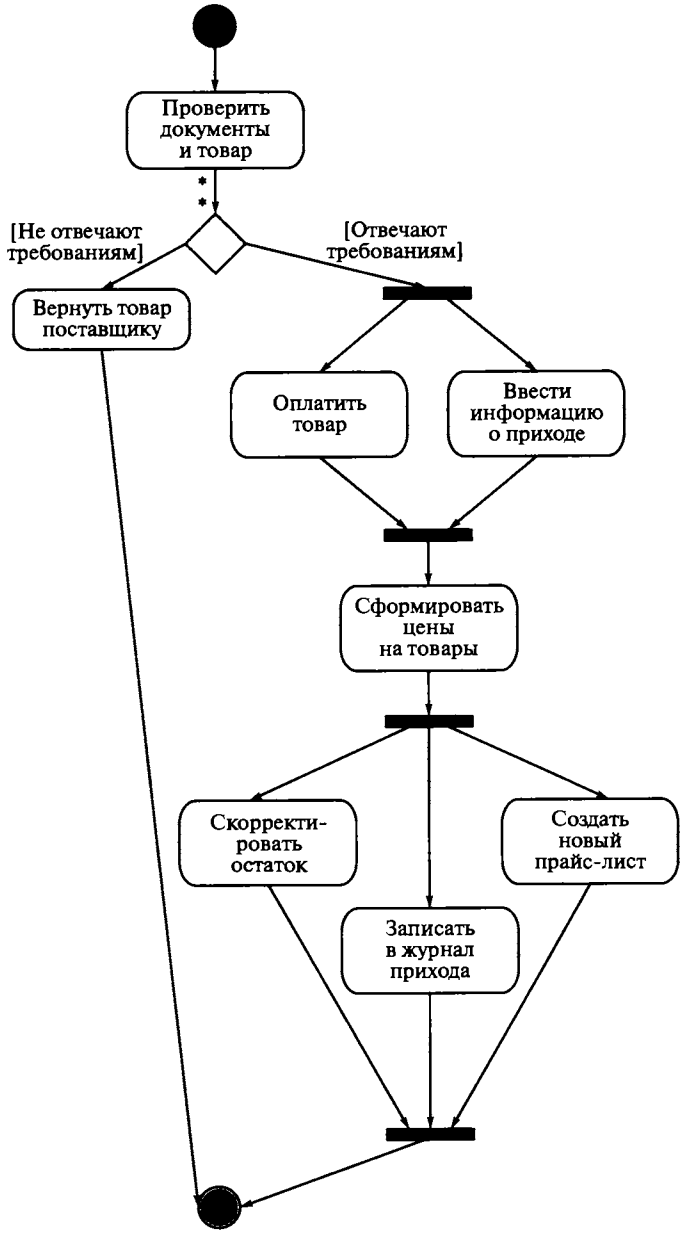


Рис. 4.8. Диаграмма деятельности для варианта использования «Поставка товара»

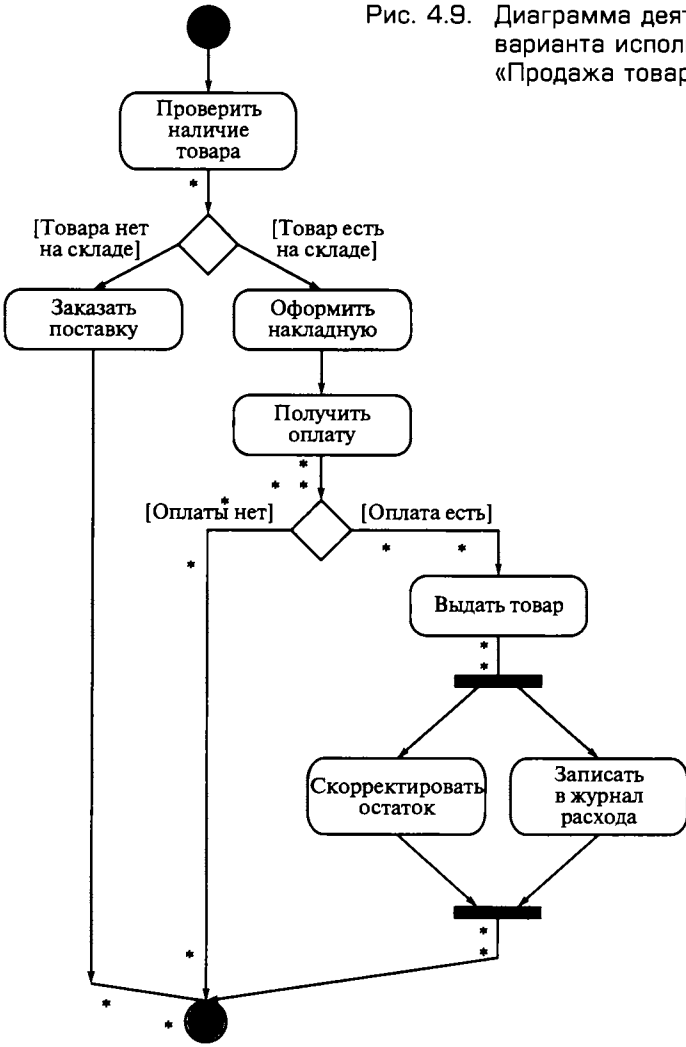


Рис. 4.9. Диаграмма деятельности для варианта использования «Продажа товара»

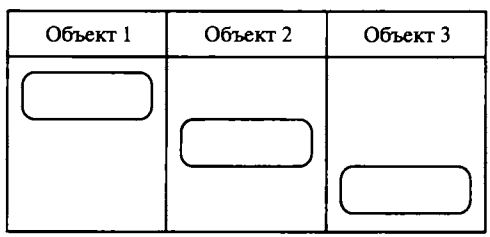


Рис. 4.10. Вариант диаграммы деятельности с дорожками

прототипов. Использование типовых решений может существенно сократить время разработки и избежать возможных ошибок при реализации проекта.

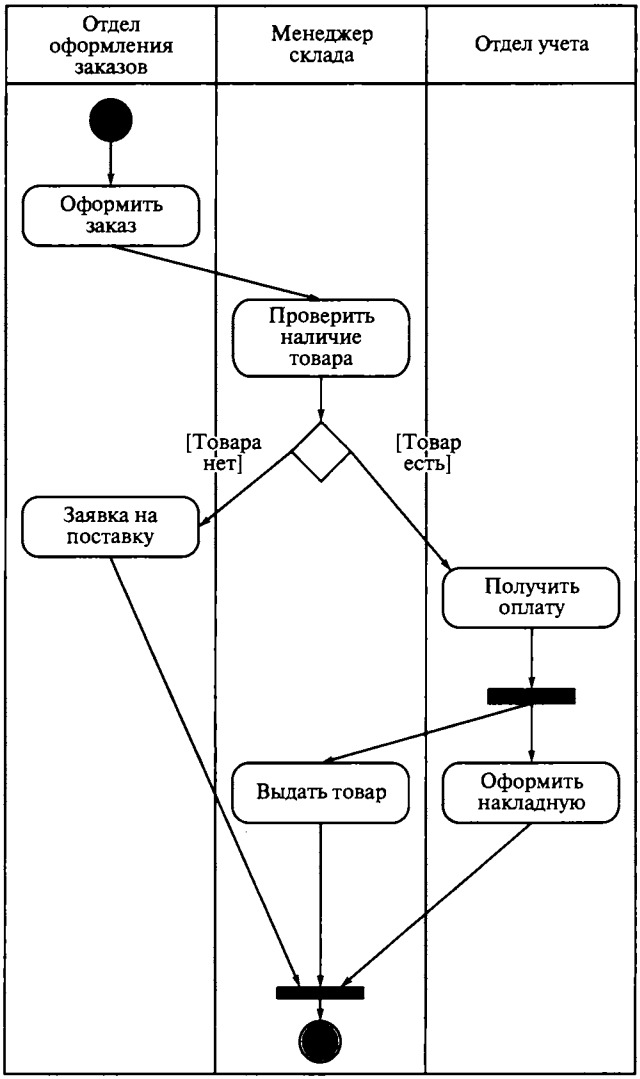


Рис. 4.11. Пример применения диаграммы деятельности с дорожками для варианта использования «Оформление заказа»

4.3. ДИАГРАММЫ ПОСЛЕДОВАТЕЛЬНОСТИ

Характеристика диаграмм последовательности. Рассмотренные диаграммы деятельности используются для спецификации динамики поведения программных систем, время в явном виде в них не присутствует. Однако временной аспект поведения может иметь существенное значение при моделировании синхронных процессов, описывающих взаимодействия объектов. Именно для этой цели в языке UML используются диаграммы последовательности.

Диаграмма последовательности системы — графическая модель, которая для определенного сценария варианта использования показывает динамику взаимодействия объектов во времени. Для построения диаграммы последовательности системы необходимо:

- идентифицировать каждое действующее лицо (объект) и изобразить для него линию жизни;
- из описания варианта использования определить множество системных событий и их последовательность;
- изобразить системные события в виде линий со стрелкой на конце между линиями жизни действующих лиц и системы, а также указать имена событий и списки передаваемых значений.

На диаграмме последовательности изображаются исключительно те объекты, которые непосредственно участвуют во взаимодействии и не показываются возможные статические ассоциации с другими объектами. Для диаграммы последовательности ключевым моментом является именно динамика взаимодействия объектов во времени. При этом диаграмма последовательности имеет как бы два измерения.

Одно измерение — 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. Внутри прямоугольника записываются имя объекта и имя класса, разделенные двоеточием. При этом вся запись подчеркивается линией, что является признаком объекта, представляющим собой экземпляр класса (рис. 4.12).

Не исключается ситуация, когда имя объекта может отсутствовать на диаграмме последовательности. В этом случае указывается только имя класса, а сам объект считается анонимным. Крайним слева на диаграмме изображается объект, который является

инициатором взаимодействия (объект 1 на рис. 4.12). Правее изображается другой объект, который непосредственно взаимодействует с первым. Таким образом, все объекты на диаграмме последовательности образуют порядок, определяемый степенью активности этих объектов при взаимодействии друг с другом.

Второе измерение диаграммы последовательности — вертикальная временна́я ось, направленная сверху вниз. Начальному моменту времени соответствует самая верхняя часть диаграммы. При этом 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. Другими словами, сообщения, расположенные на диаграмме последовательности выше, инициируются раньше тех, которые расположены ниже. При этом масштаб на оси времени не указывается, поскольку диаграмма последовательности моделирует лишь временну́ю упорядоченность взаимодействий типа «раньше-позже».

Линия жизни объекта служит для обозначения периода времени, в течение которого объект существует в системе и, следова-

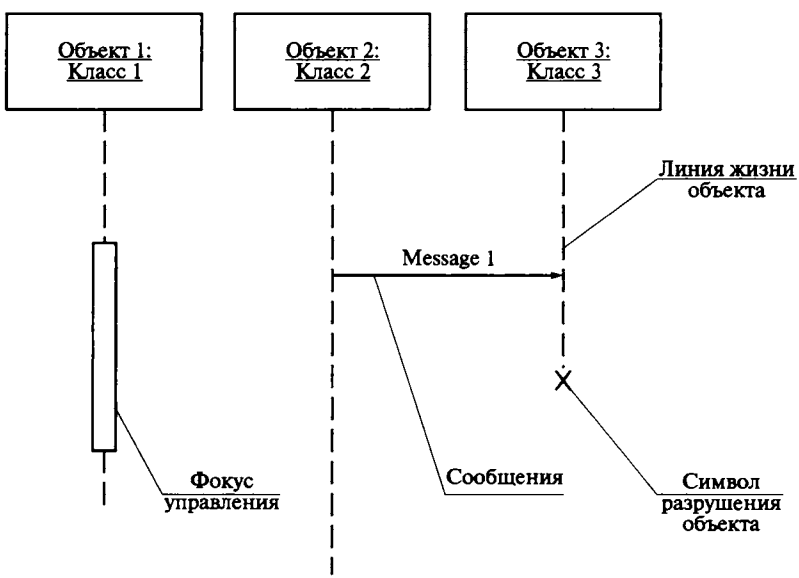


Рис. 4.12. Различные графические примитивы диаграммы последовательности

тельно, может потенциально участвовать во всех ее взаимодействиях. На диаграмме линия жизни изображается пунктирной вертикальной линией, ассоциированной с единственным объектом. Если объект существует в системе постоянно, то и его линия жизни должна продолжаться по всей плоскости диаграммы последовательности.

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 наносятся на диаграмму с соблюдением некоторого порядка инициализации сообщений. Здесь необходимо установить, какие объекты будут существовать постоянно, а какие временно — только на период выполнения ими требуемых действий. Когда объекты определены, можно приступать к спецификации сообщений. При этом следует учитывать роли, которые сообщения играют в системе.

Пример построения диаграммы последовательности. Для того чтобы более точно учитывать временной аспект поведения системы при моделировании процессов, описывающих взаимодей-

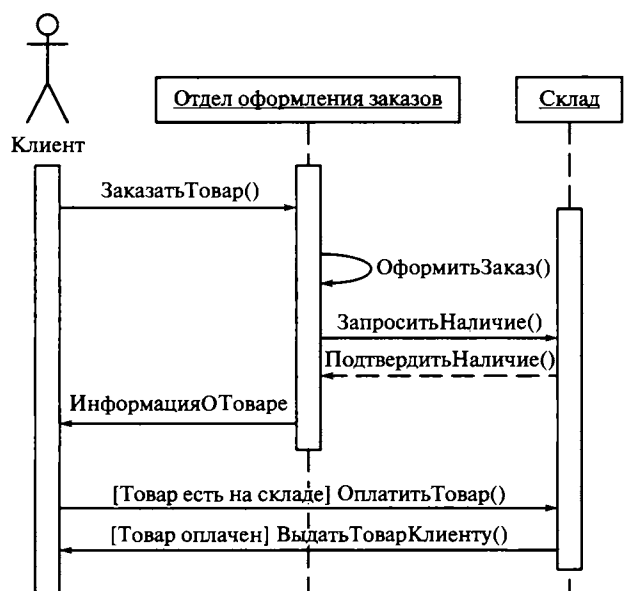


Рис. 4.13. Вариант диаграммы последовательности для моделирования операции продажи товара

ствие, можно построить диаграммы последовательности. В качестве примера построим диаграмму последовательности для реализации варианта использования «Продажа товара» в информационной системе «Склад оптовой торговли» (рис. 4.13).

Данная диаграмма содержит два объекта и одного актера. Объекты не являются постоянно активными, что показано с помощью соответствующих фокусов управления. В качестве имен сообщений указаны имена операций, которые специфицированы у соответствующих классов. Предложения—условия некоторых сообщений записаны обычным текстом в квадратных скобках. Эти условия отражают возможность ветвления процесса продажи и выполнения исключительного сценария соответствующего варианта использования, однако другие варианты использования на данной диаграмме не показаны.

4.4. ДИАГРАММЫ КЛАССОВ

Диаграммой UML, поясняющей внутреннее устройство системы, является диаграмма классов, описывающая создаваемую программную систему через ее компоненты (классы, объекты), отношения и взаимодействия между ними. В основном диаграммы классов в этих методах применяют на этапе проектирования, для того чтобы показать особенности построения конкретных классов. UML предлагает использовать три уровня диаграмм классов в зависимости от степени их детализации:

- концептуальный уровень, на котором диаграммы классов, называемые в этом случае контекстными, демонстрируют связи между основными понятиями предметной области;
- уровень спецификаций, на котором диаграммы классов отображают интерфейсы классов предметной области, т. е. связи объектов этих классов;
- уровень реализации, на котором диаграммы классов непосредственно показывают поля и методы конкретных классов.

Это три разные модели, связь между которыми неоднозначна. Так, если концептуальная модель определяет некоторое понятие предметной области как класс, то это не означает, что для реализации этого понятия будет использован отдельный класс. Однако во всех трех моделях нас интересуют типы объектов (классы) и

их статические отношения, что позволяет использовать единую нотацию.

Каждая из перечисленных моделей используется на конкретном этапе разработки программного обеспечения:

- концептуальная модель — на этапе анализа;
- диаграммы классов уровня спецификации — на этапе проектирования;
- диаграммы классов уровня реализации — на этапе реализации.

Концептуальные модели в соответствии с определением оперируют понятиями предметной области, атрибутами этих понятий и отношениями между ними. Класс при этом традиционно понимают как совокупность общих признаков некоторой группы объектов предметной области. В соответствии с этим определением на диаграмме классов каждому классу соответствует группа объектов, общие признаки которых и фиксирует класс.

На диаграммах класс изображается в виде прямоугольника, внутри которого указано имя класса (рис. 4.14, *а*). При необходимости допускается указывать характеристики класса, например, атрибуты, используя специальные секции условного обозначения (рис. 4.14, *б*), а также операции класса (рис. 4.14, *в*).

В качестве атрибутов представляют некоторые, существенные с точки зрения решаемой задачи характеристики объектов, например, идентифицирующие значения (имя, номер). Для конкретного объекта атрибут всегда имеет конкретное значение (рис. 4.15).

Имя класса должно быть уникальным в пределах пакета, который описывается некоторой совокупностью диаграмм классов (возможно, одной диаграммой). Оно указывается в первой верхней секции прямоугольника. В дополнение к общему правилу наименования элементов языка UML имя класса записывается по центру секции имени полужирным шрифтом и должно начинаться с заглавной буквы. Рекомендуется в качестве имен классов использовать существительные, записанные по практическим соображениям без пробелов. Необходимо помнить, что именно имена классов образуют словарь предметной области при объектно-ориентированном анализе и проектировании.

В первой секции обозначения класса могут находиться ссылки на стандартные шаблоны или абстрактные классы, от которых образован данный класс и соответственно от которых он наследует свойства и методы. В этой секции может приводиться информация о разработчике данного класса и статус состояния разра-

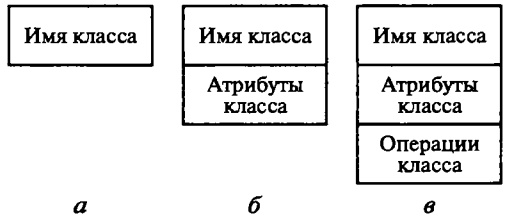


Рис. 4.14. Обозначение класса на концептуальной диаграмме классов:

а — без уточнения характеристик; *б* — с уточнением атрибутов; *в* — с указанием операций

ботки, а также могут записываться и другие общие свойства этого класса, имеющие отношение к другим классам диаграммы или стандартным элементам языка UML.

Примерами имен классов могут быть такие существительные, как «Сотрудник», «Компания», «Руководитель», «Клиент», «Продавец», «Менеджер», «Офис» и многие другие, имеющие непосредственное отношение к моделируемой предметной области и функциональному назначению проектируемой системы.

Класс может не иметь экземпляров или объектов. В этом случае он называется *абстрактным классом*, а для обозначения его имени используется наклонный шрифт (курсив). В языке UML принято общее соглашение о том, что любой текст, относящийся к абстрактному элементу, записывается курсивом.

Рис. 4.15. Пример графического изображения объектов на диаграммах языка UML

В некоторых случаях необходимо явно указать, к какому пакету относится тот или иной класс. Для этой цели используется специальный символ разделитель — двойное двоеточие «::». Синтаксис строки имени класса в этом случае будет следующий:

```
<Имя_пакета>::<Имя_класса>.
```

Другими словами, перед именем класса должно быть явно указано имя пакета, к которому его следует отнести. Например, если определен пакет с именем «Банк», то класс «Счет» в этом банке может быть записан в виде: «Банк::Счет».

Во второй сверху секции прямоугольника класса записываются его атрибуты (attributes) или свойства. В языке UML принята определенная стандартизация записи атрибутов класса, которая подчиняется некоторым синтаксическим правилам. Каждому атрибуту класса соответствует отдельная строка текста, которая состоит из квантора видимости атрибута, имени атрибута, его кратности, типа значений атрибута и, возможно, его исходного значения:

```
<квантор видимости><имя атрибута>[кратность]:
<тип атрибута> = <исходное значение>{строка-свойство}
```

Квантор видимости принимает одно из трех возможных значений и отображается при помощи специальных символов:

- «+» — атрибут с областью видимости типа общедоступный (public). Атрибут доступен или виден из любого другого класса пакета, в котором определена диаграмма;
- «#» — атрибут с областью видимости типа защищенный (protected). Атрибут недоступен или невиден для всех классов за исключением подклассов данного класса;
- «–» — атрибут с областью видимости типа закрытый (private). Атрибут с этой областью видимости недоступен или невиден для всех классов без исключения.

Квантор видимости может быть опущен, в этом случае его отсутствие просто означает, что видимость атрибута не указывается. Эта ситуация отличается от принятых по умолчанию соглашений в традиционных языках программирования, когда отсутствие квантора видимости трактуется как public или private. Однако вместо условных графических обозначений можно записывать соответствующее ключевое слово: public, protected, private.

Имя атрибута представляет собой строку текста, которая используется в качестве идентификатора соответствующего атрибута и потому должна быть уникальной в пределах данного класса.

Имя атрибута является единственным обязательным элементом синтаксического обозначения атрибута.

Кратность атрибута характеризует общее количество конкретных атрибутов данного типа, входящих в состав отдельного класса. В общем случае кратность записывается в форме строки текста в квадратных скобках после имени соответствующего атрибута. В качестве примера рассмотрим следующие варианты задания кратности атрибутов:

[0..1] означает, что кратность атрибута может принимать значение 0 или 1. При этом 0 означает отсутствие значения для данного атрибута;

[0..*] означает, что кратность атрибута может принимать любое положительное целое значение большее или равное 0. Эта кратность может быть записана короче в виде простого символа — [*];

[1.:*] означает, что кратность атрибута может принимать любое положительное целое значение большее или равное 1;

[1..5] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 4, 5;

[1..3,5,7] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 5, 7;

[1..3,7.. 10] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 7, 8, 9, 10;

[1..3,7..*] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, а также любое положительное целое значение большее или равное 7.

Если кратность атрибута не указана, то по умолчанию принимается ее значение равное 1..1, т.е. в точности 1.

Тип атрибута представляет собой выражение, семантика которого определяется языком спецификации соответствующей модели. В нотации UML тип атрибута иногда определяется в зависимости от языка программирования, который предполагается использовать для реализации данной модели. В простейшем случае тип атрибута указывается строкой текста, имеющей осмысленное значение в пределах пакета или модели, к которым относится рассматриваемый класс.

При задании атрибутов могут быть использованы две дополнительные синтаксические конструкции — это подчеркивание строки атрибута и пояснительный текст в фигурных скобках.

Подчеркивание строки атрибута означает, что соответствующий атрибут может принимать подмножество значений из некоторой области значений атрибута, определяемой его типом. Эти значения можно рассматривать как набор однотипных записей

или массив, которые в совокупности характеризуют каждый объект класса.

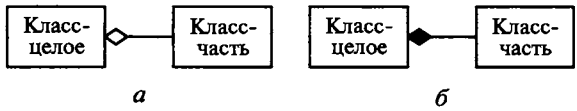
В третьей сверху секции прямоугольника записываются операции, или методы класса. *Операция* (operation) представляет собой некоторый сервис, предоставляющий каждый экземпляр класса по определенному требованию. Совокупность операций характеризует функциональный аспект поведения класса. Запись операций класса в языке UML также стандартизована и подчиняется определенным синтаксическим правилам. При этом каждой операции класса соответствует отдельная строка, которая состоит из квантора видимости операции, имени операции, выражения типа возвращаемого операцией значения и, возможно, строка—свойство данной операции:

```
<квантор видимости><имя операции>(список параметров):
<выражение типа возвращаемого значения>{строка-свойство}
```

Квантор видимости, как и в случае атрибутов класса, может принимать одно из трех возможных значений и соответственно отображается при помощи специального символа. Символ «+» обозначает операцию с областью видимости типа общедоступный (public). Символ «#» обозначает операцию с областью видимости типа защищенный (protected). И наконец, символ «-» используется для обозначения операции с областью видимости типа закрытый (private).

Квантор видимости для операции может быть опущен. В этом случае его отсутствие просто означает, что видимость операции не указывается. Вместо условных графических обозначений также можно записывать соответствующее ключевое слово: public, protected, private. Кроме внутреннего устройства или структуры классов, важную роль при разработке программной системы играют различные отношения между классами, которые также могут быть изображены на диаграмме классов:

- отношение ассоциации — наличие произвольной взаимосвязи между классами;
- отношение обобщения — отношение между более общим элементом и более частным элементом (родителем и потомком);
- отношение композиции (рис. 4.16, *а*) — частный случай отношения агрегации, когда части не могут выступать в отрыве от целого и с уничтожением целого уничтожаются и все его составные части;



а *б*

Рис. 4.16. Условные обозначения специальных видов ассоциации:
а — композиция; *б* — агрегация

- отношение агрегации (рис. 4.16, *б*) — один из классов представляет собой некоторую сущность, которая включает в себя в качестве составных частей некоторые другие сущности.

В контексте языка UML существует специальный класс — *интерфейс*, у которого имеются только операции и отсутствуют атрибуты. Интерфейсы на диаграмме служат для спецификации таких элементов модели, которые видимы извне, но их внутренняя структура остается скрытой от клиентов.

Процесс разработки диаграммы классов занимает центральное место при разработке проектов сложных систем. От умения правильно выбрать классы и установить между ними взаимосвязи зависит не только успех процессов проектирования, но и производительность программы.

Задание

Используя техническое задание из практической работы 1 (см. Приложение), провести анализ требований к программному обеспечению в соответствии с вариантом задания, применяя объектно-ориентированный подход и язык визуального моделирования UML. Результаты оформить в MS Visio.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Охарактеризуйте понятие UML.
2. Каковы преимущества использования UML?
3. Какие сущности описывают поведение системы?
4. Каковы типы сущностей в UML?
5. Перечислите виды диаграмм в UML.
6. Приведите примеры атрибутов ассоциаций. Что такое кратность ассоциации?
7. Какие сущности обычно содержат диаграммы классов?
8. Постройте диаграмму деятельности для моделирования процесса проведения экзамена.

Глава 5

РАЗРАБОТКА ПРОТОТИПА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

5.1. ОСНОВНЫЕ СВЕДЕНИЯ О ПРОТОТИПАХ

Для начала определим, для чего используются прототипы программного обеспечения.

Во-первых, пользователю часто бывает трудно сформулировать требования к тому, что он ожидает от системы. В этом случае прототип интерфейса пользователя (User Interface — UI), оперативно созданный по результатам собеседования, дает ему возможность увидеть схематичную реализацию того, как разработчик представляет соответствующую часть системы.

При использовании прототипов важен любой результат. Если программист понял требования заказчика правильно, то очевидно, что разработка программной системы продвигается в верном направлении. Если пользователь не совсем доволен предложенным ему вариантом реализации системы, польза заключается в том, что он может указать, в чем заключается непонимание задачи программиста.

Во-вторых, прототипирование дает возможность выбрать одно из альтернативных концептуальных решений. Любую техническую задачу можно решить различными способами. Это касается как задачи формулировки требований к программной системе, так и реализации ее пользовательского интерфейса.

В-третьих, часто бывает так, что комбинация функциональных и нефункциональных требований такова, что возникает риск невозможности их реализации. Как правило, такой риск связан с требованиями к быстродействию системы при известных ограничениях на среду ее реализации. В этом случае создаются прототипы (не обязательно связанные с пользовательским интерфейсом), реализующие соответствующую часть системы, имитирующие потоки данных, поступающие на ее вход, и обработку этих данных.

5.2. ВИДЫ ПРОТОТИПОВ

Прототипы могут быть горизонтальные и вертикальные, одноразовые и эволюционные, бумажные и электронные.

Горизонтальный (поведенческий) прототип моделирует интерфейс пользователя приложения, не затрагивая логики обработки и структур данных. Горизонтальные прототипы следует использовать, когда необходимо прояснить нечеткие требования, имеющие многоальтернативную реализацию. Для горизонтальных прототипов не обязательно использовать программную среду реализации, в которой будет разрабатываться конечный вариант системы. Если в разработанном интерфейсе используются базы данных, то они имитируются в программном коде; при этом тексты, отображаемые на экране, должны отражать реальную специфику проблемной области, иначе пользователю будет трудно сосредоточиться. При создании прототипа имитируются результаты расчетов и запросов, например, запросы во внешние системы. Желательно реализовать ту часть кода, которая отвечает за перемещение между экранами в процессе исполнения вариантов использования, чтобы пользователь смог понять, как будет действовать система в ответ на его действия. Прежде чем создавать горизонтальный прототип, необходимо определить, какие основные экраны будут присутствовать, какие окна будут открываться, какие правила перехода между ними будут поддерживаться. Для этого хорошо использовать модель диаграммы состояний, где с разными экранами (окнами) сопоставляются состояния, а с активными элементами управления, вызывающими закрытие одних интерфейсных элементов и открытие других, — переходы.

Вертикальный (структурный) прототип направлен не столько на проектирование интерфейса пользователя, сколько на реализацию вертикального «среза» системы, затрагивая все уровни ее реализации. При создании такого рода прототипов рекомендуется использовать те же языки и среды реализации, что и при изготовлении целевой системы. Такого рода прототипы используются для анализа применимости, проверки архитектурных концепций.

Одноразовый (исследовательский) прототип создается, когда нужно быстро получить макет разрабатываемой программной системы, те или иные ее аспекты и компоненты. Целям создания исследовательских прототипов служит технология RAD (rapid application development) — быстрая разработка приложений. Одноразовый прототип должен создаваться быстро, при его разра-

ботке не следует уделять внимание вопросам повторного использования кода, качества, быстродействия, технологичности и т. п. В результате получается «сырой» код, который может содержать значительное число дефектов. Необходимо принять меры к тому, чтобы фрагменты кода, реализующие такого рода прототипы, не стали частью целевой системы.

Эволюционный прототип создается как первое приближение системы, призванное стать впоследствии самой системой. Программный код эволюционного прототипа должен последовательно перерасти в код целевого приложения. Поэтому данный вид прототипов требует всего того, от чего следует отказаться при создании одноразовых прототипов: тщательной разработки, применения технологических методов и приемов, тестирования результатов и т. п.

Бумажный прототип — наброски интерфейсов на бумаге. Они, конечно, не заменят интерфейса, созданного в среде разработки. Однако при всех недостатках у таких прототипов есть два существенных достоинства. Заказчик не станет акцентировать внимание на цветовом решении, форме кнопок и не будет отвлекаться от анализа функциональности.

Промежуточным решением между *электронным* и бумажным вариантами прототипов UI класса является презентация, созданная при помощи средств электронного офиса (например, комбинации Microsoft Visio и Microsoft PowerPoint).

Представим иллюстрированные сценарии прецедентов. Для того чтобы разработчику лучше понять специфику проблемной области и лучше отразить ее в интерфейсе пользователя, в текст описания сценария варианта использования включают информацию, конкретизирующую те или иные его особенности. Так, информация об объемах используемых данных позволит оптимально построить пользовательский интерфейс и оценить на ранних стадиях проекта «узкие места» в обработке данных, которые могут повлиять на производительность системы. Например, в диалоге с системой при выборе из малого числа вариантов значений лучше подойдут индикаторы (checkbox) или радиокнопки (radiobutton). При выборе, ограниченном 2—3 десятками позиций, удобен выпадающий список. В ситуациях, когда приходится выбирать из сотен или тысяч вариантов, потребуется дополнительное окно для фильтрации и поиска.

Также при разработке системы будет полезной информация об интенсивности выполнения тех или иных операций. Эта информация позволит разработать удобный интерфейс, обеспечиваю-

щий высокое быстродействие системы за счет минимизации действий при выполнении различных операций, в том числе структуризации подачи информации, удаления из главных интерфейсов редко используемых опций и т. п.

5.3. ПРИМЕР ПОСТРОЕНИЯ ПРОТОТИПА

Построим горизонтальный (поведенческий) прототип программного обеспечения для реализации варианта использования «Оформление заказа» автоматизированной информационной системы «Склад оптовой торговли».

Описание варианта использования «Оформление заказа». В процессе выполнения прецедента менеджер по приему заказов выбирает заказчика из клиентской базы (до 10 000 клиентов) либо регистрирует нового клиента (5 % случаев), определяет товарные позиции из справочника (товары разбиты на несколько видов, количество позиций каждого вида товара не превышает 100) и указывает их количество (средняя закупка — 8 позиций). Система отображает на экране наименования товара, цену, сумму и количество на складе. Менеджер назначает скидку и определяет порядок оплаты (на данный момент существуют три варианта по-

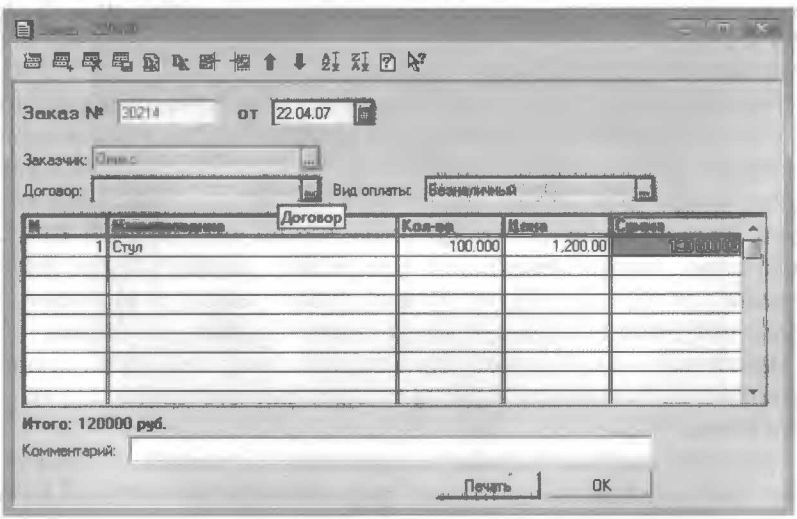


Рис. 5.1. Экранная форма оформления заказа

рядка оплаты). Система рассчитывает итоговую сумму. Заказ можно вывести на печать.

Разработка прототипа варианта использования «Оформление заказа». Для горизонтальных прототипов не обязательно использовать программную среду реализации, в которой будет разрабатываться конечный вариант системы. Для удобства и быстрой разработки прототипа можно применить средства технологической платформы «1С». Ниже представлено окно, в котором осуществляется оформление заказа (рис. 5.1). Для выбора клиента (заказчика) имеется кнопка, при активизации которой на экране появляется форма списка клиентов. Вид оплаты выбирается из ниспадающего списка при активизации соответствующей кнопки.

На форме выводится информация об итоговой сумме, предъявляемой к оплате клиенту.

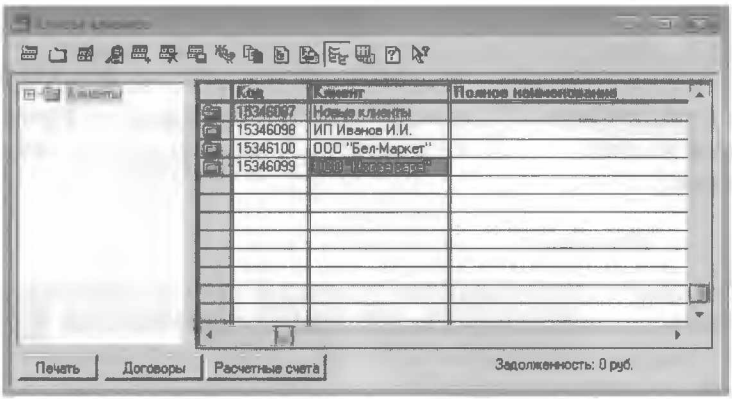
Поскольку количество клиентов довольно велико, выбор клиента из списка должен проходить в отдельном окне, где можно было бы осуществить эффективный поиск клиента или фильтрацию (рис. 5.2, *а*). В нижней части окна отображается информация о состоянии взаиморасчетов с клиентом. В этом же окне можно добавить новый элемент в справочник клиентов, т.е. вызвать окно редактирования и регистрации нового клиента.

Справочник клиентов имеет большое количество реквизитов, форма справочника построена таким образом, чтобы не отражать все реквизиты справочника как графы его визуального представления, иначе таблица получилась бы слишком широкой; поэтому ввод и редактирование сведений о клиенте осуществляются в отдельном окне (рис. 5.2, *б*).

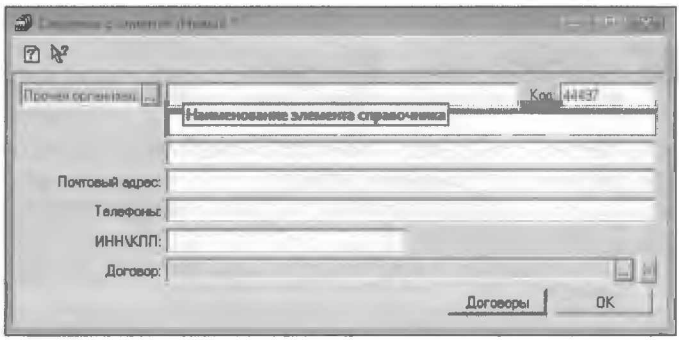
Форма для подбора товара вызывается двойным щелчком по полю наименования в таблице или при добавлении в нее новой строки. С помощью иерархического списка можно выбрать категорию товара и получить список товаров соответствующей категории. Можно осуществить фильтрацию по виду товара. Так как число видов товара невелико, то его можно выбрать через ниспадающий список (рис. 5.2, *в*). На форме отображается информация о наличии товара на складе на момент создания документа.

Найти в списке нужного клиента или товар можно по первым символам наименования или кода, т.е. в формах применим «горячий» поиск.

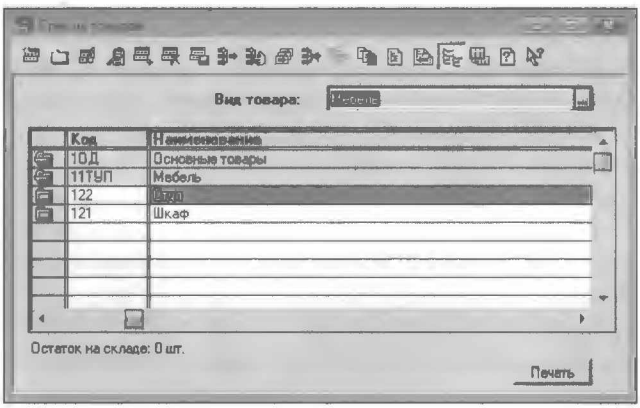
Графический интерфейс, созданный в прототипе, отвечает всем требованиям стандартного графического интерфейса — используется ограниченный набор цветов, применяется их правильное сочетание, имеется инструментальная панель быстрых кно-



а



б



в

Рис. 5.2. Экранные формы:

а — список клиентов; *б* — регистрация нового клиента; *в* — список товаров

пок, продумана последовательность переключения фокуса управляющих элементов. Для доступа к основным командам используются горячие клавиши. Имеются ярлычки подсказок, всплывающие при перемещении курсора мыши над быстрыми кнопками и иными компонентами. Существует возможность вызова файла справки.

Задание

По описаниям вариантов использования из предложенного варианта задания построить прототипы программного обеспечения. Создать презентацию в Microsoft PowerPoint для демонстрации прототипа заказчику.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какова цель разработки прототипов?
2. Перечислите виды прототипов.
3. Какие достоинства имеются у бумажных прототипов?
4. В каких случаях создается исследовательский прототип?
5. Для чего используются иллюстрированные сценарии прецедентов?

Глава 6

ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

6.1. ОСНОВНЫЕ ПРАВИЛА СОЗДАНИЯ ИНТЕРФЕЙСА

Специалисты обычно формулируют некоторый набор принципов и правил, позволяющих как оценивать удобство интерфейса, так и предлагать решения, повышающие его удобство. Приведем эти правила.

Правило доступности. Система должна быть настолько понятной, чтобы пользователь, никогда раньше не видевший ее, но хорошо разбирающийся в предметной области, мог без всякого обучения начать ее использовать. Это правило служит некоторым идеалом, к которому надо стремиться, поскольку на практике достичь такой степени доступности почти никогда не удается. Тем не менее, специалисты продолжают делать в этом направлении все возможное.

Правило эффективности. Система не должна препятствовать эффективной работе опытных пользователей, работающих с ней долгое время. Очевидным примером нарушения этого правила является нацеленность системы только на новичков, использование средств, которые хорошо подходят для неопытного пользователя, ограничивая его в возможности сделать что-то не так, но неэффективны для эксперта, который и так знает, что и где ему нужно сделать.

Правило непрерывного развития. Система должна способствовать непрерывному росту знаний, умений и навыков пользователя и приспосабливаться к его меняющемуся опыту. Плохие результаты приносит предоставление только базовых возможностей или оставление начинающего пользователя наедине со сложным интерфейсом, которым уверенно пользуются эксперты. Нарушение непрерывности при переходе от одного набора возможностей к другому также приносит неудобства, поскольку пользователь вынужден разбираться с добавленными возможностями в новом контексте.

Большинство пользователей можно поместить в три группы: новички, опытные и средние, которые уже знают больше, чем новички, и не делают столько ошибок, но еще не приобрели автоматизма при выполнении большинства операций и иногда путаются в интерфейсе. Новичкам необходима помощь в освоении новой для них системы и контроль за их действиями, опытным пользователям — высокая эффективность выполнения часто требующихся действий и возможность гибкого управления системой в таких ситуациях, которые встречаются реже, но способны вызвать проблемы при их неадекватной поддержке. Про средних же пользователей часто забывают, хотя подавляющее большинство пользователей программного обеспечения относится именно к этой категории. Им нужны достаточно высокие эффективность и гибкость вместе с возможностью быстро получать адекватную помощь по возникающим время от времени разнообразным вопросам.

Правило соблюдения контекста. Система должна быть согласована с контекстом, в котором ей предстоит работать. Это правило требует от системы быть работоспособной не «вообще», а именно в том окружении, в котором ею будут пользоваться. В контекст могут входить специфика и объемы входных и выходных данных, тип и цели организаций, в которых система должна работать, уровень пользователей, зашумленность помещений и пр.

6.2. ПРИНЦИПЫ РАЗРАБОТКИ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

Представленные выше правила определяют общие требования, которым должен удовлетворять удобный интерфейс. Следующие принципы позволяют находить решения, повышающие удобство пользовательского интерфейса.

Принцип структуризации. Пользовательский интерфейс должен быть целесообразно структурирован. Близкие по смыслу, родственные его части должны быть связаны видимым образом, а независимые — разделены; похожие элементы должны выглядеть похоже, а непохожие — различаться.

Принцип простоты. Наиболее распространенные операции должны выполняться максимально просто. При этом должны быть видимые ссылки на более сложные процедуры.

Принцип видимости. Все функции и данные, необходимые для решения определенной задачи, должны быть видны, когда пользователь пытается ее решить.

Принцип обратной связи. Пользователь должен получать сообщения о действиях системы и о важных событиях внутри нее. Сообщения должны быть информативными, краткими, однозначными и написанными на языке, понятном пользователю.

Принцип толерантности. Интерфейс должен быть гибким и терпимым к ошибкам пользователя. Ущерб от ошибок должен снижаться за счет возможности отмены и повтора действий и за счет разумной интерпретации любых разумных действий пользователя и введенных им данных. По возможности следует избегать обязывающего взаимодействия (модальных диалогов), основанного на ограничении свободы пользователя.

Принцип повторного использования. Следует стараться многократно использовать внутренние и внешние компоненты, обеспечивая тем самым унифицированность интерфейса и сходство между его похожими элементами.

Рассмотрим некоторые вопросы разработки эргономичного интерфейса пользователя.

6.3. ВЗАИМОДЕЙСТВИЕ МЕЖДУ ПОЛЬЗОВАТЕЛЕМ И КОМПЬЮТЕРОМ

Человекомашинный интерфейс обеспечивает связь между пользователем и компьютером, он позволяет достигать поставленных целей, успешно находить решение поставленной задачи. Взаимодействие — обмен действиями и реакциями на эти действия между компьютером и пользователем. Существует ряд стилей взаимодействий, которые подразделяются на два основных вида.

1. Использование интерфейса языка команд — ввод команд текстовыми средствами.

2. Непосредственное манипулирование. Таким образом, имеется ряд способов, которыми пользователь мог бы связываться с компьютером:

- языки команд — пользователь управляет системой, вводя соответствующие команды в текстовом режиме;
- вопрос и ответ — диалог, где компьютер задает вопросы, а пользователь отвечает ему (или наоборот);

- формы — пользователь заполняет формы или поля диалога, вводя данные в соответствующие поля;
- меню — пользователь обеспечен рядом опций и управляет системой, выбирая необходимые пункты;
- прямое манипулирование — пользователь управляет объектами на экране посредством устройства манипулирования типа мыши.

В разрабатываемой программной системе также применен комплексный подход к созданию интерфейса. Здесь используется прямое манипулирование, меню, формы и диалоги.

Цель создания эргономичного интерфейса состоит в том, чтобы отобразить информацию настолько эффективно, насколько это возможно для человеческого восприятия, и структурировать отображение на дисплее таким образом, чтобы привлечь внимание к наиболее важным единицам информации. Основная же цель в том, чтобы минимизировать общую информацию на экране и представить только то, что является необходимым для пользователя.

6.4. РАЗМЕЩЕНИЕ ИНФОРМАЦИИ НА ЭКРАНЕ

Количество информации, отображаемой на экране, называется *экранной плотностью*. Исследования показали, что чем меньше экранная плотность, тем отображаемая информация наиболее доступна и понятна для пользователя, и наоборот, если экранная плотность большая, это может вызвать затруднения в усвоении информации и ее ясном понимании. Однако опытные пользователи могут предпочитать интерфейсы с большой экранной плотностью. Информация на экране может быть сгруппирована и упорядочена в значимые части. Это может быть достигнуто использованием кадров (фреймов), методов типа цветового кодирования, рамок, негативного изображения или других методов для привлечения внимания.

Выделение элементов интерфейса яркостью. Для привлечения внимания к каким-либо элементам интерфейса можно воспользоваться выделением этих элементов большей яркостью на фоне других, более темных. Однако важно не переусердствовать с этим методом, поскольку большое количество ярких элементов способно вызвать дискомфорт у пользователя и можно получить обратный эффект — перегрузку интерфейса. Применять этот метод

следует только при необходимости. Существует несколько способов выделения яркостью:

- движение (мигание или изменение позиции) — очень эффективный метод, поскольку глаз имеет специальный детектор движущихся элементов;
- яркость — не очень эффективный метод, так как люди способны различить лишь несколько уровней яркости;
- цвет — использование может быть чрезвычайно эффективным;
- форма (символ, шрифт, форма символа) используется для того, чтобы дифференцировать различные категории данных;
- использование разных алфавитов (шрифтов) в раличных формах;
- размер (текста, символов) — обычно применяют увеличение выделенного объекта в 1,5 раза;
- оттенение (различная текстура объектов) — эффективный метод для привлечения внимания к какой-либо части экрана;
- окружение (подчеркивание, рамки, инвертированное изображение). Очень эффективный способ, если не переусердствовать.

Использование цвета при проектировании эргономичного интерфейса. Цвет может улучшить интерфейс, но для многих систем использование цвета практически не влияет на эффективность работы пользователя. Основное назначение цвета — создание интерфейсов, более интересных для пользователей; однако имеются случаи, когда цвет может помочь проектировщику интерфейса. Особенно это эффективно при группировке информации, выделении различий между информацией или простых сообщений (ошибки, состояния и т.д.).

Цвет — мощный визуальный инструмент, и применять его надо очень осторожно, чтобы не вызвать у пользователя дискомфорт от ошибочных цветовых комбинаций. Перечислим некоторые принципы использования цвета, которыми нужно руководствоваться при проектировании эргономичного интерфейса:

- необходимо ограничить число цветов до 4 на экране и до 7 — для последовательности экранов;
- для неактивных элементов лучше использовать бледные цвета;

- если цвет служит для кодировки информации, следует удостовериться, что пользователь правильно понимает код; например, неоплаченные счета выделяются красным цветом;
- важно при использовании цвета учитывать представления пользователя; например, для картографа зеленый — это лес, желтый — пустыня, синий — вода. Для химика красный означает горячий, синий — холодный и т. д.;
- при отображении состояния, как правило, красный означает опасность (стоп), зеленый — продолжение работы, желтый — цвет предупреждения;
- для привлечения внимания наиболее эффективны белый, желтый и красный цвета;
- при необходимости упорядочения данных можно использовать спектр 7 цветов (радуга);
- если надо разделить данные, то цвета выбирают из различных частей спектра: красный — зеленый, синий — желтый, любой цвет — белый;
- для группировки данных, объединения и подобия нужно использовать соседние цвета спектра: оранжевые — желтые, синие — фиолетовые.

Важно отметить, что около 9 % людей не различают цветов (обычно, красно-зеленые сочетания); однако эти люди могут отличать черно-белые оттенки, поэтому проектировщики автоматизированных систем должны проверять, не нарушает ли использование различных цветов в интерфейсах восприятия пользователей этой категории.

Непротиворечивость и стандартизация. Данные на экране следует располагать таким образом, чтобы пользователь знал, где найти и где ожидать вывода необходимой информации:

- информация, на которую следует немедленно обратить внимание, должна всегда отображаться на видном месте, чтобы захватить внимание пользователя (например, предупреждающие сообщения и сообщения об ошибках);
- не очень часто востребуемая информация (например, справка) не должна отображаться, но должна быть доступна по необходимости. Например, иконка «справки» или соответствующая опция меню должна быть доступна на каждом экране.

Тексты и диалоги. Приведем некоторые принципы, которыми необходимо руководствоваться при создании текстовых диалогов и отображений:

- текст в нижнем регистре читается приблизительно на 13% быстрее, чем текст, напечатанный полностью в верхнем регистре;
- символы верхнего регистра наиболее эффективны для передачи информации, которая должна привлечь внимание. Не используйте верхний регистр, если вы не хотите выделять какую-либо информацию;
- выровненный по правому краю текст труднее читать, чем равномерно распределенный текст с невыровненным правым полем;
- оптимальный интервал между строками равен или немного больше, чем высота символов.

Меню. Необходимый элемент автоматизированной системы — меню, позволяющее пользователю выполнять задачи внутри приложения и управлять процессом решения. Меню — набор опций, отображаемых на экране, где пользователи могут выбирать и выполнять действия, тем самым производя изменения в состоянии интерфейса. Достоинство меню в том, что пользователи не должны помнить название элемента или действия, которое они хотят выполнить, они должны только распознать его среди пунктов меню. Таким образом, меню может использовать даже неопытный пользователь. Однако проект меню должен быть тщательно продуман — чтобы меню было эффективным, названия его пунктов должны быть очевидными.

Меню может занимать много экранного места, но решением этой проблемы может быть использование всплывающего или ниспадающего меню, которое вызывается щелчком по пиктограмме, строке меню или другому объекту.

Основные принципы создания меню. В процессе проектирования системы меню приложения необходимо принять наилучший способ отображения меню, чтобы оно было понятным и легким в использовании. Обычно команды меню упорядочены некоторым иерархическим способом. Основная проблема состоит в том, чтобы правильно распределить различные пункты меню по различным уровням и правильно их сгруппировать.

Принципы проектирования меню:

- структура меню должна соответствовать структуре решаемой системой задачи; организация меню должна отра-

зить наиболее эффективную последовательность шагов, ведущих к решению поставленной задачи;

- пункты меню должны быть краткими, грамматически правильными и соответствовать своему заголовку. Порядок пунктов меню выбирается в соответствии с соглашением, частотой и порядком использования, а также в зависимости от потребностей задачи или пользователя;
- выбор пунктов меню должен быть обеспечен несколькими способами — с помощью клавиатуры, с помощью мыши и через другие объекты пользовательского интерфейса. Важно зафиксировать легко запоминаемые сочетания клавиш для более быстрого доступа к пунктам меню, поскольку это очень экономит время.

Формы. Формы — основной элемент интерфейса. Назначение форм — удобный ввод и просмотр данных, состояния, сообщений автоматизированной системы.

Основные принципы проектирования форм:

- форма проектируется для более удобного, понятного и быстрого решения поставленной задачи. Если форма переносится из бумажной формы, то передвижение по смежным полям не должно вызывать затруднений у пользователя;
- размещение информационных единиц на пространстве формы должно соответствовать логике ее будущего использования: это зависит от необходимой последовательности доступа к информационным единицам, частоты их использования, а также от относительной важности элементов;
- важно использовать незаполненное пространство, чтобы создать равновесие и симметрию среди информационных элементов формы, для фиксации внимания пользователя в нужном направлении;
- логические группы элементов необходимо отделять пробелами, строками, цветовыми или другими визуальными средствами;
- взаимозависимые или связанные элементы должны отображаться в одной форме.

При разработке форм необходимо продумать и указать, какие кнопки в полосе системного меню должны быть доступны в том или ином окне, должно ли окно допускать изменения пользовате-

лем его размера, каким должен быть заголовок окна. Без особой необходимости не нужно делать окна с изменяемыми размерами. При изменении размеров, если не применены специальные приемы, нарушается компоновка окна, что негативно сказывается на работе пользователя. Имеет смысл создать окно с изменяемыми размерами, если это позволяет пользователю изменять полезную площадь расположенных в нем компонентов отображения и редактировать информацию: текст, изображение, списки и т.д. Если проектируется изменяемое окно, то необходимо принять меры, чтобы компоненты в окне при этом тоже изменяли свои размеры или местоположение, равномерно распределяясь по площади окна и не оставляли пустых мест.

При проектировании форм необходимо стремиться к использованию ограниченного набора цветов и уделять внимание их правильному сочетанию. Для фона формы избираются нейтральные цвета (светло-серые). Цвет не должен использоваться в качестве основного средства передачи информации, надо выбирать системные цвета, которые пользователь может перестраивать по своему усмотрению.

Управляющие элементы и функционально связанные с ними компоненты экрана следует зрительно объединять в группы, заголовки которых коротко и четко поясняют их назначение. Каждое окно должно иметь некоторую центральную тему, которая подчиняется его композиции. Пользователь должен понимать, для чего предназначено данное окно и что в нем наиболее важно. Недопустимо перегружать окно большим числом элементов управления ввода и отображения информации. Также недопустимо, чтобы сходные по функциям органы управления в разных окнах назывались по-разному или размещались в разных местах окон. Важно позаботиться о том, как приложение впишется в общую организацию рабочего пространства системы и как оно будет взаимодействовать с другими приложениями.

Дизайн заголовков и полей. Для отдельных полей заголовок должен быть выровнен по левому краю; для полей списков заголовок должен быть выше и левее по отношению к основному полю; числовые поля выравниваются по правому полю. Длинные колоночные поля или длинные столбцы информационных единиц с одиночными полями необходимо объединять в группы по пять элементов, разделяемых пустой строкой, — это помогает пользователю мысленно обрабатывать информацию по выделенным группам.

В формах с большим количеством информации важно использовать названия разделов, которые однозначно свидетельствуют

о характере принадлежащей им информации. Необходимо четко разделить отображение заголовков и непосредственно полей ввода. Заголовки должны быть краткими, знакомыми и содержательными.

Поля, необязательные для заполнения либо не имеющие особой важности, должны отличаться визуально (цветом или другими эффектами) от полей важных и обязательных для заполнения.

Форматы ввода. Следует обеспечить ввод значений по умолчанию во все поля, которые это допускают и где такая функция не будет раздражать пользователя. Можно назначить клавиши или коды для ввода часто повторяющихся значений. Входные данные должны быть значимыми и общепринятыми. Не следует объединять поля ввода чисел и символов, поскольку числовые и алфавитные клавиши находятся неудобно относительно друг друга на клавиатуре.

Необходимо исключить частое переключение между верхним и нижним регистрами для ускорения ввода данных. Нельзя требовать от пользователя ввода незначимых цифр (например, вместо 00000010 пользователь должен ввести только 10). Аналогично, нельзя требовать от пользователя ввести информацию, которая была предварительно введена или которая может быть автоматически получена из системы. Желательно использовать значения по умолчанию, чтобы минимизировать процесс ввода информации.

Организация системы навигации и системы отображения состояний. Навигация обеспечивает пользователю возможность перемещаться между различными экранами, информационными единицами и подпрограммами в автоматизированной системе. В полноценной системе пользователь всегда может получить информацию о состоянии системы, о процессе выполнения или активной подпрограмме.

Общие принципы проектирования. Существует ряд навигационных средств и приемов, которые помогают пользователю ориентироваться в системе. Они включают использование заголовков страниц для каждого экрана, номеров страниц, строк и столбцов, отображение текущего имени файла вверху экрана.

Тип системы навигации зависит от принятого стиля интерфейса. Для интерфейсов языка команд существует очень мало способов обеспечения полноценной навигации. В интерфейсах с меню можно использовать иерархически структурированное меню. Диалоговые интерфейсы сами по себе защищают пользователя от ошибочных действий. Информация состояния обычно отображается внизу экрана и содержит в себе данные о количестве запи-

сей, числе обработанных единиц, процессе печати, очереди печати и т.д.

Проектирование сообщений. Сообщения необходимы для направления действий пользователя в нужную сторону, подсказок и предупреждений при выполнении необходимых действий на пути решения задачи. Они также включают в себя подтверждения действий со стороны пользователя и подтверждения со стороны системы, что задачи выполнены успешно либо по каким-то причинам не выполнены. Сообщения могут быть выведены в форме диалога, экранных заставок и т.п. Сообщения могут предложить пользователю:

- выбрать из предложенных альтернатив опцию или набор опций;
- ввести информацию;
- выбрать опцию из набора опций, которые могут изменяться в зависимости от текущего контекста;
- подтвердить фрагмент введенной информации перед продолжением ввода.

Сообщения могут быть помещены в модальные диалоговые окна, которые вынуждают пользователя ответить на вопрос прежде, чем начнет выполняться любое другое действие. Это может быть полезно, когда система принуждена заставить пользователя обдумать решение перед продолжением работы. Немодальные диалоговые окна позволяют работать с другими элементами интерфейса, в то время как само окно может игнорироваться.

6.5. ПРЕДОТВРАЩЕНИЕ, ОБНАРУЖЕНИЕ И ИСПРАВЛЕНИЕ ОШИБОК

Обычный человек в нормальном состоянии совершает множество ошибок разного рода. Можно сказать, что человек, в отличие от компьютера, является адаптивной аналоговой системой и успешность его «функционирования» в гораздо большей степени определяется не точностью выполнения действий и формулировки мыслей, а способностью быстро выдать хорошее приближение к нужному результату и достаточно быстро поправиться, если это необходимо.

Ошибки пользователя могут быть основаны на неправильном понимании действия или порядка действий или быть случайными, непреднамеренными, например опечатка при вводе текста.

Ошибки второго вида могут быть разделены еще на шесть подвидов:

- неточность в выборе опции (например, пользователь случайно нажал кнопку «Выход» и программа закрылась);
- ошибки при управлении данными (например, присвоение ошибочного имени файла из-за неточности отображения последнего);
- ошибки ассоциативного характера (например, сохранение файла с именем какого-либо человека, так как пользователь думал о нем в момент сохранения);
- потеря активности, когда пользователь забывает необходимую последовательность действий для продолжения работы;
- ошибка режима или состояния, когда пользователь думает, что он находится в одном состоянии, а фактически — другом; например, режим вставки взамен режима печати поверх текста в текстовом процессоре.

Пользователь всегда будет делать ошибки даже в отличной программной системе, поэтому в разрабатываемой системе всегда должна быть предусмотрена защита от ошибок. Техника такой защиты включает в себя следующие аспекты:

- принудительные действия в системе, которые предотвращают или затрудняют появление ошибок;
- обеспечение хороших и информативных сообщений об ошибках;
- использование обратимых действий, позволяющих пользователям исправлять их собственные ошибки;
- обеспечение нормальной диагностики системы, в процессе которой пользователю объясняется, в чем суть ошибки, и указываются пути ее исправления.

Рассмотрим основные принципы обработки ошибок в формах ввода:

- обеспечить возможность посимвольного редактирования введенных записей для исправления ошибок ввода (опечаток);
- если ошибка обнаружена системой, желательно вернуть курсор в поле с ошибочными данными и каким-либо образом выделить это поле визуально;

- выводить значимые сообщения об ошибках, использующие стиль языка пользователя и соответствующую терминологию;
- выводить сообщения об ошибках, которые объясняют и предлагают пути ее устранения.

Эффективность предотвращения и преодоления ошибок пользователей тем выше, чем реже пользователи ошибаются при работе с данным интерфейсом и чем меньше времени и усилий требуется для преодоления последствий уже сделанных ошибок.

6.6. ОБЩИЕ ТРЕБОВАНИЯ К ГРАФИЧЕСКОМУ ИНТЕРФЕЙСУ ПОЛЬЗОВАТЕЛЯ

Под *графическим интерфейсом* пользователя подразумевается тип экранного представления, при котором пользователь может выбирать команды, запускать задачи и просматривать в списке файлы, указывая на пиктограммы или пункты списков меню, показанных на экране (например, система Windows). Графический интерфейс пользователя любой программы должен включать в себя:

- главное меню;
- инструментальную панель быстрых кнопок, дублирующих основные разделы меню;
- контекстное меню, всплывающее при щелчке пользователя правой кнопкой мыши на том или ином компоненте;
- продуманную последовательность переключения фокуса управляющих элементов;
- клавиши быстрого доступа ко всем разделам меню и всем управляющим элементам, горячие клавиши для доступа к основным командам;
- ярлычки подсказок, всплывающие при перемещении курсора мыши над быстрыми кнопками и иными компонентами;
- полосу состояния, используемую для развернутых подсказок и выдачи различной информации пользователю;
- файл справки, темы которого отображаются при нажатии клавиши F1 или при выборе пользователем соответствующего раздела меню;

- информацию о версии, доступную пользователю при щелчке на пиктограмме приложения правой кнопкой мыши;
- возможность настройки приложения и запоминания настроек, чтобы при очередном сеансе работы восстанавливались настройки, установленные в прошлом сеансе;
- средства установки приложения, регистрация его в Windows и удаление из Windows.

Задание

Провести анализ эргономичности графического интерфейса пользователя по прототипам, созданным в предыдущей работе. Составить таблицу соответствия (несоответствия) разработанного пользовательского интерфейса требованиям стандартного графического интерфейса пользователя. Подготовить рекомендации по исправлению выявленных нарушений. (Желательно анализировать интерфейс не своей программы, а другого студента.)

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Перечислите общие принципы и правила создания удобного интерфейса.
2. Как используется цвет при разработке эргономичного интерфейса?
3. Перечислите основные требования к компоновке форм.
4. Как можно предотвратить появление ошибок при вводе?
5. Как правильно создать меню?

Глава 7

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Язык Delphi, поддерживая концепцию объектно-ориентированного программирования, дает возможность определять классы. *Класс* — это сложная структура, включающая помимо описания данных описание процедур и функций, которые могут быть выполнены над представителем класса — объектом.

Вот пример объявления простого класса:

```
TPerson = class
private
fname: string[15]; faddress: string[35];
public
procedure Show;
end;
```

Данные класса называются *полями*, процедуры и функции — *методами*. В приведенном примере TPerson — это имя класса, fname и faddress — имена полей, show — имя метода. Согласно принятому в Delphi соглашению имена полей должны начинаться с буквы f (от слова field — поле).

Описание класса помещают в программе в раздел описания типов (type).

Объекты как представители класса объявляются в программе в разделе var, например:

```
var
student: TPerson; professor: TPerson;
```

В Delphi *объект* — это динамическая структура. Переменная-объект содержит не данные, а ссылку на данные объекта, поэтому программист должен позаботиться о выделении памяти для этих данных. Выделение памяти осуществляется при помощи специального метода класса — *конструктора*, которому обычно при-

сваивают имя Create (создать). Для того чтобы подчеркнуть особую роль и поведение конструктора, в описании класса вместо слова procedure используется слово constructor.

Ниже приведено описание класса TPerson, в состав которого введен конструктор:

```
TPerson = class private
fname: string [ 15 ];
faddress: string[35];
constructor Create; // конструктор
public
procedure show; // метод
end;
```

Выделение памяти для данных объекта происходит путем присваивания значения результата применения метода-конструктора к типу (классу) объекта. Например, после выполнения инструкции

```
professor := TPerson.Create;
```

выделяется необходимая память для данных объекта professor.

Помимо выделения памяти, конструктор, как правило, решает задачу присваивания полям объекта начальных значений, т.е. осуществляет инициализацию объекта. Ниже приведен пример реализации конструктора для объекта TPerson:

```
constructor TPerson.Create;
begin
fname := '';
faddress := '';
end;
```

Если в программе какой-либо объект больше не используется, можно освободить память, занимаемую полями данного объекта. Для выполнения этого действия применяют *метод-деструктор* Free. Например, для того чтобы освободить память, занимаемую полями объекта professor, достаточно записать

```
professor.Free;
```

Концепция объектно-ориентированного программирования предполагает возможность определять новые классы посредством добавления полей, свойств и методов к уже существующим классам. Такой механизм получения новых классов называется *порождением*. При этом новый, порожденный класс (потомок) наследует свойства и методы своего базового, родительского класса.

В объявлении класса-потомка указывается класс родителя. Например, класс TEmployee (сотрудник) может быть порожден от рассмотренного выше класса TPerson путем добавления поля FDepartment (отдел). Объявление класса TEmployee в этом случае может выглядеть так:

```
TEmployee = class(TPerson)
FDepartment: integer; // номер отдела
constructor Create(Name:TName; Dep:integer);
end;
```

Заключенное в скобки имя класса TPerson показывает, что класс TEmployee является производным от класса TPerson. В свою очередь, класс TPerson является базовым для класса TEmployee.

Класс TEmployee должен иметь свой собственный конструктор, обеспечивающий инициализацию класса-родителя и своих полей. Вот пример реализации конструктора класса TEmployee:

```
constructor TEmployee.Create(Name:TName;Dep:integer);
begin
inherited Create(Name);
FDepartment:=Dep;
end;
```

В приведенном примере директивой inherited вызывается конструктор родительского класса. После этого присваивается значение полю класса-потомка.

Полиморфизм — это возможность использования одинаковых имен для методов, входящих в различные классы. Концепция полиморфизма обеспечивает в случае применения метода к объекту применение именно того метода, который соответствует классу объекта.

Пусть определены три класса, один из которых является базовым для двух других:

```
type
// базовый класс TPerson = class
fname: string; // имя
constructor Create(name:string);
function info: string;
virtual;
end;
// производный от TPerson TStud = class(TPerson)
fgr:integer; // номер учебной группы
constructor Create(name:string;gr:integer);
function info: string; override; end;
```

```
// производный от TPerson TProf = class(TPerson)
fdep:string; // название кафедры
constructor Create(name:string;dep:string);
function info: string;
override;
end;
```

В каждом из этих классов определен метод info. В базовом классе при помощи директивы virtual метод info объявлен виртуальным. Объявление метода виртуальным дает возможность дочернему классу произвести замену виртуального метода своим собственным. В каждом дочернем классе определен свой метод info, который замещает соответствующий метод родительского класса (метод порожденного класса, замещающий виртуальный метод родительского класса, помечается директивой override). Ниже приведено определение метода info для каждого класса.

```
function TPerson.info:string;
begin
result := '';
end;
function TStud.info:string;
begin
result := fname + ' гр.' + IntTostr(fgr);
end;
function TProf.info:string;
begin
result := fname + ' каф.' + fdep;
end;
```

Так как оба класса порождены от одного и того же, базового, объявить список студентов и преподавателей можно так (здесь следует вспомнить, что объект — это указатель):

```
list: array[1..SZL] of TPerson;
```

Объявить подобным образом список можно потому, что Delphi позволяет указателю на родительский класс присвоить значение указателя на дочерний класс, поэтому элементами массива list могут быть как объекты класса TStud, так и объекты класса TProf.

Вывести список студентов и преподавателей можно применением метода info к элементам массива. Например, так:

```
st := '';
for i:=1 to SZL do // SZL — размер массива-списка
if list[i] o NIL
then st := st + list[i].Info
```

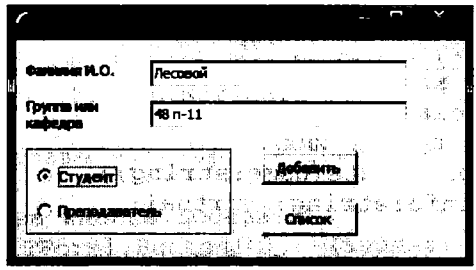


Рис. 7.1. Диалоговое окно программы

```
+ #13; ShowMessage (st);
```

Во время работы программы каждый элемент массива может содержать как объект типа TStud, так и объект типа TProf. Концепция полиморфизма обеспечивает применение к объекту именно того метода, который соответствует типу объекта.

Следующая программа формирует и выводит список студентов и преподавателей, используя рассмотренные выше объявления классов TPerson, TStud и TProf. Текст программы приведен в листинге 7.1, а диалоговое окно — на рис. 7.1.

Листинг 7.1 Демонстрация полиморфизма:

```
unit polimor_;
interface
uses
Windows, Messages, SysUtils, Classes,
Graphics, Controls, Forms, Dialogs, StdCtrls;
type
TForm1 = class(TForm) Edit1: TEdit;
Edit2: TEdit;
GroupBox1: TGroupBox;
RadioButton1: TRadioButton;
RadioButton2: TRadioButton;
Label1: TLabel;
Label2: TLabel;
Button1: TButton;
Button2: TButton;
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
```

```
  end;
  type
  // базовый класс
  TPerson = class
  fName: string; // имя
  constructor Create(name:string);
  function info:string; virtual;
  end;
  // класс Студент TStud = class(TPerson)
  fGr:integer; // номер группы
  constructor Create(name:string;gr:integer);
  function info:string;
  override;
  end;
  // класс Преподаватель
  TProf = class (TPerson)
  fdep:string; // название кафедры
  constructor Create(name:string;dep:string);
  function info:string;
  override;
  end;
  const
  SZL = 10; // размер списка
  var
  Form1: TForm1;
  List: array[1..SZL] of TPerson; // список
  n:integer =0; // кол-во людей в списке
  implementation
  {$R *.DFM}
  constructor TPerson.Create(name:string);
  begin
  fName := name; end;
  constructor TStud.Create(name:string;gr:integer);
  begin
  inherited create(name); // вызвать конструктор ба-
зового класса
  fGr := gr; end;
  constructor TProf.create(name:string; dep:string);
  begin
  inherited create(name); // вызвать конструктор ба-
зового класса
  fDep := dep; end;
  function TPerson.Info:string;
  begin
  result := fname; end;
```

```
function TStud.Info:string;
begin
result := fname + ' гр.' + IntToStr(fGr); end;
function TProf.Info:string;
begin
result := fname + ' каф.' + fDep;
end;
// щелчок на кнопке Добавить
procedure TForm1.Button1Click(Sender: TObject);
begin
if n < SZL then begin
// добавить объект в список
n:=n+1;
if Radiobutton1.Checked
then // создадим объект TStud
List[n]:=TStud.Create(Edit1.Text,StrToInt(Edit2.Text))
else // создать объект TProf
List[n]:=TProf.Create(Edit1.Text,Edit2.Text); //
очистить поля ввода
Edit1.Text := '' ; Edit2.Text := '';
Edit1.SetFocus; // курсор в поле Фамилия
end
else ShowMessage('Список заполнен!');
end;
// щелчок на кнопке Список
procedure TForm1.Button2Click(Sender: TObject);
var
i:integer; // индекс
st:string; // список
begin
for i:=1 to SZL do
if list[i] < > NIL then st:=st + list[i].info + 113;
ShowMessage('Список'+#13+st); end;
end.
```

Процедура TForm1.Button1click, которая запускается нажатием кнопки Добавить (Button1), создает объект List[n]класса TStud или TProf. Класс создаваемого объекта определяется состоянием переключателя RadioButton. Установка переключателя в положение Студент (RadioButton1) определяет класс TStud, а в положение Преподаватель (RadioButton2) — класс TProf. Процедура TForm1.Button2Click, которая запускается нажатием кнопки Список (Button2), применяя метод info к каждому объекту списка (элементу массива), формирует строку, представляющую собой весь список.

Задание

Для рассмотренного выше примера доработать программу таким образом, чтобы выводимые списки по желанию пользователя были отсортированы по алфавиту. В отчет включить задание, листинг программы и ответы на контрольные вопросы.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Дайте определение следующим понятиям: класс, объект, наследование, инкапсуляция, полиморфизм.
2. Как объявить базовый и производный классы?
3. Что называется порождением класса?
4. Какие задачи решает конструктор класса?
5. Что дает объявление метода виртуальным?

Глава 8

ВЫБОР СТРАТЕГИИ ТЕСТИРОВАНИЯ И РАЗРАБОТКА ТЕСТОВ

8.1. УРОВНИ ТЕСТИРОВАНИЯ

Тестированием называется выполнение программы в целях обнаружения ошибок. Отладкой называется локализация и исправление ошибок.

Модульное тестирование представляет собой процесс проверки отдельных программных процедур и подпрограмм, входящих в состав программ или подпрограммных систем. Модульное тестирование производится непосредственным разработчиком и позволяет проверять все внутренние структуры и потоки данных в каждом модуле. Этот вид тестирования является частью этапа разработки. При модульном тестировании выполняется набор тестов, определяемый разработчиком так, чтобы охват тестированием каждого модуля был не менее 70...75%. Элементы модульного тестирования:

- синтаксическая проверка — проверка с использованием некоторого инструментального средства для выявления синтаксических ошибок в программном коде;
- проверка соответствия стандартам кодирования — проверка кода на соответствие стандартам кодирования компании;
- технический обзор программного кода.

После успешного завершения модульного тестирования все измененные модули и наборы тестов сохраняются в базе данных проекта.

Интеграционное тестирование проводится для проверки совместной работы отдельных модулей и предшествует тестированию всей системы как единого целого. В ходе интеграционного тестирования проверяются связи между модулями, их совместимость и функциональность. Оно осуществляется независимым те-

стировщиком и входит в состав этапа тестирования. Элементы интеграционного тестирования:

- проверка функциональности — проверка соответствия отдельных функций, выполняемых совокупностями модулей, функциям, заданным в спецификациях требований;
- проверка промежуточных результатов — проверка всех промежуточных результатов и файлов на наличие и корректность;
- проверка интеграции — проверка корректности взаимной передачи модулями информации.

Ошибки, выявленные в ходе интеграционного тестирования, заносятся в базу данных ошибок. Результаты интеграционного тестирования включаются в отчет о ходе тестирования при завершении цикла тестирования.

Системное тестирование предназначено для проверки программной системы в целом, ее организации и функционирования на соответствие спецификациям требований заказчика. Его проводит независимый тестировщик после успешного завершения интеграционного тестирования.

Элементы системного тестирования:

- граничное тестирование — тестирование в граничных условиях;
- прогоночное тестирование — тестирование всех функциональных характеристик реальной работы системы;
- целевое тестирование — тестирование на целевой платформе (по возможности);
- проверка документации — проверка пользовательской документации на корректность;
- другие тесты, определяемые тестировщиком.

Ошибки, выявленные при системном тестировании, заносятся в базу данных проекта. Результаты системного тестирования включаются в отчет о ходе тестирования.

Выходное тестирование — завершающий этап тестирования, на котором проверяется готовность ПП для поставки заказчику. Данный вид тестирования проводит независимый тестировщик.

Элементы выходного тестирования:

- проверка инсталляции — проверка на ясность и корректность инструкций по инсталляции;

- проверка документации — проверка того, что вся необходимая документация полностью подготовлена и готова к передаче заказчику.

Ошибки, выявленные при выходном тестировании, заносятся в базу данных проекта. При успешном завершении выходного тестирования ПП поставляется заказчику вместе с отчетом о результатах тестирования.

Приемочное тестирование проводится организацией, отвечающей за инсталляцию, сопровождение программной системы и обучение конечного пользователя.

8.2. ТЕХНОЛОГИИ ТЕСТИРОВАНИЯ

Технология тестирования, которая применяется на этапе разработки программного обеспечения, называется *тестированием «стеклянного ящика»* (glass box). Иногда эту технологию еще называют тестированием «белого ящика» (white box) в противоположность классическому понятию «черного ящика» (black box).

При тестировании «черного ящика» программа рассматривается как объект, внутренняя структура которого неизвестна. Тестировщик вводит данные и анализирует результат, но он не знает, как именно работает программа. Подбирая тесты, специалист ищет интересные, с его точки зрения, входные данные и условия, способные привести к нестандартным результатам. Интересны для него прежде всего те представители каждого класса входных данных, при которых с наибольшей вероятностью могут проявиться ошибки тестируемой программы.

При тестировании «стеклянного ящика» ситуация совершенно иная. Тестировщик (в данном случае сам программист) разрабатывает тесты, основываясь на знании исходного кода, к которому он имеет полный доступ. В результате он получает следующие преимущества.

1. Направленность тестирования. Программист может тестировать программу по частям, разрабатывать специальные тестовые подпрограммы, которые вызывают тестируемый модуль и передают ему интересующие программиста данные. Отдельный модуль гораздо легче протестировать именно как «стеклянный ящик».

2. Полный охват кода. Программист всегда может определить, какие именно фрагменты кода работают в каждом тесте. Он видит, какие еще ветви кода остались непротестированными, и мо-

жет подобрать условия, в которых они будут протестированы. Ниже описано, как отслеживать степень охвата программного кода проведенными тестами.

3. Возможность управления потоком команд. Программист всегда знает, какая функция должна выполняться в программе следующей и каким должно быть ее текущее состояние. Чтобы выяснить, работает ли программа так, как он думает, программист может включить в нее отладочные команды, отображающие информацию о ходе ее выполнения, или воспользоваться для этого специальным программным средством, называемым отладчиком. Отладчик может отслеживать и менять последовательность выполнения команд программы, показывать содержимое ее переменных и их адреса в памяти, а также выполнять другие важные функции.

4. Возможность отслеживания целостности данных. Программисту известно, какая часть программы должна изменять каждый элемент данных. Отслеживая состояние данных (с помощью того же отладчика), он может выявить такие ошибки, как изменение данных не теми модулями, их неверная интерпретация или неудачная организация. Программист может и самостоятельно автоматизировать тестирование.

5. Ви́дение внутренних граничных точек. В исходном коде видны те граничные точки программы, которые скрыты от взгляда извне. Например, для выполнения определенного действия может быть использовано несколько совершенно различных алгоритмов и, не заглянув в код, невозможно определить, какой из них выбрал программист. Еще одним типичным примером может стать проблема переполнения буфера, используемого для временного хранения входных данных. Программист сразу может сказать, при каком количестве данных буфер переполнится, и ему не нужно при этом проводить тысячи тестов.

6. Возможность тестирования, определяемого выбранным алгоритмом. Для тестирования обработки данных, использующей очень сложные вычислительные алгоритмы, могут понадобиться специальные технологии. В качестве классических примеров можно привести преобразование матрицы и сортировку данных. Тестировщику, в отличие от программиста, нужно точно знать, какие алгоритмы используются, поэтому приходится обращаться к специальной литературе.

Тестирование «стеклянного ящика» — часть процесса программирования. Программисты выполняют эту работу постоянно, они тестируют каждый модуль после его написания, а затем еще раз

после интеграции его в систему. При выполнении модульного тестирования можно использовать технологию или структурного, или функционального тестирования, или одновременно и ту, и другую.

Структурное тестирование является одним из видов тестирования «стеклянного ящика». Его главная идея — правильный выбор тестируемого программного пути. В противоположность ему функциональное тестирование относится к категории тестирования «черного ящика». Каждая функция программы тестируется путем ввода ее входных данных и анализа выходных. При этом внутренняя структура программы учитывается очень редко.

Хотя структурное тестирование имеет гораздо более мощную теоретическую основу, большинство тестировщиков предпочитают функциональное тестирование. Структурное тестирование лучше поддается математическому моделированию, однако это вовсе не означает, что оно эффективнее. Каждая из технологий позволяет выявить ошибки, пропускаемые в случае использования другой. С этой точки зрения их можно назвать одинаково эффективными.

Тесты разрабатывают в определенном порядке. При этом по внешней спецификации различают тесты:

1) для каждого класса входных данных;

2) для граничных и особых значений входных данных.

Контролируется, все ли классы выходных данных при этом проверены, и добавляются при необходимости нужные тесты. Разрабатываются тесты для тех функций, которые не проверяются в п. 1. По тексту программы проверяется, все ли условные переходы выполнены в каждом направлении. При необходимости добавляются новые тесты. Аналогично проверяется, проходятся ли пути для каждого цикла: без выполнения тела, с однократным и максимальным числом повторений. Готовятся тесты, проверяющие исключительные ситуации, недопустимые входные данные, аварийные ситуации.

8.3. ПРОГРАММНЫЕ ОШИБКИ

Все программные ошибки можно отнести к соответствующим категориям. Рассмотрим встречающиеся наиболее часто.

1. Функциональные недостатки присущи программе, если она не делает того, что должна, выполняет одну из своих функций

плохо или не полностью. Функции программы должны быть подробно описаны в ее спецификации, и именно на основе утвержденной спецификации тестировщик строит свою работу.

2. Недостатки пользовательского интерфейса. Оценить удобство и правильность работы пользовательского интерфейса можно только в процессе работы с ним. Желательно, чтобы в этой работе принимал участие сам пользователь. Этого можно добиться с помощью разработки прототипа ПП, на котором проводятся обкатка и согласование всех требований к пользовательскому интерфейсу с дальнейшей фиксацией их в спецификации требований. После утверждения спецификации требований любые отклонения от нее или невыполнение последних являются ошибкой. Это в полной мере касается и пользовательского интерфейса.

3. Недостаточная производительность. При разработке некоторого программного продукта очень важной его характеристикой может оказаться скорость работы, иногда этот критерий задается в требованиях заказчика. Плохо, если у пользователя создается впечатление, что программа работает медленно, особенно если конкурирующие программы работают ощутимо быстрее; но еще хуже, если программа не удовлетворяет заданным в спецификации требований характеристикам. Эта ошибка должна быть устранена.

4. Некорректная обработка ошибок. Процедуры обработки ошибок — очень важная часть программы. Правильно определив ошибку, программа должна выдать о ней сообщение. Отсутствие такого сообщения является ошибкой в работе программы.

5. Некорректная обработка граничных условий. Существует много различных граничных ситуаций. Любой аспект работы программы, к которому применимы понятия «больше» или «меньше», «раньше» или «позже», «первый» или «последний», «короче» или «длиннее», обязательно должен быть проверен на границах диапазона. Внутри диапазонов программа может работать прекрасно, а вот на их границах могут происходить самые неожиданные ситуации, которые, в свою очередь, приводят к ошибкам в работе ПП.

6. Ошибки вычислений. Сюда относятся ошибки, вызванные неправильным выбором алгоритма вычислений, неправильными формулами или формулами, неприменимыми к обрабатываемым данным. Самыми распространенными являются ошибки округления.

7. Ошибки управления потоком. По логике работы программы вслед за первым действием должно быть выполнено второе. Если вместо этого выполняется третье или четвертое действие, значит, в управлении потоком допущена ошибка.

8. Ситуация гонок. Предположим, в системе ожидаются два события: А и Б. Если первым наступит событие А, то выполнение программы продолжится, а если Б, то в работе программы произойдет сбой. Разработчики предполагают, что первым всегда должно быть событие А, и не ожидают, что Б может выиграть гонки и наступить раньше. Такова классическая ситуация гонок.

Тестировать ситуации гонок довольно сложно. Наиболее типичны они для систем, где параллельно выполняются взаимодействующие процессы и потоки, а также для многопользовательских систем реального времени. Ошибки в таких системах трудновоспроизводимы, и на их выявление обычно требуется очень много времени.

9. Перегрузки. Сбои в работе программы могут происходить из-за нехватки памяти или отсутствия других необходимых системных ресурсов. У каждой программы свои пределы, программа может не справляться с повышенными нагрузками, например со слишком большими объемами данных. Вопрос в том, как поведет себя программа при перегрузках.

10. Некорректная работа с аппаратурой компьютера. Программы могут посылать аппаратным устройствам неверные данные, игнорировать их сообщения об ошибках, пытаться использовать устройства, которые заняты или отсутствуют. Даже если нужное устройство неисправно, программа должна понять это, а не «за-

Таблица 8.1. Виды программных ошибок и способы их обнаружения

Виды ошибок	Способы обнаружения
Синтаксические	Статистический контроль и диагностика компиляторами и компоновщиком
Ошибки выполнения, выявляемые автоматически: переполнение, защита памяти несоответствие типов зацикливание	Динамический контроль: аппаратурой процессора run-time системы программирования операционной системой — по превышению лимита времени
Программа не соответствует спецификации	Целенаправленное тестирование
Спецификация не соответствует требованиям	Испытания, бета-тестирование

висать» при попытке к нему обратиться. Виды программных ошибок и способы их обнаружения представлены в табл. 8.1.

8.4. ВИДЫ ТЕСТИРОВАНИЯ

Тестирование переходов между состояниями. В каждой интерактивной программе осуществляются переходы из одного очевидного состояния в другое. Простейшим примером может служить меню: после запуска программы имеется один перечень команд, после выбора одной из них состояние программы меняется и в меню появляются команды, доступные в новом состоянии.

Необходимо протестировать каждую предлагаемую программой опцию, каждую команду меню. Команда 10 может быть доступна в режиме, открываемом по команде 9 или по команде 22. В этом случае команду 10 придется протестировать дважды — в обоих режимах. Однако команд меню, всевозможных режимов программы и путей перехода в эти режимы может быть так много, что протестировать их все просто нереально. Поэтому, отбирая тесты для проверки путей выполнения программы, лучше всего руководствоваться следующими принципами:

- тестировать все наиболее вероятные последовательности действий пользователей;
- если можно предположить, что действия пользователя в одном режиме могут влиять на представление данных или набор предоставляемых программой возможностей в другом режиме, тестировать эти действия;
- кроме проведения самых необходимых тестов из описанных выше поработать с программой в произвольном режиме, случайным образом выбирая путь ее выполнения.

Переходы между состояниями могут быть гораздо сложнее, чем просто выбор команд меню. Содержимое и структура очередной формы ввода данных могут зависеть от информации, введенной в предыдущей форме, значения одних полей могут определять допустимые значения других, ввод определенной информации может инициировать серию дополнительных запросов. Например, при вводе чисел от 1 до 99 программа выводит одну форму запроса, обращенного к пользователю, а при вводе любых других чисел — другую. В этом случае вместе с классами эквивалентности и их граничными значениями следует проанализиро-

вать и возможные пути выполнения программы, чтобы составить действительно полноценный набор тестов.

Очень полезно составление схем меню. В подобной схеме отражаются все состояния программы и команды, вызывающие переходы между этими состояниями. В нее включаются команды, активизируемые через меню, графические средства (например, различные кнопки) и команды, выполняемые после нажатия на определенные клавиши. Например, в схеме может быть показан путь от меню «Файл» к команде «Открыть», затем к диалоговому окну «Открытие файла» и назад, к основному состоянию программы. Особенно удобны подобные схемы в случае если определенное диалоговое окно можно открыть несколькими способами и выйти из него в несколько различных режимов. В этом случае можно нарисовать на схеме все направления переходов и по ним протестировать программу. Это более надежный способ, чем работа с программой без всякого плана с риском пропустить важные взаимосвязи ее состояний.

Условия гонок и другие временны́е зависимости. Пробуйте вмешаться в работу программы, когда она выполняет переход между двумя состояниями, понажимайте на клавиши, особенно командные. Попробуйте выбрать какие-либо пункты меню, когда программа выполняет операции обработки данных или ввода-вывода, предложите программе ввести или вывести параллельно еще какую-нибудь информацию. Например, во время печати файла попросите ее распечатать еще один.

Если в программе определены ситуации тайм-аута, когда она ждет определенного события в течение заданного времени, а затем переходит в другое состояние, проверьте ее реакцию на действия пользователя, запросы системы или наступление ожидаемого события на границах интервала тайм-аута. Посмотрите, что будет, если событие произойдет за секунду до того, как программа должна прекратить ожидать его, или через секунду после этого.

Протестируйте систему при повышенной нагрузке. В мультизадачной среде запустите несколько других программ и посмотрите, как поведет себя ваша, успешно ли она справится со своей работой. Отправьте большой файл на принтер, чтобы процессор все время переключался на обслуживание печати. Подключите различные внешние устройства и заставьте их генерировать прерывания так часто, как только удастся. Короче говоря, замедлите и нагрузите компьютер, насколько это возможно. В результате ваша программа будет выполняться медленнее и, быстро вводя данные, можно попробовать превысить ее возможности приема.

Если в нормальном режиме работы сбоя программы добиться не удается, это может получиться при повышенной нагрузке.

Выполняя «стандартное» тестирование программы при сильно повышенной нагрузке, можно столкнуться с совершенно неожиданными ситуациями гонок. Если окажется, что программа в этом отношении уязвима, то необходимо провести в таких условиях полный цикл тестирования. Главная задача — обеспечить такую надежность разрабатываемого программного обеспечения, чтобы оно работало, пусть медленно, но без сбоев в любой системе и при любых дополнительных нагрузках. По крайней мере, необходимо совершенно точно выяснить, какие конфигурации системы являются предельными для эксплуатации программы.

Нагрузочные испытания. Важно не забыть протестировать те ограничения возможностей программного продукта, которые определены в его документации. Откройте максимальное число файлов или других структур данных, с которыми программа может работать, попробуйте подольше эксплуатировать ее в таком состоянии. Если в документации ограничения не описаны, но существуют логически допустимые значения каких-либо параметров, то следует проверить и их. Если программа не справится с большим числовым значением, которое пользователь может ввести, то составляется отчет об ошибке. Если же программа спокойно принимает и обрабатывает как очень маленькие, так и очень большие значения параметров, возможно, ограничений на них нет.

Необходимо проверить, как ведет себя программа, когда исчерпываются различные аппаратные ресурсы, например, переполняется диск или в принтере заканчивается бумага. Выясните, что произойдет, когда в системе останется очень мало свободной памяти.

Нагрузочное тестирование — это, по сути, один из видов тестирования граничных условий. Схема его проведения абсолютно аналогична. Сначала программу запускают в условиях, в которых она должна работать, а затем — в условиях, для которых она не предназначена. Имеет смысл проверить и различные комбинации условий. Вполне возможно, что, справившись с различными повышенными нагрузками по отдельности, программа не выдержит их все вместе. Нагрузив систему, проведите не просто один — два теста, а длительное и обстоятельное тестирование. Поэксплуатируйте программу в таких условиях некоторое время, возможно, сбой не сразу, но все же произойдет.

Прогнозирование ошибок. Иногда тестировщик предполагает, что определенный класс тестов вызовет сбой программы, хотя и не может это логически обосновать. Доверяйте своей интуиции и

обязательно включайте подобные тесты в общий план. Существует целый ряд ситуаций и значений, которые, хотя и не являются граничными, но часто вызывают программные сбои. Типичным примером таких значений является 0. Не стоит тратить время на поиски обоснований того, почему определенное входное значение или место программы кажется вам подозрительным, просто протестируйте его.

Случается, что в сложных ситуациях интуиция подсказывает гораздо лучшую тактику тестирования, чем тривиальная логика. Бывает, что срабатывает и ассоциативная связь: вы уже находили ошибку в подобных обстоятельствах, хотя можете этого даже не помнить. Как бы там ни было, доверяйте своему внутреннему чувству и учитесь к нему прислушиваться: с опытом оно будет становиться все более развитым и надежным.

Тестирование функциональной эквивалентности. При тестировании функциональной эквивалентности сравниваются результаты вычислений разными программами одной и той же математической функции. Термин «функциональная эквивалентность» не имеет ничего общего с термином «классы эквивалентности». Если обе программы при вычислении одной и той же функции дают одинаковые результаты, значит, в них применены эквивалентные методы вычислений.

Предположим, что тестируется программа, которая вычисляет математическую функцию и печатает результат. Это может быть простая тригонометрическая функция или гораздо более сложная функция, инвертирующая матрицу или возвращающая коэффициенты для построения кривой, отражающей некоторый набор данных. Обычно в таких случаях можно найти другую программу, выполняющую те же действия, и при этом достаточно надежную и проверенную временем. Обеим программам предлагается обработать одинаковые наборы входных данных. Если результаты совпадут, значит, тестируемая программа работает правильно.

Задание

Составить тесты для программы по своему варианту сквозной задачи. При тестировании методом «стеклянного ящика» использовать схемы алгоритмов, разработанные и уточненные в предыдущих практических работах. Выбрать несколько алгоритмов для тестирования и обозначить буквами или цифрами ветви этих алгоритмов. Выписать пути алгоритма, которые должны быть проверены тестами для выбранного метода тестирования. Записать тесты, которые позволят пройти по путям алгоритма.

Для метода «черного ящика» тесты должны быть составлены по следующим категориям:

1) полнота решения функциональных задач;

2) тестирование на предельных объемах нагрузки входного потока;

3) корректность использования ресурсов;

4) оценка производительности;

5) эффективность защиты от искажения данных и некорректных действий;

6) проверка инсталляции и конфигурации на разных платформах;

7) корректность документации.

Результат оформить в виде отчета, включающего базу (аппаратная и операционная платформа) и результаты тестирования. Последние оформляют в виде табл. 8.2.

Таблица 8.2. Результаты тестирования

Метод тестирования	Ожидаемый результат	Фактический результат

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Дайте определение технологиям тестирования «белого ящика» и «черного ящика».
2. Перечислите основные виды тестирования
3. Что означает ситуация гонок? Каким образом ее можно протестировать?
4. Что является программной ошибкой? Какие категории программных ошибок вы знаете?
5. Что вы понимаете под тестированием переходов между состояниями программного продукта?
6. Для чего проводится нагрузочное тестирование ПП?
7. Что вы думаете о прогнозировании ошибок?
8. Дайте определение функциональной эквивалентности.
9. Каким образом можно организовать случайный ввод?

Глава 9

ПРИМЕНЕНИЕ КОМПОНЕНТНОГО ПОДХОДА В ПРОГРАММИРОВАНИИ. ИСПОЛЬЗОВАНИЕ COM-ТЕХНОЛОГИЙ

9.1. ОСНОВНЫЕ ПОНЯТИЯ COM-ТЕХНОЛОГИИ И OLE АВТОМАТИЗАЦИИ

Компонентный подход предполагает построение программного обеспечения из отдельных компонентов физически отдельно существующих программных частей программного обеспечения, которые взаимодействуют между собой через стандартизированные двоичные интерфейсы. В отличие от обычных объектов объекты-компоненты можно собрать в динамически вызываемые библиотеки или исполняемые файлы, распространять в двоичном виде (без исходного текста) и использовать в любом языке программирования, поддерживающем соответствующую технологию. Компонентный подход лежит в основе технологий, разработанных на базе COM (Component Object Model — компонентная модель объектов), и технологии создания распределенных приложений CORBA (Common Object Request Broker Architecture — общая архитектура с посредником обработки запросов объектов). Эти технологии используют сходные принципы и различаются особенностями их реализации.

Технология COM фирмы Microsoft является развитием технологии OLE 1 (Object Linking and Embedding — связывание и внедрение объектов), которая использовалась в ранних версиях Windows для создания составных документов. OLE позволяет программистам создавать приложения для работы с составными документами, представляющие собой динамически связанные структуры, отдельные части которых могут разрабатываться в различных программах. Эта технология появилась как OLE 1 в Windows 3.1 и означала, что пользователь мог создавать сложные составные документы, в которых содержались объекты различного происхождения. Внедренные объекты могли редактироваться простым щелчком клавиши мыши по соответствующему элементу данных. Например, нужно было дважды щелкнуть по электрон-

ной таблице Excel, встроенной в документ редактора Word, и в отдельном окне запускался Excel с загруженным рабочим листом, готовым к редактированию. После завершения редактирования Excel позволял сохранить изменения во внедренном в документ Word объекте Excel.

Другой особенностью было связывание объектов — это позволяло связать электронную таблицу с документом Word (по сути, внутри документа Word хранился указатель на электронную таблицу). Если данные в оригинале электронной таблицы обновлялись, то при следующей загрузке документа Word ссылка обновляла документ и отражала в нем проведенные изменения.

Дальнейшее развитие внедрение и связывание получили в OLE 2.0. Основой этого усовершенствованного подхода явилась компонентная модель объекта (COM). Это модель объекта в системном обеспечении, которая предусматривает полную совместимость во взаимодействии между компонентами, написанными разными компаниями на разных языках. Ключом к успеху является модульность этих компонентов. Они могут покупаться, модернизироваться или заменяться поодиночке или группами, причем это никак не влияет на работу целого.

Новая особенность, появившаяся в OLE 2.0, — это автоматизация OLE, которая обеспечивает доступ к объектам приложения и манипуляцию с ними извне. Такие объекты, предоставленные для внешнего пользования, называются *автоматными объектами* OLE. Типы объектов, которые могут быть экспонированы в качестве автоматного объекта, — документ, абзац или предложение. Электронная таблица могла бы экспонировать таблицу, диаграмму, ячейку или группу ячеек.

Главное отличие автоматных объектов от обычных объектов OLE в том, что автоматные объекты доступны только программно, они создаются и используются при помощи программного кода и, следовательно, в принципе временны. Они не могут быть внедрены или связаны. Они могут существовать только в течение времени выполнения программ и не видны непосредственно конечному пользователю.

Существуют два типа автоматных серверов: внутри процесса и вне процесса (их еще называют локальными). Сервер *внутри процесса* является DLL (динамически связываемой библиотекой), которая экспортирует автоматные объекты. Поскольку автоматные объекты поставляются из DLL, а не из других приложений, они являются частью приложения клиента. Это избавляет от больших накладных расходов, сопутствующих каждому вызову автоматно-

го сервера. Сервер *вне процесса* — это автономный исполняемый файл, экспортирующий автоматные объекты.

Автоматизация OLE, чтобы функционировать правильно, опирается на три различных источника информации: классы, документы, элементы.

Класс OLE определяет приложение сервера, которое создает автоматный объект. Например, если класс объекта — документ .doc, то он был создан с помощью Word.

Документы OLE ссылаются на исходный файл, содержащий данные объекта OLE. Любой документ должен быть связанным объектом (т.е. не внедренным), потому что данные связанных документов должны храниться в файле.

Элемент OLE определяет, какая часть документа содержит данные, подлежащие связыванию или внедрению в другой документ. Этим способом вы можете вместо целого документа связывать с другими документами лишь небольшую часть его данных. Элементы OLE позволяют сократить размеры объектов.

Итак, что же такое COM? Чтобы ответить на этот вопрос, сначала нужно понять, каким образом одна часть программного обеспечения должна получать доступ к сервисам, предоставляемым другой частью. Приложения, скомпонованные с библиотекой, могут пользоваться ее сервисами, вызывая функции из этой библиотеки. Приложение также может использовать сервисы другого приложения, являющегося совершенно отдельным процессом. В этом случае два таких локальных процесса взаимодействуют посредством некоего механизма связи, который обычно требует определения протокола между этими приложениями (набор сообщений, позволяющий одному приложению выдавать запросы, а другому соответствующим образом отвечать на них). Например, приложение использует сервисы операционной системы или сервисы программного обеспечения, выполняемого на другой машине. Проблема в том, что одна часть программного обеспечения должна получить доступ к сервисам, предоставляемым другой частью. Но в каждом отдельном случае механизм доступа разный.

В технологии COM приложение предоставляет для использования свои службы, применяя для этого объекты COM. Одно приложение содержит, как минимум, один объект. Каждый объект имеет один или несколько интерфейсов. Каждый интерфейс объединяет методы объекта, которые обеспечивают доступ к свойствам (данным) и выполнение операций. Обычно в интерфейсе объединяются все методы, выполняющие операции одного типа или работающие с однородными свойствами.

Клиент получает доступ к службам объекта только через интерфейс и его методы. Этот механизм является ключевым. Клиенту достаточно знать несколько базовых интерфейсов, чтобы получить исчерпывающую информацию о составе свойств и методов объекта. Согласно спецификации COM уже созданный интерфейс не может быть изменен ни при каких обстоятельствах. Это гарантирует постоянную работоспособность приложений на основе COM, невзирая на любые модернизации.

Объект всегда работает в составе сервера COM. Сервер может быть динамической библиотекой или исполняемым файлом. Объект может иметь собственные свойства и методы или использовать данные и службы сервера. Для доступа к методам объекта клиент должен получить указатель на соответствующий интерфейс. Для каждого интерфейса существует собственный указатель. После этого клиент может использовать службы объекта, просто вызывая его методы. Доступ к свойствам объектов осуществляется только через его методы.

Предположим, что объект COM встроен в электронную таблицу. Взаимодействие между клиентом и объектом обеспечивается базовыми механизмами COM; при этом от клиента скрыто, где именно расположен объект: в адресном пространстве того же процесса, в другом процессе или на другом компьютере, поэтому с точки зрения разработчика клиентского программного обеспечения использование функций электронной таблицы выглядит как обычное обращение к методу объекта. Возникает вопрос: как проходит создание и инициализация объекта COM при первом обращении клиента? Операционная система не должна самостоятельно создавать экземпляры всех зарегистрированных в ней классов на случай, если один из них понадобится. А ведь для работы объекта требуются еще и серверы. Представьте, что каждый раз при загрузке Windows настойчиво запускает Word, Excel, Internet Explorer и т.д.

Любой объект COM является обычным экземпляром некоторого класса, описывающего его свойства и методы. Информация обо всех зарегистрированных и доступных в данной операционной системе классах COM собрана в специальной библиотеке COM, которая используется для запуска экземпляра класса-объекта.

Сначала клиент обращается к библиотеке COM, передавая ей имя требуемого класса и необходимого в первую очередь интерфейса. Библиотека находит нужный класс и сначала запускает сервер, который затем создает объект-экземпляр класса. После этого библиотека возвращает клиенту указатели на объект и ин-

терфейс. В последующей работе клиент может обращаться непосредственно к объекту и его интерфейсам.

После создания наступает очередь инициализации — объект должен загрузить необходимые данные, считать настройки из системного реестра и т.д. Может возникнуть ситуация, когда одновременно несколько клиентов обращаются к одному объекту. При соответствующих настройках для каждого клиента создается отдельный экземпляр класса. За выполнение этой операции отвечает специальный объект COM, который называется *фабрикой класса.*

Возникает вопрос: как клиент может получить информацию об объекте? Например, разработчик клиентского ПО знает, что электронная таблица создана в соответствии со спецификацией COM, но не имеет понятия об объектах COM, которые предоставляют клиентам ее службы. Для разрешения подобных ситуаций разработчик объекта COM может распространять вместе с объектом информацию о типе. Она включает сведения об интерфейсах, их свойствах и методах, параметрах методов. Эта информация содержится в библиотеке типов, которая создается при помощи специального языка описания интерфейса (Interface Definition Language, IDL).

Теперь рассмотрим механизм создания объектов COM в Delphi. Как говорилось выше, объект COM должен обеспечивать возможность создания произвольного числа интерфейсов, где под *интерфейсом* понимается некоторое объединение методов, доступ к которым осуществляется через указатель на интерфейс.

Реализовать такое требование напрямую в рамках стандартных подходов объектно-ориентированного программирования довольно сложно. В Delphi такая реализация обеспечивается следующим образом. Сам объект COM описывается обычным классом TComObject, который порожден непосредственно от TObject. Все свойства и методы, реализующие предназначение объекта, объявляются и описываются в его объявлении, поэтому сам класс нового объекта COM принципиально ничем не отличается от любого другого. Класс TComObject обеспечивает выполнение базовых функций объекта, которые и делают его объектом COM. Ниже приводится пример обращения к таблице Excel через механизм автоматизации OLE. Использование автоматизации OLE применительно к Excel и Word упрощается тем, что в эту программу встроена справка по VBA, содержащая описание всех объектов, свойств и методов. Так что разработчик просто получает возможность работать с ними из приложения Delphi.

9.2. ПРИМЕР ОБРАЩЕНИЯ К ТАБЛИЦЕ EXCEL ЧЕРЕЗ МЕХАНИЗМ АВТОМАТИЗАЦИИ OLE В DELPHI

Для того чтобы установить связь с сервером MS Excel в редакторе кода в раздел uses нужно добавить модуль ComObj. В этом модуле описаны все необходимые функции для работы с COM-объектами. Ниже приводится пример программного кода вывода в MS Excel списка, который хранится в таблице BookTable:

```
  procedure TMainForm.ExcelButtonClick(Sender:
TObject);
  var
  XLApp,Sheet,Colum:Variant;
  index,i:Integer;
  begin
  XLApp:= CreateOleObject('Excel.Application');
  XLApp.Visible:=true;
  XLApp.Workbooks.Add(-4167);
  XLApp.Workbooks[1].WorkSheets[1].Name:='Отчет';
  Colum:=XLApp.Workbooks[1].WorkSheets['Отчет'].Columns;
  Colum.Columns[1].ColumnWidth:=20;
  Colum.Columns[2].ColumnWidth:=20;
  Colum.Columns[3].ColumnWidth:=20;
  Colum.Columns[4].ColumnWidth:=20;
  Colum.Columns[5].ColumnWidth:=20;

  Colum:=XLApp.Workbooks[1].WorkSheets['Отчет'].Rows;
  Colum.Rows[2].Font.Bold:=true;
  Colum.Rows[1].Font.Bold:=true;
  Colum.Rows[1].Font.Color:=clBlue;
  Colum.Rows[1].Font.Size:=14;

  Sheet:=XLApp.Workbooks[1].WorkSheets["Отчет"];
  Sheet.Cells[1,2]:='Список товаров';
  Sheet.Cells[2,1]:='Наименование';
  Sheet.Cells[2,2]:='Ед.изм.';
  Sheet.Cells[2,3]:='Цена';
  Sheet.Cells[2,4]:='Остаток';
  index:=3;
  DataModule1.BookTable.First;
  for i:=0 to DataModule1.BookTable.RecordCount-1 do
  begin
  Sheet.Cells[index,1]:=DataModule1.BookTable.Fields.Fields[1].
```

```
AsString;
Sheet.Cells[index,2]:=DataModule1.BookTable.Fields.Fields[2].
AsString;
Sheet.Cells[index,3]:=DataModule1.BookTable.Fields.Fields[3].
AsInteger;
Sheet.Cells[index,4]:=DataModule1.BookTable.Fields.Fields[5].A
sInteger;
Inc(index);
DataModule1.BookTable.Next;
end;
end;
```

Первая строка кода создает объект Excel и записывает его в переменную XLApp:

```
XLApp:= CreateOleObject('Excel.Application');
```

Функция CreateOleObject позволяет наладить связь с другим приложением по технологии COM. Через эту связь можно передавать данные в другие приложения. Для этого программа, с которой происходит соединение, должна иметь соответствующие возможности для получения данных извне (как, например, Word или Excel), поэтому нам должны быть известны функции, с которыми можно работать.

Вторая строка кода заставляет отобразить Excel. Потом добавляется новая рабочая книга:

```
XLApp.Workbooks.Add(-4167);.
```

Число в скобках — константа, которая означает создание книги, и ее изменить нельзя. Подробнее обо всех константах можно почитать в руководстве разработчика. Дальше формируется название данной книги:

```
XLApp.Workbooks[1].WorkSheets[1].Name:='Отчет';
```

Это действие необязательно, но желательно, потому что название по умолчанию будет «Лист 1» .

Теперь у нас Excel запущен и создана новая книга, можно переходить к переносу данных. Но прежде чем это сделать, нужно отформатировать колонки и строки; для этого надо получить указатель на колонки рабочей книги:

```
Colum:=XLApp.Workbooks[1].WorkSheets['Отчет'].Columns;
```

Результат записывается в переменную Colum типа Variant. Теперь последовательно изменяем ширину колонок (ColumnWidth):

```
Colum.Columns[1].ColumnWidth:=20;
```

После этого в ту же переменную записываем указатель на строки рабочей книги:

```
Colum:=XLApp.Workbooks[1].WorkSheets['Отчет'].Rows;
```

Для первых двух строк отчета устанавливаем жирный шрифт:

```
Colum.Rows[2].Font.Bold:=true;
Colum.Rows[1].Font.Bold:=true;
```

В квадратных скобках теперь номер строки. Далее устанавливаем цвет первой строки в синий и размер шрифта равным 14:

```
Colum.Rows[1].Font.Color:=clBlue;
Colum.Rows[1].Font.Size:=14;
```

Форматирование окончено, теперь можно передавать (выводить) данные. Для этого получаем указатель на лист:

```
Sheet:=XLApp.Workbooks[1].WorkSheets['Отчет'];
```

Для того чтобы вывести данные, нужно просто присвоить значение в

```
Sheet.Cells[строка, колонка]
```

Рассмотрим код вывода данных таблицы:

```
index:=3;
DataModule1.BookTable.First;
for i:=0 to DataModule1.BookTable.RecordCount-1 do
begin
Sheet.Cells[index,1]:=DataModule1.BookTable.Fields.Fields[1].
AsString;
Sheet.Cells[index,2]:=DataModule1.BookTable.Fields.Fields[2].
AsString;
Sheet.Cells[index,3]:=DataModule1.BookTable.Fields.Fields[3].
AsInteger;
Sheet.Cells[index,4]:=DataModule1.BookTable.Fields.Fields[5].
AsInteger;
Inc(index);
DataModule1.BookTable.Next;
end;
```

Переменная index отображает, в какую строку таблицы Excel сейчас должны выводиться данные. Поскольку первые две строки у нас уже заняты заголовками для отчета, то данные нужно выводить с третьей строки (index=3). Строки и таблицы в Excel нумеруются, начиная с единицы, а не с нуля, как в остальных таблицах и массивах.

После этого мы переходим на первую строку таблицы в базе данных с помощью метода First компонента ADOTable. Это необ-

ходимо, потому что пользователь может перед нажатием кнопки отчета выделить любую строку в середине таблицы, и в этом случае отчет пойдет от этой выделенной строки.

Теперь подготовлены условия для запуска цикла, в котором будут перебираться все строки и информация из них будет попадать в Excel . Цикл запускается, начиная с нулевой строки, и продолжается до достижения значения количества строк в таблице:

```
For i:=0 to DataModule1.BookTable.RecordCount-1
```

Таким образом, мы перебираем все записи и каждую запись записываем в Excel, последовательно все столбцы (1—5).

На этом завершим обзор типовых операций, осуществляемых с помощью автоматизации OLE. В целом использование автоматизации OLE очень близко к VB и все не рассмотренные выше возможности работы с Excel можно реализовать, познакомившись со встроенной в Excel справкой по программированию.

Мы рассмотрели работу с Excel. Для работы с Word автоматизация OLE используется аналогично, только объекты, их свойства и методы отличаются от Excel. Вызов нового экземпляра Word может осуществляться следующим образом:

```
uses ComObj;
var
WordApp:Variant;
...
WordApp:= CreateOleObject('Word.Application');
WordApp.Visible:=true;
```

Как видно, единственное отличие от соединения с Excel — другое имя класса, а в остальном все рассмотренные приемы работы с Excel применимы и к Word с учетом особенностей.

Задание

В Delphi установить связь с сервером MS Excel/Word. Для ввоего варианта задания разработать программный модуль вывода отчета в MS Excel/Word.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Охарактеризуйте технологию COM.
2. Опишите этапы развития технологии OLE.
3. Каким образом используется автоматизация OLE применительно к Excel и Word в Delphi?

Глава 10

СОЗДАНИЕ ДИНАМИЧЕСКОЙ БИБЛИОТЕКИ ПРИ КОМПОНЕНТНОМ ПОДХОДЕ В ПРОГРАММИРОВАНИИ

10.1. ОСНОВНЫЕ СВЕДЕНИЯ О ДИНАМИЧЕСКИХ БИБЛИОТЕКАХ

Объект COM всегда реализуется внутри сервера. Сервер может быть либо динамически присоединяемой библиотекой, подгружаемой во время работы приложения, либо отдельным самостоятельным процессом.

Динамически присоединяемая библиотека (DLL), или динамическая библиотека, — это специального вида исполняемый файл, используемый для хранения функций и ресурсов отдельно от исполняемого файла. Обычно, когда разрабатывается программа и создаются функции, ресурсы, все они компонуются в исполняемый файл. Когда же используется DLL, то вызываемые из нее функции используются модулем в процессе его выполнения. DLL делает полезные, часто используемые функции доступными сразу для многих приложений одновременно, хотя работа ведется только с одной ее копией на диске и в памяти. Обычно DLL не загружается в память, пока это не станет необходимым, но, будучи однажды загружена, она делает свои функции и ресурсы доступными для любой программы.

Создание DLL повышает также гибкость программ. Например, можно создать несколько библиотек, содержащих все используемые приложением тексты, — надписи, подсказки и т.п. Каждая из этих библиотек может содержать тексты на том или ином языке: русском, английском, немецком. Тогда, в зависимости от того, какую из этих библиотек будет применять пользователь, программы будут общаться с ним на соответствующем языке.

С помощью DLL можно облегчить модификацию приложений. Пусть имеются какие-то функции, которые реализованы обычным образом и включены в приложение. Это приложение распространяется пользователям, а потом появляются какие-то новые возможности, которые могли бы расширить эти функции, напри-

мер, новые форматы файлов, на которые можно было бы настроить функции. Чтобы реализовать эти новые возможности, придется переписать прежние функции, скомпилировать новое приложение и разослать всем пользователям. А если функции реализованы в виде DLL, то перекомпиляция приложений не потребуется, достаточно создать новую небольшую DLL с тем же именем и теми же именами функций, реализованных по-новому, и разослать пользователям только эту новую DLL.

Еще одно преимущество DLL в том, что они могут использоваться приложениями, написанными на других алгоритмических языках. Например, вы можете использовать библиотеки, написанные на C++, Visual Basic, Access Basic и др. Библиотеки, созданные в Delphi, смогут использовать любые системы, которые умеют вызывать DLL, независимо от того, на каких алгоритмических языках они написаны.

10.2. ПРИМЕР СОЗДАНИЯ ДИНАМИЧЕСКОЙ БИБЛИОТЕКИ

Рассмотрим пример создания DLL, в которую поместим форму диалога, запрашивающего у пользователя его имя.

Создание DLL начните с выполнения команды File/New/Other и выбора в окне New Items на странице New пиктограммы DLL Wizard — Мастера DLL. В результате в окне редактора появится текст:

```
library Project1;
{ Important note about DLL memory management: ShareMem must be the
  first unit in your library's USES clause AND your project's (select
  Project-View Source) USES clause if your DLL exports any procedures or
  functions that pass strings as parameters or function results. This
  applies to all strings passed to and from your DLL--even those that
  are nested in records and classes. ShareMem is the interface unit to
  the BORLNDMM.DLL shared memory manager, which must be deployed along
```

```
  with your DLL. To avoid using BORLNDMM.DLL, pass
string information
  using PChar or ShortString parameters. }
  uses
  SysUtils,
  Classes;
  {$R *.res}
  begin
  end.
```

Эта заготовка DLL очень похожа на заготовку модуля. Основное отличие — ключевое слово Library вместо Unit в первой строке. Сохраните проект в каком-то отведенном вами для этого каталоге, назвав сохраняемый файл MyDLL. При этом в приведенном выше тексте автоматически имя библиотеки тоже заменится на MyDLL. При компиляции проекта с ключевым словом Library создается файл библиотеки с расширением dll.

Комментарий относится к особенностям передачи в функции DLL и получения от них строковых значений. Комментарий касается важного замечания в организации памяти DLL. Модуль ShareMem должен быть включен первым в предложения Uses в библиотеке и в проекте (выполните Project/View Source). Если DLL экспортирует какие-то процедуры или функции, которым передаются или от которых получаются строки в виде параметров или значений функции, это относится ко всем строкам, пересылаемым или получаемым из DLL, даже к тем, которые вложены в записи и классы. ShareMem является интерфейсным модулем для диспетчера разделяемой памяти BORLNDMM.DLL. Чтобы избежать использования BORLNDMM.DLL, передавайте строковую информацию посредством параметров типа PChar или ShortString.

Текст комментария можно, конечно, из кода удалить, оператор Uses подключает к библиотеке модули SysUtils и Classes. Во многих случаях эти модули не требуются, так что их подключение тоже можно убрать из кода. Впрочем, подключение модуля SysUtils может оказаться очень полезным, если во время выполнения функций и процедур библиотеки возможна генерация исключения. При отсутствии этого модуля неперехваченное в библиотеке исключение вызовет аварийное завершение приложения, причем даже не будут вызываться завершающие функции вызвавшего приложения, просто задача будет выгружена из памяти. Если модуль SysUtils подключен к библиотеке, то неперехваченное в ней исключение будет передано в вызвавшую программу и

может быть там обработано обычным оператором try ... except. Впрочем, учитывая то, что DLL может вызываться из программ, созданных на разных языках, лучше все-таки обрабатывать в ней все исключения и передавать в вызвавшую программу какой-то специальный флаг, свидетельствующий о возникшей ошибке. Директива

```
{$R *.res}
```

приведенного кода подключает к библиотеке файл ресурсов с именем, совпадающим с именем библиотеки. Эта директива нужна, если нужно что-то хранить в файле ресурсов. Хранение данных в ресурсах будет рассмотрено позднее, а если вы не намерены использовать ресурсы, то эту директиву также можно удалить. Оператор begin начинает раздел, операторы которого выполняются в момент загрузки DLL. Они могут осуществлять какие-то необходимые настройки. Если подобные настройки не предполагаются, то оператор begin также можно удалить из кода. Таким образом, в простейших случаях в коде можно оставить только заголовок library и заключительный оператор end.

Разместим в DLL функцию Code, в которую передается строка и ключ и которая возвращает строку, зашифрованную или расшифрованную этим ключом. Позднее поместим в эту DLL еще форму диалога, запрашивающего у пользователя его имя. Ниже приведены фрагменты кода этой DLL:

```
Library MyDLL;
Function Code (S: PChar; Key: integer): PChar;
stdcall;
Var i: integer;
Ss: string;
Begin
ss:=S;
for i:=1 to length(S) do
ss[i]:=char(ord(ss[i]) xor key);
code:=PChar(ss);
end;
exports
Code;
end.
```

Поясним функцию Code. Она принимает строку S и целое значение ключа Key. Затем в цикле каждый символ исходной строки заменяется символом, получаемым операцией исключающего ИЛИ xor над индексом данного символа и ключом Key. В резуль-

тате строка оказывается зашифрованной. Если повторно вызвать ту же функцию для зашифрованной строки с тем же ключом, то выполнится дешифровка и функция вернет первоначальную строку. По такому принципу обычно проходит любая шифровка текстов, а дальше уже можно применять различные правила шифрования. Можно для каждого очередного символа менять ключ по какому-то правилу. Например, если заменить оператор шифровки оператором:

```
ss[i]:=char(Ord(ss[i]) xor (key + I mod 2));
```

то ключ будет циклически меняться для каждой двойки символов.

Функцию Code было бы проще реализовать, если бы тип параметра *S* и возвращаемого значения был определен как String. В этом случае не нужно вводить локальную переменную типа String и преобразовывать строки. Но тут полезно вспомнить комментарий, который Delphi поместила в код библиотеки при ее создании и который был приведен ранее. Если требуется передать в DLL или получать из нее строки, то надо первым модулем в операторе uses указать ShareMem. Это интерфейсный модуль библиотеки BORLNDMM.DLL, обеспечивающей такую организацию памяти, при которой возможен обмен с DLL длинными строками. Более того, надо чтобы и в головном файле приложения, использующего DLL, модуль ShareMem тоже был указан первым в предложении uses. Так что лучше избежать предъявления таких требований к пользователям DLL и применять передачу строк через параметры типа PChar.

Спецификатор stdcall, указанный после заголовков экспортируемых из DLL процедур и функций, задает определенные соглашения при передаче параметров; в частности, передачу параметров в последовательности справа налево. Подобная передача принята в API Windows и в ряде языков программирования, используемых для создания приложений, которые будут вызывать DLL. Так что целесообразно всегда использовать спецификатор stdcall, поскольку в противном случае будет принят по умолчанию спецификатор register. Это обеспечит наиболее быстрый обмен параметрами, но такую DLL можно будет использовать только в приложениях, написанных на Pascal.

После текстов всех функций и процедур в коде DLL расположено предложение exports. В нем перечисляются процедуры и функции, экспортируемые DLL, т.е. те, которые сможет вызывать внешнее приложение. Помимо экспортируемых функций в DLL могут быть описаны какие-то вспомогательные функции — утили-

ты, предназначенные только для внутреннего использования внутри DLL. Такие утилиты, конечно, в предложение exports не включаются.

Если в exports указано просто имя процедуры или функции, то именно по этому имени внешнее приложение сможет ее вызвать. Но с помощью ключевого слова name можно для внешнего употребления указать другое имя. Например, если заменить приведенное выше предложение exports следующим:

```
exports
DoMy name 'My',
Code;
```

то функцию DoMy внешнее приложение должно будет вызывать, используя имя 'My'.

Внешнее приложение может вызывать функции и процедуры DLL не по именам, а по индексам. Индексы присваиваются автоматически, начиная с последнего имени, перечисленного в exports. Индексы можно задавать принудительно с помощью ключевого слова index. Например:

```
exports
DoMy index 1,
Code index 2;
```

Если предполагается возможность вызова по индексам, то лучше задавать индексы подобным образом, чтобы они не изменялись при добавлении в DLL новых функций и процедур. Вызовы по индексу могут потребоваться, если в имени функции использованы какие-то символы, непонятные для других языков и систем. Правда, задание индексов ключевым словом index допустимо только для Windows, на других платформах индексы формируются автоматически.

Предложение exports может включаться в текст DLL несколько раз и размещаться в любом месте кода, но лишь после объявления входящих в него функций и процедур. Если в DLL имеются перегруженные функции, то они должны включаться в предложение exports вместе со списком параметров и им должны даваться разные экспортируемые имена. Пусть, например, в библиотеке описаны две функции:

```
  function Average (X, Y: integer): real; overload;
stdcall;
  begin
  Average := (X + Y) / 2
```

```
end;
function Average (X, Y: real): real; overload; stdcall;
begin
Average := (X + Y) / 2
end;
```

Тогда их объявление в предложение *exports* должно иметь вид

```
exports
Average (X, Y: integer) name 'AverageInt',
Average (X, Y: real) name 'AverageReal';
```

Разные имена не помешают использованию этих функций как перегруженных в вызывающем их приложении. Теперь введем в DLL форму ввода пароля. Нецелесообразно вводить такую форму в каждое приложение, более экономно поместить ее в DLL и вызывать из всех приложений, в которых она потребуется. К тому же, если с течением времени нужно заменить эту форму, достаточно будет изменить только DLL, а все приложения могут оставаться неизменными.

Введем в DLL простую форму диалога, запрашивающего имя пользователя. Выполните команду File/New/Form, которая включит в DLL новую форму. Перенесите на форму метку, окно редактирования Edit1 и кнопку Button, в свойстве ModalResult которой задайте mrOk (рис. 10.1).

Сохраните модуль формы, задав ему имя UMyDialog. В модуле формы никаких кодов в этом случае можно не писать, а код DLL надо изменить следующим образом:

```
Library MyDLL;
uses
Forms {для переменной Application},
UMyDialog in 'UMyDialog.pas' {form1};
…
function MyDialog (User: PChar) : Pchar; stdcall;
Var form: TForm1;
```

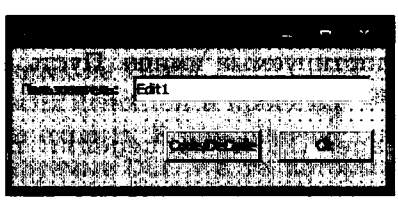


Рис. 10.1. Форма диалога

```
begin
Form := TForm1.Create (Application);
Form.Edit1.Text := User;
Form.ShowModal;
Result := PChar (Form.Edit1.Text);
Form.Free;
end;
exports
Code,
MyDialog;
end.
```

В приведенном коде введена функция MyDialog, вызывающая диалог и возвращающая строку, которую пользователь указал в окне Edit1. В качестве параметра в функцию передается строка User — начальное значение имени пользователя. В функцию введена локальная переменная Form. Первый выполняемый оператор создает экземпляр формы диалога, в окно редактирования этой формы заносится строка User. Затем методом ShowModal форма показывается пользователю как модальная. После завершения работы пользователя с этой формой текст окна редактирования заносится в значение, возвращаемое функцией, и форма удаляется из памяти методом Free.

Таким образом, вызов диалога не отличается от вызова любой другой функции, только в операторе uses добавляется ссылка на модуль формы и на модуль Forms, без которого компилятор не поймет переменной Application.

Задание

Создать DLL ввода логина и пароля для идентификации пользователя при загрузке приложения.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Опишите преимущества использования DLL.
2. Для чего используется спецификатор stdcall?
3. Для чего в операторе uses указывается ShareMem?
4. Для чего служит подключение модуля SysUtils?

Глава 11

СОЗДАНИЕ ДОКУМЕНТАЦИИ ДЛЯ ПОЛЬЗОВАТЕЛЯ. РАЗРАБОТКА СПРАВОЧНОЙ СИСТЕМЫ ПРОГРАММНОГО ПРОДУКТА

11.1. СОЗДАНИЕ ПРОГРАММНОГО ДОКУМЕНТА «РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ»

Составление документации для пользователей имеет свои особенности, связанные с тем, что пользователь, как правило, не является профессионалом в области разработки программного обеспечения. При разработке «Руководства пользователя» за основу можно взять эксплуатационные документы Единой системы программной документации «Руководство программиста» ГОСТ 19.504—79 и «Руководство оператора» ГОСТ 19.505—79. «Руководство пользователя», предположительно, содержит следующие разделы:

«Назначение программы»;

«Условия выполнения программы»;

«Выполнение программы»;

«Сообщения пользователю».

В зависимости от особенностей программы можно объединять отдельные разделы или вводить новые.

В разделе «Назначение программы» должны быть указаны сведения о назначении программы, краткое описание ее функций, реализованных методов и возможных областей применения.

«Условия выполнения программы» должны содержать условия, необходимые для выполнения программы (минимальный и (или) максимальный состав аппаратных и программных средств и т.п.).

В разделе «Выполнение программы» следует указать последовательность действий пользователя, обеспечивающих загрузку, выполнение и завершение программы, привести описание функций и пользовательского интерфейса. В качестве примера приведем небольшой фрагмент «Руководства пользователя».

В разделе «Сообщения пользователю» должны быть приведены сообщения, выдаваемые в ходе выполнения программы, описание их содержания и соответствующие действия пользователя (в случае сбоя, повторный запуск программы и т.п.).

Содержание разделов можно иллюстрировать поясняющими примерами, таблицами, схемами, графиками. В приложения к руководству пользователя включаются различные материалы, которые нецелесообразно помещать в разделы руководства.

Рекомендации по написанию пользовательской документации:

- учитывайте интересы пользователей, руководство должно содержать все инструкции, необходимые пользователю;
- излагайте ясно, используйте короткие предложения;
- избегайте технического жаргона и узко специальной терминологии, если все же необходимо использовать некоторые термины, то их следует пояснить;
- будьте точны и рациональны, длинных и запутанных руководств обычно никто не читает, например, лучше привести рисунок формы, чем долго ее описывать.

На основе материала, разработанного для руководства пользователя, формируется справочная система программного продукта или попросту файл справки. Он предназначен для предоставления пользователю программы полной и исчерпывающей информации о том, как работать с программой и для чего данный программный продукт нужен.

Справочная система должна удовлетворять следующим требованиям:

- давать полное описание по вопросам использования программы;
- иметь графические материалы;
- быть доступной для вызова из любой формы программы;
- иметь контекстные описания и удобную систему поиска информации;
- иметь минимально возможный размер.

11.2. ПРИМЕР РАЗРАБОТКИ ФРАГМЕНТА ДОКУМЕНТА «РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ» ДЛЯ АИС «СКЛАД ОПТОВОЙ ТОРГОВЛИ»

В качестве примера рассмотрим порядок работы с документом «Заказ товара» из раздела «Выполнение программы».

Документ «Заказ товаров» предназначен для оформления предварительной договоренности о намерении покупателя приобрести товарно-материальные ценности. Заказ не является обязательным для исполнения документом, основное его назначение — формирование печатной формы и выписка на ее основании расходной накладной. Экранная форма документа состоит из шапки и табличной части (см. рис 5.1). В шапке указываются:

«Заказчик» — элемент справочника «Клиенты»;

«Договор» — договор, по которому оформляется заказ;

«Вид оплаты» — наличный или безналичный расчет.

Заполнение табличной части документа выполняется обычным вводом новой строки документа. При этом открывается окно справочника «Товары», в нем можно выбирать и вносить в документ произвольное число видов товаров (двойным щелчком левой клавиши мыши по выбранному товару или нажатием клавиши Enter). Все выбранные товары записываются в табличную часть документа. Если в справочнике «Товары» у заказанного товара заполнен реквизит «Цена», то значение реквизита переносится в табличную часть. Сумма подсчитывается автоматически. Форма заполнения документа имеет дополнительный элемент управления — кнопку «Печать». При нажатии этой кнопки формируется печатная форма заказа, которую можно вывести на принтер. Документ «Заказ» не формирует бухгалтерских проводок. После заполнения экранной формы нужно щелкнуть по кнопке ОК.

11.3. РАЗРАБОТКА СПРАВОЧНОЙ СИСТЕМЫ

Справочная система представляет собой набор файлов определенной структуры, используя которые, программа Winhelp, являющаяся составной частью Windows, выводит справочную информацию, находящуюся в hlp-файле. Процесс создания справочной системы (hlp-файла) можно представить как последовательность следующих двух шагов:

1) подготовка справочной информации (создание файла документа справочной информации);

2) преобразование файла справочной информации в файл справочной системы.

Файлы документа справочной системы представляют собой rtf-файл определенной структуры. Будем называть эти файлы тематическими (topic files). Для получения из тематических файлов го-

товых файлов справки (hlp) их нужно обработать (скомпилировать) программой Help Workshop (hwc.exe). Кроме собственно текста и рисунков тематические файлы могут содержать также специальную разметку, которая несет в себе информацию, нужную для создания переходов по ссылкам, связи оглавления справки с ее темами для реализации различных возможностей Winhelp.

Обычно справка содержит несколько тем и оглавление, из которого можно перейти к этим темам. Самый простой вариант: тема одна и оглавления нет. В таком случае просто пишем то, что нужно, и сохраняем это в файле с расширением .rtf. Для создания нескольких тем процесс немного усложняется.

Каждая тема должна заканчиваться жестким переходом на новую станицу. Для этого после окончания темы нужно в меню (MS Word) «Вставка» выбрать «Разрыв/Начать новую страницу».

Для того чтобы тема была доступна из оглавления к справке, следует задать ей идентификатор. Для этого нужно в то место текста, куда будет впоследствии происходить переход из оглавления (начало темы или другое место), вставить специальную разметку, а именно концевую сноску. Символом сноски выбирается знак «#» (рис. 11.1). Идентификатором темы служит текст сноски. Например, создадим тему «Поддержка», отделим ее от других тем разрывами страниц и зададим ей идентификатор support. Для этого поместим каретку ввода около заголовка темы и выберем в меню «Вставка/Сноска...». В диалоговом окне выбираем вид снос-

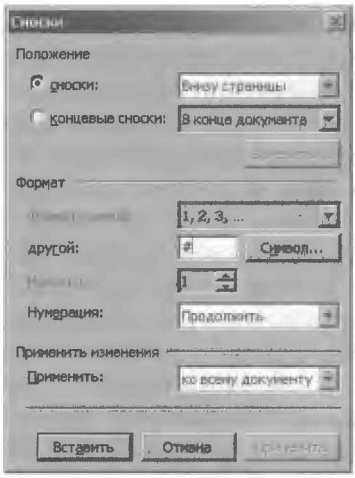


Рис. 11.1. Добавление концевой сноски

ки — «концевая», нумерация — «другая», в окошке для ввода символа пишем «#» (без кавычек). Щелкаем по кнопке OK, ссылка добавлена и каретка автоматически переведена к тексту ссылки. Пишем support. Готово. Повторяем то же самое для всех тем справки. Сохраняем файл.

Запускаем программу Help Workshop (файл Hcw.exe). Создаем новый проект через меню File/New/Help Project. Справа на панели есть ряд кнопок, из которых выбираем Files... . В диалоговом окне добавляем наш тематический файл и закрываем это окно. Сохраним проект — это будет файл с расширением hpj (Help Project). После первого запуска Help Workshop связывает себя с файлами hpj, а также с файлами оглавления справки (cnt), так что их потом можно открывать двойным щелчком мыши. Для создания help-файла можно просто щелкнуть по кнопке Save and Compile. Откроется новое окно с сообщением о результате компиляции. Предположим, что все в порядке, закроем это окно. Теперь в директории, где находился наш проект (.hpj), должен появиться файл справки. Однако при двойном щелчке мышью на нем мы сможем просмотреть только первую тему; чтобы просматривать все темы и перемещаться между ними, нужно добавить файл оглавления.

Файл оглавления справки имеет простой текстовый формат, но создавать его удобно тоже в Help Workshop. Для этого выбираем в меню Help Workshop File/New/Help Contents. В верхней части окна нужно вписать имя главного файла справки (файлов, вообще говоря, может быть несколько) и заголовок (title) для оглавления справки. То же можно сделать в диалоговом окне, которое открывается при нажатии кнопки Edit... .

Теперь создаем собственно оглавление. Оно состоит из элементов следующих двух типов: разделы справки, которые включают в себя несколько тем и представлены в оглавлении справки значком книжки, и сами темы — текст и картинки, представленные в оглавлении справки значком листа со знаком вопроса на нем (можно посмотреть это в оглавлении любой справки). Также в оглавление можно вставить макросы и включить файлы (*include*). Справа на панели есть набор кнопок для добавления и манипуляции элементами оглавления (Add Below — Добавить ниже, Add Above — Добавить выше, Move Right — Сдвинуть вправо, Move Left — Сдвинуть влево, Edit, Delete). При помощи них создаем нужную структуру оглавления. При добавлении раздела в диалоговом окне нужно указать только его название; при добавлении темы указывается название, идентификатор (тот, который мы

задали ей в rtf-файле, когда вставляли концевую сноску), имя help-файла и имя окна. Если тема находится в том же help-файле, который мы указали как главный, то имя help-файла указывать не нужно. Имя окна указывать тоже не обязательно — тема откроется в окне по умолчанию. Нужно сохранить файл оглавления (он будет иметь расширение .cnt) в той же директории, где находится help-файл, лучше с тем же именем, что и help-файл. Теперь нужно снова открыть файл проекта .hpj и, щелкнув по кнопке Options, в открывшемся диалоговом окне на закладке Files указать наш файл оглавления (Contents file). Закрываем диалоговое окно, снова щелкаем по кнопке Save and Compile. Теперь при двойном щелчке мышью по значку файла справки должно открыться ее оглавление, из которого можно получить доступ ко всем темам.

Как правило, разделы справки содержат ссылки на другие разделы. В окне справочной системы понятия (слова), выбор которых вызывает переход к другому разделу справки, выделяются отличным от основного текста справки цветом и подчеркиваются. Во время подготовки текста справочной информации слово-ссылку, выбор которого должен обеспечить переход к другому разделу справки, следует подчеркнуть двойной линией и сразу за этим словом без пробела поместить идентификатор раздела справки, к которому должен быть выполнен переход. Вставленный идентификатор необходимо оформить как скрытый текст.

Бывает, что в справку нужно поместить изображения. Это можно сделать, просто добавив их в документ rtf обычным для MS Word способом. Если одно изображение используется в нескольких местах, то можно воспользоваться специальной разметкой, предусмотренной для вставки изображений в справку, так что изображение будет храниться в одном экземпляре.

Чтобы справка была доступна в программе, написанной в среде Delphi, нужно указать программе на файл .hlp. Самый простой способ размещения файла справки — разместить его в той же папке, где находится исполняемый файл. Определить для приложения файл справки можно так: открыв проект (приложения) в Delphi и выбрав меню Project/Options/Application, вписать название файла справки в поле ввода Help file. При этом нужно указать название файла без пути. Когда WinHelp пытается найти справочный файл, одна из просматриваемых директорий та — где расположен исполняемый файл программы. Другой способ: в обработчик события OnCreate главной формы программы вставить строку:

```
Application.HelpFile:=ExtractFilePath(Application.ExeName)+
+«MyHelp.hlp»;
```

где MyHelp.hlp — название файла справки.

Чтобы из меню в программе вызвать оглавление справки, нужно воспользоваться функцией

```
Application.HelpCommand(HELP_FINDER, 0);
```

Чтобы перейти к одной из определенных нами тем справки, нужно вызвать функцию

```
Application.HelpJump('MyTopic'),
```

где *MyTopic* — идентификатор темы.

Один из способов вызова справки — нажатие клавиши F1. Можно организовать вызов контекстной справки при нажатии клавиши F1, когда активным является тот или иной элемент управления. Для этого соответствующей теме справки нужно присвоить номер, а затем этот номер присвоить свойству HelpContext элемента управления. Чтобы задать номера для тем справки, нужно открыть проект справки в HelpWorkshop и щелкнуть по кнопке Map в правой части окна. Щелкаем в диалоговом окне по кнопке Add, вводим идентификатор темы и произвольный номер, повторяем это для всех нужных тем (каждой — свой номер), закрываем окно и нажимаем в очередной раз Save and Compile. Затем в Delphi в окне инспектора объектов присваиваем нужные номера нужным элементам.

Задание

Для своего варианта задания разработать программный документ «Руководство пользователя». На основе созданного документа построить справочную систему.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чем особенность программного документа «Руководство пользователя»?
2. Назовите основные разделы «Руководства пользователя».
3. Назовите основные этапы разработки справочной системы.
4. Как составить оглавление к файлу справки?
5. Как в одном разделе справки сделать ссылку на другой раздел?

Глава 12

СОЗДАНИЕ ИНСТАЛЛЯЦИИ ПРОГРАММНОГО ПРОДУКТА

12.1. НЕОБХОДИМОСТЬ СОЗДАНИЯ ИНСТАЛЛЯЦИИ

Процесс установки программы для многих пользователей является довольно трудной задачей. Он, как правило, предполагает не только создание каталога и перенос в него выполняемых файлов и файлов данных с промежуточного носителя, но и настройку системы. Создание инсталляции программного продукта позволяет автоматизировать процесс установки программного продукта на компьютеры пользователей, предоставляя им при этом возможность выбора различных сценариев установки и обеспечивая корректность его дальнейшей работы. Кроме того, у разработчиков программного продукта появляется хорошая возможность запретить несанкционированные установки. Процесс инсталляции программного продукта обязательно должен быть описан в «Руководстве пользователя». Задачи, решаемые во время инсталляции, являются типовыми, поэтому существуют инструментальные средства, используя которые можно быстро создать инсталляционную программу, точнее, установочный диск, не написав ни одной строчки кода.

Для создания инсталляции программного продукта существует много инструментальных средств, один из них — программа InstallShield Express.

12.2. ПРОЦЕСС СОЗДАНИЯ ИНСТАЛЛЯЦИИ ПРОГРАММНОГО ПРОДУКТА

Перед тем как непосредственно приступить к созданию установочной программы в InstallShield Express, нужно выполнить подготовительную работу:

Таблица 12.1. Список файлов, требующих установки

Файл	Назначение	Куда устанавливать

1) составить список файлов, которые должны быть установлены на компьютере пользователя;

2) используя редактор текста, подготовить RTF-файлы лицензионного соглашения (EULA — End User Licensia Agreement) и краткой справки (Readme-файл).

Список файлов программы, которые должны быть перенесены на компьютер пользователя, можно свести в табл. 12.1.

После того как будет составлен список файлов, нужно запустить InstallShield Express, из меню File выбрать команду New и в поле Project Name and Location ввести имя файла проекта.

После щелчка по кнопке OK открывается окно проекта создания инсталляционной программы. В левой части окна перечислены этапы процесса создания и команды, при помощи которых задаются параметры создаваемой инсталляционной программы. Команды настройки объединены в группы, название и последовательность которых отражает суть процесса создания инсталляционной программы. Заголовки групп пронумерованы, настройка программы установки выполняется путем последовательного выбора команд. В результате выбора команды в правой части главного окна появляется список параметров, выполненные команды помечаются галочками.

Таблица 12.2. Список параметров, требующих установки

Параметр	Что определяет	Значение
Product Name	Название устанавливаемой программы	Sklad 2008
Product Version	Версия устанавливаемой программы	1.01.0001
INSTALLDIR	Каталог компьютера пользователя, в который будет установлена программа	[ProgramFilesFolder] Sklad 2008

Команды группы Organize Your Setup позволяют задать структуру программы установки. Значения большинства параметров, за исключением тех, которые идентифицируют устанавливаемую программу и ее разработчика, можно оставить без изменения. Параметры, значения которых нужно изменить, приведены в табл. 12.2.

Следует обратить внимание на параметр INSTALLDIR. По умолчанию предполагается, что программа будет установлена в каталог, предназначенный для программ. Поскольку во время создания инсталляционной программы нельзя знать, как на компьютере пользователя называется каталог программ и на каком диске он находится, то вместо имени реального каталога используется его псевдоним — [ProgramFilesFolder]. В процессе установки программы на компьютер пользователя инсталляционная программа получит из реестра Windows имя каталога программ и заменит псевдоним на это имя.

Другие псевдонимы, используемые в программе InstallShield Express:

[WindowsVolume] — корневой каталог диска, на котором находится Windows;

[Windows Folder] — каталог Windows, например C:\Windows;

[SystemFolder] — системный каталог Windows, например C:\Windows\System32;

[ProgramFilesFolder] — каталог программ, например C:\Program Files;

[PersonalFolder] — папка «Мои документы» на рабочем столе (расположение папки зависит от версии операционной системы и способа входа в систему).

Очевидно, что возможности инсталлированной программы определяются составом установленных компонентов. Например, если установлены файлы справочной системы, то пользователю в процессе работы с программой доступна справочная информация. Команда Features (возможности) позволяет создать (определить) группы компонентов, которые определяют возможности программы и которые могут устанавливаться по отдельности. Разделение компонентов на группы позволяет организовать многовариантную, в том числе и определяемую пользователем, установку программы.

В простейшем случае группа Features состоит из одного элемента Always Install. Чтобы добавить элемент в группу Features, нужно щелкнуть правой клавишей мыши по слову Features, из появившегося контекстного меню выбрать команду New Feature Ins

и ввести имя новой группы, например Help Files and Samples. После этого в поле Description следует ввести краткую характеристику элемента, а в поле Comments — комментарий.

Команда Setup Types позволяет задать, будет ли пользователю во время установки программы предоставлена возможность выбрать (в диалоговом окне Setup Type) вариант установки. Установка может быть обычной (Typical), минимальной (Minimal) или выборочной (Custom). Если устанавливаемая программа сложная, состоит из нескольких независимых компонентов, то эта возможность обычно предоставляется. Если для программы предполагается только один вариант установки — Typical, то флажки Minimal и Custom нужно сбросить.

Команды группы Specify Application Data (рис. 12.1) позволяют определить компоненты программы, которые должны быть установлены на компьютер пользователя. Если в проекте определены несколько групп компонентов (см. команду Features), то нужно определить компоненты для каждой группы.

В результате выбора команды Files правая часть окна будет разделена на области (см. рис. 12.1). В области Source computer's files можно выбрать файлы, которые необходимо перенести на компьютер пользователя. В области Destination computer's folders надо выбрать папку, в которую эти файлы должны быть помещены. Для того чтобы указать, какие файлы нужно установить на компьютер пользователя, следует просто «перетащить» требуемые файлы из области Source computer's files в область Destination computer's files. Если в группе Features несколько элементов, то надо определить файлы для каждого элемента.

Команда Object/Merge Modules позволяет задать, какие объекты, например, динамические библиотеки или пакеты компонентов, должны быть помещены на компьютер пользователя и, следовательно, на установочную дискету. Объекты, которые нужно поместить на установочную дискету, выбираются в списке InstallShield Objects/Merge Modules.

Команды группы Configure the Target System позволяют задать, какие изменения нужно внести в систему пользователя, чтобы настроить систему на работу с устанавливаемой программой.

Команда Shortcuts/Folders позволяет указать, куда нужно поместить ярлык, обеспечивающий запуск устанавливаемой программы. В результате выбора этой команды в правой части окна открывается иерархический список, в котором перечислены меню и папки, куда можно поместить ярлык программы. В этом списке необходимо выбрать меню, в которое должен быть помещен яр-

Рис. 12.1. Выбор файлов, которые нужно перенести на компьютер пользователя

лык, щелкнуть правой клавишей мыши и в появившемся списке выбрать команду New Shortcut.

Затем в диалоговом окне Browse for Shortcut Target нужно выбрать файл программы, щелкнуть по кнопке Open и ввести имя ярлыка. После этого можно выполнить окончательную настройку ярлыка — например, в поле Arguments ввести параметры командной строки, а в поле Working Directory — рабочий каталог.

Для взаимодействия с пользователем программа установки использует стандартные диалоговые окна. Разрабатывая программу инсталляции, программист может задать, какие диалоги увидит пользователь в процессе инсталляции программы. Чтобы задать

диалоговые окна, которые будут появляться на экране монитора во время работы инсталляционной программы, надо в группе Customize the Setup Appearance выбрать команду Dialogs и в открывшемся списке Dialogs отметить диалоги, которые нужно включить в программу установки (рис. 12.2). Эти диалоги появятся в процессе установки программы на компьютер пользователя.

В таблице Properties (справа от списка диалогов) перечислены свойства выбранного диалога. Программист может изменить значение этих свойств и тем самым выполнить настройку диалога. Например, для диалога Readme нужно задать имя файла (свойство Readme File), в котором находится краткая справка об устанавливаемой программе.

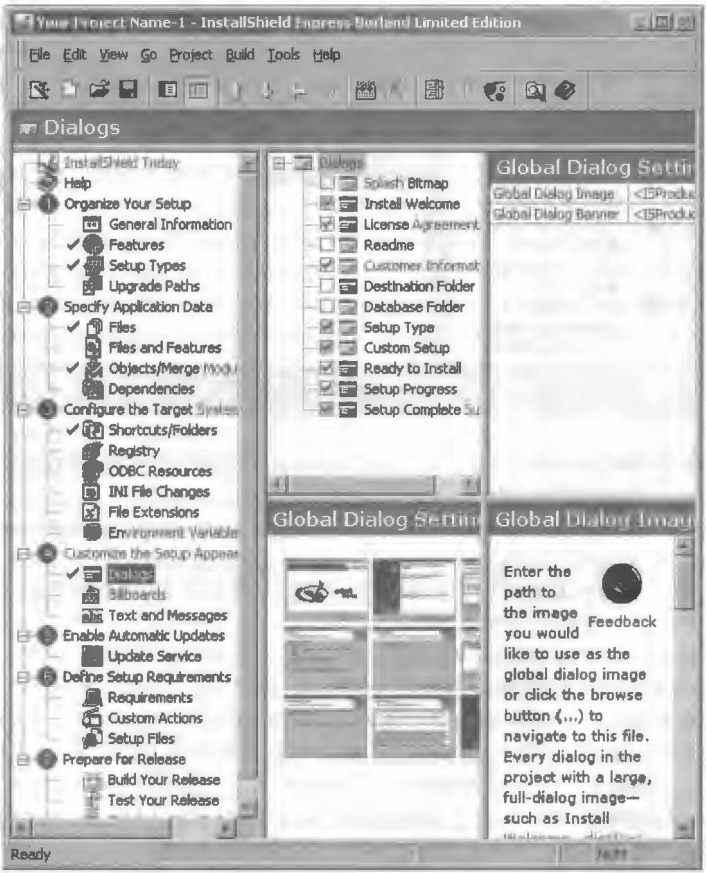


Рис. 12.2. Список Dialogs

Для большинства диалогов можно определить баннер (свойство Banner Bitmap) — иллюстрацию, которая отображается в верхней части окна диалога.

Ниже перечислены диалоговые окна, которые могут появиться во время работы инсталляционной программы:

Splash Bitmap — вывод иллюстрации, которая может служить в качестве информации об устанавливаемой программе;

Install Welcome — вывод информационного сообщения на фоне иллюстрации;

License Agreement — вывод находящегося в RFT-файле лицензионного сообщения. Позволяет прервать процесс установки программы в случае несогласия пользователя с предлагаемыми условиями;

Readme — вывод краткой информации об устанавливаемой программе;

Customer Information — запрашивает информацию о пользователе (имя, название организации) и, возможно, серийный номер устанавливаемой копии;

Destination Folder — предоставляет пользователю возможность изменить предопределенный каталог, в который устанавливается программа;

Database Folder — предоставляет пользователю возможность изменить предопределенный каталог, предназначенный для баз данных;

Setup Type — позволяет пользователю выбрать тип установки программы (Typical — обычная установка, Minimal — минимальная установка, Custom — выборочная установка);

Custom Setup — дает пользователю возможность выбрать устанавливаемые компоненты при выборочной (Custom) установке;

Setup Complete Success — информирует пользователя о завершении процесса установки. Позволяет задать программу, которая должна быть запущена после завершения установки (как правило, это сама установленная программа), а также дает возможность вывода содержимого файла Readme;

Setup Progress — показывает процент выполненной работы во время установки программы;

Ready to Install — вывод информации, введенной пользователем на предыдущих шагах, с целью ее проверки перед началом непосредственной установки программы.

Для того чтобы диалоговое окно появлялось во время работы инсталляционной программы, необходимо установить флажок, расположенный слева от названия диалогового окна. Для окон

License Agreement и Readme нужно задать имена RTF-файлов, в которых находится соответствующая информация.

В простейшем случае программа инсталляции может ограничиться выводом следующих диалогов:

Readme;

Destination Folder;

Ready to Install;

Setup Progress;

Setup Complete Success.

Если устанавливаемая программа предъявляет определенные требования к ресурсам системы, то, используя команды группы Define Setup Requirements and Actions, эти требования можно за-

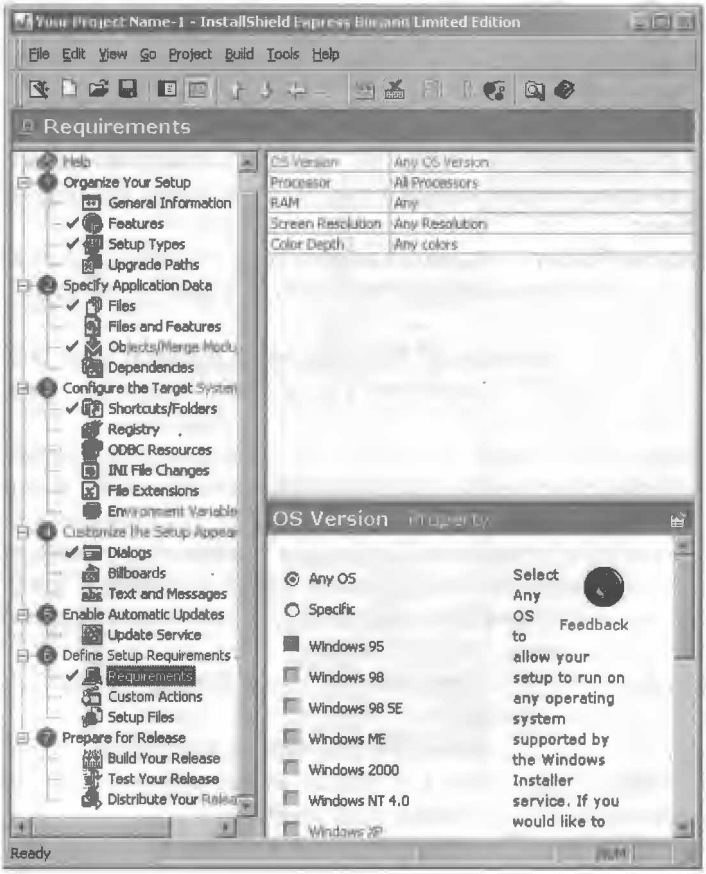


Рис. 12.3. Параметры, характеризующие систему

дать. В результате выбора команды Requirements на экране появляется таблица (рис. 12.3), в которую надо ввести значения параметров, характеризующих систему: версию операционной системы (OS Version), тип процессора (Processor), объем оперативной памяти (RAM), разрешение экрана (Screen Resolution) и цветовую палитру (Color Depth). Значения характеристик задаются путем выбора из раскрывающегося списка, значок которого появляется в результате щелчка в поле значения параметра.

Если программа не предъявляет особых требований к конфигурации системы, то команды группы Define Setup Requirements and Actions можно пропустить.

Команды группы Prepare for Release позволяют создать образ установочного диска (CD-ROM) и проверить, как работает программа установки. Для того чтобы активизировать процесс создания образа установочного диска (CD-ROM), нужно выбрать команду Build Your Release, щелкнуть правой кнопкой мыши по значку носителя, на который предполагается поместить программу установки, и из появившегося контекстного меню выбрать команду Build. В результате этих действий на диске компьютера в папке проекта будет создан образ установочного диска.

Задание

Создать инсталляцию программного продукта по своему варианту сквозной задачи.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какова цель создания инсталляции программного продукта?
2. Какие проблемы могут возникнуть в процессе инсталляции и как их можно решить?
3. Какое назначение имеет команда Setup Types в программе InstallShield Express?
4. Назовите диалоговые окна, которые могут появиться во время работы инсталляционной программы.

Глава 13

КУРСОВОЕ ПРОЕКТИРОВАНИЕ

13.1. СТРУКТУРА И СОДЕРЖАНИЕ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ К КУРСОВОМУ ПРОЕКТУ

Курсовое проектирование является заключительным этапом изучения студентами дисциплины «Технология разработки программного продукта». В процессе курсового проектирования у студентов формируются навыки ведения самостоятельной работы и разработки проектных решений по информационному, технологическому и программному обеспечению. Студенты учатся:

1) анализировать информационную среду предметной области и устанавливать структурное представление и взаимосвязи с другими компонентами информационного пространства;

2) анализировать информационные потоки, выстраивать логическую структуру проекта;

3) организовывать базы данных;

4) алгоритмизировать предметную область и организовывать программное обеспечение;

5) использовать современные алгоритмические языки программирования и СУБД.

В процессе выполнения курсового проекта студент должен разработать программную документацию, что, как правило, вызывает больше трудностей и проблем, нежели непосредственная работа с программой. Ниже приводится примерная структура курсового проекта и содержание его разделов, отражаемых в пояснительной записке.

Итак, программный документ «Пояснительная записка» в рамках курсового проекта может иметь следующую структуру.

Введение

1. Разработка системного проекта

1.1. Назначение разработки

1.2. Требования к функциональным характеристикам

1.3. Требования к надежности и безопасности

1.4. Требования к составу и параметрам технических средств

1.5. Требования к информационной и программной совместимости

2. Разработка технического проекта

2.1. Анализ требований и определение спецификаций программного обеспечения

2.2. Проектирование модели данных

2.3. Детальное проектирование программного обеспечения (конструирование прототипа).

3. Реализация

3.1. Обоснование выбора средств разработки

3.2. Описание основных программных модулей

4. Тестирование программного продукта.

Заключение

Приложения

Список литературы

Представим содержание разделов.

Введение

Указывается цель проекта, краткая характеристика области применения разрабатываемого программного продукта и описание объекта, в котором ее используют (описание предметной области).

1. Разработка системного проекта

Этот раздел фактически представляет собой техническое задание к курсовому проектированию.

1.1. Назначение разработки содержит определение функциональных и эксплуатационных задач, которые должна решить разрабатываемая система для достижения поставленной цели.

1.2. Требования к функциональным характеристикам включают в себя описание состава выполняемых функций, требования к входной и выходной информации, а также к сервисным функциям программы.

1.3. Требования к надежности и безопасности содержат требования к обеспечению надежного и устойчивого функционирования программного продукта, к контролю входной и выходной информации, ко времени восстановления после отказа и т.п.

1.4. Требования к составу и параметрам технических средств включают указания на необходимый состав технических средств

и их основных характеристик, а именно, минимальные системные требования, необходимые для работы программы.

1.5. Требования к информационной и программной совместимости содержат требования к информационным структурам, языкам программирования и программным средствам.

2. Разработка технического проекта

На этом этапе на основе системного проекта осуществляется собственно проектирование системы, включающее выбор технологии проектирования и построение моделей.

2.1. Анализ требований и определение спецификаций программного обеспечения

2.1. Выбор технологии проектирования

Здесь выбирается подход к проектированию программного обеспечения — структурный или объектно-ориентированный. Сравним эти подходы. Каждый из них имеет свои преимущества и недостатки. При структурном подходе процессы и данные существуют отдельно друг от друга (как в модели деятельности организации, так и в модели программной системы), причем проектирование ведется от процессов к данным. Это является главным недостатком структурного подхода. В объектно-ориентированном подходе основная категория объектной модели — класс. Он объединяет в себе на элементарном уровне как данные, так и операции, которые над ними выполняются (методы). Разделение процессов и данных преодолено. Данные по сравнению с процессами являются более стабильной и относительно редко изменяющейся частью системы, поэтому объектно-ориентированные системы более открыты и легче поддаются внесению изменений. Кроме того, объектно-ориентированная модель наиболее адекватно отражает реальный мир, представляющий собой совокупность взаимодействующих объектов, взаимодействие происходит посредством обмена сообщениями между объектами. Однако на практике диаграммы, отражающие специфику объектного подхода (диаграммы классов и т.п.), гораздо менее наглядны и плохо понимаемы непрофессионалами, поэтому понимание заказчиком разработчика обеспечивается на сегодняшний день, в основном, структурными методами.

2.1.2. Построение моделей

На этом этапе строятся модели проектируемой системы, которые детализируются и уточняются до необходимого уровня (в зависимости от выбранной технологии: диаграммы потоков данных, функциональные диаграммы, варианты использования, диаграм-

мы деятельности и т.д.). В данном подразделе должны быть представлены разработанные модели с текстовым описанием.

2.2. Проектирование модели данных

В случае применения структурного подхода после завершения функционального анализа системы определяется состав потоков данных и конструируется концептуальная схема данных в форме одной модели или нескольких локальных моделей. Наиболее распространенным средством моделирования данных считаются диаграммы «сущность—связь» (ERD).

2.3. Детальное проектирование программного обеспечения

Строится структурная схема программной системы, выполняется детальное описание функционирования системы через функциональные схемы, осуществляется проектирование экранных форм, отчетов, диалогов (создание прототипа программного обеспечения). С помощью диаграмм последовательностей экранных форм моделируется иерархия экранных форм. Совокупность таких диаграмм представляет собой абстрактную модель пользовательского интерфейса системы, отражающую последовательность появления экранных форм в приложении.

3. Реализация

На стадии реализации выполняется непосредственно разработка программного приложения. На основе полученных моделей, а также требований нефункционального характера (требований к надежности, производительности и т.п.) формируется программный код, выполняется модульное тестирование.

3.1. Обоснование выбора средств разработки

Здесь дается обоснование выбора языка программирования, приводятся основные факторы, влияющие на выбор среды разработки (сравнительная пригодность языка программирования для данной задачи, избранная методология и т.д.).

3.2. Описание основных программных модулей

Описание основных программных модулей системы выполняется в соответствии с ГОСТ 19.701—90 (ИСО 5807—85) ЕСПД. В него включаются исходные коды программных модулей, схемы алгоритмов программ, описание используемых методов, описание структуры программы.

4. Тестирование программного продукта

Одновременно с началом этапа планирования и создания спецификаций требования разрабатывается стратегия тестирования. После утверждения спецификаций требований разрабатывается и детализируется план тестирования, создаются наборы тестов для

проведения интеграционного и системного тестирования. Тестирование завершается созданием отчета о тестировании, в котором представляются все результаты его проведения. Отчет должен содержать следующие разделы:

1. Объект испытаний;
2. Цель испытаний;
3. Состав предъявляемой документации;
4. Технические требования;
5. Порядок проведения испытаний;
6. Методы испытаний.

В первых трех разделах указывают: наименование, область применения, обозначение испытуемой программной системы; цель проведения испытаний; перечень документации, предъявляемой перед проведением испытаний.

Раздел «Технические требования» может состоять из двух подразделов: 1) требования к программной документации; 2) требования к техническим характеристикам.

В первом подразделе должны быть указаны требования к комплектности, содержанию и качеству предъявляемой документации; второй подраздел содержит описание требований к характеристикам программы применительно к условиям эксплуатации и требований к информационной и программной совместимости.

«Порядок проведения испытаний» предполагает указания: на последовательность испытаний, четкий порядок проведения каждого испытательного эксперимента, состав и структуру технических средств, с помощью которых будут проводиться испытания, перечень дополнительных программных и технических средств, необходимых для проведения испытаний.

В разделе «Методы испытаний» приводятся описания используемых методов проведения испытаний. Методы следует приводить в последовательности, соответствующей последовательности перечисления технических характеристик в разделе «Технические требования». При этом должны быть приведены описания проверок с указанием результатов проведения испытаний, к которым могут относиться: перечень тестовых примеров, контрольных распечаток самих примеров и их результатов, таблиц, графиков и т. п. Сами тестовые примеры (распечатки, таблицы, графики и т. п.) даются в приложении.

Заключение

Содержит анализ выполненной работы, выводы о значимости проекта, рекомендации по использованию проекта, рекоменда-

ции, касающиеся возможности дальнейшей доработки или модернизации проекта и т.д.

Практическим результатом работы над курсовым проектом является работоспособная программа и пакет документации, включающий в себя программные документы «Пояснительная записка» и «Руководство пользователя». Кроме того, на магнитном носителе должен быть представлен текст программы (исходный код) с необходимыми комментариями.

Для защиты курсового проекта создается презентация из 10—12 слайдов, в которой отражаются требования к программной системе, основные этапы ее разработки, диаграммы, модели данных, выводы по работе.

13.2. ОФОРМЛЕНИЕ КУРСОВЫХ ПРОЕКТОВ

Программное обеспечение автоматизированной информационной системы «Абитуриент». Ниже приведен фрагмент оформления программного продукта. Пояснительная записка курсового проекта по теме «Программное обеспечение автоматизированной информационной системы "Абитуриент"».

Введение

Автоматизированная информационная система «Абитуриент» представляет собой часть автоматизированной системы управления средним профессиональным учебным заведением. Цель разработки — автоматизация работы приемной комиссии по организации приема абитуриентов, подготовка информации для проведения вступительных испытаний и формирования необходимой отчетности по итогам приема.

1. Разработка системного проекта

1.1. Назначение разработки

Подсистема приема абитуриентов используется в приемной комиссии и предназначена для решения следующих задач:

- ведение базы данных абитуриентов;
- контроль и проведение вступительных испытаний;
- зачисление студентов по результатам вступительных испытаний;
- формирование необходимой отчетности по итогам приема заявлений и проведения вступительных экзаменов (составление различных списков, формирование приказов, статистических отчетов и т.д.).

1.2. Требования к функциональным характеристикам

1.2.1. Состав выполняемых функций

Разрабатываемая автоматизированная информационная система «Абитуриент» должна обеспечивать:

- сбор сведений о поступающих в учебное заведение;
- быстрый поиск необходимой информации об абитуриентах по различным критериям, задаваемым пользователями, и вывод ее на печать;
- печатание экзаменационных листов;
- формирование статистических отчетов по итогам подачи заявлений и проведения вступительных экзаменов.

Исходная информация в систему поступает из заявлений абитуриентов, документов об образовании, справок и т.д. Входными данными являются следующие:

- дата и номер заявления;
- фамилия, имя, отчество;
- пол;
- дата рождения;
- место рождения;
- гражданство;
- ИНН;
- страховой номер;
- домашний адрес;
- паспортные данные;
- сведения о законченном образовании;
- документы, дающие право на льготы;
- дополнительные сведения (инвалид, сирота, участник военных действий и т.д.);
- сведения о родителях (ФИО, место работы);
- средний балл аттестата.

Абитуриенты выбирают желаемую специальность, форму обучения. Все сведения об абитуриентах вводятся в базу данных.

Система должна функционировать в многопользовательском режиме и давать возможность:

- просматривать записи базы данных, в том числе и с помощью различных фильтров;
- добавлять новые записи и изменять существующие;
- удалять записи.

Результаты проведения вступительных экзаменов тоже сохраняются в базе данных.

Источниками выходных данных являются:

- отчеты по поданным заявлениям абитуриентов;
- отчеты по результатам сдачи экзаменов;
- списки поступивших в учебное заведение;
- статистические отчеты по приему и зачислению;
- регламентированные отчеты в вышестоящие организации по приему студентов.

Примечание. Формы входных документов и образцы выходных отчетов обязательно должны быть представлены в приложении к курсовому проекту.

1.3. Требования к надежности и безопасности

Система должна функционировать в многопользовательском режиме, поэтому каждый пользователь должен иметь свой пароль доступа в систему. Кроме того, в системе должна быть предусмотрена возможность резервного копирования и восстановления данных, а также программная система должна иметь возможность самовосстановления после сбоев в операционной системе или отключения электропитания.

1.4. Требования к составу и параметрам технических средств Системные требования:

- тактовая частота процессора > 2 000 Гц;
- объем оперативной памяти 512 Мб;
- объем свободного дискового пространства 50 Мб;
- разрешение монитора 1 024×768.

1.5. Требования к информационной и программной совместимости

На компьютерах должна быть установлена операционная система Windows 2000/XP. Все формируемые отчеты должны иметь возможность экспортирования в редактор электронных таблиц MS Office Excel 2003/2007. На компьютере не должно быть установлено пакетов, программно и аппаратно конфликтующих с системой защиты «1С:Предприятие».

2. Разработка технического проекта

2.1. Анализ требований и определение спецификаций программного обеспечения (построение функциональной модели в нотации IDEF0)

Разработку программного обеспечения начнем с анализа требований к будущему программному продукту. Построим общую модель предметной области как некоторой части реального мира, с которой будет тем или иным способом взаимодействовать разрабатываемое программное обеспечение, после чего конкретизируем его основные функции.

Спецификации должны однозначно восприниматься как заказчиком, так и разработчиком. Обеспечить это требование можно, только разработав некоторую *формальную модель* этого программного обеспечения. На этапе анализа и определения спецификаций можно применить структурный подход. Построим функциональную диаграмму, для того чтобы выявить основные функции и составные части проектируемой программной системы и, по возможности, обнаружить и устранить существенные ошибки. Одной из наиболее важных особенностей методологии функционального моделирования является постепенное введение все больших уровней детализации по мере создания диаграмм, отображающих модель.

Диаграмма, показанная на рис. 13.1, представляет собой диаграмму верхнего уровня. На ней хорошо видно, что служит исходными данными для программы и получения каких результатов мы ожидаем. На вход системы подаются сведения об абитуриентах, на выходе — отчеты по работе Приемной комиссии. Управляющей информацией выступают Правила приема, Свидетельство об аккредитации учебного заведения, лицензии и т.д.

Диаграмма на рис. 13.2 уточняет функции программы. На диаграмме четыре блока: 1) прием и оформление документов; 2) проведение вступительных экзаменов; 3) зачисление по результатам экзаменов; 4) формирование отчетов.

Правила приема в среднее профессиональное учебное заведение регламентируют процедуру приема заявлений и зачисления студентов. На основании свидетельства об аккредитации формируется перечень специальностей.

Декомпозиция блока *A*1 (функция приема и оформления документов) представлена на рис. 13.3.

Процедура приема и оформления документов при поступлении в учебное заведение включает в себя прежде всего выбор абитуриентом специальности и формы обучения из имеющихся в учебном заведении, затем — заполнение заявления установленного образца. Все сведения о поступающих в учебное заведение должны сохраняться в базе данных для формирования списков абитуриентов, печатания экзаменационных листов, внесения резуль-

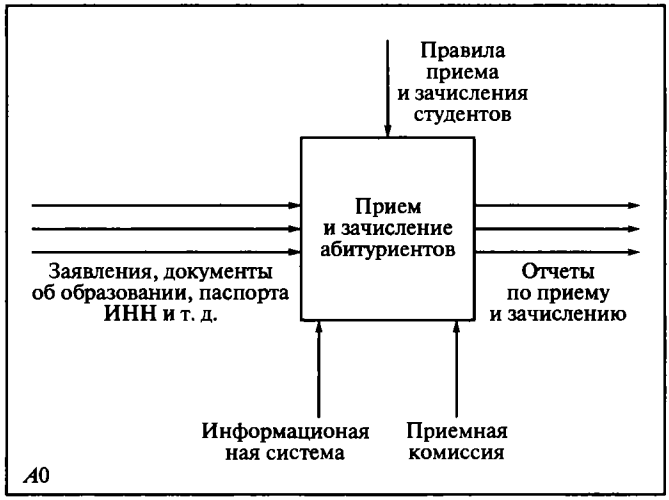


Рис. 13.1. Функциональная диаграмма начального уровня

татов проведения вступительных экзаменов и последующего анализа результатов приема. Заявление заполняется на основании личных документов абитуриента — паспорта, страхового свидетельства, документа об образовании, медицинской справки и т.д. После оформления заявления абитуриенту выдается экзаменационный лист. Список дисциплин для проведения вступительных испытаний определяется выбранной специальностью и правилами приема.

2.2. Проектирование модели данных

Теперь разработаем диаграмму «сущность—связь» (ER-модель данных), которая обеспечивает способ определения данных и отношений между ними. Модель данных включает сущности и связи между ними. Диаграммы «сущность—связь» в отличие от функциональных диаграмм определяют спецификации структур данных программного обеспечения. В модели данных разрабатываемой автоматизированной информационной системы главной сущностью выступает, конечно, Абитуриент.

Проанализируем атрибуты этой сущности: пол, дата рождения, место рождения, гражданство, ИНН, страховой номер, домашний адрес, паспортные данные, сведения о законченном образовательном учреждении, документы, дающие право на льготы, дополнительные сведения (инвалид, сирота, участник военных действий и т.д.), сведения о родителях (ФИО, место работы), средний балл

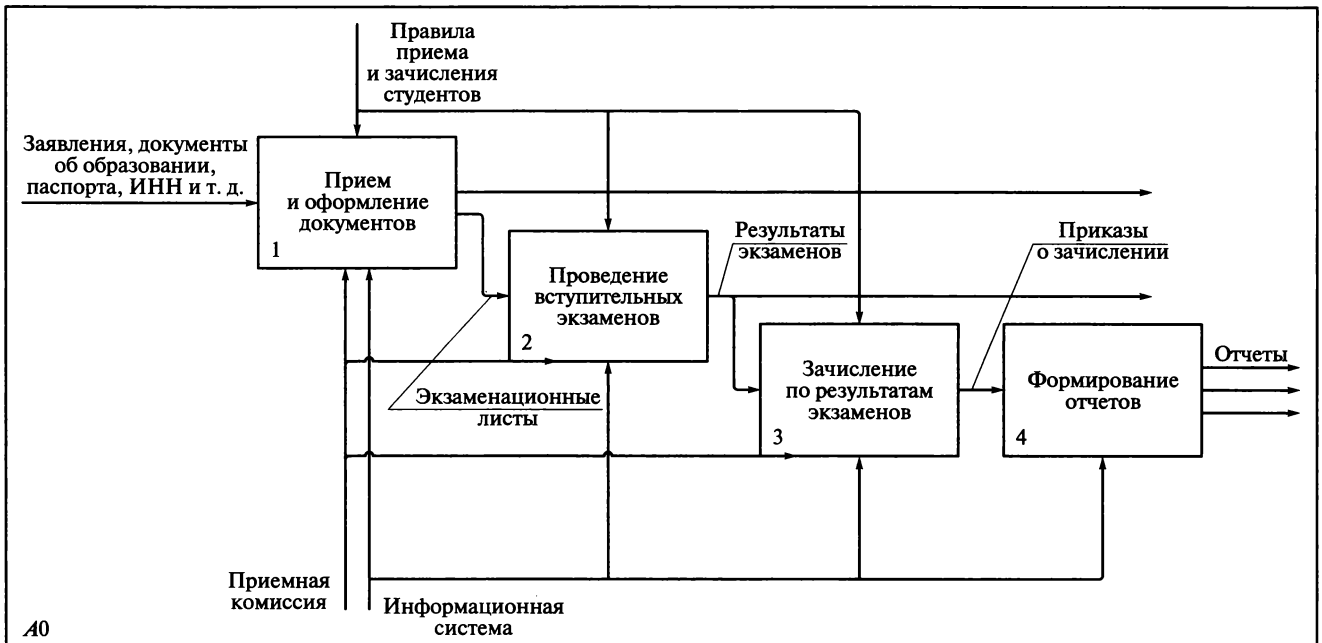


Рис. 13.2. Функциональная диаграмма нулевого уровня (более подробный вариант)

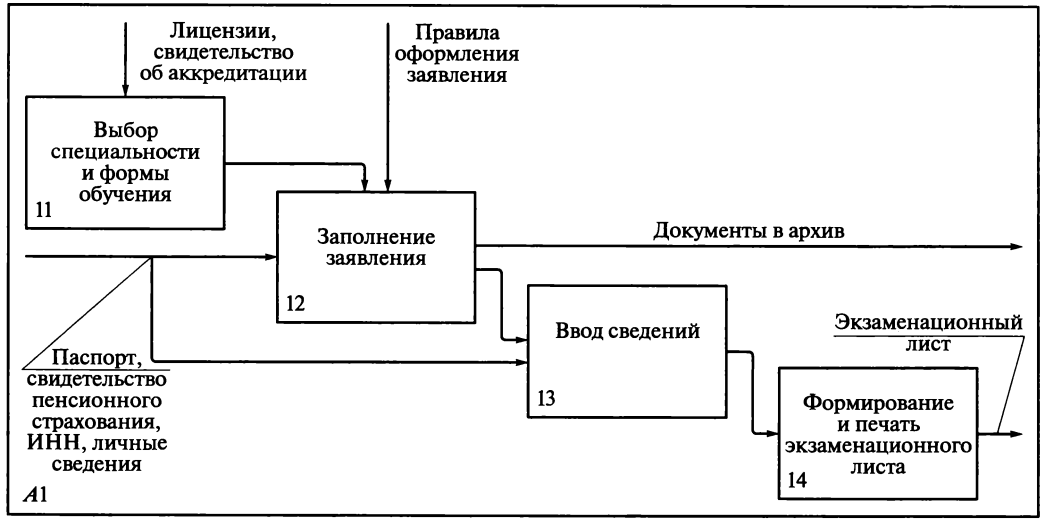


Рис. 13.3. Прием и оформление документов

аттестата и т.д. Атрибуты «Школа», «Гражданство», «Форма обучения», «Специальность», «Улица», «Город», «Область» являются наименованиями отдельных сущностей; при этом значение атрибута «Населенный пункт» полностью определяется значением сущности «Улица», а значение атрибута «Область» определяется значением сущности «Населенный пункт».

Основные отношения между указанными сущностями показаны на рис. 13.4. На следующем шаге определяем атрибуты каждой сущности и уточняем их типы (атрибуты, используемые для дополнительной идентификации сущности другой сущностью, не указываются, так как они описываются в соответствующей сущности). Теперь можно внести все это в диаграмму (рис. 13.5).

2.3. Детальное проектирование программного обеспечения (конструирование прототипа)

Детальное проектирование программного обеспечения включает в себя разработку структурной схемы, которая дает достаточно полное представление о проектируемом программном обеспечении. Разрабатываемую программную систему «Абитуриент» можно декомпозировать на ряд функциональных подсистем: ведения базы данных абитуриентов, ведения справочной информации, формирования выходной информации, сервисных функций.

Исходя из этого, строится структурная схема программы и на ее основании проектируется структура Главного меню програм-

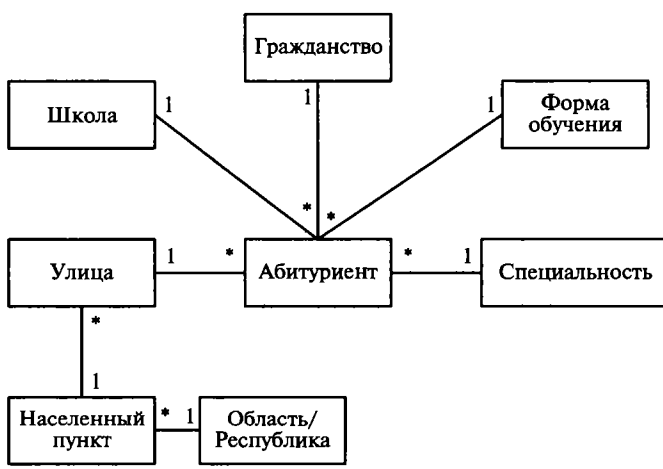


Рис. 13.4. Диаграмма «сущность—связь» для описания базы данных информационной системы «Абитуриент» (начальный вариант)

мы. Структурная схема программного обеспечения разрабатываемой информационной системы «Абитуриент» представлена на рис. 13.6.

Исходя из структурной схемы программы, можно построить абстрактную модель пользовательского интерфейса системы, отражающую последовательность появления экранных форм. Главная форма, очевидно, должна содержать Меню, которое соответствует структурной схеме программы. Через его опции можно активизировать: ввод и редактирование абитуриентов, ведение справочников, формирование отчетности, настройки программы и т.д.

Для ведения базы данных абитуриентов создается экранная форма, через инструментальную панель которой можно произве-

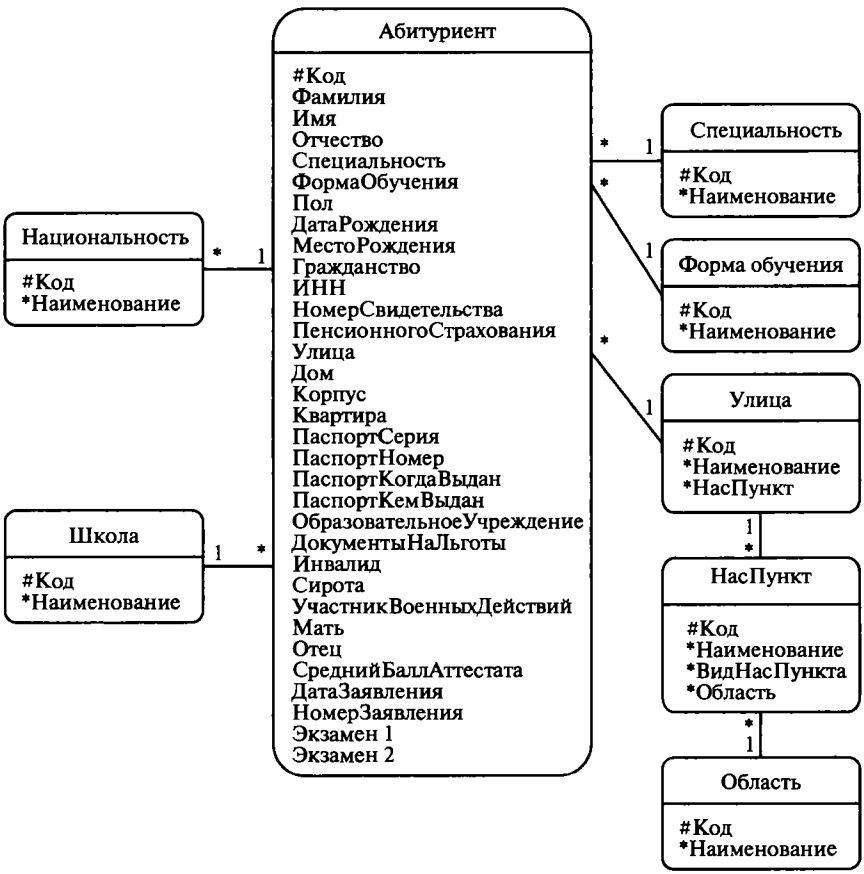


Рис. 13.5. Окончательный вариант ER-диаграммы

Рис. 13.6. Структурная схема программной системы «Абитуриент»

сти удаление записей, вызвать окно ввода и редактирования сведений, осуществить поиск и отбор необходимой информации (рис. 13.7). Ввод и редактирование сведений об абитуриентах осуществляется в отдельном окне. Для создания гибких отчетов по запросу пользователя проектируются формы диалога, в которых можно задать критерии формирования той или иной отчетной формы (рис. 13.8, *а*, *б*, *в*, *г*).

Таким образом, создается эволюционный прототип, который впоследствии будет доработан до уровня готовой программной системы.

3. Реализация

3.1. Обоснование выбора средств разработки

Наиболее подходящей, с точки зрения соответствия поставленной задаче, является платформа «1С:Предприятие», массово используемая для разработки индивидуальных решений. Во-первых, эта система предназначена для автоматизации задач управления и учета и обладает всеми необходимыми механизмами для быст-

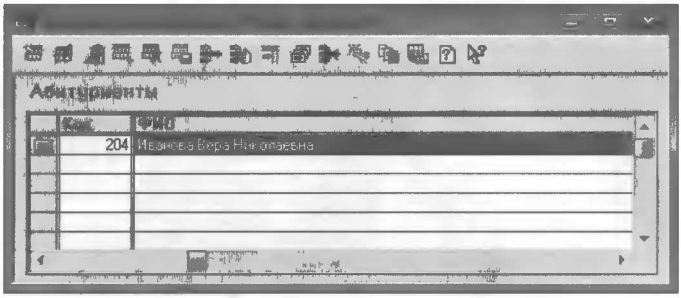


Рис. 13.7. Форма ведения базы данных абитуриентов

рой разработки прикладных решений. Во-вторых, ее использование позволяет существенно упростить и ускорить разработку проекта и значительно снизить его стоимость.

3.2. Описание основных программных модулей

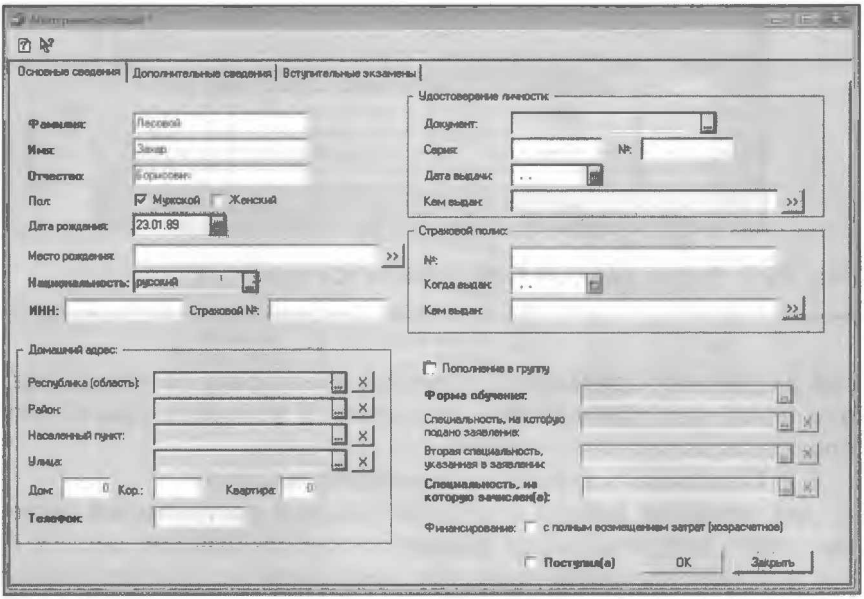
Для описания работы отдельных модулей программной системы можно выбрать модули формирования отчетов (см. рис. 13, *б*, *в* и *г*). Схема алгоритма процедур прерываний стартовой формы показана на рис. 13.9. Символом 1 на рисунке обозначено действие, заключающееся во вводе критериев формирования отчета. Необходимые данные вносятся в соответствующие текстовые поля. Символом 2 обозначено действие, заключающееся в щелчке по кнопке «Сформировать». В процедуре, связанной с этой кнопкой, осуществляется обращение к программному модулю, в котором непосредственно реализуется алгоритм формирования отчета. Щелчком по кнопке «Закрыть» (символ 3) происходит закрытие формы.

Далее следует описать алгоритм формирования отчета в соответствии со стандартом ГОСТ 19.701—90, делая ссылки на текст программы, который должен быть помещен в приложении к курсовому проекту.

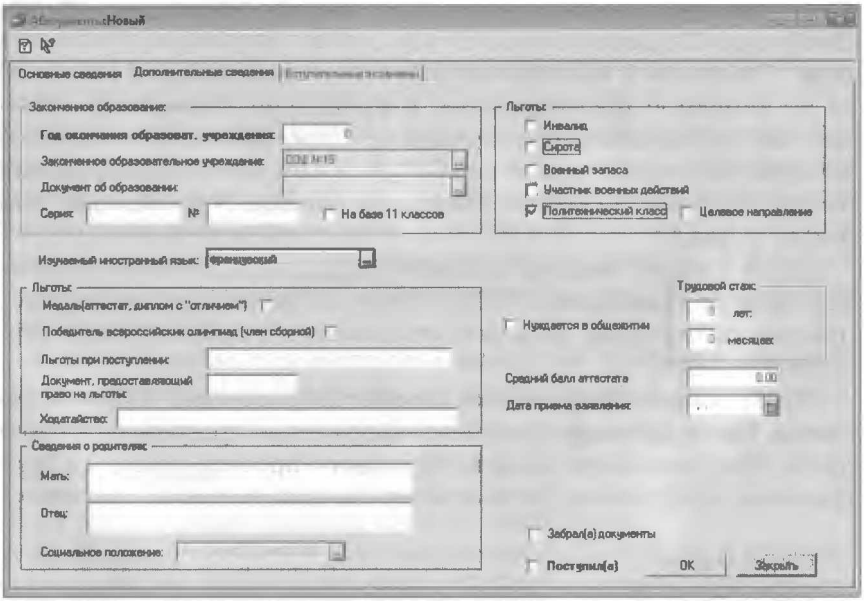
Программное обеспечение автоматизированной тестовой системы. Ниже приведен фрагмент оформления программного продукта. Пояснительная записка курсового проекта по теме «Программное обеспечение автоматизированной тестовой системы».

Введение

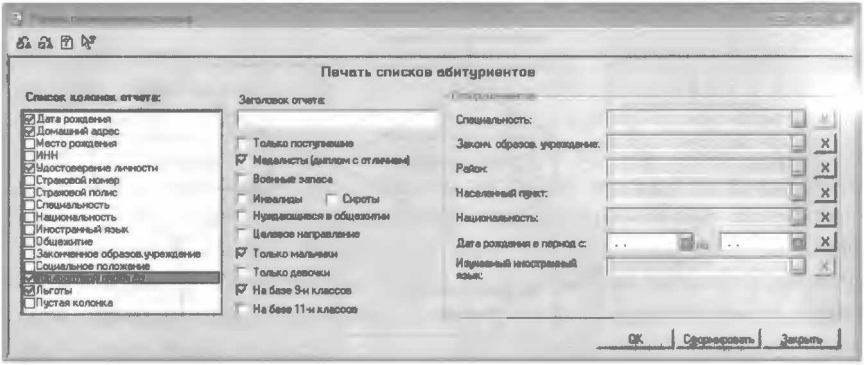
Система компьютерного тестирования знаний — это программная система, призванная обеспечить глубокую и регулярную проверку знаний учащихся. Система может использоваться


Основные сведения
Дополнительные сведения
Вступительные экзамены
Фамилия:
Лесовой
Имя:
Захар
Отчество:
Борисович
Пол:
Мужской
Женский
Дата рождения:
23.01.89
Место рождения:
Национальность:
русский
ИНН:
Страховой №:
Удостоверение личности:
Документ:
Серия:
№:
Дата выдачи:
Кем выдан:
Страховой полис:
№:
Когда выдан:
Кем выдан:
Домашний адрес:
Республика (область):
Район:
Населенный пункт:
Улица:
Дом:
Кор.:
Квартира:
Телефон:
Пополнение в группу
Форма обучения:
Специальность, на которую подано заявление:
Вторая специальность, указанная в заявлении:
Специальность, на которую зачислен(а):
Финансирование:
с полным возмещением затрат (хозрасчетное)
Поступил(а)
OK
Закрыть

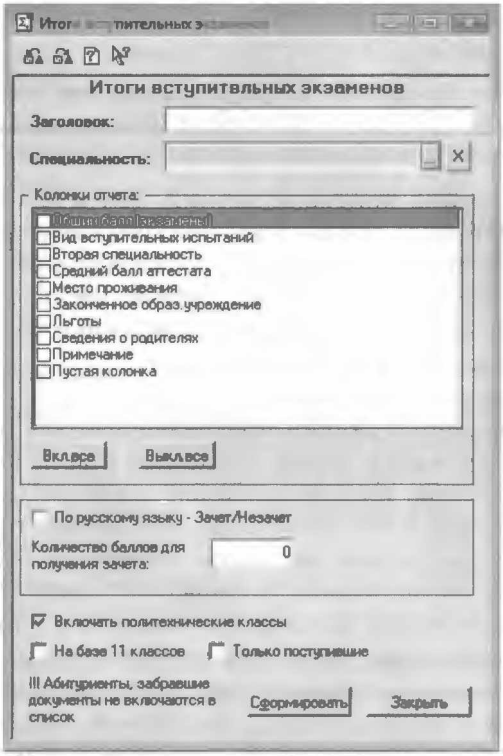
а


Новый
Основные сведения
Дополнительные сведения
Законченное образование:
Год окончания образоват. учреждения:
Законченное образовательное учреждение:
СОШ №15
Документ об образовании:
Серия:
№:
На базе 11 классов
Льготы:
Инвалид
Сирота
Военный запаса
Участник военных действий
Политехнический класс
Целевое направление
Изучаемый иностранный язык:
французский
Льготы:
Медаль(аттестат, диплом с "отличием")
Победитель всероссийских олимпиад (член сборной)
Льготы при поступлении:
Документ, предоставляющий право на льготы:
Ходатайство:
Трудовой стаж:
лет:
месяцев
Нуждается в общежитии
Средний балл аттестата
0.00
Дата приема заявления:
Сведения о родителях:
Мать:
Отец:
Социальное положение:
Забрал(а) документы
Поступил(а)
OK
Закрыть

б



в



г

Рис. 13.8. Экранные формы:

а — ввод сведений об абитуриенте; *б* — редактирование сведений об абитуриенте; *в* — создание списков абитуриентов; *г* — создание отчетов по итогам вступительных экзаменов

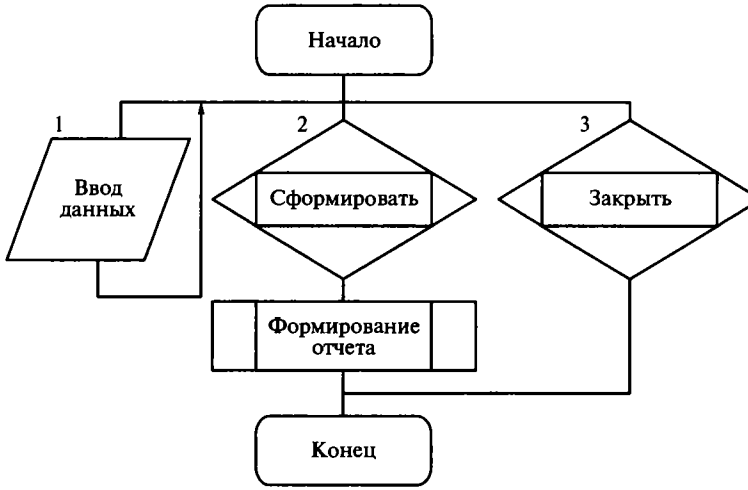


Рис. 13.9. Схема алгоритма процедур прерываний стартовой формы

как для собственно контроля знаний, так и для самоконтроля (пробного тестирования). Цель — использование новейших компьютерных технологий обучения, а также изучение психолого-педагогических вопросов при использовании тестов и электронных учебников в учебном процессе в учреждениях образования.

1. Разработка системного проекта

1.1. Назначение разработки

Система проверки знаний предназначена для автоматизации процесса тестирования знаний студентов при любом количестве групп тестируемых, по любым темам и дисциплинам. Система должна обеспечить возможность задавать любые темы тестирования, выбирать различные варианты ответов на вопросы, оценивать результат тестирования, устанавливать минимально необходимый балл для оценки знаний, накапливать и обрабатывать результаты тестирования, вести список тестируемых.

В системе должен использоваться дифференцированный подход по временно́му параметру к участникам тестирования, ограничиваться доступ тестируемых к просмотру и редактированию заданий, сопоставляться ответы тестируемых с правильными вариантами ответов. Результаты тестирования должны сохраняться во внешнем файле и распечатываться. В системе должна быть организована передача материалов по сети Интернет.

1.2. Требования к функциональным характеристикам

1.2.1. Состав выполняемых функций

В разрабатываемой автоматизированной тестовой системе должны храниться названия тестов, вопросы, иллюстрации к вопросам, варианты ответов. Для каждого вопроса должно выделяться определенное количество времени на подготовку ответа и количество баллов за правильный ответ. В базе данных должны также сохраняться сведения о тестируемых студентах (ФИО, группа), дата тестирования и результаты тестирования (номер вопроса, номер выбранного ответа, верный или неверный был дан ответ), количество баллов, набранное студентом. Система должна включать в себя тренировочные тесты по отдельным разделам дисциплин, объединенные контрольные тесты по целым темам, а также специальные итоговые тесты по всем темам различных курсов.

1.2.2. Организация входных и выходных данных

Входные данные тестовой системы:

- тесты — название теста, создатель теста, дисциплина; вопросы к тесту, варианты ответов, номер правильного ответа, количество баллов за верный ответ («вес» вопроса), время на ответ, иллюстрация вопросу, комментарий к вопросу;
- сведения о тестируемых студентах (ФИО, группа), дата тестирования и результаты тестирования (номер вопроса, номер выбранного ответа, верный или неверный был дан ответ), количество баллов, набранное студентом.

Выходные данные:

- протоколы тестирования, в которых указываются дата тестирования, название теста, дисциплина и результаты тестирования в виде таблицы (вопрос, номер выбранного ответа, верный или неверный был дан ответ), количество набранных баллов;
- анализ успеваемости по результатам тестирования (по группам, по дисциплинам, по отдельным студентам).

1.3. Требования к надежности и безопасности

Система должна функционировать в многопользовательском режиме, поэтому каждый пользователь должен иметь свой пароль доступа в систему. Необходимо разграничить пользовательские права, т.е. не дать студентам возможности редактировать тесты и изменять результаты тестирования, редактировать справочники.

Одновременно в системе может находиться несколько десятков пользователей (например, при тестировании нескольких

групп). Система должна обеспечить одновременный доступ к одним и тем же данным в базе данных, для чего необходимо применение архитектуры клиент—сервер.

Кроме того, в системе должно быть предусмотрено резервное копирование и восстановление данных, а также самовосстановление системы после сбоев в операционной системе или отключения электропитания.

1.4. Требования к составу и параметрам технических средств

Системные требования для компьютера-сервера:

- тактовая частота процессора > 2 000 Гц;
- объем оперативной памяти 512 Мб;
- объем свободного дискового пространства 50 Мб;
- разрешение монитора 1 024 × 768.

Системные требования для рабочей станции:

- тактовая частота процессора ~ 1 000 Гц;
- объем оперативной памяти 64 Мб;
- объем свободного дискового пространства 20 Мб;
- разрешение монитора 1 024 × 768.

1.5. Требования к информационной и программной совместимости

На сервере должны быть установлены Windows Server 2003 и SQL Server 2003. На компьютерах-клиентах должна быть установлена операционная система Windows 2000/XP. Все формируемые отчеты должны иметь возможность экспортирования в редактор электронных таблиц MS Office Excel 2003/2007.

2. Разработка технического проекта

2.1. Анализ требований и определение спецификаций программного обеспечения

Рассмотрим определение прецедентов (вариантов использования). Система тестирования требуется прежде всего следующим заинтересованным лицам:

- обучаемый (студент);
- составитель тестов (преподаватель);
- преподаватель, принимающий экзамен;
- заведующий учебной частью, осуществляющий контроль за успеваемостью;
- администратор сети учебного заведения.

На начальном этапе создания системы можно ограничиться только двумя важными ролями действующих лиц: студент (тестируемый) и администратор (он же преподаватель и составитель тестов).

Соответственно основные прецеденты (варианты использования) для разрабатываемой системы следующие.

Прецедент для студента:

- П1 — пройти тестирование.

Прецеденты для администратора:

- П2 — создать (изменить) тест;
- П3 — просмотреть результаты тестирования;
- П4 — добавить (изменить) пользователей и др.

Вариант использования можно описать кратко или подробно. Краткая форма описания содержит название варианта использования, его цель, действующих лиц, тип варианта использования (основной, вспомогательный, дополнительный) и его краткое описание (табл. 13.1). Подробное описание варианта использования «Тестирование» приведено в табл. 13.2.

Для большей наглядности можно построить диаграмму вариантов использования (рис. 13.10).

Для того чтобы учесть временной аспект поведения системы, можно построить диаграмму последовательности. В качестве примера построим диаграмму последовательности для реализации

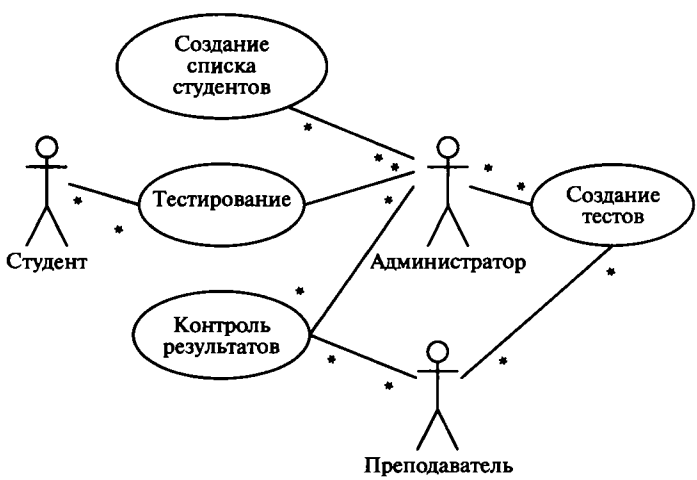


Рис. 13.10. Диаграмма вариантов использования тестовой системы

Таблица 13.1. Описание варианта использования

Название варианта	Тестирование
Цель	Прохождение теста, получение оценки
Действующие лица (актеры)	Студент
Краткое описание	Регистрация студента, запуск теста, получение вопроса и вариантов ответов, выбор ответа из нескольких предложенных или ввод ответа, завершение теста, получение оценки
Тип варианта	Базовый

Таблица 13.2. Описание варианта использования «Тестирование»

Действия исполнителя	Отклик системы
1. Студент вводит свои данные (ФИО, группа), регистрируется в системе	2. Система проверяет данные, предлагает выбрать тест
3. Студент выбирает тест	4. Система запускает тест
5. Студент последовательно отвечает на вопросы	6. Система регистрирует правильные и неправильные ответы
7. Студент завершает тестирование	8. Система подсчитывает процент правильных ответов
9. Студент ожидает результат	10. Система демонстрирует результат и предлагает: сохранить результаты распечатать протокол тестирования
11. Студент решает, печатать результат или нет	12. Если выбрана печать, система выводит на печать протокол тестирования
13. Студент решает, сохранять результат или нет	14. Если выбрано сохранение, система сохраняет результат
15. Студент завершает работу	16. Система завершает работу

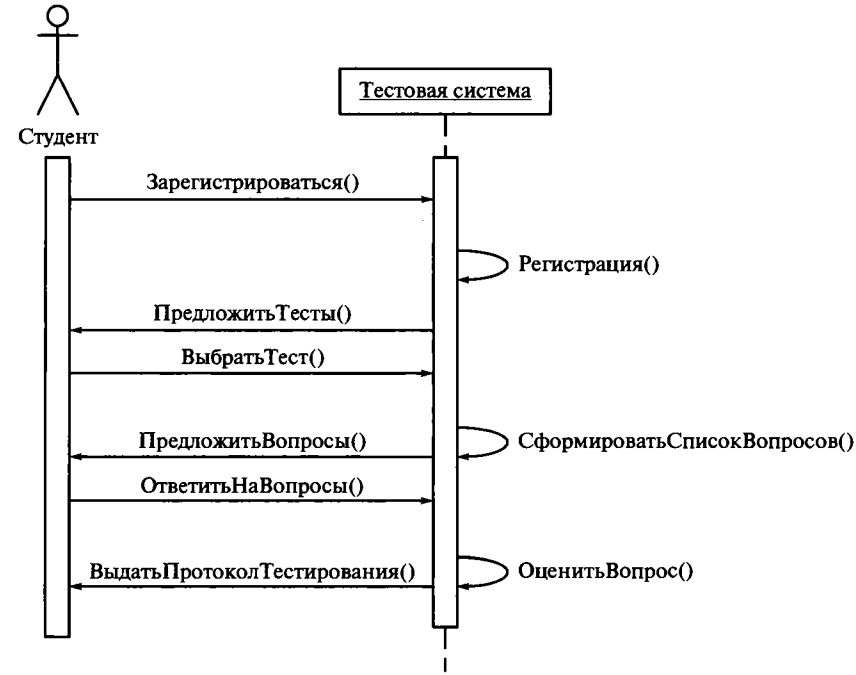


Рис. 13.11. Диаграмма последовательности для реализации варианта использования «Тестирование» в информационной тестовой системе

варианта использования «Тестирование» в информационной тестовой системе (рис. 13.11). Для анализа требований можно использовать и структурный подход. На рис. 13.12, 13.13 представлена диаграмма потоков данных в нотации Гейна—Сарсона, описывающая основные процессы, происходящие в системе. Сначала составляем контекстную диаграмму начального уровня. Вне-

Рис. 13.12. Начальная контекстная диаграмма DFD тестовой системы (нулевого уровня)

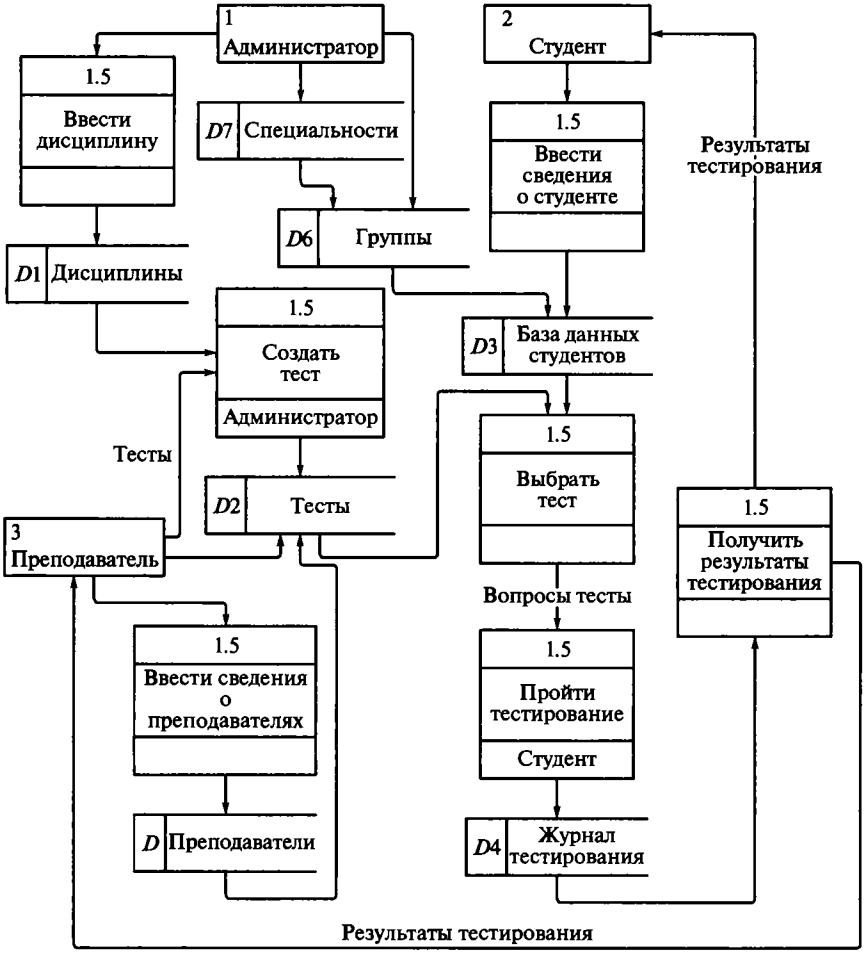


Рис. 13.13. Диаграмма системных процессов первого уровня

Рис. 13.14. Начальный вариант концептуальной модели данных

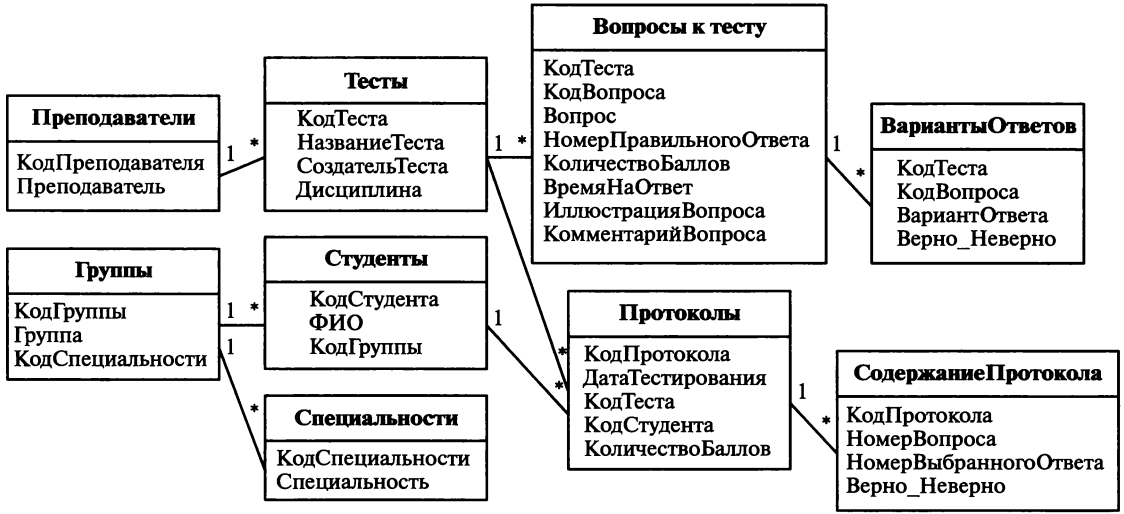


Рис. 13.15. Диаграмма «сущность—связь» для автоматизированной тестовой системы

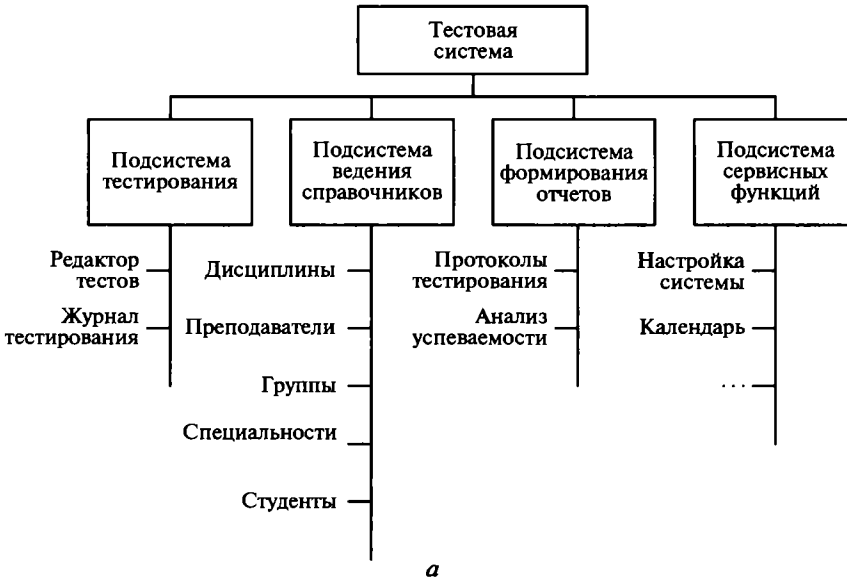


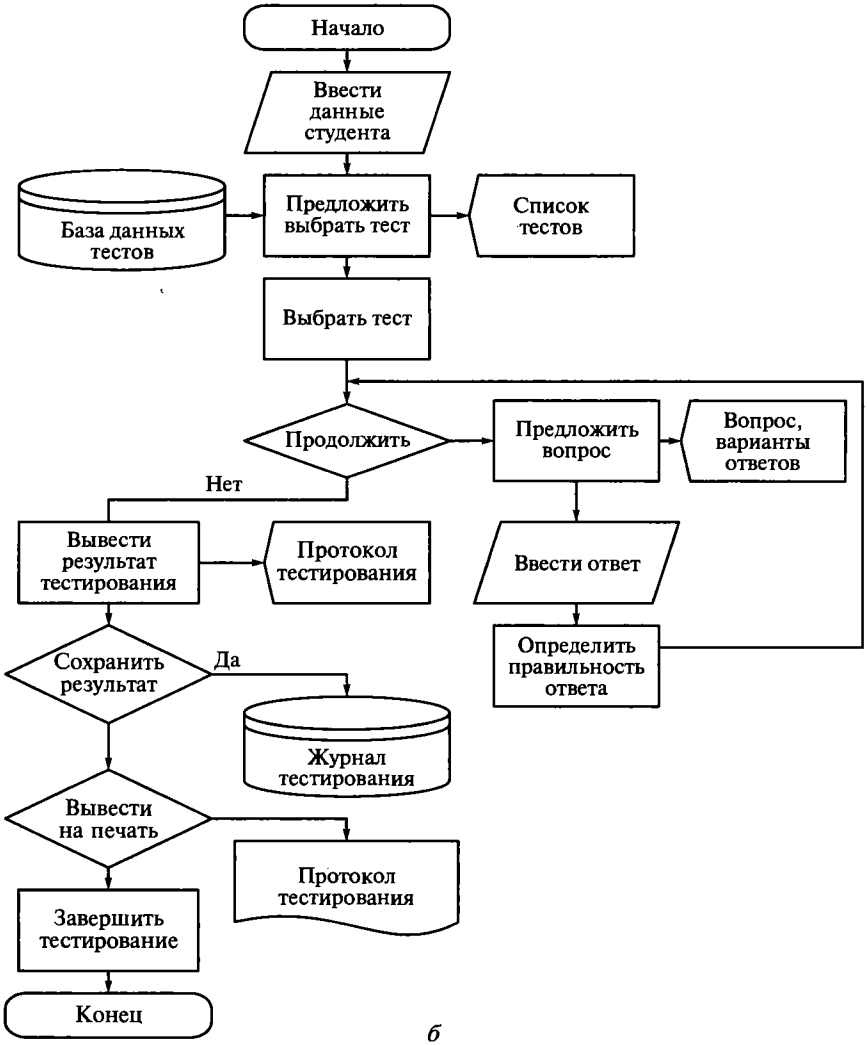
Рис. 13.16. Схемы автоматизированной тестовой системы:
а — структурная; *б* — функциональная

шними сущностями на ней являются Студенты и Преподаватели, только они взаимодействуют с системой. Преподаватели создают тесты и вводят их в систему, студенты выбирают тест и отвечают на предлагаемые вопросы (см. рис. 13.12). На следующем уровне детализации происходит выделение всех процессов и потоков данных в системе, выделяются накопители данных, которые будут описаны посредством структур данных (см. рис. 13.13). После того как определен состав потоков данных, на основании исходной информации можно приступить к конструированию концептуальной модели данных.

2.2. Проектирование модели данных

На основании моделей, построенных в предыдущем разделе, выделим и нарисуем сущности для каждого объекта данных в автоматизированной тестовой системе. Установим существование связи между ними:

1) Преподаватели составляют Тесты;
2) Тесты содержат Вопросы;
3) Вопросам соответствуют Варианты ответов;
4) Студенты обучаются в Группе;



б

5) Группа принадлежит Специальности;

6) Студенты проходят тестирование (получают Протокол);

7) Протокол содержит результаты (Содержание протокола).

Связь должна отражать взаимодействие между сущностями, причем в системе должна сохраняться информация об этом взаимодействии. Нарисуем диаграмму «сущность—связь» (рис. 13.14).

Далее определяем атрибуты каждой сущности. Диаграмма «сущность — связь» показана на рис. 13.15.

2.3. Детальное проектирование программного обеспечения (конструирование прототипа)

Для детального проектирования программного обеспечения построим структурную и функциональную схемы программы (рис.13.16, *а*, *б*). С использованием этих схем легко построить иерархию экранных форм (сконструировать прототип).

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Чему учатся студенты в процессе курсового проектирования?
2. Из каких основных разделов состоит «Пояснительная записка к курсовому проектированию»?
3. В каком порядке оформляют курсовой проект?

ПРИЛОЖЕНИЕ

Варианты заданий для выполнения на практических занятиях

1. Разработать программное обеспечение автоматизированной информационной системы «Книжный магазин». База данных должна содержать сведения о поступлении книг, включая дату поступления, номер документа и сведения о поставщике; сведения о книгах — жанр, название, автор(ы), год издания, издательство, место издания, количество страниц, цена; сведения о реализации книг (дата продажи, количество экземпляров, сумма). Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

2. Разработать программу для построения графика функции, заданной параметрическим уравнением. При выполнении этого проекта предполагается развитый интерфейс, позволяющий изменять масштаб, менять цвета фона и линий. Предусмотреть возможность вывода координат курсора мыши и параметра t при нажатии на правую кнопку.

3. Разработать программное обеспечение автоматизированной информационной системы «Студенческая библиотека». База данных должна содержать сведения о книгах — жанр, название, инвентарный номер, автор(ы), год издания, издательство, место издания, количество страниц, цена; сведения о выдаче и возврате книг студентам (формуляр); данные о списании книг. Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

4. Разработать программное обеспечение автоматизированной информационной системы «Городская библиотека». База данных должна содержать сведения о книгах — жанр, название, инвентарный номер, автор(ы), год издания, издательство, место издания, количество страниц, цена; сведения о выдаче и возврате книг читателям (формуляр); сведения о читателях (ФИО, адрес, паспортные данные). Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

5. Разработать простейший графический редактор, обладающий функциональными возможностями редактора Paint.

6. Разработать программное обеспечение автоматизированной информационной системы «Школьная библиотека». База данных должна содержать сведения о поступлении книг, включая дату поступления и поставщиков, номер документа; сведения о книгах — жанр, название, инвентарный номер, автор(ы), год издания, издательство, место издания, количество страниц, цена; сведения о выдаче и возврате книг учащимся (формуляр); данные о списании книг. Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

7. Разработать программное обеспечение автоматизированной информационной системы «Отдел кадров предприятия». База данных должна содержать сведения о работниках предприятия, включая ФИО, пол, дату рождения, образование, должность, профессию, подразделение, дату поступления на работу, оклад, паспортные данные, адрес. Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

8. Разработать программное обеспечение автоматизированной информационной системы «Студенческий отдел кадров». База данных должна содержать сведения о студентах техникума, включая ФИО, пол, дату рождения, адрес проживания, телефон, сведения о родителях, рабочие телефоны родителей, курс, группу, специальность, отделение, вид финансирования, год поступления, год окончания, номер студенческого билета. Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

9. Разработать программное обеспечение автоматизированной информационной системы «Магазин музыкальных инструментов». База данных должна содержать сведения о поступлении музыкальных инструментов в магазин (включая дату поступления, номер документа, сведения о поставщике, количество, сумму), сведения об инструментах (название, вид инструмента, цена); сведения о продажах музыкальных инструментов покупателям (дата продажи, количество, сумма). Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

10. Разработать программное обеспечение автоматизированной информационной системы «Музыкальный салон». База данных должна содержать все сведения о кассетах и CD-дисках, поступающих для продажи. В ней должны быть данные о музыкальных произведениях (жанр, название, исполнитель, год выпуска), сведения о поступлении музыкальных кассет и дисков (включая дату поступления, номер документа, сведения о поставщике, количество поставляемых дисков (кассет), сумму поступления), а также сведения о продажах музыкальных дисков (дата продажи, количество проданных дисков, сумма продажи). Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

11. Разработать программное обеспечение автоматизированной информационной системы «Склад оптовой торговли». База данных должна содержать сведения о поступлении товаров на склад (включая дату поступления, номер документа, сведения о поставщике, количество товара, сумму), сведения о товаре (название, вид товара, цена); сведения о продажах (дата продажи, количество проданного товара, сумма продажи, сведения о покупателе). Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

12. Разработать программное обеспечение автоматизированной информационной системы «Салон сотовой связи». База данных должна содержать сведения о сотовых телефонах и аксессуарах, имеющихся в салоне. Необходимо указывать модель телефона, фирму-производителя, цену, краткую характеристику, гарантийный срок использования. Нуж-

но вводить сведения о поступлении телефонов и аксессуаров (включая дату поступления, номер документа, сведения о поставщике, количество поставляемого товара, сумму), а также сведения о продажах (дата продажи, количество проданного товара, сумму продажи). Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

13. Разработать программу проектирования плана города. Необходимо поддерживать библиотеку условных обозначений и элементов. В проекте должны быть реализованы функции масштабирования, рисования и удаления элементов.

14. Разработать программное обеспечение автоматизированной информационной системы «Материальный склад». База данных должна содержать:

- сведения о поступлении материалов на склад (включая дату поступления, номер документа, сведения о поставщике, количество материала, сумму);
- сведения о материалах (название, вид материала, цена);
- сведения о списании материалов (дата списания, номер документа, количество, сумма);
- сведения о передачи материала в производство (дата передачи, номер документа, количество, сумма).

Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

15. Разработать программное обеспечение автоматизированной информационной системы «Учет и выдача спецодежды на предприятии». База данных должна содержать:

- сведения о поступлении спецодежды на склад (включая дату поступления, номер документа, данные о поставщике, количество поставляемой спецодежды);
- сведения о спецодежде (название, вид спецодежды (обувь, халат и т.д.), цена);
- сведения о выдаче спецодежды сотрудникам — дата выдачи, данные о сотруднике, получившем спецодежду (ФИО, профессия, должность), срок использования спецодежды.

Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

16. Разработать программное обеспечение автоматизированной информационной системы «Учебная часть». База данных должна содержать все сведения, которые заносятся в зачетную книжку студента (номер группы, специальность, отделение, ФИО студента, семестры, дисциплины, дата сдачи экзамена или зачета, преподаватели, оценки). Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

17. Разработать программное обеспечение автоматизированной информационной системы «Абитуриент». База данных должна содержать анкетные данные, которые указывает при подаче заявления поступающий. ФИО, дата рождения, гражданство, пол, домашний адрес, выбран-

ная специальность, телефон, законченное образовательное учреждение и год его окончания, данные о родителях, дополнительные сведения (инвалид, сирота, нуждается в общежитии), изучаемый иностранный язык, средний балл аттестата. Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

18. Разработать программное обеспечение автоматизированной информационной системы, предназначенной для контроля поступления оплаты за обучение студентов («Поступление оплаты»). В базе данных хранятся сведения о студентах (ФИО, группа, курс, специальность), сведения о родителях, сведения о поступлении денег в кассу (дата поступления и сумма). Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

19. Разработать программу для администрирования локальной сети. Программа должна устанавливаться на компьютерах-клиентах и компьютере-сервере, с которого будет производиться опрос устройств компьютеров-клиентов. Пользователь на сервере может в любое время управлять компьютером-клиентом.

20. Разработать программное обеспечение автоматизированной тестовой системы, в которой хранятся название тестов, вопросы, иллюстрации к вопросам, варианты ответов, ограничение времени на ответ, номер правильного ответа, количество баллов за правильный ответ. Кроме того, должны сохраняться сведения о тестируемых студентах (ФИО, группа), дата тестирования и результаты тестирования (номер вопроса, номер выбранного ответа, верный или неверный был дан ответ), количество баллов, набранное студентом. Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

21. Разработать программное обеспечение для ведения электронного классного журнала. В базе данных хранятся номер группы, списки студентов в каждой группе. По каждой дисциплине указываются дата заполнения журнала, преподаватель, тема дисциплины, заносятся оценки и отметки о пропуске занятия. Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

22. Разработать программу для администрирования локальной сети. Программа должна устанавливаться на компьютерах-клиентах и компьютере-сервере, с которого будет производиться опрос устройств компьютеров-клиентов. Пользователь может в любое время получить список устройств и системного программного обеспечения, имеющихся на компьютерах в локальной сети.

23. Разработать программное обеспечение для автоматизированной информационной системы «Учебная группа». База данных должна содержать сведения о студентах одной студенческой группы техникума, включая ФИО, пол, дату рождения, адрес проживания, телефон, сведения о родителях, рабочие телефоны родителей, номер студенческого билета, а также сведения об успеваемости студентов (данные зачетной книжки). Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

24. Разработать программное обеспечение автоматизированной информационной системы «Электронный каталог CD-дисков».

25. Разработать программное обеспечение автоматизированной информационной системы «Компьютерный салон». База данных должна содержать сведения о поступлении и реализации компьютеров и их комплектующих: дата поступления, номер документа, поставщик, тип комплектующего устройства, его модель и производитель. Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

26. Разработать программу проектирования схем локальных вычислительных сетей (ЛВС). Проект должен поддерживать возможность создания плана отдельного этажа здания, создания и редактирования элементов ЛВС, инструментарий для расстановки этих элементов по плану здания.

27. Разработать программное обеспечение автоматизированной информационной системы учета компьютеров и комплектующих, имеющихся в учебном заведении. База данных должна содержать сведения о наличии и перемещении компьютеров и их комплектующих внутри учебного заведения: номер компьютера, место нахождения (номер кабинета и материально ответственное лицо), состав компьютера — перечень комплектующих устройств (тип комплектующего устройства, его модель и производитель). Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

28. Разработать программное обеспечение автоматизированной информационной системы «Городской телефонный справочник». База данных должна содержать фамилию имя и отчество абонента, домашний адрес и номер телефона. Должны учитываться городские поселки и села. Кроме того, в базе данных должны учитываться служебные телефоны (наименование предприятия, адрес, отдел, номер телефона).

29. Разработать программное обеспечение автоматизированной информационной системы «Банк данных жителей города». База данных должна содержать анкетные данные: ФИО, дата рождения, гражданство, пол, домашний адрес, место рождения, ИНН, страховой номер, телефон, семейное положение, дополнительные сведения (инвалид, сирота), место работы, номер избирательного участка. Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

30. Разработать программу, которая наглядно иллюстрирует работу методов сортировки: выборкой и вставкой.

Провести сравнение этих сортировок по количеству сравнений, по количеству обменов. Для этого нужно построить графики зависимостей данных величин от количества элементов массива.

31. Разработать программу, которая наглядно иллюстрирует работу методов сортировки: простой вставкой и бинарной вставкой.

Провести сравнение этих сортировок по количеству сравнений, по количеству обменов. Для этого нужно построить графики зависимостей данных величин от количества элементов массива.

32. Разработать программу, которая наглядно иллюстрирует работу методов сортировки: выборкой, простой вставкой и бинарной вставкой.

Провести сравнение этих сортировок по количеству сравнений, по количеству обменов. Для этого нужно построить графики зависимостей данных величин от количества элементов массива.

33. Разработать программное обеспечение автоматизированной информационной системы «Видеосалон». База данных должна содержать все сведения о кассетах и CD-дисках, поступающих для продажи. В ней должны быть данные о произведениях (жанр, название, исполнители, год и страна выпуска), сведения о поступлении видеокассет и дисков (включая дату поступления, номер документа, сведения о поставщике, количество поставляемых дисков (кассет), сумму поступления), а также сведения о продажах видеодисков (дата продажи, количество проданных дисков, сумма продажи). Создать экранные формы для ввода и редактирования данных в таблицах и все необходимые выходные отчеты.

Список литературы

1. ГОСТ 19.701—90 (ИСО 5807-85). Единая система программной документации (ЕСПД).
2. *Архангельский А.Я.* Delphi 7 / А.Я.Архангельский. — М.: ЗАО «Издательство БИНОМ», 2004.
3. *Архангельский А.Я.* Приемы программирования в Delphi / А.Я.Архангельский. — М.: ЗАО «Издательство БИНОМ», 2004.
4. Базы данных / под ред. А.Д.Хомоненко. — СПб.: КОРОНА принт, 2002.
5. *Бобровский С.* Delphi 7 / С.Бобровский. — СПб.: Питер, 2005.
6. *Бутаков Е.А.* Методы создания качественного программного обеспечения ЭВМ / Е.А.Бутаков. — М.: Энергоатомиздат, 1984.
7. *Ван-Тассел Д.* Стиль, разработка, эффективность, отладка и испытание программ / Д.Ван-Тассел. — М.: Мир, 1981.
8. *Вендров А.М.* Практикум по проектированию программного обеспечения экономических информационных систем / А.М.Вендров. — М.: Финансы и статистика, 2002.
9. *Гагарина Л.Г.* Основы технологии разработки программных продуктов / Л.Г.Гагарина, Б.Д.Виснадул, А.В.Игошин.— М.: ФОРУМ — ИНФРА-М, 2006.
10. *Голицына О.Л.* Базы данных / О.Л.Голицына, Н.В.Максимов, И.И.Попов. — М.: ФОРУМ — ИНФРА-М, 2003.
11. *Горев А.* Эффективная работа с СУБД / А.Горев, Р.Ахаян, С.Макашарипов. — СПб.: Питер, 1997.
12. *Дарахвелидзе П.Г.* Программирование в Delphi 7 / П.Г.Дарахвелидзе, Е.П.Марков, О.А.Котенок. — СПб.: БХВ — Санкт-Петербург, 2005.
13. *Емельянова Н.З.* Основы построения автоматизированных информационных систем / Н.З.Емельянова, Т.Л.Партыка, И.И.Попов. — М.: ФОРУМ: ИНФРА-М, 2005.
14. *Жилинский А.А.* Самоучитель Microsoft SQL SERVER 2005 / А.А.Жилинский. — СПб.: БХВ-Петербург, 2007.
15. *Жоголев Е.А.* Введение в технологию программирования: конспект лекций / Е.А.Жоголев. — М.: ДИАЛОГ-МГУ, 1994.
16. Информатика / под ред. Н.В.Макаровой. — 3-е изд., перераб.; — М.: Финансы и статистика, 2004.

17. *Кандзюба С. П.* Delphi. Базы данных и приложения / С. П. Кандзюба, В. Н. Громов. — СПб.: ООО «ДиаСофтЮп», 2005.
18. *Ковязин А.* Мир InterBase / А. Ковязин, С. Востриков. — М.: КУДИЦ-ОБРАЗ, 2006.
19. *Кузьменко В. Г.* Базы данных в Visual Basic VBA. Самоучитель / В. Г. Кузьменко. — М.: ООО «Бином-Пресс», 2007.
20. *Культин Н.* Программирование на Object Pascal / Н. Культин. — СПб.: БХВ-Петербург, 1998.
21. *Орлов В. В.* Технологии разработки программных продуктов / В. В. Орлов. — СПб.: Питер, 2003.
22. *Пономарев В. А.* COM и ActiveX в Delphi / В. А. Пономарев. — СПб.: БХВ — Петербург, 2001.
23. *Рудаков А. В.* Технология разработки программных продуктов / А. В. Рудаков. — М.: Издательский центр «Академия», 2005.
24. *Сорокин А. В.* Delphi. Разработка баз данных / А. В. Сорокин. — Спб.: Питер, 2005.
25. *Тейлор Аллен Дж.* SQL для «чайников» / Аллен Дж. Тейлор; пер. с англ. — М.: Вильямс, 2005.
26. *Фаронов В. В.* Программирование баз данных в Delphi 7 / В. В. Фаронов. — СПб.: Питер, 2005.
27. *Фаронов В. В., П. В. Шумаков.* Delphi. Руководство разработчика баз данных / В. В. Фаронов, П. В. Шумаков. — М.: Нолидж, 1999.
28. *Фленов М. Е.* Delphi 2005. Секреты программирования / М. Е. Фленов. — СПб.: Питер, 2006.
29. *Фленов М. Е.* Библия Delphi / М. Е. Фленов. — СПб.: БХВ-Петербург, 2005.
30. *Шкрыль А. А.* Разработка клиент-серверных приложений в Delphi / А. А. Шкрыль. — СПб.: БХВ-Петербург, 2006.
31. *Шураков В. В.* Надежность программного обеспечения / В. В. Шураков. — М.: Финансы и статистика, 1987.

Оглавление

Учебное издание

Рудаков Александр Викторович,
Федорова Галина Николаевна

Технология разработки программных продуктов. Практикум

Учебное пособие

4-е издание, стереотипное

Редактор *М.Ю.Волкова.*
Технический редактор *Н.И.Горбачёва*
Компьютерная верстка: *Е.Ю.Назарова*
Корректоры *А.П.Сизова, Г.А.Форысенкова*

Изд. № 104114016. Подписано в печать 02.09.2013. Формат 60×90/16.
Гарнитура «Балтика». Бумага офс. № 1. Печать офсетная. Усл. печ. л. 12,0.
Тираж 1 200 экз. Заказ № 34741.

ООО «Издательский центр «Академия». www.academia moscow.ru
129085, Москва, пр т Мира, 101В, стр. 1.
Тел./факс: (495) 648 0507, 616 00 29.
Санитарно эпидемиологическое заключение № РОСС RU. АЕ51. Н 16476 от 05.04.2013.

Отпечатано в соответствии с качеством предоставленных издательством
электронных носителей в ОАО «Саратовский полиграфкомбинат».
410004, г. Саратов, ул. Чернышевского, 59. www.sarpk.ru