

И Н Ж Е Н Е Р И Я ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические указания
к лабораторным занятиям по дисциплине

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра информационных технологий и компьютерных систем

ИНЖЕНЕРИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические указания
к лабораторным занятиям по дисциплине
для обучающихся очной формы обучения
направления подготовки
09.03.01 – Информатика и вычислительная техника

Севастополь
СевГУ
2024

УДК 004.413(076.5)
ББК 32.972я73
И621

Р е ц е н з е н т:

В. Ю. Карлусов – доцент кафедры «Информационные системы»,
канд. техн. наук, доцент

Составитель: М. А. Лебедева

И621 Инженерия программного обеспечения : методические указания к лабораторным занятиям по дисциплине «Инженерия программного обеспечения» для обучающихся очной формы обучения направления подготовки 09.03.01 – Информатика и вычислительная техника / Севастопольский государственный университет, Институт информационных технологий, кафедра информационных технологий и компьютерных систем ; сост. М. А. Лебедева – Севастополь : СевГУ, 2024. – 73 с. – Текст : электронный.

Дисциплина «Инженерия программного обеспечения» включена в блок обязательных дисциплин учебного плана. Целью методических указаний является оказание помощи обучающимся в изучении дисциплины. Методические указания содержат краткое изложение основных теоретических положений, инструкции по выполнению лабораторных работ, контрольные вопросы и список рекомендованной литературы.

УДК 004.413(076.5)
ББК 32.972я73

Методические указания рассмотрены и утверждены на заседании кафедры «Информационные технологии и компьютерные системы», протокол № 10 от 12 апреля 2024 г.

© Лебедева М.А., сост., 2024
© ФГАОУ ВО «Севастопольский
государственный университет», 2024

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА № 1. РАЗРАБОТКА ТРЕБОВАНИЙ К ПРОГРАММНОМУ ИЗДЕЛИЮ	4
ЛАБОРАТОРНАЯ РАБОТА № 2. РАЗРАБОТКА ТЕХНИЧЕСКОГО ЗАДАНИЯ	11
ЛАБОРАТОРНАЯ РАБОТА № 3. ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОГРАММНОГО ПРОДУКТА.....	15
ЛАБОРАТОРНАЯ РАБОТА № 4. ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ	21
ЛАБОРАТОРНАЯ РАБОТА № 5. УПРАВЛЕНИЕ ПРОЕКТОМ.....	30
ЛАБОРАТОРНАЯ РАБОТА № 6. РАСЧЕТ МЕТРИК ЧИДАМБЕРА - КЕМЕРЕРА.	39
ЛАБОРАТОРНАЯ РАБОТА № 7. ОПРЕДЕЛЕНИЕ ПОКАЗАТЕЛЕЙ КАЧЕСТВА ПРОГРАММНОГО СРЕДСТВА.	44
ЛАБОРАТОРНАЯ РАБОТА № 8. СТРУКТУРНОЕ ТЕСТИРОВАНИЕ.	52
ЛАБОРАТОРНАЯ РАБОТА № 9. ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ	56
ЛАБОРАТОРНАЯ РАБОТА № 10. ПРОГРАММНАЯ ДОКУМЕНТАЦИЯ.	63
БИБЛИОГРАФИЧЕСКИЙ СПИСОК	68
ПРИЛОЖЕНИЕ А	69

Лабораторная работа № 1

РАЗРАБОТКА ТРЕБОВАНИЙ К ПРОГРАММНОМУ ИЗДЕЛИЮ

Цель работы: Изучить содержание основных этапов разработки требований к программному продукту, получить практические навыки разработки требований и построения моделей программного обеспечения.

1.1. Теоретические сведения

1.1.1. Этапы разработки требований

Инженерия программного обеспечения (ПО) – отрасль компьютерной науки, которая охватывает все аспекты создания ПО, а также оценку трудозатрат, производительности и качества.

Схему последовательности выполнения работ при разработке ПО называют моделью жизненного цикла. В настоящее время существует большое количество моделей жизненных циклов ПО, их модификаций и технологий разработки ПО, в том числе и стандарты (ГОСТ 34.601-90, ISO/IEC 12207-1995). Несмотря на многообразие моделей, технология разработки ПО включает следующие базовые процессы:

- разработка требований;
- проектирование и реализация;
- аттестация;
- сопровождение и модернизация;
- снятие с эксплуатации.

Каждый из этапов разработки ПО имеет входные данные, необходимые для выполнения работ на данном этапе, и выходные, являющиеся результатом выполняемых работ.

Создание ПО начинается с определения функциональных возможностей и ограничений, накладываемых на проектируемое программное средство. Процесс формирования, анализа, документирования и проверки этих функциональных возможностей и ограничений называется разработкой требований. Разработка требований часто является критическим этапом создания программного средства, поскольку ошибки, допущенные на этом этапе, ведут к возникновению проблем на этапах проектирования и разработки. Результатом его выполнения является разработка документации, формализующей требования, то есть создание спецификации.

Различают четыре основных этапа процесса разработки требований:

1. анализ технической осуществимости создания системы,
2. формирование и анализ требований,
3. специфицирование требований и создание соответствующей документации,
4. верификация и валидация требований.

Обобщенная модель процесса формирования и анализа требований представлена на рисунке 1.1.

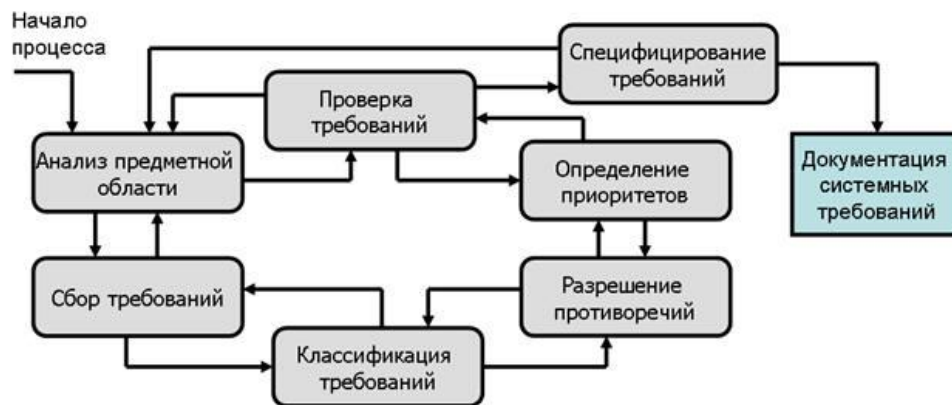


Рисунок 1.1 – Процесс формирования и анализа требований

Процесс формирования и анализа требований проходит через ряд этапов.

1. Анализ предметной области.
2. Сбор требований. Это процесс взаимодействия с лицами, формирующими требования.
3. Классификация требований. На этом этапе бесформенный набор требований преобразуется в логически связанные группы требований.
4. Разрешение противоречий. Требования многочисленных лиц, занятых в процессе формирования требований, будут противоречивыми. На этом этапе определяются и разрешаются противоречия различного рода.
5. Назначение приоритетов. На этом этапе совместно с лицами, формирующими требования, определяются наиболее важные требования.
6. Проверка требований. На этом этапе определяется их полнота, последовательность и непротиворечивость.

1.1.2. Анализ предметной области

Для того чтобы верно определить требования, разработчики системы должны понимать контекст (часть предметной области), в котором работает система.

Модель предметной области описывает важные понятия предметной области и их связи между собой. Модель описывает только объекты предметной области, но не показывает, как программная система будет с ними работать. Например, для предметной области «Телефонная связь» объектами являются вызовы, телефоны, линии, каналы связи, абоненты и параметры.

Цели этапа анализа предметной области:

- 1) определение границ, или контура, системы;
- 2) описание объектов автоматизации и/или формализации знаний об этих объектах;
- 3) выявление или определение потребностей заказчика ПО.

В процессе изучения предметной области должна быть создана ее модель.

1.1.3. Сбор требований

В процессе формулирования требований принимают участие:

- представители от заказчика из нескольких профессиональных групп;
- операторы, обслуживающие систему;
- разработчики системы.

Источниками сведений о требованиях могут быть:

– цели и задачи системы, которые формулирует заказчик. Для однозначного их понимания разработчику системы необходимо их тщательно осмыслить и согласовать с заказчиком;

– действующая система или коллектив, выполняющий ее функции. Система, которую заказывают, может заменять собою старую систему, переставшую удовлетворять заказчика или действующий персонал. Изучение и фиксация ее функциональных возможностей дает основу для учета имеющегося опыта и формулирования новых требований к системе.

Методами сбора требований являются:

- интервью с носителями интересов заказчика и операторами;
- наблюдение за работой действующей системы;
- сценарии (примеры) возможных случаев выполнении ее функций, ролей лиц, запускающих эти сценарии или взаимодействующих с системой во время ее функционирования.

1.1.4. Классификация требований

Требования делятся на функциональные и нефункциональные.

Функциональные требования (functional requirements) описывают поведение системы и сервисы, которые она должна выполнять. («Система должна по электронной почте отправлять пользователю подтверждение о заказе»). Должно быть указано, как система реагирует на те или иные входные данные, как она ведет себя в определенных ситуациях. Иногда указывается, что система не должна делать.

Нефункциональные требования описывают цели и атрибуты качества. Атрибуты качества (quality attributes) представляют собой дополнительное описание функций продукта, выраженное через описание его характеристик, важных для пользователей или разработчиков. К таким характеристикам относятся:

- легкость и простота использования;
- легкость перемещения;
- целостность;
- эффективность и устойчивость к сбоям;
- внешние взаимодействия между системой и внешним миром;
- ограничения дизайна и реализации.

Количественные показатели для нефункциональных требований показаны в таблице 1.1

Таблица 1.1 – Количественные показатели для нефункциональных требований

Показатель	Единицы измерения
Скорость	Количество выполненных транзакций в секунду, время реакции на действия пользователя
Размер	Килобайт
Простота эксплуатации	Время обучения персонала
Надежность	Средняя продолжительность времени между последовательными проявлениями ошибок в программе, вероятность выхода системы из строя
Устойчивость к сбоям	Время восстановления системы после сбоя, процент событий, приводящих к сбоям, вероятность потери данных при сбоях
Переносимость	Процент машинно-зависимых операторов

Существует деление на пользовательские и системные требования, которые могут быть как функциональными, так и нефункциональными.

Требования пользователей (user requirements) описывают цели и задачи, которые пользователи будут решать с помощью системы. Эти требования определяют только внешнее поведение системы. («Система редактирования должна располагать возможностью отображения сетки»). Они должны быть написаны естественным языком с использованием простых таблиц и наглядных диаграмм.

Системные требования (system requirements) – более детализированное описание пользовательских требования. Для описания системных требований используют такие средства, как графические нотации, математические спецификации, языки программирования.

Системные требования включают в себя:

1. Требования к архитектуре системы. Например, число и размещение хранилищ и серверов приложений.

2. Требования к параметрам оборудования. Например, частота процессоров серверов и клиентов, объём хранилищ, размер оперативной и видео памяти, пропускная способность канала и т.д.

3. Требования к параметрам системы. Например, время отклика на действие пользователя, максимальный размер передаваемого файла, максимальная скорость передачи данных, максимальное число одновременно работающих пользователей и т.д.

4. Требования к программному интерфейсу.

- Требования к структуре системы. Например, масштабируемость, распределённость, модульность, открытость.

- масштабируемость – возможность распространения системы на большое количество машин, не приводящая к потере работоспособности и эффективности, при этом способность системы наращивать свою мощность должна определяться только мощностью соответствующего аппаратного обеспечения.

- распределенность - система должна поддерживать распределённое хранение данных.

- модульность - система должна состоять из отдельных модулей, интегрированных между собой.
- открытость - наличие открытых интерфейсов для возможной доработки и интеграции с другими системами.
- Требования по взаимодействию и интеграции с другими системами. Например, использование общей базы данных, возможность получения данных из баз данных определённых систем и т.д.
- Для описания требований используют различные методики. Для описания объектных моделей систем обычно применяют методику вариантов использования, основанную на сценариях.

1.2. Задание на лабораторную работу

1. Изучить теоретический материал.
2. Выбрать объект разработки. Примерные варианты заданий приведены в п. 1.4. Обязательные функции системы: авторизация пользователя, задание критериев поиска информации, взаимодействие с базой данных, визуализация результатов поиска.
3. Для программного изделия выполнить анализ предметной области, привести описание объекта автоматизации и потребности заказчика, построить диаграмму классов, используя язык UML (Приложение А).
4. Выполнить анализ требований разрабатываемой системы, привести описание функциональных и нефункциональных требований.
5. Построить диаграмму вариантов использования, используя язык UML (Приложение А).
6. Оформить отчет.

1.3. Содержание отчета

Постановка задачи с указанием названия и краткого описания разрабатываемой системы, описание результатов анализа предметной области, описание функциональных и нефункциональных требований, диаграммы классов и вариантов использования, выводы.

1.4. Примерные варианты заданий

1. Учет успеваемости студентов. Программный модуль предназначен для оперативного учета успеваемости студентов в сессию деканом и сотрудниками деканата. Сведения об успеваемости студентов должны храниться в течение всего срока их обучения и использоваться при составлении справок о прослушанных курсах и приложений к диплому.
2. Личные дела студентов. Все документы, касающиеся студента: номер студенческого, зачетка, диплом, заявления, отчеты, курсовые и т.д. Разделение: документ, который содержит оценку и влияет на дальнейшую судьбу студента (оперативный) или просто документ (справочный). Программный модуль

предназначен для получения сведений о студентах сотрудниками деканата, профкома и отдела кадров. Сведения должны храниться в течение всего срока обучения студентов и использоваться при составлении справок и отчетов.

3. Картотека автомагазина, предназначенная для использования работниками агентства. В базе содержатся сведения об автомобилях (марка, объем двигателя, дата выпуска и др.). При поступлении заявки на покупку производится поиск подходящего варианта. Если такого нет, клиент заносится в клиентскую базу и оповещается, когда вариант появляется.

4. Автокасса. Содержит сведения о наличии свободных мест на автобусные маршруты. В базе должны содержаться сведения о номере рейса, маршруте, водителе, типе автобуса, дате и времени отправления, а также стоимости билетов. При поступлении заявки на билеты программа производит поиск подходящего рейса.

5. Автостоянка. В программе содержится информация о марке автомобиля, его владельце, дате и времени въезда, стоимости стоянки, скидках, задолженности по оплате и др.

6. Инвентаризация оборудования. Учет компьютерной техники, ее составных частей (клавиатур, мониторов, комплектующих и т.д.) в пределах кафедры (месторасположение, предназначение, ответственный за технику на кафедре, зав. лабораториями).

7. Планирование занятости аудиторий на кафедре. Ведение реестра аудиторий, характеристики аудиторий, ведение расписания занятий с контролем пересечений. Предназначен для преподавателей и зав. лабораториями.

8. Ведение списка студентов и сотрудников кафедры. Процессы: прием студентов, отчисление, перевод из группы в группу, то же – для сотрудников. Учесть, что студент может быть одновременно сотрудником. Программный модуль предназначен для получения сведений о сотрудниками деканата и отдела кадров.

9. Выполнение и контроль лабораторных работ. Процессы: ведение списка студентов, разбиение по подгруппам и бригадам, учет выполнения работы, составления отчета, защиты отчета. Программный модуль предназначен для кураторов и преподавателей дисциплин.

10. Поддержание в актуальном состоянии схемы сети. Процессы: учет компьютеров и сетевого оборудования, связей между ними, и истории изменения этих связей. Программный модуль предназначен для руководства ВЦ и зав. лабораториями.

11. Кадровое агентство, содержатся сведения о вакансиях и резюме. Программный модуль предназначен как для поиска сотрудника, отвечающего требованиям руководителей фирмы, так и для поиска подходящей работы.

12. Книжный магазин, содержатся сведения о книгах (автор, название, издательство, год издания, цена). Покупатель оформляет заявку на нужные ему книги, если таковых нет, он заносится в базу и оповещается, когда нужные книги поступают в магазин.

13. Картотека абонентов АТС. Картотека содержит сведения о телефонах и их владельцах. Фиксирует задолженности по оплате (абонентской и повременной). Считается, что повременная оплата местных телефонных разговоров уже введена.

14. Приложение Windows «Органайзер». Приложение предназначено для записи, хранения и поиска адресов и телефонов физических лиц и организаций, а также расписания, встреч и др. Приложение предназначено для любых пользователей компьютера.

1.5. Список контрольных вопросов

1. Какова цель анализа предметной области?
2. Перечислите методы сбора требований.
3. Чем отличаются пользовательские и системные требования?
4. Модели разработки ПО.
5. Какова цель анализа технической осуществимости проекта?
6. Какие требования относятся к функциональным?
7. Какие требования относятся к нефункциональным?
8. Что такое масштабируемость?

Лабораторная работа № 2

РАЗРАБОТКА ТЕХНИЧЕСКОГО ЗАДАНИЯ

Цель работы: научиться составлять техническое задание (ТЗ) на разработку программного продукта.

2.1. Теоретические сведения

Техническое задание (ТЗ, техзадание) – исходный документ для проектирования сооружения или промышленного комплекса, конструирования технического устройства (прибора, машины, системы управления и т. д.), разработки информационных систем, стандартов либо проведения научно-исследовательских работ (НИР).

2.1.1 Стандарты для составления технического задания

Таблица 2.1 – Стандарты для составления технического задания

Наименование стандартов	Описание
ГОСТ 19.201-78 ГОСТ 34.602-89	
IEEE 830-1998 Recommended Practice for Software Requirements Specifications (рекомендуемые методы спецификации требований к ПО).	Стандарт 830-1998-IEEE Recommended Practice for Software Requirements Specifications предполагает наличие в техническом задании для создания программного обеспечения следующих критериев: Введение – назначение, область действия, определения, ссылки или обзор. Описание проекта – взаимодействие продукта, его функции, характеристики, ограничения и допущения. Требования к внешним интерфейсам, функциям, производительности, а также нефункциональные требования. Ограничения для проекта. Приложения. Алфавитный указатель.
ISO/IEC/ IEEE 29148-2011	Этот стандарт необходим для единой трактовки всех процессов, которые используются во время создания проекта, связанного со сферой технологий. Содержит: Введение – назначение и обзор проекта. Терминологию. Системные требования – функционал, юзабилити, производительность, интерфейс, операции. Физические характеристики. Соответствие безопасности. Правила. Аббревиатуры и сокращения.

2.1.2. Содержание ТЗ (ГОСТ 34)

ТЗ должно включать следующие разделы:

Введение

1 Основание для разработки

2 Назначение разработки

3 Требования к программе

4 Требования к программной документации

5 Техничко-экономические показатели

6 Стадии и этапы разработки

7 Порядок контроля и приемки

В разделе “Введение” указывают наименование, краткую характеристику области применения программы или программного изделия и объекта, в котором будут использовать программу.

1. В разделе “Основания для разработки” указывается документ, на основании которого ведется разработка (приказ по университету, задание на лаб. работу и т.п.); организация, утвердившая документ и дата его утверждения; наименование темы разработки.

2. В разделе “Назначение разработки” указывают функциональное и эксплуатационное назначение программы.

3. Раздел “Требования к программе” должен содержать следующие подразделы:

3.1 “Требования к функциональным характеристикам” – состав функций, которые будет выполнять программа, организация входных и выходных данных (синтаксис и семантика входных данных, форматы выходных сообщений и соответствующие им ситуации), временные характеристики. В этом подразделе должно быть описано поведение системы с точки зрения соотношения входа и выхода без конкретизации внутренней структуры и реакция программы на непредусмотренные данные на входе.

3.2 “Требования к надежности” – контроль входных и выходных данных, последствия возможных отказов, время восстановления, защита от несанкционированного доступа и др.

3.3 “Условия эксплуатации” – характеристики операционной среды, вид обслуживания, количество и квалификация персонала, затрачиваемое время процессора и каналов связи, число пользователей и др., а также допустимые параметры окружающей среды.

3.4 “Требования к составу и параметрам технических средств” – конфигурация системы, основные характеристики требуемых устройств.

3.5 “Требования к информационной и программной совместимости” – требования к методам решения, языкам программирования, программным средствам, используемым системой, протоколам обмена, к СУБД и операционным системам.

3.6 “Требования к маркировке и упаковке” – варианты и способы упаковки (обычно специальных требований не предъявляется).

3.7 “Требования к транспортированию и хранению” – места хранения, условия и сроки, способы создания и хранения резервных копий (обычно специальных требований не предъявляется).

Отдельные разделы и подразделы по согласованию с заказчиком могут быть опущены.

4. В разделе “Требования к программной документации” – состав документации и специальные требования к ней. Виды программных

документов: спецификация, текст программы, описание программы, пояснительная записка, ТЗ, программа и методика испытаний, руководство программиста, руководство системного программиста, руководство оператора, руководство по техническому обслуживанию и т.д.

5. В разделе “Технико-экономические показатели” – экономические преимущества, предполагаемая экономическая эффективность и годовая потребность, экономические преимущества разработки, предельный объем программы, время реакции программы.

6. В разделе “Стадии и этапы разработки” - перечень стадий, разбивка на этапы, содержание, сроки разработки (дни, недели и т.д.) и исполнители. Стадии должны соответствовать этапам жизненного цикла.

7. В разделе “Порядок контроля и приемки – виды испытаний, требования к приемке работ, способы проверки важнейших характеристик. Цель испытаний – установление степени соответствия готового продукта и характеристикам технического задания.

Формы представления результатов: программная документация, конструкторская документация на изделие, программное изделие.

В приложении приводят перечень проведенных научных и исследовательских работ, схемы алгоритмов, таблицы, описания и т.д.

2.1.3. Мастер ТЗ

Мастер ТЗ – бесплатная программа для создания технического задания к разработке программного обеспечения.

Программа обеспечивает легкое создание профессионального ТЗ на разрабатываемую программу в режиме пошагового мастера в соответствии с ГОСТ.

В первом пункте необходимо ввести название программы. Формулировку названия следует взять из темы. Далее следует задать назначение и область применения программы. Под назначением понимается та область деятельности, к которой применим программный продукт.

После завершения разработки технического задания, его следует экспортировать в удобный для распространения формат, поскольку хранить и передавать документ, который можно открыть только при помощи специальной программы, неудобно.

В меню выберете команду экспорта технического задания Файл-Экспорт ТЗ.

2.2. Задание на лабораторную работу

Для выбранного по индивидуальному заданию программного продукта разработать техническое задание в соответствии с ГОСТ 19.201-78. При разработке технического задания использовать результаты лабораторной работы №1. В техническом задании отразить функциональные и нефункциональные, системные и пользовательские требования. Отчет по лабораторной работе должен содержать разделы технического задания.

Для разработки ТЗ можно использовать бесплатные программы «Мастер ТЗ» или «Конструктор ТЗ». («Конструктор ТЗ» используют для сайтов, но его можно использовать для главного окна информационной системы).

2.3. Список контрольных вопросов

1. Назначение технического задания?
2. Кто составляет и утверждает ТЗ?
3. На каком этапе разработки программного изделия составляется ТЗ?
4. Какими документами регламентируется написание ТЗ?
5. Что содержит раздел ТЗ «Введение»?
6. Какие функциональные характеристики отражаются в ТЗ?
7. Что содержит раздел ТЗ «Требования к информационной и программной совместимости»?
8. Каков порядок контроля и приемки программного изделия?

Лабораторная работа № 3

ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОГРАММНОГО ПРОДУКТА

Цель работы: Изучить подходы и методы проектирования архитектуры ПО, получить практические навыки реализации и описания основных этапов проектирования.

3.1. Теоретические сведения

3.1.1. Этапы архитектурного проектирования

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

Первым этапом процесса проектирования является архитектурное проектирование, на котором определяются подсистемы, а также структура управления и взаимодействие подсистем.

Проектирование архитектуры системы проводится разными методами, основанными на известных подходах (структурный, объектно-ориентированный, компонентный и др.). Однако можно выделить основные этапы, общие для всех подходов архитектурного проектирования:

1. Структурирование системы. Система разделяется на несколько подсистем, где под подсистемой понимается независимый программный компонент. Определяются взаимодействия подсистем.

2. Моделирование управления. Определяется модель связей управления между частями системы.

3. Декомпозиция подсистем на модули. Каждая подсистема разбивается на модули. Определяются типы модулей и межмодульные соединения.

Результатом процесса архитектурного проектирования является документ, отображающий архитектуру системы. Он состоит из набора графических схем представлений моделей программной системы, таких как:

- статическая структурная модель;
- динамическая модель процессов;
- интерфейсная модель;
- модели отношений.

3.1.2. Структурирование системы

Структурной называют схему, отражающую состав и взаимодействие по управлению частей разрабатываемого ПО. На первом этапе процесса проектирования архитектуры система разбивается на несколько взаимодействующих подсистем. Подсистема – это система, операции (методы) которой не зависят от сервисов, предоставляемых другими подсистемами. Подсистемы состоят из модулей и имеют определенные интерфейсы. Модуль –

компонент системы, который предоставляет один или несколько сервисов для других модулей. Каждая подсистема может разрабатываться независимо от остальных подсистем.

На самом абстрактном уровне систему можно изобразить графически с помощью схемы, в которой отдельные подсистемы представлены отдельными блоками. Структурными компонентами программной системы или программного комплекса могут служить программы, подсистемы, базы данных, библиотеки ресурсов и т. п. Структурные компоненты изображаются прямоугольниками, а потоки данных или потоки управления изображаются стрелками. Пример структурной схемы приемки товара от клиента показан на рисунке 3.1.

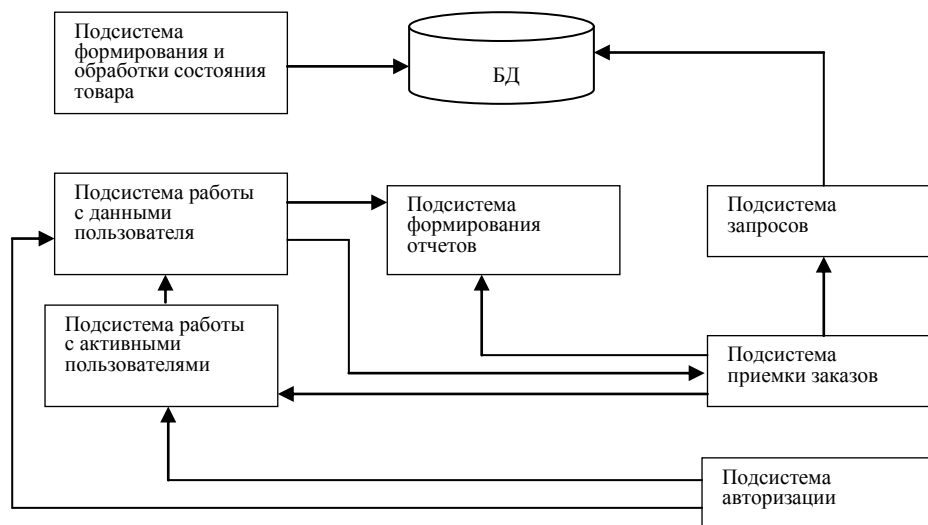


Рисунок 3.1 - Структурная схема системы

Архитектура системы может строиться в соответствии с определенной архитектурной моделью. На практике используются следующие модели:

- модель хранилища данных;
- модель клиент-сервер;
- трехуровневая модель;
- модель абстрактной машины.

В модели хранилища данных (рисунок 3.2, а) подсистемы разделяют данные, находящиеся в общей памяти. Как правило, данные образуют БД. Предусматривается система управления этой базой.

Модель клиент-сервер используется для распределенных систем, где данные распределены по серверам (рисунок 3.2, б). Для передачи данных применяют сетевой протокол, например TCP/IP.

Трехуровневая модель является развитием модели клиент-сервер (рисунок 3.2, в). Уровень графического интерфейса пользователя запускается на машине клиента. Бизнес-логику образуют модули, осуществляющие функциональные обязанности системы. Этот уровень запускается на сервере приложения. Реляционная СУБД хранит данные, требуемые уровню бизнес-логики. Этот

уровень запускается на втором сервере — сервере базы данных. Преимущества трехуровневой модели:

- упрощается модификация уровня, которая не влияет на другие уровни;
- отделение прикладных функций от функций управления БД упрощает оптимизацию всей системы.

Модель абстрактной машины отображает многослойную систему (рисунок 3.2, г). Каждый текущий слой реализуется с использованием средств, обеспечиваемых слоем-фундаментом.

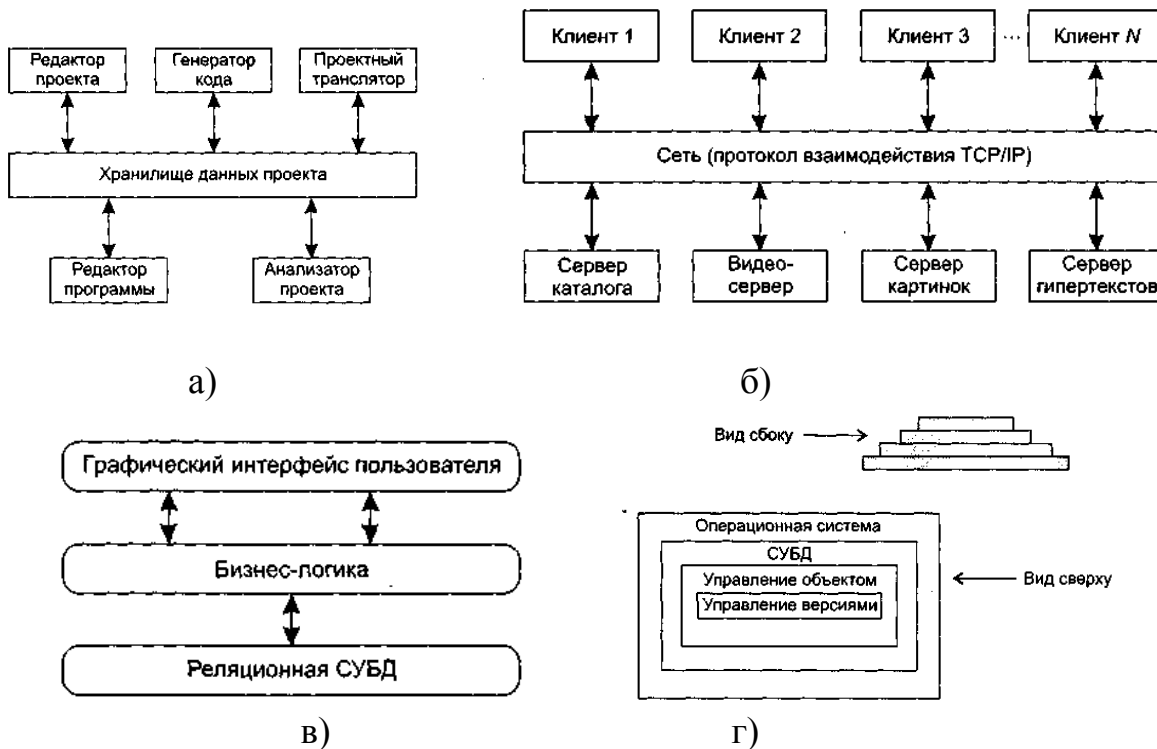


Рисунок 3.2 – Модели архитектуры:
 а) хранилища данных; б) клиент-сервер;
 в) трехуровневая модель; г) абстрактной машины

3.1.3. Моделирование управления

В моделях управления на уровне архитектуры проектируется поток управления между подсистемами. Модель управления дополняет структурные модели.

Известны два типа моделей управления:

- модель централизованного управления;
- модель событийного управления.

В модели централизованного управления одна подсистема выделяется как системный контроллер. Ее обязанности — руководить работой других подсистем. Различают две разновидности моделей централизованного управления: *модель вызов-возврат* (рисунок 3.3, а) и *Модель менеджера* (рисунок 3.3, в), которая используется в системах параллельной обработки.

Управление от первой подсистемы может перейти к другой подсистеме, однако потом обязательно возвращается к первой.

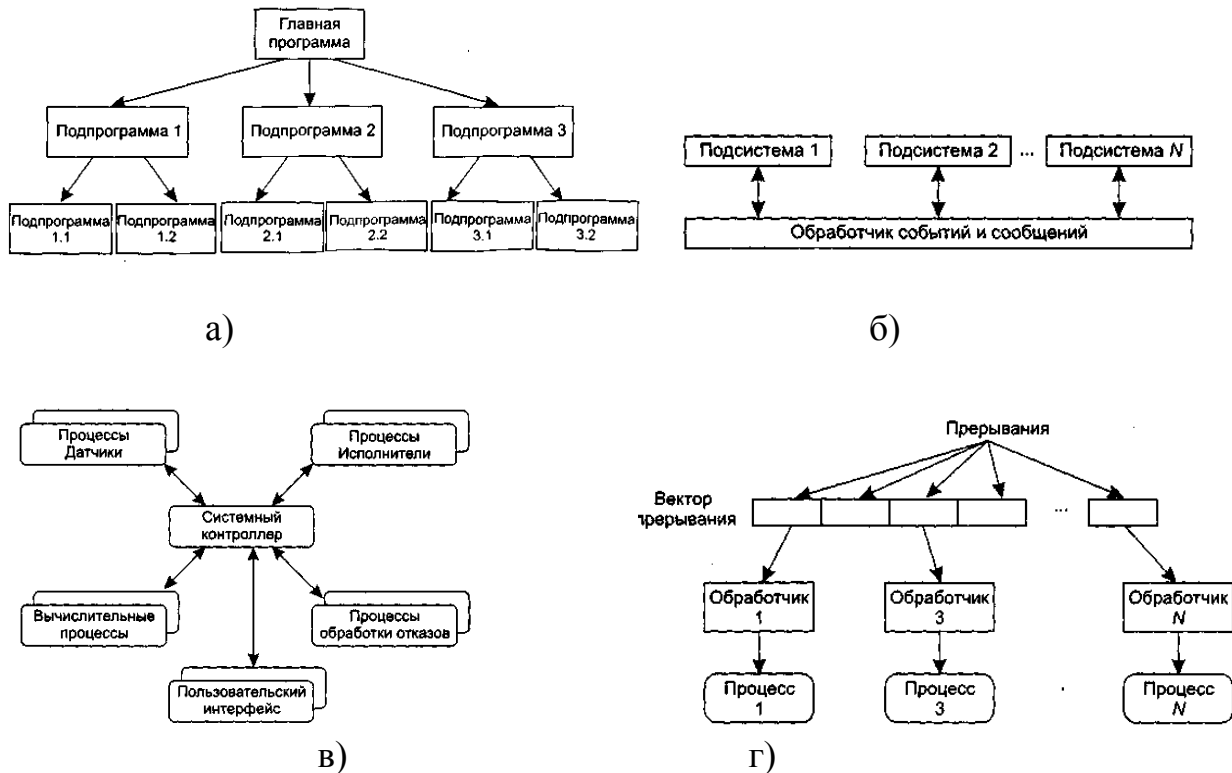


Рисунок 3.3 – Модель управления:

- а) вызов-возврат; б) широковещательная модель;
в) модель менеджера; г) управляемая событиями

В модели событийного управления системой управляют внешние события. Здесь вместо одной подсистемы, ответственной за управление, на внешние события может отвечать любая подсистема. События, на которые реагирует система, могут происходить либо в других подсистемах, либо во внешнем окружении.

Используются две разновидности модели событийного управления: широковещательная модель и модель, управляемая прерываниями.

В *широковещательной модели* (рисунок 3.3, б) каждая подсистема уведомляет обработчика о своем интересе к конкретным событиям. Когда событие происходит, обработчик пересылает его подсистеме, которая может обработать это событие. Функции управления в обработчик не встраиваются.

В *модели, управляемой прерываниями* (рисунок 3.3, г), все прерывания разбиты на группы — типы, которые образуют вектор прерываний. Для каждого типа прерывания есть свой обработчик. Каждый обработчик реагирует на свой тип прерывания и запускает свой процесс.

3.1.4. Модульная декомпозиция

После этапа разработки системной структуры в процессе проектирования следует этап декомпозиции систем на модули. Модуль — фрагмент программного текста, являющийся строительным блоком для физической структуры системы. Как правило, модуль состоит из интерфейсной части и части-реализации. Модульность — свойство системы, которая может подвергаться декомпозиции на ряд внутренне связанных и слабо зависящих

Известны два типа моделей модульной декомпозиции:

- модель объектов;
- модель потоков данных.

В объектно-ориентированной модели модули представляют собой объекты как совокупности данных, к которым привязаны некоторые процедуры их обработки (методы). В модели потоков данных модули выполняют функциональные преобразования.

Объектно-ориентированная модель структурирует систему в виде совокупности слабо связанных объектов с четко определенными интерфейсами. Объекты вызывают сервисы, предоставляемые другими объектами. Основными принципами модели являются: абстрагирование, инкапсуляция, модульность, иерархичность, типизация, параллелизм и сохраняемость. Для представления объектно-ориентированной модели системы используют диаграмму классов.

В модели потоков данных данные поступают на вход системы, проходят через последовательность преобразований и достигают выхода системы. Для представления модели потоков данных используют диаграммы потоков данных (DFD), описывающих асинхронный процесс преобразования информации от ее ввода в систему до выдачи пользователю.

3.2. Задание на лабораторную работу

1. Изучить теоретический материал.
2. В соответствии с вариантом задания для программного изделия построить структурную схему системы.
3. Выбрать модель управления проектируемой системы.
4. Создать проект в среде ООП. Проект должен содержать упрощенную реализацию классов, разработанных в первой лабораторной работе. Каждый класс должен включать: 1) Объявление переменных (свойств класса). 2) конструктор с параметрами; 3) конструктор с параметрами; 4) методы-геттеры; 5) методы-сеттеры; 6) метод toString(), выдающий строку описания объекта; Функциональность разрабатываемой системы (методы класса) не реализовывать, за исключением метода main для вывода информации о созданных классах и методов, используемых для создания объектов (при необходимости). Реализовать связи между классами. путем создания объектов внутри классов. 5. Построить диаграмму классов в используемой среде
5. Оформить отчет.

3.3. Содержание отчета

Постановка задачи, структурная схема системы, модель управления, текст программы, диаграмма классов, выводы.

3.4. Список контрольных вопросов

1. Какие модели относятся к структурным?
2. Достоинства и недостатки объектной модели.
3. Особенности построения диаграммы классов.
4. Перечислите архитектурные модели.
5. Перечислите модели централизованного управления
6. Перечислите модели событийного управления
7. Что такое модульная декомпозиция?
8. Каковы особенности объектно-ориентированной модели?

Лабораторная РАБОТА № 4

ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

Цель работы: Изучить основные принципы проектирования интерфейса, получить практические навыки реализации и описания основных этапов проектирования интерфейсов.

4.1. Теоретические сведения

4.1.1. Графический интерфейс

Пользовательский интерфейс (ПИ) – система правил и средств, регламентирующая и обеспечивающая взаимодействие программы с пользователем. Графический интерфейс пользователя (GraphicsUserInterface - GUI) является обязательным компонентом большинства современных программных продуктов, ориентированных на работу конечного пользователя. К графическому интерфейсу пользователя предъявляются высокие требования как с чисто инженерной, так и с художественной стороны разработки, при его разработке ориентируются на возможности человека.

Наиболее часто графический интерфейс реализуется в интерактивном режиме работы пользователя для программных продуктов, функционирующих в среде Windows, и строится в виде системы спускающихся меню с использованием в качестве средства манипуляции мыши и клавиатуры. Работа пользователя осуществляется с экранными формами, содержащими объекты управления, панели инструментов с пиктограммами режимов и команд обработки.

Таблица 4.1 – Принципы проектирования интерфейсов пользователя

Принципы	Описание
Учет знаний пользователя	Использование терминов и понятий, взятые из опыта пользователей
Согласованность	Однотипные (но различные) операции должны выполняться одним и тем же способом
Минимум неожиданностей	Поведение системы должно быть прогнозируемым
Способность к восстановлению	Интерфейс должен иметь средства, позволяющие восстановить данные после ошибочных действий
Руководство пользователя	Поддержка средств контекстно-зависимой справки
Учет разнородности пользователя	Средства взаимодействия с пользователями разной квалификации

Интерфейс пользователя компьютерного приложения включает:

- средства отображения информации, отображаемую информацию, форматы и коды;
- командные режимы, язык «пользователь — интерфейс»;
- устройства и технологии ввода данных;

- диалоги, взаимодействие и транзакции между пользователем и компьютером, обратную связь с пользователем;
- поддержку принятия решений в конкретной предметной области;
- порядок использования программы и документацию на неё.

Пользовательский интерфейс часто понимают только как внешний вид программы. Однако, на деле пользователь воспринимает через него всю программу в целом, а значит, такое понимание является слишком узким. В действительности ПИ объединяет в себе все элементы и компоненты программы, которые способны оказывать влияние на взаимодействие пользователя с программным обеспечением (ПО), а не только экран, который видит пользователь.

В процессе разработки интерфейса можно выделить три основных этапа, а именно первоначальное проектирование, создание прототипа и тестирование/модификация прототипа.

4.1.2. Первоначальное проектирование интерфейса пользователя

Собственно проектирование состоит из следующих этапов:

1. Определение необходимой функциональности системы.
2. Создание пользовательских сценариев.
3. Проектирование общей структуры.
4. Конструирование отдельных блоков.
5. Сборка и начальная проверка полной схемы системы.

На практике проектирование пользовательского интерфейса начинается с выявления требуемой функциональности создаваемого приложения, а также изучения особенностей его потенциальных пользователей. Описания главных категорий пользователей называют профилями. Одна из таких категорий может быть принята за основной профиль. Например, потенциальными пользователями программного обеспечения информационной системы являются менеджеры по направлению товара, торговые представители.

После выделения одного или нескольких основных профилей пользователей и определения целей и задач, стоящих перед ними, переходят к следующему этапу проектирования. Этот этап связан с составлением пользовательских сценариев. Как правило, начинают с персонификации профилей (присваивания каждому профилю условного имени), затем формулируют сценарии. Сценарий - это описание действий, выполняемых пользователем в рамках решения конкретной задачи на пути достижения его цели. Примером могут служить приведенные ниже сценарии действий пользователей.

1) Менеджер общается с клиентами по телефону. По просьбе клиента он предварительно просматривает данные о запрошенных им товарах, затем приступает к формированию нового заказа. Она вводит данные клиента, после чего выбирает указанный(ые) товар(ы) из списка и добавляет его (их) в заказ и сохраняет заказ.

2) Менеджер, общаясь с клиентами по телефону, создает новые заказы. При формировании нового заказа, она выбирает клиента из списка, если его там нет, то вводит клиента в список клиентов. Затем добавляет в заказ необходимые товары, используя сложный поиск. Она распечатывает информацию заказа, после этого она сохраняет ее.

3) Менеджер просматривает данные о товаре, выполняет поиск заказа по товару, редактирует в заказе информацию по некоторым товарам, сохраняет информацию и распечатывает ее.

После формирования сценариев становится известным перечень отдельных функций. Для профиля менеджер можно сформировать следующий перечень функций необходимых в приложении:

- создать новый заказ (1);
- сложный поиск заказа (2);
- редактирование заказа (3);
- добавление клиента из списка клиентов в заказ (4);
- ввод/редактирование клиента в списке клиентов (5);
- выбор товара из списка товаров (6);
- сложный поиск товаров в списке товаров (7);
- просмотр подробных данных о товаре (8);
- добавление товара из списка товаров в заказ (9);
- сохранение заказа (10);
- печать заказа (11);
- формирование счета (12).

В приложении функция представлена функциональным блоком с соответствующей экранной формой (формами). Возможно, что несколько функций объединяются в один функциональный блок. Таким образом, на этом этапе устанавливается необходимое число экранных форм. Важно определить навигационные взаимосвязи функциональных блоков. На практике установлено, что наиболее подходящее число связей для одного блока равно трем. Если последовательность выполнения функций жестко определена, их экранные формы вызываются последовательно одна из другой. Навигационные связи между отдельными функциональными блоками отображаются на схеме навигационной системы.

Отдельные функциональные блоки соответствуют работе пользователей с информацией:

- по заказам (функции 1,2,3,4,9,10,11,12): по общему журналу заказов и по конкретному (текущему) заказу;
- по клиентам (функции 4,5): по списку клиентов в целом и по конкретному клиенту;
- по товарам (функции 6,7,8,9): по списку товаров и по данному товару.

В приложении три функциональных блока и шесть экранных форм:

Журнал заказов; Текущий заказ; Список клиентов; Карта клиента; Список товаров; Карта товара.

Схема навигации по формам показана на рисунке рисунок 4.1

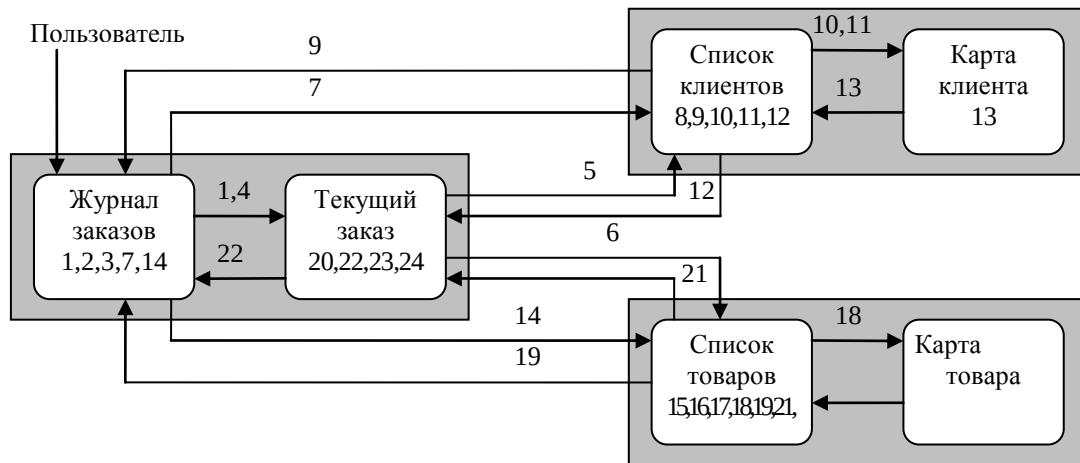


Рисунок 4.1 – Схема навигации

Цифрами на рисунке обозначены отдельные операции, выполняемые пользователями:

- 1) создать новый заказ;
- 2) задать атрибуты поиска заказа;
- 3) найти заказ по текущим атрибутам поиска;
- 4) открыть текущий заказ на редактирование;
- 5) открыть список клиентов для добавления в текущий заказ;
- 6) открыть список товаров для добавления в текущий заказ;
- 7) просмотреть список клиентов;
- 8) выбрать клиента из списка клиентов;
- 9) добавить атрибуты текущего клиента к поиску заказа;
- 10) ввести данные нового клиента в текущий заказ;
- 11) редактировать данные текущего клиента в списке клиентов;
- 12) добавить текущего клиента в текущий заказ;
- 13) сохранить данные о текущем клиенте;
- 14) просмотреть список товаров;
- 15) задать атрибуты поиска товаров;
- 16) найти товар по текущим атрибутам;
- 17) выбрать товар из списка товаров;
- 18) просмотреть подробные данные текущего товара;
- 19) добавить атрибуты текущего товара к поиску заказа;
- 20) редактировать данные по текущему товару в текущем заказе;
- 21) добавить данные текущего товара в текущий заказ;
- 22) сохранить текущий заказ;
- 23) распечатать информацию по текущему заказу;
- 24) сформировать счет по текущему заказу.

Основным навигационным элементом приложения является главное меню. Формирование меню начинается с анализа функций приложения. Для этого в рамках каждой из них выделяют отдельные элементы: операции, выполняемые пользователями, и объекты, над которыми осуществляются эти

операции. Выделенные элементы группируются в общие разделы главного меню. Группировка отдельных элементов происходит в соответствии с представлениями об их логической связи.

После разработки моделей – прототипов экранных форм для каждого из функциональных блоков создается общая схема интерфейса. Для ее изображения можно использовать Visio. Самой важной целью этого этапа является создание плана обработки системой исключительных ситуаций интерфейса. Необходимо определить, что делать системе, если пользователь вызвал команду, которую для этого конкретного объекта выполнить невозможно (например, пользователь пытается послать письмо человеку, почтовый адрес которого системе неизвестен).

4.1.3. Количественная оценка интерфейса

Для анализа качества интерфейсов используется множество количественных и эвристических методов. Одним из лучших подходов к количественному анализу моделей интерфейсов является классическая модель GOMS (goals, objects, methods and selection rules).

Метод, использующий модель GOMS, основан на разбиении всех действий пользователя на отдельные составляющие. Для каждой из них с помощью тщательных лабораторных исследований получен набор временных интервалов, необходимых для ее выполнения. В таблице 4.2 приведена номенклатура элементарных действий и соответствующие временные интервалы.

Таблица 4.2 – Модель GOMS

Нажатие клавиши клавиатуры, включая клавиши Alt, Ctrl, Shift	0,28 с	К
Нажатие клавиши мыши	0,1 с	М
Указание – перемещение курсора мыши, чтобы указать какую-либо позицию на экране монитора	1,1 с	П
Перемещение – перенос руки пользователя с клавиатуры на мышь или обратно	0,4 с	В
Ментальная подготовка – мысленный выбор пользователем своего следующего элементарного действия	1,2	Д
Ответ – реакция системы на элементарное действие пользователя	-	Р

Этот метод пригоден для проведения сравнительной оценки между двумя моделями интерфейса по уровню эффективности их использования. Расчет времени, необходимого для выполнения некоторого действия начинают с разбиения его на элементарные действия, которые соответствуют номенклатуре приведенной в таблице 4.2. Проще всего выделить движения К, М, П, В. Проблему составляет определение моментов, когда пользователь должен остановиться, чтобы выполнить бессознательную ментальную операцию.

Таблица 4.3 – Основные правила преобразования операций

Правило 0 Начальная расстановка операторов Д	Операторы Д следует устанавливать перед всеми операторами К и М (нажатие клавиши), также перед всеми операторами П, предназначенными для выбора команд. Но перед операторами П, предназначенными для указания на аргументы этих команд, ставить оператор Д не следует.
Правило 1 Удаление ожидаемых операторов Д	Если оператор, следующий за оператором Д, является полностью ожидаемым, предшествующего Д, то этот оператор Д может быть удален. Если пользователь перемещает мышь с намерением нажать на ее кнопку по достижении цели движения, то следует удалить оператор Д, установленный по правилу 0. Так последовательность действий П Д К преобразуется в П К.
Правило 2 Удаление операторов Д внутри когнитивных единиц	Если строка Д К Д К Д К... принадлежит когнитивной единице, то следует удалить все операторы Д, кроме первого. Когнитивной единицей является непрерывная последовательность вводимых символов, которые образуют название команды или аргумент. Например <i>У, переместить, 4564.23</i> – это когнитивные единицы.
Правило 3 Удаление операторов Д перед последовательными разделителями	Если оператор К означает лишний разделитель, стоящий в конце когнитивной единицы (например, разделитель команды, следующий сразу за разделителем аргумента этой команды), то следует удалить оператор Д, стоящий перед ним.
Правило 4 Удаление операторов Д, которые являются прерывателями команд	Если оператор К является разделителем, стоящим после постоянной строки (например, название команды, которая каждый раз вводится в неизменном виде), то следует удалить оператор Д, стоящий перед ним. Но если оператор К является разделителем строки аргументов или любой другой изменяемой строки, то оператор Д следует сохранить перед ним.
Правило 5 Удаление перекрывающих операторов Д	Любую часть оператора Д, которая перекрывает оператор Р, означающий задержку, связанную с ожиданием ответа компьютера, учитывать не следует.

В этих правилах под строкой понимается некоторая последовательность символов.

Разделителем считается символ, которым обозначено начало или конец значимого фрагмента текста, такого как, например, слово естественного языка или телефонный номер. Так, пробел является разделителем для большинства слов, а точка используется в конце предложений для разделения. В качестве разделителей могут выступать скобки для ограничений пояснений или замечаний и т.д.

Если для выполнения команды требуется дополнительная информация, она называется здесь аргументом данной команды.

Выполним на примере операции, связанной с созданием нового заказа, расчет времени по правилам GOMS.

При использовании команды меню эта операция раскладывается на следующие действия:

- 1) перемещение руки к мыши, В;
- 2) указание на раздел меню Действия, П;

- 3) нажатие клавиши мыши, М;
- 4) указание на команду Создать, П;
- 5) нажатие клавиши мыши, М.

В соответствии с правилом 0 расстановки ментальных операторов Д получим следующую последовательность операторов

В Д П Д М Д П Д М

В соответствии с правилом 1 следует удалить ожидаемые операторы Д

В Д П М Д П М

Остальные правила в этом примере не используются. Складывая соответствующие значения операторов получим общее время

$0,4+1,2+1,1+0,1+1,2+1,1+0,1=5,2$ с.

Очевидно, что это время велико. Необходимо предусмотреть другие варианты выполнения команды.

При использовании командной кнопки операция по созданию нового заказа раскладывается на действия:

- 1) перемещение руки к мыши, В;
- 2) указание на командную кнопку Создать, П;
- 3) нажатие клавиши мыши, М.

При расстановке операторов Д здесь также используются правила 0 и 1. Результирующая последовательность выглядит как

В Д П М

Общее время на выполнение операции составляет

$0,4+1,2+1,1+0,1=2,8$ с.

Это почти в 2 раза быстрее, чем предыдущий вариант выполнения команды.

4.1.4. Создание прототипа интерфейса и его тестирование

Начальный этап тестирования связан с разработкой прототипа интерфейса. На этом этапе проектировщик использует имеющиеся результаты проектирования: общую схему приложения, планы отдельных экранных форм. Эти результаты сводятся воедино в общую схему, которую необходимо проверить по сформулированным ранее сценариям. Целью такой проверки является выявление несоответствия последовательности действий, описанной в сценарии, и структуры полной схемы. Обнаруженные несоответствия должны быть устранены за счет модификации экранных форм и/или корректировки общей схемы приложения.

Имея полную схему приложения, приступают к формированию электронного прототипа. Прототип должен в первую очередь отображать функциональность интерфейса результирующей системы, поэтому его первые версии делают достаточно «примитивными». Последующие версии прототипа могут быть эстетически более совершенными.

Электронный прототип пользовательского интерфейса представляет собой демонстрационный ролик, выполненный в одной из презентационных программ – MS PowerPoint, MS Visio и др. Каждая экранная форма соответствует

отдельному слайду, результат нажатия кнопок имитируется переходами между слайдами. Переходы реализуются с помощью организации гиперссылок. Электронная версия прототипа пользовательского интерфейса позволяет тестировать довольно сложные взаимодействия человека с приложением.

Успех тестирования зависит от правильности и корректности постановки задачи тестирования. Тестирование может быть направлено на подтверждение:

- производительности действий при использовании продукта. Оценивается по длительности выполнения задач (тестовых заданий) пользователем;

- полезности продукта. Полезный продукт позволяет увеличить число пользователей, способных успешно выполнить задание;

- простоты обучения. Оценивается по времени тренинга, необходимого для достижения пользователем определенного уровня владения продуктом;

- субъективной оценки пользователей. Пользователи оценивают свое отношение к продукту по десятибалльной шкале. Продукт можно считать успешным, если определенная часть пользователей оценила его на 8 и выше баллов.

Разработка пользовательского интерфейса приложения представляется собой итеративный процесс.

4.2. Задание на лабораторную работу

1. Для главного профиля разрабатываемого программного продукта определить функциональные блоки приложения, составить схему навигации.

2. Разработать прототипы экранных форм в среде разработки приложения. Добавить экранные формы в проект, созданный в предыдущей работе. Обеспечить переходы между формами.

3. Для одной из форм провести количественную оценку элементов интерфейса, по результатам количественной оценки по возможности внести необходимые усовершенствования в модели форм.

4. Имитировать ввод с клавиатуры для прототипа в виде презентации невозможно. От слайда с пустой формой можно организовать переход к слайду со списком, либо с нужными значениями. После указания нужной записи организуют переход к слайду с формой с заполненными данными.

5. Оформить отчет.

4.3. Содержание отчета

Постановка задачи, описание функциональных блоков, навигационная схема системы, внешний вид разработанных экранных форм, расчеты количественного анализа, выполненного для фрагментов интерфейса; тестовые задания, сформированные в соответствии с пользовательскими сценариями; задачи тестирования и оценки качества интерфейса; выводы.

4.4. Список контрольных вопросов

1. Основные принципы проектирования интерфейса.
2. Понятие сценария.
3. Из каких элементов состоит схема навигации?
4. На чем основана количественная оценка на основании модели GOMS?
5. Что такое ментальная подготовка в модели GOMS?
6. Для чего нужны прототипы?
7. Что изображают на схеме навигации?
8. В чем заключается тестирование интерфейса?

Лабораторная работа № 5

УПРАВЛЕНИЕ ПРОЕКТОМ

Цель работы: познакомиться с теоретическими основами управления проектами и получить практические навыки по созданию проекта

5.1. Этапы разработки программного обеспечения

Анализ требований к проекту

На этом этапе формулируются цели и задачи проекта, выделяются базовые сущности и взаимосвязи между ними. При анализе требований определяются сроки и стоимость разработки ПО, формируется и подписывается ТЗ на разработку программного обеспечения.

Проектирование

На основе предыдущего этапа проводится проектирование системы. Эта методология проектирования соединяет в себе объектную декомпозицию, приемы представления физической, логической, а также динамической и статической моделей системы.

Во время проектирования разрабатываются проектные решения по выбору платформы, где будет функционировать система языка или языков реализации, назначаются требования к пользовательскому интерфейсу, определяется наиболее подходящая СУБД. Разрабатывается функциональная спецификация ПО: выбирается архитектура системы, оговариваются требования к аппаратному обеспечению, определяется набор орг. мероприятий, которые необходимы для внедрения ПО, а также перечень документов, регламентирующих его использование.

Реализация

Данный этап разработки программного обеспечения организован в соответствии с моделями эволюционного типа жизненного цикла ПО. При разработке применяются экспериментирование и анализ, строятся прототипы, как целой системы, так и ее частей.

Тестирование продукта

Тестирование тесно связано с такими этапами разработки программного обеспечения как проектирование и реализация. Результатом тестирования является устранение всех недостатков системы и заключение о ее качестве.

Внедрение и поддержка

Внедрения системы обычно предусматривает следующие шаги: установка системы, обучение пользователей, эксплуатация.

К любой разработке прилагается полный пакет документации, который включает в себя описание системы, руководства пользователей и алгоритмы работы.

5.2. Исполнители проекта разработки ПО

Состав команды определяется опытом и уровнем коллектива, особенностями проекта, применяемыми технологиями и уровнем этих технологий.

«Классический» вариант состава команды включает следующие позиции

Менеджер проекта - главное действующее лицо, обладающее знаниями и навыками, необходимыми для успешного управления проектом. Его основные функции: подбор и управление кадрами, подготовка и исполнение плана проекта, руководство командой, обеспечение связи между подразделениями, обеспечение готовности продукта

Проектировщик - это функция проектирования архитектуры высокого уровня и контроля ее выполнения. В небольших командах функция распределяется между менеджером и разработчиками. Основными функциями проектирования являются: анализ требований, разработка архитектуры и основных интерфейсов, участие в планировании проекта, контроль выполнения проекта

Разработчик – роль, ответственная за непосредственное создание конечного продукта. Помимо собственно программирования (кодирования) в его функции входит: контроль архитектурных и технических спецификаций продукта, подбор технологических инструментов и стандартов, диагностика и разрешение всех технических проблем, мониторинг состояния продукта, подбор инструментов разработки, метрик и стандартов. Контроль их использования.

Тестировщик – роль, ответственная за удовлетворение требований к продукту (функциональных и нефункциональных). В функции тестировщика входит: составление плана тестирования, контроль выполнения плана, разработка тестов, автоматизация тестирования, выбор инструментов, метрик, стандартов для организации процесса тестирования.

Инженер по качеству. Составление плана качества, описание и оценка процессов, повышение качества процессов

Технический писатель или разработчик пользовательской (и иной) документации как части программного продукта. Функциями технического писателя являются: разработка плана документирования, выбор и разработка стандартов и шаблонов подготовки документов, выбор средств автоматизации документирования, разработка документации, тестирования документации

Технолог разработки ПО обеспечивает выполнение следующих задач: поддержка модели ЖЦ, создание и сопровождение среды сборки продукта, создание и сопровождение процедуры установки с тем, чтобы каждая сборка устанавливалась автоматически с учетом версии и конфигураций сред.

5.3. Основные понятия управления проектами

Проект - это некоторая работа, у которой есть начало и конец (время), заданные результаты (масштаб), установленный уровень качества (качество) и стоимость

Четыре параметра проекта – *время, масштаб, качество и стоимость* - связаны между собой, при этом стоимость проекта является функцией от времени, качества и масштаба. Одна из задач управления проектом заключается в эффективном распределении ресурсов для выполнения проекта так, как было задумано в срок, с установленным качеством, не выходя за рамки сметы.

Управление проектами — это процесс планирования, организации и управления задачами и ресурсами с целью достижения определенной цели, обычно при наличии ограничений по времени, ресурсам или затратам. План проекта может быть простым, состоящим из списка задач со временем их начала и завершения или сложным, содержащим взаимосвязанные задачи и ресурсы с бюджетом в сотни тысяч гривен.

Проект обладает определенными свойствами.

1. Проект всегда имеет четко определенную цель, которая выражается в получении некоторого результата.

2. Проект имеет четко очерченное начало, которое совпадает с началом первой работы, направленной на достижение поставленной цели. Начало может задаваться директивно, либо рассчитываться в результате составления плана работ по проекту.

3. Проект имеет четко очерченный конец, который совпадает с концом последней работы, направленной на получение заданного результата. Как и начало, конец проекта может задаваться директивно, или рассчитываться при составлении плана работ.

4. Проект исполняется командой, в состав которой входит руководитель проекта, менеджеры, исполнители.

5. Проект имеет бюджет. Стоимость проекта складывается из стоимости израсходованных материальных ресурсов, затрат по оплате труда реализующей его команды и прочих расходов, связанных с особенностями конкретных видов работ.

6. Проект имеет ограничения трех видов.

- Ограничения по бюджету устанавливают предельную стоимость всего проекта или отдельных видов работ.

- Ограничения по времени задают предельные сроки окончания либо всего проекта, либо некоторых работ. Например, тестовые испытания должны проводиться в присутствии представителя заказчика, который будет присутствовать в заданный период времени.

- Ограничения по ресурсам определяются ограниченным составом команды или графиками поступления материальных ресурсов.

Управляемыми параметрами проекта являются:

1. объемы и виды работ;
2. стоимость, издержки, расходы по проекту;

3. временные параметры, включающие сроки, продолжительности и резервы выполнения работ и этапов проекта, а также взаимосвязи между работами;

4. ресурсы, требуемые для осуществления проекта, в том числе человеческие, финансовые, материально-технические, а также ограничения по ресурсам;

5. качество проектных решений, ресурсов, компонентов проекта и прочее.

Сетевое планирование и управление состоит из структурного и календарного планирования и оперативного управления.

Структурное планирование заключается в разбиении проекта на этапы и работы, оценки их длительности, определении последовательности их выполнения. Результатом структурного планирования является *сетевой график работ*, который используется для оптимизации проекта по длительности.

Календарное планирование заключается в составлении временной диаграммы *работ* и распределении между работами трудовых ресурсов (исполнителей). Результатом календарного планирования является *диаграмма Ганта*, графически отображающая периоды выполнения *работ* на оси времени. На этом этапе может выполняться *оптимизация* ресурсов и бюджета проекта.

Оперативное управление состоит в регулярном сопоставлении фактического *графика работ* с плановым. Результатом серьезных отклонений является принятие решений об изменении первоначального структурного или календарного плана.

5.3.1. Структурное планирование

Структурное планирование включает в себя несколько этапов:

1. разбиение проекта на совокупность отдельных работ, выполнение которых необходимо для реализации проекта;

2. построение сетевого графика, описывающего последовательность выполнения работ;

3. оценка временных характеристик работ и анализ сетевого графика.

Основную роль на этапе структурного планирования играет сетевой график.

Сетевой график – это ориентированный граф, в котором вершинами обозначены работы проекта, а дугами – временные взаимосвязи работ.

Сетевой график должен удовлетворять следующим свойствам.

1. Каждой работе соответствует одна и только одна вершина. Ни одна работа не может быть представлена на сетевом графике дважды. Однако любую работу можно разбить на несколько отдельных работ, каждой из которых будет соответствовать отдельная вершина графика.

2. Ни одна работа не может быть начата до того, как закончатся все непосредственно предшествующие ей работы. То есть если в некоторую

вершину входят дуги, то работа может начаться только после окончания всех работ, из которых выходят эти дуги.

3. Ни одна работа, которая непосредственно следует за некоторой работой, не может начаться до момента ее окончания. Другими словами, если из работы выходит несколько дуг, то ни одна из работ, в которые входят эти дуги, не может начаться до окончания этой работы.

4. Начало и конец проекта обозначены работами с нулевой продолжительностью. Такие работы называются **вехами** и обозначают начало или конец наиболее важных этапов проекта.

Пример. Проект "Разработка программного комплекса". состоит из работ, характеристики которых приведены в Таблице 1

Таблица 5.1 – Характеристики работ проекта

Номер работы	Название работы	Длительность
1	Начало реализации проекта	0
2	Постановка задачи	10
3	Разработка интерфейса	5
4	Разработка модулей обработки данных	7
5	Разработка структуры базы данных	6
6	Заполнение базы данных	8
7	Отладка программного комплекса	5
8	Тестирование и исправление ошибок	10
9	Составление программной документации	5
10	Завершение проекта	0

Сетевой график для данного проекта изображен на рисунке 5.1. На нем вершины, соответствующие обычным работам, обведены тонкой линией, а толстой линией обведены *вехи* проекта.

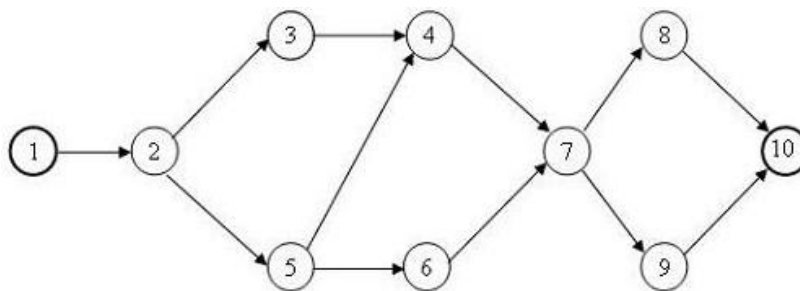


Рисунок 5.1 – Сетевой график проекта

Сетевой график позволяет по заданным значениям длительностей работ найти критические работы проекта и его критический путь.

Критической называется такая работа, для которой задержка ее начала приведет к задержке срока окончания проекта в целом. Такие работы не имеют запаса времени. Некритические работы имеют некоторый запас времени, и в пределах этого запаса их начало может быть задержано.

Критический путь – это путь от начальной к конечной вершине сетевого графика, проходящий только через критические работы. Суммарная длительность работ критического пути определяет минимальное время реализации проекта.

Для критических работ резерв времени равен нулю. Поэтому усилия менеджера проекта должны быть направлены в первую очередь на обеспечение своевременного выполнения этих работ.

Для некритических работ резерв времени больше нуля, что дает менеджеру возможность маневрировать временем их начала и используемыми ими ресурсами.

5.3.2. Календарное планирование

На этапе календарного планирования разрабатывается календарный график, который называется диаграммой Ганта. Диаграмма Ганта отображает следующие параметры проекта: структуру работ, полученную на основе сетевого графика; состав используемых ресурсов и их распределение между работами; календарные даты, к которым привязываются моменты начала и завершения работ.

Предположим, что в качестве ресурсов выступают только исполнители, и они распределены между работами согласно таблице 5.2

Таблица 5.2 – Распределение работ между исполнителя

Работы	Название работы	Исполнитель
1	Начало реализации проекта	-
2	Постановка задачи	Постановщик
3	Разработка интерфейса	Программист1
4	Разработка модулей обработки данных	Программист1
5	Разработка структуры базы данных	Программист2
6	Заполнение базы данных	Программист2
7	Отладка программного комплекса	Программист1 Программист2
8	Тестирование и исправление ошибок	Программист1 Программист2 Постановщик
9	Составление программной документации	Постановщик

Календарный график (диаграмма Ганта) изображен на рисунке 5.2, где ромбиками обозначены вехи, сплошными линиями – продолжительность работ, сплошными линиями со стрелками – резерв времени работ, пунктирными линиями – связь между окончанием предшествующих и началом последующих работ.

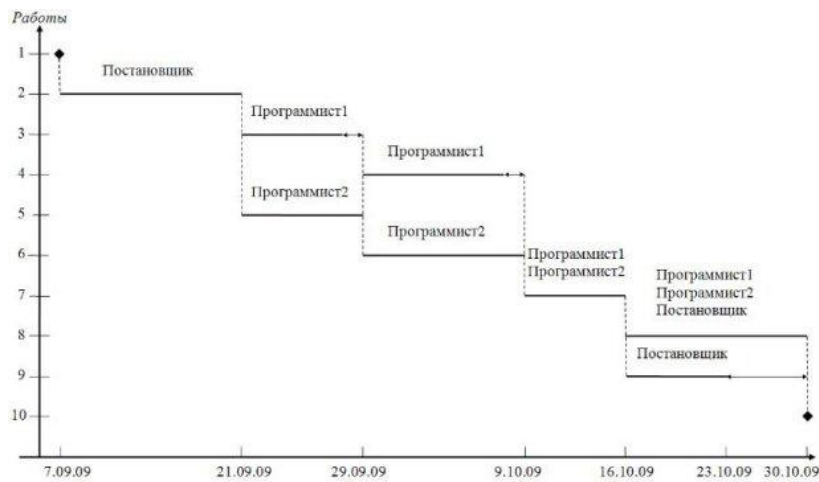


Рисунок 5.2 – Диаграмма Ганта

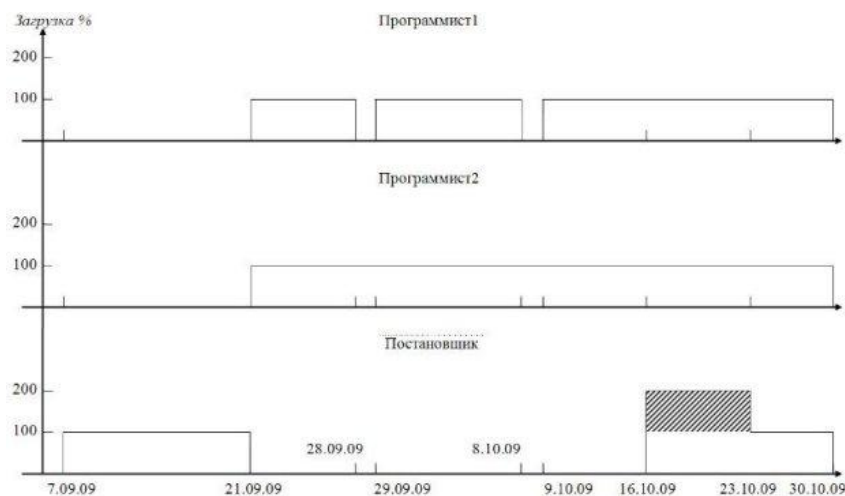


Рисунок 5.3 - График загруженности ресурсов

На основании диаграммы Ганта может быть построен график загруженности ресурсов. Этот график показывает процент загрузки конкретного трудового ресурса в ходе выполнения проекта. По оси абсцисс откладывается временной интервал проекта, а по оси ординат – суммарный процент загруженности исполнителя по всем задачам проекта, которые он выполняет в текущий момент времени.

Обычно исполнитель целиком занят решением некоторой задачи и по ее завершении переходит к следующей. Это соответствует 100 % загрузки. Однако в некоторых случаях он может быть параллельно задействован в 2 или более задачах, выделяя для их решения часть рабочего времени. Например, две задачи по 50 % каждая, то есть по половине рабочего дня на задачу. График загруженности ресурса позволяет в этом случае контролировать суммарную занятость исполнителя и выявить возможные периоды перегрузки, когда ему запланировано больше работы, чем он может выполнить в течение рабочего дня. Об этом свидетельствует суммарная загруженность более 100 %.

Пример графиков загруженности ресурсов проекта изображен на рисунке 5.3. Он построен, исходя из предположения, что каждый работник занят на 100 % выполнением запланированной ему задачи.

5.3.3. Оперативное управление

На этапе оперативного управления происходит выполнение работ по проекту и непрерывный контроль над ходом его реализации. Каким бы хорошим ни был первоначальный план, жизнь обязательно внесет в него свои коррективы. Поэтому задачами менеджера являются:

1. отслеживание фактического графика выполнения работ;
2. сравнение фактического графика с плановым;
3. принятие решений по ликвидации наметившихся отклонений от плана;
4. перепланирование проекта в случае значительных отклонений.

Первые две задачи решаются при помощи диаграммы Ганта. На ней параллельно линиям продолжительности работ наносятся линии, обозначающие процент фактического выполнения этих работ. Это позволяет легко обнаружить возникшие отклонения.

Метод ликвидации отклонения зависит от имеющихся в распоряжении *менеджера ресурсов*. Для завершения запаздывающей работы можно либо привлечь дополнительных работников (дополнительные ресурсы), либо использовать тот же состав работников в сверхурочном режиме. В обоих случаях за ликвидацию отклонения придется платить увеличением стоимости проекта (незапланированная ранее оплата дополнительных работников, ресурсов и сверхурочных работ).

Если же отклонение таково, что не может быть исправлено привлечением дополнительных и сверхурочных ресурсов, или увеличение стоимости проекта недопустимо, нужно заново **перепланировать** проект.

5.4. Порядок выполнения работы

1. Для проекта «Разработка программного обеспечения» выделить перечень работ, исполнителей, оценить длительность в днях и для каждой работы установить номера тех работ, до окончания которых она не может быть начата. Результат занести в таблицу

№	Название работы	Длительность	Предшественники	Исполнитель
1.	Анализ предметной области	3	-	Менеджер проекта
2.	Формирование и анализ требований	7	1	Аналитик
3	Разработка моделей системы	5	1,2	Программист

...

Количество этапов 10-13. Продолжить таблицу самостоятельно

2. Выполнить структурное планирование, составить сетевой график работ, по сетевому графику работ найти критические работы проекта и его критический путь. Для этого

В программе *Openproj* выполнить следующие действия:

- Создать новый проект
- Настроить календарь (Инструменты-Изменить время работы, определить праздники)
- В столбец «Название» внести названия работ (задач), при необходимости выполнить разбиение на подзадачи, задать их продолжительность. Для подзадач остановить уровни иерархии (кнопки Увеличить отступ)
- Описать зависимости между задачами, для этого выделить две задачи, нажать кнопку Связи
- Задать для задач ограничения – двойной щелчок по задаче, в окне «Информация о задаче» - Дополнительно-Тип ограничения. Важную задачу пометить как веху.

3. Выбрать конкретную дату начала проекта и составить календарный план проекта. Составить график загрузки по одному из исполнителей, диаграмму Использование задачи. Определить стоимость проекта
Для этого:

- Задать информацию о ресурсах (Вид-Ресурсы). Ресурсы делятся на материальные и трудовые. (столбец Тип). Двойной щелчок по ресурсу – задание информации (стоимость, заработная плата и т.д.)
- Назначить материальный ресурс (с указанием количества) и трудовой ресурс (с указанием процента рабочего времени) задаче: выделить задачу, Инструменты – назначить ресурс)

5.5. Контрольные вопросы

1. Что такое проект?
2. Что такое управление проектами?
3. В чем состоит процесс управления проектами?
4. Этапы управления проектами.
5. Возможности PROJ по управлению проектами.
6. С чего начинается создание нового проекта?
7. Какими способами можно создать проектный план?
8. Какие календари существуют в программе и для чего они предназначены?

Лабораторная работа № 6

РАСЧЕТ МЕТРИК ЧИДАМБЕРА - КЕМЕРЕРА

Цель работы: Оценка объектно-ориентированных программных систем путем расчета проектных метрик, ориентированных на классы.

6.1. Теоретические сведения

Метрика качества программ – система измерений качества программ. Внешние и внутренние метрики задаются на этапе формирования требований к ПО и являются предметом планирования способов достижения качества конечного программного продукта.

Внешние метрики продукта:

- надежности продукта, которые служат для определения числа дефектов;
- функциональности, с помощью которых устанавливаются наличие и правильность реализации функций в продукте;
- сопровождения, с помощью которых измеряются ресурсы продукта (скорость, память, среда);
- применимости продукта, которые способствуют определению степени доступности для изучения и использования;
- стоимости, которыми определяется стоимость созданного продукта.

Внутренние метрики продукта:

- метрики размера, необходимые для измерения продукта с помощью его внутренних характеристик (количество операций);
- метрики сложности, необходимые для определения сложности продукта (количество циклов, глубина циклов);
- метрики стиля, которые служат для определения подходов и технологий создания отдельных компонентов продукта и его документов.

При конструировании объектно-ориентированных систем значительная часть затрат приходится на создание визуальных моделей. Решение задачи оценки качества этих моделей требует введения специального метрического аппарата. Метрики Чидамбера и Кемерера ориентированы на классы, которые являются фундаментальными элементами объектно-ориентированных (ОО) систем.

Метрика 1: Взвешенные методы на класс WMC (Weighted Methods Per Class).

Допустим, что в классе C определены n методов со сложностью $C_1, C_2, \dots, C_n$. В этом случае

$$WMC = C_1 + C_2 + C_3 + \dots + C_n \quad (6.1)$$

Количество методов и их сложность являются индикатором затрат на реализацию и тестирование классов. С ростом количества методов в классе его применение становится все более специфическим, тем самым ограничивается

возможность многократного использования. По этим причинам метрика WMC должна иметь разумно низкое значение.

Очень часто применяют упрощенную версию метрики. При этом полагают $C_i = 1$, и тогда WMC - количество методов в классе.

При подсчете количества методов учитываются только методы текущего класса. Унаследованные методы игнорируются. Обоснование - унаследованные методы уже подсчитаны в тех классах, где они определялись. Наиболее важным источником информации для понимания того, что делает класс, являются его собственные операции. Если класс не может отреагировать на сообщение (например, в нем отсутствует собственный метод), тогда он пошлет сообщение родителю. Класс, имеющий максимальное количество методов среди классов одного с ним уровня, является наиболее сложным; скорее всего, он специфичен для данного приложения и содержит наибольшее количество ошибок.

Метрика 2: Высота дерева наследования DIT (Depth of Inheritance Tree)

DIT определяется как максимальная длина пути от листа до крон дерева наследования классов. Соответственно, для отдельного класса DIT, это длина максимального пути от данного класса до корневого класса в иерархии классов. По мере роста DIT вероятно, что классы нижнего уровня будут наследовать много методов. Высокая иерархия классов (большое значение DIT) приводит к большей сложности проекта, так как означает привлечение большего количества методов и классов. Вместе с тем, большое значение DIT подразумевает, что многие методы могут использоваться многократно.

Метрика 3: Количество детей NOC (Number of children)

Подклассы, которые непосредственно подчинены суперклассу, называются его детьми. Значение NOC равно количеству детей, то есть количеству непосредственных наследников класса в иерархии классов. С увеличением NOC возрастает многократность использования, так как наследование - это форма повторного использования. Однако при возрастании NOC ослабляется абстракция родительского класса. Это означает, что в действительности некоторые из детей уже не являются членами родительского класса и могут быть неправильно использованы.

Метрика 4: Сцепление между классами объектов CBO (Coupling between object classes)

CBO - это количество сотрудничеств, предусмотренных для класса, то есть количество классов, с которыми он соединен. Соединение означает, что методы данного класса используют методы или экземплярные переменные другого класса. Другое определение метрики имеет следующий вид: CBO равно количеству сцеплений класса; сцепление образует вызов метода или свойства в другом классе. Данная метрика характеризует статическую составляющую внешних связей классов. С ростом CBO многократность использования класса уменьшается и наоборот, чем больше независимость класса, тем легче его повторно использовать в другом приложении. Высокое значение CBO усложняет модификацию и тестирование, которое следует за выполнением

модификации. Поэтому СВО для каждого класса должно иметь разумно низкое значение.

Метрика 5: Отклик для класса RFC (Response For a Class)

Введем вспомогательное определение. Множество отклика класса RS - это множество методов, которые могут выполняться в ответ на прибытие сообщений в объект этого класса. Формула для определения RS имеет вид:

$$RS = \{M\} \cup \text{all } i \{R_i\} \quad (6.2)$$

где $\{R_i\}$ - множество методов, вызываемых методом i , $\{M\}$ - множество всех методов в классе.

Метрика RFC равна количеству методов во множестве отклика, то есть, равна мощности этого множества:

$RFC = \text{card}\{RS\}$.

Другое определение метрики: RFC - это количество методов класса плюс количество методов других классов, вызываемых из данного класса.

Метрика RFC является мерой потенциального взаимодействия данного класса с другими классами, позволяет судить о динамике поведения соответствующего объекта в системе. Данная метрика характеризует динамическую составляющую внешних связей классов. С ростом RFC увеличивается сложность класса. Наихудшая величина отклика может использоваться при определении времени тестирования.

Метрика 6: Недостаток связности в методах LCOM (Lack of Cohesion in Methods)

Каждый метод внутри класса обращается к одному или нескольким свойствам (экземплярным переменным). Метрика LCOM показывает, насколько методы не связаны друг с другом через свойства (переменные). Если все методы обращаются к одинаковым свойствам, то LCOM - 0.

Введем обозначения:

НЕ СВЯЗАНЫ - количество пар методов без общих экземплярных переменных;

СВЯЗАНЫ - количество пар методов с общими экземплярными переменными. Q

I_j - набор экземплярных переменных, используемых методом M_j .

Формула для вычисления недостатка связности в методах имеет вид:

$LCOM = \text{НЕ СВЯЗАНЫ} - \text{СВЯЗАНЫ}$, если $(\text{НЕ СВЯЗАНЫ} > \text{СВЯЗАНЫ})$;

$LCOM = 0$ в противном случае.

Можно определить метрику по-другому: LCOM - это количество пар методов, не связанных по свойствам класса, минус количество пар методов, имеющих такую связь.

Рассмотрим примеры применения метрики LCOM.

Пример 1: В классе имеются методы: M_1, M_2, M_3, M_4 . Каждый метод работает со своим набором экземплярных переменных:

$I_1 - \{a, b\}; I_2 - \{a, c\}; I_3 - \{x, y\}; I_4 - \{m, n\}$. В этом случае

НЕ СВЯЗАНЫ - $\text{card}(I_{13}, I_{14}, I_{23}, I_{24}, I_{34}) - 5$; СВЯЗАНЫ - $\text{card}(I_{12}) - 1$.

$\text{LCOM} = 5 - 1 = 4$.

Связность методов внутри класса должна быть высокой, так как это содействует инкапсуляции. Если LCOM имеет высокое значение, то методы слабо связаны друг с другом через свойства. Это увеличивает сложность, в связи с чем возрастает вероятность ошибок при проектировании.

Высокие значения LCOM означают, что класс, вероятно, надо спроектировать лучше (разбиением на два или более отдельных класса). Любое вычисление LCOM помогает определить недостатки в проектировании классов, так как эта метрика характеризует качество упаковки данных и методов в оболочку класса.

6.2. Пример использование метрик Чидамбера - Кемерера

Пример расчета метрик для структуры, показанной на рисунке 6.1, представлен в таблице 1.1.

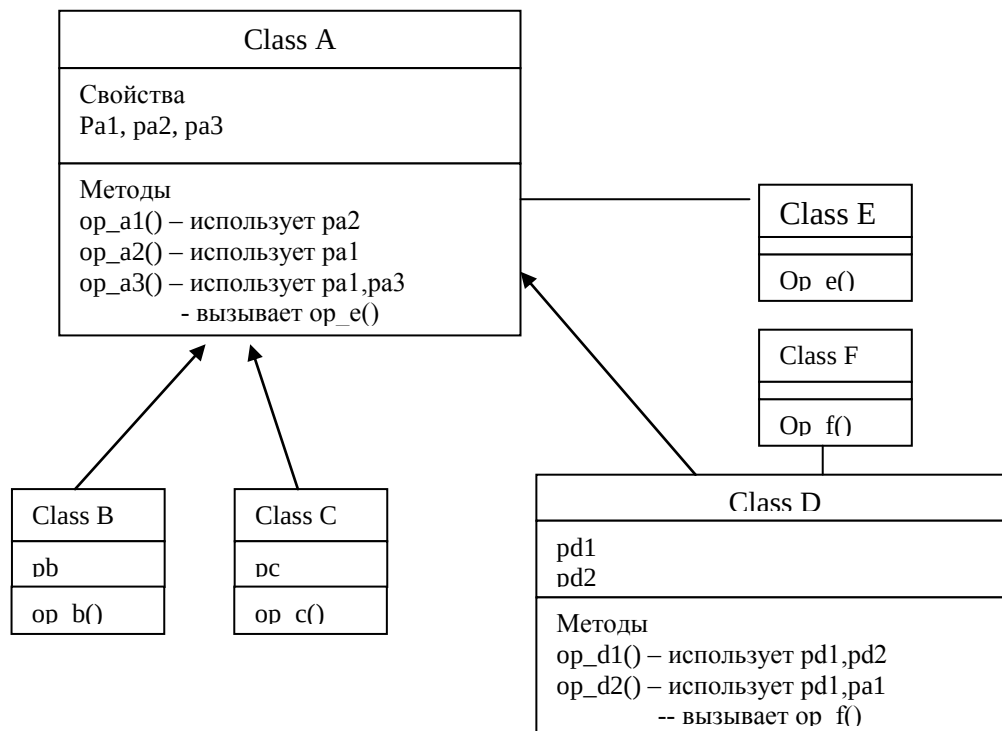


Рисунок 6.1 - Структура классов для расчета метрик Чидамбера - Кемерера

Таблица 6.1 – Пример расчета метрик Чидамбера - Кемерера

Имя класса	WMC	DIT	NOC	CBO	RFC	LCOM
Class A	3	0	3	1	4	1
Class B	1	1	0	0	1	0
Class C	1	1	0	0	1	0
Class D	2	1	0	2	3	0

Класс Class A имеет три метода (op_al(), op_a2(), op_a3()), трех детей (Class B, Class C, Class D) и является корневым классом. Поэтому метрики WMC, NOC и DIT имеют, соответственно, значения 3, 3 и 0. Метрика CBO для класса Class A равна 1, так как он использует один метод из другого класса (метод op_e() из класса Class E, он вызывается из метода op_a3()). Метрика RFC для класса Class A равна 4, так как в ответ на прибытие в этот класс сообщений возможно выполнение четырех методов (три объявлены в этом классе, а четвертый метод op_e() вызывается из op_a3()).

Поскольку в классе три метода, возможны три пары: op_al() и op_a2(), op_al(); op_a3(), op_a2() и op_a3(). Общее свойство имеет только третья пара. Количество несвязанных пар равно 2, количество связанных равно 1, LCOM=2-1=1.

Для класса Class D метрика CBO равна 2, так как здесь используются свойство pal и метод op_f() из других классов. Метрика LCOM в этом классе равна 0, поскольку методы op_d1() и op_d2() связаны по свойству pd1, а отрицательное значение запрещено.

6.2. Порядок выполнения работы

1. Изучить назначение метрик и пример их расчета.
2. Выбрать текст программы с использованием ООП. Текст должен содержать не менее 3-х классов, связанных между собой через методы или свойства.
3. Построить диаграмму связей и рассчитать метрики.
4. Результаты расчета представить в виде таблицы. Провести анализ результатов исследуемой программы.

6.3. Содержание отчета о выполнении работы

Краткие теоретические сведения; текст программы, диаграмма классов, структура классов, результаты расчетов; выводы о качестве программы;

6.4. Контрольные вопросы

1. Охарактеризуйте класс программ, для которых предложенные метрики являются наиболее эффективными.
2. Приведите недостатки предложенных метрик.
3. Какие свойства ОО-программ не учитывают метрики?
4. Что означают высокие значения метрики LCOM?
5. Какие метрики относятся к внешним?
6. Какие метрики относятся к внутренним?
7. Какова цель использования метрик?
8. Какие параметры программы определяют метрики сложности?

Лабораторная работа № 7

ОПРЕДЕЛЕНИЕ ПОКАЗАТЕЛЕЙ КАЧЕСТВА ПРОГРАММНОГО СРЕДСТВА

Цель работы: ознакомление с методами оценки качества программных средств, формирование номенклатуры показателей качества, расчет показателей качества программного средства.

7.1. Теоретические сведения

Испытание – завершающий этап разработки программных средств (ПС). На этом этапе экспериментально определяются характеристики свойств ПС, на основании которых можно сделать заключение о пригодности ПС для использования его по назначению. Совокупность свойств ПС, обуславливающих его пригодность удовлетворять определенные потребности в соответствии с назначением, называется качеством программного средства. Показатели качества отражаются в технических заданиях на разработку ПС.

Цель испытаний:

- определение комплексных показателей качества (функциональности, надежности функционирования, удобства использования, рациональности, сопровождаемости, перенесенности), характеризующих несколько свойств ПС;
- определение единичных показателей, значения которых определены в техническом задании или в описании, характеризующих одно из свойств ПС.

Для определения значений показателей качества применяются следующие методы:

1. Измерительный – основан на получении информации свойствах и характеристиках ПС с использованием измерительных, технических и программных средств. (Определение объема памяти, числа строк исходного текста, числа выполненных операторов, количества ветвей в программе).

2. Регистрационный – основан на получении информации при регистрации определенных событий (времени начала и окончания работы, число сбоев и отказов, время передачи управления от модуля к модулю).

3. Расчетный – основан на использовании теоретических и эмпирических зависимостей статистических данных, накапливаемых при испытаниях ПС. Используется для определения показателей точности, надежности, ресурсоемкости.

4. Экспертный – основан на опыте и интуиции группы экспертов-специалистов, компетентных в решении данной задачи. Применяется для определения таких показателей, как удобство применения, анализируемость, адаптируемость, документируемость, структурированность.

Для оценки значений показателей используются следующие типы шкал (множеств возможных значений показателей):

- 1) метрическая (1.1 – абсолютная, 1.2 – отношения, 1.3 – интервальная);

- 2) порядковая (ранговая), позволяющая ранжировать характеристики путем сравнения их с опорными значениями (ранжирование – разбивка значений ПК на диапазоны, соответствующие различным степеням требований);
- 3) номинальная (классификационная), характеризующая наличие или отсутствие рассматриваемого свойства у рассматриваемого ПС.

7.2. Номенклатура показателей качества

В зависимости от количества характеризующих свойств различают единичные и групповые (комплексные) показатели качества (ПК). Значения единичных показателей подлежат непосредственной оценке, значения групповых ПК определяются расчетным путем.

Номенклатура свойств и показателей качества ПС (НПК) – многоуровневая иерархическая структура. 1-й (верхний) уровень составляют группы ПК, нижний уровень – единичные ПК. Промежуточные уровни – подгруппы ПК. Группы и подгруппы свойств ПК программных средств устанавливаются ДСТУ 2850-94 и приведены в таблице 7.1. Единичные ПК устанавливаются для каждого индивидуального ПС.

Таблица 7.1 – Показатели качества программного продукта

Наименование характеризуемых свойств (группы ПК)	Определение характеризующих свойств (подгруппы ПК)
1. ФУНКЦИОНАЛЬНОСТЬ группа свойств ПС, обуславливающая способность выполнять в заданной среде определенный перечень функций в соответствии с назначением ПС.	1.1. Функциональная полнота - подгруппа свойств, характеризующаяся наличием и степенью достаточности основных функций для решения задач в соответствии с значением ПС. 1.2. Способность к взаимодействию - подгруппа свойств, характеризующаяся возможностью его взаимодействия при функционировании с другими ПС или системами. 1.3. Защищенность - подгруппа свойств, характеризующаяся его способностью предотвращать несанкционированный доступ к программам и данным. 1.4. Массовость - подгруппа свойств, характеризующаяся диапазоном и предельными значениями входных данных, которые могут быть преобразованы в правильный результат. 1.5. Согласованность - подгруппа свойств, характеризующаяся степенью соблюдения установленных стандартов, соглашений, правил и рекомендаций. 1.6. Производительность - подгруппа свойств ф, характеризующаяся количеством видов работ по обработке данных (в том числе и однотипных),

2.НАДЕЖНОСТЬ ФУНКЦИОНИРОВАНИЯ Группа свойств ПС, обуславливающая способность сохранять работоспособность и преобразовывать исходные данные в искомый результат в заданных условиях за установленный период времени	выполняемых в единицу времени функционирования. 2.1 Безотказность - родгруппа свойств надежности, характеризующаяся частотой отказов из-за ошибок или несовершенства ПС. (Отказ – проявление неработоспособности ПС). Устойчивость к аномалиям - подгруппа свойств надежности функционирования, обуславливающая способность ПС выполнять свои функции в аномальных условиях (аномальные условия – сбои и отказы технических средств, ошибки операторов, ошибки в исходных данных). 2.2 Восстанавливаемость - родгруппа свойств надежности функционирования, обуславливающая способность к восстановлению работоспособности ПС и данных после отказов. 2.3 Точность - подгруппа свойств надежности, характеризующаяся близостью результатов обработки данных к истинным, специфицированным или теоретически верным значениям. 2.4Реактивность Подгруппа свойств надежности функционирования, характеризующаяся способностью своевременно преобразовывать входные данные в искомый результат.
3.УДОБСТВО ИСПОЛЬЗОВАНИЯ Группа свойств ПС, обеспечивающая установленному или предполагаемому кругу пользователей необходимые условия для использования ПС (условия для использования должны использовать психофизические возможности человека)	3.1 Осваиваемость - подгруппа свойств удобства использования, характеризующаяся условиями, необходимыми для освоения пользователями условий, процедур и правил применения ПС. 3.2. Простота подготовки к работе - подгруппа свойств удобства использования ПС, обуславливающая легкость подготовки входных данных и запуска в работу ПС. 3.3 Удобство пользовательского интерфейса - подгруппа свойств удобства использования ПС, обуславливающая необходимые условия пользовательского интерфейса в процессе функционирования ПС 3.4 Анализируемость результатов - подгруппа свойств удобства использования ПС, обуславливающая легкость понимания результатов функционирования (конечных выходных данных) 3.5 Документированность - подгруппа свойств удобства использования ПС, характеризующаяся степенью поддержки документацией ПС процессов освоения ПС, подготовки их к работе, взаимодействия пользователя с процессом обработки данных и анализа результатов обработки.

4. РАЦИОНАЛЬНОСТЬ Группа свойств ПС, характеризующаяся степенью соответствия используемых ресурсов среды функционирования уровню качества функционирования ПС при заданных условиях применения. Ресурсы среды функционирования могут включать другие ПС, технические средства, материалы и услуги персонала.	4.1. Занятость ресурсов - подгруппа свойств рациональности ПС, характеризующаяся объемом ресурсов среды функционирования, используемых в процессе применения ПС. 4.2 Использование ресурсов - подгруппа свойств рациональности ПС, характеризующаяся степенью занятости ресурсов вычислительной системы, используемых при выполнении ПС за определенный период времени
5. СОПРОВОЖДАЕМОСТЬ Группа свойств ПС, характеризующаяся усилиями, необходимыми для выполнения определенных модификаций. (Модификация – изменение, не являющееся адаптацией к конкретным техническим средствам)	5.1. Анализируемость - подгруппа свойств сопровождаемости ПС, обуславливающая их приспособленность к диагностике недостатков и отказов, выявлению частей, которые необходимо изменить и прогнозу последствий этих изменений. 5.2. Исправляемость - подгруппа свойств сопровождаемости ПС, характеризующаяся усилиями, необходимыми для устранения в ПС выявленных недостатков 5.3. Модернизируемость - подгруппа свойств сопровождаемости ПС, обуславливающая их приспособленность к модернизации. 5.4. Тестируемость - подгруппа свойств сопровождаемости ПС, характеризующаяся усилиями, необходимыми для аттестации модифицируемого ПС.
6. ПЕРЕНОСИМОСТЬ Группа свойств ПС, обуславливающая его приспособленность для переноса из одной среды функционирования в другие	6.1. Адаптируемость - подгруппа свойств переносимости ПС, обуславливающая его способность к адаптации к функционированию в средах, отличных от тех, для которых оно разрабатывалось. 6.2. Настраиваемость - подгруппа свойств переносимости ПС, характеризующаяся усилиями, необходимыми для переноса ПС в одну из заранее предусмотренных сред функционирования

Формирование НПК для индивидуального ПС заключается в установлении перечня показателей качества и ведется сверху вниз: от групповых до единичных показателей. Свойства, входящие в группу должны взаимно дополнять друг друга. Совокупность свойств должна быть

упорядочена в виде дерева свойств. Каждый из ПК должен быть коррелирован с определенным свойством ПС. Для каждого из выбранных показателей следует установить коэффициент весомости.

Выбранная НПК должна удовлетворять следующим критериям: полнота, корреляция с качеством, однозначность определений, верифицируемость, избыточность, независимость показателей, прогрессивность.

7.3. Методы оценки уровня качества ПС

7.3.1. Дифференциальный метод

Основан на определении единичных показателей качества и определении того, достигнут ли уровень базовых значений показателей качества.

Относительное значение отдельных показателей:

$$Q_i = \frac{P_i}{P_{ib}} ; \quad (7.1)$$

$$Q_i = \frac{P_{ib}}{P_i} , \quad i=1,2,\dots,N, \quad (7.2)$$

где P_i - значение i -го показателя качества

P_{ib} - базовое значение i -го показателя

N - количество показателей качества.

Формула (1) используется в случае соблюдения условия «лучше, если значение показателя больше». В противном случае используется формула (2).

При использовании дифференциального метода уровень качества (УК) считается выше или равным уровню базовых значений, если все $Q_i > 1$. В противном случае УК ниже уровня базовых значений. Если часть $Q_i > 1$, а часть значений $Q_i < 1$, используют комплексный или смешанный методы оценки.

Базовые (эталонные) значения ПК должны соответствовать:

- значениям ПК лучших ПС из числа аналогов;
- нормативным значениям ПК;
- значениям ПК, устанавливаемым в техническом задании на разработку ПС.

7.3.2 Комплексный метод

Основан на применении обобщенного показателя, являющегося функцией от нескольких единичных (групповых) показателей и коэффициентов их весомости.

Установление коэффициентов весомости производят экспертным путем, используя при этом 5-бальную шкалу (5 – крайне важно, чтобы данный показатель имел высокое значение, 4 - важно, чтобы данный показатель имел

высокое значение, 3 – хорошо бы иметь высокое значение данного показателя, 2 – при низких значениях показателя ощутимых потерь нет).

Приведенное значение коэффициента весомости:

$$M_i = \frac{M_i}{\sum_{i=1}^N M_i}, \quad (7.3)$$

где M_i - значение i -го коэффициента весомости.

Если $\sum_{i=1}^N M_i = 1$, то коэффициенты весомости называются параметрами весомости.

Значения коэффициентов (параметров) весомости определяются при написании технического задания.

Обобщенный показатель качества выражается средним взвешенным показателем U :

$$U = \sum_{i=1}^N Q_i \cdot M_i, \quad (7.4)$$

где Q_i - относительный i -й ПК, вычисляемый по формулам (1), (2).

M_i - параметр весомости i -го показателя, входящего в обобщенный показатель;

N – число показателей, составляющих средний взвешенный показатель.

При трехуровневой иерархической структуре НПК расчет УК производится снизу вверх от единичных показателей до получения обобщенного ПК:

$$U_{yk} = \sum_{k=1}^K M_k \sum_{ik=1}^{Ik} M_{ik} \sum_{jik=1}^{Jik} Q_{jik} M_{jik}, \quad (7.5)$$

где Q_{jik} - относительное значение j -го единичного показателя i -й подгруппы k -й группы, $jik=1,2,\dots,Jik$;

M_{jik} - коэффициент весомости j -го единичного показателя i -й подгруппы k -й группы;

M_{ik} - коэффициент весомости i -й группы k -й подгруппы; $i=1,2,\dots,Ik$;

M_k - коэффициент k -й группы показателей; $k=1,2,\dots,K$.

7.3.3. Смешанный метод

Основан на совместном применении единичных и комплексных показателей, причем часть единичных показателей объединяется в группы. После объединения вычисляются значения групповых показателей и некоторых единичных по формулам (1), (2) и (3). Смешанный метод оценки УК применяется в случаях:

- 1) Совокупность единичных показателей является слишком обширной;
- 2) Обобщенный ПК в комплексном методе не позволяет сделать выводы о соответствии значений некоторых групповых показателей требуемому уровню.

7.4. Порядок выполнения работы

Провести оценку качества индивидуального программного средства. Процесс оценки качества включает этапы:

- установление цели оценки;
- формирование номенклатуры показателей качества (НПК);
- определение (уточнение) требований к качеству;
- формирование базовых (эталонных) значений показателей качества (ПК);
- выбор методов определения значений ПК и шкал оценок;
- определение значений ПК;
- определение уровня качества.

Сформировать 3-х уровневую НПК. Группы и подгруппы свойств показателей качества (1 и 2 уровни) выбрать из таблицы 7.2. Включить в НПК 2-3 группы показателей. Единичные ПК (3 уровень) установить самостоятельно, исходя из особенностей индивидуального ПС, с учетом назначения и условий использования ПС. Пример единичных показателей - машинонезависимость, доступность, структурированность, надежность, точность, число строк исходного текста, число выполненных операторов, количество ветвей в программе и т.д..

Построить дерево НПК.

Полученные данные оформить в виде таблицы «Условия оценки уровня качества ПС».

Таблица 7.2

Индекс ПК	Наименование показателей	Метод определения значений ПК	Знаки лучше (+), хуже (-)	Шкала оценок	Базовые значения показателей	Параметры (коэффициенты) весомости показателей
1	2	3	4	5	6	7

1. Для обозначения методов определения значений ПК в графе 3 используются следующие условные знаки: И – измерительный; Рег – регистрационный; Р – расчетный; Э – экспертный.

2. Знак (+) в графе 4 означает, что качество ПС улучшается с увеличением значения этого ПК, знак (-) наоборот.

3. При установлении шкал оценок (в графе 5) используются следующие значения: 1.1 – абсолютная; 1.2 – шкала отношений; 1.3 – интервальная; 2.0 – порядковая; 3.0 – номинальная.

4. Для показателей групп и подгрупп столбцы 5 и 6 не заполняются. Метод определения – расчетный. В графе 6 для единичных показателей кроме базовых значений в скобках указать выбранные значения единичных показателей.

Для обозначения индекса ПК в графе 1 используют 3 десятичные позиции, разделяющиеся точками. Цифра в 1 позиции – номер группы, во 2 – номер подгруппы, в 3 – номер единичного показателя. Пример в таблице 7.3.

Таблица 7.3

Индекс ПК	Наименование показателей	Метод определения значений ПК	Знаки лучше (+), хуже (-)	Шкала оценок	Базовые значения показателей	Параметры (коэффициенты) весовости показателей
1	2	3	4	5	6	7
1	функциональность	Р	+			5
1.1	Функциональная полнота	Р	+			2
1.1.1	Количество входных переменных	И	-	1.1	10 (8)	3

7.5. Содержание отчета

Цель работы; Номенклатура показателей качества; Описание этапов оценки качества ПС; Заполненную таблицу «Условия оценки уровня качества ПС»; Рассчитанное значение показателя качества программного изделия; Выводы.

7.6. Список контрольных вопросов

1. Перечислить методы определения значений единичных показателей качества.
2. Что такое номенклатура показателей качества?
3. Как производится формирование НПК?
4. Каким критериям должна удовлетворять НПК?
5. Перечислите показатели качества и дайте им характеристику.
6. Какие подгруппы включает функциональность программного средства?
7. Что включает дифференциальный метод оценки уровня качества?
8. К какой группе ПК относится тестируемость?

Лабораторная работа № 8

СТРУКТУРНОЕ ТЕСТИРОВАНИЕ

Цель работы: изучение принципов тестирования на основании анализа структуры программы, формирование маршрутов для тестирования, разработка тестов по критерию покрытия всех ветвей графа управления программы (критерий C1).

8.1. Задачи и методы тестирования

Тестирование - процесс испытания программы тестами с целью обнаружения и локализации ошибок, а также проверки соответствия программы требованиям, сформулированным в ТЗ.

Основными задачами в процессе тестирования являются:

- планирование тестирования в соответствии с выбранными критериями;
- разработка конкретных тестовых значений и соответствующих им эталонов;
- тестирование программы;
- обработка и анализ результатов тестирования.

Существует два основных подхода к решению задачи тестирования:

- а) функциональное тестирование - тесты строятся на основании функциональных свойств программы без учета ее внутренней структуры;
- б) структурное тестирование - учитывается внутренняя структура программы, анализируется прохождение данных от входа к выходу, и эта информация используется для рациональной организации тестирования.

8.1.1. Структурное тестирование

Метод структурного тестирования или белого ящика (white-box testing, glass-box testing) – тестирование, при котором тестировщик имеет доступ к коду. Его еще называют тестированием стеклянного ящика или тестированием прозрачного ящика. Тесты основаны на знании кода приложения и его внутренних механизмов.

Метод белого ящика часто используется на стадии, когда приложение ещё не собрано воедино, но необходимо проверить каждый из его компонентов, модулей, процедур и подпрограмм. Структурная информация понятна и доступна разработчикам подсистем и модулей приложения. Структурные критерии тестирования базируются на основных элементах управляющего графа программы - операторах, ветвях и путях.

При рассмотрении методов структурного тестирования удобно рассматривать программу в некоторой графической форме, например в виде графа или блок-схемы.

8.1.2. Управляющий граф программы

Управляющий граф программы (УГП) отображает поток управления программы. Это граф $G(V, A)$, где $V(V_1, \dots, V_m)$ – множество вершин (операторов), $A(A_1, \dots, A_n)$ – множество дуг (управлений), соединяющих вершины.

Путь – последовательность вершин и дуг УГП, в которой любая дуга выходит из вершины V_i и приходит в вершину V_j

Ветвь – *путь* $(V_1, V_2, \dots, V_k)$, где V_1 – либо первый, либо условный оператор, V_k – либо условный оператор, либо оператор выхода из программы, а все остальные операторы – безусловные.

Число путей в программе может быть не ограничено (пути, различающиеся хотя бы числом проходов цикла – разные). Ветви – линейные участки программы, их конечное число.

Существуют реализуемые и нереализуемые пути в программе, в нереализуемые пути в обычных условиях попасть нельзя.

Как правило, вершинам соответствуют условные утверждения программ (условия), а дугам – последовательности операторов присваивания и безусловного перехода. Последовательность дуг, ведущая от начальной вершины к конечной, называют путем или маршрутом. Пример представления программы графом управления представлен на рисунке 8.1.

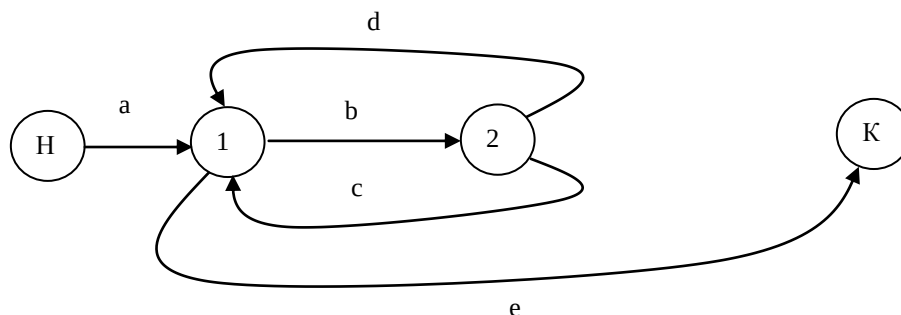


Рисунок 8.1 – Пример графа управления программы

Наиболее часто при тестировании структуры программных модулей используют следующие критерии выделения тестируемых маршрутов:

- критерий С1 - выполнение в процессе тестирования всех дуг графа программы;
- критерий В1 - покрытие границ условий. Для условных утверждений реализуются три граничных условия, например, если условие $i=0$, то требуется сформировать тесты таким образом, чтобы выполнились все три комбинации: $i=0$, $i=-1$, $i=1$;
- критерий С2 - покрытие всех пар соседних дуг графа программы (всех комбинаций вход-выход для вершин графа);
- критерий В2 - покрытие всех комбинаций "входная дуга - граница условия" для всех вершин (комбинация В1 и С2);
- критерий СР - покрытие всех пар достижимых дуг графа;

- критерий РН - покрытие всех возможных путей в графе программы при заданном ограничении на число повторений циклов.

Простейший критерий C1 - критерий покрытия всех ветвей графа управления программы. По этому критерию граф программы должен быть покрыт минимальным набором путей, проходящих через каждый оператор ветвления по каждой дуге. Повторная проверка дуг не оценивается и считается избыточной. При этом в процессе проверки гарантируется выполнение всех передач управления между операторами программы и каждого оператора не менее одного раза.

Для графа управления на рисунке 8.1, для удовлетворения критерия C1, требуется сформировать такие тестовые примеры, чтобы были пройдены пути: abce, abde. Для сравнения, по критерию C2 требуется прохождение: abce, abde, abcbce, abdbde, ae.

8.1.3. Пример использования критерия структурного тестирования

Пусть задан текст программы:

```
public void Method (ref int x) {
  if (x>17)
    x = 17-x;
  if (x== -13)
    x = 0;}

```

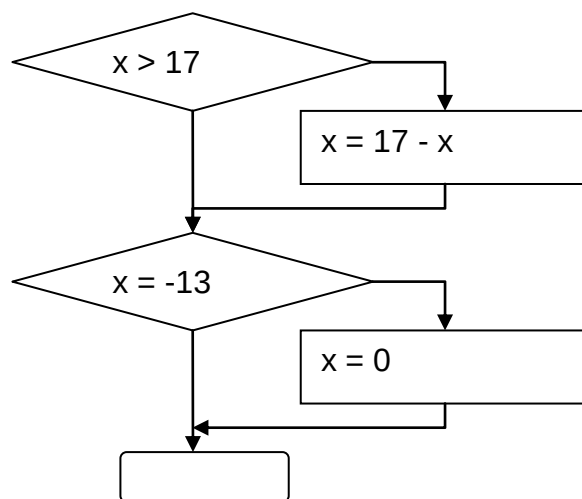


Рисунок 8.2 – Схема алгоритма

Построение тестов

критерий команд (C0): (вх, вых) = {(30, 0)} - все операторы трассы 1-2-3-4-5-6

критерий ветвей (C1): (вх, вых) = {(30, 0), (17, 17)}

критерий путей (C2): (вх, вых) = {(30,0), (17,17), (-13,0), (21,-4)}

8.2. Задание на выполнение работы

1. Составить граф управления индивидуального модуля. Считать вершинами - условия, а дугами - операторы присваивания, ввода, вывода и др. Обозначить дуги (например, латинскими буквами).
2. По графу выбрать минимальное множество маршрутов, соединяющих начальную и конечную вершины графа, удовлетворяющих критериям $S1, V1, S2$. Маршрут записать как последовательность латинских букв.
3. Определить значения входных переменных, соответствующих каждому выбранному маршруту. Сформировать тесты.
4. Провести тестирование программы, выполнив ее для каждого тестового примера.
5. Оформить отчет.

8.3. Содержание отчета

Отчет должен включать: блок-схему с обозначенными дугами; набор маршрутов; соответствующие маршрутам тесты; результаты работы программы для каждого тестового примера; выводы.

8.4. Список контрольных вопросов

1. Как структурное тестирование учитывает конкретную реализацию программы?
2. Перечислить недостатки структурного тестирования?
3. С какой целью используется редуцированный граф?
4. Как учитывается кратность повторения циклов?
5. Перечислите критерии структурного тестирования
6. Что представляет собой граф управления программы?
7. На каком этапе работы с программой используется структурное тестирование?
8. Что представляет собой маршрут на графе управления?

Лабораторная работа № 9

ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ

Цель работы: изучение принципов функционального тестирования программы на основании эквивалентных разбиений и функциональных диаграмм, приобретение практических навыков построения тестов.

9.1. Теоретические сведения

9.1.1. Функциональное тестирование (метод черного ящика)

Функциональное тестирование (functional testing) – процесс проверки программного обеспечения, сконцентрированный на анализе соответствия ПО требованиям и спецификациям. Функциональное тестирование ещё называют поведенческим или тестированием на поведенческом уровне.

Цели функционального тестирования:

- Обнаружить дефекты в программном продукте.
- Определить степень соответствия программного продукта требованиям и ожиданиям заказчика.
- Принять решение о возможности передачи продукта заказчику.

Для обнаружения всех ошибок в программе, используя управление по данным, необходимо выполнить *исчерпывающее* тестирование, т.е. тестирование на всех возможных наборах данных. Для тех же программ, где исполнение команды зависит от предшествующих ей событий, необходимо проверить и все возможные последовательности. Очевидно, что проведение исчерпывающего тестирования для подавляющего большинства случаев невозможно. Поэтому обычно выполняют «разумное» или «приемлемое» тестирование, которое ограничивается прогонами программы на небольшом подмножестве всех возможных входных данных. Этот вариант не дает гарантии отсутствия отклонений от спецификаций.

- Правильно выбранный тест должен уменьшать, причем более чем на единицу, число других тестов, которые должны быть разработаны для обеспечения требуемого качества программного обеспечения.

9.1.2. Метод эквивалентных разбиений

Для обнаружения всех ошибок в программе, используя управление по данным, необходимо выполнить тестирование на всех без исключения возможных наборах данных. Очевидно, что проведение исчерпывающего тестирования для подавляющего большинства случаев невозможно. Поэтому обычно выполняют «разумное» или «приемлемое» тестирование, которое ограничивается прогонами программы на небольшом подмножестве всех возможных входных данных.

Тесты, построенные по принципу эквивалентных разбиений, должны обладать двумя свойствами:

1) уменьшать число других тестов, которые должны быть разработаны для достижения заранее определенной цели "приемлемого" тестирования, т.е. каждый тест должен включать столько различных входных условий, сколько это возможно, с тем, чтобы минимизировать число различных тестов;

2) покрывать значительную часть других возможных тестов, т.е. необходимо пытаться разбить входную область программы на конечное число классов эквивалентности так, чтобы можно было предположить, что каждый тест, являющийся представителем некоторого класса, эквивалентен любому другому тесту этого класса. Иными словами, если один тест класса эквивалентности обнаруживает ошибку, то и все другие тесты этого класса эквивалентности будут обнаруживать ту же самую ошибку. Наоборот, если тест не обнаруживает ошибки, то следует ожидать, что ни один тест этого класса эквивалентности не будет обнаруживать ошибки.

Разработка тестов методом эквивалентного разбиения осуществляется в два этапа:

- выделение классов эквивалентности;
- построение тестов.

Классы эквивалентности выделяются путем анализа функциональной спецификации, которая обычно зафиксирована в техническом задании в требованиях к функциональным характеристикам. Входные условия разбиваются на две или более группы. Для проведения этой операции используют таблицу 9.1.

Таблица 9.1

Входные условия	Правильные классы эквивалентности	Неправильные классы эквивалентности
-----------------	-----------------------------------	-------------------------------------

Различают два типа классов эквивалентности: правильные, представляющие правильные входные данные программы и неправильные, представляющие все другие возможные состояния, т.е. ошибочные входные значения. Выделение классов эквивалентности представляет собой эвристический процесс. При этом существует ряд правил:

1. Если входное значение описывает область значений (например, целое число может принимать значения от 1 до 99), то определяются один правильный класс эквивалентности ($1 \leq \text{число} \leq 99$) и два неправильных ($\text{число} < 1$ и $\text{число} > 99$).

2. Если входное условие описывает число значений (например, "в автомобиле могут ехать от одного до шести человек"), то определяются один правильный класс эквивалентности и два неправильных (ни одного и более шести человек).

3. Если входное условие описывает множество входных значений и каждое значение программа трактует особо, то определяется один правильный класс эквивалентности для каждого значения («да») и один неправильный класс эквивалентности («нет»).

4. Если входное условие описывает ситуацию "должно быть" (например, первым символом идентификатора должна быть буква, то определяется один правильный класс эквивалентности (первый символ - буква) и один неправильный (первый символ - не буква).

5. Если есть основание считать, что различные элементы класса эквивалентности трактуются программой неодинаково, то данный класс эквивалентности разбивается на меньшие классы эквивалентности.

Процесс построения тестов включает в себя:

1. Назначение каждому классу эквивалентности уникального номера.

2. Проектирование тестов, каждый из которых покрывает как можно большее число непокрытых правильных классов эквивалентности до тех пор, пока все правильные классы эквивалентности не будут покрыты. Если нет правильных классов, содержащих перечисления, то обычно для всех правильных классов строится один тестовый набор.

3. Проектирование тестов, каждый из которых покрывает один и только один из непокрытых неправильных классов эквивалентности, до тех пор, пока все неправильные классы эквивалентности не будут покрыты тестами.

Причина покрытия неправильных классов эквивалентности индивидуальными тестами состоит в том, что может присутствовать взаимовлияние одних ошибок на другие.

	Классы корректных данных (позитивные тесты)	Классы некорректных данных (негативные тесты)	Граничные и специальные значения	Примечания
Первое слагаемое	от -99 до -10 от -9 до -1 0 от 1 до 9 от 10 до 99	> 99 < -99	0, 1, -1, 9, -9 10, -10 99, -99 100, -100	

Рисунок 9.1 – Пример построения классов эквивалентности

9.1.3. Метод функциональных диаграмм

Метод функциональных диаграмм или диаграмм причинно-следственных связей предназначен для исследования комбинаций входных условий. Поскольку число комбинаций обычно велико, выбор произвольного подмножества может привести к неэффективному тесту. Метод функциональных диаграмм помогает систематически выбирать высокорезультативные тесты. Он дает полезный побочный эффект, так как позволяет обнаруживать неполноту и неоднозначность исходных спецификаций.

Функциональная диаграмма представляет собой формальный язык, на который транслируется спецификация, написанная на естественном языке.

Диаграмме можно сопоставить цифровую логическую сеть. Построение тестов этим методом осуществляется в несколько этапов:

1. Спецификация разбивается на “рабочие” участки. Например, при тестировании компилятора в качестве рабочего участка можно рассматривать каждый отдельный оператор языка программирования.

2. В спецификации определяются причины и следствия. *Причина* – отдельное входное условие или класс эквивалентности входных условий. *Следствие* – результат, действие, которое входное условие оказывает на состояние программы или системы. Причины и следствия определяются путем последовательного чтения спецификации. Каждой причине и следствию присписывается отдельный номер.

3. Анализируется семантическое содержание спецификации, которая преобразуется в булевый граф, связывающий причины и следствия. Это и есть функциональная диаграмма.

4. Диаграмма снабжается примечаниями, задающими ограничения и описывающими комбинации причин и (или) следствий, которые являются невозможными из-за синтаксических или внешних ограничений.

5. Путем отслеживания состояний условий диаграммы, она преобразуется в таблицу решений. Каждый столбец таблицы решений соответствует тесту.

6. Столбцы таблицы решений преобразуются в тесты.

Каждый узел диаграммы может находиться в двух состояниях: 0 или 1; 0 обозначает состояние “отсутствует”, а 1 – “присутствует”.

1. Функция *тождество* устанавливает, что если $a=1$, то $b=1$; в противном случае $b=0$

2. Функция *не* устанавливает, что если $a=1$, то $b=0$; в противном случае $b=1$.

3. Функция *или* устанавливает, что если $a=1$ или $b=1$ то $c=1$; в противном случае $c=0$.

4. Функция *и* устанавливает, что если и $a=1$ и $b=1$ то $c=1$; в противном случае $c=0$. Функции *и*, *или* допускают любое число входов.

В большинстве программ определенные комбинации входных условий невозможны из-за синтаксических или внешних ограничений. В этом случае используются дополнительные входные схемы:

5. Ограничение *исключает* устанавливает, что при $E=1$, хотя бы одно значение a или b принимает значение 1 (a и b не могут быть 1 одновременно).

6. Ограничение *включает* устанавливает, что по крайней мере одна из величин a или b всегда должна быть равна 1 (не могут принимать значение 0 одновременно).

7. Ограничение *Одно и только одно* устанавливает, что одна и только одна из величин a или b должна быть равна 1.

8. Ограничение *R* устанавливает, что если $a=1$, то и b должна быть равна 1.

9. Ограничение *M* устанавливает, что если $a=1$, то b должна быть равна 0.

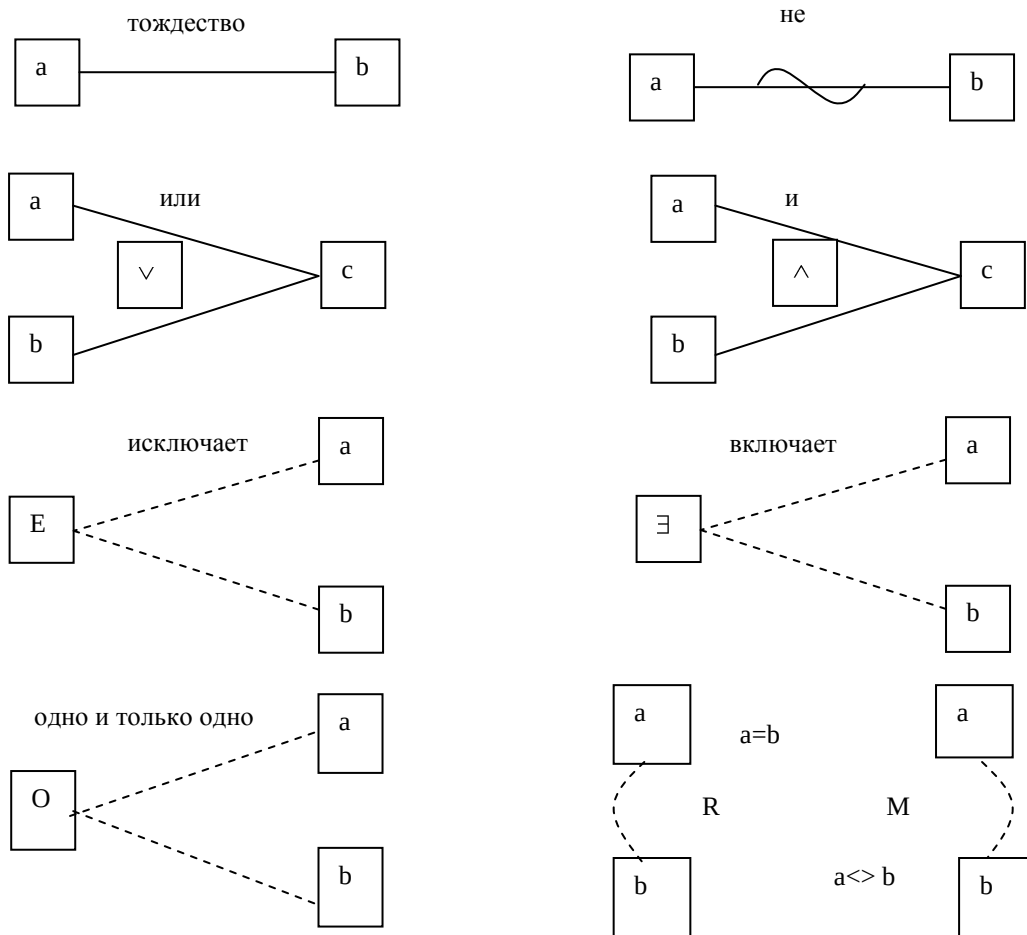


Рисунок 9.2 – Базовые символы для записи функциональных диаграмм

Рассмотрим диаграмму, изображающую спецификацию: Первый вводимый символ должен быть либо буквой А либо В, а второй – цифрой. Если первый символ неправильный, выдается сообщение X1, если второй символ неправильный – сообщение X2. В случае, если оба символа правильные, происходит обновление некоторого файла F (сообщение X3).

В качестве причин можно выбрать следующие:

1. Первый символ буква А;
2. Первый символ буква В;
3. Второй символ цифра.

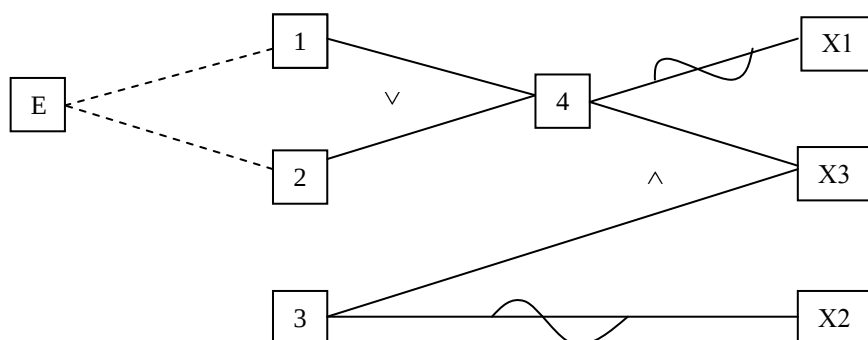


Рисунок 9.2 – Функциональная диаграмма

Генерация таблицы решений заключается в следующем:

1. Выбрать некоторое следствие, которое должно быть в состоянии 1.
2. Найти все комбинации причин, которые устанавливают это следствие в 1, прокладывая из этого следствия обратную трассу через диаграмму.
3. Построить столбец в таблице решений для каждой комбинации причин.
4. Определить состояние других следствий и поместить их в соответствующий столбец решений. На этом шаге необходимо учитывать следующие положения:

П1. Если обратная трасса прокладывается через узел *или*, выход которого равен 1, не следует устанавливать в 1 более одного входа в этот узел. Цель данного правила – избежать пропуска определенных ошибок из-за того, что одна причина маскируется другой.

П2. Если обратная трасса прокладывается через узел *или*, выход которого равен 0, то следует перечислить все ситуации, при которых оба входа равны 0.

ПЗ.1. Если обратная трасса прокладывается через узел *и*, выход которого равен 0, то следует рассмотреть все комбинации входов, приводящие выход в 0.

ПЗ.2. Когда исследуется ситуация, где один вход – 0, а один или более других входов – 1, не обязательно перечислять все условия, при которых остальные входы могут быть равны 1.

П4. Если обратная трасса прокладывается через узел *и*, выход которого должен принимать значение 0, необходимо указать лишь одно условие, согласно которому все входы являются 0.

П5. Если обратная трасса прокладывается через узел *и*, выход которого должен принимать значение 1, необходимо рассмотреть все условия, при которых входы являются 1.

П6. Если состояние причины несущественно, либо состояние причины очевидно вследствие состояний других зависимых причин, то вместо 0 или 1 ставится пробел.

Эти положения позволяют исключить малорезультативные тесты.

Таблица решений (таблица 9.2), построенная на основании функциональной диаграммы, имеет вид:

Таблица 9.2

	1	2	3	4
1	0	1	0	1
2	0	0	1	0
3	1	1	1	0
X1	1	0	0	0
X3	0	1	1	0
X2	0	0	0	1

Столбцы 2 и 3 построены с учетом правила П5.

Столбец 4 построен с учетом правила ПЗ.2.

Набор из четырех тестов представлен ниже.

1 12 2 A1 3 B1 4 AC

9.2. Порядок выполнения работы

1. Выделить классы эквивалентности для индивидуального модуля. Для записи классов эквивалентности использовать таблицу 1.
2. Построить множество тестов.
3. Выполнить программу для каждого тестового набора.
4. Построить функциональную диаграмму для индивидуального модуля. Для задания причин можно воспользоваться входными условиями из метода эквивалентных разбиений.
6. Провести преобразование функциональной диаграммы в таблицу решений.
7. На основании полученной таблицы решений построить тестовые наборы.
Сравнить с тестами, полученными по методу эквивалентных разбиений.
8. Выполнить программу для полученных тестовых наборов. Результаты работы программы проанализировать.
9. Оформить отчет.

9.3. Содержание отчета

Отчет должен включать: таблицу эквивалентности, множество эквивалентных тестов, результаты испытания программы по методу эквивалентности, функциональную диаграмму, таблицу решений, тестовые наборы, построенные на основании таблицы решений, результаты тестирования, выводы.

9.4. Контрольные вопросы

1. Как в методе функциональных диаграмм используются особенности программы?
2. В чем заключаются недостатки рассматриваемого метода?
3. Что дает функциональное тестирование?
4. В чем отличие метода функциональных диаграмм от метода эквивалентных разбиений?
5. Каковы принципы построения классов эквивалентности?
6. Как генерируются тесты для правильного класса эквивалентности и почему?
7. Как генерируются тесты для неправильных классов эквивалентности и почему?
8. Базовые символы для записи функциональных диаграмм.

Лабораторная работа № 10 ПРОГРАММНАЯ ДОКУМЕНТАЦИЯ

Цель работы: ознакомиться с основными видами программных документов, получить навыки работы с ними.

10.1. Теоретические сведения

Документация - печатный текст, сопровождающий программное обеспечение для объяснения принципов его функционирования или использования.

Цели документирования:

- посредничество между разработчиками ПО;
- упрощение сопровождения и эволюции;
- информация для планирования и оценки затрат в процессе разработки;
- инструкции по использованию и управлению программной системой;
- основание для сертификации системы

К программным документам относят документы, содержащие сведения, необходимые для разработки, изготовления, сопровождения и эксплуатации программ.

Основные виды программных документов и их содержание приведены в таблице 10.1.

Таблица 10.1

Вид программного документа	Содержание программного документа
Спецификация	Состав программы и документации на нее
Текст программы	Запись программы с необходимыми комментариями
Программа и методика испытаний	Требования, подлежащие проверке при испытании программы, а также порядок и методы их контроля
Техническое задание	Назначение и область применения программы, технические, технико-экономические и специальные требования, предъявляемые к программе, необходимые стадии и сроки разработки, виды испытаний
Пояснительная записка	Схема алгоритма, общее описание алгоритма и (или) функционирования программы, а также обоснование принятых технических и технико-экономических решений
Эксплуатационные документы	Сведения для обеспечения функционирования и эксплуатации программы

Виды эксплуатационных документов и их содержание приведены таблице 10.2.

Таблица 10.2

Вид эксплуатационного документа	Содержание эксплуатационного документа
Описание применения	Сведения о назначении программы, области применения, применяемых методах, классе решаемых задач, ограничениях для применения, минимальной конфигурации технических средств
Руководство системного программиста	Сведения для проверки, обеспечения функционирования и настройки программы на условия конкретного применения
Руководство программиста	Сведения для эксплуатации программы
Руководство оператора	Сведения для обеспечения процедуры общения оператора с вычислительной системой в процессе выполнения программы

Руководство программиста (ГОСТ 19504-79)

Руководство программиста должно содержать следующие разделы:

- назначение и условия применения программ;
- характеристика программы;
- обращение к программе;
- входные и выходные данные;
- сообщения.

В разделе «Назначение и условия применения программ» должны быть указаны назначение и функции, выполняемые программой, условия, необходимые для выполнения программы (объем оперативной памяти, требования к составу и параметрам периферийных устройств, требования к программному обеспечению и т.п.).

В разделе «Характеристика программы» должно быть приведено описание основных характеристик и особенностей программы (временные характеристики, режим работы, средства контроля правильности выполнения и самовосстанавливаемости программы и т.п.).

В разделе «Обращение к программе» должно быть приведено описание процедур вызова программы (способы передачи управления и параметров данных и др.).

В разделе «Входные и выходные данные» должно быть приведено описание организации используемой входной и выходной информации и, при необходимости, ее кодирования.

В разделе «Сообщения» должны быть указаны тексты сообщений, выдаваемых программисту или оператору в ходе выполнения программы, описание их содержания и действий, которые необходимо предпринять по этим сообщениям.

Описание программы (гост 19.402-78)

Основная часть документа должна состоять из вводной части и следующих разделов:

- введение;
- функциональное назначение;
- описание логики.
- условия применения;
- состав и функции.

В вводной части документа приводится информация общего характера о программе - полное наименование, обозначение, ее возможные применения и т.п.

Например: Программа "Автоматизированное рабочее место разработчика САУ" предназначена для ... реализована на Программа поддерживает ...

В разделе Назначение указывают назначение программы и приводят общее описание функционирования программы, ее основные характеристики, сведения об ограничениях, накладываемых на область применения программы, а также указывают типы электронных вычислительных машин и устройств, которые используются при работе.

Например: Программа предназначена для решения задач ... Программа представляет собой ядро автоматизированного рабочего места ...

Пользователь имеет возможность ..., осуществить ..., запустить ..., проанализировать ..., получить результаты анализа и обработки ..., построить ... и т.п.

В разделе "Описание логики" указывают:

описание структуры программы и ее основных частей с указанием их функций и связей между ними.

Например: В состав программы входит следующее:

- пользовательский интерфейс,
- модуль определения путей в графе,
- модуль расчета передаточной функции,
- модуль построения амплитудно- и фазочастотных характеристик,
- модуль построения реакции на полиномиальное воздействие,
- текстовый редактор).

В разделе Условия применения указываются условия, необходимые для выполнения программы (требования к необходимым для данной программы техническим средствам, и другим программам, общие характеристики входной и выходной информации, а также требования и условия организационного, технического и технологического характера и т.п.).

Например: Программа эксплуатируется на персональном компьютере (ПК) типа IBM PC/AT. Для работы в диалоговом режиме используется экран дисплея, клавиатура и манипулятор типа "мышь". Для поддержки графического режима необходим адаптер EGA (VGA). Входные данные хранятся на флоппи- и/или жестком дисках. Программа работает под управлением ОС ...

Текст программы (гост 19.401-78)

Требования к оформлению текста программы достаточно просты и естественны для грамотного программиста. Основное, чем требуется руководствоваться при создании этого документа – это то, чтобы текст программы был удобочитаемым.

Программа и методика испытаний (гост 19.301-79)

В этом документе содержится описание того, что и как необходимо сделать, дабы убедиться (и убедить Заказчика) в правильности работы программы. Фактически, этот документ является определяющим для приемосдаточных испытаний. Грамотно составленная программа и методика испытаний – это залог подписания акта сдачи-приемки, т.е. того, во имя чего вы потратили столько сил и времени.

Документ содержит описание объекта и цели испытаний, требования к программе и к программной документации, средства и порядок испытаний, а также описание тестовых примеров.

Пример.

Объект испытаний

Объектом испытаний является программа ..., предназначенная для ...

Цель испытаний

Проверка надежности функционирования программы.

Требования к программе

Функционирование программы не должно приводить к сбою (фатальному нарушению работы системы). Организация диалога должна предусматривать защиту от ввода некорректных данных. Программа должна выдавать диагностику состояния системы и сообщения о любых возникших ошибках ... и т.п.

Требования к программной документации

Состав программной документации, предъявляемой на испытании:

- описание программы (ГОСТ 19.402-78);
- программа и методика испытаний (ГОСТ 19.301-79);
- текст программы (ГОСТ 19.401-78).

Средства и порядок испытаний

Программа работает в соответствии с условиями эксплуатации ОС Windows XP.

Порядок проведения испытаний:

- Запуск программы осуществляется
- Выбирается ...

- Нажимается ...
- Последовательно выбираются ...

Тестовые примеры

Для проведения испытаний предлагаются ..., описание которых содержатся в файлах ... Содержимое тестовых файлов и результаты работы программы приведены в Приложении 1.

10.2. Задание на лабораторную работу

В соответствии с разработанным в первой работе Техническим Заданием реализовать программный продукт. Язык программирования Java. Отчет по лабораторной работе должен содержать следующие программные документы:

- Руководство программиста (ГОСТ 19504-79);
- Описание программы (ГОСТ 19.402-78);
- Текст программы (ГОСТ 19.401-78);
- Программа и методика испытаний (ГОСТ 19.301-79).

10.3. Контрольные вопросы

1. Виды программных документов?
2. Содержание документа «Руководство программиста (ГОСТ 19504-79)».
3. Содержание документа «Описание программы (ГОСТ 19.402-78)».
4. Содержание документа «Текст программы (ГОСТ 19.401-78)».
5. Содержание документа «Программа и методика испытаний (ГОСТ 19.301-79)».
6. Каковы цели документирования программных средств?
7. Каковы требования к программной документации?
8. Какие документы относятся к эксплуатационным?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Введение в программную инженерию : учебник / В.А. Антипов, А.А. Бубнов, А.Н. Пылькин, В.К. Столчнев. — Москва : КУРС : ИНФРА-М, 2024. — 336 с. - URL: <https://znanium.ru/catalog/product/1905724> (дата обращения: 02.05.2024). — Режим доступа: по подписке. - ISBN 978-5-906923-22-6. - Текст : электронный.
2. Коваленко, В. В. Проектирование информационных систем : учебное пособие / В. В. Коваленко. — 2-е изд., перераб. и доп. — Москва : ИНФРА-М, 2023. — 357 с. - URL: <https://znanium.com/catalog/product/1894610> (дата обращения: 19.04.2024). — Режим доступа: по подписке. — (Высшее образование: Бакалавриат). — DOI 10.12737/987869. - ISBN 978-5-00091-783-1. - Текст : электронный.
3. Лаврищева, Е. М. Программная инженерия и технологии программирования сложных систем : учебник для вузов / Е. М. Лаврищева. — 2-е изд., испр. и доп. — Москва : Юрайт, 2023. — 432 с. — URL: <https://urait.ru/bcode/513067> (дата обращения: 19.04.2024). — (Высшее образование). — ISBN 978-5-534-07604-2. — Текст : электронный.
4. Лаврищева, Е. М. Программная инженерия. Парадигмы, технологии и CASE-средства : учебник для вузов / Е. М. Лаврищева. — 2-е изд., испр. — Москва : Издательство Юрайт, 2024. — 280 с. — (Высшее образование). — ISBN 978-5-534-01056-5. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: <https://urait.ru/bcode/537884> (дата обращения: 19.04.2024).
5. Тузовский, А. Ф. Проектирование и разработка web-приложений : учебное пособие для вузов / А. Ф. Тузовский. — Москва : Юрайт, 2024. — 219 с. — URL: <https://urait.ru/bcode/537106> (дата обращения: 19.04.2024). — (Высшее образование). — ISBN 978-5-534-16300-1. — Текст : электронный.
6. Управление программными проектами : учебное пособие для вузов / В. Е. Гвоздев [и др.] ; под редакцией Р. Ф. Маликова. — Москва : Юрайт, 2024. — 167 с. — URL: <https://urait.ru/bcode/543929> (дата обращения: 19.04.2024). — (Высшее образование). — ISBN 978-5-534-14329-4. — Текст : электронный.
7. Черткова, Е. А. Программная инженерия. Визуальное моделирование программных систем : учебник для вузов / Е. А. Черткова. — 3-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2024. — 146 с. — URL: <https://urait.ru/bcode/534516> (дата обращения: 19.04.2024). — (Высшее образование). — ISBN 978-5-534-18197-5. — Текст : электронный.

ПРИЛОЖЕНИЕ А

1. Основные компоненты языка UML

Язык UML представляет собой язык визуального моделирования, который разработан для спецификации, визуализации, проектирования и документирования компонентов программного обеспечения, бизнес-процессов и других систем. В рамках языка UML все представления о модели сложной системы фиксируются в виде специальных графических конструкций, получивших название диаграмм.

2. Диаграмма классов (class diagram)

Класс (class) в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов. Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции. В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы). Обязательным элементов обозначения класса является его имя

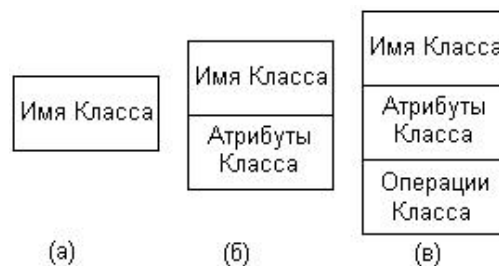


Рисунок 1 - Графическое изображение класса на диаграмме классов

На диаграмме также указываются различные отношения между классами

1. **Отношение зависимости.** Используется в такой ситуации, когда некоторое изменение одного элемента модели может потребовать изменения другого зависимого от него элемента модели. Графически изображается пунктирной линией между соответствующими элементами со стрелкой на одном из ее концов ("→" или "←"). При этом стрелка направлена от класса-клиента (А) зависимости к независимому классу или классу-источнику (Б) (рисунок 2).

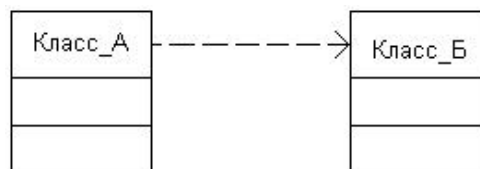


Рисунок 2 - Графическое изображение отношения зависимости на диаграмме классов

2. **Отношение ассоциации.** Соответствует наличию некоторого отношения между классами. Обозначается сплошной линией с дополнительными специальными символами, которые характеризуют отдельные свойства конкретной ассоциации (имя ассоциации, имена и кратность классов-ролей). Наиболее простой случай данного отношения - бинарная ассоциация. Она связывает в точности два класса и, как исключение, может связывать класс с самим собой. Для бинарной ассоциации на диаграмме может быть указан порядок

следования классов с использованием треугольника в форме стрелки рядом с именем данной ассоциации. Направление этой стрелки указывает на порядок классов, один из которых является первым (со стороны треугольника), а другой - вторым (со стороны вершины треугольника). Отсутствие данной стрелки означает, что порядок следования классов в рассматриваемом отношении не определен.



Рисунок 3 - Графическое изображение отношения бинарной ассоциации между классами

N-арная ассоциация графически обозначается ромбом, от которого ведут линии к символам классов данной ассоциации. В этом случае ромб соединяется с символами соответствующих классов сплошными линиями. Имя N-арной ассоциации записывается рядом с ромбом соответствующей ассоциации. Порядок классов в N-арной ассоциации на диаграмме не фиксируется. Отношение между тремя классами: "Футбольная команда", "Год" и "Игра" указывает на наличие отношения между этими тремя классами, которое может представлять информацию об играх футбольных команд в национальном чемпионате в течение нескольких последних лет.



Рисунок 40 - Графическое изображение ассоциации между тремя классами

Кратность отдельного класса обозначается в виде интервала целых чисел, аналогично кратности атрибутов и операций классов. Интервал записывается рядом с концом ассоциации и для N-арной ассоциации означает потенциальное число отдельных экземпляров. Так, для примера на рисунке 3 кратность "1" для класса "Компания" означает, что каждый сотрудник может работать только в одной компании. Кратность "1..*" для класса "Сотрудник" означает, что в каждой компании могут работать несколько сотрудников, общее число которых заранее неизвестно и ничем не ограничено.

Частным случаем отношения ассоциации является исключаящая ассоциация (Xor-association), указывающая на тот факт, что из нескольких потенциально возможных вариантов в каждый момент времени может использоваться только один. Изображается пунктирной линией, рядом с которой записывается строка-ограничение "{xor}". Например, счет в банке может быть открыт для клиента, в качестве которого может выступать физическое лицо или компания (рисунок 5).

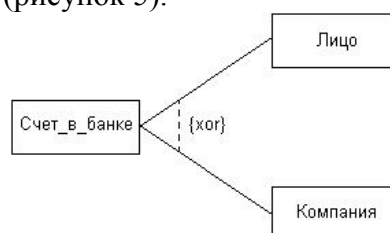


Рисунок 5 - Графическое изображение исключаящей ассоциации между тремя классами

3. **Отношение агрегации.** Имеет место между несколькими классами в том случае, если один из классов представляет собой некоторую сущность, включающую в себя в качестве составных частей другие сущности. Графически изображается сплошной линией, один из концов которой представляет собой незакрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой "целое". Остальные классы являются его "частями" (рисунок 6).

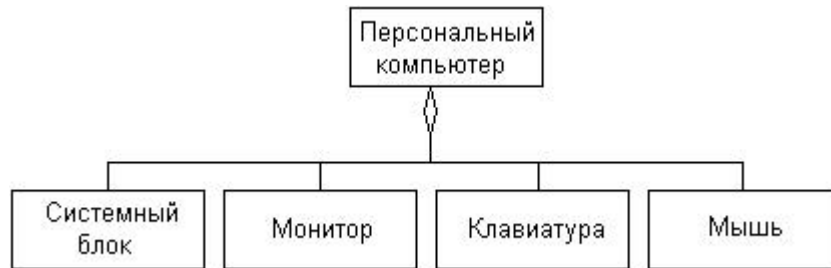


Рисунок 6 - Диаграмма классов для иллюстрации отношения агрегации на примере ПК

4. **Отношение композиции.** Служит для выделения специальной формы отношения "часть-целое", при которой составляющие части находятся внутри целого. Пример - окно интерфейса программы, которое может состоять из строки заголовка, кнопок управления размером, полос прокрутки, главного меню, рабочей области и строки состояния. Окно представляет собой класс, а его компоненты являются как классами, так и атрибутами или свойствами окна. Графически отношение композиции изображается сплошной линией, один из концов которой представляет собой закрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой класс-композицию или "целое". Остальные классы являются его "частями" (рисунок 7).

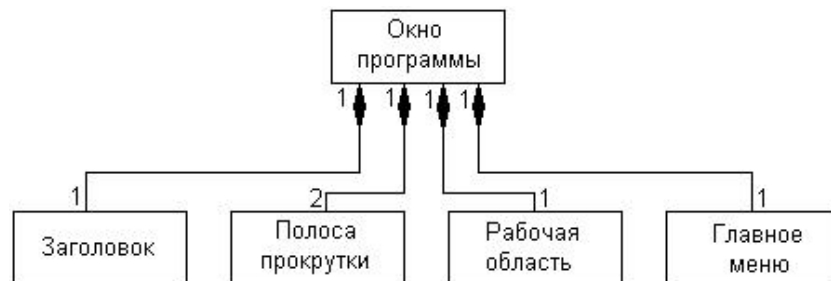


Рисунок 7 - Диаграмма классов для иллюстрации отношения композиции

5. **Отношение обобщения.** Описывает иерархическое строение классов и наследование их свойств и поведения. При этом предполагается, что класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет свои собственные свойства и поведение, которые отсутствуют у класса-предка. На диаграммах обозначается сплошной линией с треугольной стрелкой на одном из концов (рисунок 8). Стрелка указывает на более общий класс (класс-предок или суперкласс), а ее отсутствие - на более специальный класс (класс-потомок или подкласс).



Рисунок 8 - Пример графического изображения отношения обобщения классов

Рядом со стрелкой обобщения может размещаться строка текста, указывающая на наличие ограничений, в качестве которых быть использованы ключевые слова языка UML:

- {complete} - других классов-потомков у данного класса-предка быть не может. Пример - класс Клиент_банка является предком для двух классов: Физическое_лицо и Компания, и других классов-потомков он не имеет.
- {disjoint} - что классы-потомки не могут содержать объектов, одновременно являющихся экземплярами двух или более классов.
- {incomplete} - предполагается, что на диаграмме указаны не все классы-потомки. В последующем возможно восполнить их перечень не изменяя уже построенную диаграмму.
- {overlapping} отдельные экземпляры классов-потомков могут принадлежать одновременно нескольким классам. Пример - класс "Многоугольник" является классом-предком для класса "Прямоугольник" и класса "Ромб". Однако существует отдельный класс "Квадрат", экземпляры которого являются объектами первых двух классов.

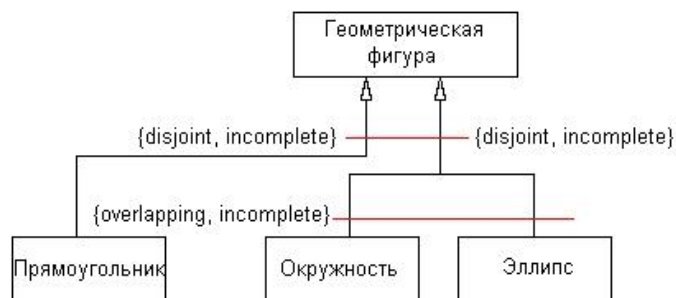


Рисунок 9 - Отношения обобщения классов с использованием строки-ограничения

Интерфейсы. Для их изображения используется специальный графический символ - прямоугольник класса с ключевым словом или стереотипом "interface". При этом секция атрибутов у прямоугольника отсутствует, а указывается только секция операций.

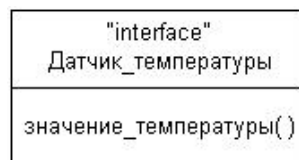


Рисунок 10 - Пример графического изображения интерфейса на диаграмме классов

Объект. Является отдельным экземпляром класса, который создается на этапе выполнения программы. Он имеет свое собственное имя и конкретные значения атрибутов. Для графического изображения объектов используется такой же символ прямоугольника, что и для классов. Имена объектов обязательно подчеркиваются (рисунок 11). При этом запись имени объекта представляет собой строку текста "имя объекта:имя класса".



Рисунок 12 - Пример графического изображения объектов на диаграммах языка UML

Шаблоны или параметризованные классы. Предназначен для обозначения класса, который имеет один (или более) нефиксированный формальный параметр. Обычно параметрами шаблонов служат типы атрибутов классов, такие как целые числа, перечисление, массив строк и др. Графически шаблон изображается прямоугольником, к верхнему правому углу которого присоединен маленький прямоугольник из пунктирных линий (рисунок 13), большой прямоугольник может быть разделен на секции, аналогично обозначению для класса. В верхнем прямоугольнике указывается список формальных параметров для тех классов, которые могут быть получены на основе данного шаблона.

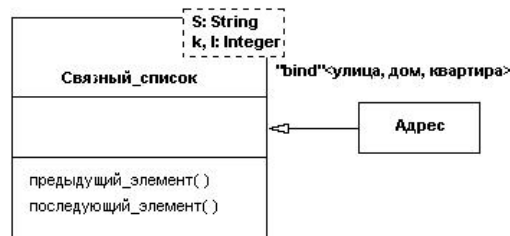


Рисунок 12 - Пример использования шаблона на диаграмме классов

Пример. Диаграмма классов для моделирования предметной области компьютерной обучающей системы.



**ИНЖЕНЕРИЯ
ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

Методические указания

Составитель: Лебедева Марина Анатольевна

В авторской редакции

Изд. № 82/2024. Объем 4,75 п.л. Усл. печ. л. 4,42. Уч.-изд. л. 4,66.
Издательский центр ФГАОУ ВО «Севастопольский государственный университет»