

Федеральное государственное бюджетное образовательное  
учреждение высшего образования  
«Саратовский государственный технический университет имени  
Гагарина Ю.А.»

Филиал федерального государственного бюджетного образовательного  
учреждения высшего образования  
«Саратовский государственный технический университет имени  
Гагарина Ю.А.» в г. Петровске

УТВЕРЖДАЮ  
Директор филиала СГТУ  
имени Гагарина Ю.А. в г. Петровске  
Е.А. Бесшапошникова  
«06» 07 2024 г.



## **МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ**

по междисциплинарному курсу  
МДК.01.02 «Поддержка и тестирование программных модулей»

специальности  
«Информационные системы и программирование»

Методические указания рассмотрены  
на заседании предметной (цикловой) комиссии  
обще профессиональных дисциплин,  
профессиональных модулей специальностей  
технического профиля  
«14» июня 2024 года, протокол №12

Председатель ПЦК  /Ю.А. Табарова/

Петровск 2024

## **Пояснительная записка**

Методические указания по выполнению практических работ подготовлены на основе рабочей программы учебной дисциплины МДК.01.02 «Поддержка и тестирование программных модулей», разработанной на основе ФГОС СПО по специальности 09.02.07 «Информационные системы и программирование» и соответствующих общих (ОК) и профессиональных (ПК) компетенций:

ПК 1.3. Выполнять отладку программных модулей с использованием специализированных программных средств.

ПК1.4. Выполнять тестирование программных модулей

ПК 1.5. Осуществлять рефакторинг и оптимизацию программного кода.

ОК.01.Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам

ОК.02. Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ОК.03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях.

ОК.04. Эффективно взаимодействовать и работать в коллективе и команде.

ОК.05. Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста.

ОК.06. Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей, в том числе с учетом гармонизации межнациональных и межрелигиозных отношений, применять стандарты антикоррупционного поведения.

ОК.07. Содействовать сохранению окружающей среды, ресурсосбережению, применять знания об изменении климата, принципы бережливого производства, эффективно действовать в чрезвычайных ситуациях.

ОК 08. Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности.

ОК 09. Пользоваться профессиональной документацией на государственном и иностранном языках.

ОК.10. Использовать знания по финансовой грамотности, планировать предпринимательскую деятельность в профессиональной сфере.

Целью освоения учебной дисциплины «Методы создания и корректировки компьютерных моделей» является:

При выполнении практических работ студент должен **знать**:

- основные этапы разработки программного обеспечения;
- основные принципы технологии структурного и объектно-ориентированного программирования;
- способы оптимизации и приемы рефакторинга;
- основные принципы отладки и тестирования программных продуктов

При выполнении практических работ студент должен **уметь**:

- осуществлять разработку кода программного модуля на языках низкого и высокого уровней;
- создавать программу по разработанному алгоритму как отдельный модуль;
- выполнять отладку и тестирование программы на уровне модуля; осуществлять разработку кода программного модуля на современных языках программирования;
- уметь выполнять оптимизацию и рефакторинг программного кода;
- оформлять документацию на программные средства

Содержание практических занятий определено рабочей программой и тематическим планированием, соответствует теоретическому материалу изучаемых разделов учебной дисциплины.

Объём практических занятий по дисциплине определяется учебным планом по данной специальности.

Продолжительность практического занятия - 2 академических часа. Перед проведением практического занятия преподавателем организуется инструктаж, а по ее окончании – обсуждение итогов.

Комплект методических указаний по выполнению практических работ дисциплины «Поддержка и тестирование программных модулей» содержит 27 практических занятий.

Перечень практических работ по дисциплине «Методы создания и  
корректировки компьютерныхмоделей»

**ПРАКТИЧЕСКАЯ РАБОТА №1**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №2**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №3**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №4**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №5**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА № 6**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №7**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №8**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №9**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №10**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №11**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №12**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №13**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №14**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №15**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №16**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №17**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №18**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №19**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №20**

Тема: Отладка и тестирование программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №21**

Тема: Отладка и тестирование программного обеспечения

ПРАКТИЧЕСКАЯ РАБОТА №22

Тема: Документирование

ПРАКТИЧЕСКАЯ РАБОТА №23

Тема: Документирование

ПРАКТИЧЕСКАЯ РАБОТА №24

Тема: Документирование

ПРАКТИЧЕСКАЯ РАБОТА №25

Тема: Документирование

ПРАКТИЧЕСКАЯ РАБОТА №26

Тема: Документирование

ПРАКТИЧЕСКАЯ РАБОТА №27

Тема: Документирование

## **ИНСТРУКЦИИ ДЛЯ ОБУЧАЮЩИХСЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ**

Прежде чем приступить к выполнению заданий, внимательно прочитайте данные рекомендации. Практические работы включают в себя задания следующих видов:

### **Выполнение тестовых заданий**

Для проверки и последующего анализа своих знаний Вам предлагается пройти тестовые задания. Выбор заданий осуществляется тестирующей системой случайным образом.

Тестовые задания интерактивны. По структуре формирования ответа различают следующие типы заданий:

- тесты восстановления соответствия - предусматривают восстановление соответствия между одинаковыми по величине, но различными по записи числами.
- тесты восстановления порядка - предусматривают расстановку чисел в соответствие с указанным порядком.
- тесты единственного выбора - предусматривают выбор одного правильного ответа из нескольких предложенных вариантов,
- тесты открытого типа - предусматривают ввод текстовых данных.

При вводе ответа необходимо соблюдать следующие правила:

- курсор нужно поместить в окно для ввода,
- вписывать слова нужно без сокращения,
- вписывать числовые выражения нужно без пробелов, строго следуя образцу, приведенному в задании.

Несоблюдение правил выполнения тестов открытого типа приведет к обозначению ответа как неверного.

Перед выполнением задания внимательно прочитайте его формулировку и предлагаемые варианты ответа. Отвечайте только после того, как Вы поняли вопрос и проанализировали все варианты ответа.

Выполняйте задания в том порядке, в котором они представлены в тесте. Выбор правильных ответов осуществляется путем выбора правильных ответов из списка.

Тестовые задания оцениваются в баллах. Все вопросы имеют свое балльное значение, что определяется, в первую очередь, сложностью самого вопроса.

Постарайтесь выполнить как можно больше заданий и набрать наибольшее количество баллов. По завершении тестирования баллы суммируются.

После выполнения тестовых заданий обязательно сохраните Ваши ответы и предоставьте их учителю.

### **Создание презентаций**

#### **ПРАВИЛА ПОСТРОЕНИЯ СОДЕРЖАНИЯ.**

Правило 1. Содержание должно быть структурировано.

Содержание презентации должно быть четко структурировано: каждый новый слайд должен логически вытекать из предыдущего и одновременно подготавливать появление следующего. Лучший способ проверить, правильно ли построена презентация, — быстро прочитать только заголовки. Если после этого станет ясно, о чем презентация — значит, структура построена верно.

**Правило 2. Краткость — сестра убедительности.**

После того как содержание презентации собрано, с ним следует аккуратно поработать, сократив его насколько возможно. Оптимальным объемом презентации считается 24 традиционных слайда, если презентация уместится в 16 слайдов — еще лучше, ну а 12 и менее слайдов — это то, что редко встречается и крепко запоминается. В среднем, один слайд - это 1,5 минуты выступления.

## **ПРАВИЛА СОЗДАНИЯ СЛАЙДОВ.**

**Правило 1. Думать о зрителе.**

При разработке формы презентации всегда следует думать о том, как зритель ее будет видеть. В первую очередь нужно решить, где зрители будут смотреть вашу презентацию: на бумаге, экране монитора или на большом экране с помощью проектора. На конкурс вы создаете презентации для экрана монитора! И возможно, вашу презентацию захотят распечатать. Это следует учитывать при выборе размера и цвета шрифтов.

**Правило 2. Последовательность и единство оформления.**

Все однотипные элементы должны всегда быть в одном месте: если зритель знает, где ждать заголовков, а где график, он лучше схватывает суть дела. Заголовок – всегда в одном месте экрана. График – всегда в одном месте экрана. И т.д. Однотипные подписи – одинакового цвета и размера. И т.д.

**Правило 3. Нет тексту!**

«Нет» любому тексту, кроме абсолютно необходимого. Читать страницу за страницей и запоминать текст совсем непросто. Количество текста на слайдах должно составить не более 35% от всего содержимого слайдов. Весь ненужный текст следует оставить либо для устного выступления (для текста доклада, т.к. у нас заочная конференция), либо заменить его графиками, картинками и т.д.

## **ВАЖНЫЕ ЗАПРЕТЫ.**

1. Изображения и текст на слайдах не должны быть мелкими (даже если презентация готовится для экрана).

2. Если презентация будет цветной, то следует избегать ярких, так называемых чистых тонов — алого, ярко-синего, зеленого, фиолетового (они режут глаз). Такие краски следует зарезервировать для выделения действительно ключевых моментов, а для рядовых изображений использовать пастельные тона и контрастные сочетания цветов шрифта и фона.

3. Пестрота на экране (больше четырех цветов одновременно).

4. Самый главный запрет - спецэффекты. Анимации наподобие вращающихся заголовков, переворачивающихся слайдов, любые звуки - все это лишь отвлекает слушателей и необоснованно растягивает время презентации.

## ОСНОВНЫЕ ПРАВИЛА ВЫСТУПЛЕНИЯ.

Презентация состоит из двух частей: демонстрация слайдов и сопровождение их текстом. Слайды — поддержка выступления, а не наоборот. Очень часто докладчик вместо выступления просто зачитывает текст на слайдах. Таких ораторов слушатели не уважают, текст они могут и сами прочитать.

Именно поэтому на конкурс мы обязательно требуем **ТЕКСТ ДОКЛАДА**.

**Правило 1.** Стройте выступление на аргументах, а не на слайдах.

Если презентация сделана правильно и текст хорошо сбалансирован другими визуальными элементами, то все равно не следует вести свою аудиторию по презентации, как экскурсовод туристов: «посмотрите налево, посмотрите направо». Презентер должен вести аудиторию не от слайда к слайду, а от тезиса к аргументу, от аргумента к примеру, от вывода к выводу. Нельзя говорить «перейдем на страницу 7», надо — «как именно мы решаем эту проблему, рассказывается на слайде 7». Нельзя говорить «посмотрите на следующий слайд», надо «и что же из этого следует? А вот что!» - и показываем слайд.

**Правило 2.** Готовьтесь к выступлению.

Выступление должно быть подготовлено, прорепетировано и отхронометрировано (подогнано под временные рамки).

**Правило 3.** Помните, что аудитория — это живые люди. Позволяйте себе эмоции.

Позволяйте себе в тексте восклицательные знаки. Текст вовсе не должен быть сухим! Вы не диктор ТВ, вы живой человек, который свято верит в то, о чем он рассказывает

## **Работа за компьютером**

При любой работе должны соблюдаться определённые правила поведения и безопасности, чтобы сохранить своё здоровье и уберечься от возможных травм или каких-либо заболеваний. Профилактика лучше лечения, поэтому правила работы за компьютером необходимо знать всем, ведь мы всё больше и больше времени проводим именно за компьютером — за ним сидим на работе, и за ним же сидим дома.

Памятка ниже будет весьма полезна для людей всех возрастных категорий, чья жизнь или работа напрямую связана с ПК и на компьютере приходится долго и часто работать.

1. Сидите прямо.
2. Вам должно быть удобно. Но это не значит, что надо подгибать ноги под себя или класть ногу на ногу, сутулиться. Этого делать **НЕЛЬЗЯ!**
3. Верхняя часть монитора должна быть расположена на уровне глаз или чуть ниже, а нижняя чуть ближе к Вам.
4. Расстояние между монитором и глазами должно быть 45-75 см.
5. Освещение должно падать так же как и при писании с левой стороны, свет не должен быть сильно ярким или тусклым.



6. Не забывайте моргать, при моргании глаз омывается слёзной жидкостью и не пересыхает, а пересыхание глаза вредит зрению.

7. Периодически необходима зарядка для глаз, которую можно делать и на работе, и дома.

8. Каждый час работы за компьютером делайте перерыв на 15-20 минут.

9. Можете купить специальные очки для работы за ПК, их можно найти в каждой оптике.

10. Если Вы устали, началось чувство сонливости или тяжести в глазах, Вы не должны продолжать работу!

11. Обязательно каждый день надо проветривать комнату, вытирать пыль, влажная уборка только на пользу пойдёт.

## **Пример разработки технического задания на программный продукт**

### **1. Введение**

Настоящее техническое задание распространяется на разработку программы сортировки одномерного массива методами пузырька, прямого выбора, Шелла и быстрой сортировки, предназначенной для использования школьниками старших классов при изучении курса школьной информатики.

- **2. Основание для разработки**

- 2.1. Программа разрабатывается на основе учебного плана кафедры «Информатика и программное обеспечение вычислительных систем».

- 2.2. Наименование работы:

«Программа сортировки одномерного массива».

- 2.3. Исполнитель: компания Вез180Й.

- 2.4. Соисполнители: нет.

- **3. Назначение**

Программа предназначена для использования школьниками при изучении темы «Обработка одномерных массивов» в курсе «Информатика».

- **4. Требования к программе или программному изделию**

- 4.1. Требования к функциональным характеристикам

- 4.1.1. Программа должна обеспечивать возможность выполнения следующих функций:

- • ввод размера массива и самого массива;
- • хранение массива в памяти;
- • выбор метода сортировки;
- • вывод текстового описания метода сортировки;
- • вывод результата сортировки.

- 4.1.2. Исходные данные:
  - • размер массива, заданный целым числом;
  - • массив.
- 4.1.3. Организация входных и выходных данных

Входные данные поступают с клавиатуры.

Выходные данные отображаются на экране и при необходимости выводятся на печать.

Требования к надежности

Предусмотреть контроль вводимой информации.

Предусмотреть блокировку некорректных действий пользователя при работе с системой.

Требования к составу и параметрам технических средств.

Система должна работать на IBM-совместимых персональных компьютерах.

Минимальная конфигурация:

- • тип процессора. .... Pentium и выше;
- • объем оперативного запоминающего устройства. .... 32 Мб и более;
- • объем свободного места на жестком диске. .... 40 Мб.

Рекомендуемая конфигурация:

- • тип процессора. .... Pentium II 400;
- • объем оперативного запоминающего устройства. .... 128 Мб;
- • объем свободного места на жестком диске. .... 60 Мб.

Требования к программной совместимости.

Программа должна работать под управлением семейства операционных систем Win 32 (Windows 95/98/2000/ME/XP и т. и.).

- 5. Требования к программной документации

• 5.1. Разрабатываемые программные модули должны быть самодокументированы, т. е. тексты программ должны содержать все необходимые комментарии.

• 5.2. Разрабатываемая программа должна включать справочную информацию о работе программы, описания методов сортировки и подсказки учащимся.

- 5.3. В состав сопровождающей документации должны входить:
- 5.3.1. Пояснительная записка на пяти листах, содержащая описание разработки.
- 5.3.2. Руководство пользователя.

### **Пример технического задания на разработку**

#### **1. Введение**

Работа выполняется в рамках проекта «Автоматизированная система оперативно-диспетчерского управления электротеплоснабжением корпусов Московского института».

- 2. Основание для разработки
- 2.1. Основанием для данной работы служит договор № 1234 от 10 марта 2003 г.
- 2.2. Наименование работы:  
«Модуль автоматизированной системы оперативно-диспетчерского управления теплоснабжением корпусов Московского института».
- 2.3. Исполнители: ОАО «Лаборатория создания программного обеспечения».
- 2.4. Соисполнители: нет.
- 3. Назначение разработки

Создание модуля для контроля и оперативной корректировки состояния основных параметров теплообеспечения корпусов Московского института.

- 4. Технические требования
- 4.1. Требования к функциональным характеристикам.
- 4.1.1. Состав выполняемых функций.

Разрабатываемое ПО должно обеспечивать:

- • сбор и анализ информации о расходовании тепла, горячей и холодной воды по данным теплосчетчиков 5А-94 на всех тепловых выходах;
- • сбор и анализ информации с устройств управления системами воздушного отопления и кондиционирования типа РТ1 и РТ2 (разработки кафедры СММЭ и ТЦ);
- • предварительный анализ информации на предмет нахождения параметров в допустимых пределах и сигнализирование при выходе параметров за пределы допуска;
- • выдачу рекомендаций по дальнейшей работе;

- • отображение текущего состояния по набору параметров — циклически постоянно (режим работы круглосуточный), при сохранении периодичности контроля прочих параметров;
- • визуализацию информации по расходу теплоносителя:
  - — текущую, аналогично показаниям счетчиков;
  - — с накоплением за прошедшие сутки, неделю, месяц — в виде почасового графика для информации за сутки и неделю;
  - — суточный расход — для информации за месяц.
- Для устройств управления приточной вентиляцией текущая информация должна содержать номер приточной системы и все параметры, выдаваемые на собственный индикатор.
- По отдельному запросу осуществляются внутренние настройки.
- В конце отчетного периода система должна архивировать данные.
- 4.1.2. Организация входных и выходных данных.
- Исходные данные в систему поступают в виде значений с датчиков, установленных в помещениях института. Эти значения отображаются на компьютере диспетчера. После анализа поступившей информации оператор диспетчерского пункта устанавливает необходимые параметры для устройств, регулирующих отопление и вентиляцию в помещениях. Возможна также автоматическая установка некоторых параметров для устройств регулирования.
- Основной режим использования системы — ежедневная работа.
- 4.2. Требования к надежности.
- Для обеспечения надежности необходимо проверять корректность получаемых данных с датчиков.
- 4.3. Условия эксплуатации и требования к составу и параметрам технических средств.

Для работы системы должен быть выделен ответственный оператор.

Требования к составу и параметрам технических средств уточняются на этапе эскизного проектирования системы.

#### 4.4. Требования к информационной и программной совместимости.

Программа должна работать на платформах Windows 98/

- 1СТ/2000.

#### 4.5. Требования к транспортировке и хранению.

Программа поставляется на лазерном носителе информации.

Программная документация поставляется в электронном и печатном виде.

- 4.6. Специальные требования:

- • программное обеспечение должно иметь дружелюбный интерфейс, рассчитанный на пользователя (в плане компьютерной грамотности) квалификации;
- • ввиду объемности проекта задачи предполагается решать поэтапно, при этом модули ПО, созданные в разное время, должны предполагать возможность наращивания системы и быть совместимы друг с другом, поэтому документация на принятое эксплуатационное ПО должна содержать полную информацию, необходимую для работы программистов с ним;
- • язык программирования — по выбору исполнителя, должен обеспечивать возможность интеграции программного обеспечения с некоторыми видами периферийного оборудования (например, счетчик ЗА-94 и т. п.).

## • 5. Требования к программной документации

Основными документами, регламентирующими разработку будущих программ, должны быть документы Единой Системы Программной Документации (ЕСПД): руководство пользователя, руководство администратора, описание применения.

## 6. Техничко-экономические показатели

Эффективность системы определяется удобством использования системы для контроля и управления основными параметрами теплообеспечения помещений Московского института, а также экономической выгодой, полученной от внедрения аппаратно-программного комплекса.

## 7. Порядок контроля и приемки

После передачи Исполнителем отдельного функционального модуля программы Заказчику последний имеет право тестировать модуль в течение 7 дней. После тестирования Заказчик должен принять работу по данному этапу или в письменном виде изложить причину отказа принятия. В случае обоснованного отказа Исполнитель обязуется доработать модуль.

## 8. Календарный план работ

№ этапа	Название этапа	Сроки этапа	Чем заканчивается этап
1	Изучение предметной области. Проектирование системы. Разработка предложений по реализации системы	01.02.200_ - 28.02.200_	Предложения по работе системы. Акт сдачи-приемки
2	Разработка программного модуля по сбору и анализу информации со счетчиков и устройств управления. Внедрение системы для одного из корпусов МИЭТ	01.03.200_ - 31.08.200_	Программный комплекс, решающий поставленные задачи для пилотного корпуса МИЭТ. Акт сдачи-приемки
3	Тестирование и отладка модуля. Внедрение системы во всех	01.09.200_ — 30.12.200_	Готовая система контроля теплообеспечения МИЭТ,

	корпусах МИЭТ		установленная в диспетчерском пункте. Программная документация. Акт сдачи-приемки работ
--	---------------	--	--

Руководитель работ

## **ПРАКТИЧЕСКАЯ РАБОТА №1**

Тема: Отладка и тестирование программного обеспечения

### **Цель**

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

### **Содержание работы**

#### **1. Организационный момент**

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

#### **2. Постановка темы и цели урока**

#### **3. Повторение изученного материала**

### **Задание:**

#### **1.Тестирование «белым ящиком»**

#### **Порядок выполнения работы**

1. Разработать техническое задание на программный продукт
2. Оформить работу в соответствии с ГОСТ 19.106—78. При оформлении использовать MS Office.
3. Сдать и защитить работу.

#### **Защита отчета по практической работе**

Отчет по практической работе должен состоять из:

1. Постановки задачи.
2. Технического задания на программный продукт.

Защита отчета по практической работе заключается в предъявлении преподавателю полученных результатов (на экране монитора), демонстрации полученных навыков и ответах на вопросы преподавателя.

#### **Контрольные вопросы**

1. Приведите этапы разработки программного обеспечения.
2. Что включает в себя постановка задачи и предпроектные исследования?
3. Перечислите функциональные и эксплуатационные требования к программному продукту.
4. Перечислите правила разработки технического задания.
5. Назовите основные разделы технического задания.

## **ПРАКТИЧЕСКАЯ РАБОТА №2**

Тема: Отладка и тестирование программного обеспечения

**Цель:** Произвести отладку и тестирование программного обеспечения

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

### **Содержание работы**

#### **1. Организационный момент**

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

#### **2. Постановка темы и цели урока**

#### **3. Повторение изученного материала**

**Задание:**

#### **1.Тестирование «белым ящиком»**

### **Порядок выполнения работы**

1. На основе технического задания из практической работы № 1 выполнить анализ функциональных и эксплуатационных требований к программному продукту.
2. Определить основные технические решения (выбор языка программирования, структура программного продукта, состав функций ПП, режимы функционирования) и занести результаты в документ, называемый «Эскизным проектом» (см. приложение 4).
3. Определить диаграммы потоков данных для решаемой задачи.
4. Определить диаграммы «сущность—связь», если программный продукт содержит базу данных.
5. Определить функциональные диаграммы.
6. Определить диаграммы переходов состояний.
7. Определить спецификации процессов.
8. Добавить словарь терминов.
9. Оформить результаты, используя MS Office или MS Visio в виде эскизного проекта.
10. Сдать и защитить работу.

### **Защита отчета по практической работе**

Отчет по практической работе должен состоять из:

1. Постановки задачи.
2. Документа «Эскизный проект», содержащего:
  - выбор метода решения и языка программирования;
  - спецификации процессов;
  - все полученные диаграммы;
  - словарь терминов.

Защита отчета по практической работе заключается в предъявлении преподавателю полученных результатов (на экране монитора), демонстрации полученных навыков и ответах на вопросы преподавателя.



### Контрольные вопросы

1. Назовите этапы разработки программного обеспечения.
2. Что такое жизненный цикл программного обеспечения?
3. В чем заключается постановка задачи и предпроектные исследования?
4. Назовите функциональные и эксплуатационные требования к программному продукту.
5. Перечислите составляющие эскизного проекта.
6. Охарактеризуйте спецификации и модели.

## ПРАКТИЧЕСКАЯ РАБОТА №3

Тема: Отладка и тестирование программного обеспечения

**Цель** Произвести отладку и тестирование программного обеспечения

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

**Содержание работы**

1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

2. Постановка темы и цели урока

3. Повторение изученного материала

**Задание:**

1.Тестирование «белым ящиком»

*Порядок выполнения работы*

1. На основе технического задания из практической работы № 1 и спецификаций из практической работы № 2 разработать уточненные алгоритмы программ, составляющих заданный программный модуль. Использовать метод пошаговой детализации

2. На основе уточненных и доработанных алгоритмов разработать структурную схему программного продукта

3. Разработать функциональную схему программного продукта

4. Представить структурную схему в виде структурных карт Константайна

5. Представить структурную схему в виде структурных карт Джексона

6. Оформить результаты, используя MS Office или MS Visio в виде технического проекта.

7. Сдать и защитить работу.

*Защита отчета по практической работе*

Отчет по практической работе должен состоять из:

1. Структурной схемы программного продукта.

2. Функциональной схемы.

3. Алгоритма программы.

4. Структурной карты Константайна.

5. Структурной карты Джексона.

6. Законченного технического проекта программного модуля.

Защита отчета по практической работе заключается в предъявлении преподавателю полученных результатов (на экране монитора), демонстрации полученных навыков и ответах на вопросы преподавателя.

#### Контрольные вопросы

1. Назовите этапы разработки программного обеспечения.
2. В чем заключается проектирование программного обеспечения?
3. Перечислите составляющие технического проекта.
4. Охарактеризуйте структурный подход к программированию.
5. Из чего состоят структурная и функциональная схемы?
6. Охарактеризуйте метод пошаговой детализации при составлении алгоритмов программ.
7. Приведите понятие псевдокода.
8. В чем заключается методика Константайна?
9. В чем заключается методика Джексона?

## ПРАКТИЧЕСКАЯ РАБОТА №4

Тема: Отладка и тестирование программного обеспечения

**Цель** Произвести отладку и тестирование программного обеспечения

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

**Содержание работы**

1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

2. Постановка темы и цели урока

3. Повторение изученного материала

**Задание:**

1.Тестирование «белым ящиком»

порядок выполнения работы

1. По результатам лабораторных работ № 1—3 написать код программ для решения поставленной задачи на языке программирования, выбранном на этапе эскизного проектирования.

2. Отладить программный модуль.

3. Получить результаты работы.

4. Оформить документацию к разработанному программному обеспечению.

5. Сдать и защитить работу.

Защита отчета по практической работе

Отчет по практической работе должен состоять из:

1. Листингов программ.

2. Интерфейса пользователя.

3. Документации к программному обеспечению (руководство пользователя, руководство системного программиста, руководство программиста, руководство оператора).

4. Результатов работы программ.

Защита отчета по практической работе заключается в предъявлении преподавателю полученных результатов (на экране монитора), демонстрации полученных навыков и ответах на вопросы преподавателя.

Контрольные вопросы

1. В чем состоит этап реализации и отладки программного обеспечения?

2. Какие существуют инструментальные средства разработки?
3. Охарактеризуйте этап стихийного программирования.
4. Охарактеризуйте этапы структурного и модульного программирования.
5. Что такое документация к программному обеспечению?

## **ПРАКТИЧЕСКАЯ РАБОТА №5**

Тема: Отладка и тестирование программного обеспечения

**Цель** Произвести отладку и тестирование программного обеспечения

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

### **Содержание работы**

#### **1. Организационный момент**

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

#### **2. Постановка темы и цели урока**

#### **3. Повторение изученного материала**

**Задание:**

#### **1.Тестирование «белым ящиком»**

**Порядок выполнения работы**

1. Спроектировать тесты по принципу «белого ящика» для программы, разработанной в практической работе № 4. Использовать схемы алгоритмов, разработанные и уточненные в лабораторных работах № 2, 3.

2. Выбрать несколько алгоритмов для тестирования и обозначить буквами или цифрами ветви этих алгоритмов.

3. Выписать пути алгоритма, которые должны быть проверены тестами для выбранного метода тестирования.

4. Записать тесты, которые позволят пройти по путям алгоритма.

5. Протестировать разработанную вами программу. Результаты оформить в виде таблиц

6. Проверить все виды тестов и сделать выводы об их эффективности.

7. Оформить отчет по практической работе.

8. Сдать и защитить работу.

**Защита отчета по практической работе**

**Отчет по практической работе должен состоять из:**

1. Постановки задачи.

2. Блок-схемы программ.

3. Тестов.

4. Таблиц тестирования программы.

5. Выводов по результатам тестирования (не забывайте, что целью тестирования является обнаружение ошибок в программе).

**Контрольные вопросы**

1. Охарактеризуйте этап реализации и тестирования программного продукта.
2. Какие существуют виды тестирования?
3. Назовите критерии выбора тестов.
4. Перечислите свойства тестов.
5. Приведите критерии надежности программ.
6. В чем заключается оценка надежности программ?

## ПРАКТИЧЕСКАЯ РАБОТА №6

Тема: Отладка и тестирование программного обеспечения

**Цель** Произвести отладку и тестирование программного обеспечения

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

### **Содержание работы**

#### 1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

#### 2. Постановка темы и цели урока

#### 3. Повторение изученного материала

Задание:

#### 1.Тестирование «черным ящиком»

### **Тестирование по принципу «черного ящика»**

Одним из способов проверки программ является стратегия тестирования, называемая стратегией "черного ящика" или тестированием с управлением по данным. В этом случае программа рассматривается как "черный ящик" и такое тестирование имеет целью выяснения обстоятельств, в которых поведение программы не соответствует спецификации.

Для обнаружения всех ошибок в программе 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 построение исчерпывающего входного теста для большинства случаев невозможно. Поэтому, обычно выполняется "*разумное*" *тестирование*, при котором тестирование программы ограничивается прогонами на небольшом подмножестве всех возможных входных данных. Естественно при этом целесообразно выбрать наиболее подходящее подмножество (подмножество с наивысшей вероятностью обнаружения ошибок).

Правильно выбранный тест подмножества должен обладать следующими свойствами:

1) уменьшать, причем более чем на единицу число других тестов, которые должны быть разработаны для достижения заранее определенной цели «приемлемого» тестирования:

2) покрывать значительную часть других возможных тестов, что в некоторой степени свидетельствует о наличии или отсутствии ошибок до и после применения этого ограниченного множества значений входных данных.



Стратегия "черного ящика" включает в себя следующие методы формирования тестовых наборов:

- эквивалентное разбиение;
- анализ граничных значений;
- анализ причинно-следственных связей;
- предположение об ошибке.

### **Эквивалентное разбиение**

*Основу метода составляют два положения:*

1. Исходные данные программы необходимо разбить 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 тест какого-либо класса обнаруживает ошибку, то предполагается, что все другие тесты этого класса эквивалентности тоже обнаружат эту ошибку и наоборот
2. Каждый тест должен включать по возможности максимальное количество различных входных условий, что позволяет минимизировать общее число необходимых тестов.

Первое положение используется для разработки набора "интересных" условий, которые должны быть протестированы, а второе - для разработки минимального набора тестов.

Разработка тестов методом эквивалентного разбиения осуществляется в два этапа:

- выделение классов эквивалентности;
- построение тестов.

### **Выделение классов эквивалентности**

Классы эквивалентности выделяются путем выбора каждого входного условия (обычно это предложение или фраза из спецификации) и разбиением его на две или более групп. Для этого используется таблица следующего вида:

Входное условие	Правильные классы эквивалентности	Неправильные классы эквивалентности

Правильные классы включают правильные данные, неправильные классы - неправильные данные.

Выделение классов эквивалентности является эвристическим процессом, однако при этом существует ряд правил:

- Если входные условия описывают *область* 999 и два неправильных  $X \leq X \leq \text{значений}$  (например «целое данное может принимать значения от 1 до 999»), то выделяют один правильный класс  $1 < 1$  и  $X > 999$ .
- Если входное условие описывает *число* значений (например, «в автомобиле могут ехать от одного до шести человек»), то определяется один правильный класс эквивалентности и два неправильных (ни одного и более шести человек).
- Если входное условие описывает множество входных значений и есть основания полагать, что каждое значение программист трактует особо (например, «известные способы передвижения на АВТОБУСЕ, ГРУЗОВИКЕ, ТАКСИ, МОТОЦИКЛЕ или ПЕШКОМ»), то определяется правильный класс эквивалентности для каждого значения и один неправильный класс (например «на ПРИЦЕПЕ»).
- 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).
- Если есть любое основание считать, что различные элементы класса эквивалентности трактуются программой неодинаково, то данный класс разбивается на меньшие классы эквивалентности.

### ***Построение тестов***

Этот шаг заключается в использовании классов эквивалентности для построения тестов. Этот процесс включает в себя:

- Назначение каждому классу эквивалентности уникального номера.
- Проектирование новых тестов, каждый из которых покрывает как можно большее число непокрытых классов эквивалентности, до тех пор, пока все правильные классы не будут покрыты (только не общими) тестами.
- Запись тестов, каждый из которых покрывает один и только один из непокрытых неправильных классов эквивалентности, до тех пор, пока все неправильные классы не будут покрыты тестами.

Разработка индивидуальных тестов для неправильных классов эквивалентности обусловлено тем, что определенные проверки с ошибочными входами скрывают или заменяют другие проверки с ошибочными входами.

Недостатком метода эквивалентных разбиения в том, что он не исследует комбинации входных условий.

### **Анализ граничных значений.**

*Граничные условия* - это ситуации, возникающие на, выше или ниже границ входных классов эквивалентности. Анализ граничных значений отличается от эквивалентного разбиения следующим:

- Выбор любого элемента в классе эквивалентности в качестве представительного при анализе граничных условий осуществляется таким образом, чтобы проверить тестом каждую границу этого класса.
- При разработке тестов рассматриваются не только входные условия (*пространство входов*), но и *пространство результатов*.

Применение метода анализа граничных условий требует определенной степени творчества и специализации в рассматриваемой проблеме. Тем не менее, существует несколько общих правил этого метода:

- Построить тесты для границ области и тесты с неправильными входными данными для ситуаций незначительного выхода за границы области, если входное условие описывает область значений (например, для области входных значений от -1.0 до +1.0 необходимо написать тесты для ситуаций -1.0, +1.0, -1.001 и +1.001).
- Построить тесты для минимального и максимального значений условий и тесты, большие и меньшие этих двух значений, если входное условие удовлетворяет дискретному ряду значений. Например, если входной файл может содержать от 1 до 255 записей, то проверить 0, 1, 255 и 256 записей.
- Использовать правило 1 для каждого выходного условия. Причем, важно проверить границы пространства результатов, поскольку не всегда границы входных областей представляют такой же набор условий, как и границы выходных областей. Не всегда также можно получить результат вне выходной области, но, тем не менее, стоит рассмотреть эту возможность.
- Использовать правило 2 для каждого выходного условия.
- Если вход или выход программы есть упорядоченное множество (например, последовательный файл, линейный список, таблица), то сосредоточить внимание на первом и последнем элементах этого множества.
- Попробовать свои силы в поиске других граничных условий.

Анализ граничных условий, если он применен правильно, является одним из наиболее полезных методов проектирования тестов. Однако следует помнить, что граничные условия могут быть едва уловимы и определение их связано с большими трудностями, что является недостатком этого метода. Второй недостаток связан с тем, что метод анализа граничных условий не позволяет проверять различные сочетания исходных данных.

### **Анализ причинно-следственных связей.**

Метод анализа причинно-следственных связей помогает системно выбирать высокорезультативные тесты. Он дает полезный побочный эффект, позволяя обнаруживать неполноту и неоднозначность исходных спецификаций.

Для использования метода необходимо понимание булевой логики (логических операторов - и, или, не). Построение тестов осуществляется в несколько этапов.

1) Спецификация разбивается на «рабочие» участки, так как таблицы причинно-следственных связей становятся громоздкими при применении метода к большим спецификациям. Например, при тестировании компилятора в качестве рабочего участка можно рассматривать отдельный оператор языка.

2) В спецификации определяются множество причин и множество следствий. *Причина* есть отдельное входное условие или класс эквивалентности входных условий. *Следствие* есть выходное условие или преобразование системы. Каждому причине и следствию приписывается отдельный номер.

3) На основе анализа семантического (смыслового) содержания спецификации строится таблица истинности, в которой последовательно перебираются все возможные комбинации причин и определяются следствия каждой комбинации причин. Таблица снабжается примечаниями, задающими ограничения и описывающими комбинации причин и/или следствий, которые являются невозможными из-за синтаксических или внешних ограничений. Аналогично, при необходимости строится таблица истинности для класса эквивалентности.

Примечание. При этом можно использовать следующие приемы:

- По возможности выделять независимые группы причинно-следственных связей в отдельные таблицы.
- Истина обозначается "1". Ложь обозначается "0". Для обозначения безразличных состояний условий применять обозначение "X", которое предполагает произвольное значение условия (0 или 1).

4) Каждая строка таблицы истинности преобразуется в тест. При этом:

- по возможности следует совмещать тесты из независимых таблиц;
- для классов эквивалентности входных условий дополнительно необходимо

Недостаток метода - неадекватно исследует граничные условия.

### **Предположение об ошибке.**

Часто программист с большим опытом выискивает ошибки "без всяких методов". При этом он подсознательно использует метод "предположение об ошибке". Процедура метода предположения об ошибке в значительной степени основана на интуиции. Основная идея метода состоит в том, чтобы перечислить в некотором списке возможные ошибки или ситуации, в которых они могут появиться, а затем на основе этого списка составить тесты. Другими словами, требуется перечислить те специальные случаи, которые могут быть не учтены при проектировании.

## ПРАКТИЧЕСКАЯ РАБОТА №7

Тема: Отладка и тестирование программного обеспечения

**Цель** Произвести отладку и тестирование программного обеспечения

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

**Содержание работы**

1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

2. Постановка темы и цели урока

3. Повторение изученного материала

**Задание:**

1.Тестирование «черным ящиком»

### **Пример применения методов тестирования «черным ящиком»**

Пусть необходимо выполнить тестирование программы, определяющей точку пересечения двух прямых на плоскости. Попутно, она должна определять параллельность прямой одной из осей координат.

В основе программы лежит решение системы линейных уравнений:

$$Ax + By = C \text{ и } Dx + Ey = F.$$

Используя **метод эквивалентных разбиений**, получаем для всех коэффициентов один правильный класс эквивалентности (коэффициент - вещественное число) и один неправильный (коэффициент - не вещественное число). Откуда можно предложить 7 тестов:

- 1) все коэффициенты - вещественные числа;
- 2)- 7) поочередно каждый из коэффициентов - не вещественное число.

**По методу граничных условий:**

можно считать, что для исходных данных граничные условия отсутствуют (коэффициенты - "любые" вещественные числа);

для результатов - получаем, что возможны варианты: единственное решение, прямые сливаются (множество решений), прямые параллельны (отсутствие решений). Следовательно, можно предложить тесты, с результатами внутри области:

- |    |  |
|----|--|
| 1. | $\Delta \neq 0$ ; результат - единственное решение ( $x \Delta \neq y \Delta = 0$ );     |
| 2. | $\Delta = 0$ и результат - множество решений ( $x \Delta = y \Delta = 0$ );              |
| 3. | $\Delta = 0$ , но результат - отсутствие решений ( $x \Delta 0$ или $y \Delta \neq 0$ ); |

и с результатами на границе:

1.  $\delta = 0,01; \delta$
2.  $\delta = -0,01; \delta$
3.  $\delta = 0, \delta_x \delta = 0,01, y = 0;$
4.  $\delta = 0, \delta_y \delta = -0,01, x = 0.$

По методу анализа причинно-следственных связей:

Определяем множество условий.

а) для определения типа прямой:

$$a = 0 \square$$

$$b = 0 \square$$

$c = 0 \square$  - для определения типа и существования первой прямой;

$$d = 0 \square$$

$$e = 0 \square$$

$f = 0 \square$  - для определения типа и существования второй прямой;

б) для определения точки пересечения:

$$= 0 \delta$$

$$\delta_x = 0$$

$$\delta_y = 0$$

Выделяем три группы причинно-следственных связей (определение типа и существования первой линии, определение типа и существования второй линии, определение точки пересечения) и строим таблицы истинности.

$=0$	$=0$	$=0$	Результат
			прямая общего положения
			прямая, параллельная оси ОХ
			ось ОХ
			прямая, параллельная оси ОУ
			ось ОУ
			множество точек

			ПЛОСКОСТИ
--	--	--	-----------

Такая же таблица строится для второй прямой.

$= 0\delta$	$x = 0$	$y = 0$	Е д. реш.	М н.реш.	Г еш. нет
			1	0	0
			0	0	1
			0	0	1
			0	1	0

Каждая строка этих таблиц преобразуется в тест. При возможности (с учетом независимости групп) берутся данные, соответствующие строкам сразу двух или всех трех таблиц.

В результате к уже имеющимся тестам добавляются:

1. проверки всех случаев расположения обеих прямых - 6 тестов по первой прямой вкладываются в 6 тестов по второй прямой так, чтобы варианты не совпадали, - 6 тестов;
2. выполняется отдельная проверка несовпадения условия  $x\delta = 0$  или  $y = 0$  (в зависимости от того, какой тест был выбран по методу граничных условий) - тест также можно совместить с предыдущими 6 тестами;

По методу предположения об ошибке добавим тест:  
все коэффициенты - нули.

Всего получили 20 тестов по всем четырем методикам. Если еще попробовать вложить независимые проверки, то возможно число тестов можно еще сократить.

## ПРАКТИЧЕСКАЯ РАБОТА №8

Тема: Отладка и тестирование программного обеспечения

**Цель** Произвести отладку и тестирование программного обеспечения

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

### **Содержание работы**

#### 1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

#### 2. Постановка темы и цели урока

#### 3. Повторение изученного материала

Задание:

#### 1.Тестирование «черным ящиком»

### **Общая стратегия тестирования.**

Все методологии проектирования тестов могут быть объединены в общую стратегию. Это оправдано тем, что каждый метод обеспечивает создание определенного набора тестов, но ни один из них сам по себе не может дать полный набор тестов. Приемлемая стратегия состоит в следующем:

1. Если спецификация состоит из комбинации входных условий, то начать рекомендуется с применения метода функциональных диаграмм.
2. В любом случае необходимо использовать анализ граничных значений.
3. Определить правильные и неправильные классы эквивалентности для входных и выходных данных и дополнить, если это необходимо, тесты, построенные на предыдущих шагах.
4. Для получения дополнительных тестов рекомендуется использовать метод предположения об ошибке.

### **Порядок выполнения работы:**

1. Ознакомиться с теоретическими сведениями по стратегиям тестирования.
2. В соответствии с вариантом задачи, подготовить тесты по методикам стратегии "черного ящика".
3. Предлагаемые тесты свести в таблицу.

Номер	Назначение	Значения	Ожидаемый	Реакция	Вывод
-------	------------	----------	-----------	---------	-------



теста	теста	исходных данных	результат	программы	
-------	-------	--------------------	-----------	-----------	--

4. Разработать программу.
5. Выполнить тестирование. Занести в таблицу результаты.
6. Сделать вывод о роли тестирования с использованием стратегии "черного ящика" и возможностях его применения. Сформулировать его достоинства и недостатки.

## ПРАКТИЧЕСКАЯ РАБОТА №9

Тема: Отладка и тестирование программного обеспечения

### Цель

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:** 1, 2.

### Содержание работы

#### 1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

#### 2. Постановка темы и цели урока

#### 3. Повторение изученного материала

Задание:

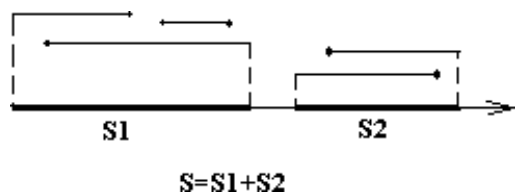
#### 1. Тестирование «черным ящиком»

#### Задача 1.

Разработать программу решения уравнения  $ax^2 + bx + c = 0$ , где  $a, b, c$  - любые вещественные числа.

#### Задача 2.

Разработать программу определения суммарной длины тени, которую отбрасывают на ось  $OX$  отрезки, параллельные этой оси и заданные координатами  $x$  начала и конца отрезка:



#### Задача 3.

Разработать программу исследования уравнений второго порядка с двумя неизвестными  $Ax^2 + 2Bxy + Cy^2 + 2Dx + 2Ey + F = 0$ . Программа должна определять вид графика: эллипс, парабола, гипербола, две пересекающиеся прямые, две параллельные прямые, две мнимые прямые.

Примечание. Вид прямой второго порядка определяется по двум дискриминантам

$$\Delta = \begin{vmatrix} A & B & D \\ B & C & E \\ D & E & F \end{vmatrix} \quad \text{и малому} \quad \delta = \begin{vmatrix} A & B \\ B & C \end{vmatrix}.$$

большому:

Малый дискриминант для эллипса положителен, для гиперболы отрицателен, для параболы равен нулю. Если большой дискриминант равен нулю, то линия второго порядка распадается на две прямых:

для эллиптического вида - пересекающиеся мнимые прямые (точка), для гиперболического вида - пара пересекающихся действительных прямых, для параболического вида - две параллельные прямые.

## ПРАКТИЧЕСКАЯ РАБОТА №10

Тема: Отладка и тестирование программного обеспечения

**Цель** Произвести отладку и тестирование программного обеспечения

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

**Содержание работы**

1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

2. Постановка темы и цели урока

3. Повторение изученного материала

**Задание:**

1.Тестирование «черным ящиком»

*Задача 1.*

Разработать программу определения вида треугольника, заданного длинами его сторон: равносторонний, равнобедренный, прямоугольный, разносторонний.

*Задача 2.*

Разработать программу определения вида четырехугольника, заданного координатами вершин на плоскости: квадрат, прямоугольник, параллелограмм, ромб, равнобедренная трапеция, прямоугольная трапеция, трапеция общего вида, четырехугольник общего вида.

*Задача 2.*

Разработать программу, определяющую взаимное расположение прямых в пространстве: параллельны, пересекаются, скрещиваются и отдельно, расположение каждой прямой (параллельна оси, перпендикулярна плоскости или общего расположения). Прямые задаются координатами двух точек.

Примечание. Две прямые лежат в одной плоскости, если

$$\begin{vmatrix} x_1^2 - x_1^1 & y_1^2 - y_1^1 & z_1^2 - z_1^1 \\ l^1 & m^1 & n^1 \\ l^2 & m^2 & n^2 \end{vmatrix} = 0, \text{ прямые параллельны если } \frac{l^1}{l^2} = \frac{m^1}{m^2} = \frac{n^1}{n^2},$$

где  $l = x_2 - x_1$ ,  $m = y_2 - y_1$ ,  $n = z_2 - z_1$  (верхний индекс соответствует номеру прямой).

## ПРАКТИЧЕСКАЯ РАБОТА №11

Тема: Отладка и тестирование программного обеспечения

**Цель** Произвести отладку и тестирование программного обеспечения

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

### **Содержание работы**

#### 1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

#### 2. Постановка темы и цели урока

#### 3. Повторение изученного материала

Задание:

#### 1. Модульное тестирование

### **Юнит-тестирование**

Модульное тестирование, или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок. В данной работе мы будем использовать юнит-тесты для проверки функциональных требований программы.

Для использования юнит-тестов будем использовать JUnit. JUnit — библиотека для модульного тестирования программного обеспечения на языке Java. Скачать свежую версию данной библиотеки можно по адресу .

Пример юнит-теста, проверяющего равенство  $2+2=4$ , приведен в листинге 4.1.

Листинг 4.1 – пример юнит-теста

```
import org.junit.Test;
import junit.framework.Assert;

public class MathTest {
    @Test
    public void testEquals() {
        Assert.assertEquals(4, 2 + 2);
        Assert.assertTrue(4 == 2 + 2);
    }
}
```

```
}  
@Test  
public void testNotEquals() {  
    Assert.assertFalse(5 == 2 + 2);  
}  
}
```

Для модульного тестирования необходимо использовать драйверы и заглушки.

Unit (Элемент) — наименьший компонент, который можно скомпилировать.

Драйверы — модули тестов, которые запускают тестируемый элемент.

Заглушки — заменяют недостающие компоненты, которые вызываются элементом и выполняют следующие действия:

- возвращаются к элементу, не выполняя никаких других действий;
- отображают трассировочное сообщение и иногда предлагают тестеру продолжить тестирование;
- возвращают постоянное значение или предлагают тестеру самому ввести возвращаемое значение;
- осуществляют упрощенную реализацию недостающей компоненты;
- имитируют исключительные или аварийные условия.

### **Предварительная подготовка к работе**

Так как алгоритм реализован на языке Java, то юнит-тесты следует писать с использованием библиотеки JUnit (). Допускается использовать любую удобную среду разработки (IDEA / Eclipse / др.).

Для упрощения задачи, предлагается выполнить интеграционное тестирование по восходящему подходу. При использовании этого метода подразумевается, что сначала тестируются все программные модули, входящие в состав системы, и только затем они объединяются для интеграционного тестирования. При таком подходе значительно упрощается локализация ошибок: если модули протестированы по отдельности, то ошибка при их совместной работе есть проблема их интерфейса.

Несмотря на то, что интеграционные тесты не являются юнит-тестами в классическом понимании, при выполнении задач практической работы необходимо использовать библиотеку JUnit и написать модуль юнит-тестирования вручную (используя Java-аннотации @Test и документацию библиотеки JUnit).

Реализовать юнит-тесты следует так, чтобы они выполнял функциональное тестирование программы сортировки в соответствии с целями работы.

## JUnit

Таблица 1. Аннотации.

Аннотация	Описание
@Test public void method()	Помечает метод как тестовый
@Before public void method()	Метод будет выполняться перед каждым тестом, может быть использован для подготовки тестового окружения (инициализации, чтения данных)
@After public void method()	Метод будет выполняться после каждого теста. Может использоваться для отчистки окружения, удаления временных данных.

Таблица 2. Тестовые методы

Сигнатура	Описание
fail(String)	Вызывает сбой. Может быть использован, чтобы удостовериться, что определенная часть кода не достигнута или пока тестовый метод не реализован.
assertTrue(true) / assertFalse(false)	Постоянно будет true / false. Может быть использован, чтобы предопределить результат теста, пока он не реализован.
assertEquals([String message], expected, actual)	Проверяет, равны ли две величины. Важно: для массивов проверяется ссылка а не содержимое.
assertEquals([String message], expected, actual, tolerance)	Проверяет, совпадают ли величины float и double
assertNull([message], object)	Проверяет, что объект null.
assertNotNull([message], object)	Проверяет, что объект не null.
assertSame([String], expected, actual)	Проверяет, что обе переменные ссылаются на один объект.
assertNotSame([String], expected, actual)	Проверяет, что переменные ссылаются на разные объекты.
assertTrue([message], boolean condition)	Проверяет условие на истинность.

## Порядок выполнения работы

1. Откройте выбранную IDE и создайте проект на основе существующих программных кодов, реализующих алгоритм пирамидальной сортировки.

2. Подключите к проекту библиотеку JUnit.

3. Создайте каркас для юнит-тестов (Например, в IDE Eclipse можно выбрать нужный класс, открыть контекстное меню, и выбрать New->JUnit test case, в появившемся диалоговом окне выбрать методы, для которых понадобятся юнит-тесты).

4. Создайте юнит-тест согласно описанным требованиям.

5. Отладьте и запустите юнит-тест.

6. Оцените результаты выполнения юнит-тестирования и сделайте соответствующие выводы.

### **Содержание отчёта**

1. Необходимо представить исходный код, описание и результаты работы юнит-теста на проверку правильности интеграции модулей системы.

2. Изложить вводы по результатам тестирования и проделанной практической работе.



## ПРАКТИЧЕСКАЯ РАБОТА №12

Тема: Отладка и тестирование программного обеспечения

**Цель** Произвести отладку и тестирование программного обеспечения

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

### **Содержание работы**

#### 1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

#### 2. Постановка темы и цели урока

#### 3. Повторение изученного материала

Задание:

#### 1.Модульное тестирование

## Выполнение работы

Текст TestGenericDAO

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using NHibernate;
using NHibernate.Criterion;
using System.Collections.Generic;
using FluentNHibernate;
using FluentNHibernate.Cfg;
using FluentNHibernate.Cfg.Db;
using FluentNHibernate.Automapping;
using NHibernate.Tool.hbm2ddl;
using System.Reflection;
using FluentNHibernate.Mapping;
using NHibernate.Cfg;
using lab6.mapping;
using lab6.dao;
using lab6.domain;
namespace lab6
{
```

```

[TestClass()]
public abstract class TestGenericDAO<T> where T : EntityBase
{
    protected static ISessionFactory factory;
    protected static ISession session;
    protected DaoFactory daoFactory;
    protected TestContext testContextInstance;
    /** DAO that will be tested */
    protected IGenericDAO<T> dao = null;
    /** First entity that will be used in tests */
    protected T entity1 = null;
    /** Second entity that will be used in tests */
    protected T entity2 = null;
    /** Third entity that will be used in tests */
    protected T entity3 = null;
    public TestGenericDAO()
    {
        session = openSession("localhost", 5432, "market",
            "postgres", "06031992");
    }
    public TestContext TestContext
    {
        get
        {
            return testContextInstance;
        }
        set
        {
            testContextInstance = value;
        }
    }
    /**Getting dao this test case works with*/
    public IGenericDAO<T> getDAO()
    {

```

```

return dao;
}
/*Setting dao this test case will work with*/
public void setDAO(IGenericDAO<T> dao)
{
this.dao = dao;
}
[ClassCleanup]
public static void ClassCleanup()
{
session.Close();
}
[TestInitialize]
public void TestInitialize()
{
Assert.IsNotNull(dao,
"Please, provide IGenericDAO implementation in constructor");
createEntities();
Assert.IsNotNull(entity1, "Please, create object for entity1");
Assert.IsNotNull(entity2, "Please, create object for entity2");
Assert.IsNotNull(entity3, "Please, create object for entity3");
checkAllPropertiesDiffer(entity1, entity2);
checkAllPropertiesDiffer(entity1, entity3);
checkAllPropertiesDiffer(entity2, entity3);
saveEntitiesGeneric();
}
[TestCleanup]
public void TestCleanup()
{
try
{
if ((entity1 = dao.GetById(entity1.Id)) != null)
dao.Delete(entity1);
}
}

```

```

catch (Exception)
{
    Assert.Fail("Problem in cleanup method");
}
try
{
    if ((entity2 = dao.GetById(entity2.Id)) != null)
        dao.Delete(entity2);
}
catch (Exception)
{
    Assert.Fail("Problem in cleanup method");
}
try
{
    if ((entity3 = dao.GetById(entity3.Id)) != null)
        dao.Delete(entity3);
}
catch (Exception)
{
    Assert.Fail("Problem in cleanup method");
}
entity1 = null;
entity2 = null;
entity3 = null;
}
[TestMethod]
public void TestGetByIdGeneric()
{
    T foundObject = null;
    // Should not find with inexistent id
    try
    {
        long id = DateTime.Now.ToFileTime();
    }

```

```

foundObject = dao.GetById(id);
Assert.IsNull(foundObject, "Should return null if id is inexistent");
}
catch (Exception)
{
Assert.Fail("Should return null if object not found");
}
// Getting all three entities
getEntityGeneric(entity1.Id, entity1);
getEntityGeneric(entity2.Id, entity2);
getEntityGeneric(entity3.Id, entity3);
}
[TestMethod]
public void TestGetAllGeneric()
{
List<T> list = getListOfAllEntities();
Assert.IsTrue(list.Contains(entity1),
"After dao method GetAll list should contain entity1");
Assert.IsTrue(list.Contains(entity2),
"After dao method GetAll list should contain entity2");
Assert.IsTrue(list.Contains(entity3),
"After dao method GetAll list should contain entity3");
}
[TestMethod]
public void TestDeleteGeneric()
{
try
{
dao.Delete((T)null);
Assert.Fail("Should not delete entity will null id");
}
catch (Exception)
{
}
}

```

```

// Deleting second entity
try
{
dao.Delete(entity2);
}
catch (Exception)
{
Assert.Fail("Deletion should be successful of entity2");
}
// Checking if other two entities can be still found
getEntityGeneric(entity1.Id, entity1);
getEntityGeneric(entity3.Id, entity3);
// Checking if entity2 can not be found
try
{
T foundEntity = null;
foundEntity = dao.GetById(entity2.Id);
Assert.IsNull(foundEntity,
"After deletion entity should not be found with id " + entity2.Id);
}
catch (Exception)
{
Assert.Fail("Should return null if finding the deleted entity");
}
// Checking if other two entities can still be found in getAll list
List<T> list = getListOfAllEntities();
Assert.IsTrue(list.Contains(entity1),
"After dao method GetAll list should contain entity1");
Assert.IsTrue(list.Contains(entity3),
"After dao method GetAll list should contain entity3");
}
protected abstract void createEntities();
protected abstract void checkAllPropertiesDiffer(T entityToCheck1,
T entityToCheck2);

```

```

protected abstract void checkAllPropertiesEqual(T entityToCheck1,
T entityToCheck2);
protected void saveEntitiesGeneric()
{
T savedObject = null;
try
{
dao.SaveOrUpdate(entity1);
savedObject = getPersistentObject(entity1);
Assert.IsNotNull(savedObject,
"DAO method saveOrUpdate should return entity if successfull");
checkAllPropertiesEqual(savedObject, entity1);
entity1 = savedObject;
}
catch (Exception)
{
Assert.Fail("Fail to save entity1");
}
try
{
dao.SaveOrUpdate(entity2);
savedObject = getPersistentObject(entity2);
Assert.IsNotNull(savedObject,
"DAO method saveOrUpdate should return entity if successfull");
checkAllPropertiesEqual(savedObject, entity2);
entity2 = savedObject;
}
catch (Exception)
{
Assert.Fail("Fail to save entity2");
}
try
{
dao.SaveOrUpdate(entity3);

```

```

savedObject = getPersistentObject(entity3);
Assert.IsNotNull(savedObject,
"DAO method saveOrUpdate should return entity if successfull");
checkAllPropertiesEqual(savedObject, entity3);
}
catch (Exception)
{
Assert.Fail("Fail to save entity3");
}
}
protected T getPersistentObject(T nonPersistentObject)
{
ICriteria criteria =
session.CreateCriteria(typeof(T)).Add(Example.Create(nonPersistentObject));
IList<T> list = criteria.List<T>();
Assert.IsTrue(list.Count >= 1,
"Count of grups must be equal or more than 1");
return list[0];
}
protected void getEntityGeneric(long id, T entity)
{
T foundEntity = null;
try
{
foundEntity = dao.GetById(id);
Assert.IsNotNull(foundEntity,
"Service method getEntity should return entity if successfull");
checkAllPropertiesEqual(foundEntity, entity);
}
catch (Exception)
{
Assert.Fail("Failed to get entity with id " + id);
}
}
protected List<T> getListOfAllEntities()

```



```

{
List<T> list = null;
// Should get not null and not empty list
try
{
list = dao.GetAll();
}
catch (Exception)
{
Assert.Fail(
"Should be able to get all entities that were added before");
}
Assert.IsNotNull(list,
"DAO method GetAll should return list of entities if successfull");
Assert.IsFalse(list.Count == 0,
"DAO method should return not empty list if successfull");
return list;
}
//Метод открытия сессии
public static ISession openSession(String host, int port,
String database, String user, String passwd)
{
ISession session = null;
if (factory == null)
{
FluentConfiguration configuration = Fluently.Configure()
.Database(PostgreSQLConfiguration
.PostgreSQL82.ConnectionString(c => c
.Host(host)
.Port(port)
.Database(database)
.Username(user)
.Password(passwd)))
.Mappings(m => m.FluentMappings.Add<TovarsMap>().Add<ProdMap>())

```

```

.ExposeConfiguration(BuildSchema);
factory = configuration.BuildSessionFactory();
}
//Открытие сессии
session = factory.OpenSession();
return session;
}
//Метод для автоматического создания таблиц в базе данных
private static void BuildSchema(Configuration config)
{
    new SchemaExport(config).Create(false, true);
}
}
}
}

```

Текст TestProdDAO

```

using System;
using System.Collections.Generic;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using lab6.domain;
using lab6.dao;
namespace lab6
{
    [TestClass]
    public class TestProdDAO : TestGenericDAO<Prod>
    {
        protected IProdDAO prodDAO = null;
        protected Tovars tovar1 = null;
        protected Tovars tovar2 = null;
        protected Tovars tovar3 = null;
        public TestProdDAO()
        : base()
        {
            DaoFactory daoFactory = new NHibernateDAOFactory(session);
            prodDAO = daoFactory.getProdDAO();
        }
    }
}

```

```

setDAO(prodDAO);
}
protected override void createEntities()
{
    entity1 = new Prod();
    entity1.NameProd = "1";
    entity1.Surname = "1";
    entity1.God = 1992;
    entity2 = new Prod();
    entity2.NameProd = "2";
    entity2.Surname = "2";
    entity2.God = 1993;
    entity3 = new Prod();
    entity3.NameProd = "3";
    entity3.Surname = "3";
    entity3.God = 1994;
}
protected override void checkAllPropertiesDiffer(Prod entityToCheck1,
Prod entityToCheck2)
{
    Assert.AreNotEqual(entityToCheck1.NameProd, entityToCheck2.NameProd,
"Values must be different");
    Assert.AreNotEqual(entityToCheck1.Surname, entityToCheck2.Surname,
"Values must be different");
    Assert.AreNotEqual(entityToCheck1.God, entityToCheck2.God, "Values
must be different");
}
protected override void checkAllPropertiesEqual(Prod entityToCheck1,
Prod entityToCheck2)
{
    Assert.AreEqual(entityToCheck1.NameProd, entityToCheck2.NameProd,
"Values must be equal");
    Assert.AreEqual(entityToCheck1.Surname, entityToCheck2.Surname,
"Values must be equal");
    Assert.AreEqual(entityToCheck1.God, entityToCheck2.God,

```

```

"Values must be equal");
}
[TestMethod]
public void TestGetByIdProd()
{
base.TestGetByIdGeneric();
}
[TestMethod]
public void TestGetAllProd()
{
base.TestGetAllGeneric();
}
[TestMethod]
public void TestDeleteProd()
{
base.TestDeleteGeneric();
}
// [TestMethod]
// public void TestGetProdByName()
// {
// Prod prod1 = prodDAO.getProdByName(entity1.NameProd);
// Assert.IsNotNull(prod1,
// "Service method getGroupByName should return group if successful");
// Prod prod2 = prodDAO.getProdByName(entity2.NameProd);
// Assert.IsNotNull(prod2,
// "Service method getGroupByName should return group if successful");
// Prod prod3 = prodDAO.getProdByName(entity3.NameProd);
// Assert.IsNotNull(prod3,
// "Service method getGroupByName should return group if successful");
// checkAllPropertiesEqual(prod1, entity1);
// checkAllPropertiesEqual(prod2, entity2);
// checkAllPropertiesEqual(prod3, entity3);
// }
[TestMethod]

```

```

public void TestGetAllTovarsOfProd()
{
    createEntitiesForTovars();
    Assert.IsNotNull(tovar1, "Please, create object for student1");
    Assert.IsNotNull(tovar2, "Please, create object for student2");
    Assert.IsNotNull(tovar3, "Please, create object for student3");
    entity1.TovarsList.Add(tovar1);
    tovar1.Prod = entity1;
    entity1.TovarsList.Add(tovar2);
    tovar2.Prod = entity1;
    entity1.TovarsList.Add(tovar3);
    tovar3.Prod = entity1;
    Prod savedObject = null;
    try
    {
        dao.SaveOrUpdate(entity1);
        savedObject = getPersistentObject(entity1);
        Assert.IsNotNull(savedObject,
            "DAO method saveOrUpdate should return entity if successfull");
        checkAllPropertiesEqual(savedObject, entity1);
        entity1 = savedObject;
    }
    catch (Exception)
    {
        Assert.Fail("Fail to save entity1");
    }
    IList<Tovars> tovarsList =
    prodDAO.getAllTovarsOfProd(entity1.NameProd);
    Assert.IsNotNull(tovarsList, "List can't be null");
    Assert.IsTrue(tovarsList.Count == 3,
        "Count of students in the list must be 3");
    checkAllPropertiesEqualForTovars(tovarsList[0], tovar1);
    checkAllPropertiesEqualForTovars(tovarsList[1], tovar2);
    checkAllPropertiesEqualForTovars(tovarsList[2], tovar3);
}

```

```

}
//[[TestMethod]
//public void TestDelProdByName()
//{
// try
// {
// prodDAO.delProdByName(entity2.NameProd);
// }
// catch (Exception)
// {
// Assert.Fail("Deletion should be successful of entity2");
// }
// Checking if other two entities can be still found
// getEntityGeneric(entity1.Id, entity1);
// getEntityGeneric(entity3.Id, entity3);
// Checking if entity2 can not be found
// try
// {
// Prod foundProd = null;
// foundProd = dao.GetById(entity2.Id);
// Assert.IsNull(foundProd,
// "After deletion entity should not be found with groupName " +
// entity2.NameProd);
// }
// catch (Exception)
// {
// Assert.Fail("Should return null if finding the deleted entity");
// }
// Checking if other two entities can still be found in getAll list
// List<Prod> list = getListOfAllEntities();
// Assert.IsTrue(list.Contains(entity1),
// "After dao method GetAll list should contain entity1");
// Assert.IsTrue(list.Contains(entity3),
// "After dao method GetAll list should contain entity3");

```

```

// }
protected void createEntitiesForTovars()
{
    tovar1 = new Tovars();
    tovar1.Name = "C";
    tovar1.Price = 10;
    tovar1.Ves = 100;
    tovar2 = new Tovars();
    tovar2.Name = "B";
    tovar2.Price = 20;
    tovar2.Ves = 200;
    tovar3 = new Tovars();
    tovar3.Name = "D";
    tovar3.Price = 30;
    tovar3.Ves = 300;
}

protected void checkAllPropertiesEqualForTovars(Tovars entityToCheck1,
Tovars entityToCheck2)
{
    Assert.AreEqual(entityToCheck1.Name, entityToCheck2.Name,
"Values must be equal");
    Assert.AreEqual(entityToCheck1.Price, entityToCheck2.Price,
"Values must be equal");
    Assert.AreEqual(entityToCheck1.Ves, entityToCheck2.Ves,
"Values must be equal");
}
}
}

Текст TestTovarsDAO
using System;
using System.Collections.Generic;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using lab6.domain;
using lab6.dao;

```

```

using NHibernate.Criterion;
using NHibernate;
namespace lab6
{
[TestClass]
public class TestTovarsDAO : TestGenericDAO<Tovars>
{
protected ITovarsDao tovarsDAO = null;
protected IProdDAO prodDAO = null;
protected Prod prod = null;
public TestTovarsDAO()
: base()
{
DaoFactory daoFactory = new NHibernateDAOFactory(session);
tovarsDAO = daoFactory.getTovarsDAO();
prodDAO = daoFactory.getProdDAO();
setDAO(tovarsDAO);
}
protected override void createEntities()
{
entity1 = new Tovars();
entity1.Name = "C";
entity1.Price = 10;
entity1.Ves = 100;
entity2 = new Tovars();
entity2.Name = "B";
entity2.Price = 20;
entity2.Ves = 200;
entity3 = new Tovars();
entity3.Name = "D";
entity3.Price = 30;
entity3.Ves = 300;
}
protected override void checkAllPropertiesDiffer(Tovars entityToCheck1,

```



```

Tovars entityToCheck2)
{
Assert.AreNotEqual(entityToCheck1.Name, entityToCheck2.Name,
"Values must be different");
Assert.AreNotEqual(entityToCheck1.Price, entityToCheck2.Price,
"Values must be different");
Assert.AreNotEqual(entityToCheck1.Ves, entityToCheck2.Ves,
"Values must be different");
}
protected override void checkAllPropertiesEqual(Tovars entityToCheck1,
Tovars entityToCheck2)
{
Assert.AreEqual(entityToCheck1.Name, entityToCheck2.Name,
"Values must be equal");
Assert.AreEqual(entityToCheck1.Price, entityToCheck2.Price,
"Values must be equal");
Assert.AreEqual(entityToCheck1.Ves, entityToCheck2.Ves,
"Values must be equal");
}
[TestMethod]
public void TestGetByIdTovars()
{
base.TestGetByIdGeneric();
}
[TestMethod]
public void TestGetAllTovars()
{
base.TestGetAllGeneric();
}
[TestMethod]
public void TestDeleteTovars()
{
base.TestDeleteGeneric();
}

```

```

[TestMethod]
public void TestGetTovarByProdName()
{
    Prod prod = new Prod();
    prod.NameProd = "1";
    prod.Surname = "1";
    prod.God = 1992;
    prod.TovarsList.Add(entity1);
    entity1.Prod = prod;
    prod.TovarsList.Add(entity2);
    entity2.Prod = prod;
    prod.TovarsList.Add(entity3);
    entity3.Prod = prod;
    Prod savedProd = null;
    try
    {
        prodDAO.SaveOrUpdate(prod);
        savedProd = getPersistentProd(prod);
        Assert.IsNotNull(savedProd,
            "DAO method saveOrUpdate should return group if successfull");
        checkAllPropertiesEqualProd(savedProd, prod);
        prod = savedProd;
    }
    catch (Exception)
    {
        Assert.Fail("Fail to save group");
    }
    getTovarByProdName(entity1, prod.NameProd,
        entity1.Name);
    getTovarByProdName(entity2, prod.NameProd,
        entity2.Name);
    getTovarByProdName(entity3, prod.NameProd,
        entity3.Name);
    prod.TovarsList.Remove(entity1);
}

```

```

prod.TovarsList.Remove(entity2);
prod.TovarsList.Remove(entity3);
entity1.Prod = null;
entity2.Prod = null;
entity3.Prod = null;
prodDAO.Delete(prod);
}
protected void getTovarByProdName(Tovars tovar, string nameProd, string
name)
{
Tovars foundTovars = null;
try
{
foundTovars = tovarsDAO.getTovarByProdName(
nameProd,name);
Assert.IsNotNull(tovarsDAO,
"Service method should return student if successfull");
checkAllPropertiesEqual(foundTovars, tovar);
}
catch (Exception)
{
Assert.Fail("Failed to get student with nameProd " +
nameProd + " name " + name);
}
}
protected Prod getPersistentProd(Prod nonPersistentProd)
{
ICriteria criteria = session.CreateCriteria(typeof(Prod))
.Add(Example.Create(nonPersistentProd));
IList<Prod> list = criteria.List<Prod>();
Assert.IsTrue(list.Count >= 1,
"Count of grups must be equal or more than 1");
return list[0];
}
+protected void checkAllPropertiesEqualProd(Prod entityToCheck1,

```

```
Prod entityToCheck2)
{
Assert.AreEqual(entityToCheck1.NameProd, entityToCheck2.NameProd,
"Values must be equal");
Assert.AreEqual(entityToCheck1.Surname, entityToCheck2.Surname,
"Values must be equal");
Assert.AreEqual(entityToCheck1.God, entityToCheck2.God,
"Values must be equal");
}
}
}
```

## ПРАКТИЧЕСКАЯ РАБОТА №13

Тема: Отладка и тестирование программного обеспечения

### Цель

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

### Содержание работы

#### 1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

#### 2. Постановка темы и цели урока

#### 3. Повторение изученного материала

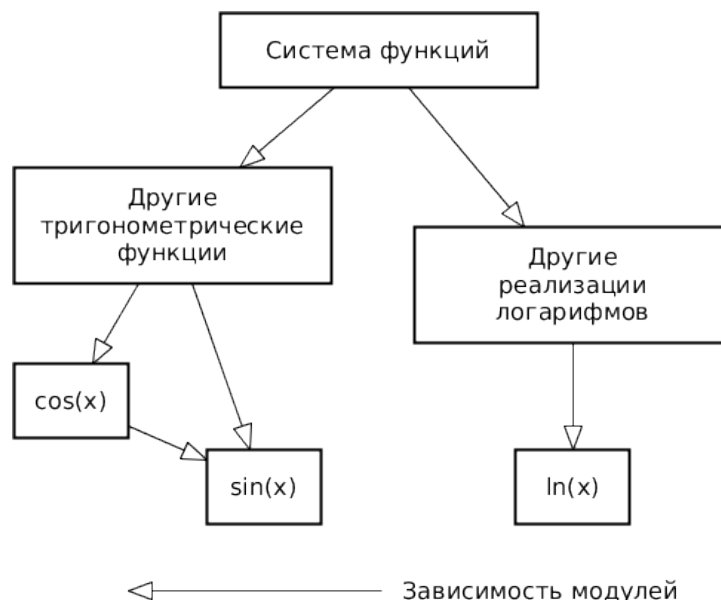
Задание:

#### 1. Модульное тестирование

### Правила выполнения работы:

1. Все составляющие систему функции (как тригонометрические, так и логарифмические) должны быть выражены через базовые (тригонометрическая зависит от варианта; логарифмическая - натуральный логарифм).

2. Структура приложения, тестируемого в рамках практической работы, должна выглядеть следующим образом (пример приведён для базовой тригонометрической функции  $\sin(x)$ ):



3. Обе "базовые" функции (в примере выше -  $\sin(x)$  и  $\ln(x)$ ) должны быть реализованы при помощи разложения в ряд с задаваемой погрешностью. Использовать тригонометрические / логарифмические преобразования для упрощения функций ЗАПРЕЩЕНО.

4. Для КАЖДОГО модуля должны быть реализованы табличные заглушки. При этом, необходимо найти область допустимых значений функций, и, при необходимости, определить взаимозависимые точки в модулях.

5. Разработанное приложение должно позволять выводить значения, выдаваемое любым модулем системы, в csv файл вида «X, Результаты модуля (X)», позволяющее произвольно менять шаг наращивания X. Разделитель в файле csv можно использовать произвольный.

#### **Порядок выполнения работы:**

1. Разработать приложение, руководствуясь приведёнными выше правилами.

2. С помощью JUNIT4 разработать тестовое покрытие системы функций, проведя анализ эквивалентности и учитывая особенности системы функций. Для анализа особенностей системы функций и составляющих ее частей можно использовать сайт <https://www.wolframalpha.com/>.

3. Собрать приложение, состоящее из заглушек. Провести интеграцию приложения по 1 модулю, с обоснованием стратегии интеграции, проведением интеграционных тестов и контролем тестового покрытия системы функций.

#### **Отчёт по работе должен содержать:**

1. Текст задания, систему функций.
2. UML-диаграмму классов разработанного приложения.
3. Описание тестового покрытия с обоснованием его выбора.
4. Графики, построенные csv-выгрузкам, полученным в процессе интеграции приложения.
5. Выводы по работе.

#### **Вопросы к защите практической работы:**

1. Цели и задачи интеграционного тестирования. Расположение фазы интеграционного тестирования в последовательности тестов; предшествующие и последующие виды тестирования ПО.

2. Алгоритм интеграционного тестирования.

3. Концепции и подходы, используемые при реализации интеграционного тестирования.

4. Программные продукты, используемые для реализации интеграционного тестирования. Использование JUnit для интеграционных тестов.

5. Автоматизация интеграционных тестов. ПО, используемое для автоматизации интеграционного тестирования.



## ПРАКТИЧЕСКАЯ РАБОТА №14

Тема: Отладка и тестирование программного обеспечения

**Цель** Произвести отладку и тестирование программного обеспечения

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

### **Содержание работы**

#### 1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

#### 2. Постановка темы и цели урока

#### 3. Повторение изученного материала

Задание:

#### 1.Модульное тестирование

Сформировать варианты использования, разработать на их основе тестовое покрытие и провести функциональное тестирование интерфейса сайта (в соответствии с вариантом).

**Вариант №2:** YouTube. YouTube is a way to get your videos to the

people who matter to you. - <http://www.youtube.com>

### **Требования к выполнению работы:**

1. Тестовое покрытие должно быть сформировано на основании набора прецедентов использования сайта.
2. Тестирование должно осуществляться автоматически - с помощью системы автоматизированного тестирования Selenium.
3. Шаблоны тестов должны формироваться при помощи Selenium IDE и исполняться при помощи Selenium RC в браузерах Firefox и Chrome.
4. Предполагается, что тестируемый сайт использует динамическую генерацию элементов на странице, т.е. выбор элемента в DOM должен осуществляться не на основании его ID, а с помощью XPath.

### **Требования к содержанию отчёта:**

1. Текст задания.
2. UseCase-диаграмму с прецедентами использования тестируемого сайта.



3. CheckList тестового покрытия.
4. Описание набора тестовых сценариев.
5. Результаты тестирования.
6. Выводы.

**Вопросы к защите практической работы:**

1. Функциональное тестирование. Основные понятия, способы организации и решаемые задачи.
2. Система Selenium. Архитектура, принципы написания сценариев, способы доступа к элементам страницы.
3. Язык XPath. Основные конструкции, системные функции, работа с множествами элементов.

## **ПРАКТИЧЕСКАЯ РАБОТА №15**

Тема: Отладка и тестирование программного обеспечения

### **Цель**

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

### **Содержание работы**

#### **1. Организационный момент**

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

#### **2. Постановка темы и цели урока**

#### **3. Повторение изученного материала**

### **Задание:**

#### **1.Модульное тестирование**

Тестирование на основе потока данных. Этот вид тестирования направлен на выявление ссылок на неинициализированные переменные и избыточные присваивания (аномалий потока данных). Предложенная стратегия требовала тестирования всех взаимосвязей, включающих в себя ссылку (использование) и определение переменной, на которую указывает ссылка (т. е. требуется покрытие дуг информационного графа программы). Недостаток стратегии в том, что она не включает критерий C1, и не гарантирует покрытия решений.

Стратегия требуемых пар также тестирует упомянутые взаимосвязи. Использование переменной в предикате дублируется в соответствии с числом выходов решения, и каждая из таких требуемых взаимосвязей должна быть протестирована. К популярным критериям принадлежит критерий CP, заключающийся в покрытии всех таких пар дуг  $v$  и  $w$ , что из дуги  $v$  достижима дуга  $w$ , поскольку именно на дуге может произойти потеря значения переменной, которая в дальнейшем уже не должна использоваться. Для "покрытия" еще одного популярного критерия Cdu достаточно тестировать пары (вершина, дуга), поскольку определение переменной происходит в вершине УГП, а ее использование - на дугах, исходящих из решений, или в вычислительных вершинах.

Методы проектирования тестовых путей для достижения заданной степени тестируемости в структурном тестировании. Процесс построения набора тестов при структурном тестировании принято делить на три фазы:

Конструирование управляющего графа программы (УГП).

Выбор тестовых путей.

Генерация тестов, соответствующих тестовым путям.

Первая фаза соответствует статическому анализу программы, задача которого состоит в получении графа программы и зависящего от него и от критерия тестирования множества элементов, которые необходимо покрыть тестами.

На третьей фазе по известным путям тестирования осуществляется поиск подходящих тестов, реализующих прохождение этих путей.

Вторая фаза обеспечивает выбор тестовых путей. Выделяют три подхода к построению тестовых путей:

Статические методы.

Динамические методы.

Методы реализуемых путей.

Статические методы. Самое простое и легко реализуемое решение - построение каждого пути посредством постепенного его удлинения за счет добавления дуг, пока не будет достигнута выходная вершина управляющего графа программы. Эта идея может быть усилена в так называемых адаптивных методах, которые каждый раз добавляют только один тестовый путь (входной тест), используя предыдущие пути (тесты) как руководство для выбора последующих путей в соответствии с некоторой стратегией. Чаще всего адаптивные стратегии применяются по отношению к критерию C1. Основным недостатком статических методов заключается в том, что не учитывается возможная нереализуемость построенных путей тестирования.

Динамические методы. Такие методы предполагают построение полной системы тестов, удовлетворяющих заданному критерию, путем одновременного решения задачи построения покрывающего множества путей и тестовых данных. При этом можно автоматически учитывать реализуемость или нереализуемость ранее рассмотренных путей или их частей. Основной идеей динамических методов является подсоединение к начальным реализуемым отрезкам путей дальнейших их частей так, чтобы: 1) не терять при этом реализуемости вновь полученных путей; 2) покрыть требуемые элементы структуры программы.

Методы реализуемых путей. Данная методика заключается в выделении из множества путей подмножества всех реализуемых путей. После чего покрывающее множество путей строится из полученного подмножества реализуемых путей.

Достоинство статических методов состоит в сравнительно небольшом количестве необходимых ресурсов, как при использовании, так и при разработке. Однако их реализация может содержать непредсказуемый процент брака (нереализуемых путей). Кроме того, в этих системах переход от покрывающего множества путей к полной системе тестов пользователь должен осуществить вручную, а эта работа достаточно трудоемкая. Динамические методы требуют значительно больших ресурсов как при разработке, так и при эксплуатации, однако увеличение затрат происходит, в основном, за счет разработки и эксплуатации аппарата определения реализуемости пути (символический интерпретатор, решатель неравенств). Достоинство этих методов заключается в том, что их продукция имеет некоторый качественный уровень - реализуемость путей. Методы реализуемых путей дают самый лучший результат.

Пример модульного тестирования

Предлагается протестировать класс TCommand, который реализует команду для склада. Этот класс содержит единственный метод TCommand.GetFullName(), спецификация которого описана следующим образом:

...

Операция GetFullName() возвращает полное имя команды, соответствующее ее допустимому коду, указанному в поле NameCommand. В противном случае возвращается сообщение "ОШИБКА : Неверный код команды". Операция может быть применена в любой момент.

...

Разработаем спецификацию тестового случая для тестирования метода GetFullName на основе приведенной спецификации класса (Табл. 5.1):

Таблица 5.1. Спецификация теста	
Название класса: TCommand	Название тестового случая: TCommandTest1
Описание тестового случая: Тест проверяет правильность работы метода GetFullName - получения полного названия команды на основе кода команды. В тесте подаются следующие значения кодов команд (входные значения): -1, 1, 2, 4, 6, 20, (причем -1 - запрещенное значение).	
Начальные условия: Нет.	
Ожидаемый результат:	
Перечисленным входным значениям должны соответствовать следующие выходные:	
Коду команды -1 должно соответствовать сообщение "ОШИБКА: Неверный код команды"	
Коду команды 1 должно соответствовать полное название команды "ПОЛУЧИТЬ ИЗ ВХОДНОЙ ЯЧЕЙКИ"	
Коду команды 2 должно соответствовать полное название команды "ОТПРАВИТЬ ИЗ ЯЧЕЙКИ В ВЫХОДНУЮ ЯЧЕЙКУ"	
Коду команды 4 должно соответствовать полное название команды "ПОЛОЖИТЬ В РЕЗЕРВ"	
Коду команды 6 должно соответствовать полное название команды "ПРОИЗВЕСТИ ЗАКУПКИ"	
Коду команды 20 должно соответствовать полное название команды "ЗАВЕРШЕНИЕ КОМАНД ВЫДАЧИ"	

Для тестирования метода класса TCommand.GetFullName() был создан тестовый драйвер - класс TCommandTester. Класс TCommandTester содержит метод TCommandTest1(), в котором реализована вся функциональность теста. В данном случае для покрытия спецификации достаточно перебрать следующие значения кодов команд: -1, 1, 2, 4, 6, 20, (-1 - запрещенное значение) и получить соответствующее им полное название команды с помощью метода GetFullName() (Пример 5.1 ). Пары значений (X, Yв) при

исполнении теста заносятся в log-файл для последующей проверки на соответствие спецификации.

После завершения теста следует просмотреть журнал теста, чтобы сравнить полученные результаты с ожидаемыми, заданными в спецификации тестового случая TCommandTest1 (Пример 5.2).

```
class TCommandTester:Tester // Тестовый драйвер
{
...
TCommand OUT;
public TCommandTester()
{
OUT=new TCommand();
Run();
}
private void Run()
{
TCommandTest1();
}
private void TCommandTest1()
{
int[] commands = {-1, 1, 2, 4, 6, 20};
for(int i=0;i<=5;i++)
{
Command=commands[i];
LogMessage(commands[i].ToString()+
": "+OUT.GetFullName());
}
}
...
}
```

Пример 5.1. Тестовый драйвер

-1 : ОШИБКА : Неверный код команды

1 : ПОЛУЧИТЬ ИЗ ВХОДНОЙ ЯЧЕЙКИ

2 : ОТПРАВИТЬ ИЗ ЯЧЕЙКИ В ВЫХОДНУЮ ЯЧЕЙКУ

4 : ПОЛОЖИТЬ В РЕЗЕРВ

6 : ПРОИЗВЕСТИ ЗАНУЛЕНИЕ

20 : ЗАВЕРШЕНИЕ КОМАНД ВЫДАЧИ

Пример 5.2. Спецификация классов тестовых случаев

## ПРАКТИЧЕСКАЯ РАБОТА №16

Тема: Отладка и тестирование программного обеспечения

### Цель

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

### Содержание работы

#### 1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

#### 2. Постановка темы и цели урока

#### 3. Повторение изученного материала

Задание:

#### 1.Интеграционное тестирование

### *Задачи и цели интеграционного тестирования*

Результатом тестирования и верификации отдельных модулей, составляющих программную систему, является заключение о том, что эти модули являются внутренне непротиворечивыми и соответствуют требованиям. Однако отдельные модули редко функционируют сами по себе, поэтому следующая задача после тестирования отдельных модулей – тестирование корректности взаимодействия нескольких модулей, объединенных в единое целое. Такое тестирование называют интеграционным. Его цель – удостовериться в корректности совместной работы компонент системы.

Интеграционное тестирование называют еще тестированием архитектуры системы. С одной стороны, это название объясняется тем, что, интеграционные тесты включают в себя проверки всех возможных видов взаимодействий между программными модулями и элементами, которые определяются в архитектуре системы, – таким образом, интеграционные тесты проверяют полноту взаимодействий в тестируемой реализации системы. С другой стороны, результаты выполнения интеграционных тестов – один из основных источников информации для процесса улучшения и уточнения архитектуры системы, межмодульных и межкомпонентных интерфейсов. Т.е. с этой точки зрения интеграционные тесты проверяют корректность взаимодействия компонент системы.

В результате проведения интеграционного тестирования и устранения всех выявленных дефектов получается согласованная и целостная архитектура программной системы, т.е. можно считать, что интеграционное тестирование – это тестирование архитектуры и низкоуровневых функциональных требований.

Интеграционное тестирование, как правило, представляет собой итеративный процесс, при котором проверяется функциональность все более и более увеличивающейся в размерах совокупности модулей.

### ***Задачи и цели интеграционного тестирования***

#### **Структурная классификация методов интеграционного тестирования**

Как правило, интеграционное тестирование проводится уже по завершении модульного тестирования для всех интегрируемых модулей. Однако это далеко не всегда так. Существует несколько методов проведения интеграционного тестирования:

- *восходящее тестирование;*
- *монолитное тестирование;*
- *нисходящее тестирование;*

**Восходящее тестирование.** При использовании этого метода подразумевается, что сначала тестируются все программные модули, входящие в состав системы, и только затем они объединяются для интеграционного тестирования. При таком подходе значительно упрощается локализация ошибок: если модули протестированы по отдельности, то ошибка при их совместной работе есть проблема их интерфейса. Тогда область поиска проблем у тестирующего становится достаточно узкой, а поэтому гораздо выше вероятность правильно идентифицировать дефект.

Однако у *восходящего метода* тестирования есть существенный недостаток – необходимость в разработке драйвера и заглушек для модульного тестирования перед проведением интеграционного тестирования и необходимость в разработке драйвера и заглушек при интеграционном тестировании части модулей системы.

С одной стороны, драйверы и заглушки – мощный инструмент тестирования, с другой – их разработка требует значительных ресурсов, особенно при изменении состава интегрируемых модулей. Т.е. может потребоваться один набор драйверов для модульного тестирования каждого модуля, отдельный драйвер и заглушки для тестирования интеграции двух модулей из набора, отдельный – для тестирования интеграции трех модулей и т.п. В первую очередь, причина в том, что при интеграции модулей отпадает необходимость в некоторых заглушках, а также требуется изменение драйвера, которое будет поддерживать новые тесты, затрагивающие несколько модулей.

**Монолитное тестирование** предполагает, что отдельные компоненты системы серьезного тестирования не проходили. Основное преимущество данного метода – отсутствие необходимости в разработке тестового окружения, драйверов и заглушек. После разработки всех модулей выполняется их интеграция, затем система проверяется вся в целом, как она есть. Этот подход не следует путать с системным тестированием. Несмотря на то, что при монолитном тестировании проверяется работа всей системы в

целом, основная задача этого тестирования – определить проблемы взаимодействия отдельных модулей системы. Задачей же системного тестирования является оценка качественных и количественных характеристик системы с точки зрения их приемлемости для конечного пользователя.

Тем не менее, *монолитное тестирование* имеет ряд серьезных недостатков:

- очень трудно выявить источник ошибки (идентифицировать ошибочный фрагмент кода);
- трудно организовать исправление ошибок;
- процесс тестирования плохо автоматизируется.

**Нисходящее тестирование** предполагает, что процесс интеграционного тестирования движется следом за разработкой. Сначала при нисходящем подходе тестируют только самый верхний управляющий уровень системы, без модулей более низкого уровня. Затем постепенно с более высокоуровневыми модулями интегрируются более низкоуровневые. В результате применения такого метода отпадает необходимость в драйверах (роль драйвера выполняет более высокоуровневый модуль системы), однако сохраняется нужда в заглушках.

#### Временная классификация методов интеграционного тестирования

На практике чаще всего в различных частях проекта применяются все рассмотренные в предыдущем разделе методы в совокупности. Каждый модуль тестируют по мере готовности отдельно, а потом включают в уже готовую композицию. Для одних частей тестирование получается нисходящим, для других – восходящим. В связи с этим представляется полезным рассмотреть еще один тип классификации типов интеграционного тестирования – классификацию по частоте интеграции:

- тестирование с поздней интеграцией;
- тестирование с постоянной интеграцией;
- тестирование с регулярной или послойной интеграцией.

**Тестирование с поздней интеграцией** – практически полный аналог монолитного тестирования. Интеграционное тестирование при такой схеме откладывается на как можно более поздние сроки проекта. Этот подход оправдывает себя в том случае, если система представляет собой конгломерат слабо связанных между собой модулей, которые взаимодействуют по какому-либо стандартному интерфейсу, определенному вне проекта (например, в случае, если система состоит из отдельных Web-сервисов).

Схематично тестирование с поздней интеграцией может быть изображено в виде цепочки R-C-V-R-C-V-R-C-V-I-R-C-V-R-C-V-I, где R – разработка требований на отдельный модуль, C – разработка программного кода, V – тестирование модуля, I – интеграционное тестирование всего, что было сделано раньше.



**Тестирование с постоянной интеграцией** подразумевает, что как только разрабатывается новый модуль системы, он сразу же интегрируется со всей остальной системой. Тесты для этого модуля проверяют как сугубо его внутреннюю функциональность, так и его взаимодействие с остальными модулями системы. Таким образом, этот подход совмещает в себе модульное тестирование и интеграционное. Разработки заглушек при таком подходе не требуется, но может потребоваться разработка драйверов. В настоящее время именно этот подход называют *unit testing*, несмотря на то, что в отличие от классического модульного тестирования здесь не проверяется функциональность изолированного модуля. Локализация ошибок межмодульных интерфейсов при таком подходе несколько затруднена, но все же значительно ниже, чем при монолитном тестировании. Большая часть таких ошибок выявляется достаточно рано именно за счет частоты интеграции и за счет того, что за одну итерацию тестирования проверяется сравнительно небольшое число межмодульных интерфейсов.

Схематично тестирование с постоянной интеграцией может быть изображено в виде цепочки R-C-I-R-C-I-R-C-I, в которой *фаза тестирования* модуля намеренно опущена и заменена на тестирование интеграции.

При **тестировании с регулярной или послойной интеграцией** интеграционному тестированию подлежат сильно связанные между собой группы модулей (слои), которые затем также интегрируются между собой. Такой вид интеграционного тестирования называют также иерархическим интеграционным тестированием, поскольку укрупнение интегрированных частей системы, как правило, происходит по иерархическому принципу. Однако, в отличие от нисходящего или восходящего тестирования, направление прохода по иерархии в этом подходе не задано.

Таблица 23.1. Основные характеристики различных видов интеграционного тестирования

	Восходящая	Нисходящая	Монолитная	Поздняя интеграция	Постоянная интеграция	Регулярная интеграция
<b>Время интеграции</b>	поздно (после тестирования модулей)	рано (параллельно разработке)	поздно (после разработки всех модулей)	поздно (после разработки и всех модулей)	рано (параллельно разработке)	рано (параллельно разработке)
<b>Частота интеграции</b>	Редко	часто	редко	редко	часто	часто

**ии**

<b>Нужны ли драйверы</b>	Да	нет	нет	нет	да	да
--------------------------	----	-----	-----	-----	----	----

<b>Нужны ли заглушки</b>	Да	да	нет	нет	нет	да
--------------------------	----	----	-----	-----	-----	----

Таблица 23.1 представляет основные характеристики рассмотренных выше видов интеграционного тестирования. Время интеграции характеризует момент времени, когда проводится первое интеграционное тестирование и все последующие, частота интеграции – насколько часто при разработке выполняется интеграция. Необходимость в драйверах и заглушках определена в последних двух строках таблицы.

## ПРАКТИЧЕСКАЯ РАБОТА №17

Тема: Отладка и тестирование программного обеспечения

**Цель** Произвести отладку и тестирование программного обеспечения

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

### **Содержание работы**

#### 1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

#### 2. Постановка темы и цели урока

#### 3. Повторение изученного материала

Задание:

#### 1.Интеграционное тестирование

### *На примере "Калькулятора"*

Как уже отмечалось, в MVSTE под unit-testing подразумевается именно интеграционное тестирование, а конкретно — тестирование с постоянной интеграцией. В "Автоматизация модульного тестирования" мы уже протестировали метод RunEstimate(), при интеграции классов AnalizerClass и CalcClass. Аналогично, составив требования к этой подсистеме из двух классов (а это будут требования ко всем методам AnalizerClass ), можно провести следующий этап тестирования. В качестве примера протестируем все методы такой подсистемы, сделав по одному тестовому примеру на каждый метод:

```
/// <summary>
    /// A test for RunEstimate ()
    /// Проверяем, что, если в стеке находится корректное выражение,
представленное обратной польской записью, то
    /// метод RunEstimate правильно посчитает это выражение
    ///</summary>
    [DeploymentItem("BaseCalculator.exe")]
    [TestMethod()]
    public void RunEstimateTest()
    {
        string expected = "3";
        string actual;
        TestProjectCalculator.BaseCalculator_AnalizerClassAccessor.opz =
            new ArrayList();
        ArrayList _opz =
TestProjectCalculator.BaseCalculator_AnalizerClassAccessor.opz;
```

```

        _opz.Add("7");
        _opz.Add("8");
        _opz.Add("+");
        _opz.Add("5");
        _opz.Add("/");

```

```

        actual

```

```

        =

```

```

TestProjectCalculator.BaseCalculator_AnalaizerClassAccessor.RunEstimate();

```

```

        Assert.AreEqual(expected, actual,
            "BaseCalculator.AnalaizerClass.RunEstimate did not return
the expected value.");
    }

```

```

    /// <summary>
    /// A test for CheckCurrency ()
    /// Проверяет, что, если в выражении нарушена скобочная
структура, то метод возвращает false
    /// </summary>
    [DeploymentItem("BaseCalculator.exe")]
    [TestMethod()]
    public void CheckCurrencyTest()
    {
        bool expected = false;
        bool actual;

```

```

TestProjectCalculator.BaseCalculator_AnalaizerClassAccessor.expression =
    "((5+3)*2-(3*10))-2)+((5+3)";

```

```

        actual

```

```

        =

```

```

TestProjectCalculator.BaseCalculator_AnalaizerClassAccessor.CheckCurrency();

```

```

        Assert.AreEqual(expected, actual,
            "BaseCalculator.AnalaizerClass.CheckCurrency did not
return the expected value.");
        Assert.AreEqual(18,

```

```

TestProjectCalculator.BaseCalculator_AnalaizerClassAccessor.erposition,
            "BaseCalculator.AnalaizerClass.CheckCurrency did not
return the expected value.");
    }

```

```

    /// <summary>
    /// A test for CreateStack ()
    /// Проверяет, что если отформатированное выражение равно

```

```

+ "
    ///"( ( 5 + 3 ) * 2 - ( 3 * 10 ) ) - 2 + ( 5 + 3 ) ",
    /// то стек содержит следующие элементы "5 3 + 2 * 3 10 * - 2 - 5 3 +

```

```

    </summary>
    [DeploymentItem("BaseCalculator.exe")]
    [TestMethod()]
    public void CreateStackTest()
    {
        string expected = "5 3 + 2 * 3 10 * - 2 - 5 3 + + ";
        ArrayList actual;

```

```

TestProjectCalculator.BaseCalculator_AnalaizerClassAccessor.expression =
    "( ( 5 + 3 ) * 2 - ( 3 * 10 ) ) - 2 + ( 5 + 3 ) ";

```

```

        actual
TestProjectCalculator.BaseCalculator_AnalaizerClassAccessor.CreateStack();

```

```

        string actualstr = "";
        foreach (object obj in actual)
        {
            actualstr += obj.ToString() + " ";
        }
        Assert.AreEqual(expected, actualstr,
            "BaseCalculator.AnalaizerClass.CreateStack did not return
the expected value.");
    }

```

```

    <summary>
    ///A test for Estimate ()
    ///Проверяет, что , если выражение , которое необходимо
    проверить,равно
    ///((5+3)*2-(3*10))-2+(5+3), то метод вернет его значение,
    /// равное -8
    </summary>
    [DeploymentItem("BaseCalculator.exe")]
    [TestMethod()]
    public void EstimateTest()
    {
        string expected = "-8";
        string actual;

```

```

TestProjectCalculator.BaseCalculator_AnalaizerClassAccessor.expression =
    "((5+3)*2-(3*10))-2+(5+3)";
    actual
TestProjectCalculator.BaseCalculator_AnalaizerClassAccessor.Estimate();

    Assert.AreEqual(expected, actual,
        "BaseCalculator.AnalaizerClass.Estimate did not return the
expected value.");
}

```

```

/// <summary>
/// A test for Format ()
/// Проверяет, что если выражение равно "
/// ((5+3)*2-(3 *10))-2+ (5+3)", то метод отформатирует его к виду:
/// "( ( 5 + 3 ) * 2 - ( 3 * 10 ) ) - 2 + ( 5 + 3 ) "
/// </summary>
[DeploymentItem("BaseCalculator.exe")]
[TestMethod()]
public void FormatTest()
{
    string expected = "( ( 5 + 3 ) * 2 - ( 3 * 10 ) ) - 2 + ( 5 + 3 ) ";
    string actual;

```

```

TestProjectCalculator.BaseCalculator_AnalaizerClassAccessor.expression =
    "((5+3)*2-(3 *10))-2+ (5+3)";
    actual
TestProjectCalculator.BaseCalculator_AnalaizerClassAccessor.Format();

    Assert.AreEqual(expected, actual,
        "BaseCalculator.AnalaizerClass.Format did not return the
expected value.");
}

```

23.1.

Запустив тесты, можно убедиться в корректной (для данных тестовых примеров) работе методов.

**Замечание.** Тестовых требований, как и функциональных, к методам нет, однако их можно составить по системным требованиям и описанию методов.

Как уже отмечалось в "Модульное тестирование", эти методы обладают многими недостатками, и некоторые из них генерируют исключения на некорректных входных данных. Это, в принципе, логично, так как запуск

метода RunEstimate() подразумевает, что входное выражение уже обработано в методах CheckCurrency(), Format(), CreateStack(), а отдельно от них этот метод вызываться не будет. Для этого необходимо сделать уровень доступа к ним private и протестировать их только на корректных данных. Основное же внимание стоит уделить методу Estimate(), который поочередно запускает все вышеперечисленные методы и имеет уровень доступа public. Один из тестов для него приведен выше.

После того, как такая система из двух модулей протестирована, переходим к тестированию всей системы, присоединив последние модули.

**Замечание.** Модуль BaseCalc, который отвечает за пользовательский интерфейс, мы не тестируем, так как это выходит за рамки курса.

Теперь можно использовать системные требования, для тестирования программы в целом. Проведем тесты для метода Main(string[] args), так как это единственный метод, не считая уже протестированного Estimate(), который не относится к графическому интерфейсу и с которым работают пользователи:

```
/// <summary>
    ///A test for Main (string[])
    /// 4.2.1.1. Для чисел, каждое из которых меньше либо равно
MAXINT и больше либо равно MININT,
    /// функция суммирования должна возвращать правильную сумму с
точки зрения математики
    ///</summary>
    [DeploymentItem("BaseCalculator.exe")]
    [TestMethod()]
    public void MainTest()
    {
        string[] args = new string[1]; // TODO: Initialize to an appropriate
value
        args[0] = "2+2";

        int expected = 0;
        int actual;

        actual
=
TestProjectCalculator.BaseCalculator_ProgramAccessor.Main(args);
        Assert.AreEqual(expected, actual,
            "BaseCalculator.Program.Main did not return the expected
value.");
    }

    /// <summary>
    ///A test for Main (string[])
    /// 4.2.1.2. Для чисел, сумма которых больше чем MAXINT и
меньше чем MININT,
```

```

        /// а также в случае, если любое из слагаемых больше чем MAXINT
или меньше чем MININT,
        /// программа должна выдавать ошибку Error 06(см 2.2.3)
        ///</summary>
        [DeploymentItem("BaseCalculator.exe")]
        [TestMethod()]
        public void MainTest1()
        {
            string[] args = new string[1]; // TODO: Initialize to an appropriate
value
            args[0] = "2711477380+1000000";

            int expected = 6;
            int actual;

            actual =
TestProjectCalculator.BaseCalculator_ProgramAccessor.Main(args);

            Assert.AreEqual(expected, actual,
                "BaseCalculator.Program.Main did not return the expected
value.");
        }
    }
}
23.2.

```

И так далее. Таким образом, проверив методы Main() и Estimate(), а также некоторые методы визуального интерфейса, можно убедиться в соответствии системы требованиям или, наоборот, обнаружить какие-то ошибки в межмодульном взаимодействии.



## ПРАКТИЧЕСКАЯ РАБОТА №18

Тема: Отладка и тестирование программного обеспечения

**Цель** Произвести отладку и тестирование программного обеспечения

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

### Содержание работы

#### 1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

#### 2. Постановка темы и цели урока

#### 3. Повторение изученного материала

Задание:

#### 1.Интеграционное тестирование

Спецификация тестового случая для интеграционного тестирования

Названия взаимодействующих классов	TCommandQueue, TCommand
Название теста	TCommandQueueTest1
Описание теста	тест проверяет возможность создания объекта типа TCommand и добавления его в очередь при вызове метода AddCommand
Начальные условия	очередь команд пуста
Ожидаемый результат	в очередь будет добавлена одна команда

На основе этой спецификации разработан тестовый драйвер пример 6.1 — класс TCommandQueueTester, который наследуется от класса Tester.

Класс содержит:

конструктор, в котором создаются объекты классов TStore, TTerminalBearing и объект типа TcommandQueue

Методы, реализующие тесты. Каждый тест реализован в отдельном методе.

Метод Run, в котором вызываются методы тестов.

Метод dump, который сохраняет в Log-файле теста информацию обо всех командах, находящихся в очереди в формате — Номер позиции в очереди: полное название команды

Точку входа в программу — метод Main, в котором происходит создание экземпляра класса TCommandQueueTester.

```

public TCommandQueueTester()
{
    TB = new TTerminalBearing();
    S = new TStore();
    CommandQueue=new TCommandQueue(S,TB);
    S.CommandQueue=CommandQueue;
    ...
}

```

Пример 6.1. Объект типа TcommandQueue

```

TCommandQueueTester::TCommandQueueTester()
{
    TB = new TTerminalBearing();
    S = new TStore();
    CommandQueue=new TCommandQueue(S,TB);
    S->CommandQueue=CommandQueue;
}

```

Пример 6.1.1. Объект типа TcommandQueue (C++)

Теперь создадим тест, который проверяет, создается ли объект типа TCommand, и добавляется ли команда в конец очереди.

```

private void TCommandQueueTest1()
{
    LogMessage("///// TCommandQueue Test1 /////");
    LogMessage("Проверяем, создается ли
объект типа TCommand");
    // В очереди нет команд
    dump();
    // Добавляем команду
    // параметр = -1 означает, что команда
    // должна быть добавлена в конец очереди
    CommandQueue.AddCommand(TCommand.GetR,0,0,0,
    new TBearingParam(),new TAxleParam(),-1);
    LogMessage("Command added");
    // В очереди одна команда
    dump();
}

```

Пример 6.2. Тест

```

void TCommandQueueTester::TCommandQueueTest1()
{
    LogMessage("///// TCommandQueue Test1 /////");
    LogMessage("Проверяем, создается ли
объект типа TCommand");
    // В очереди нет команд
    dump();
}

```

```

// Добавляем команду
// параметр = -1 означает, что команда
// должна быть добавлена в конец очереди
CommandQueue.AddCommand(GetR,0,0,0,
new TBearingParam(),
new TAxleParam(),-1);
LogMessage("Command added");
// В очереди одна команда
dump();
}

```

#### Пример 6.2.1. Тест (C++)

В класс включены еще два разработанных теста.

После завершения теста следует просмотреть текстовый журнал теста, чтобы сравнить полученные результаты с ожидаемыми результатами, заданными в спецификации тестового случая TCommandQueueTest1 пример 6.3.

```

///// TCommandQueue Test1 /////
Проверяем, создается ли объект типа TCommand
0 commands in command queue
Command added
1 commands in command queue
0: ПОЛУЧИТЬ ИЗ ВХОДНОЙ ЯЧЕЙКИ

```

#### Пример 6.3. Спецификация результатов теста (html, txt)

## ПРАКТИЧЕСКАЯ РАБОТА №19

Тема: Отладка и тестирование программного обеспечения

Цель Произвести отладку и тестирование программного обеспечения

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

**Содержание работы**

1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

2. Постановка темы и цели урока

3. Повторение изученного материала

Задание:

1.Интеграционное тестирование

**Тестирование программы  $P$  по некоторому критерию  $C$  означает покрытие множества компонентов программы  $P$   $M = \{m_1...m_k\}$  по элементам или по связям**

$T = \{t_1...t_n\}$  — кортеж неизбыточных тестов  $t_i$ .

Тест  $t_i$  неизбыточен, если существует покрытый им компонент  $m_i$  из  $M(P,C)$ , не покрытый ни одним из предыдущих тестов  $t_1...t_{i-1}$ . Каждому  $t_i$  соответствует неизбыточный путь  $p_i$  — последовательность вершин от входа до выхода.

$V(P,C)$  — сложность тестирования  $P$  по критерию  $C$  — измеряется max числом неизбыточных тестов, покрывающих все элементы множества  $M(P,C)$

$DV(P,C,T)$  — остаточная сложность тестирования  $P$  по критерию  $C$  — измеряется max числом неизбыточных тестов, покрывающих элементы множества  $M(P,C)$ , оставшиеся непокрытыми, после прогона набора тестов  $T$ . Величина  $DV$  строго и монотонно убывает от  $V$  до 0.

$TV(P,C,T) = (V-DV)/V$  — оценка степени тестированности  $P$  по критерию  $C$ .

Критерий окончания тестирования  $TV(P,C,T) \geq L$ , где  $(0 \leq L \leq 1)$ .  $L$  — уровень оттестированности, заданный в требованиях к программному продукту.

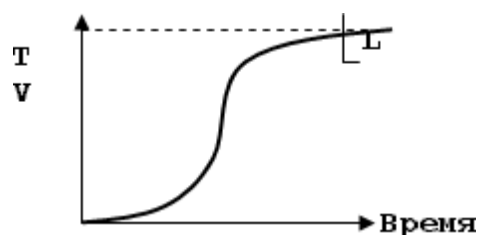


Рис. 4.1. Метрика оттестированности приложения

Рассмотрим две модели программного обеспечения, используемые при оценке оттестированности.

Для оценки степени оттестированности часто используется УГП — управляющий граф программы. УГП многокомпонентного объекта  $G$  (Рис. 4.2, Пример 4.4), содержит внутри себя два компонента  $G_1$  и  $G_2$ , УГП которых раскрыты.

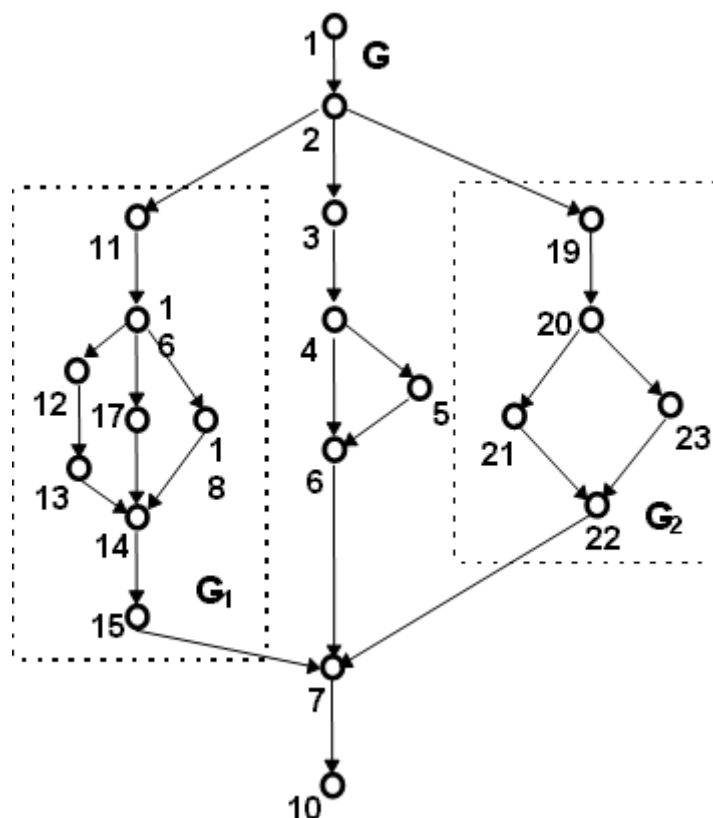


Рис. 4.2. Плоская модель УГП компонента  $G$

В результате УГП компонента  $G$  имеет такой вид, как если бы компоненты  $G_1$  и  $G_2$  в его структуре специально не выделялись, а УГП компонентов  $G_1$  и  $G_2$  были вставлены в УГП  $G$ . Для тестирования компонента  $G$  в соответствии с критерием путей потребуется прогнать тестовый набор, покрывающий следующий набор трасс графа  $G$  (Пример 4.1):

$$P1(G) = 1-2-3-4-5-6-7-10;$$

$$P2(G) = 1-2-3-4-6-7-10;$$

$$P3(G) = 1-2-11-16-18-14-15-7-10;$$

$$P4(G) = 1-2-11-16-17-14-15-7-10;$$

$$P5(G) = 1-2-11-16-12-13-14-15-7-10;$$

$$P6(G) = 1-2-19-20-23-22-7-10;$$

$$P7(G) = 1-2-19-20-21-22-7-10;$$

Пример 4.1. Набор трасс, необходимых для покрытия плоской модели УГП компонента  $G$

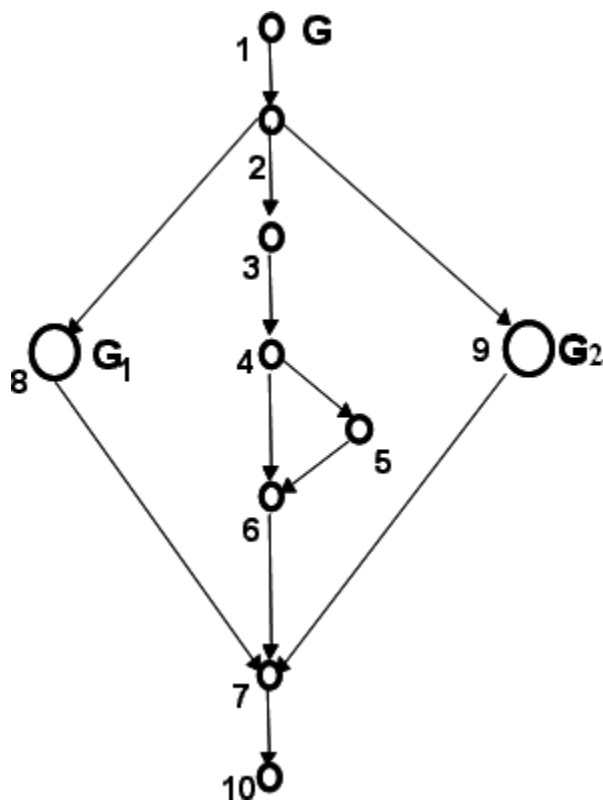


Рис. 4.3. Иерархическая модель УГП компонента G

УГП компонента G, представленный в виде иерархической модели, приведен на Рис. 4.3, Пример 4.5. В иерархическом УГП G входящие в его состав компоненты представлены ссылками на свои УГП G1 и G2 (Рис. 4.4, Пример 4.5)

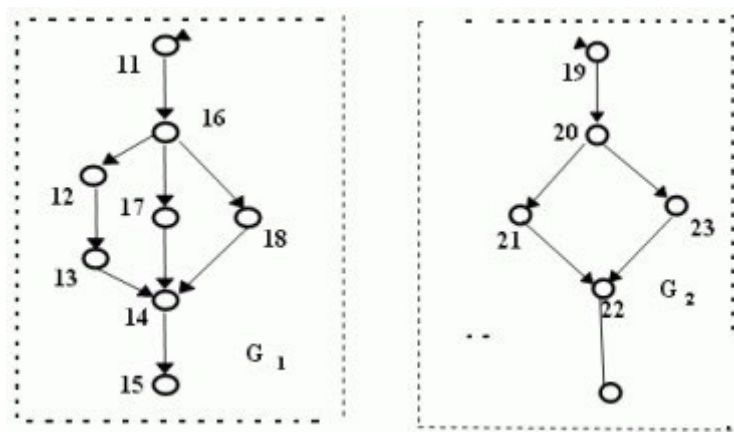


Рис. 4.4. Иерархическая модель: УГП компонент G1 и G2

Для исчерпывающего тестирования иерархической модели компонента G в соответствии с критерием путей требуется прогнать следующий набор трасс (Пример 4.2):

$$P1(G) = 1-2-3-4-5-6-7-10;$$

$$P2(G) = 1-2-3-4-6-7-10;$$

$$P3(G) = 1-2-8-7-10;$$

$$P4(G) = 1-2-9-7-10.$$

Пример 4.2. Набор трасс, необходимых для покрытия иерархической модели УГП компонента G

Приведенный набор трасс достаточен при условии, что компоненты G1 и G2 в свою очередь исчерпывающе протестированы. Чтобы обеспечить выполнение этого условия в соответствии с критерием путей, надо прогнать все трассы Пример 4.3.

$P11(G1)=11-16-12-13-14-15;$

$P21(G2)=19-20-21-22;$

$P12(G1)=11-16-17-14-15;$

$P22(G2)=11-16-18-14-15;$

$P13(G1)=19-20-23-22.$

Пример 4.3. Набор трасс иерархической модели УГП, необходимых для покрытия УГП компонентов G1 и G2

## ПРАКТИЧЕСКАЯ РАБОТА №20

Тема: Отладка и тестирование программного обеспечения

Цель Произвести отладку и тестирование программного обеспечения

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:** 1,2.

**Содержание работы**

1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

2. Постановка темы и цели урока

3. Повторение изученного материала

Задание:

1. Интеграционное тестирование

**Тестирование программы  $P$  по некоторому критерию  $C$  означает покрытие множества компонентов программы  $P$   $M = \{m_1...m_k\}$  по элементам или по связям**

$T = \{t_1...t_n\}$  — кортеж неизбыточных тестов  $t_i$ .

Тест  $t_i$  неизбыточен, если существует покрытый им компонент  $m_i$  из  $M(P,C)$ , не покрытый ни одним из предыдущих тестов  $t_1...t_{i-1}$ . Каждому  $t_i$  соответствует неизбыточный путь  $p_i$  — последовательность вершин от входа до выхода.

$V(P,C)$  — сложность тестирования  $P$  по критерию  $C$  — измеряется  $\max$  числом неизбыточных тестов, покрывающих все элементы множества  $M(P,C)$

$DV(P,C,T)$  — остаточная сложность тестирования  $P$  по критерию  $C$  — измеряется  $\max$  числом неизбыточных тестов, покрывающих элементы множества  $M(P,C)$ , оставшиеся непокрытыми, после прогона набора тестов  $T$ . Величина  $DV$  строго и монотонно убывает от  $V$  до 0.

$TV(P,C,T) = (V-DV)/V$  — оценка степени тестированности  $P$  по критерию  $C$ .

Критерий окончания тестирования  $TV(P,C,T) \geq L$ , где  $(0 \leq L \leq 1)$ .  $L$  — уровень оттестированности, заданный в требованиях к программному продукту.

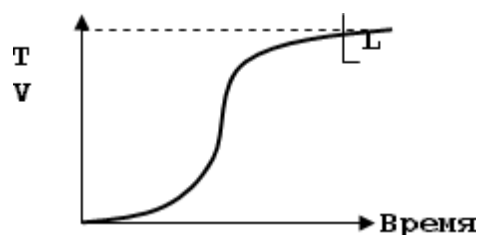


Рис. 4.1. Метрика оттестированности приложения



Рассмотрим две модели программного обеспечения, используемые при оценке оттестированности.

Для оценки степени оттестированности часто используется УГП — управляющий граф программы. УГП многокомпонентного объекта  $G$  (Рис. 4.2, Пример 4.4), содержит внутри себя два компонента  $G_1$  и  $G_2$ , УГП которых раскрыты.

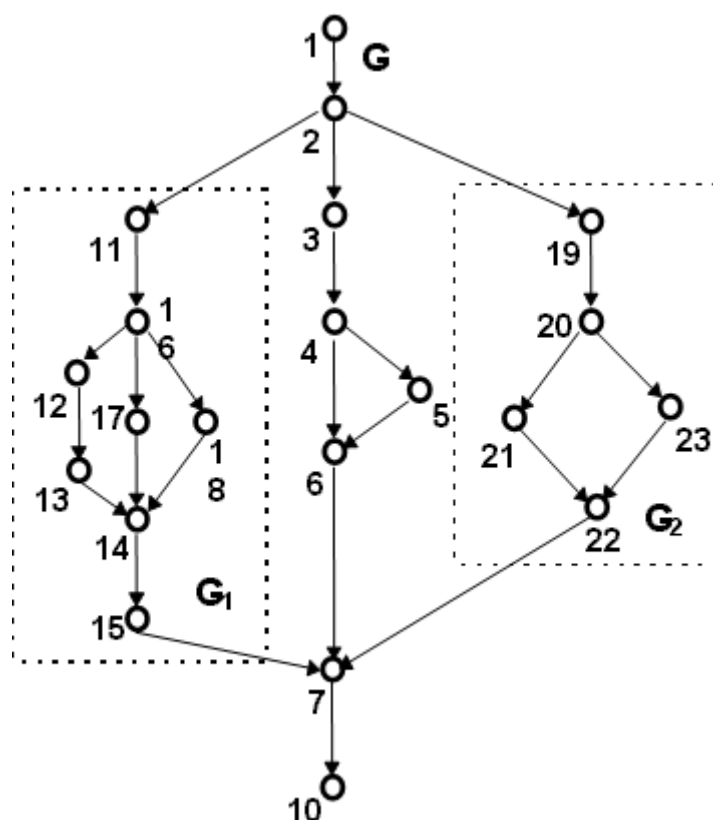


Рис. 4.2. Плоская модель УГП компонента  $G$

В результате УГП компонента  $G$  имеет такой вид, как если бы компоненты  $G_1$  и  $G_2$  в его структуре специально не выделялись, а УГП компонентов  $G_1$  и  $G_2$  были вставлены в УГП  $G$ . Для тестирования компонента  $G$  в соответствии с критерием путей потребуется прогнать тестовый набор, покрывающий следующий набор трасс графа  $G$  (Пример 4.1):

$$P1(G) = 1-2-3-4-5-6-7-10;$$

$$P2(G) = 1-2-3-4-6-7-10;$$

$$P3(G) = 1-2-11-16-18-14-15-7-10;$$

$$P4(G) = 1-2-11-16-17-14-15-7-10;$$

$$P5(G) = 1-2-11-16-12-13-14-15-7-10;$$

$$P6(G) = 1-2-19-20-23-22-7-10;$$

$$P7(G) = 1-2-19-20-21-22-7-10;$$

Пример 4.1. Набор трасс, необходимых для покрытия плоской модели УГП компонента  $G$

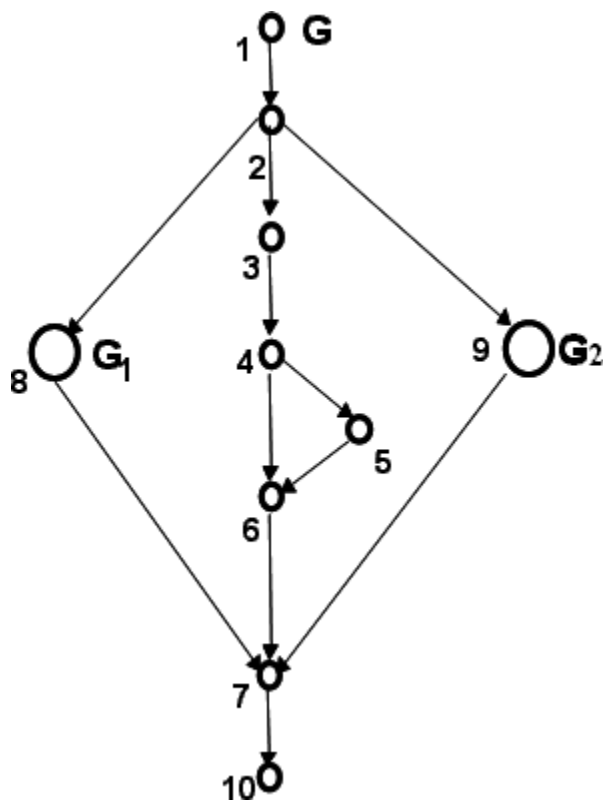


Рис. 4.3. Иерархическая модель УГП компонента G

УГП компонента G, представленный в виде иерархической модели, приведен на Рис. 4.3, Пример 4.5. В иерархическом УГП G входящие в его состав компоненты представлены ссылками на свои УГП G1 и G2 (Рис. 4.4, Пример 4.5)

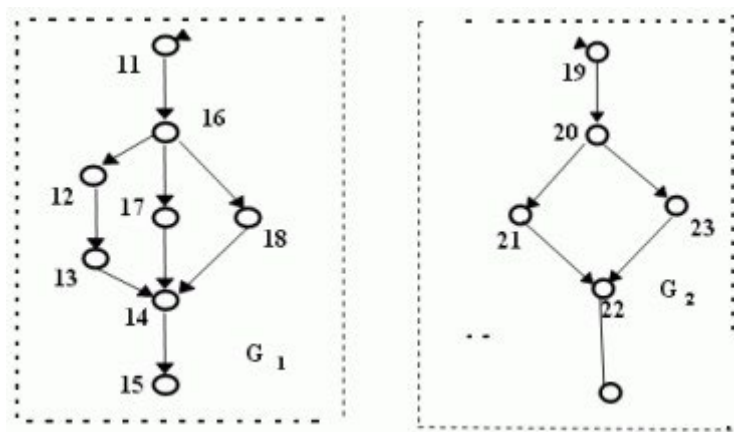


Рис. 4.4. Иерархическая модель: УГП компонент G1 и G2

Для исчерпывающего тестирования иерархической модели компонента G в соответствии с критерием путей требуется прогнать следующий набор трасс (Пример 4.2):

$$P1(G) = 1-2-3-4-5-6-7-10;$$

$$P2(G) = 1-2-3-4-6-7-10;$$

$$P3(G) = 1-2-8-7-10;$$

$$P4(G) = 1-2-9-7-10.$$

Пример 4.2. Набор трасс, необходимых для покрытия иерархической модели УГП компонента G

Приведенный набор трасс достаточен при условии, что компоненты G1 и G2 в свою очередь исчерпывающе протестированы. Чтобы обеспечить выполнение этого условия в соответствии с критерием путей, надо прогнать все трассы Пример 4.3.

$P11(G1)=11-16-12-13-14-15;$

$P21(G2)=19-20-21-22;$

$P12(G1)=11-16-17-14-15;$

$P22(G2)=11-16-18-14-15;$

$P13(G1)=19-20-23-22.$

Пример 4.3. Набор трасс иерархической модели УГП, необходимых для покрытия УГП компонентов G1 и G2

## **ПРАКТИЧЕСКАЯ РАБОТА №21**

Тема: Отладка и тестирование программного обеспечения

**Цель** Произвести отладку и тестирование программного обеспечения

**Оборудование:** ПК, Microsoft Office Word

**Справочный материал:**1,2.

**Содержание работы**

1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

2. Постановка темы и цели урока

3. Повторение изученного материала

Задание:

1.Интеграционное тестирование

**Рассматривается пример тестов на C# для класса TCommand (приложение 3 (HLD)). При выполнении заданий необходимо будет самостоятельно написать тесты для других классов приложения. Параллельно с изучением этого раздела полезно открыть проект ModuleTesting\ModuleTests.sln.**

Рассмотрим тестирование класса TCommand. Этот класс реализует единственную операцию GetFullName(), которая возвращает полное название команды в виде строки. Разработаем спецификацию тестового случая для тестирования метода GetFullName на основе спецификации этого класса (приложение 3):

Название класса: TCommand

Название тестового случая: TCommandTest1

Описание тестового случая: Тест проверяет правильность работы метода GetFullName — получения полного названия команды на основе кода команды. В тесте подаются следующие значения кодов команд (входные значения): -1, 1, 2, 4, 6, 20, где -1 — запрещенное значение

Начальные условия: Нет

Ожидаемый результат:

Перечисленным входным значениям должны соответствовать следующие выходные:

Коду команды -1 должно соответствовать сообщение "ОШИБКА: Неверный код команды"

Коду команды 1 должно соответствовать полное название команды "ПОЛУЧИТЬ ИЗ ВХОДНОЙ ЯЧЕЙКИ"

Коду команды 2 должно соответствовать полное название команды "ОТПРАВИТЬ ИЗ ЯЧЕЙКИ В ВЫХОДНУЮ ЯЧЕЙКУ"

Коду команды 4 должно соответствовать полное название команды "ПОЛОЖИТЬ В РЕЗЕРВ"

Коду команды 6 должно соответствовать полное название команды "ПРОИЗВЕСТИ ЗАНУЛЕНИЕ"

Коду команды 20 должно соответствовать полное название команды "ЗАВЕРШЕНИЕ КОМАНД ВЫДАЧИ"

На основе спецификации был создан тестовый драйвер — класс TCommandTester, наследующий функциональность абстрактного класса Tester.

```
public class Log
{
    static private StreamWriter log=new
    StreamWriter("log.log"); //Создание лог файла
    static public void Add(string msg)
    //Добавление сообщения в лог файл
    {
        log.WriteLine(msg);
    }
    static public void Close() //Закреть лог файл
    {
        log.Close();
    }
}
abstract class Tester
{
    protected void LogMessage(string s)
    //Добавление сообщения в лог-файл
    {
        Log.Add(s);
    }
}
class TCommandTester:Tester // Тестовый драйвер
{
    TCommand OUT;
    public TCommandTester()
    {
        OUT=new TCommand();
        Run();
    }
}
```

```

private void Run()
{
    TCommandTest1();
}
private void TCommandTest1()
{
    int[] commands = {-1, 1, 2, 4, 6, 20};
    for(int i=0;i<=5;i++)
    {
        OUT.NameCommand=commands[i];
        LogMessage(commands[i].ToString()+" :
"+OUT.GetFullName());
    }
}
[STAThread]
static void Main()
{
    TCommandTester CommandTester = new TCommandTester();
    Log.Close();
}
}

```

### Листинг 2.1. Тестовый драйвер

Класс TCommandTester содержит метод TCommandTest1(), в котором реализована вся функциональность теста. В данном случае для покрытия спецификации достаточно перебрать следующие значения кодов команд: -1, 1, 2, 4, 6, 20, где -1 — запрещенное значение, и получить соответствующие им полное название команды с помощью метода GetFullName(). Пары соответствующих значений заносятся в log-файл для последующей проверки на соответствие спецификации.

Таким образом, для тестирования любого метода класса необходимо:

Определить, какая часть функциональности метода должна быть протестирована, то есть при каких условиях он должен вызываться. Под условиями здесь понимаются параметры вызова методов, значения полей и свойств объектов, наличие и содержимое используемых файлов и т. д.

Создать тестовое окружение, обеспечивающее требуемые условия.

Запустить тестовое окружение на выполнение.

Обеспечить сохранение результатов в файл для их последующей проверки.

После завершения выполнения сравнить полученные результаты со спецификацией.

## ПРАКТИЧЕСКАЯ РАБОТА №22

Тема: Документирование

**Цель Разработать документацию и зафиксировать знания на практике**

**Оборудование: ПК, Microsoft Office Word**

**Справочный материал:1,2.**

**Содержание работы**

1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

2. Постановка темы и цели урока

3. Повторение изученного материала

**Задание:**

1. Построить диаграмму классов предметной области задачи.
2. Построить диаграмму вариантов использования предметной области задачи.
3. Составить отчет по практической работе.

**Содержание отчета:**

1. созданные диаграммы классов (для диаграммы классов из пункта 2 задания должен быть указан сценарий, для которого данная диаграмма построена);
2. краткое описание каждого созданного класса и отношений между классами.

Диаграммы классов (class diagram) используются для моделирования статического вида системы с точки зрения проектирования. *Диаграмма классов* - диаграмма, на которой показано множество классов, интерфейсов, коопераций и отношений между ними. Используется в следующих целях:

- для *моделирования словаря* системы: предполагает принятие решения о том, какие абстракции являются частью системы, а какие - нет. С помощью диаграмм классов можно определить эти абстракции и их обязанности;
- для *моделирования простых коопераций*. Кооперация - это сообщество классов, интерфейсов и других элементов, работающих совместно для обеспечения некоторого кооперативного поведения;
- для *моделирования логической схемы базы данных*.

Согласно Мартину Фаулеру существуют три различные точки зрения на построение диаграмм классов или любой другой модели:

- *концептуальная точка зрения* - диаграммы классов служат для представления понятий изучаемой предметной области. Эти понятия будут

соответствовать реализующим их классам, но прямое соответствие может отсутствовать. Концептуальная модель может иметь слабое отношение или вообще не иметь никакого отношения к реализующему ее программному обеспечению, поэтому ее можно рассматривать без привязки к какому-то языку программирования;

- *точка зрения спецификации* - рассматривается программная система, при этом рассматривается только ее интерфейсы, но не реализация;
- *точка зрения реализации* - классы диаграммы соответствуют реальным классам программной системы.

### Пример выполнения работы.

#### 1. Создание диаграммы классов для сценария "Добавить новый заказ" прецедента "Работа с заказом"

Диаграммы классов будем рассматривать с концептуальной точки зрения. Для упрощения задачи и чтобы не загромождать диаграммы несущественными деталями методы setX, getX для каждого атрибута X классов задавать не будем.

Создадим в Логическом представлении браузера новую диаграмму классов и назовем ее "Add New Order". В поле документации запишем для нее следующий текст: "Диаграмма классов для сценария "Добавить новый заказ" прецедента "Работа с заказом"". Заполнение диаграммы начнем с определения классов-сущностей. Рассматриваемый сценарий состоит из:

- самого заказа;
- клиента, который делает заказ;
- комплектующих изделий, которые входят в заказ.

Создадим классы-сущности *Order* (Заказ), *Client* (Клиент) и *ComponentPart* (Комплектующее изделие). Поскольку в один заказ может входить много разных комплектующих изделий, и одно комплектующее изделие может входить во много заказов, то введем еще один класс-сущность *OrderItem* (Состав заказа). Опишем каждый класс.

#### Класс *Client*:

Параметр	Значение
Комментарий	Класс, представляющий собой клиента фирмы
Атрибуты	name : String - наименование клиента address : String - адрес клиента phone : String - телефон клиента Все атрибуты имеют модификатор доступа - private



Операции	AddClient() - добавление нового клиента RemoveClient() - удаление существующего клиента GetInfo() - получить информацию о клиенте Все операции имеют модификатор доступа - public
----------	--

### Класс *Order*:

Параметр	Значение
Комментарий	Класс, представляющий собой заказ, который делает клиент
Атрибуты	orderNumber : Integer - номер заказа orderDate : Date - дата оформления заказа orderComplete : Date - дата выполнения заказа Все атрибуты имеют модификатор доступа - private
Операции	Create() - создание нового заказа SetInfo() - занести информацию о заказе GetInfo() - получить информацию о заказе Все операции имеют модификатор доступа - public

### Класс *OrderItem*:

Параметр	Значение
Комментарий	Класс, представляющий собой пункт заказа, который делает клиент
Атрибуты	itemNumber : Integer - номер пункта заказа quantity : Integer - количество комплектующих изделий price : Double - цена за единицу Все атрибуты имеют модификатор доступа - private
Операции	Create() - создание новой строки заказа SetInfo() - занести информацию о строке заказа GetInfo() - получить информацию о строке заказа Все операции имеют модификатор доступа - public

### Класс *ComponentPart*:

Параметр	Значение
Комментарий	Класс, представляющий собой комплектующие изделия
Атрибуты	name : String - наименование manufacturer : String - производитель price : Double - цена за единицу description - описание Все атрибуты имеют модификатор доступа - private
Операции	AddComponent() - добавление нового комплектующего изделия RemoveComponent() - удаление комплектующего изделия GetInfo() - получить информацию о комплектующем изделии Все операции имеют модификатор доступа - public

Результат создания классов-сущностей показан на рис. 1:

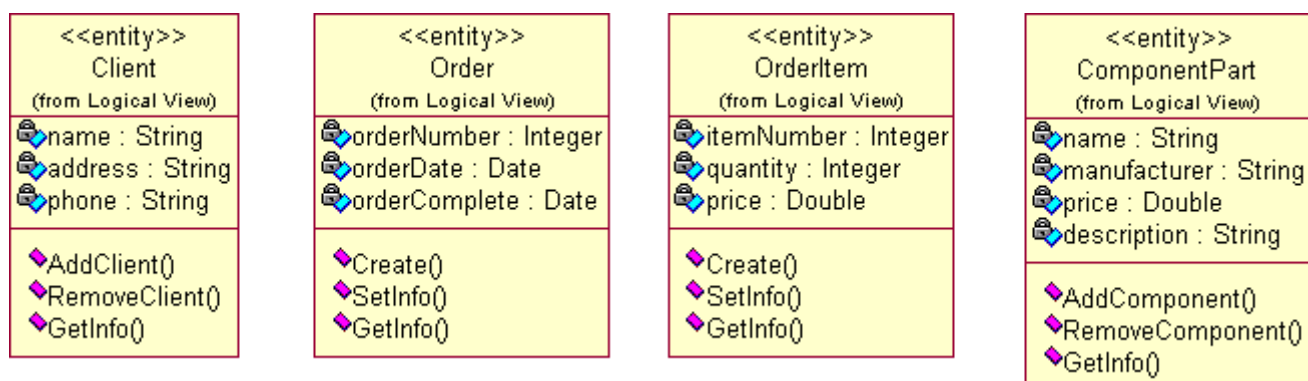


Рисунок 1. Созданные классы-сущности

Добавим отношения между классами (рис. 2):

- класс *Client* и *Order* - отношение ассоциации, поскольку данные два класса просто связаны друг с другом и никакие другие типы связей здесь применить нельзя. Один клиент может сделать несколько заказов, каждый заказ поступает только от одного клиента, поэтому кратность связи со стороны класса *Client* - 1, со стороны *Order* - 1..n;
- класс *Order* и *OrderItem* - отношение композиции, поскольку строка заказа является частью заказа, и без него существовать не может. В один заказ может входить несколько строк заказа, строка заказа относится

только к одному заказу, поэтому кратность связи со стороны *Order* - 1, со стороны *OrderItem* - 1..n;

- класс *OrderItem* и *ComponentPart* - отношение агрегации, поскольку комплектующие изделия являются частями строки заказа, но и те, и другие, являются самостоятельными классами. Одно комплектующее изделие может входить во много строк заказа, в одну строку заказа входит только одно комплектующее изделие, поэтому кратность связи со стороны *ComponentPart* - 1, со стороны *OrderItem* - 1..n.

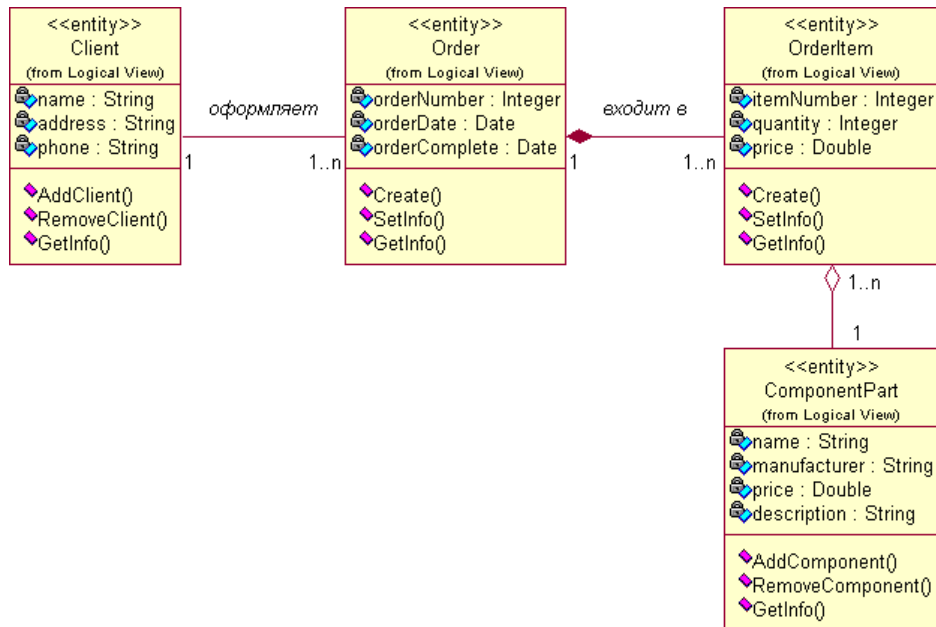


Рисунок 2. Классы-сущности и отношения между ними

Добавим теперь на диаграмму граничные и управляющие классы (рис. 3). Рассматриваемый сценарий - это только одно из действий, которые обеспечивает прецедент "Работа с заказом". Прецедент также позволяет просмотреть, отредактировать или удалить заказ. Это означает, что необходимо предусмотреть механизм, который позволяет выбирать необходимое действие. Создадим для этого граничный класс *OrderOptions* (Параметры работы с заказом) с комментарием "Класс, обеспечивающий механизм работы с заказами". Также создадим граничный класс *AddNewOrder* (Добавление нового заказа), который будет служить для добавления новых заказов (комментарий - "Класс служит для добавления новых заказов"). Отношение между этими классами - агрегация, поскольку в данном случае класс *AddNewOrder* рассматривается как часть класса *OrderOptions*, частями которого также будут классы для просмотра, редактирования и удаления заказов. Кратность связи 1 к 1, поскольку в состав класса *OrderOptions* входит только один класс *AddNewOrder*.

Перейдем теперь к управляющим классам. Добавим управляющий класс *OrderManager* (Менеджер по работе с заказами) с комментарием "Управляющий класс для обработки потока событий прецедента "Работа с заказами"", который будет обеспечивать обработку потока событий для

рассматриваемого прецедента. Данный класс будет связан с классами *AddNewOrder* и *Order*. Отношение между классами *AddNewOrder* и *OrderManager* - однонаправленная ассоциация с кратностью связи 1 к 1, поскольку один экземпляр класса *AddNewOrder* взаимодействует только с одним экземпляром класса *OrderManager*. Отношение между классами *OrderManager* и *Order* - однонаправленная ассоциация с кратностью связи 1 к 1..n, поскольку один класс *OrderManager* может взаимодействовать с несколькими классами *Order*.

Окончательный вариант диаграммы классов показан на рис. 3:

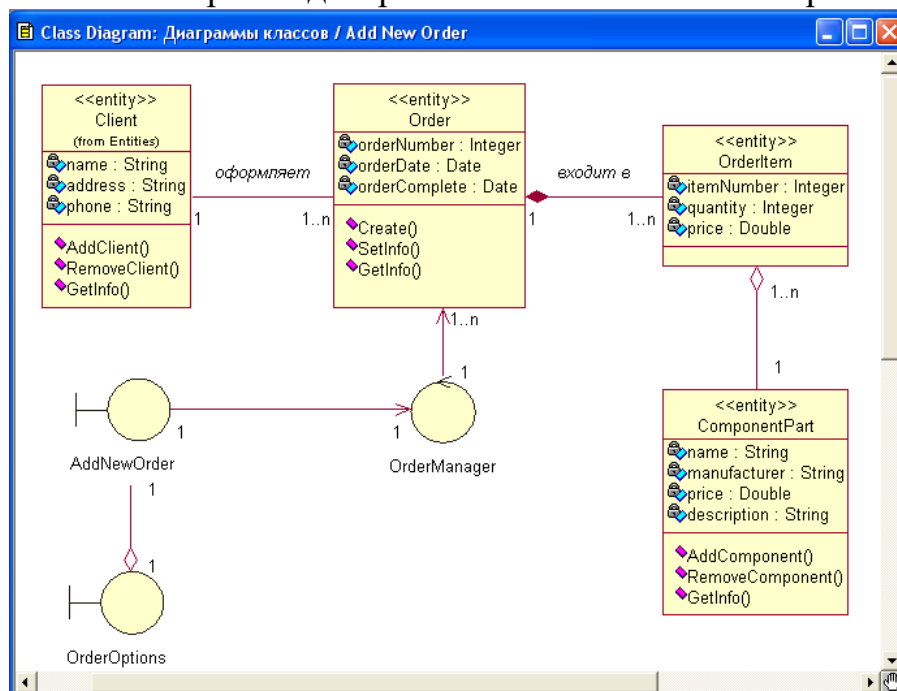


Рисунок 3. Итоговая диаграмма классов

## 2. Создание пакетов

Пакеты предназначены для группировки элементов в группы по определенным критериям. В простейшем случае классы можно группировать по их стереотипам. Создадим три пакета: *Entities* (классы-сущности), *Boundaries* (граничные классы) и *Control* (управляющие классы). Для этого необходимо щелкнуть правой кнопкой мыши на Логическом представлении браузера (Logical View), в появившемся контекстном меню выбрать пункт New > Package (Создать > Пакет), и ввести имя пакета. Результат создания пакетов показан на рис.4:

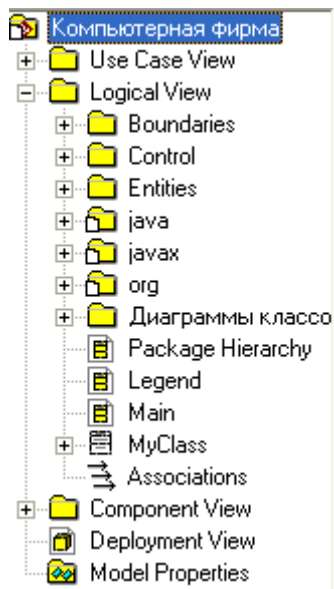


Рисунок 4. Созданные пакеты

В поле документации (Documentation) для каждого пакета зададим комментарий:

- для пакета Entities комментарий: пакет содержит классы-сущности;
- для пакета Boundaries комментарий: пакет содержит граничные классы;
- для пакета Control комментарий: пакет содержит управляющие классы.

### 3. Группировка классов в пакеты

Группировка классов в пакеты осуществляется путем перетаскивания в Логическом представлении браузера соответствующего класса в соответствующий пакет. Группировать созданные классы будем следующим образом:

- классы *Client*, *Order*, *OrderItem* и *ComponentPart* перенесем в пакет *Entities*;
- классы *OrderOptions* и *AddNewOrder* перенесем в пакет *Boundary*;
- класс *OrderManager* перенесем в пакет *Control*.

Итоговый результат приведен на рис. 5.

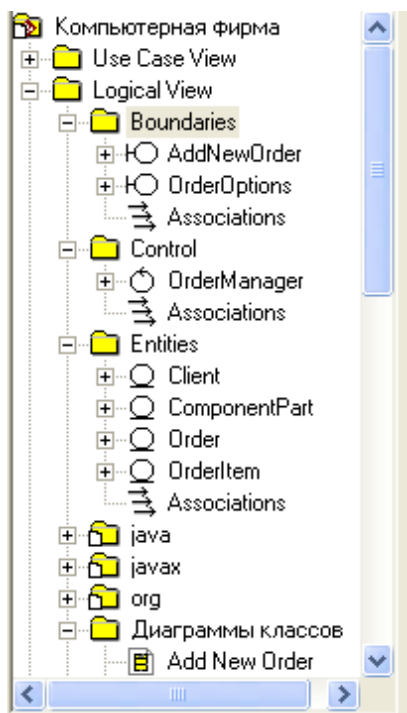


Рисунок 5. Классы и пакеты для сценария "Добавление нового заказа"

#### 4. Добавление диаграммы классов для каждого пакета

Для добавления диаграммы к пакету следует щелкнуть правой кнопкой мыши по пакету, в появившемся контекстном меню выбрать пункт New > Class Diagram (Создать > Диаграмма Классов), ввести имя класса *Main* (Главная), далее открыть диаграмму, дважды щелкнув по ней, и перенести на нее нужные классы. Отношения между классами, принадлежащие одному пакету, будут перенесены автоматически. Результат создания диаграммы классов для пакета *Boundaries* показан на рис.6, для пакета *Control* - на рис.7, для пакета *Entities* - на рис. 8.

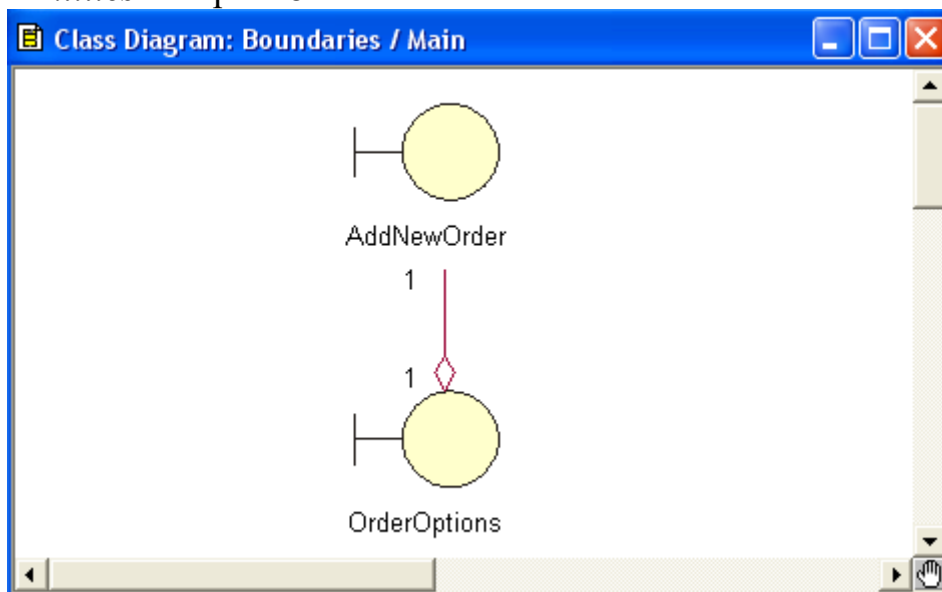


Рисунок 6. Диаграмма классов пакета Boundaries

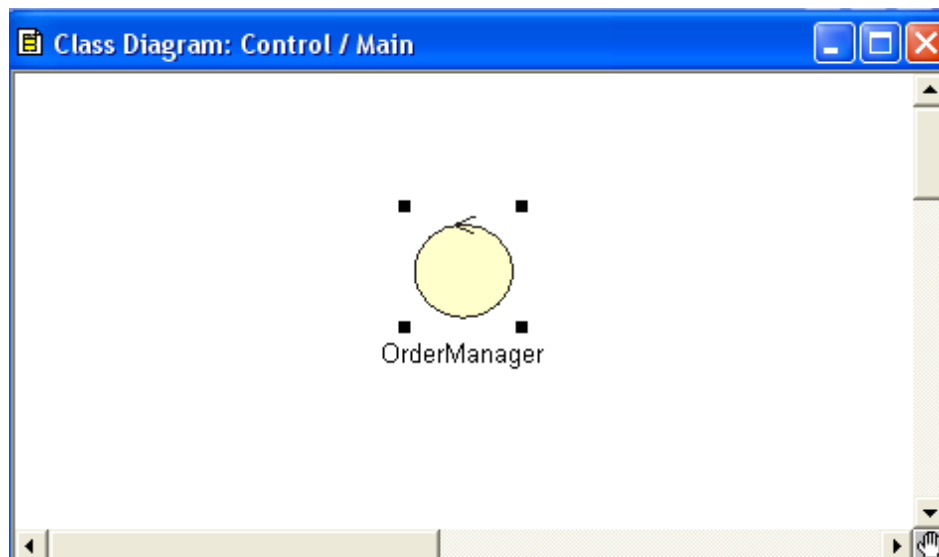


Рисунок 7. Диаграмма классов пакета Control

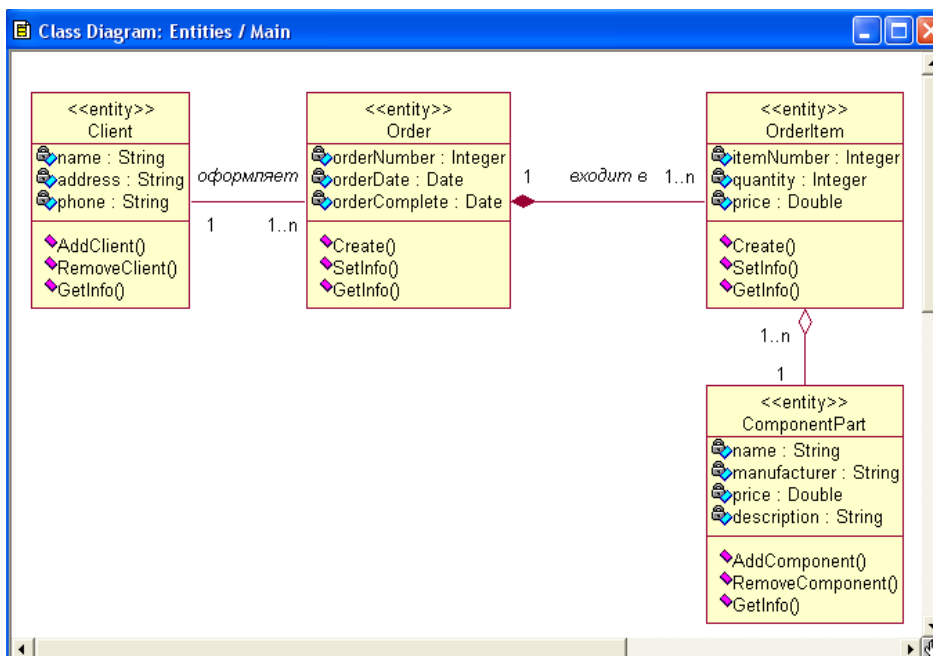


Рисунок 8. Диаграмма классов пакета Entities

## 5. Создание главной диаграммы классов

Главная диаграмма в логическом представлении модели обычно отображает пакеты системы. По умолчанию в Логическом представлении браузера уже существует главная диаграмма классов (*Main*). Для ее заполнения необходимо открыть ее, дважды щелкнув по ней в Логическом представлении браузера, и перетащить на нее три созданные нами пакеты (рис.9):

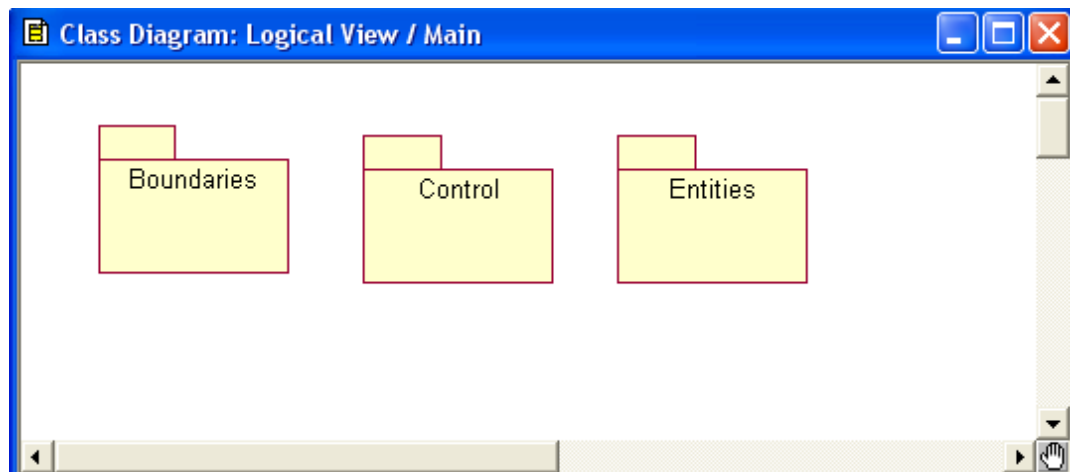


Рисунок 9. Главная диаграмма классов



## ПРАКТИЧЕСКАЯ РАБОТА №23

Тема: Документирование

**Цель Разработать документацию и зафиксировать знания на практике**

**Оборудование: ПК, Microsoft Office Word**

**Справочный материал:1,2.**

**Содержание работы**

1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

2. Постановка темы и цели урока

3. Повторение изученного материала

**Задание:**

1. Описать этапы проектирования модулей программы.
2. Составить в виде блок-схемы алгоритм решения задачи.
3. Составить отчет по практической работе.

Линейный алгоритм является аналогом обычного последовательного решения какой-либо задачи, когда все действия записываются поочередно. В программировании реализация линейного алгоритма является наиболее простой конструкцией, так как подразумевает выполнение всего трех этапов: **ввод** данных; **вычисления** с помощью операторов присваивания; **вывод** данных.

**Пример программного кода**

```
program summa;  
var a,b,s:real;  
begin  
  writeln('введите первое число'); readln(a);  
  writeln('введите второе число'); readln(b);  
  s:=a+b;  
  writeln('сумма чисел равна',s:5:2);  
  readln;  
end.
```

Рис. 1. Пример блок-схемы линейного алгоритма

Процедура, которая в режиме диалога с клавиатуры присваивает значение для переменной величины, называется процедурой ввода. В языке Паскаль для этой цели служит оператор **read (readln)**. Оба оператора ввода в Паскале идентичны по своему назначению, но отличие оператора **readln** заключается в том, что после своего завершения он переводит курсор на следующую экранную строку.

Процедура, которая выводит содержимое переменных на экран, называется процедурой вывода на экран. В Паскале используется оператор **write (writeln)** (различие аналогично оператору ввода **read** – есть или нет «перевода строки»).

С помощью линейных структур решаются различные профессиональные задачи обработки структурных данных разного типа. В данной работе необходимо решить задачи с применением линейных конструкций.

Задача А. Написать программу, которая запрашивает четыре целых числа и выдает на экран: сумму чисел; сумму квадратов чисел; среднее арифметическое чисел; сумму парных произведений.

Задача В. Написать программу вычисления третьей стороны треугольника по известным двум и углу между ними.

Задача С. Написать программу вычисления площади треугольника по его сторонам.

Программа работы

1. Записать полную форму линейной конструкции на языке программирования Паскаль.
2. Нарисовать блок-схему линейной конструкции.
3. Написать программный код для решения задач.
4. Открыть среду программирования на языке Паскаль.
5. Набрать программный код и протестировать его на нескольких данных.
6. Входные и выходные данные записать в отчет по работе.

Контрольные вопросы

1. Из каких этапов состоит линейная конструкция?
2. С помощью какого операторы осуществляется процедура ввода данных?
3. С помощью какого операторы осуществляется процедура вывода данных?
4. Чем отличаются операторы **read** и **readln**?
5. Как производится форматный вывод вещественных и целых чисел?

## **ПРАКТИЧЕСКАЯ РАБОТА №24**

Тема: Документирование

**Цель Разработать документацию и зафиксировать знания на практике**

**Оборудование: ПК, Microsoft Office Word**

**Справочный материал:1,2.**

**Содержание работы**

1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

2. Постановка темы и цели урока

3. Повторение изученного материала

**Задание:**

1. Создать интерфейс программы решения задачи с использованием любой визуальной среды программирования.
2. Отладить программу.

### **Общие положения**

**Microsoft Visual Studio** - интегрированная среда разработки программного обеспечения. Используется для разработки как консольных приложений, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

**Microsoft Visual Studio** дает возможность создавать программы в стиле визуального конструирования, т.е. пользователь оформляет свою будущую программу, и видит результаты своей работы еще до запуска самой программы.

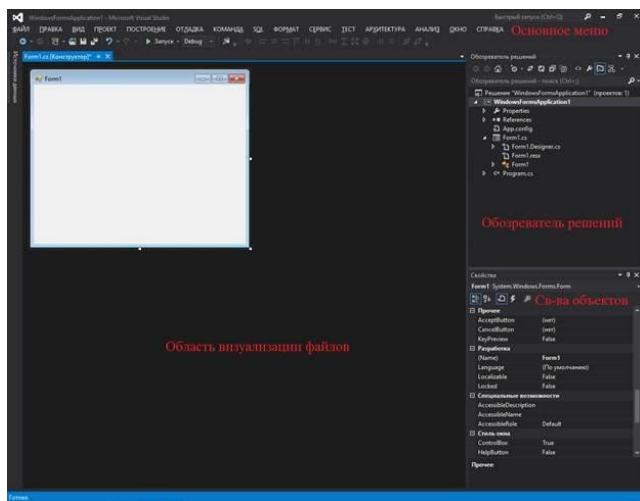
**Процесс написания приложения разделяется на две части:**

**Первая часть** – конструирование. Программист располагает на рабочей форме своей программы необходимые элементы объектов Visual Studio, называемых компонентами. Он позиционирует эти элементы, устанавливает нужные размеры, меняет свойства.

**Вторая часть** – написание программного кода. Программист описывает свойства элементов, доступных только во время работы приложения, реакцию на событие появление окна, нажатия на кнопку и т.п.

## Элементы интерфейса

После запуска Microsoft Visual Studio откроется главное окно среды разработки. Оно может отличаться в зависимости от версии, но основа остается не изменой. Мы будем рассматривать Microsoft Visual Studio 2012.



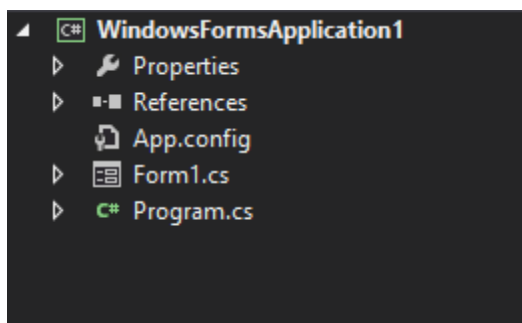
## Главное окно :

осуществляет основные функции управления проектом создаваемой программы. В нем находится основное меню и панели инструментов.

## Основное меню :

Содержит команды, необходимые для разработки и тестирования приложений и используется так же, как любое стандартное Windows-меню. Основное меню содержит несколько вложенных меню, в которых расположены различные команды:

Файл - действия над файлами: создание проекта, открытие проекта, закрытие проекта, обращение к системе управления исходным кодом проекта, выход из Visual Studio. **Команда Файл/Создать/Проект/ Visual C #/Приложение Windows Forms** создает новый проект. После выполнения этой команды в менеджере проектов можно увидеть все составляющие проекта в обозревателе решений.



**Properties** – конфигурационные

параметры формы.

### Свойства объектов :

Это окно предназначено для просмотра и внесения изменений в заданные во время разработки свойства и события выделенных компонентов. Эта панель имеет две вкладки – **Properties** (Свойства) и **Events** (События). На первой вкладке (Properties) постоянно отображаются все доступные свойства выбранного компонента. В левой колонке содержится список, а в правой – текущие значения по умолчанию.

На второй закладке (Events) отображаются возможные обработчики событий для выбранного компонента. В левой колонке – названия, а в правой – соответствующие свойства или процедуры. Можно найти окно **Свойства** в меню **Вид**. Также его можно открыть, нажав клавишу F4.

### Окно редактора кода :

Представляет собой текстовый редактор с подсветкой синтаксиса языка программирования – различные элементы языка окрашиваются различными цветами для удобства чтения кода.

Включается при нажатии на F7 или правой кнопкой мыши.

### Примечание

1. Сохранить проект можно так же с помощью команды **Файл/Сохранить все**
2. Для каждого нового проекта лучше отводить отдельную папку, т.к. проекты включают множество файлов. Это позволит вам упростить процесс копирования проекта на другой компьютер.
3. Откомпилируйте и выполните «пустую» программу, созданную в п.1 – на клавиатуре нажмите кнопку **F6**. В результате компиляции создается exe файл, который можно будет запускать на выполнение.
4. Закройте запущенную программу, щелкнув мышью на стандартной кнопке «X».



## ПРАКТИЧЕСКАЯ РАБОТА №25

Тема: Документирование

**Цель Разработать документацию и зафиксировать знания на практике**

**Оборудование: ПК, Microsoft Office Word**

**Справочный материал:1,2.**

**Содержание работы**

1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

2. Постановка темы и цели урока

3. Повторение изученного материала

Задание:

1. Сформировать диаграмму вариантов использования
2. Сгенерировать набор тестов

3. Разработка диаграммы вариантов использования преследует цели:

4. · Сформулировать общие требования к функциональному поведению проектируемой системы.

5. · Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей.

6. · Подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

7. Конструкция или стандартный элемент языка UML **вариант использования (use case)** применяется для спецификации общих особенностей поведения системы или любой другой сущности предметной области без рассмотрения внутренней структуры этой сущности. Каждый вариант использования определяет последовательность действий, которые должны быть выполнены проектируемой системой при взаимодействии ее с соответствующим актером. Диаграмма вариантов может дополняться пояснительным текстом, который раскрывает смысл или семантику составляющих ее компонентов. Такой пояснительный текст получил название примечания или **сценария**.

8. Работа над проектом в среде Rational Rose начинается с общего анализа проблемы и построения диаграммы вариантов использования, который отражает функциональное назначение проектируемой программной системы.

9. Для разработки диаграммы вариантов использования в среде **Rational Rose** необходимо активизировать соответствующую диаграмму в окне диаграммы. Это можно сделать различными способами:

10. · Раскрыть представление вариантов использования в браузере (Use Case View) и дважды щелкнуть на пиктограмме Main (Главная).

Через пункт меню Browse à Use Case Diagram (Браузер - Диаграмма вариантов использования).

При этом открывается основное окно системы со специальной панелью инструментов, содержащей графические примитивы, характерные для разработки диаграммы вариантов использования (рис. 1).

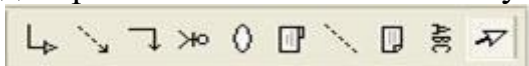
На этой панели инструментов присутствуют все необходимые для построения диаграммы вариантов использования элементы. Там же находится панель с набором инструментов – графических примитивов.

Для добавления элемента нужно нажать кнопку с изображением соответствующего примитива, после чего щелкнуть мышью на свободном месте диаграммы. На диаграмме появится изображение выбранного элемента с маркерами изменения.

Имя элемента может быть изменено. По щелчку правой кнопкой мыши на выбранном элементе вызывается контекстное меню элемента, среди опций которого имеется пункт Open Specification (Открыть спецификацию). Открывается окно, в поля которых можно занести всю информацию по данному элементу.

Основное окно системы при разработке диаграммы вариантов использования приведено ниже. Пример построенной таким способом диаграммы вариантов использования может иметь следующий вид:

Для удаления элемента не только из диаграммы, но и из модели в целом необходимо щелкнуть правой кнопкой мыши на удаляемый элемент на диаграмме и воспользоваться пунктом меню à Delete from Model.



Вариант использования

Вариант использования обозначается на диаграмме эллипсом, который содержит его краткое название или имя в форме глагола с пояснительными словами:



Проверить состояние счета  
клиента банка

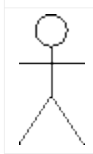
Цель варианта использования заключается в том, чтобы определить законченный аспект или фрагмент поведения некоторой сущности без раскрытия внутренней структуры этой сущности. В качестве такой сущности может выступать исходная система или любой другой элемент модели, который обладает собственным поведением, подобно подсистеме или классу в модели системы.

Актер

Актер представляет собой любую внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач. При этом актеры служат для обозначения



согласованного множества ролей, которые могут играть пользователи в процессе взаимодействия с проектируемой системой. Каждый актер может рассматриваться как некая отдельная роль относительно конкретного варианта использования. Стандартным графическим обозначением актера на диаграммах является фигурка "человечка", под которой записывается конкретное имя актера:



В некоторых случаях актер может обозначаться в виде прямоугольника класса с ключевым словом "актер" и обычными составляющими элементами класса. Имена актеров должны записываться заглавными буквами и следовать рекомендациям использования имен для типов и классов модели.

Отношения на диаграмме вариантов использования

Между компонентами диаграммы вариантов использования могут существовать различные отношения, которые описывают взаимодействие экземпляров одних актеров и вариантов использования с экземплярами других актеров и вариантов. Один актер может взаимодействовать с несколькими вариантами использования. В этом случае этот актер обращается к нескольким сервисам данной системы. В свою очередь один вариант использования может взаимодействовать с несколькими актерами, предоставляя для всех них свой сервис.

В языке UML имеется несколько стандартных видов отношений между актерами и вариантами использования:

- Отношение ассоциации (**association relationship**).
- Отношение расширения (**extend relationship**).
- Отношение обобщения (**generalization relationship**).
- Отношение включения (**include relationship**).

При этом общие свойства вариантов использования могут быть представлены тремя различными способами, а именно, с помощью отношений расширения, обобщения и включения.

Отношение ассоциации

**Отношение ассоциации** является одним из фундаментальных понятий в языке UML и в той или иной степени используется при построении всех графических моделей систем в форме канонических диаграмм. Таким образом, это отношение устанавливает, какую конкретную роль играет актер при взаимодействии с экземпляром варианта использования. На диаграмме вариантов использования, так же как и на других диаграммах, отношение ассоциации обозначается сплошной линией. Эта линия может иметь дополнительные условные обозначения, такие, например, как имя и кратность:



Кратность (**multiplicity**) ассоциации указывается рядом с обозначением компонента диаграммы, который является участником данной ассоциации. Кратность характеризует общее количество конкретных экземпляров данного компонента, которые могут выступать в качестве элементов данной ассоциации. Применительно к диаграммам вариантов использования кратность имеет специальное обозначение в форме одной или нескольких цифр и, возможно, специального символа "\*" (звездочка).

Кратность можно установить, щелкнув правой кнопкой мыши на линию и выбрав из раскрывшегося меню пункт **Multiplicity**.

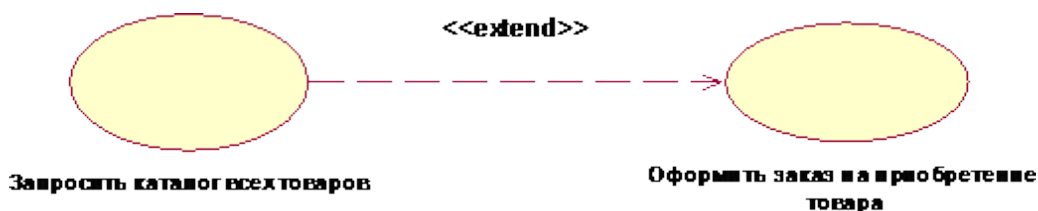
Для диаграмм вариантов использования наиболее распространенными являются четыре основные формы записи кратности отношения ассоциации:

1. Целое неотрицательное число (включая цифру 0).
2. Два целых неотрицательных числа, разделенные двумя точками и записанные в виде: "первое число.. второе число".
3. Два символа, разделенные двумя точками. При этом первый из них является целым неотрицательным числом или 0, а второй — специальным символом "\*".
4. Единственный символ "\*", который является сокращением записи интервала "0..\*".

Отношение расширения

**Отношение расширения** определяет взаимосвязь экземпляров отдельного варианта использования с более общим вариантом, свойства которого определяются на основе способа совместного объединения данных экземпляров. Так, если имеет место отношение расширения от варианта использования А к варианту использования В, то это означает, что свойства экземпляра варианта использования В могут быть дополнены благодаря наличию свойств у расширенного варианта использования А.

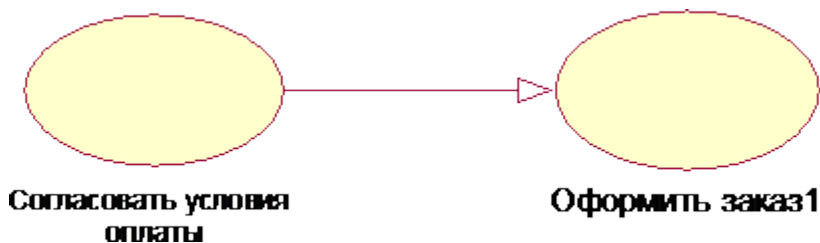
Отношение расширения между вариантами использования обозначается пунктирной линией со стрелкой (вариант отношения зависимости), направленной от того варианта использования, который является расширением для исходного варианта использования. Данная линия со стрелкой помечается ключевым словом "extend" ("расширяет):



Отношение расширения отмечает тот факт, что один из вариантов использования может присоединять к своему поведению некоторое дополнительное поведение, определенное для другого варианта использования.

Отношение обобщения

Отношение обобщения служит для указания того факта, что некоторый вариант использования А может быть обобщен до варианта использования В. В этом случае вариант А будет являться специализацией варианта В. При этом В называется предком или родителем по отношению А, а вариант А - потомком по отношению к варианту использования В. Следует подчеркнуть, что потомок наследует все свойства и поведение своего родителя, а также может быть дополнен новыми свойствами и особенностями поведения. Графически данное отношение обозначается сплошной линией со стрелкой в форме незакрашенного треугольника, которая указывает на родительский вариант использования. Эта линия со стрелкой имеет специальное название — стрелка "обобщение".



Между отдельными актерами также может существовать отношение обобщения. Данное отношение является направленным и указывает на факт специализации одних актеров относительно других. Например, отношение обобщения от актера А (кассира) к актеру В (служащему банка) отмечает тот факт, что каждый экземпляр актера А является одновременно экземпляром актера В и обладает всеми его свойствами. В этом случае актер В является родителем по отношению к актеру А, а актер А. соответственно потомком актера В. При этом актер А обладает способностью играть такое же множество ролей, что и актер В. Графически данное отношение также обозначается стрелкой обобщения, т. е. сплошной линией со стрелкой в форме незакрашенного треугольника, которая указывает на родительского актера:



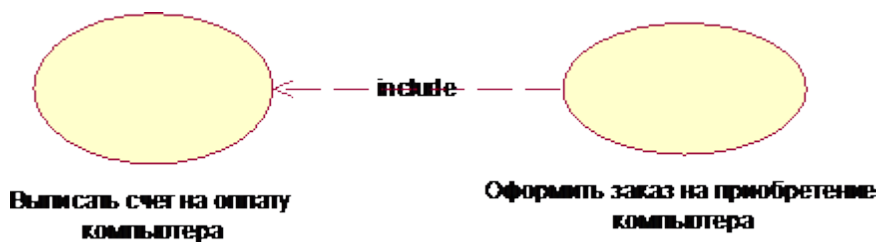
Отношение включения

Отношение включения между двумя вариантами использования указывает, что некоторое заданное поведение для одного варианта использования включается в качестве составного компонента в последовательность поведения другого варианта использования. Данное отношение является

направленным бинарным отношением в том смысле, что пара экземпляров вариантов использования всегда упорядочена в отношении включения.

Семантика этого отношения определяется следующим образом. Когда экземпляр первого варианта использования в процессе своего выполнения достигает точки включения в последовательность поведения экземпляра второго варианта использования, экземпляр первого варианта использования выполняет последовательность действий, определяющую поведение экземпляра второго варианта использования, после чего продолжает выполнение действий своего поведения.

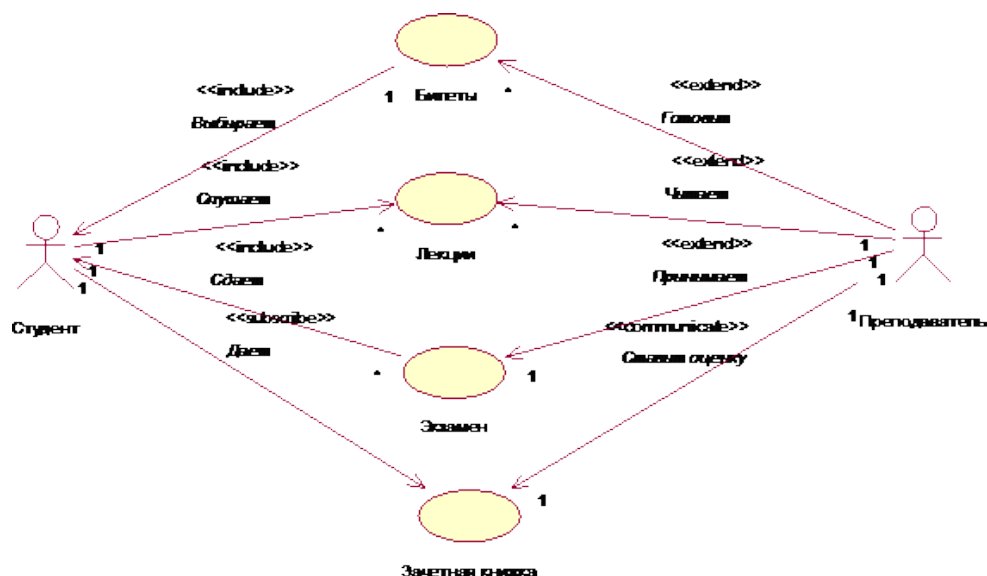
Отношение включения, направленное от варианта использования А к варианту использования В, указывает, что каждый экземпляр варианта А включает в себя функциональные свойства, заданные для варианта В. Эти свойства специализируют поведение соответствующего варианта А на данной диаграмме. Графически данное отношение обозначается пунктирной линией со стрелкой (вариант отношения зависимости), направленной от базового варианта использования к включаемому. При этом данная линия со стрелкой помечается ключевым словом "include" («включает»):



### Пример построения диаграммы вариантов использования

Пример модели «Сдача экзамена»:

Получить полный текст



Подготовить вариант в соответствии с заданием:

1. Пассажир бронирует билет на рейс у агента. Актеры: **пассажир с атрибутами**: Имя, фамилия, адрес, №паспорта, город вылета, город прилета; **с операциями**: заказать, купить. **Агент с атрибутами**: Фамилия, номер агента, **с операциями**: бронировать, продать.
2. Клиент сдает автомобиль в автосервис. Актеры: **Клиент с атрибутами**: Фамилия, марка машины, пробег, неисправность; **с операциями**: сдать в ремонт, взять из ремонта, **приемщик с атрибутами**: фамилия, дата приема, дата выдачи, **с операциями**: принять машину, выдать машину; **слесарь с атрибутами**: фамилия, специализация.
3. Покупатель покупает книгу в книжном магазине. Актеры: **покупатель с атрибутами**: специальность, интерес, **продавец с атрибутами**, **кладовщик с атрибутами**.
4. Актеры: **Гостиница**: содержит данные о номерах: (порядковый номер, количество мест в номере, этаж, удобства (телевизор, холодильник, телефон и т. д.), стоимость (с завтраком или без); (б) **Постоялец с атрибутами**: имя, фамилия, адрес, номер, им занимаемый, дата приезда, дата отъезда, состояние оплаты; **Служащий гостиницы**: выдает справки о свободных номерах и о конкретном номере; справки о доходах гостиницы за год, месяц, день.
5. Пассажир приходит на регистрацию рейса в аэропорт. Актеры: **пассажир с атрибутами**: фамилия, дата вылета, город прилета, **агент с атрибутами**, **приемщик багажа с атрибутами**.

#### Контрольные вопросы

1. Для чего используются диаграммы языка UML?
2. Что такое вариант использования?
3. Что такое диаграмма вариантов использования?
4. Как задается кратность ассоциации?
5. Какие существуют ассоциации?

## ПРАКТИЧЕСКАЯ РАБОТА №26

Тема: Документирование

**Цель Разработать документацию и зафиксировать знания на практике**

**Оборудование: ПК, Microsoft Office Word**

**Справочный материал:1,2.**

**Содержание работы**

1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

2. Постановка темы и цели урока

3. Повторение изученного материала

**Задание:**

1. Составить в виде блок-схемы алгоритм решения задачи.
2. Создать программу решения задачи на любом алгоритмическом языке программирования.
3. Отладить программу.

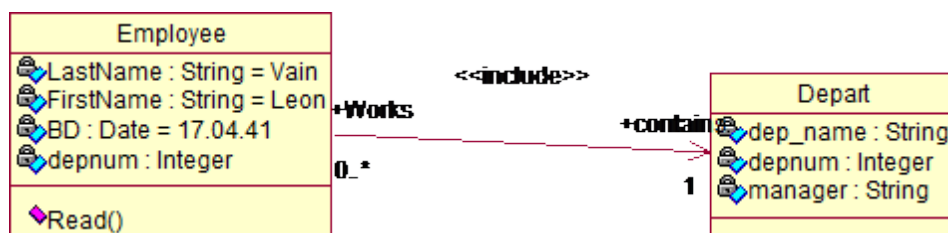
Как создать заголовок и тело компонента

1. Двойным щелчком на элементе дерева в окне Browser, представляющем диаграмму компонентов, открыть окно диаграммы.
2. Расположить курсор мыши над элементом диаграммы, отвечающим требуемому компоненту, и щелкнуть правой кнопкой, чтобы активизировать контекстное меню.
3. Выбрать элемент меню Open Specification.
4. Перейти на вкладку General диалогового окна Component Specification.
5. В раскрывающемся списке Stereotype выбрать значение стереотипа Package Specification для файла заголовка компонента либо значение Package Body - для файла, содержащего тело кода компонента.
6. Закрыть диалоговое окно щелчком на кнопке **ОК**.

Ниже приводится пример диаграммы классов для генерации кода

Создаем диаграмму классов.

Для этого: Основное меню – Browse à Class Diagram.

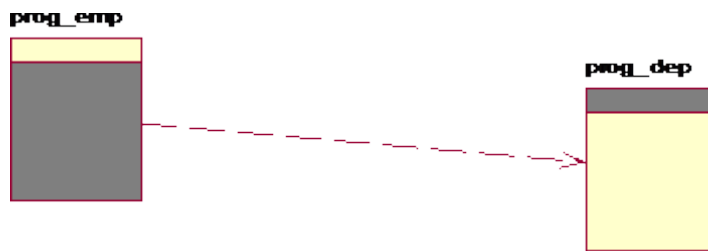


Создаем диаграмму компонентов соответствующую диаграмме классов.

Для этого: Основное меню – Browse à Component Diagram.

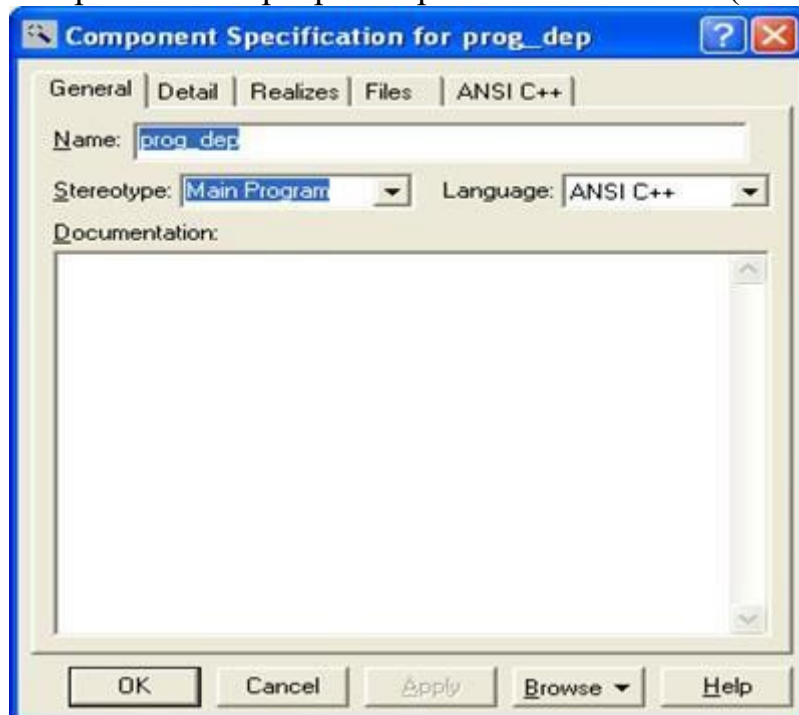
Открывается окно диаграммы компонентов.

Выбираем элементы диаграммы и соединяем их связью Dependency (Зависимость):



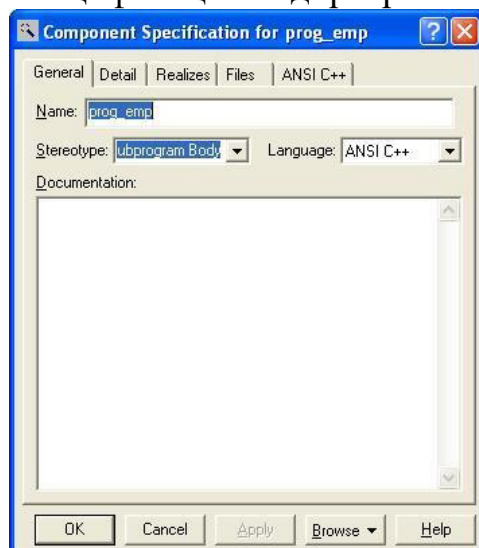
Спецификация основной программы prog\_dep stereotype (Main Program).

Выбрали язык программирования ANSI C++ (Language ANSI C++).



Выбрали язык программирования ANSI C++ (Language ANSI C++).

Спецификация подпрограммы prog\_emp stereotype (Subprogram body):



Сопоставляем элемент диаграммы классов элементу диаграммы компонентов.

Для этого от названия элемента диаграммы классов (левое окно Browse) правой кнопкой мыши протягиваем на название элемента диаграммы компонентов в правом окне.

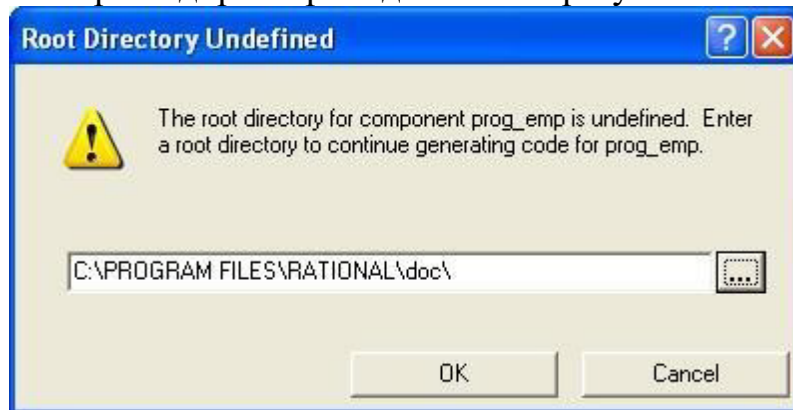
На диаграмме классов появляется двойное название:

Employee (Prog\_emp)

Depart (prog\_dep)

Дальше: Tools → ANSI C++ → Generate code или правой кнопкой на элементе диаграммы компонентов и ANSI C++ → Generate code.

Выбираем директорию для записи результата генерирования:



Сообщение о завершении генерации кодов (Code Generation Complete):



Сообщение о результатах генерирования кодов:

11:46:48| Changed 2 files: C:\PROGRAM FILES\RATIONAL\doc\Employee. cpp,

C:\PROGRAM FILES\RATIONAL\doc\Employee. h

11:13:13| Changed 2 files: C:\PROGRAM FILES\RATIONAL\doc\Depart. cpp,

C:\PROGRAM FILES\RATIONAL\doc\Depart. h

Сгенерированные программы Depart (Header и C++):

Сгенерированные программы Employee (Header и C++):

Файл **Header**:

```
#ifndef DEPART_H_HEADER_INCLUDED_B278F74C
#define DEPART_H_HEADER_INCLUDED_B278F74C
///ModelId=4D
class Depart
{
///ModelId=4DCE
String dep_name;
```



```

///##ModelId=4DC8
Integer depnum;
///##ModelId=4DAF
String manager;
};
#endif /* DEPART_H_HEADER_INCLUDED_B278F74C */
Файл C++
#ifndef DEPART_H_HEADER_INCLUDED_B278F74C
#define DEPART_H_HEADER_INCLUDED_B278F74C
///##ModelId=4D
class Depart
{
///##ModelId=4DCE
String dep_name;
///##ModelId=4DC8
Integer depnum;
///##ModelId=4DAF
String manager;
};
#endif /* DEPART_H_HEADER_INCLUDED_B278F74C */
Программа Employee (Header и C++):
{
Файл Header:
#ifndef EMPLOYEE_H_HEADER_INCLUDED_B278A382
#define EMPLOYEE_H_HEADER_INCLUDED_B278A382
class Depart;
///##ModelId=4D87047E0290
class Employee
{
public:
///##ModelId=4D
Read();
///##ModelId=4D8705B401F4
Depart *contains;
private:
///##ModelId=4D8704AE005D
String LastName;
///##ModelId=4D8704D901B5
String FirstName;
///##ModelId=4D8704EF0290
Date BD;
///##ModelId=4DAB
Integer depnum;
};
#endif /* EMPLOYEE_H_HEADER_INCLUDED_B278A382 */

```

Файл C++:

```
#include "Employee. h"
```

```
#include "Depart. h"
```

```
///##ModelId=4D
```

```
Employee::Read()
```

Варианты заданий

1. Пассажир бронирует билет на рейс у агента. Классы: **Пассажир с атрибутами:** имя, фамилия, адрес, №паспорта, город вылета, город прилета; **с операциями:** заказать, купить. **Агент с атрибутами:** фамилия, номер агента, **с операциями:** бронировать, продать.

2. Клиент сдает автомобиль в автосервис. Классы: **Клиент с атрибутами:** фамилия, марка машины, пробег, неисправность; **с операциями:** сдать в ремонт, взять из ремонта, **приемщик с атрибутами:** фамилия, дата приема, дата выдачи, **с операциями:** принять машину, выдать машину; **слесарь с атрибутами:** фамилия, специализация.

3. Покупатель покупает книгу в книжном магазине. Актеры: **Покупатель с атрибутами:** специальность, интерес, **продавец с атрибутами, кладовщик с атрибутами**

4. Пассажир приходит на регистрацию рейса в аэропорт. Актеры: **Пассажир с атрибутами:** фамилия, дата вылета, город прилета, **агент с атрибутами, приемщик багажа с атрибутами.**

## **ПРАКТИЧЕСКАЯ РАБОТА №27**

Тема: Документирование

**Цель Разработать документацию и зафиксировать знания на практике**

**Оборудование: ПК, Microsoft Office Word**

**Справочный материал:1,2.**

**Содержание работы**

1. Организационный момент

- Проверка готовности учащихся к уроку.
- Приветствие.
- Проверка готовности ребят к уроку

2. Постановка темы и цели урока

3. Повторение изученного материала

**Задание:**

1. Оформить внешнюю спецификацию.

2. Составить в виде блок-схемы алгоритм решения задачи.

3. Создать программу решения задачи на любом алгоритмическом языке программирования.

### **Вариант 1**

Данные о преподавателях, учебных дисциплинах и группах

Исходные данные:

(а) Список преподавателей (ФИО, кафедра, должность, номер\_преп, дата приема на работу);

(б) список дисциплин (название, код, семестр, специальность);

(в) список дисциплин кафедры (код, номер\_преп, количество часов).

### **Вариант 2**

Обслуживание клиентов видеокассетами

Исходные данные:

(а) сведения о видеофильмах: (компания-производитель, название, год выпуска, основные исполнители, характер фильма (боевик, триллер и т. д.));

(б) сведения о компании-производителе: страна, город, название, год основания;

(в) данные о выдачах: номер фильма, фамилия и адрес клиента, дата выдачи, дата возвращения, залог, оплата.

### **Вариант 3**

Деятельность отдела персонала

Исходные данные:

(а) сведения о сотрудниках: (имя, фамилия, номер отдела, номер должности, дата приема, семейное положение, образование, пол, адрес);

(б) список должностей: (название, номер должности, вилка оклада (напр. р.));

(в) список отделов: (название, номер отдела, руководитель).

## **Вариант 4**

Деятельность книжного магазина

Исходные данные:

(а) Данные о продаваемых книгах: название, автор, год выпуска, тематика, дата поступления в магазин, количество экземпляров;

(б) ежедневный отчет о проданных книгах: дата, автор, название, количество экземпляров, время продажи;

(в) предложения на поставку книг: название, автор, год выпуска, тематика, количество экземпляров.

Отчет должен содержать

1. Название лабораторной работы.
2. Цель работы и краткое ее описание.
3. Вариант задания.

Контрольные вопросы

1. Что такое технология ?
2. Как установить связь приложения с базой данных?
3. Как создать набор данных DataSet?
4. Что такое объект DataAdapter?

## **Информационное обеспечение обучения**

### **Печатные издания**

#### **Основные учебные издания**

1. Аблязов, Р.З. Программирование на ассемблере на платформе x86-64/Р.З. Аблязов. — 2-е изд. — Саратов: Профобразование, 2019. — 301 с. — ISBN 978-5-4488-0117-4. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: <https://www.iprbookshop.ru/88005.html>

2. Введение в разработку приложений для ОС Android: учебное пособие / Ю. В. Березовская, О. А. Юфрякова, В. Г. Вологодина [и др.]. — 3-е изд. — Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. — 427 с. — ISBN 978-5-4497-0890-8. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: <https://www.iprbookshop.ru/102000.html>

3. Зыков, С. В. Введение в теорию программирования. Объектно-ориентированный подход: учебное пособие для СПО / С. В. Зыков. — Саратов: Профобразование, 2021. — 187 с. — ISBN 978-5-4488-0995-8. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/102188>

4. Кариев, Ч. А. Разработка Windows-приложений на основе Visual C#: учебное пособие / Ч. А. Кариев. — 3-е изд. — Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. — 978 с. — ISBN 978-5-4497-0909-7. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: <https://www.iprbookshop.ru/102057.html>

5. Котляров, В.П. Основы тестирования программного обеспечения: курс лекций / Котляров В.П. — Москва: Интуит НОУ, 2016. — 348 с. — ISBN 978-5-9556-0027-7. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: <https://book.ru/book/917951>

6. Лебедева, Т. Н. Технология программирования: учебное пособие для СПО/ Т. Н. Лебедева, С. С. Юнусова. — Саратов: Профобразование, 2019. — 140 с. — ISBN 978-5-4488-0351-2. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/86081>

#### **Дополнительные учебные издания:**

7. Авдеев, В. А. Периферийные устройства: интерфейсы, схемотехника, программирование / В. А. Авдеев. — 2-е изд. — Саратов: Профобразование, 2019. — 848 с. — ISBN 978-5-4488-0053-5. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: <https://www.iprbookshop.ru/88002.html>

8. Биллиг, В. А. Основы программирования на C#: учебное пособие / В. А. Биллиг. — 3-е изд. — Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. — 573 с. — ISBN 978-5-4497-0893-9. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/102033>

9.Зубкова, Т. М. Технология разработки программного обеспечения: учебное пособие для СПО / Т. М. Зубкова. — Саратов: Профобразование, 2019. — 468 с. — ISBN 978-5-4488-0354-3. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/86208>

10.Пирская, Л. В. Разработка мобильных приложений в среде AndroidStudio: учебное пособие / Л. В. Пирская. — Ростов-на-Дону, Таганрог: Издательство Южного федерального университета, 2019. — 123 с. — ISBN 978-5-9275-3346-6. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: <https://www.iprbookshop.ru/100196.html>

11.Семакова, А. Введение в разработку приложений для смартфонов на ОС Android: учебное пособие для СПО/ А. Семакова. — Саратов: Профобразование, 2021. — 102 с. — ISBN 978-5-4488-0994-1. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/102187>

#### **Электронные издания (электронные ресурсы)**

12. Учебники по программированию <http://programm.ws/index.php>