МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

Колледж Пятигорского института (филиал) СКФУ

МДК.02.03 ПОДДЕРЖКА И ТЕСТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ

Специальности СПО

09.02.11 Разработка и управление программным обеспечением

Пятигорск 2026г.

Методические указания для выполнения лабораторных работ по дисциплине МДК.02.03 Поддержка и тестирование программных модулей составлены в соответствии с требованиями ФГОС СПО. Предназначены для студентов, обучающихся по специальности 09.02.11 Разработка и управление программным обеспечением.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Программа дисциплины «Поддержка и тестирование программных модулей» предусматривает изучение поддержки и тестирования программных модулей.

При изучении предмета следует соблюдать единство терминологии и обозначения в соответствии с действующими стандартами, Международной системной единицы (СИ). В результате изучения дисциплины «Поддержка и тестирование программных модулей» студенты *должны* **знать**:

- основные виды и процедуры обработки информации, модели и методы решения задач обработки информации;
- основные платформы для создания, исполнения и управления информационной системой;
- основные процессы управления проектом разработки;
- основные модели построения информационных систем, их структуру, особенности и области применения;
- методы и средства проектирования, разработки и тестирования информационных систем;
- систему стандартизации, сертификации и систему обеспечения качества продукции. **уметь**:
- осуществлять постановку задач по обработке информации;
- проводить анализ предметной области;
- осуществлять выбор модели и средства построения информационной системы и программных средств;
- использовать алгоритмы обработки информации для различных приложений;
- решать прикладные вопросы программирования и языка сценариев для создания программ;
- разрабатывать графический интерфейс приложения;
- создавать и управлять проектом по разработке приложения;
- проектировать и разрабатывать систему по заданным требованиям и спецификациям. **иметь практический опыт в**:
- управлении процессом разработки приложений с использованием инструментальных средств;
- обеспечении сбора данных для анализа использования и функционирования информационной системы;
- программировании в соответствии с требованиями технического задания;
- использовании критериев оценки качества и надежности функционирования информационной системы;
- применении методики тестирования разрабатываемых приложений;
-определении состава оборудования и программных средств разработки информационной системы;
- разработке документации по эксплуатации информационной системы;

- проведении оценки качества и экономической эффективности информационной системы в рамках своей компетенции;
- модификации отдельных модулей информационной системы.

В результате освоения учебной дисциплины студент должен овладевать:

Общими компетенциями:

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 правовой и 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 российских духовно-нравственных 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.

Профессиональными компетенциями:

ПК 2.1. Проектировать модули программного обеспечения

ПК 2.2. Разрабатывать модули программного обеспечения

ПК 2.3. Выполнять интеграцию модулей и компонентов программного обеспечения

ПК 2.4. Выполнять тестирование и отладку программного обеспечения

ПК 2.5. Осуществлять документирование программных модулей программного обеспечения

ОБЩИЕ МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

По дисциплине Поддержка и тестирование программных модулей лабораторные работы содержат задачи и теоретические вопросы. Варианты для каждого обучающегося – индивидуальные.

Задачи и ответы на вопросы, выполненные не по своему варианту, не засчитываются.

Лабораторная работы выполняется в отдельной тетради. Условия задачи и формулировки вопросов переписываются полностью. Формулы, расчеты, ответы на вопросы пишутся ручкой, а чертежи, схемы и рисунки выполняются карандашом, на графиках и диаграммах указывается масштаб. Вначале задача решается в общем виде, затем делаются расчёты по условию задания. Решение задач обязательно ведется в Международной системе единиц (СИ).

При выполнении лабораторной работы необходимо следовать методическим указаниям: повторить краткое содержание теории, запомнить основные формулы и законы, проанализировать пример выполнения аналогичного задания, затем преступить непосредственно к решению задачи. К зачету допускаются студенты, получившие положительные оценки по всем лабораторным работам.

Правила выполнения лабораторных работ.

1. Студент должен прийти на лабораторную работу подготовленным к выполнению лабораторной работы.
2. Каждый студент после проведения работы должен представить отчет о проделанной работе с анализом полученных результатов и выводом по работе.
3. Таблицы и рисунки следует выполнять с помощью чертежных инструментов (линейки, циркуля, и.т.д.) карандашом с соблюдением ЕСКД.
4. Расчет следует проводить с точностью до двух значащих цифр.
5. Исправления проводить на обратной стороне листа. При мелких исправлениях неправильное слово (буква, число и т.п.) аккуратно зачеркивается и над ним пишут правильное пропущенное слово (букву, число и т.п.).
6. Вспомогательные расчеты можно выполнять на отдельных листах, а при необходимости на листах отчета.
7. Если студент не выполнит лабораторную работу или часть работы, то он выполнит ее во внеурочное время, согласованное с преподавателем.
8. Оценку по лабораторной работе студент получает с учетом срока выполнения работы, если;

- расчеты выполнены правильно и в полном объеме;
- сделан анализ проделанной работы и вывод по результатам работы;
- студент может пояснить выполнение любого этапа работы;
- отчет выполнен в соответствии с требованиями к выполнению работы.

Лабораторная работа №1

Тема: Анализ и оценка качества программного модуля с использованием метрик качества программных модулей

Цель: Получить практические навыки оценки качества программного модуля с помощью метрических показателей, выявить сильные и слабые стороны модуля, разработать рекомендации по улучшению его качества.

Теоеретическая часть:

Качество программного обеспечения — совокупность характеристик и атрибутов, определяющих способность программного продукта удовлетворять потребности и ожидания пользователей. Качество ПО оценивается с точки зрения технических характеристик, надежности, производительности, безопасности, сопровождаемости и удобства использования.

Метрики качества программных модулей

Метрики качества позволяют количественно оценить характеристики программного модуля и его соответствие установленным стандартам и требованиям. Существуют различные категории метрик:

1. Метрики размера:
   - LOC (Lines of Code) — Количество строк кода.
   - NLOC (Non-Commented Lines of Code) — Количество небезкомментированных строк кода.
   - Halstead Metrics — Сложность и объем программы, рассчитанные по количеству операторов и операндов.
2. Метрики сложности:
   - Cyclomatic Complexity (CC) — Цикломатическая сложность (число путей выполнения в программе).
   - McCabe's Metric — Оценка сложности по числу независимых путей.
   - Depth of Inheritance Tree (DIT) — Глубина наследования (для классов).
3. Метрики взаимосвязанности:
   - Coupling Between Objects (CBO) — Степень связи между классами (чем больше связей, тем сложнее сопровождение).
   - Lack of Cohesion in Methods (LCOM) — Отсутствие когезионности методов (низкая связь методов внутри класса).
4. Метрики сопровождения:
   - Maintainability Index (MI) — Индекс поддерживаемости (оценка поддержки программы по нескольким параметрам).
   - Change Risk Analysis and Predictions (CRAP) — Анализ риска изменений и предсказания возможных дефектов.

Процесс оценки качества модуля
Оценка качества программного модуля включает следующие шаги:

1. Выбор подходящего набора метрик для оценки качества.
2. Сбор данных по выбранным метрикам.
3. Анализ полученных данных и выявление слабых мест.
4. Формулировка рекомендаций по повышению качества.

Задание:

1. Выбор модуля для анализа.Выбрать программный модуль (Java-класс или группу классов), имеющий достаточную функциональность для анализа. Желательно, чтобы модуль имел минимум одну сложную логику (циклы, ветвления, взаимодействие с другими классами).
2. Определение набора метрик.Исходя из характера модуля, выберите ряд значимых метрик для оценки его качества. Рекомендуемые метрики:
    - Размер кода (NLOC, LOC).
    - Сложность (цикломатическая сложность, глубина наследования DIT).
    - Связанность (CBO, LCOM).
    - Поддерживаемость (Maintainability Index MI).
3. Сбор данных по метрикам.Используйте доступные инструменты для сбора метрик (SonarQube, Checkstyle, PMD). Зарегистрируйте значения каждой метрики отдельно.
4. Анализ полученных данных.Изучите результаты и сделайте выводы:
    - Высокие значения цикломатической сложности указывают на низкую читаемость и повышенную сложность поддержки.
    - Высокая взаимосвязанность классов затрудняет внесение изменений и усложняет рефакторинг.
    - Низкий показатель поддерживаемости свидетельствует о трудностях в сопровождении и развитии.
5. Формулировка рекомендаций по улучшению качества.Основываясь на анализе метрик, предложите меры по улучшению качества модуля:
    - Рефакторинг сложных участков кода.
    - Оптимизация зависимостей между классами.
    - Упрощение архитектуры для снижения сложности и увеличения поддерживаемости.

Пример оформления отчета:
Название модуля:Класс реализации алгоритма шифрования CaesarCipher.java
Используемые метрики:

- Локальная сложность (LOC/NLOC)
- Цикломатическая сложность (CC)

- Глубина наследования (DIT)
- Коэффициент внутренней связи (LCOM)
- Индекс поддерживаемости (MI)

Результаты расчета метрик:

- LOC: 250 строк
- NLOC: 200 строк
- CC: 15
- DIT: 1 уровень
- LCOM: 0,2
- MI: 80%

Лабораторная работа №2

Тема: Использование статического анализа кода для выявления дефектов

Цель: Изучение методов и инструментов статического анализа программного кода с целью обнаружения потенциальных ошибок и повышения качества разрабатываемого ПО.

Теоеретическая часть:

Статический анализ кода является одним из важнейших этапов тестирования информационной системы. Его задача заключается в выявлении дефектов и уязвимостей путем анализа исходного кода программы без запуска самого приложения.

Что такое статический анализ?

Статический анализ (Static Code Analysis) — это процесс автоматизированного исследования программного кода без фактического исполнения программы. Этот метод позволяет обнаружить дефекты на ранних этапах разработки, минимизируя риски появления проблем в продакшене и сокращая затраты на исправление багов позже.

Зачем нужен статический анализ?

Применение статического анализа обеспечивает следующие преимущества:

- Раннее выявление ошибок и потенциальной небезопасности.
- Улучшение читаемости и структурированности кода.
- Сокращение числа регрессий.
- Повышение общей надежности приложений.

Основные инструменты статического анализа

Существует большое количество специализированных инструментов для разных платформ и языков программирования. Среди наиболее популярных решений:

SonarQube

Это мощная система статического анализа, поддерживающая большинство современных языков программирования (Java, Python, C#, JavaScript и др.). SonarQube автоматически проверяет код на наличие

нарушений стандартов качества, возможных ошибок и антипаттернов проектирования.

ESLint (для JavaScript)

Один из самых распространенных инструментов для анализа JS-кода. ESLint обнаруживает стилистические нарушения, возможные проблемы производительности и безопасности.

Pylint (для Python)

Для проверки Python-приложений широко используется Pylint. Инструмент помогает находить синтаксические ошибки, неоптимальные конструкции и прочие типичные недостатки.

Checkstyle (для Java)

Этот инструмент предназначен специально для Java-разработчиков. Checkstyle контролирует соблюдение установленных соглашений по стилю написания кода, повышает качество продукта и снижает вероятность будущих ошибок.

Как проходит статический анализ?

Процесс включает несколько ключевых шагов:

1. Подготовка среды: Установка необходимого инструмента и настройка конфигурационных файлов проекта.
2. Запуск анализа: Автоматическое сканирование всех заданных директорий и файлов.
3. Обработка результатов: Анализ отчетов и фиксация найденных недостатков.
4. Исправления: Корректировка выявленных проблем согласно рекомендациям анализатора.
5. Контроль повторного анализа: Повторение процедуры для подтверждения устранения ошибок.

Задание:

Используя инструмент статического анализа (например, SonarQube, ESLint или аналогичный), выполните проверку реального проекта (или демонстрационного примера). Следуйте следующим этапам:

1. Установите выбранный вами инструмент статического анализа на локальной машине или настройте облачную версию (если используете онлайн-сервис).
2. Загрузите проект с открытым исходным кодом или подготовьте собственный тестовый пример.
3. Запустите анализ вашего проекта с использованием выбранного инструмента.
4. Получите отчёт с результатами анализа.
5. Проанализируйте отчёты и найдите самые распространённые типы ошибок или предупреждений.
6. Исправьте выявленные критические проблемы в коде.
7. Выполните повторный запуск анализа и убедитесь, что ранее зафиксированные ошибки устранены.

8. Подготовьте итоговую документацию, включающую описание процесса анализа, список найденных дефектов и рекомендации по улучшению качества проекта.

Лабораторная работа №3

Тема: Освоение методики подготовки тест-кейсов и оформления баг-репортов для эффективного выявления и документирования дефектов в информационных системах.

Цель: Изучение понятий абсолютный, измерительный и относительный уровни передачи

Теоеретическая часть:

Разработка качественных тест-кейсов и правильная регистрация дефектов являются ключевыми элементами успешного тестирования информационных систем. Правильно составленный тест-кейс позволит систематически проверять функциональность системы, а грамотно оформленный баг-репорт обеспечит четкое понимание характера возникшей проблемы разработчиками и руководителями проекта.
Тест-кейс
Тест-кейс (Test Case) представляет собой документированный сценарий, содержащий шаги действий, ожидаемые результаты и критерии успешности прохождения теста. Это своего рода инструкция для проверки конкретной функциональности или поведения системы.
Основные элементы тест-кейса:

- Идентификатор тест-кейса (ID)
- Название тест-кейса
- Описание сценария
- Шаги выполнения
- Ожидаемый результат
- Фактический результат
- Статус выполнения (успешно / неуспешно)
- Приоритет и уровень риска

Фото пример тест-кейса:

Номер	1
Заголовок	Отправка сообщения через форму обратной связи на странице “Контакты”
Предусловие	Открыта главная страница сайта victorz.ru. Есть доступ к почте администратора сайта victorz.ru
Шаг	**Ожидаемый результат**
В верхнем меню сайта нажать на ссылку “Контакты”	Открылась страница “Контакты”
Ввести значение в поле “Ваше имя” состоящее из латинских букв, кириллицы	В поле “Ваше имя” отображается введённое имя
Ввести корректный email в поле “Ваш e-mail”	В поле “Ваш e-mail” отображается введённый email
Ввести в поле “Тема” значение состоящее из латинских букв, кириллицы, спецсимволов и чисел	В поле “Тема” отображается введённый текст
Ввести в поле “Сообщение” значение состоящее из латинских букв, кириллицы, спецсимволов и чисел	В поле “Сообщение” отображается введённый текст
Ввести в поле капчи требуемое капчей значение	В поле капчи отображается введённое значение
Нажать под заполняемой формой на кнопку “Отправить”	Под кнопкой «Отправить» появился текст “Спасибо. Ваше сообщение было отправлено.” Все заполненные поля очищены.
Проверить почту администратора сайта	На почту пришло сообщение, отправленное с сайта через форму обратной связи и содержащее в теле сообщения данные введённые на шагах 1-5.

Баг-репорт

Баг-репорт (Bug Report) служит для фиксирования обнаруженных дефектов в продукте. Грамотно оформленное сообщение помогает команде разработчиков точно воспроизвести проблему и быстро её устранить.

Ключевые разделы баг-репорта:

- Уникальный номер дефекта
- Краткое описание проблемы
- Подробное пошаговое руководство для воспроизведения ошибки
- Факты (окружающая среда, версии компонентов, используемые устройства и ОС)
- Ожидаемый и полученный результат

- Уровень серьезности и приоритет исправления
- Скриншоты, видеозаписи или дополнительные файлы, подтверждающие дефект

Шаблон баг-репорта:

Поле	**описание**
Краткое описание	
Проект	
Версия	
Важность	
Срочность	
Статус	
автор	
исполнитель	
Шаги к воспроизведению	
Фактический результат	
Ожидаемый результат	
Дополнения	Сюда прикладываются скриншоты найденных багов

Задание:

1. Ознакомьтесь с описанием функционала изучаемой информационной системы.
2. Разработайте серию тест-кейсов для покрытия основных функций системы. Убедитесь, что каждый кейс чётко описывает шаги и ожидаемые результаты.
3. Проверьте систему вручную, следуя каждому разработанному тест-кейсу.
4. Если выявлены отклонения или сбои, создайте подробный баг-репорт, включая всю необходимую информацию для точной идентификации и воспроизводства ошибки.
5. Оформите отчет по результатам выполнения заданий, приложив сами тест-кейсы и зарегистрированные баги.

Лабораторная работа №4

Тема: Проведение полноценного код-ревью проекта

Цель: Овладеть техникой проведения качественного ревью исходного кода с целью выявления дефектов, улучшения структуры и стиля реализации проекта.

Теоеретическая часть:

Код-ревью (Code Review) — это процесс совместного изучения и оценки написанного программного кода коллегами-тестировщиками или разработчиками перед интеграцией изменений в основную ветку проекта. Код-ревью направлено на повышение качества программного обеспечения, предотвращение попадания дефектов в продакшн и обучение членов команды лучшим практикам программирования.

Важность код-ревью

Регулярное проведение ревью кода способствует нескольким ключевым аспектам:

- Обнаружению скрытых ошибок и узких мест, незаметных при автоматическом тестировании.
- Унификации подходов к разработке внутри команды.
- Поддержанию единого уровня качества и чистоты кода.
- Распространению лучших практик среди участников проекта.

Этапы проведения код-ревью

1. Получение задач на ревью. Обычно проводится выбор конкретных участков кода, нуждающихся в проверке.
2. Анализ кода. Ревьювер внимательно просматривает строки кода, обращая внимание на структуру, стиль, алгоритмы и реализацию требований.
3. Фиксация комментариев. Выявленные ошибки, неясности или несоответствия стандартам документируются в виде замечаний.
4. Обсуждение вопросов. Часто разработчики собираются вместе, чтобы обсудить спорные моменты и прийти к общему решению.
5. Коррекция замечаний. Авторы вносят правки согласно полученной обратной связи.
6. Перепроверка изменений. Проводится дополнительный просмотр исправленного участка кода для контроля правильности внесённых улучшений.

Ключевые метрики эффективности ревью

При оценке результата код-ревью полезно учитывать следующие показатели:

- Количество выявленных дефектов.
- Время на прохождение полного цикла ревью.
- Качество рекомендаций по оптимизации.
- Уровень согласия команды относительно принятых предложений.

Задание:

1. Выберите участок существующего открытого исходного кода или заранее подготовленную программу, подлежащую ревью.
2. Составьте перечень критериев оценки кода: архитектура, производительность, безопасность, удобство сопровождения и прочее.

3. Осуществите тщательное изучение выделенного фрагмента кода с точки зрения каждого критерия.
4. Зарегистрируйте все замеченные недостатки и советы по улучшению в форме отчёта.
5. Предложите оптимизированную версию рассматриваемого модуля.
6. Представьте коллегам результаты проведённого ревью и обсудите ключевые выводы.

Критерии оценки кода
Вот некоторые важные аспекты, которые следует учитывать при проведении код-ревью:

- Чёткость и ясность структуры: Логичность архитектуры и отсутствие запутанных конструкций.
- Производительность: Оптимальность используемых алгоритмов и эффективность потребления ресурсов.
- Безопасность: Отсутствие известных уязвимостей и защита конфиденциальных данных.
- Документированность: Наличие адекватных комментариев и документации.
- Соблюдение стилей и стандартов: Соответствие принятым правилам именования переменных, формату отступов и прочим соглашениям.

Лабораторная работа №5

Тема: Основные конструкции для обработки исключительных ситуаций

Цель: Изучение способов обработки исключительных ситуаций. Результатом практической работы является отчет, в котором должны быть приведены исходные коды программы, демонстрирующей умение обрабатывать исключения.

Теоеретическая часть:
Исключительная ситуация (исключение) возникает тогда, когда программа сталкивается с ошибкой, которая нарушает нормальное выполнение потока инструкций. Такие ситуации требуют специальной обработки для предотвращения сбоя программы и потери данных.
Исключения в Java
В Java существует два типа исключений:

1. Checked exceptions (контролируемые): исключения, которые компилятор требует обработать явно (IOException, SQLException, etc.).
2. Unchecked exceptions (неконтролируемые): наследники класса RuntimeException, которые не обязательно объявлять и обрабатывать явно (NullPointerException, ArrayIndexOutOfBoundsException, etc.).

Механизмы обработки исключений
Наиболее часто используемая конструкция для обработки исключений в Java — блок try-catch. Общий вид выглядит так:

```
try {
   // Потенциально опасный код
} catch(ExceptionType ex) {
   // Обработка исключения
} finally {
   // Необязательный блок для гарантированного выполнения
}
```

Также используются блоки throws и throw для объявления и выброса исключений соответственно.

Пример базового подхода к обработке исключений:

```
public class Main {
   public static void main(String[] args) {
      try {
         int divisionResult = safeDivide(10, 0); // делим на ноль
         System.out.println("Результат деления: " + divisionResult);
      } catch (ArithmeticException ae) {
         System.err.println("Ошибка вычислений: деление на ноль.");
      } finally {
         System.out.println("Программа завершила свою работу.");
      }
   }

   private static int safeDivide(int numerator, int denominator) throws ArithmeticException {
      if(denominator == 0) {
         throw new ArithmeticException("Деление на ноль недопустимо.");
      }
      return numerator / denominator;
   }
}
```

Здесь мы видим следующую последовательность:

- Бросаем собственное исключение (ArithmeticException) при попытке деления на ноль.
- Ловим его в блоке catch.
- Гарантируем завершение некоторых важных действий независимо от результата операции благодаря блоку finally.

Задание:

1. Напишите небольшую программу на Java, содержащую потенциально опасные участки (делением на ноль, доступ к массиву вне диапазона индексов, неверный ввод формата данных).
2. Используйте механизм обработки исключений (try-catch), чтобы перехватывать возникающие исключения и корректно реагировать на них.
3. Реализуйте собственную контролируемую ситуацию, бросив исключение самостоятельно через оператор throw.
4. Сделайте отчёт о проведенных экспериментах, подробно опишете наблюдаемое поведение программы при возникновении исключений.

Лабораторная работа №6

Тема: Практическое использование исключений в реальной задаче

Цель: Освоение практики применения механизмов обработки исключений в реальных проектах на примере Java для обеспечения устойчивости и надежности программных продуктов.

Теоеретическая часть:

Исключения представляют собой важный элемент любого современного языка программирования, позволяющий надежно контролировать ошибки и избегать неожиданных сбоев программы. Правильная обработка исключений делает программное обеспечение надежным и предсказуемым даже в сложных ситуациях.
Искусство обработки исключений
Правильно организованная обработка исключений предполагает следующее:

- Минимизацию незавершенных транзакций и повреждения данных.
- Предоставление информативных сообщений пользователям и специалистам поддержки.
- Возможность гибкого управления потоком выполнения программы.

Типы исключений в Java

- Checked Exceptions — это исключения, которые должны быть явно обработаны или заявлены методом, генерирующим их. Например, FileNotFoundException, IOException.
- Unchecked Exceptions — сюда относятся ошибки времени выполнения, которые необязательно явно указывать в сигнатуре метода. Например, NullPointerException, ArrayIndexOutOfBoundsException.

Базовые конструкции обработки исключений
Стандартная структура обработки исключений в Java состоит из блоков:

- try — здесь размещают потенциально опасный код, способный вызвать исключение.
- catch — этот блок захватывает возникшее исключение и реализует соответствующую обработку.
- finally — блок выполняется всегда, независимо от наличия исключения, полезен для освобождения ресурсов.

Примеры базовых конструкций:

```
try {
   // Опасный код
} catch(SomeException e) {
   // Обработка исключения
} finally {
   // Завершающие действия
}
```

Кроме того, иногда бывает удобно самим вызывать исключение при помощи оператора throw:

```
if (condition) {
   throw new SomeException("Описание причины исключения");
}
```

Пример использования исключений в реальности

Предположим, ваша программа взаимодействует с базой данных и пытается прочитать файл конфигурации. Возможные сценарии:

- Файл конфигурации отсутствует → бросить исключение FileNotFoundException
- Неправильный формат файла → создать кастомное исключение вроде InvalidConfigFormatException
- Проблема подключения к базе данных → вызвать стандартное исключение SQLException

Задание:

Создайте приложение на языке Java, которое моделирует реальный сценарий взаимодействия с файлом конфигурации и базой данных. Программа должна поддерживать правильную обработку исключений и выводить полезные сообщения при возникновении ошибок.

Алгоритм выполнения:

1. Создание простого приложения, которое работает с внешним файлом конфигурации.
2. Реализация нескольких классов, связанных с работой с файлами и сетевыми ресурсами (например, чтение JSON-файла и подключение к базе данных).
3. Применение стандартных исключений Java, таких как FileNotFoundException, IOException, SQLException.

4. Создание собственного кастомного исключения, например, InvalidConfigurationException, которое будет выбрасываться при обнаружении неверного формата конфиг-файла.
5. Организация работы с исключениями средствами языка Java (блоки try-catch-finally, операторы throw и throws).
6. Отчет о результатах: покажите поведение программы при нормальном выполнении и при различных вариантах исключений.

Лабораторная работа №7

Тема: Тестирование методами белого ящика. Метод покрытия операторов. Метод покрытия условий.

Цель: Освоение техник тестирования белым ящиком, изучение методов покрытия операторов и условий для достижения максимальной тестовой уверенности в качестве программного продукта.

Теоеретическая часть:

Метод тестирования белым ящиком (white-box testing) основан на глубоком понимании внутренней структуры и логики программы. Целью является проверка внутреннего функционирования программы, оценка её соответствия спецификациям и требованиям.
Основные понятия
Операторы: инструкции, выполняющие конкретные действия (арифметические операции, присваивания, сравнения и т.п.)
Условия: выражения, определяющие переход программы по различным путям выполнения (условные операторы if, циклы for, while)
Методы покрытия

1. Покрытие операторов (Statement Coverage)Цель: убедиться, что каждая строка кода исполнялась хотя бы раз.Критерий: минимум одно исполнение каждого оператора.

Пример:

```
void exampleMethod() {
   if (a > 0) {              // Оператор условного перехода
      b += c;                // Оператор арифметической операции
   } else {
      d -= e;                // Еще один оператор арифметической операции
   }
}
```

Чтобы достичь покрытия операторов, нам достаточно одного набора значений, при котором условие истинно, и другого, при котором оно ложно.

2. Покрытие условий (Condition Coverage)Цель: проверить каждую ветвь каждого условия.Критерий: минимальное покрытие всех возможных состояний (истинных и ложных) каждого отдельного условия.

Пример:

```
void conditionCoverageExample(boolean flagA, boolean flagB) {
   if ((flagA || flagB) && !flagC) {   // Сложное условие
      executeAction();                 // Подлежит выполнению действие
   }
}
```

Необходимо обеспечить четыре набора тестов, покрывающих каждую комбинацию значений флагов A, B и отрицательного флага C.

Задание:

Задание предназначено для освоения методов покрытия операторов и условий. Необходимо реализовать тесты, обеспечивающие полное покрытие указанных критериев.

Этап 1. Покрытие операторов
Напишите простую программу на языке Java с несколькими операторами и условиями. Затем реализуйте набор тестов, гарантирующий покрытие каждого оператора вашей программы.

Этап 2. Покрытие условий
Дополните вашу программу сложными условиями (логическими выражениями), состоящими из двух и более частей. Создайте набор тестов, удовлетворяющий покрытию условий (каждое условие должно быть выполнено как истиной, так и ложной стороной).

Лабораторная работа №8

Тема: Тестирование методами белого ящика. Метод комбинаторного покрытия условий.

Цель: Освоение метода комбинаторного покрытия условий при тестировании программного обеспечения с применением техник белого ящика для повышения полноты и точности тестирования.

Теоеретическая часть:

Метод комбинаторного покрытия условий относится к технике тестирования белыми методами (White Box Testing), ориентированной на проверку отдельных логических условий в программном коде. Данный

метод гарантирует, что каждое отдельное условие было протестировано всеми возможными значениями ("истина"/"ложь").
Понятие покрытия условий
Покрытие условий (Condition Coverage) — это методика тестирования, направленная на достижение полного охвата всех вариантов значений булевых выражений, участвующих в условиях.
Например, рассмотрим следующий фрагмент кода:

```
if (x > 0 && y != null) {
   processData();
}
```

Для покрытия условий в данном случае потребуется провести тестирование так, чтобы оба условия (x > 0 и y != null) были выполнены дважды: сначала истинно, потом ложно.
Особенности метода комбинаторного покрытия условий
Комбинаторное покрытие условий нацелено на проверку всех сочетаний простых условий (булевых подвыражений) в рамках сложного условия. Это особенно актуально, когда в условии участвуют конъюнкции или дизъюнкции, содержащие несколько подусловий.

Пусть дано сложное условие вида:
cond=(cond1^cond2 ∨ cond3)*cond*=(*cond*1^*cond*2 ∨ *cond*3)

Тогда тестовая стратегия предусматривает построение таблиц или матриц покрытий, в которых будут отражены все комбинации условий
(cond_1, cond_2, cond_3).

Задание:
Рассмотрите простой фрагмент кода на Java с условием, содержащим несколько вложенных булевых выражений. Ваша задача — построить тестовый набор, соответствующий принципу комбинаторного покрытия условий.

Лабораторная работа №9

Тема: Тестирование по черному ящику. Метод классов эквивалентности.

Цель: Освоение технологии тестирования чёрным ящиком, применение метода классов эквивалентности для сокращения количества тестов и повышения их эффективности.

Теоеретическая часть:

Метод тестирования по чёрному ящику (Black Box Testing) основывается на представлении системы как непрозрачного объекта, когда внутреннее

устройство системы неизвестно или неважно. Основное внимание уделяется внешней функциональности системы и поведению при различных входных данных.

Метод классов эквивалентности

Метод классов эквивалентности направлен на сокращение количества тестов путём разделения множества допустимых входных данных на группы (классы), которые ведут себя одинаково с точки зрения тестируемой системы. Такой подход позволяет уменьшить объём тестирования, сохраняя высокую степень покрытия функционала.

Процесс выделения классов эквивалентности

1. Определение границ входных данных. Рассматриваются все возможные границы области определения (например, минимальный/максимальный размер массива, диапазон чисел и т.д.).
2. Выделение классов эквивалентности. Входные данные группируются в классы, характеризующие схожее поведение системы.
3. Выбор представителей классов. Для каждого класса выбираются репрезентативные данные, на которых проводятся тесты.
4. Проверка граничных значений. Дополнительно исследуются граничные случаи (граничные значения, граничные условия), чтобы исключить возможные ошибки вблизи пределов областей определений.

Преимущества метода

- Экономия времени и усилий при подготовке тестов.
- Повышенная уверенность в покрытии функционала.
- Простота и наглядность классификации входных данных.

Задание:

Имеется простая функция для расчёта стоимости заказа такси:

```
double calculateTaxiFare(double distanceInKm) {
   if(distanceInKm <= 0) {
      return 0.0;
   } else if(distanceInKm <= 5) {
      return distanceInKm * 10;
   } else {
      return 50 + (distanceInKm - 5) * 8;
```

```
    }
}
```

Шаги выполнения:

1. Определите область определения аргумента функции (distanceInKm).
2. Выделите классы эквивалентности исходя из описания задачи.
3. Подберите представители классов и граничный случай для тестирования.
4. Подготовьте тестовые сценарии и проверьте их вручную либо с помощью инструментов автоматического тестирования.

Лабораторная работа №10

Тема: Анализ требований к программному обеспечению и составление планов тестирования. Использование систем контроля дефектов программного обеспечения

Цель: Освоение методик анализа требований к программному обеспечению, разработка планов тестирования и получение опыта работы с системами отслеживания дефектов (bug tracking systems).

Теоеретическая часть:

Перед началом тестирования крайне важно детально проанализировать требования к проекту и составить исчерпывающий план тестирования. Эти процессы позволяют значительно снизить риск упущения значимых функциональных возможностей или критичных дефектов.

1. Анализ требований
Требования к программному обеспечению определяют желаемое поведение системы, функциональные и нефункциональные свойства, интерфейсы, взаимодействие с внешними компонентами и многое другое. Важно выделить ключевые моменты:

Функциональные требования (описывают, что система должна делать).
Нефункциональные требования (производительность, масштабируемость, доступность, защищенность и т.д.).
Бизнес-требования (соответствуют ожиданиям пользователей и бизнеса).

Эффективный анализ требований необходим для формирования стратегии тестирования и постановки целей тестирования.

2. Составление планов тестирования
План тестирования — это документ, определяющий стратегию и подход к процессу тестирования. Хороший план включает:

Цели и объем тестирования.
Стратегию тестирования (тип тестов, подходы к выбору тестовых данных).
Ресурсы и сроки выполнения работ.
Оценку рисков и ограничений.
Описания критериев начала и окончания тестирования.
Таким образом, план тестирования становится основой координации всей деятельности команды QA.

3. Использование систем контроля дефектов
Современные проекты используют специальные системы отслеживания дефектов (Jira, Bugzilla, Trello, Redmine и др.) для учета и контроля над обнаруженными проблемами. Эти системы помогают организовать работу с требованиями, тестовыми сценариями и собственно дефектами, предоставляя возможность отслеживать состояние и приоритеты ошибок.

Основные функции систем контроля дефектов включают:

Регистрацию дефектов с присвоением уникальных номеров.
Определение состояния дефектов (открытый, назначенный, исправленный, закрытый).
Присваивание приоритетов (низкий, средний, высокий).
Автоматизацию уведомлений и мониторинга статуса дефектов.
Использование таких систем существенно улучшает контроль качества и координацию командной работы.

Задание:

Этап 1. Анализ требований
Определите требования к простому информационному порталу (пример сайта электронной библиотеки):

- Пользователи могут регистрироваться и входить в систему.
- Администраторы могут добавлять новые книги и редактировать существующие записи.
- Гостевые пользователи видят каталог книг, но не могут изменять контент.

Постарайтесь описать функциональные и нефункциональные требования к данному сервису.

Этап 2. Составление плана тестирования
Разработайте план тестирования для вышеуказанной системы, включающий:

- Объем тестирования (регистрация, вход, администрирование, гостевой режим).
- Список приоритетных задач и критерии успеха.
- Временные рамки и ресурсы.

Этап 3. Работа с системой контроля дефектов
Используя любую популярную систему контроля дефектов (например, Jira), зарегистрируйтесь и создайте карточку дефекта:

- Произвольный уникальный номер.
- Краткое описание дефекта (например, "невозможность войти в систему после неудачной попытки регистрации").
- Степень тяжести (низкая/средняя/высокая).
- Состояние карточки (новый, назначен, закрыт).

Попробуйте зафиксировать несколько типичных ошибок, встречающихся в таких системах, и проследите изменение статусов карточек дефектов.

Лабораторная работа №11

Тема: Разработка модульных тестов для проверки коллекций.

Цель: Освоение технологий модульного тестирования и приобретение навыков создания тестов для проверки функциональности коллекций в Java, таких как списки, карты и множества.

Теоеретическая часть:

Модульное тестирование (unit testing) — это процесс проверки небольших изолированных единиц программного кода, таких как отдельные классы или методы. Оно позволяет убедиться, что небольшие части системы работают должным образом и соответствуют ожидаемому поведению.
Коллекции в Java представляют собой объекты, предназначенные для хранения, сортировки и манипулирования группами объектов. Наиболее распространенные коллекции:

- List (списки): упорядоченная коллекция элементов с возможностью дубликатов.
- Set (множества): неупорядоченная коллекция уникальных элементов.
- Map (карты): хранят пары ключ-значение, где ключи уникальны.

При создании модульных тестов для коллекций необходимо сосредоточиться на тестировании основных операций, таких как добавление, удаление, поиск элемента, сортировку и итерацию по содержимому.
Основные правила модульного тестирования:

1. Независимость тестов: каждый тест должен быть самостоятельным и не зависеть от других тестов.
2. Предсказуемость результатов: результаты тестов должны быть одинаковыми при каждом запуске.
3. Простота понимания: тесты должны быть легко интерпретируемы и лаконичны.
4. Минимальная сложность: тесты должны фокусироваться на проверке отдельной единицы функционала.

Библиотеки для модульного тестирования:
Самая популярная библиотека для тестирования в Java — это JUnit. Она предоставляет удобный API для написания тестов и удобные аннотации для организации тестовых классов и методов.
Пример использования JUnit:

```
@Test
public void testAddElement() {
   List<String> list = new ArrayList<>();
   list.add("element");
   assertEquals(1, list.size()); // Проверяем, что размер списка стал равным 1
}
```

Задание:

Этап 1. Выбор и подготовка среды
Установите среду разработки (IDE) с поддержкой JUnit, например IntelliJ IDEA или Eclipse.

Этап 2. Проектирование тестов
Создайте класс-коллекцию, который будет служить объектом тестирования. Предположим, это будет класс, использующий стандартный интерфейс Collection, например, ArrayList.

Этап 3. Написание тестов
Создайте тест-класс, в котором будут размещены тесты для проверки основных операций:

Добавление элементов в коллекцию.
Удаление элементов из коллекции.
Проверка существования элемента в коллекции.
Проверка размеров коллекции.
Итеративный обход элементов коллекции.
Каждый тест должен начинаться с пустой коллекции и последовательно проверять корректность работы соответствующих методов.

Этап 4. Запуск и анализ результатов

Проверьте работоспособность ваших тестов, запустив их через интегрированную среду разработки. Зафиксируйте любые неисправности и исправляйте их по мере необходимости.

Лабораторная работа №12

Тема: Тестирование интеграции. Написание и выполнение тестов для проверки взаимодействия между модулями

Цель: Освоение приемов тестирования интеграции модулей программного обеспечения, создание и выполнение тестов для проверки согласованного взаимодействия между отдельными частями системы.

Теоеретическая часть:

Интеграционное тестирование (Integration Testing) — это разновидность тестирования, предназначенная для проверки совместной работы отдельных компонентов или модулей системы. Основная задача интеграционного тестирования — убедиться, что модули взаимодействуют друг с другом корректно и соответствуют проектной архитектуре.

Основные концепции интеграционного тестирования:

1. Модуль: Отдельный компонент системы, имеющий строго определенную функцию.
2. Интерфейс: Способ обмена данными и сообщениями между модулями.
3. Протокол взаимодействия: Правила коммуникации между модулями, определяющие порядок передачи данных и управляющих сигналов.

Типичная схема интеграционного тестирования:

- Big Bang Integration Testing: Все модули объединяются одновременно и проверяется их совместная работа.
- Bottom-up Integration Testing: Сначала интегрируют низкоуровневые модули, постепенно поднимаясь вверх по иерархии.
- Top-down Integration Testing: Начинают с верхних уровней системы, заменяя нижележащие модули заглушками (stubs/mocks).

Процесс интеграционного тестирования:

1. Проектирование тестов: определяются сценарии тестирования, строится карта зависимостей между модулями.
2. Настройка окружения: создаются необходимые окружающие условия (базы данных, серверы, внешние сервисы).
3. Написание тестов: создается набор тестов, проверяющих обмен данными и взаимодействия между модулями.
4. Выполнение тестов: запуск тестов и фиксация результатов.

5. Анализ результатов: оценка работоспособности системы и устранение выявленных дефектов.

Используемые инструменты:

- JUnit/Maven/Selenium: инструменты для автоматизации интеграционного тестирования в среде Java.
- Postman: средство для тестирования RESTful API.
- Mockito: библиотека для создания mock-объектов, имитирующих поведение внешних сервисов.

Задание:

Представьте себе систему, состоящую из трех модулей:

- Модуль аутентификации (AuthModule).
- Модуль обработки заказов (OrderProcessingModule).
- Модуль ведения статистики продаж (SalesStatisticsModule).

Эти модули взаимодействуют между собой через API-интерфейсы. Ваша задача — написать интеграционные тесты, проверяющие правильность их взаимодействия.

Этап 2. Проектирование тестов

Определите тестовые сценарии для проверки взаимодействия модулей. Например:

Аутентификация пользователя и последующая передача данных в модуль обработки заказов.

Отправка статистических данных о заказах в модуль ведения статистики.

Этап 3. Настройка окружения

Создайте базовые классы и методы для модулей. Используйте заглушки (mocks/stubs) там, где это необходимо, чтобы смоделировать поведение внешних сервисов.

Этап 4. Написание тестов

Используя библиотеку JUnit и Mockito, напишите тесты для проверки интеграционной целостности модулей. Вот шаблон для одного из тестов:

```
@RunWith(MockitoJUnitRunner.class)
public class AuthAndOrderProcessingITest {

    @Mock
    OrderProcessingModule orderProcessingModule;

    @InjectMocks
    AuthModule authModule;

    @BeforeEach
    public void setUp() {
        MockitoAnnotations.initMocks(this);
    }
    @Test
    public void testUserAuthenticationAndOrderCreation() {
        User user = new User("test_user", "password");
        Order order = new Order(1L, "Product X", 100);

when(orderProcessingModule.createOrder(any(Order.class))).thenReturn(true);

        boolean isAuthenticated = authModule.authenticate(user);
        assertTrue(isAuthenticated);

        boolean isOrderCreated = authModule.processOrder(user, order);
        verify(orderProcessingModule, times(1)).createOrder(order);
```

```
        assertTrue(isOrderCreated);
    }
}
```

Этап 5. Выполнение тестов

Запустите созданные тесты и проанализируйте результаты. Устраняйте выявленные ошибки и повторно запускайте тесты до тех пор, пока все тесты не пройдут успешно.

Лабораторная работа №13

Тема: Тестирование RESTful API

Цель: Освоение приёмов тестирования RESTful API, формирование навыков создания и выполнения тестов для проверки работоспособности и корректности работы HTTP API.

Теоеретическая часть:

REST (Representational State Transfer) — это архитектурный стиль для создания распределённых систем, основанный на взаимодействии через HTTP-протокол. RESTful API (Application Programming Interface) предоставляет интерфейс для взаимодействия клиентов с серверами, позволяя получать, отправлять и управлять данными через стандартные HTTP-методы (GET, POST, PUT, PATCH, DELETE и др.).
Особенности RESTful API:

- Интерфейс предоставляется через URL (Resource URIs).
- Данные возвращаются в удобочитаемом формате (JSON, XML).
- Общение осуществляется в стиле stateless (без сохранения состояния).

2. Что значит тестирование RESTful API?
Тестирование RESTful API — это комплекс мероприятий, направленных на проверку корректности работы API. Включает в себя:

- Функциональное тестирование: проверка соответствия API своей спецификации.
- Тестирование производительности: измерение скорости отклика и пропускной способности API.
- Тестирование безопасности: проверка защиты API от несанкционированного доступа и угроз.
- Загрузочное тестирование: симуляция большого объема трафика для выявления слабых мест API.

3. Методы и инструменты тестирования RESTful API

Основными методами тестирования API являются:

- Manual Testing: ручное тестирование с использованием браузеров или специальных инструментов (например, Postman, Insomnia).
- Automated Testing: автоматизация тестов с помощью библиотек и фреймворков (например, Rest Assured, SuperTest, SoapUI).

Распространённые инструменты для тестирования RESTful API:

- Postman: мощный инструмент для отправки запросов и визуализации ответов.
- SoapUI: специализированный инструмент для тестирования SOAP и REST API.
- JMeter: популярный open-source инструмент для нагрузочного тестирования.
- Rest Assured: библиотека для автоматизации тестирования REST API на Java.

Задание:

Этап 1. Подготовка

Выберите одну из готовых реализаций RESTful API (например, демо API сервиса публикации статей или базу данных фильмов). Изучите документацию API и схему его работы.

Этап 2. Проектирование тестов
Спроектируйте тесты для проверки основных методов API (GET, POST, PUT, DELETE). Ниже приведены шаблоны тестов:

1. Тестирование метода GET:
   - Позитивный тест: получение конкретного ресурса (статья, фильм).
   - Негативный тест: попытка получить несуществующий ресурс.
2. Тестирование метода POST:
   - Позитивный тест: создание нового ресурса (статьи, фильма).
   - Негативный тест: попытка создать ресурс с неполными или некорректными данными.
3. Тестирование метода PUT/PATCH:
   - Позитивный тест: обновление существующего ресурса.
   - Негативный тест: попытка обновить несуществующий ресурс.
4. Тестирование метода DELETE:
   - Позитивный тест: удаление существующего ресурса.
   - Негативный тест: попытка удалить несуществующий ресурс.

Этап 3. Реализация тестов
Реализуйте тесты с использованием выбранного инструмента (например, Postman или Rest Assured). Вот пример реализации теста на Java с использованием Rest Assured:

```
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;

public class ApiTests {
    @Test
    public void testGetArticle() {
        given().
            contentType("application/json").
        when().
            get("/articles/{id}", 1).
        then().
            statusCode(200).
            body("title", equalTo("Title of Article 1"));
    }

    @Test
    public void testCreateArticle() {
        Map<String, Object> article = new HashMap<>();
        article.put("title", "New Title");
        article.put("content", "This is the content of my new article.");

        given().
            contentType("application/json").
            body(article).
        when().
            post("/articles").
        then().
            statusCode(201).
            body("title", equalTo("New Title"));
    }
}
```

Этап 4. Анализ результатов

Выполните тесты и зафиксируйте результаты. Обратите внимание на код ответа (200, 201, 404 и т.д.) и содержание ответа (правильность формата данных, соответствие спецификации).

Лабораторная работа №14

Тема: Разработка документации на программное обеспечение в соответствии со стандартами. Ведение журнала изменений и фиксация обновления программных модулей.

Цель: Освоение методик и стандартов разработки технической документации на программное обеспечение, ведение журнала изменений и правил фиксации обновления программных модулей.

Теоеретическая часть:

Документация на программное обеспечение играет ключевую роль в процессе разработки, эксплуатации и поддержки программного продукта. Четко сформулированная техническая документация облегчает понимание программы, ускоряет процесс внесения изменений и поддерживает преемственность проекта.
Стандарт ISO/IEC 26514
Одним из международных стандартов, регламентирующих разработку технической документации на программное обеспечение, является ISO/IEC 26514. Согласно этому стандарту, документация делится на следующие уровни:

- User Documentation (документация пользователя): руководство пользователя, справочная информация, учебные материалы.
- System Documentation (системная документация): техническое описание системы, схемы архитектуры, описание API.
- Developer Documentation (разработческая документация): комментарии к коду, спецификации модулей, руководства по установке и развертыванию.

Требования к содержанию технической документации:

- Ясность изложения: текст должен быть простым и понятным, свободным от двусмысленности.
- Последовательность: информация должна следовать логическому порядку, соответствующему этапам жизненного цикла разработки.
- Полезность: документ должен помогать пользователю решать реальные задачи, а разработчику — понимать структуру и назначение компонентов.

Журнал изменений (Change Log)
Журнал изменений — это официальный документ, фиксирующий историю версий программного обеспечения, в которой указаны вносимые изменения, исправления ошибок и дополнения функциональности. Основной задачей журнала является поддержание прозрачности развития продукта и помощь команде в сопровождении программы.
Правила ведения Change Log:

- Каждая версия продукта должна сопровождаться записью о внесении изменений.
- Изменения должны быть описаны ясно и понятно.
- Используется общепринятая терминология и обозначения.
- Дата и ответственный за внесение изменений должны быть указаны.

Фиксация обновления программных модулей

Обновление программных модулей производится через специальную процедуру, включающую:

- Контроль версий: использование Git, SVN или аналогичных систем для отслеживания истории изменений.
- Сборка и деплоймент: автоматизированные скрипты сборки (CI/CD), позволяющие оперативно доставлять новую версию пользователям.
- Версионирование: обозначение релизов с номерами версий (major.minor.patch).

Задание:

Этап 1. Подготовка

Ознакомьтесь с примерами технических документов и журналов изменений. Выберите подходящий образец для дальнейшей работы.

Этап 2. Проектирование

Создайте технический документ для небольшого программного продукта (например, калькулятора или мобильного приложения). Документ должен соответствовать основным требованиям стандарта ISO/IEC 26514.

Затем начните вести журнал изменений, начиная с первой версии продукта. Отобразите в журнале все последующие изменения (исправления ошибок, добавление новых функций и т.д.).

Этап 3. Реализация

Оформите созданный документ и журнал изменений в формате PDF или Word-документа.

Основная литература:

1. Алдан, А. Введение в генерацию программного кода / А. Алдан. - 2-е изд., испр. - М. : Национальный Открытый Университет «ИНТУИТ», 2026. - 189 с. : схем. - Библиогр. в кн. ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=428825

2. Соколов В.П. Кодирование в системах защиты информации [Электронный ресурс]:
учебное пособие/ Соколов В.П., Тарасова Н.П.— Электрон. текстовые данные.— М.: Московский технический университет связи и информатики, 2026.— 94 c.— Режим доступа: http://www.iprbookshop.ru/61485.html.— ЭБС «IPRbooks»

3. Антимиров, В. М. Проектирование аппаратуры систем автоматического управления. В 2 ч. Ч. 1 : учебное пособие для СПО / В. М. Антимиров. — 2-е изд. — Саратов, Екатеринбург : Профобразование, Уральский федеральный университет, 2022. — 92 c. — ISBN 978-5-4488-04014, 978-5-7996-2834-5. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/87852.html. — Режим доступа: для авторизир.
пользователей

Дополнительная литература:

1. Федорова Г.И. Разработка, внедрение и адаптация программного обеспечения отраслевой направленности. Учебное пособие. Изд.: КУРС, Инфра-М. Среднее профессиональное образование. 2025 г. 336 стр.