ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ЦИФРОВЫХ ОБРАЗОВАТЕЛЬНЫХ ТЕХНОЛОГИЙ

ПИ (филиал) ДГТУ в г. Таганроге
ЦМК «Прикладная информатика»

Практикум
по дисциплине

«Тестирование и эксплуатация информационных систем»

Автор
Погорелов А.А.

Ростов-на-Дону, 2026

Аннотация

«Методические рекомендации по выполнению практических работ по дисциплине «Тестирование и эксплуатация информационных систем» предназначены для студентов очной формы обучения по специальности 09.02.12 «Техническая эксплуатация и сопровождение информационных систем».

Автор

Преподаватель ЦМК «Прикладная информатика»
Погорелов А.А.

Оглавление

Введение

Настоящие методические рекомендации разработаны для обучающихся по специальности 09.02.12 Техническая эксплуатация и сопровождение информационных систем, осваивающих профессиональный модуль ПМ.01 «Тестирование и эксплуатация информационных систем».

Целью практических работ по данному модулю является формирование у обучающихся устойчивых практических умений и навыков в области обеспечения качества, тестирования и последующей эксплуатации информационных систем. В ходе выполнения работ обучающиеся закрепляют теоретические знания, полученные на лекциях, и осваивают ключевые технологии, используемые в современных процессах разработки и сопровождения программного обеспечения.

Обучающийся должен знать:

Актуальный профессиональный и социальный контекст, в котором приходится работать и жить; основные источники информации и ресурсы для решения задач и проблем в профессиональном и социальном контексте; алгоритмы выполнения работ в профессиональной и смежных областях; методы работы в профессиональной и смежных сферах; структуру плана для решения задач; порядок оценки результатов решения задач профессиональной деятельности. Номенклатуру информационных источников, применяемых в профессиональной деятельности; приемы структурирования информации; формат оформления результатов поиска информации, современные средства и устройства информатизации; порядок их применения и программное обеспечение в профессиональной деятельности в том числе с использованием цифровых средств.

Обучающийся должен уметь:

Определять задачи для поиска информации; определять необходимые источники информации; планировать процесс поиска; структурировать получаемую информацию; выделять

наиболее значимое в перечне информации; оценивать практическую значимость результатов поиска; оформлять результаты поиска, применять средства информационных технологий для решения профессиональных задач; использовать современное программное обеспечение; использовать различные цифровые средства для решения профессиональных задач. Распознавать задачу и/или проблему в профессиональном и/или социальном контексте; анализировать задачу и/или проблему и выделять её составные части; определять этапы решения задачи; выявлять и эффективно искать информацию, необходимую для решения задачи и/или проблемы; составлять план действия; определять необходимые ресурсы; владеть актуальными методами работы в профессиональной и смежных сферах; реализовывать составленный план; оценивать результат и последствия своих действий (самостоятельно или с помощью наставника).

Общие положения. Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Перед выполнением обучающийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет). Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию. Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию.

Обучающийся, пропустивший практическое занятие по уважительной, либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

Форма отчета: титульный лист; введение (цель и задачи); выполнение (описание выполнения каждого задания с подтверждающими материалами — скриншотами, листингами, таблицами); заключение (вывод о проделанной работе, какие навыки приобретены).

Практическое занятие №1 «Анализ и оценка качества информационной системы с использованием метрик качества»

Цель работы: Закрепить знания о подходах к оценке качества ПО и стандартах в этой области. Научиться выбирать и рассчитывать метрики качества для оценки характеристик информационной системы. Приобрести практические навыки использования инструментальных средств для сбора и анализа метрик качества кода и производительности.

Теоретическая часть

Качество программного обеспечения (ПО) — это степень, в которой система удовлетворяет заявленные и подразумеваемые потребности заинтересованных сторон. Для объективной оценки качества используются метрики — количественные показатели, характеризующие различные аспекты ПО.

Метрики качества можно разделить на несколько категорий:

– **Метрики кода**: сложность (цикломатическая сложность Маккейба), количество строк кода (LOC), глубина наследования, связность модулей.

– **Метрики надежности**: плотность дефектов (количество ошибок на тысячу строк кода), среднее время наработки на отказ (MTBF), полнота тестового покрытия.

– **Метрики производительности**: время отклика системы, пропускная способность, загрузка процессора и памяти.

– **Метрики стандартов качества**: соответствие стандартам ISO/IEC 25000 (характеристики: функциональность, надежность, удобство использования, эффективность, сопровождаемость, переносимость).

Инструменты для сбора метрик: статические анализаторы (SonarQube, ESLint), профилировщики (JProfiler, VisualVM), системы сбора логов (ELK Stack).

Рабочее задание

1.Анализ требований к качеству. На основе предложенного технического задания на разработку веб-приложения (например, интернет-магазин или система учета) составить перечень критически важных характеристик качества согласно ISO/IEC 25000. Определить пороговые значения для выбранных метрик (например, время отклика не более 2 секунд, тестовое покрытие не менее 70 %).

2.Расчет метрик качества кода. Получить от преподавателя исходный код учебного проекта. С помощью инструмента статического анализа (например, SonarQube или его локальный анализатор) рассчитать основные метрики кода: количество строк, цикломатическую сложность, дублирование кода, наличие критических ошибок. Составить отчет с выявленными проблемами.

3.Оценка производительности. Провести нагрузочное тестирование небольшого веб-приложения с помощью Apache JMeter или онлайн-сервиса. Замерить метрики производительности: время отклика при разном количестве одновременных пользователей, пропускную способность. Сравнить полученные значения с пороговыми, определенными в задании 1.

Практическое занятие № 2 «Использование статического анализа кода для выявления дефектов»

Цель работы: Изучить методологию статического анализа кода и его роль в обеспечении качества. Освоить приемы работы с современными инструментами статического анализа. Научиться интерпретировать отчеты статического анализа и классифицировать найденные дефекты по степени критичности. Применять ПК 1.5 для исправления дефектов и несоответствий в коде.

Теоретическая часть

Статический анализ кода — это процесс выявления дефектов и уязвимостей в исходном коде программы без её фактического выполнения. Анализ проводится путем сканирования кода специальными инструментами (линтерами), которые сравнивают написанный код с набором правил (best practices, стандарты кодирования, поиск потенциально опасных конструкций).

Основные типы обнаруживаемых проблем:

- Синтаксические ошибки и опечатки.
- Нарушение стандартов кодирования (Code Conventions).
- Потенциальные уязвимости (SQL-инъекции, XSS-атаки).
- "Запахи кода" (Code Smells): дублирование кода, слишком длинные методы, пустые catch-блоки.
- Проблемы с производительностью.

Примеры инструментов: SonarQube, ESLint (для JavaScript), Pylint (для Python), SpotBugs (для Java), PVS-Studio.

Рабочее задание

1.Запуск статического анализатора. Развернуть на локальной машине инструмент статического анализа (например, запустить анализ в IDE с помощью плагина SonarLint или Pylint). Выполнить анализ предоставленного преподавателем фрагмента кода (содержащего заведомые ошибки).

2.Классификация дефектов. Составить таблицу выявленных замечаний, классифицируя их по категориям: критическая ошибка, потенциальная уязвимость, запах кода, нарушение стандарта оформления.

3.Исправление ошибок. Последовательно исправить все выявленные дефекты в коде, фиксируя изменения. Повторно запустить анализ для подтверждения исправления. Отразить в отчете, какие изменения были внесены для устранения каждой категории ошибок.

Практическое занятие № 3 «Разработка стратегии отладки и исправление ошибок в программном обеспечении»

Цель работы: Изучить основные стратегии и инструменты отладки программного обеспечения. Приобрести практические навыки локализации и устранения различных типов ошибок (синтаксических, логических, ошибок времени выполнения). Научиться применять отладчик IDE для анализа состояния программы в процессе выполнения. Исправить дефекты и несоответствия в коде (ПК 1.5).

Теоретическая часть

Отладка (debugging) — это процесс поиска, локализации и устранения причины ошибки, обнаруженной в ходе тестирования или эксплуатации ПО. Стратегия отладки определяет подход к поиску причины дефекта.

Основные стратегии:

- **Метод индукции**: анализ данных об ошибке (симптомов) и изучение кода для поиска причин;
- **Метод дедукции**: выдвижение гипотез о возможных причинах и их проверка;
- **Обратное прослеживание**: движение по коду от места проявления ошибки назад к предполагаемому источнику;
- **Тестирование "на грабли"**: сужение области поиска путем исключения корректно работающих частей кода.

Инструменты отладки: встроенные отладчики IDE (точки останова, пошаговое выполнение, просмотр переменных), логирование, профилировщики памяти, снифферы трафика (Wireshark, Fiddler).

Рабочее задание

1.Локализация ошибки с помощью отладчика. Получить исполняемый код, который содержит логическую ошибку (например, неверно считается сумма или неправильно работает сортировка). Используя точки останова и пошаговое выполнение в IDE, локализовать строку кода, в которой возникает ошибка. Задокументировать процесс поиска (какие гипотезы выдвигались, какие переменные проверялись).

2.Анализ ошибки по логам. Получить файл с логами работы программы, которая "упала" с исключением. На основе анализа стека вызовов (stack trace) определить класс, метод и строку кода, вызвавшую ошибку. Описать последовательность действий, приведшую к ошибке.

3.Исправление ошибки. Модифицировать код для устранения найденной причины. Провести повторное тестирование (модульное или ручное), чтобы убедиться, что

ошибка исправлена и не появились новые дефекты (регрессия).

Практическое занятие № 4 «Анализ требований к программному обеспечению и составление планов тестирования. Использование систем контроля дефектов программного обеспечения»

Цель работы: Освоить методику анализа требований к ПО на предмет полноты и тестируемости. Научиться составлять план тестирования для заданного проекта. Изучить структуру и функциональные возможности систем контроля дефектов. Применить ПК 1.4 для документирования тестовых случаев в соответствии с требованиями организации.

Теоретическая часть

Тестирование начинается с анализа требований. Нечеткие, противоречивые или неполные требования делают тестирование невозможным.

- **Анализ требований:** выявление некорректных формулировок, проверка требований на соответствие критериям (атомарность, однозначность, законченность, непротиворечивость, прослеживаемость).
- **План тестирования (Test Plan):** документ, описывающий объем работ, подходы, ресурсы и график тестирования. Включает: объект тестирования, стратегию, критерии начала и окончания тестирования, распределение ролей, риски.
- **Системы контроля дефектов (Bug Tracking Systems):** инструменты для учета ошибок (багов). Позволяют зарегистрировать дефект, назначить ответственного, отслеживать статус (New, Assigned, Fixed, Verified, Closed), анализировать статистику. Примеры: Jira, Redmine, YouTrack, Bugzilla.

Рабочее задание

1.Анализ требований. Получить текст Технического задания на разработку модуля (например, "Форма регистрации пользователя"). Выявить в требованиях не менее трех недостатков (неточности, противоречия, отсутствие информации о граничных значениях). Сформулировать уточняющие вопросы к заказчику.

2.Составление плана тестирования. На основе уточненных требований разработать фрагмент плана тестирования, включающий: объект тестирования, виды тестирования (функциональное, UI, безопасности), критерии начала и окончания тестирования.

3.Работа с баг-трекером. Изучить интерфейс предложенной системы (например, учебная версия Jira или Redmine). Зарегистрировать в системе три учебных дефекта, корректно заполнив все поля: краткое описание, подробное описание (шаги воспроизведения, фактический и ожидаемый результат), окружение, приоритет, серьезность.

Практическое занятие № 5 «Разработка тестовых сценариев»

Цель работы: Изучить основные техники тест-дизайна (классы эквивалентности, анализ граничных значений). Приобрести практические навыки разработки тестовых сценариев на основе требований. Научиться применять различные техники проектирования тестов (тест-дизайна) (ПК 1.4). Оформлять тестовые сценарии в соответствии с требованиями организации.

Теоретическая часть

Тестовый сценарий (Test Case) — это артефакт, описывающий совокупность шагов, конкретных условий и

параметров, необходимых для проверки реализации тестируемой функции или её части.

Техники тест-дизайна (разработки тестов):

- **Классы эквивалентности (Equivalence Partitioning):** разбиение входных данных на группы, обработка которых предполагается одинаковой. Достаточно проверить одно значение из каждого класса.
- **Анализ граничных значений (Boundary Value Analysis):** проверка значений на границах классов эквивалентности (максимум, минимум, сразу за границей). Наиболее вероятное место скопления ошибок.
- **Таблица принятия решений (Decision Table):** для проверки сложной логики с несколькими условиями.
- **Тестирование состояний и переходов (State Transition Testing):** для систем, поведение которых зависит от предыстории (состояния).

Структура тест-кейса: Уникальный ID, Название, Предусловия, Шаги, Ожидаемый результат, Постусловия, Статус (PASS/FAIL).

Рабочее задание

1.Применение техники классов эквивалентности. Для функции "Ввод возраста" (допустимые значения от 18 до 65 лет) выделить классы эквивалентности. Разработать тестовые сценарии, покрывающие каждый класс.

2.Применение анализа граничных значений. Для той же функции "Ввод возраста" определить граничные значения. Разработать дополнительные тестовые сценарии, проверяющие значения на границах и за границами допустимого диапазона.

3.Составление полного набора тест-кейсов. На основе требований к форме регистрации (поля: Имя (текст),

Email (формат), Пароль (мин. 8 символов), Чекбокс "Согласен с правилами") разработать не менее 7-10 тестовых сценариев. Оформить их в табличной форме (в Excel, Google Sheets или системе управления тестированием), комбинируя техники из заданий 1 и 2.

Практическое занятие № 6 «Поиск и документирование дефектов, используя системы контроля дефектов программного обеспечения»

Цель работы: Освоить методологию поиска дефектов в готовом программном продукте. Научиться четко и структурированно описывать дефекты, заполняя все необходимые поля в баг-трекинговой системе. Применять ПК 1.4 для документирования тестовых случаев и найденных дефектов.

Теоретическая часть

Дефект (баг) — это несоответствие фактического результата работы программы ожидаемому результату, описанному в требованиях. Качество описания дефекта напрямую влияет на скорость его исправления. Правила составления хорошего отчета об ошибке (Bug Report):

- **Краткое описание (Summary):** емко и точно отражает суть проблемы. Формула: "Что? Где? При каких условиях?" (например, "Кнопка 'Сохранить' неактивна в профиле пользователя после загрузки аватара").
- **Детальное описание (Description):** Шаги воспроизведения (Steps to Reproduce), Фактический результат (Actual Result), Ожидаемый результат (Expected Result).
- **Приложения (Attachments):** скриншоты, видео, логи, дампы.
- **Окружение (Environment):** ОС, браузер, версия ПО, разрешение экрана.

- **Атрибуты:** Серьезность (Severity) — влияние на систему (S1 Блокирующая, S2 Критическая, S3 Значительная...), Приоритет (Priority) — срочность исправления (P1 Высокий, P2 Средний, P3 Низкий).

Рабочее задание

1.Исследовательское тестирование. Получить доступ к тестовой версии веб-приложения (или его демо-стенду). Провести исследовательское тестирование одного из модулей (например, "Личный кабинет" или "Корзина") с целью поиска дефектов. Найти не менее 2-3 различных ошибок.

2.Оформление дефекта (баг-репорта). Для каждого найденного дефекта создать отчет в системе контроля дефектов (или в предоставленном шаблоне). Корректно заполнить поля: краткое описание, шаги воспроизведения, фактический и ожидаемый результат, приложить скриншот.

3.Классификация дефектов. Определить и обосновать для каждого найденного дефекта уровень серьезности (Severity) и приоритет (Priority).

Практическое занятие № 7 «Тестирование методами белого ящика»

Цель работы: Изучить критерии покрытия кода (операторов, ветвлений, условий). Научиться анализировать код и разрабатывать тестовые сценарии, обеспечивающие заданный уровень покрытия. Применять ПК 1.4 для разработки тестов, нацеленных на проверку внутренней логики. Исправлять дефекты и несоответствия в коде (ПК 1.5).

Теоретическая часть

Тестирование белого ящика (White Box Testing), или структурное тестирование, основано на анализе внутренней

структуры программы (кода). Тесты разрабатываются на основе знания реализации.

Основные критерии покрытия кода:

- **Покрытие операторов (Statement Coverage):** каждая ли исполняемая строка кода была выполнена хотя бы один раз?
- **Покрытие решений/ветвлений (Decision/Branch Coverage):** каждое ли логическое условие (if, case) принимало значения true и false хотя бы один раз?
- **Покрытие условий (Condition Coverage):** каждая ли часть сложного условия (например, A && B) принимала оба логических значения?
- **Покрытие путей (Path Coverage):** все ли возможные маршруты выполнения программы были пройдены?

Методы белого ящика используются для модульного и интеграционного тестирования, проверки безопасности кода.

Рабочее задание

1.Анализ графа потока управления. Получить от преподавателя фрагмент кода, содержащий условные операторы (if-else) и циклы. Построить граф потока управления для этого фрагмента.

2.Разработка тестов для покрытия ветвлений. На основе построенного графа разработать набор тестов (входных данных), который обеспечивает 100 % покрытие всех ветвлений (каждая ветка if-else должна быть выполнена).

3.Оценка покрытия. Запустить код с разработанными тестами в среде разработки, используя инструмент замера покрытия (например, Coverage в PyCharm или Eclipse). Предоставить скриншот, подтверждающий достигнутый

процент покрытия. Если покрытие не 100 %, добавить недостающие тесты.

Практическое занятие № 8 «тестирование по черному ящику»

Цель работы: Закрепить понимание методологии тестирования «черного ящика». Научиться применять различные техники тест-дизайна черного ящика для проверки функциональности без доступа к коду. Разрабатывать тестовые сценарии на основе пользовательских историй. Применять ПК 1.4 для проектирования тестов на основе требований.

Теоретическая часть

Тестирование черного ящика (Black Box Testing), или функциональное тестирование, рассматривает программу как "черный ящик". Внутренняя структура неизвестна и неважна. Тесты строятся на основе внешней документации: требований, спецификаций, пользовательских сценариев. Цель — проверка соответствия поведения системы заявленным требованиям.

Техники черного ящика (повторение и углубление):

- **Классы эквивалентности:** позволяют сократить количество тестов;
- **Анализ граничных значений:** позволяют найти ошибки на краях диапазонов;
- **Диаграммы состояний:** для проверки переходов;
- **Таблицы решений:** для сложной бизнес-логики;
- **Тестирование на основе сценариев использования (Use Cases):** проверка типичных путей пользователя (счастливый путь) и исключительных ситуаций.

Рабочее задание

1.Сценарное тестирование (Use Case). Для интернет-магазина дан сценарий "Покупка товара с использованием промокода". Разработать "счастливый путь" (happy path) — набор шагов, приводящий к успешной покупке. Затем разработать не менее 3 негативных сценариев (например, промокод просрочен, товара нет в наличии, отказ от покупки на этапе оплаты).

2.Таблица принятия решений. Составить таблицу решений для модуля "Скидка". Правила: если сумма покупки > 5000 И покупатель постоянный (флаг в БД), то скидка 10 %. Если сумма > 5000 И покупатель НЕ постоянный, то скидка 5 %. В остальных случаях скидка 0%. Заполнить таблицу и разработать тесты для проверки всех комбинаций условий.

3.Попарное тестирование (Pairwise). Дана форма фильтрации автомобилей: Марка (Toyota, BMW, Lada), Цвет (Красный, Белый, Черный), Коробка передач (МКПП, АКПП). Используя технику попарного тестирования (или онлайн-инструменты для pairwise), составить оптимальный набор тестов для проверки работы фильтра.

Практическое занятие № 9 «Разработка модульных тестов»

Цель работы: Изучить структуру и назначение модульных тестов. Освоить работу с фреймворком для модульного тестирования (pytest или JUnit). Научиться писать модульные тесты для отдельных функций и классов. Применять ПК 1.4 для разработки скриптов и программных модулей для автоматизации тестирования.

Теоретическая часть

Модульное тестирование (Unit Testing) — процесс проверки отдельных наименьших тестируемых компонентов

программы (модулей, функций, классов) на корректность их работы. Это первый и самый важный уровень защиты от дефектов.

Фреймворки для модульного тестирования (xUnit):

- Java: JUnit, TestNG.
- Python: unittest, pytest.
- JavaScript: Jest, Mocha.
- C#: NUnit, MSTest.

Структура модульного теста (часто по шаблону AAA):

- **Arrange (Подготовка):** инициализация объектов, задание входных данных.
- **Act (Действие):** вызов тестируемого метода.
- **Assert (Проверка):** сравнение полученного результата с ожидаемым.

Принципы хороших юнит-тестов: быстрые, изолированные (не зависят друг от друга), повторяемые, самопроверяемые (проходят или падают автоматически).

Рабочее задание

1.Настройка окружения. Установить и настроить фреймворк для модульного тестирования (pytest для Python или JUnit для Java) в среде разработки.

2.Тестирование математической функции. Для предоставленной функции (например, вычисление факториала, проверка простого числа) написать набор модульных тестов, проверяющих:

– Работу с граничными значениями (0, 1).

– Работу с типичными значениями (5, 10).

– Реакцию на невалидные входные данные (отрицательное число, строка) — ожидается исключение.

3.Тестирование класса. Для предоставленного простого класса (например, "Калькулятор", "Банковский счет") написать модульные тесты, покрывающие все его публичные методы. Убедиться, что все тесты проходят успешно. Создать ситуацию падения теста и исправить код класса (ПК 1.5).

Практическое занятие № 10 «Тестирование производительности»

Цель работы: Изучить виды тестирования производительности и цели их проведения. Освоить базовые навыки работы с инструментом нагрузочного тестирования (Apache JMeter). Научиться интерпретировать результаты нагрузочного тестирования и выявлять узкие места. Применять ПК 1.6 для анализа производительности развернутой системы.

Теоретическая часть

Тестирование производительности (Performance Testing) — это вид нефункционального тестирования, проводимый для определения скорости, отзывчивости и стабильности системы под определенной рабочей нагрузкой.

Основные подвиды:

- **Нагрузочное тестирование (Load Testing):** проверка поведения системы при ожидаемых (прогнозируемых) нагрузках.
- **Стресс-тестирование (Stress Testing):** проверка поведения системы при нагрузках, превышающих нормальные, вплоть до "точки перелома" (break point).
- **Тестирование стабильности (Soak/Endurance Testing):** проверка работы системы под длительной средней нагрузкой для выявления утечек памяти.

- **Объемное тестирование (Volume Testing):** проверка работы с большими объемами данных.

Ключевые метрики: время отклика (response time), пропускная способность (throughput — запросов в секунду), количество одновременных пользователей, использование ресурсов (CPU, RAM, Network).

Рабочее задание

1.Создание профиля нагрузки. Для тестового веб-приложения (например, локально развернутый сайт) разработать сценарий нагрузки в Apache JMeter, включающий обращение к главной странице и странице каталога. Настроить параметры: количество потоков (виртуальных пользователей) = 10, время нарастания (ramp-up) = 5 секунд.

2.Запуск теста и сбор метрик. Провести нагрузочный тест. Используя встроенные слушатели (Listeners) в JMeter, собрать данные о времени отклика и проценте ошибок.

3.Анализ результатов. Проанализировать полученные отчеты (Aggregate Report, Graph Results). Построить график зависимости времени отклика от номера запроса. Сделать вывод о соответствии производительности системы заданным критериям. Предложить возможные пути оптимизации.

Практическое занятие № 11 «Тестирование документации и требований»

Цель работы: Освоить методику тестирования технической и пользовательской документации. Научиться выявлять ошибки, неточности и несоответствия в документации. Применять ПК 1.5 для исправления несоответствий в документации.

Теоретическая часть

Тестирование документации так же важно, как и тестирование кода, поскольку документация — это интерфейс между заказчиком, аналитиками, разработчиками, тестировщиками и конечными пользователями.

Что проверяется в документации:

- **Требования (SRS — Software Requirements Specification):** Полнота, однозначность, непротиворечивость, тестируемость, прослеживаемость.
- **Руководство пользователя (User Manual):** Соответствие текущей версии интерфейса, понятность инструкций, отсутствие орфографических и грамматических ошибок.
- **Помощь (Help):** Работа гиперссылок, поиск, соответствие контексту.
- **Маркетинговые материалы:** Соответствие реальному функционалу (чтобы не вводить в заблуждение). Дефекты документации могут приводить к неправильной реализации и неверному использованию системы.

Рабочее задание

1.Рецензирование требований. Получить фрагмент технического задания (ТЗ) на разработку функции. Провести анализ требований на предмет неоднозначности и неполноты. Составить список замечаний и предложений по улучшению формулировок.

2.Проверка руководства пользователя. Получить текст руководства пользователя для небольшой программы. Сверить инструкции из руководства с реальным интерфейсом программы (если программа доступна) или с ее макетом. Найти не менее 3 несоответствий или устаревших инструкций.

3.Коррекция документации. На основе выявленных замечаний из заданий 1 и 2 подготовить исправленный вариант соответствующих разделов документации, оформив правки в виде отчета.

Практическое занятие № 12 «Тестирование юзабилити»

Цель работы: Изучить критерии и принципы оценки удобства использования программного продукта. Освоить методы проведения юзабилити-тестирования и сбора обратной связи. Научиться выявлять проблемы интерфейса и формулировать рекомендации по их устранению.

Теоретическая часть

Юзабилити-тестирование (Usability Testing) — метод оценки качества пользовательского интерфейса, направленный на выявление проблем, с которыми сталкиваются пользователи при взаимодействии с продуктом.

Основные принципы юзабилити (по Якобу Нильсену):

- **Узнаваемость состояния системы:** пользователь всегда должен понимать, что происходит.
- **Соответствие реальному миру:** язык интерфейса должен быть понятен пользователю.
- **Свобода и контроль:** пользователь может легко отменить действие или выйти из нежелательного состояния.
- **Согласованность и стандарты:** элементы интерфейса должны вести себя предсказуемо.
- **Предотвращение ошибок:** лучше предотвратить ошибку, чем показывать сообщение о ней.
- **Гибкость и эффективность:** наличие "горячих клавиш" для опытных пользователей.

- **Эстетика и минимализм:** только нужная информация.

Методы проведения: наблюдение за пользователем, A/B тестирование, опросы, анализ кликов (click-tracking), составление CJM (Customer Journey Map).

Рабочее задание

1.Разработка сценария. Для заданного веб-сайта (например, сайт университета или интернет-магазин) разработать сценарий юзабилити-тестирования, состоящий из 3-5 задач (например, "найти расписание группы", "положить товар в корзину и оформить заказ").

2.Проведение тестирования. Провести тестирование с участием 1-2 добровольцев (или выступить в роли испытуемого, фиксируя свои ощущения). Наблюдать за действиями пользователя, фиксировать моменты, где он испытывает затруднения или совершает ошибки. Записывать время выполнения каждой задачи.

3.Анализ и отчет. Составить отчет по итогам тестирования, включив в него:

– Список выявленных проблем юзабилити.

– Оценку серьезности каждой проблемы (критическая, серьезная, незначительная).

– Рекомендации по улучшению интерфейса для каждой найденной проблемы.

Практическое занятие № 13 «тестирование интеграции»

Цель работы: Изучить стратегии интеграции программных модулей. Научиться разрабатывать тестовые сценарии для проверки взаимодействия компонентов. Применять ПК 1.4 для проверки интеграции модулей.

Теоретическая часть

Интеграционное тестирование (Integration Testing) следует за модульным. Его цель — проверка взаимодействия между несколькими модулями (компонентами) системы после их объединения. Ошибки часто возникают именно на стыках модулей: в передаче данных, в очередности вызовов, в несоответствии форматов.

Подходы к интеграции:

- **"Большой взрыв" (Big Bang):** все модули собираются вместе и тестируются сразу. Просто, но сложно локализовать ошибку.
- **Восходящая (Bottom-Up):** сначала тестируются низкоуровневые модули, затем к ним подключаются вышестоящие. Требуются "драйверы" (driver) для вызова тестируемых модулей.
- **Нисходящая (Top-Down):** сначала тестируются модули верхнего уровня, а заглушки (stubs, mocks) имитируют работу отсутствующих нижних.
- **Смешанная (сэндвич):** комбинация восходящей и нисходящей.

Заглушки (Test Doubles): объекты-заменители реальных компонентов для изоляции тестируемого модуля (Stub, Mock, Fake).

Рабочее задание

1.Анализ архитектуры. Получить описание архитектуры простой системы из двух-трех модулей (например, Модуль Аутентификации и Модуль Профиля). Определить возможные точки сбоев при их взаимодействии.

2.Разработка интеграционных тестов. Для связки этих модулей разработать сценарии, проверяющие:

– Корректную передачу данных (ID пользователя) от модуля к модулю.

– Реакцию на некорректные данные от соседнего модуля (например, пустой ID).

– Обработку ситуации, когда один из модулей недоступен.

3.Работа с заглушками. Написать простую заглушку (stub) для базы данных или внешнего API, которая возвращает заранее заданные ответы. Использовать эту заглушку для тестирования основного модуля в изоляции.

Практическое занятие № 14 «Документирование результатов тестирования»

Цель работы: Изучить структуру и содержание итоговой отчетности по тестированию. Научиться собирать статистику и анализировать результаты прогона тестов. Освоить формирование отчета о тестировании в соответствии с требованиями организации (ПК 1.4).

Теоретическая часть

Результаты тестирования должны быть задокументированы для информирования команды и заказчика о качестве продукта. Основные отчетные документы:

- **Отчет о тестировании (Test Summary Report):** итоговый документ по завершении этапа тестирования.
    - Статистика: всего тестов, пройдено (PASS), упало (FAIL), заблокировано.
    - Количество найденных дефектов (открыто, закрыто, по критичности).
    - Анализ качества продукта (готовность к релизу, риски).

- **Протокол выполнения тестов (Test Run Log):** детальная запись о прогоне конкретного набора тестов.
- **Метрики:** графики динамики открытия/закрытия дефектов, процент покрытия требований тестами (requirements coverage).

Инструменты: TestRail, Zephyr, QTest, или просто Excel/Google Sheets.

Рабочее задание

1.Сбор данных. Получить от преподавателя выгрузку данных из баг-трекинговой системы и результаты последнего прогона тестов (или использовать данные, полученные на предыдущих занятиях).

2.Расчет метрик. На основе полученных данных рассчитать ключевые метрики: общее количество тестов, процент успешных тестов, плотность дефектов (количество дефектов на модуль), распределение дефектов по серьезности.

3.Составление отчета. Разработать итоговый отчет о тестировании (Test Summary Report) для гипотетического релиза. Отчет должен включать: краткую информацию об объекте, сводную статистику, график динамики дефектов, вывод о готовности продукта и перечень основных рисков. Оформить отчет в формате .docx или .pdf.

Практическое занятие № 15 «Работа с системой автоматизированного тестирования»

Цель работы: Изучить архитектуру и компоненты систем автоматизированного тестирования. Приобрести практические навыки создания простых автоматизированных

тестов для UI или API. Применять специализированное программное обеспечение для создания автотестов и ПК 1.4.

Теоретическая часть

Автоматизация тестирования — это использование специальных программных средств для выполнения тестов и сравнения ожидаемых и фактических результатов. Автоматизация позволяет быстро прогонять регрессионные тесты и ускоряет выход продукта на рынок.

Уровни автоматизации:

- **Модульные тесты** (Unit Tests).
- **Интеграционные тесты** (Integration Tests).
- **API тесты:** проверка бэкенда через HTTP-запросы (Postman, SoapUI, REST Assured).
- **UI-тесты (End-to-End):** эмуляция действий пользователя в браузере (Selenium WebDriver, Cypress, Playwright).

Фреймворки автоматизации: структура, объединяющая библиотеки, правила и инструменты для упрощения процесса написания и поддержки автотестов.

Рабочее задание

1.Настройка инструмента. Установить и настроить один из инструментов для автоматизации:

- **Вариант А (API):** Postman. Создать коллекцию с запросами к публичному тестовому API (например, JSONPlaceholder).
- **Вариант Б (UI):** Selenium IDE (расширение для браузера). Записать сценарий навигации по любому сайту.

2.Разработка автотеста. В выбранном инструменте добавить проверки (assertions):

- Для API: проверить код ответа (200) и наличие определенных полей в JSON-ответе.
- Для UI: проверить наличие текста на странице после клика.

3.Запуск и анализ. Запустить выполнение автотеста. Получить отчет о его прохождении. Намеренно внести ошибку в проверяемые данные (изменить ожидаемый результат), чтобы тест "упал", и проанализировать сообщение об ошибке.

Практическое занятие № 16 «Ревьюирование, рефакторинг и оптимизация кода»

Цель работы: Изучить цели и процесс ревьюирования кода. Научиться выявлять "запахи кода" и проблемные места в исходном коде. Применять ПК 1.5 для проведения рефакторинга кода и исправления дефектов.

Теоретическая часть

- **Ревьюирование кода (Code Review):** систематическая проверка исходного кода программы с целью обнаружения и исправления ошибок, которые остались незамеченными на начальных этапах разработки. Цели: повышение качества кода, поиск дефектов, обучение команды, соблюдение стандартов.
- **Рефакторинг (Refactoring):** процесс изменения внутренней структуры программы, не затрагивающий её внешнее поведение и направленный на улучшение её качества. Цели: улучшение читаемости, снижение сложности, устранение "запахов кода" ("дублирование кода", "длинный метод", "большой класс").
- **Оптимизация:** улучшение характеристик производительности (скорости, потребления памяти). Часто требует компромиссов с читаемостью.

Рабочее задание

1.Проведение Code Review. Получить фрагмент "плохого" кода (с нарушениями стандартов, дублированием, неинформативными именами переменных). Провести ревью этого кода, составив список замечаний по каждому проблемному месту.

2.Рефакторинг. Выполнить рефакторинг полученного кода: переименовать переменные, разбить длинный метод на несколько более мелких, устранить дублирование. Убедиться, что после изменений код по-прежнему выполняет ту же функцию (прогнать модульные тесты, если они есть, или провести ручную проверку).

3.Анализ результатов. Сравнить "до" и "после". Описать, какие "запахи кода" были устранены и как проведенный рефакторинг улучшит дальнейшую поддержку этого программного модуля.

Практическое занятие № 17 «Анализ логов и отчетов об ошибках»

Цель работы: Освоить методику чтения и анализа лог-файлов. Научиться извлекать из стек-трейса информацию для локализации ошибки. Применять ПК 1.6 для анализа инцидентов и восстановления данных (в контексте поиска причин сбоев).

Теоретическая часть

Логи (журналы событий) — это файлы, в которые программы и ОС записывают информацию о своей работе: ошибки, предупреждения, информационные сообщения. Умение читать логи — ключевой навык для эксплуатации и тестирования.

Уровни логирования (Log Levels): DEBUG (отладка), INFO (информация), WARN (предупреждение), ERROR (ошибка), FATAL (критическая ошибка).

Стек-трейс (Stack Trace) — это отчет об активных фреймах стека в момент возникновения исключения. Позволяет точно определить последовательность вызовов методов, приведшую к ошибке.

Инструменты анализа: централизованные системы сбора логов (Splunk, ELK Stack — Elasticsearch, Logstash, Kibana), grep, tail, Less.

Рабочее задание

1.Поиск по логам. Получить файл с логами работы веб-сервера. С помощью команды grep (или поиска в текстовом редакторе) найти все записи уровня ERROR за определенный промежуток времени.

2.Анализ стек-трейса. Получить фрагмент лога, содержащий исключение (Exception) и стек-трейс. Проанализировать его и определить:

- Тип исключения.
- Класс и метод, в котором произошла ошибка.
- Строку кода, вызвавшую исключение.
- Возможную причину (например, деление на ноль, обращение к null).

3.Составление отчета об инциденте. На основе анализа логов из задания 2 составить краткий отчет об инциденте, включающий: время инцидента, описание симптомов, локализацию ошибки (класс, метод) и предполагаемую причину.

ПЕРЕЧЕНЬ ИСПОЛЬЗОВАННЫХ ИНФОРМАЦИОННЫХ РЕСУРСОВ

1. Тестирование программного обеспечения. Учебное пособие для СПО. 4-е издание, стереотипное автор А. В. Игнатьев, 2025, 54 стр. ISBN: 978-5-507-54395-3

2. Качество и тестирование программного обеспечения. Метрология программного обеспечения, автор А. В. Проскуряков, 198 стр. ISBN:978-5-9275-4044-0

3. Основы тестирования программного обеспечения. Учебное пособие для СПО. 5-е издание, стереотипное автор С. М. Старолетов, 190 стр. ISBN: 978-5-507-53820-1