Приложение 1.2.3 к ОПОП по специальности
09.02.11 Разработка и управление программным обеспечением

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Магнитогорский государственный технический университет им. Г.И. Носова»

Многопрофильный колледж

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ЛАБОРАТОРНЫХ ЗАНЯТИЙ МЕЖДИСЦИПЛИНАРНОГО КУРСА

МДК.02.03 ПОДДЕРЖКА И ТЕСТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ

для обучающихся специальности
09.02.11 Разработка и управление программным обеспечением

СОДЕРЖАНИЕ

1 ВВЕДЕНИЕ

Важную часть теоретической и профессиональной практической подготовки обучающихся составляют лабораторные занятия.

Состав и содержание лабораторных занятий направлены на реализацию Федерального государственного образовательного стандарта среднего профессионального образования.

Ведущей дидактической целью лабораторных занятий является экспериментальное подтверждение и проверка существенных теоретических положений (законов, зависимостей).

В соответствии с рабочей программой профессионального модуля «Разработка и интеграция модулей программного обеспечения» предусмотрено проведение лабораторных занятий.

В результате их выполнения, обучающийся должен

уметь:

У 2.4.1 анализировать требования к программному обеспечению и составлять планы тестирования;

У 2.4.2 создавать тестовые сценарии и тест-кейсы для проверки функциональности и соответствия требованиям;

У 2.4.3 выполнять тестирование программного обеспечения вручную и автоматизировать процесс тестирования;

У 2.4.4 анализировать результаты тестирования и документировать найденные ошибки;

У 2.4.5 разрабатывать стратегии отладки и исправлять ошибки в программном обеспечении;

У 2.4.6 выполнять модульные тесты с использованием инструментов тестирования, в том числе автоматизированного тестирования;

У 2.4.7 использовать системы контроля дефектов ПО;

У 2.4.8 составлять отчет о выполнении тестирования ПО.

Содержание практических и лабораторных занятий ориентировано на освоение вида деятельности программы подготовки специалистов среднего звена по специальности и овладению ***профессиональными компетенциями:***

ПК 2.4 Выполнять тестирование и отладку программного обеспечения

А также формированию общих компетенций:

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

Лабораторные занятия проводятся в рамках соответствующей темы, после освоения дидактических единиц, которые обеспечивают наличие знаний, необходимых для ее выполнения.

2 МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Тема 3.1 Основы обеспечения качества ПО

Лабораторное занятие №55.

Анализ качества информационной системы с использованием метрик качества

Цель: Освоить методики количественной оценки и анализа качества информационной системы на основе метрик, научиться выявлять проблемные области и обосновывать необходимость доработок.

Выполнив работу, вы будете уметь:

У 2.4.4 анализировать результаты тестирования и документировать найденные ошибки

Материальное обеспечение:

1. Доступ к тестируемой информационной системе (или её демо-версии).
2. Спецификация требований к системе.
3. Шаблоны для сбора метрик (таблицы, чек-листы).
4. Доступ к электронно-библиотечной системе (ЭБС) для изучения теоретических материалов.

Задание:

1. Изучить теоретические основы оценки качества ПО по стандартам (например, ГОСТ Р ИСО/МЭК 25010).
2. Выбрать не менее 5 ключевых метрик качества, релевантных для данной системы.
3. Провести сбор данных и рассчитать значения выбранных метрик.
4. Проанализировать полученные результаты и подготовить отчет.

Порядок выполнения работы:

1. Ознакомьтесь с краткими теоретическими сведениями о метриках качества ПО (см. ЭБС).
2. Изучите спецификацию требований к информационной системе.
3. Выберите метрики: Определите набор метрик (например, среднее время отклика, количество ошибок на 1000 строк кода, процент покрытия тестами, количество открытых дефектов).
4. Сбор данных: Проведите необходимые измерения или анализ логов системы для получения исходных данных.
5. Расчет: Вычислите значения выбранных метрик на основе собранных данных.
6. Анализ: Сравните полученные значения с эталонными или нормативными. Выявите «узкие места» системы.
7. Рекомендации: Сформулируйте предложения по улучшению качества системы на основе анализа.

Форма представления результата:

Отчет в виде аналитической записки, содержащей:

1. Перечень и описание выбранных метрик.
2. Таблицу с исходными данными и результатами расчетов.
3. Графики или диаграммы для визуализации данных (при необходимости).
4. Выводы о качестве системы и список рекомендаций по доработке.

Лабораторное занятие №56.

Составление отчетов о дефектах. Работа с баг-трекинговыми системами (Jira, Redmine)

Цель: Научиться корректно регистрировать дефекты в баг-трекинговых системах, использовать их функционал для управления жизненным циклом ошибок и коммуникации в команде разработки.

Выполнив работу, вы будете уметь:

У 2.4.7 использовать системы контроля дефектов ПО

Материальное обеспечение:

1. Учебный проект в баг-трекинговой системе (Jira/Redmine) с доступом для студентов.

2. Ссылка на тестируемое приложение (или его тестовая версия).

Задание:

1. Зарегистрироваться/авторизоваться в учебной баг-трекинговой системе.
2. Найти в тестируемом приложении не менее 3-х дефектов разного типа (функциональная ошибка, визуальный баг, проблема производительности).
3. Создать задачи (Issues/Баги) для каждого найденного дефекта с корректным заполнением всех полей.
4. Продемонстрировать переход задачи по жизненному циклу (например, из статуса "To Do" в "In Progress" и "Done").

Порядок выполнения работы:

1. Ознакомьтесь с интерфейсом выбранной баг-трекинговой системы (Jira/Redmine).
2. Изучите правила оформления баг-репортов в вашей организации/команде. Обратите внимание на обязательные поля: Summary (Краткое описание), Description (Подробное описание), Steps to Reproduce (Шаги воспроизведения), Expected Result (Ожидаемый результат), Actual Result (Фактический результат), Priority (Приоритет), Severity (Серьезность).
3. Запустите тестируемое приложение и проведите его исследование с целью поиска дефектов.
4. Создание задачи: Для каждого найденного дефекта создайте новую задачу в системе. Заполните все обязательные поля максимально подробно. Прикрепите скриншоты или видеозапись экрана, демонстрирующую ошибку.
5. Назначение и отслеживание: Назначьте задачу на себя или другого исполнителя. Измените статус задачи согласно инструкции преподавателя. Добавьте комментарий к задаче с уточняющим вопросом или статусом проверки.
6. Связи: Если найденный дефект блокирует выполнение другой задачи, создайте связь между ними в системе.

Форма представления результата:

Скриншоты созданных задач в баг-трекинговой системе. В отчете должны быть представлены скриншоты карточки дефекта со всеми заполненными полями и скриншот истории изменений статуса задачи.

Лабораторное занятие №57.
Документирование дефектов и формирование отчета о тестировании

Цель: Систематизировать информацию о найденных дефектах и сформировать итоговый документ — отчет о тестировании — для передачи заинтересованным сторонам (менеджменту, команде разработки).

Выполнив работу, вы будете уметь:

У 2.4.8 составлять отчет о выполнении тестирования ПОСодержание практических и лабораторных занятий ориентировано на освоение вида деятельности программы подготовки специалистов среднего звена по специальности и овладению **Материальное обеспечение:**

Список дефектов, зарегистрированных в ходе выполнения ЛР №56 или предоставленных преподавателем.

Задание:

1. Провести анализ базы дефектов из баг-трекинговой системы за определенный период тестирования.
2. Сгруппировать дефекты по категориям (например, по модулям системы или типу ошибки).
3. Рассчитать ключевые показатели тестирования (количество найденных/исправленных/открытых дефектов).
4. Сформировать итоговый отчет о тестировании на основе полученных данных.

Порядок выполнения работы:

1. Используя функционал баг-трекинговой системы (Jira/Redmine), выгрузите список всех дефектов, созданных за время тестирования учебного проекта. При необходимости используйте фильтры для отбора задач по проекту или дате создания.
2. Анализ дефектов: Изучите список дефектов. Для каждого дефекта определите его категорию (например, UI, Backend, База данных). Оцените распределение дефектов по серьезности (Blocker, Critical, Major, Minor, Trivial) и приоритету (High, Medium, Low).
3. Статистика: Подсчитайте общее количество дефектов. Рассчитайте долю дефектов каждой категории и каждого уровня серьезности в общем объеме. Определите количество дефектов в статусе "Open" (открыто), "In Progress" (в работе), "Resolved" (исправлено) и "Closed" (закрыто).
4. Формирование отчета: На основе проведенного анализа составьте отчет о тестировании. Включите в него краткое описание целей тестирования, методику, основные выводы и статистику.

Форма представления результата:

Итоговый отчет о тестировании в формате документа (например, .docx). Отчет должен содержать:

1. Титульный лист и оглавление.
2. Введение (цели и область тестирования).
3. Сводную таблицу статистики дефектов (общее количество, распределение по категориям/серьезности).
4. Диаграммы распределения дефектов.
5. Выводы о качестве продукта на основе анализа дефектов.
6. Приложение со списком наиболее критичных открытых дефектов.

Тема 3.2. Тест-дизайн и тестирование ПО
Лабораторное занятие №58.
Проектирование тест-кейсов для модульного тестирования

Цель: Освоить методики проектирования и документирования тест-кейсов для проверки корректности работы отдельных программных модулей (unit-тестов) в соответствии с требованиями.

Выполнив работу, вы будете уметь:

У 2.4.2 создавать тестовые сценарии и тест-кейсы для проверки функциональности и соответствия требованиям

Материальное обеспечение:

1. Спецификация требований к программному модулю (ТЗ).
2. Текстовый редактор или специализированный инструмент для управления тестированием.
3. Доступ к ЭБС.

Задание:

1. Изучить спецификацию требований к заданному программному модулю.
2. Спроектировать не менее 5 тест-кейсов для проверки основных функций модуля.
3. Оформить тест-кейсы в соответствии с шаблоном.

Порядок выполнения работы:

1. Ознакомьтесь с краткими теоретическими сведениями о проектировании тест-кейсов и техниках тест-дизайна.
2. Изучите предоставленную спецификацию требований к программному модулю.

3. Выделите входные данные, которые необходимо проверить. Примените техники эквивалентного разбиения и анализа граничных значений для выбора наиболее эффективных тестов.
4. Для каждого сценария спроектируйте тест-кейс, заполнив следующие поля:

– ID: Уникальный идентификатор (например, TC-01).
– Приоритет: Уровень важности теста.
– Предусловия: Состояние системы перед началом теста.
– Шаги воспроизведения: Последовательность действий для выполнения теста.
– Ожидаемый результат: Что должно произойти при корректной работе модуля.
– Постусловия: Состояние системы после выполнения теста.

5. Проведите перекрестную проверку (Peer Review) разработанных тест-кейсов на полноту и однозначность.

Форма представления результата:

Таблица, оформленная в текстовом редакторе, содержащая спроектированные тест-кейсы. Каждый тест-кейс должен быть представлен как отдельная строка таблицы со всеми обязательными полями.

Лабораторное занятие №59.
Разработка и выполнение автоматизированных тестов с использованием Selenium

Цель: Получить практические навыки создания и запуска автоматизированных тестов для веб-интерфейсов с использованием фреймворка Selenium WebDriver.

Выполнив работу, вы будете уметь:

У 2.4.3 выполнять тестирование программного обеспечения вручную и автоматизировать процесс тестирования

Материальное обеспечение:

1. Среда разработки (IDE) с установленным языком программирования (Python/Java).
2. Библиотека Selenium WebDriver и соответствующий драйвер браузера (ChromeDriver/GeckoDriver).
3. Доступ к тестируемому веб-приложению.

Задание:

1. Настроить рабочее окружение для работы с Selenium.
2. Разработать автоматизированный тест, выполняющий сценарий авторизации пользователя на веб-сайте.
3. Запустить тест и зафиксировать результат его выполнения.

Порядок выполнения работы:

1. Установите необходимые инструменты: язык программирования, библиотеку Selenium и драйвер для браузера. Создайте новый проект в вашей IDE.
2. Ознакомьтесь с базовыми командами Selenium WebDriver для открытия браузера, навигации по страницам и поиска элементов (по id, name, xpath).
3. Разработка теста:
4. Создайте скрипт, который открывает браузер и переходит на страницу авторизации тестируемого веб-приложения.
5. Найдите на странице поля для ввода логина и пароля, а также кнопку входа. Используйте соответствующие методы Selenium для ввода текста (send_keys) и нажатия на элемент (click).
6. Добавьте проверку (assert), которая подтверждает успешность входа. Например, проверка наличия элемента, видимого только для авторизованных пользователей.
7. Запустите созданный скрипт. Убедитесь, что браузер открывается, действия выполняются, и тест завершается успешно или падает с ошибкой.

8. В случае падения теста проанализируйте логи и скриншот страницы (если реализовано) для определения причины ошибки.

Форма представления результата:

1. Исходный код разработанного автоматизированного теста (файл .py или .java).
2. Скриншот окна браузера с результатом выполнения теста (успешный вход или сообщение об ошибке).
3. Краткий отчет о выполнении: описание сценария, результат (Passed/Failed), возможные проблемы.

Лабораторное занятие №60.
Тестирование документации на ПО

Цель: Научиться проводить проверку технической документации на соответствие стандартам качества, полноту, непротиворечивость и понятность для конечного пользователя.

Выполнив работу, вы будете уметь:

У 2.4.4 анализировать результаты тестирования и документировать найденные ошибки

Материальное обеспечение:

1. Комплект технической документации на программное обеспечение (руководство пользователя, техническое задание).
2. Текстовый редактор с функцией рецензирования или система отслеживания дефектов.

Задание:

1. Изучить предоставленный комплект документации на ПО.
2. Провести проверку документации на соответствие критериям качества.
3. Выявить не менее 5 дефектов различного типа и зарегистрировать их в виде отчета.

Порядок выполнения работы:

1. Ознакомьтесь с критериями качества технической документации: полнота, точность, непротиворечивость, актуальность, понятность, соответствие стандартам оформления.
2. Внимательно изучите предоставленный документ (например, «Руководство пользователя»).
3. Проведите проверку по следующим пунктам:

– Полнота: Описаны ли все заявленные функции ПО? Нет ли пропущенных шагов в инструкциях?
– Точность: Соответствуют ли описания реальному поведению программы? Нет ли фактических ошибок?
– Непротиворечивость: Не противоречат ли разные разделы документа друг другу?
– Понятность: Ясен ли текст конечному пользователю? Нет ли сложных терминов без объяснения?
– Оформление: Соблюдены ли стили заголовков, шрифтов? Корректны ли ссылки и оглавление?

4. Для каждого найденного дефекта составьте отчет о дефекте по аналогии с баг-репортом в системе отслеживания задач:

– ID: Уникальный номер дефекта в документации (Doc-01).
– Описание: Краткое описание проблемы (например, «Отсутствует описание функции X»).
– Местонахождение: Точное место в документе (страница, раздел).
– Серьезность/Приоритет: Оценка влияния дефекта на пользователя.
– Шаги воспроизведения/Анализ: Как была обнаружена проблема.

5. Сформируйте итоговый список дефектов.

Форма представления результата:

Таблица со списком выявленных дефектов в документации. Таблица должна содержать столбцы: ID дефекта, Описание, Местонахождение, Серьезность/Приоритет.

Лабораторное занятие №61.
Выполнение модульных тестов с использованием инструментов тестирования

Цель: Приобрести практический опыт запуска существующих модульных тестов (unit tests) из интегрированной среды разработки (IDE) и анализа результатов их выполнения для оценки корректности работы модуля.

Выполнив работу, вы будете уметь:

У 2.4.6 выполнять модульные тесты с использованием инструментов тестирования, в том числе автоматизированного тестирования

Материальное обеспечения:

1. Проект программного модуля с уже написанными модульными тестами (исходный код + тесты).
2. Интегрированная среда разработки (IDE, например, PyCharm, IntelliJ IDEA) или система сборки (Maven, Gradle, pytest).

Задание:

1. Открыть проект с исходным кодом и модульными тестами в IDE или системе сборки.
2. Выполнить все имеющиеся модульные тесты.
3. Проанализировать результаты выполнения. В случае наличия упавших тестов — локализовать и описать причину ошибки в коде модуля.

Порядок выполнения работы:

1. Импортируйте предоставленный проект в вашу среду разработки (IDE) или перейдите в каталог проекта в терминале.
2. Найдите в структуре проекта каталог или файлы с модульными тестами. Ознакомьтесь с их названиями и кратким описанием того, что они проверяют.
3. Запустите выполнение всех модульных тестов. В большинстве IDE это делается через контекстное меню проекта или с помощью специальной кнопки запуска тестов. В терминале это может быть команда pytest, mvn test или аналогичная.
4. Дождитесь окончания выполнения. Система отобразит отчет:

– Количество выполненных тестов.
– Количество успешных тестов (Passed).
– Количество упавших тестов (Failed).
– Если есть упавшие тесты, откройте отчет о выполнении. Изучите сообщение об ошибке и стек вызовов (stack trace), чтобы понять, какой именно метод или строка кода вызвали сбой.

5. Сравните ожидаемый результат из теста с фактическим результатом работы модуля.
6. Сделайте вывод о корректности работы программного модуля на основе результатов тестирования.

Форма представления результата:

Скриншот окна терминала или IDE с отчетом о выполнении модульных тестов. Если были упавшие тесты — краткое описание причины их падения со ссылкой на строку кода из стека вызовов. Итоговый вывод о качестве модуля («Модуль работает корректно» / «Обнаружены ошибки...»).

Лабораторное занятие №62.
Разработка тестов для проверки подключения баз данных

Цель: Научиться проектировать и выполнять тесты для проверки корректности взаимодействия приложения с системой управления базами данных (СУБД), включая проверку подключения и выполнение базовых SQL-запросов.

Выполнив работу, вы будете уметь:

У 2.4.1 анализировать требования к программному обеспечению и составлять планы тестирования

Материальное обеспечение:

1. Тестируемое приложение с конфигурационным файлом для подключения к БД (или его заготовка).
2. Доступ к СУБД (локальной или удаленной) с известными учетными данными.
3. Инструмент для прямого подключения к БД (например, DBeaver) для проверки данных вручную.

Задание:

1. Настроить параметры подключения к базе данных в конфигурационном файле приложения.
2. Разработать и выполнить тесты для проверки успешного подключения к БД и выполнения простого SQL-запроса (например, SELECT).
3. Проверить поведение приложения при использовании некорректных параметров подключения.

Порядок выполнения работы:

1. Получите параметры подключения к СУБД: адрес сервера (host), порт (port), имя базы данных (database name), логин (username) и пароль (password). Убедитесь через прямой клиент БД (DBeaver), что база доступна по этим данным.
2. Откройте конфигурационный файл приложения (например, config.properties, application.yml или app.config) и укажите полученные параметры подключения. Сохраните изменения.
3. Запустите приложение. Если оно выводит логи подключения к БД при старте, убедитесь по ним, что соединение установлено успешно. Если нет — перейдите к следующему шагу.
4. Разработайте простой скрипт или используйте существующий функционал приложения для выполнения SQL-запроса к базе данных. Например, запрос на получение списка таблиц (SHOW TABLES;) или выборку данных из известной таблицы (SELECT * FROM users LIMIT 1;).
5. Выполните этот запрос через приложение. Убедитесь, что результат получен без ошибок. Сверьте результат с тем, что вы видите при выполнении этого же запроса напрямую в клиенте БД (DBeaver).
6. Тест на негативный сценарий: Временно измените один из параметров подключения в конфиге на заведомо неверный (например, пароль). Повторно запустите приложение или выполните запрос. Убедитесь, что приложение корректно обрабатывает ошибку подключения (выводит понятное сообщение об ошибке или логирует исключение), а не "падает" с критической ошибкой.
7. Верните правильные параметры в конфигурационный файл.

Форма представления результата:

1. Скриншот конфигурационного файла с параметрами подключения к БД (с закрашенными паролями).
2. Скриншот вывода логов приложения или консоли со строкой об успешном подключении к БД и результатом выполнения SQL-запроса.
3. Скриншот вывода логов приложения при попытке подключения с неверными параметрами (демонстрация обработки ошибки).

Тема 3.3 Регрессионное тестирование и восстановление после сбоев

Лабораторное занятие №63.
Выполнение регрессионного тестирования и анализ результатов

Цель: Освоить процесс выполнения регрессионного тестирования для подтверждения того, что внесенные изменения в код не нарушили существующую функциональность, и научиться анализировать полученные результаты.

Выполнив работу, вы будете уметь:

У 2.4.3 выполнять тестирование программного обеспечения вручную и автоматизировать процесс тестирования

У 2.4.4 анализировать результаты тестирования и документировать найденные ошибки

Материальное обеспечение:

1. Доступ к системе контроля версий (Git) с ветками проекта.
2. Существующий набор автоматизированных или ручных тестов.
3. Инструмент для сравнения файлов/отчетов.

Задание:

1. Выполнить регрессионное тестирование программного модуля после внесения изменений.
2. Зафиксировать и проанализировать результаты, сравнив их с предыдущим прогоном.
3. Оформить отчет о выполнении регрессионного тестирования.

Порядок выполнения работы:

1. Ознакомьтесь с краткими теоретическими сведениями о регрессионном тестировании и его целях.
2. Получите доступ к ветке кода, в которую были внесены изменения.
3. Подготовьте тестовую среду: убедитесь, что она чистая и соответствует требованиям.
4. Запустите существующий набор регрессионных тестов (автоматизированный или ручной).
5. Зафиксируйте результаты выполнения: количество пройденных и упавших тестов, время выполнения.
6. Сравните полученные результаты с результатами предыдущего стабильного прогона (базовой линией). Используйте инструменты сравнения для выявления новых или повторно открытых дефектов.
7. Проанализируйте причины падения тестов: являются ли они следствием новых изменений или нестабильностью самих тестов.
8. Сформируйте итоговый отчет.

Форма представления результата:

Отчет о выполнении регрессионного тестирования, содержащий:

- Сводную таблицу сравнения результатов текущего и предыдущего прогонов.
- Список новых дефектов, выявленных в ходе регрессии.
- Выводы о стабильности продукта после внесенных изменений.

Лабораторное занятие №64.
Имитация сбоя и восстановление автоматизированных тестов

Цель: Научиться диагностировать причины падения автоматизированных тестов, отличать сбои в коде теста от сбоев в тестируемом приложении, и восстанавливать работоспособность тестового набора.

Выполнив работу, вы будете уметь:

У 2.4.5 разрабатывать стратегии отладки и исправлять ошибки в программном обеспечении

Материальное обеспечение:

1. Проект с набором автоматизированных тестов (например, на Selenium/Playwright).
2. Интегрированная среда разработки (IDE) с отладчиком.
3. Тестируемое веб-приложение.

Задание:

1. Запустить набор автоматизированных тестов и зафиксировать упавший тест.
2. Проанализировать отчет об ошибке и локализовать причину сбоя.
3. Восстановить работоспособность теста путем исправления кода или обновления данных.
4. Повторно запустить тест для подтверждения исправления.

Порядок выполнения работы:

1. Запустите существующий набор автоматизированных тестов. Дождитесь падения одного из них.
2. Изучите отчет о выполнении. Откройте логи и стек вызовов (stack trace) для упавшего теста.
3. Определите тип ошибки:
4. Ошибка в приложении: Тест падает, потому что фактический результат не совпадает с ожидаемым (например, кнопка не нажимается). В этом случае создайте баг-репорт в системе отслеживания задач.
5. Ошибка в тесте: Тест падает из-за проблем в коде самого теста (например, изменился id элемента на странице, и локатор больше не работает).
6. Если ошибка в тесте, откройте исходный код упавшего теста в IDE. Используйте отладчик для пошагового выполнения кода и определения точного места сбоя.
7. Исправьте ошибку. Например, обновите локатор элемента, добавьте ожидание (wait) для загрузки динамического контента или исправьте логику проверки.
8. Сохраните изменения и повторно запустите только этот упавший тест. Убедитесь, что он проходит успешно.
9. Запустите весь набор тестов еще раз, чтобы убедиться, что ваше исправление не сломало другие тесты.

Форма представления результата:

1. Скриншот отчета о выполнении тестов с ошибкой до исправления.
2. Скриншот стека вызовов (stack trace) и фрагмента кода с внесенными правками.
3. Скриншот отчета о выполнении тестов после исправления (тест помечен как "Passed").
4. Краткое описание проделанной работы: причина сбоя и способ ее устранения.

Лабораторное занятие №65.
Проверка исправленных дефектов и оформление отчета

Цель: Научиться проводить верификацию исправлений (Verification) — проверку того, что дефект действительно устранен разработчиком в соответствии с описанием в баг-репорте, — и корректно оформлять ее результат.

Выполнив работу, вы будете уметь:

У 2.4.8 составлять отчет о выполнении тестирования ПОСодержание практических и лабораторных занятий ориентировано на освоение вида деятельности программы подготовки специалистов среднего звена по специальности и овладению **Материальное обеспечение:**

1. Баг-трекинговая система (Jira/Redmine) с назначенными на вас задачами на верификацию.
2. Доступ к новой сборке ПО с внесенными исправлениями.

Задание:

1. Получить задачу на проверку исправления дефекта из баг-трекинговой системы.
2. Выполнить шаги по воспроизведению дефекта согласно описанию в баг-репорте на новой версии ПО.
3. Оформить результат проверки, изменив статус задачи или добавив комментарий.

Порядок выполнения работы:

1. В баг-трекинговой системе найдите задачу со статусом "Ready for QA" или "На проверке", назначенную на вас. Ознакомьтесь с описанием дефекта и шагами его воспроизведения из оригинального отчета.
2. Получите доступ к новой сборке программного обеспечения, где разработчик должен был исправить дефект.
3. Верификация: Последовательно выполните все шаги воспроизведения дефекта, описанные в задаче. Будьте внимательны к деталям и ожидаемому результату.
4. Оценка результата:

– Если дефект не воспроизводится, а система ведет себя согласно ожидаемому результату — дефект исправлен корректно. Измените статус задачи на "Verified" / "Проверено" или "Closed" / "Закрыто".
– Если дефект по-прежнему воспроизводится, сделайте скриншоты или видеозапись экрана, подтверждающие это. Добавьте их в задачу как вложение. Измените статус задачи на "Reopen" / "Переоткрыть" с комментарием "Дефект не исправлен. Проблема сохраняется".
– Если дефект не воспроизводится, но поведение системы не соответствует ожидаемому результату, описанному в задаче (исправлено не так), также переоткройте задачу с подробным описанием нового поведения.

5. Если при проверке вы нашли новый дефект, не связанный с текущей задачей, создайте для него новую отдельную задачу в системе.

Форма представления результата:

1. Скриншоты из баг-трекинговой системы:
2. Скриншот карточки задачи до начала проверки (со статусом "Ready for QA").
3. Скриншот карточки задачи после завершения проверки со статусом "Verified" или "Reopen" и вашим комментарием/вложениями (скриншоты/видео).

Лабораторное занятие №66.
Настройка системы логирования и анализ логов

Цель: Научиться настраивать систему логирования в приложении для записи событий разного уровня важности и анализировать файлы журналов (log files) для диагностики проблем и расследования инцидентов.

Выполнив работу, вы будете уметь:

У 2.4.7 использовать системы контроля дефектов ПО

Материальное обеспечение:

1. Исходный код тестируемого приложения (или его части).
2. Библиотека логирования (например, log4j, logback для Java; logging для Python).
3. Текстовый редактор для просмотра файлов журналов.

Задание:

1. Настроить конфигурационный файл системы логирования приложения.
2. Добавить операторы логирования в исходный код модуля.
3. Сгенерировать события в приложении для записи логов.
4. Проанализировать полученный файл журнала для поиска информации об ошибке или конкретном действии пользователя.

Порядок выполнения работы:

1. Найдите в проекте конфигурационный файл системы логирования (например, log4j2.xml, logback.xml или logging.conf). Ознакомьтесь с текущими настройками: куда пишутся логи (file appender), какой уровень логирования установлен (INFO, DEBUG, ERROR).
2. Измените настройки:

3. Установите уровень логирования на DEBUG для более детального вывода во время теста.
4. Настройте ротацию файлов логов (если требуется), чтобы они не занимали слишком много места.
5. Убедитесь, что формат вывода включает дату/время, уровень лога (ERROR, WARN, INFO), имя класса/метода и само сообщение.
6. Откройте исходный код программного модуля. В ключевых точках кода (в начале методов, после важных операций, в блоках try-catch) добавьте операторы записи логов:

– logger.info("Начало обработки запроса...")
– logger.debug("Получены параметры: {}", params);
– logger.error("Ошибка при сохранении данных", exception);

7. Соберите и запустите приложение. Выполните несколько действий пользователя или операций, которые должны сгенерировать записанные вами сообщения в логах.
8. Найдите файл журнала приложения. Откройте его в текстовом редакторе или IDE. Используйте функции поиска (Ctrl+F) по ключевым словам (например, по ERROR или по идентификатору операции), чтобы найти нужную информацию о последовательности действий и возникших ошибках.
9. Проанализируйте найденные записи для определения причины сбоя или подтверждения корректного выполнения операции.

Форма представления результата:

1. Фрагмент измененного конфигурационного файла логирования с выделенными новыми параметрами.
2. Фрагмент исходного кода с добавленными операторами logger.info/debug/error.
3. Скриншот файла журнала (*.log), на котором обведены или выделены ключевые записи, подтверждающие выполнение задания или наличие ошибки.

Лабораторное занятие №67.
Составление сценариев поведения пользователей для регрессионного тестирования

Цель: Научиться проектировать высокоуровневые сценарии использования системы (User Scenarios / User Journeys) на основе анализа требований и реального опыта пользователей для формирования эффективного набора регрессионных тестов.

Выполнив работу, вы будете уметь:

У 2.4.2 создавать тестовые сценарии и тест-кейсы для проверки функциональности и соответствия требованиям

Материальное обеспечение:

1. Спецификация требований к программному продукту (ТЗ).
2. Документация по пользовательскому интерфейсу (UI-кит, макеты).

Задание:

1. Изучить документацию на программный продукт и выделить основные пользовательские роли.
2. Для одной из ролей составить не менее двух сценариев поведения пользователей ("счастливый путь" и альтернативный сценарий).
3. Декомпозировать один из сценариев на детальные шаги для создания тест-кейсов регрессии.

Порядок выполнения работы:

1. Ознакомьтесь с краткими теоретическими сведениями о сценарном тестировании и понятии "счастливый путь" (Happy Path) (ЭБС, раздел «Тест-дизайн»).

2. Изучите спецификацию требований к программному продукту. Определите основные типы пользователей системы (роли), например: «Администратор», «Менеджер», «Клиент».
3. Выберите одну ключевую роль. Определите основную цель пользователя этой роли при работе с системой (например, «Клиент хочет купить товар»).
4. Составьте сценарий «Счастливый путь»: опишите последовательность действий пользователя от начала до успешного достижения цели без каких-либо ошибок или отклонений. Используйте формат «Роль -> Цель -> Шаги».
5. Составьте один альтернативный сценарий, описывающий ситуацию с отклонением от основного пути или обработкой ошибки (например, «Клиент вводит неверный пароль»).
6. Выберите один из сценариев («Счастливый путь» является приоритетным) и декомпозируйте его на конкретные шаги взаимодействия с интерфейсом. Каждый шаг должен быть однозначным действием («Нажать кнопку 'Войти'», «Ввести текст 'test' в поле 'Логин'»). Эти шаги станут основой для создания детальных тест-кейсов регрессионного пакета.

Форма представления результата:

Документ в формате .docx, содержащий:

- Описание выделенных пользовательских ролей.
- Два сценария поведения пользователей («Счастливый путь» и альтернативный), оформленные по шаблону «Роль -> Цель -> Шаги».
- Декомпозицию выбранного сценария на детальные шаги взаимодействия с системой.

Тема 3.4 Модульное тестирование

Лабораторное занятие №68.
Разработка модульных автотестов для настольных приложений

Цель: Приобрести практические навыки проектирования и написания автоматизированных тестов для проверки функциональности настольных (desktop) приложений с использованием специализированных инструментов.

Выполнив работу, вы будете уметь:

У 2.4.6 выполнять модульные тесты с использованием инструментов тестирования, в том числе автоматизированного тестирования

Материальное обеспечение:

1. Установленный инструмент автоматизации (например, WinAppDriver).
2. Среда разработки (IDE) с поддержкой языка программирования (Python, C#).
3. Тестируемое настольное приложение.

Задание:

1. Настроить среду для автоматизации тестирования настольного приложения.
2. Разработать и реализовать автотест для проверки базовой функциональности приложения (например, авторизация).
3. Запустить автотест и убедиться в его успешном выполнении.

Порядок выполнения работы:

1. Ознакомьтесь с краткими теоретическими сведениями об автоматизации тестирования десктопных приложений (ЭБС, раздел «Автоматизация тестирования»).
2. Установите и запустите драйвер автоматизации (например, WinAppDriver).
3. В среде разработки создайте новый проект. Установите необходимые библиотеки (например, pywinauto, Appium).
4. Запустите тестируемое приложение.
5. Разработка теста:

6. Напишите код для инициализации драйвера и подключения к запущенному приложению.
7. Используя локаторы (ID элемента, имя класса, текст), найдите необходимые элементы управления (кнопки, поля ввода).
8. Напишите последовательность действий: ввод данных в поля, нажатие кнопок.
9. Добавьте проверку (assert) ожидаемого результата (например, появление окна приветствия после входа).
10. Запустите написанный скрипт. Проанализируйте результат выполнения.

Форма представления результата:

1. Исходный код разработанного автотеста (файл .py или .cs).
2. Скриншот окна приложения с результатом выполнения теста.
3. Скриншот консоли или отчета с успешным прохождением теста.

Лабораторное занятие №69.
Оформление отчетов о тестировании (создание библиотеки тестов)

Цель: Научиться структурировать код автотестов, используя паттерн проектирования "Библиотека тестов" (Page Object Model / Page Factory), для повышения удобства поддержки и повторного использования кода.

Выполнив работу, вы будете уметь:

У 2.4.8 составлять отчет о выполнении тестирования ПОСодержание практических и лабораторных занятий ориентировано на освоение вида деятельности программы подготовки специалистов среднего звена по специальности и овладению

Материальное обеспечение:

1. Проект с существующими автотестами, требующий рефакторинга.
2. Среда разработки (IDE).

Задание:

1. Проанализировать существующий набор автотестов на предмет дублирования кода.
2. Спроектировать и реализовать библиотеку классов для взаимодействия с элементами пользовательского интерфейса.
3. Переработать существующие тесты для использования созданной библиотеки.

Порядок выполнения работы:

1. Откройте проект с набором простых автотестов. Найдите в них повторяющиеся блоки кода (например, поиск одного и того же элемента, последовательность действий).
2. Создайте новый пакет/папку pages или ui_library.
3. Для каждой страницы или крупного компонента приложения создайте отдельный класс (например, LoginPage, MainPage).
4. В каждом классе определите приватные поля для хранения локаторов элементов.
5. Создайте публичные методы, инкапсулирующие действия на странице. Например, вместо driver.findElement(By.id("login")).click() метод должен называться loginPage.clickLoginButton().
6. Перепишите существующие тесты. Вместо прямых обращений к драйверу они должны теперь использовать методы из созданных классов-объектов страниц.
7. Запустите переработанные тесты, чтобы убедиться, что они все еще работают корректно.

Форма представления результата:

Структура проекта в виде скриншота проводника файлов/проекта, показывающая созданные классы-объекты страниц и их методы. Фрагмент кода до и после рефакторинга.

Лабораторное занятие №70.
Запуск автотестов и сбор статистики. Оформление отчета

Цель: Научиться запускать наборы (test suite) автотестов, собирать метрики их выполнения и формировать итоговый отчет о тестировании для передачи заинтересованным сторонам.

Выполнив работу, вы будете уметь:

У 2.4.3 выполнять тестирование программного обеспечения вручную и автоматизировать процесс тестирования

Материальное обеспечение:

1. Проект с набором готовых автотестов.
2. Фреймворк для запуска тестов (например, pytest, JUnit, TestNG) с установленными плагинами для генерации отчетов (например, pytest-html, allure-pytest).

Задание:

1. Запустить полный набор (test suite) автотестов.
2. Собрать статистику о результатах выполнения.
3. Сформировать и проанализировать отчет о тестировании.

Порядок выполнения работы:

1. Убедитесь, что в проекте настроен фреймворк для запуска тестов и генерации отчетов.
2. Запустите выполнение всего набора тестов из корня проекта с помощью команды фреймворка (например, pytest --html=report.html или mvn test).
3. Дождитесь завершения выполнения. Фреймворк автоматически соберет статистику: общее количество тестов, количество успешных (Passed), упавших (Failed) и пропущенных (Skipped) тестов.
4. Найдите сгенерированный файл отчета (например, report.html в директории reports/). Откройте его в браузере.
5. Проанализируйте отчет:
6. Изучите общую статистику.
7. Просмотрите детали по каждому упавшему тесту: скриншоты, логи, стек вызовов (stack trace).
8. Оцените общее время выполнения набора тестов.
9. Сделайте вывод о текущем качестве продукта на основе результатов автоматизированного тестирования.

Форма представления результата:

Скриншот консоли с результатом запуска набора тестов и скриншот итогового отчета о тестировании (HTML-страницы) с открытой статистикой.

Лабораторное занятие №71.
Сборка и запуск тестов из консоли

Цель: Освоить процесс непрерывной интеграции (CI) путем автоматизации сборки проекта и запуска тестов из командной строки без использования графического интерфейса среды разработки (IDE).

Выполнив работу, вы будете уметь:

У 2.4.3 выполнять тестирование программного обеспечения вручную и автоматизировать процесс тестирования

Материальное обеспечение:

1. Проект с автотестами и настроенным файлом сборки (pom.xml, build.gradle, Makefile).
2. Командная строка (Terminal, PowerShell, cmd.exe).

Задание:

1. Очистить проект от предыдущих сборок и артефактов.
2. Собрать проект из исходного кода с помощью команды сборки.

3. Запустить автотесты из консоли.
4. Проанализировать вывод консоли на наличие ошибок.

Порядок выполнения работы:

1. Откройте терминал/командную строку и перейдите в корневую директорию проекта (где находится файл сборки).
2. Очистка: Выполните команду очистки проекта от старых артефактов. Например:

Для Maven: mvn clean

Для Gradle: gradle clean

Сборка: Выполните команду сборки проекта. Это скомпилирует исходный код и подготовит все необходимые файлы. Например:

Для Maven: mvn install

Для Gradle: gradle build

Запуск тестов: Выполните команду запуска тестов. Например:

Для Maven: mvn test

Для pytest (Python): pytest

3. Внимательно изучите вывод в консоли. Найдите строку "BUILD SUCCESS" или количество пройденных тестов. Если есть ошибки "BUILD FAILED" или упавшие тесты, проанализируйте текст ошибки выше по выводу, чтобы понять причину.
4. Убедитесь, что отчет о тестировании был успешно сгенерирован в указанной директории.

Форма представления результата:

Скриншот окна терминала/командной строки с последовательностью введенных команд (cd, mvn clean install test) и их успешным результатом вывода.

Лабораторное занятие №72.
Выполнение приемочного тестирования и оформление отчета

Цель: Научиться проводить приемочное тестирование (User Acceptance Testing - UAT) — финальный этап проверки продукта перед сдачей заказчику — на основе заранее согласованных критериев приемки и оформлять итоговый протокол испытаний.

Выполнив работу, вы будете уметь:

У 2.4.4 анализировать результаты тестирования и документировать найденные ошибки

Материальное обеспечение:

1. Тестируемая версия программного продукта (близкая к релизу).
2. Документ с критериями приемки (Acceptance Criteria) или пользовательскими историями (User Stories).

Задание:

1. Изучить критерии приемки для предоставленной функциональности.
2. Выполнить сценарии тестирования согласно этим критериям.
3. Оформить протокол приемочных испытаний с итоговым решением о приемке продукта.

Порядок выполнения работы:

1. Ознакомьтесь с документом, содержащим критерии приемки (Acceptance Criteria) для проверяемой части системы. Это могут быть формальные требования из ТЗ или неформальные описания из User Stories.
2. Подготовьте тестовую среду и получите доступ к тестируемой версии продукта.
3. Последовательно выполните каждый сценарий, описанный в критериях приемки. Действуйте строго по шагам. Фиксируйте результат каждого шага:

– Успех: Фактический результат полностью соответствует ожидаемому.

– Провал: Есть расхождение. Опишите его подробно и приложите доказательства (скриншот).

4. По завершении проверки всех критериев оцените общее состояние продукта:
– Если все критерии выполнены успешно — продукт принимается.
– Если хотя бы один критерий не выполнен — продукт не принимается и отправляется на доработку.
5. Оформите Протокол приемочных испытаний, в котором укажите версию продукта, дату, список проверенных критериев и итоговый вердикт ("Принято" / "Не принято").

Форма представления результата:
Заполненный шаблон Протокола приемочных испытаний, содержащий:
– Название проекта и версию сборки.
– ФИО проверяющего.
– Таблицу с перечислением критериев приемки, шагов проверки и фактических результатов.
– Итоговое заключение о приемке продукта.

Лабораторное занятие №73.
Тестирование пользовательской документации и оформление отчета

Цель: Систематизировать навыки проверки документации на соответствие критериям качества, полноту информации и корректность описания пользовательских сценариев.

Выполнив работу, вы будете уметь:
У 2.4.4 анализировать результаты тестирования и документировать найденные ошибки

Материальное обеспечение:
1. Комплект пользовательской документации (руководство пользователя, справка).
2. Доступ к тестируемому программному продукту.

Задание:
1. Провести проверку комплекта пользовательской документации по чек-листу качества.
2. Выявить не менее трех дефектов различного типа (ошибка в тексте, пропущенный шаг, несоответствие интерфейсу).
3. Оформить отчет о дефектах в документации по стандартному шаблону баг-репорта.

Порядок выполнения работы:
1. Ознакомьтесь с чек-листом качества технической документации (полнота, точность, однозначность, актуальность).
2. Возьмите руководство пользователя или инструкцию по работе с одной из функций ПО.
3. Последовательно выполните все действия, описанные в документации на реальном приложении. Сверяйте каждый шаг инструкции с тем, что происходит на экране.
4. В процессе проверки ищите следующие типы дефектов:
– Пропущенный шаг: В инструкции не хватает действия для достижения результата.
– Ошибка в описании: Текст инструкции не соответствует фактическому состоянию интерфейса (неверное название кнопки/поля).
– Фактическая ошибка: Приведенные данные неверны (например, неправильная формула расчета).
– Для каждого найденного дефекта создайте запись в отчете по аналогии с баг-репортом:
– ID: Уникальный идентификатор дефекта в документации (Doc-XX).
– Описание: Краткая суть проблемы ("Отсутствует описание шага X").
– Местонахождение: Точное место в документе (раздел, страница).
– Серьезность: Оценка влияния на пользователя (Блокирующая/Критическая/Тривиальная).
5. Сформируйте итоговый список дефектов в виде таблицы.

Форма представления результата:
Таблица со списком выявленных дефектов в документации со столбцами:
ID дефекта | Описание | Местонахождение | Серьезность | Шаги воспроизведения |Анализ.

Тема 3.5 Основы интеграционного и системного тестирования
Лабораторное занятие №74.
Разработка и запуск тестов пользовательского интерфейса

Цель: Освоить практические навыки проектирования, разработки и выполнения автоматизированных тестов для проверки графического пользовательского интерфейса (GUI) веб-приложений с использованием фреймворков (например, Selenium, Playwright).

Выполнив работу, вы будете уметь:

У 2.4.3 выполнять тестирование программного обеспечения вручную и автоматизировать процесс тестирования

Материальное обеспечение:

1. Среда разработки (IDE) с установленным языком программирования (Python, Java).
2. Библиотека для автоматизации браузера (Selenium WebDriver, Playwright) и соответствующий драйвер.
3. Доступ к тестируемому веб-приложению.

Задание:

1. Разработать автоматизированный тест, имитирующий полный пользовательский сценарий (например, регистрация нового пользователя).
2. Запустить тест и проанализировать результат его выполнения.

Порядок выполнения работы:

1. Настройка: Установите необходимые библиотеки и драйверы браузера. Создайте новый проект в вашей IDE.
2. Проектирование: Опишите шаги пользовательского сценария в текстовом виде. Например: "Открыть страницу регистрации -> Ввести логин -> Ввести email -> Ввести пароль -> Нажать кнопку 'Зарегистрироваться' -> Проверить появление сообщения об успешной регистрации".
3. Реализация: Напишите код теста, реализующий описанные шаги. Используйте методы для навигации, поиска элементов (по id, name, xpath), ввода текста и нажатия на кнопки. Добавьте проверки (assertions) для верификации ожидаемого результата (например, проверка текста на странице).
4. Выполнение: Запустите тест из вашей IDE или командной строки.
5. Анализ: Если тест завершился неудачно, проанализируйте отчет об ошибке и стек вызовов (stack trace) для локализации проблемы.

Форма представления результата:

1. Исходный код разработанного автотеста (файл .py или .java).
2. Скриншот окна браузера с результатом выполнения теста (успешная регистрация или сообщение об ошибке).
3. Краткий отчет о выполнении: описание сценария, результат (Passed/Failed), возможные проблемы.

Лабораторное занятие №75.
Настройка автоматического запуска API-тестов

Цель: Научиться настраивать и запускать автоматические тесты для программных интерфейсов (API) с использованием специализированных инструментов и систем непрерывной интеграции.

Выполнив работу, вы будете уметь:

У 2.4.3 выполнять тестирование программного обеспечения вручную и автоматизировать процесс тестирования

Материальное обеспечение:

1. Инструмент для тестирования API (например, коллекция Postman или скрипт на Python с библиотекой requests).
2. Доступ к документации и тестовой среде API.

Задание:

1. Разработать набор автотестов для проверки эндпоинтов API.
2. Настроить их автоматический запуск с помощью встроенных средств инструмента или системы сборки.

Порядок выполнения работы:

1. Разработка тестов: Создайте коллекцию запросов в Postman или напишите скрипт (например, на Python), который отправляет запросы к API и проверяет ответы. Тесты должны проверять как успешные сценарии (200 OK), так и негативные (400 Bad Request, 404 Not Found).
2. Автоматизация в Postman: Если используете Postman, настройте запуск коллекции через Newman (командная строка) или встроенный Collection Runner. Создайте Environment для хранения переменных (URL, токены).
3. Автоматизация в коде: Если используете скрипт, настройте его запуск через систему сборки (Maven, Gradle) или планировщик задач. Убедитесь, что результаты выполнения выводятся в консоль или сохраняются в файл.
4. Запуск: Выполните команду для автоматического запуска тестов. Проанализируйте отчет о прохождении.

Форма представления результата:

1. Скриншот коллекции Postman со списком запросов ИЛИ исходный код скрипта для тестирования API.
2. Скриншот консоли с результатом запуска тестов (количество пройденных/упавших тестов).
3. Файл отчета о прохождении тестов (если инструмент его генерирует).

Лабораторное занятие №76.
Разработка тестов для проверки подключения баз данных

Цель: Научиться проектировать и выполнять тесты для проверки корректности взаимодействия приложения с системой управления базами данных (СУБД), включая проверку подключения, выполнение запросов и обработку ошибок.

Выполнив работу, вы будете уметь:

У 2.4.1 анализировать требования к программному обеспечению и составлять планы тестирования

Материальное обеспечение:

1. Тестируемое приложение с конфигурационным файлом для подключения к БД.
2. Доступ к СУБД с известными учетными данными.
3. Инструмент для прямого подключения к БД (например, DBeaver) для проверки данных вручную.

Задание:

1. Настроить параметры подключения к базе данных в приложении.
2. Разработать и выполнить тесты для проверки успешного подключения и выполнения SQL-запроса.
3. Проверить поведение приложения при некорректных параметрах подключения.

Порядок выполнения работы:

1. Получите параметры подключения к СУБД (host, port, database, user, password) и убедитесь в их корректности через прямой клиент БД.

2. Откройте конфигурационный файл приложения (например, config.properties, application.yml) и укажите эти параметры. Сохраните изменения.
3. Запустите приложение. Проверьте логи на наличие успешного подключения к БД.
4. Разработайте простой скрипт или используйте функционал приложения для выполнения SQL-запроса (например, SELECT COUNT(*) FROM users;). Выполните его и убедитесь в получении результата.
5. Тест на сбой: Измените пароль в конфиге на неверный. Перезапустите приложение или выполните запрос. Убедитесь, что приложение корректно обрабатывает ошибку подключения (выводит сообщение об ошибке), а не "падает" с критическим сбоем.
6. Верните правильные параметры в конфигурационный файл.

Форма представления результата:

1. Скриншот конфигурационного файла с параметрами подключения к БД (с закрашенным паролем).
2. Скриншот вывода логов приложения со строкой об успешном подключении к БД и результатом выполнения SQL-запроса.
3. Скриншот вывода логов приложения при попытке подключения с неверными параметрами (демонстрация обработки ошибки).

Лабораторное занятие №77.
Разработка автотеста с заглушками и имитаторами (моками)

Цель: Освоить применение техники модульного тестирования с использованием тестовых дублёров (mock-объектов) для изоляции тестируемого модуля от его зависимостей.

Выполнив работу, вы будете уметь:

У 2.4.2 создавать тестовые сценарии и тест-кейсы для проверки функциональности и соответствия требованиям

Материальное обеспечение:

1. Среда разработки (IDE).
2. Библиотека для создания моков (unittest.mock, Mockito, Moq).
3. Исходный код тестируемого модуля и его интерфейса/зависимости.

Задание:

1. Изолировать модуль от внешней зависимости (например, сервис оплаты или база данных) с помощью мока.
2. Разработать автотест, который проверяет логику модуля без реального вызова внешней службы.
3. Проверить корректность взаимодействия модуля с зависимостью.

Порядок выполнения работы:

1. Определите внешнюю зависимость вашего модуля, которую нужно изолировать (например, класс EmailService, который отправляет письма).
2. Создайте мок-объект этой зависимости. Настройте его поведение: определите, какой ответ он должен возвращать при вызове определенных методов (например, метод send() всегда возвращает True).
3. Передайте созданный мок-объект в ваш тестируемый модуль вместо реальной зависимости.
4. Вызовите метод тестируемого модуля, который использует эту зависимость.
5. Проверка результата: Проверьте, что модуль отработал корректно на основе ответа от мока.
6. Проверка взаимодействия: Используйте возможности мока для проверки того, что метод зависимости был вызван нужное количество раз и с нужными аргументами. Это ключевое отличие моков от простых заглушек.

Форма представления результата:

1. Фрагмент кода теста с созданием мока и его настройкой (when(...).thenReturn(...)).

2. Фрагмент кода теста с проверкой взаимодействия (verify(...)).
3. Скриншот консоли с успешным результатом выполнения теста (OK или Passed).

Лабораторное занятие №78.
Настройка выбранной системы логирования с учетом ротации файлов

Цель: Научиться детально конфигурировать подсистему логирования приложения для обеспечения управляемости журналов событий, включая настройку уровней логирования, форматов вывода и политики ротации файлов журналов.

Выполнив работу, вы будете уметь:

У 2.4.7 использовать системы контроля дефектов ПО

Материальное обеспечение:

1. Проект приложения с уже подключенной библиотекой логирования (log4j2, logback, logging).
2. Текстовый редактор для редактирования конфигурационных файлов XML/JSON/YAML/INI.

Задание:

1. Найти и изучить текущий конфигурационный файл системы логирования проекта.
2. Настроить ротацию файлов журналов по размеру (например, не более 10 МБ) с сохранением до 5 архивных файлов.
3. Настроить разные уровни логирования для разных частей приложения (например, DEBUG для пакета dao и INFO для всего остального).
4. Изменить формат вывода логов в файл, включив в него имя потока (thread name) и полный стек вызовов (stack trace) ошибок.

Порядок выполнения работы:

1. Найдите в ресурсах проекта файл конфигурации логирования (log4j2.xml, logback.xml и т.д.).
2. Найдите раздел <Appenders>. Создайте или настройте FileAppender/RollingFileAppender.
3. Настройка ротации: Внутри аппендера настройте политику ротации:
4. Для Log4j2/Logback: Укажите <SizeBasedTriggeringPolicy size="10MB"/> или аналогичный параметр. Настройте <DefaultRolloverStrategy max="5"/> для ограничения количества архивов.
5. Для Python logging: Используйте класс RotatingFileHandler(maxBytes=10*1024*1024, backupCount=5).
6. Настройка уровней: Найдите раздел <Loggers>. Добавьте запись <Logger name="com.example.dao" level="debug" additivity="false"><AppenderRef ref="fileAppender"/></Logger> перед корневым логгером (<Root>), чтобы переопределить уровень только для этого пакета.
7. Настройка формата: Найдите <PatternLayout>/<encoder>. Измените шаблон (pattern) вывода. Например, добавьте %t (имя потока) и %xEx (полный стек вызовов) к существующему шаблону %d{ISO8601} [%t] %-5p %c{1}:%L - %m%n.
8. Сохраните изменения, перезапустите приложение и сгенерируйте несколько событий для проверки новой конфигурации.

Форма представления результата:

1. Фрагмент измененного конфигурационного файла системы логирования с выделенными новыми или измененными параметрами (ротация, уровни, формат).
2. Скриншот каталога с файлами логов после работы приложения, показывающий наличие архивных файлов (*.log.1, *.log.2).

Лабораторное занятие №79.

Анализ логов и подготовка отчета о результатах мониторинга

Цель: Научиться эффективно анализировать файлы журналов приложений для выявления аномалий, диагностики причин сбоев и подготовки сводного отчета о состоянии системы на основе данных мониторинга.

Выполнив работу, вы будете уметь:

У 2.4.4 анализировать результаты тестирования и документировать найденные ошибки

Материальное обеспечение:

1. Файлы журналов приложения за определенный период времени (несколько файлов из-за ротации).
2. Доступ к командной строке (Linux/macOS Terminal, Windows PowerShell) или продвинутому текстовому редактору (VS Code, Notepad++).

Задание:

1. Проанализировать предоставленные файлы журналов за период времени, когда в системе наблюдался сбой.
2. Найти все записи об ошибках и предупреждениях, связанные с инцидентом.
3. Восстановить хронологию событий по временным меткам из разных файлов логов.
4. Подготовить отчет о результатах анализа с выводами о причине сбоя.

Порядок выполнения работы:

1. Сбор данных: Определите временной интервал инцидента. Используйте команду ls -ltr или аналогичную для просмотра списка файлов логов и найдите те, которые соответствуют этому периоду.
2. Поиск ошибок: Используйте команду grep -i "error\|fail\|exception\|warn" для поиска всех записей об ошибках в файлах за нужный период. Используйте флаг -r для рекурсивного поиска в каталоге с логами.
3. Анализ стека вызовов: Найдите первую ошибку в хронологии. Изучите стек вызовов (stack trace) непосредственно под ней. Это укажет на класс и метод, где произошла ошибка, а также на вызвавшую ее причину (например, NullPointerException или ConnectionTimeout).
4. Восстановление хронологии: Просматривайте логи вверх от первой ошибки (tail -n +1000 file.log | grep -n "timestamp"), чтобы найти события, которые могли привести к сбою (например, "Database connection pool exhausted").
5. Подготовка отчета: Сформулируйте отчет:

– Время начала инцидента: По первой найденной ошибке.
– Описание проблемы: Краткая суть ошибки из стека вызовов.
– Возможная причина: Гипотеза на основе анализа предшествующих событий в логах.
– Рекомендации: Что нужно проверить разработчикам (например, "Проверить нагрузку на базу данных в указанное время").

Форма представления результата:

Текстовый документ-отчет о результатах анализа логов со следующими разделами:

– Временной интервал анализа.
– Список ключевых сообщений об ошибках с указанием времени и файла-источника.
– Описание восстановленной хронологии событий.
– Итоговый вывод о вероятной причине сбоя.