



**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение
высшего образования

«ИРКУТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ФГБОУ ВО «ИГУ»

Факультет бизнес-коммуникаций и информатики
Кафедра естественнонаучных дисциплин

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ

по дисциплине Б1.В.05 Тестирование программного обеспечения

направление подготовки 09.03.03 Прикладная информатика

направленность (профиль) Прикладная информатика (разработка программного обеспечения)

Одобрено
УМК факультета бизнес-коммуникаций
и информатики

Разработан в соответствии с ФГОС ВО

с учетом требований проф. стандарта

Председатель УМК

М.Г. Синчурина

ФИО, должность, ученая степень, звание

подпись, печать

Разработчики:

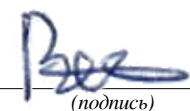

(подпись)

доцент

(занимаемая должность)

А.Г. Балахчи

(инициалы, фамилия)


(подпись)

старший преподаватель

(занимаемая должность)

А.М. Веснин

(инициалы, фамилия)

Цель фонда оценочных средств. Оценочные средства предназначены для контроля и оценки образовательных достижений обучающихся, освоивших программу учебной дисциплины «Б1.В.05 Тестирование программного обеспечения». Перечень видов оценочных средств соответствует рабочей программе дисциплины.

Фонд оценочных средств включает контрольные материалы для проведения текущего контроля (в следующих формах: практическое задание, тест, решение задач) и промежуточной аттестации в форме вопросов и заданий к зачету.

Структура и содержание заданий – задания разработаны в соответствии с рабочей программой дисциплины «Б1.В.05 Тестирование программного обеспечения».

1. Паспорт фонда оценочных средств

Компетенция	Индикаторы компетенций	Результаты обучения
ПК-14 РГ-1 Способен руководить процессами разработки компьютерного программного обеспечения	ПК-14 РГ-1 .1 Руководит разработкой программного кода	• ПК-14.1 РГ-1.1 У-3. Оценивает качество и эффективность программного кода.
	ПК-14 РГ-1 .2 Руководит проверкой работоспособности компьютерного программного обеспечения	• ПК-14.2 РГ-1.2 З-3. Знает методы и средства рефакторинга и оптимизации программного кода • ПК-14.2 РГ-1.2 У-1. Умеет распределять задачи на проверку работоспособности компьютерного программного обеспечения между исполнителями • ПК-14.2 РГ-1.2 У-2. Умеет оценивать результаты проверки работоспособности компьютерного программного обеспечения • ПК-14.2 РГ-1.2 У-3. Умеет принимать управленческие решения по результатам проверки работоспособности компьютерного программного обеспечения об исправлении ошибок, рефакторинге, оптимизации и инспекции кода
	ПК-14 РГ-1 .3 Руководит интеграцией программных модулей и компонентов компьютерного программного обеспечения	• ПК-14.3 РГ-1.3 У-3. Умеет принимать управленческие решения по результатам проверки работоспособности выпусков программного продукта (решение о выпуске/невыпуске версии, отправка задач на доработку, добавление новых задач, передача на тестирование)

Компетенция	Индикаторы компетенций	Результаты обучения
ПК-2 ППК-Р2 Способен проверять работоспособность и проводить рефакторинг кода программного обеспечения	ПК-2 ППК-Р2.1 Разрабатывает тестовые наборы данных для проверки работоспособности компьютерного программного обеспечения	• ПК-2 ППК-Р2.1 З-1. Знает методы создания и документирования контрольных примеров и тестовых наборов данных • ПК-2 ППК-Р2.1 З-2. Знает требования к структуре и форматам хранения тестовых наборов данных • ПК-2 ППК-Р2.1 З-3. Знает правила, алгоритмы и технологии создания тестовых наборов данных • ПК-2 ППК-Р2.1 У-1. Умеет разрабатывать и оформлять контрольные примеры для проверки работоспособности компьютерного программного обеспечения • ПК-2 ППК-Р2.1 У-2. Умеет готовить тестовые наборы данных в соответствии с выбранной методикой тестирования компьютерного программного обеспечения
	ПК-2 ППК-Р2.2 Проверяет работоспособность компьютерного программного обеспечения	• ПК-2 ППК-Р2.2 З-1. Знает методы и средства проверки работоспособности компьютерного программного обеспечения • ПК-2 ППК-Р2.2 З-2. Знает государственные стандарты испытания автоматизированных систем • ПК-2 ППК-Р2.2 З-3. Знает руководящие документы по стандартизации требований к документам автоматизированных систем • ПК-2 ППК-Р2.2 У-1. Умеет применять методы и средства проверки работоспособности компьютерного программного обеспечения • ПК-2 ППК-Р2.2 У-2. Умеет интерпретировать диагностические данные проверки работоспособности компьютерного программного обеспечения • ПК-2 ППК-Р2.2 У-3. Умеет анализировать значения полученных характеристик компьютерного программного обеспечения
	ПК-2 ППК-Р2.3 Исправляет дефекты программного кода, зафиксированных в базе данных дефектов	• ПК-2 ППК-Р2.3 З-1. Знает типичные ошибки, возникающие при разработке компьютерного программного обеспечения, методы их диагностики и исправления • ПК-2 ППК-Р2.3 З-2. Знает методы и приемы отладки программного кода • ПК-2 ППК-Р2.3 У-1. Умеет воспроизводить дефекты программного кода, зафиксированные в базе данных дефектов • ПК-2 ППК-Р2.3 У-2. Умеет выяснять причины возникновения дефектов программного кода • ПК-2 ППК-Р2.3 У-3. Умеет вносить изменений в программный код для устранения выявленных дефектов
	ПК-2 ППК-Р2.4 Выполняет рефакторинг и инспекцию программного кода (код-ревью)	• ПК-2 ППК-Р2.4 З-1. Знает методы и средства рефакторинга и инспекции программного кода • ПК-2 ППК-Р2.4 З-2. Знает нормативно-технические документы (стандарты и регламенты), регламентирующие требования к программному коду, порядок отражения изменений в системе управления версиями, порядок отражения результатов рефакторинга, оптимизации и инспекции в коллективной базе знаний • ПК-2 ППК-Р2.4 У-1. Умеет анализировать программный код на соответствие требованиям по читаемости и производительности • ПК-2 ППК-Р2.4 У-2. Умеет проводить инспекцию программного кода для поиска не обнаруженных на ранних стадиях разработки компьютерного программного обеспечения ошибок и критических мест • ПК-2 ППК-Р2.4 У-3. Умеет применять методы и средства рефакторинга и инспекции программного кода • ПК-2 ППК-Р2.4 У-4. Умеет публиковать результаты рефакторинга и инспекции в коллективной базе знаний

Компетенция	Индикаторы компетенций	Результаты обучения
ПК-6 ИП-2 Способен оптимизировать производительность программного обеспечения	ПК-6 ИП-2.1 Выполняет мониторинг производительности программного обеспечения	• ПК-6 ИП-2.1 З-1. Знает методы и средства мониторинга производительности компьютерного программного обеспечения • ПК-6 ИП-2.1 З-2. Знает метрики производительности программного обеспечения • ПК-6 ИП-2.1 З-3. Знает современные инструменты мониторинга производительности программного обеспечения • ПК-6 ИП-2.1 У-1. Умеет применять методы и средства мониторинга производительности компьютерного программного обеспечения • ПК-6 ИП-2.1 У-2. Умеет интерпретировать диагностические данные мониторинга производительности компьютерного программного обеспечения • ПК-6 ИП-2.1 У-3. Умеет определять "узкие места" программного обеспечения
	ПК-6 ИП-2.2 Выполняет оптимизацию программного кода	• ПК-6 ИП-2.2 З-1. Знает методы и средства оптимизации производительности компьютерного программного обеспечения • ПК-6 ИП-2.2 З-2. Знает современные инструменты оптимизации производительности программного обеспечения • ПК-6 ИП-2.2 У-1. Умеет оптимизировать программный код с использованием специализированных программных средств • ПК-6 ИП-2.2 У-2. Умеет вырабатывать варианты оптимизации производительности компьютерного программного обеспечения • ПК-6 ИП-2.2 У-3. Умеет проводить оценку и обоснование рекомендуемых решений по оптимизации производительности
ПК-8 ППК-Р7 Способен применять искусственный интеллект (ИИ) для генерации и отладки программного кода	ПК-8 ППК-Р7.1 Применяет ИИ-инструменты для генерации программного кода	• ПК-8 ППК-Р7.1 У-1. Умеет формулировать корректные текстовые запросы (промты) для генерации кода • ПК-8 ППК-Р7.1 У-2. Умеет интегрировать ИИ-инструменты в среду разработки
	ПК-8 ППК-Р7.2 Использует ИИ для анализа и отладки кода	• ПК-8 ППК-Р7.2 З-2. Знает форматы и инструменты для автоматизированного тестирования с ИИ • ПК-8 ППК-Р7.2 У-1. Умеет настраивать ИИ-инструменты для поиска уязвимостей • ПК-8 ППК-Р7.2 У-2. Умеет интерпретировать рекомендации ИИ по исправлению кода
	ПК-8 ППК-Р7.3 Оптимизирует код с помощью ИИ	• ПК-8 ППК-Р7.3 З-1. Знает методы ИИ-оптимизации • ПК-8 ППК-Р7.3 З-2. Знает критерии качества кода, применяемые ИИ-системами • ПК-8 ППК-Р7.3 У-1. Умеет использовать ИИ для рефакторинга • ПК-8 ППК-Р7.3 У-2. Умеет проверять корректность оптимизаций, предложенных ИИ
	ПК-8 ППК-Р7.4 Оценивает этические и профессиональные аспекты применения ИИ в разработке	• ПК-8 ППК-Р7.4 У-1. Умеет проверять код на соответствие стандартам после ИИ-генерации • ПК-8 ППК-Р7.4 У-2. Умеет документировать использование ИИ в разработке

2. Показатели и критерии оценивания компетенций, шкалы оценивания

2.1. Показатели и критерии оценивания компетенций

№ п/п	Раздел, тема	Код индикатора компетенции	Наименование ОС	
			ТК	ПА
1	Инструменты и автоматизация тестирования	ПК-2 ППК-Р2.1, ПК-2 ППК-Р2.2, ПК-2 ППК-Р2.3, ПК-2 ППК-Р2.4	Пз	Нет!

№ п/п	Раздел, тема	Код индикатора компетенции	Наименование ОС	
			ТК	ПА
2	Виды и уровни тестирования	ПК-2 ППК-Р2.2, ПК-14 РГ-1 .1, ПК-2 ППК-Р2.1, ПК-2 ППК-Р2.3, ПК-2 ППК-Р2.4, ПК-6 ИП-2.1, ПК-6 ИП-2.2, ПК-8 ППК-Р7.1, ПК-8 ППК-Р7.2	Тест, Пз	Тест
3	Тест-дизайн и управление дефектами	ПК-14 РГ-1 .1, ПК-14 РГ-1 .2, ПК-14 РГ-1 .3, ПК-2 ППК-Р2.1, ПК-2 ППК-Р2.2, ПК-2 ППК-Р2.3, ПК-2 ППК-Р2.4	Пз	Нет!
4	Инструменты и автоматизация тестирования	ПК-2 ППК-Р2.1, ПК-2 ППК-Р2.2, ПК-2 ППК-Р2.3, ПК-2 ППК-Р2.4, ПК-6 ИП-2.1, ПК-6 ИП-2.2, ПК-8 ППК-Р7.2, ПК-8 ППК-Р7.3, ПК-14 РГ-1 .2, ПК-14 РГ-1 .3	Тест, РЗ, Пз	Тест

2.2. Критерии оценивания результатов обучения для текущего контроля успеваемости и промежуточной аттестации

Оценочное средство	Критерии оценивания	Шкала оценивания
Практическое задание	Задание выполнено верно. Выбран оптимальный путь решения. Присутствует развернутое описание алгоритма решения	Отлично
	Задание выполнено верно. Допущены негрубые логические ошибки при описании алгоритма решения. Отсутствуют пояснения к решению задания	Хорошо
	Ход решения задания верный, но допущены ошибки приведшие к неправильному ответу	Удовлетворительно
	В работе получен неверный ответ, связанный с грубыми ошибками допущенными в ходе решения, либо решение отсутствует полностью	Неудовлетворительно
Тест	Студентом даны правильные ответы на 91-100% заданий	Отлично
	Студентом даны правильные ответы на 81-90% заданий	Хорошо
	Студентом даны правильные ответы на 71-80% заданий	Удовлетворительно
	Студентом даны правильные ответы менее чем на 70% заданий	Неудовлетворительно

Оценочное средство	Критерии оценивания	Шкала оценивания
Решение задач	Решение задачи выполнено верно. Выбран оптимальный путь решения. Присутствует развернутое описание алгоритма решения	Зачтено
	В задаче получен неверный ответ, связанный с грубыми ошибками допущенными в ходе решения, либо решение отсутствует полностью	Незачтено

2.3. Оценочные средства для текущего контроля (примеры)

2.3.1. Материалы для компьютерного тестирования обучающихся

Общие критерии оценивания

Процент правильных ответов	Оценка
91% – 100%	5 (отлично)
81% – 90%	4 (хорошо)
71% – 80%	3 (удовлетворительно)
Менее 70%	2 (неудовлетворительно)

Соответствие вопросов теста индикаторам формируемых и оцениваемых компетенций

№ вопроса в тесте	Код индикатора компетенции
1	ПК-2 ППК-Р2.4
2	ПК-2 ППК-Р2.2
3	ПК-2 ППК-Р2.3
4	ПК-8 ППК-Р7.3
5	ПК-2 ППК-Р2.1
6	ПК-2 ППК-Р2.2
7	ПК-14 РГ-1 .3
8	ПК-6 ИП-2.2
9	ПК-8 ППК-Р7.2
10	ПК-6 ИП-2.1
11	ПК-14 РГ-1 .2

Ключ ответов

№ вопроса в тесте	Номер ответа (или ответ, или соответствие)
1	d
2	b
3	a
4	a
5	b
6	d
7	b

№ вопроса в тесте	Номер ответа (или ответ, или соответствие)
8	c
9	d
10	c
11	d

Перечень тестовых вопросов

№ 1. Задание с единичным выбором. Выберите один правильный ответ.

Какое из перечисленных действий относится к рефакторингу, а не к исправлению дефекта?

- a. Добавление проверки входных данных
- b. Изменение условия ветвления для корректной работы
- c. Исправление деления на ноль
- d. Переименование переменных для улучшения читаемости без изменения логики

№ 2. Задание с единичным выбором. Выберите один правильный ответ.

Какое действие относится непосредственно к проверке работоспособности программного обеспечения?

- a. Оптимизация алгоритма
- b. Запуск программы с тестовыми данными и анализ фактического результата
- c. Рефакторинг кода
- d. Написание технического задания

№ 3. Задание с единичным выбором. Выберите один правильный ответ.

Какой порядок действий является наиболее корректным при исправлении дефекта, зафиксированного в системе баг-трекинга?

- a. Проанализировать баг → воспроизвести → исправить → проверить → закрыть
- b. Исправить код → закрыть баг → написать тест
- c. Исправить код без воспроизведения дефекта
- d. Закрыть баг → исправить код → проверить

№ 4. Задание с единичным выбором. Выберите один правильный ответ.

Какой результат свидетельствует об эффективном использовании ИИ для оптимизации кода?

- a. Снижено время выполнения алгоритма при сохранении корректности
- b. ИИ переписал код без объяснений
- c. Код стал короче, но работает медленнее
- d. Добавлены комментарии

№ 5. Задание с единичным выбором. Выберите один правильный ответ.

Какой набор входных данных наиболее корректно использовать для проверки функции, вычисляющей среднее значение массива чисел?

- a. Только массив положительных чисел
- b. Набор данных, включающий нормальные, граничные и некорректные значения
- c. Массив с одинаковыми значениями
- d. Пустой массив

№ 6. Задание с единичным выбором. Выберите один правильный ответ.

Какое действие относится непосредственно к проверке работоспособности программного обеспечения?

- a. Оптимизация алгоритма
- b. Рефакторинг кода
- c. Написание технического задания
- d. Запуск программы с тестовыми данными и анализ фактического результата

№ 7. Задание с единичным выбором. Выберите один правильный ответ.

Какое решение должен принять руководитель при интеграции нескольких программных модулей?

- a. Передать интеграцию одному разработчику без проверки
- b. Назначить интеграционное тестирование и определить порядок объединения модулей
- c. Отложить тестирование до релиза
- d. Проверить каждый модуль только отдельно

№ 8. Задание с единичным выбором. Выберите один правильный ответ.

Какое действие является примером оптимизации программного кода?

- a. Добавление логирования
- b. Добавление комментариев
- c. Замена линейного поиска на хеш-таблицу
- d. Переименование функций

№ 9. Задание с единичным выбором. Выберите один правильный ответ.

В каком случае использование ИИ наиболее оправдано для анализа и отладки кода?

- a. Для запуска программы
- b. Для форматирования кода
- c. Для поиска синтаксических ошибок компилятором
- d. Для анализа стектрейса и предложения возможной причины ошибки

№ 10. Задание с единичным выбором. Выберите один правильный ответ.

Какой показатель наиболее непосредственно используется при мониторинге производительности программного обеспечения?

- a. Количество строк кода
- b. Стил оформления кода
- c. Время отклика и использование ресурсов
- d. Количество комментариев

№ 11. Задание с единичным выбором. Выберите один правильный ответ.

Какое действие относится к руководству проверкой работоспособности программного обеспечения, а не к индивидуальному тестированию?

- a. Исправление одного дефекта
- b. Запуск модульного теста
- c. Написание одного тест-кейса
- d. Планирование этапов тестирования и распределение задач в команде

2.3.2. Практические задания для оценки компетенции «ПК-14 РГ-1 .2»

№ 1. Подготовка рабочего места.

Домашняя работа

Тема: Подготовка рабочего места для тестирования программного обеспечения

Цель работы

Освоить базовые навыки настройки рабочего окружения тестировщика, необходимые для выполнения ручного и автоматизированного тестирования программного обеспечения, а также сформировать понимание воспроизводимости тестового окружения.

Исходные условия

Тестирование выполняется на локальном компьютере обучающегося с использованием открытого программного обеспечения. Допускается использование операционной системы Linux или Windows (с поддержкой виртуализации/контейнеризации).

Задание

1. Настройка операционной среды

1. Установить и настроить операционную систему:
 - Linux (рекомендуется Ubuntu или Debian) **или**
 - Windows с поддержкой командной строки.
2. Проверить работу терминала и базовых системных утилит.
3. Настроить структуру рабочих каталогов для проектов тестирования.

2. Установка и настройка инструментов разработки

1. Установить систему контроля версий **Git**.
2. Создать локальный репозиторий для учебных заданий по тестированию.
3. Настроить пользовательские параметры Git (имя, e-mail).

3. Подготовка среды для автоматизированного тестирования

1. Установить интерпретатор **Python** (актуальной стабильной версии).
2. Создать виртуальное окружение Python.
3. Установить фреймворк **PyTest** в виртуальное окружение.
4. Проверить корректность установки путём запуска тестового примера.

4. Инструменты для тестирования веб-приложений и API

1. Установить современный веб-браузер.
2. Изучить и активировать инструменты разработчика браузера.
3. Установить инструмент для тестирования API (например, Postman или аналог).
4. Выполнить тестовый запрос к открытому API.

5. Контейнеризация тестового окружения (ознакомительно)

1. Установить **Docker**.
 2. Проверить работоспособность Docker.
 3. Запустить тестовый контейнер.
 4. Кратко описать преимущества контейнеризации для тестирования.
-

6. Документирование результата

Подготовить отчёт, включающий:

- перечень установленного ПО;
 - краткое описание этапов настройки;
 - скриншоты или вывод команд, подтверждающие успешную установку;
 - выводы о готовности рабочего места к тестированию ПО.
-

Формат сдачи

- отчёт в формате PDF / DOCX / Markdown;
 - структура отчёта должна соответствовать этапам задания;
 - допускается размещение материалов в Git-репозитории.
-

Критерии оценки

Критерий	Баллы
Корректность установки и настройки окружения	
Работоспособность инструментов	
Полнота и аккуратность отчёта	
Понимание назначения инструментов	

Ожидаемый результат

В результате выполнения домашней работы обучающийся:

- подготовит рабочее место тестировщика;
- освоит базовые инструменты тестирования ПО;
- поймёт роль окружения в качестве и воспроизводимости тестирования;
- будет готов к выполнению практических и лабораторных работ по дисциплине.

№ 2. Кейс: "Проектирование и реализация тестирования модуля сервиса *EnergyMonitor*".

КЕЙС Проектирование и реализация тестирования модуля сервиса EnergyMonitor (Эн+ Диджитал)

Цель работы

Закрепить навыки применения видов и уровней тестирования программного обеспечения в рамках учебного проекта, а также сформировать практические умения разработки и сопровождения автоматизированных тестов с использованием фреймворка PyTest.

Описание кейса

В рамках учебного проекта рассматривается программный модуль сервиса **EnergyMonitor**, предназначенного для анализа энергопотребления промышленных объектов компании **Эн+ Диджитал**. Вам необходимо спроектировать и реализовать систему тестирования данного модуля, обеспечивающую проверку корректности бизнес-логики и устойчивости работы функций.

Вы выступаете в роли **инженера-разработчика**, отвечающего за качество программного кода.

Задание

Часть 1. Анализ и проектирование тестирования

1. Кратко опишите функциональное назначение тестируемого модуля.
 2. Определите:
 - применимые **уровни тестирования**;
 - соответствующие **виды тестирования**.
 3. Сформируйте перечень тестируемых сценариев, включая:
 - корректные входные данные;
 - граничные случаи;
 - потенциально ошибочные ситуации.
 4. Опишите риски для системы при отсутствии или неполном тестировании.
-

Часть 2. Реализация автоматизированных тестов

5. Создайте структуру проекта для тестирования (исходный код и каталог с тестами).
6. Реализуйте:
 - модульные тесты для отдельных функций;
 - не менее одного интеграционного теста для проверки взаимодействия функций.
7. Используйте возможности PyTest:
 - assert-проверки;
 - параметризацию тестов;
 - осмысленные имена тестовых функций.

Часть 3. Анализ и оформление результатов

8. Запустите тесты и зафиксируйте результаты выполнения.

9. Проанализируйте:

- какие дефекты выявляются тестами;
- какие сценарии требуют дополнительного покрытия.

10. Сформулируйте вывод о готовности модуля к использованию и дальнейшему развитию.

Требования к оформлению и сдаче

- язык программирования: **Python**;
 - фреймворк тестирования: **PyTest**;
 - форма сдачи:
 - Git-репозиторий **или**
 - архив с проектом и отчётом;
 - отчёт оформляется в формате PDF / DOCX / Markdown и включает:
 - описание проекта;
 - обоснование выбора видов и уровней тестирования;
 - краткие выводы.
-

Критерии оценки

- корректность применения понятий «виды тестирования» и «уровни тестирования»;
- полнота и качество тестовых сценариев;
- работоспособность и читаемость тестового кода;
- использование возможностей PyTest;
- аргументированность аналитической части.

2.3.3. Практические задания для оценки компетенции «ПК-14 РГ-1 .3»

№ 3. Подготовка рабочего места.

Домашняя работа

Тема: Подготовка рабочего места для тестирования программного обеспечения

Цель работы

Освоить базовые навыки настройки рабочего окружения тестировщика, необходимые для выполнения ручного и автоматизированного тестирования программного обеспечения, а также сформировать понимание воспроизводимости тестового окружения.

Исходные условия

Тестирование выполняется на локальном компьютере обучающегося с использованием открытого программного обеспечения. Допускается использование операционной системы Linux или Windows (с поддержкой виртуализации/контейнеризации).

Задание

1. Настройка операционной среды

1. Установить и настроить операционную систему:
—
Linux (рекомендуется Ubuntu или Debian) **или**
— Windows с поддержкой командной строки.
 2. Проверить работу терминала и базовых системных утилит.
 3. Настроить структуру рабочих каталогов для проектов тестирования.
-

2. Установка и настройка инструментов разработки

1. Установить систему контроля версий **Git**.
 2. Создать локальный репозиторий для учебных заданий по тестированию.
 3. Настроить пользовательские параметры Git (имя, e-mail).
-

3. Подготовка среды для автоматизированного тестирования

1. Установить интерпретатор **Python** (актуальной стабильной версии).
 2. Создать виртуальное окружение Python.
 3. Установить фреймворк **PyTest** в виртуальное окружение.
 4. Проверить корректность установки путём запуска тестового примера.
-

4. Инструменты для тестирования веб-приложений и API

1. Установить современный веб-браузер.
 2. Изучить и активировать инструменты разработчика браузера.
 3. Установить инструмент для тестирования API (например, Postman или аналог).
 4. Выполнить тестовый запрос к открытому API.
-

5. Контейнеризация тестового окружения (ознакомительно)

1. Установить **Docker**.
 2. Проверить работоспособность Docker.
 3. Запустить тестовый контейнер.
 4. Кратко описать преимущества контейнеризации для тестирования.
-

6. Документирование результата

Подготовить отчёт, включающий:

- перечень установленного ПО;
- краткое описание этапов настройки;
- скриншоты или вывод команд, подтверждающие успешную установку;
- выводы о готовности рабочего места к тестированию ПО.

Формат сдачи

- отчёт в формате PDF / DOCX / Markdown;
 - структура отчёта должна соответствовать этапам задания;
 - допускается размещение материалов в Git-репозитории.
-

Критерии оценки

Критерий	Баллы
Корректность установки и настройки окружения	
Работоспособность инструментов	
Полнота и аккуратность отчёта	
Понимание назначения инструментов	

Ожидаемый результат

В результате выполнения домашней работы обучающийся:

- подготовит рабочее место тестировщика;
- освоит базовые инструменты тестирования ПО;
- поймёт роль окружения в качестве и воспроизводимости тестирования;
- будет готов к выполнению практических и лабораторных работ по дисциплине.

№ 4. Кейс: "Проектирование и реализация тестирования модуля сервиса EnergyMonitor".

КЕЙС Проектирование и реализация тестирования модуля сервиса EnergyMonitor (Эн+ Диджитал)

Цель работы

Закрепить навыки применения видов и уровней тестирования программного обеспечения в рамках учебного проекта, а также сформировать практические умения разработки и сопровождения автоматизированных тестов с использованием фреймворка PyTest.

Описание кейса

В рамках учебного проекта рассматривается программный модуль сервиса **EnergyMonitor**, предназначенного для анализа энергопотребления промышленных объектов компании **Эн+ Диджитал**. Вам необходимо спроектировать и реализовать систему тестирования данного модуля, обеспечивающую проверку корректности бизнес-логики и устойчивости работы функций.

Вы выступаете в роли **инженера-разработчика**, отвечающего за качество программного кода.

Задание

Часть 1. Анализ и проектирование тестирования

1. Кратко опишите функциональное назначение тестируемого модуля.
 2. Определите:
 - применимые **уровни тестирования**;
 - соответствующие **виды тестирования**.
 3. Сформируйте перечень тестируемых сценариев, включая:
 - корректные входные данные;
 - граничные случаи;
 - потенциально ошибочные ситуации.
 4. Опишите риски для системы при отсутствии или неполном тестировании.
-

Часть 2. Реализация автоматизированных тестов

5. Создайте структуру проекта для тестирования (исходный код и каталог с тестами).
 6. Реализуйте:
 - модульные тесты для отдельных функций;
 - не менее одного интеграционного теста для проверки взаимодействия функций.
 7. Используйте возможности PyTest:
 - assert-проверки;
 - параметризацию тестов;
 - осмысленные имена тестовых функций.
-

Часть 3. Анализ и оформление результатов

8. Запустите тесты и зафиксируйте результаты выполнения.
 9. Проанализируйте:
 - какие дефекты выявляются тестами;
 - какие сценарии требуют дополнительного покрытия.
 10. Сформулируйте вывод о готовности модуля к использованию и дальнейшему развитию.
-

Требования к оформлению и сдаче

- язык программирования: **Python**;
 - фреймворк тестирования: **PyTest**;
 - форма сдачи:
 -
 - Git-репозиторий **или**
 - архив с проектом и отчётом;
 - отчёт оформляется в формате PDF / DOCX / Markdown и включает:
 -
 - описание проекта;
 - обоснование выбора видов и уровней тестирования;
 - краткие выводы.
-

Критерии оценки

- корректность применения понятий «виды тестирования» и «уровни тестирования»;
- полнота и качество тестовых сценариев;
- работоспособность и читаемость тестового кода;
- использование возможностей PyTest;
- аргументированность аналитической части.

2.3.4. Практические задания для оценки компетенции «ПК-2 ППК-Р2.1»

№ 5. Подготовка рабочего места.

Домашняя работа

Тема: Подготовка рабочего места для тестирования программного обеспечения

Цель работы

Освоить базовые навыки настройки рабочего окружения тестировщика, необходимые для выполнения ручного и автоматизированного тестирования программного обеспечения, а также сформировать понимание воспроизводимости тестового окружения.

Исходные условия

Тестирование выполняется на локальном компьютере обучающегося с использованием открытого программного обеспечения. Допускается использование операционной системы Linux или Windows (с поддержкой виртуализации/контейнеризации).

Задание

1. Настройка операционной среды

1. Установить и настроить операционную систему:
 -
 - Linux (рекомендуется Ubuntu или Debian) **или**
 - Windows с поддержкой командной строки.
2. Проверить работу терминала и базовых системных утилит.

3. Настроить структуру рабочих каталогов для проектов тестирования.

2. Установка и настройка инструментов разработки

1. Установить систему контроля версий **Git**.
2. Создать локальный репозиторий для учебных заданий по тестированию.
3. Настроить пользовательские параметры Git (имя, e-mail).

3. Подготовка среды для автоматизированного тестирования

1. Установить интерпретатор **Python** (актуальной стабильной версии).
2. Создать виртуальное окружение Python.
3. Установить фреймворк **PyTest** в виртуальное окружение.
4. Проверить корректность установки путём запуска тестового примера.

4. Инструменты для тестирования веб-приложений и API

1. Установить современный веб-браузер.
2. Изучить и активировать инструменты разработчика браузера.
3. Установить инструмент для тестирования API (например, Postman или аналог).
4. Выполнить тестовый запрос к открытому API.

5. Контейнеризация тестового окружения (ознакомительно)

1. Установить **Docker**.
2. Проверить работоспособность Docker.
3. Запустить тестовый контейнер.
4. Кратко описать преимущества контейнеризации для тестирования.

6. Документирование результата

Подготовить отчёт, включающий:

- перечень установленного ПО;
- краткое описание этапов настройки;
- скриншоты или вывод команд, подтверждающие успешную установку;
- выводы о готовности рабочего места к тестированию ПО.

Формат сдачи

- отчёт в формате PDF / DOCX / Markdown;
 - структура отчёта должна соответствовать этапам задания;
 - допускается размещение материалов в Git-репозитории.
-

Критерии оценки

Критерий	Баллы
Корректность установки и настройки окружения	
Работоспособность инструментов	
Полнота и аккуратность отчёта	
Понимание назначения инструментов	

Ожидаемый результат

В результате выполнения домашней работы обучающийся:

- подготовит рабочее место тестировщика;
- освоит базовые инструменты тестирования ПО;
- поймёт роль окружения в качестве и воспроизводимости тестирования;
- будет готов к выполнению практических и лабораторных работ по дисциплине.

№ 6. Кейс: "Проектирование и реализация тестирования модуля сервиса EnergyMonitor".

КЕЙС Проектирование и реализация тестирования модуля сервиса EnergyMonitor (Эн+ Диджитал)

Цель работы

Закрепить навыки применения видов и уровней тестирования программного обеспечения в рамках учебного проекта, а также сформировать практические умения разработки и сопровождения автоматизированных тестов с использованием фреймворка PyTest.

Описание кейса

В рамках учебного проекта рассматривается программный модуль сервиса **EnergyMonitor**, предназначенного для анализа энергопотребления промышленных объектов компании **Эн+ Диджитал**. Вам необходимо спроектировать и реализовать систему тестирования данного модуля, обеспечивающую проверку корректности бизнес-логики и устойчивости работы функций.

Вы выступаете в роли **инженера-разработчика**, отвечающего за качество программного кода.

Задание

Часть 1. Анализ и проектирование тестирования

1. Кратко опишите функциональное назначение тестируемого модуля.
2. Определите:
 - применимые **уровни тестирования**;
 - соответствующие **виды тестирования**.

3. Сформируйте перечень тестируемых сценариев, включая:
 - корректные входные данные;
 - граничные случаи;
 - потенциально ошибочные ситуации.
 4. Опишите риски для системы при отсутствии или неполном тестировании.
-

Часть 2. Реализация автоматизированных тестов

5. Создайте структуру проекта для тестирования (исходный код и каталог с тестами).
 6. Реализуйте:
 - модульные тесты для отдельных функций;
 - не менее одного интеграционного теста для проверки взаимодействия функций.
 7. Используйте возможности PyTest:
 - assert-проверки;
 - параметризацию тестов;
 - осмысленные имена тестовых функций.
-

Часть 3. Анализ и оформление результатов

8. Запустите тесты и зафиксируйте результаты выполнения.
 9. Проанализируйте:
 - какие дефекты выявляются тестами;
 - какие сценарии требуют дополнительного покрытия.
 10. Сформулируйте вывод о готовности модуля к использованию и дальнейшему развитию.
-

Требования к оформлению и сдаче

- язык программирования: **Python**;
- фреймворк тестирования: **PyTest**;
- форма сдачи:
 - Git-репозиторий **или**
 - архив с проектом и отчётом;
- отчёт оформляется в формате PDF / DOCX / Markdown и включает:
 - описание проекта;
 - обоснование выбора видов и уровней тестирования;

Критерии оценки

- корректность применения понятий «виды тестирования» и «уровни тестирования»;
- полнота и качество тестовых сценариев;
- работоспособность и читаемость тестового кода;
- использование возможностей PyTest;
- аргументированность аналитической части.

№ 7. Кейс. Проектирование тест-дизайна и управление дефектами.

Домашняя работа по проекту

Тема

КЕЙС. Проектирование тест-дизайна и управление дефектами для программного модуля (Эн+ Диджитал)

Цель домашней работы

Закрепить практические навыки применения техник тест-дизайна, разработки автоматизированных тестов и оформления дефектов в системе управления задачами (Jira) при тестировании программного обеспечения, используемого в промышленной ИТ-компании.

Описание проекта

В рамках учебного проекта вы работаете с программным модулем сервиса **EnergyMonitor**, предназначенного для анализа и обработки данных об энергопотреблении. Модуль реализован на языке Python и содержит как корректные участки логики, так и потенциальные дефекты.

Вы выполняете роль **инженера-разработчика / QA-инженера** компании **Эн+ Диджитал**, отвечающего за качество программного продукта.

Задание

Часть 1. Тест-дизайн

1. На основе исходного кода модуля:
 - проанализируйте его функциональность;
 - определите ключевые точки риска.
2. Разработайте **чек-лист или набор тест-кейсов**, используя техники тест-дизайна:
 - эквивалентное разбиение;
 - анализ граничных значений;
 - негативное тестирование.
3. Для каждого пункта чек-листа / тест-кейса укажите:
 - тестируемую функцию;
 - входные данные;
 - ожидаемый результат.

Часть 2. Реализация автоматизированных тестов

4. Реализуйте автоматизированные тесты на **Python** с использованием **PyTest**:

- модульные тесты для отдельных функций;
- не менее одного интеграционного теста.

5. В тестах обязательно:

- использовать `assert`;
- применить параметризацию;
- покрыть позитивные и негативные сценарии.

Часть 3. Управление дефектами

6. По результатам выполнения тестов:

- выявите дефекты в программном коде;
- для каждого дефекта оформите **баг-репорт**.

7. Каждый баг-репорт должен содержать:

- заголовок (Summary);
- шаги воспроизведения (Steps to Reproduce);
- ожидаемый и фактический результат;
- окружение;
- приоритет и критичность.

8. Оформите баг-репорты в **Jira** (или в виде макета баг-репорта, если доступ к Jira отсутствует).

Часть 4. Анализ и выводы

9. Кратко опишите:

- какие дефекты были выявлены с помощью тест-дизайна;
- какие дефекты удалось обнаружить только при автоматизированном тестировании;
- почему корректное оформление баг-репортов важно для командной разработки.

Требования к оформлению и сдаче

- язык программирования: **Python**;
- фреймворк тестирования: **PyTest**;
- формат сдачи:
 - Git-репозиторий с кодом и тестами **или**
 - архив с проектом;

— отчёт (1–2 страницы) в формате PDF / DOCX / Markdown, включающий:

- описание тест-дизайна;
- примеры баг-репортов;
- выводы.

Критерии оценки

- корректность применения техник тест-дизайна;
- полнота и качество тестового покрытия;
- корректность и читаемость автоматизированных тестов;
- качество оформления баг-репортов;
- аргументированность выводов.

2.3.5. Практические задания для оценки компетенции «ПК-2 ППК-Р2.2»

№ 8. Подготовка рабочего места.

Домашняя работа

Тема: Подготовка рабочего места для тестирования программного обеспечения

Цель работы

Освоить базовые навыки настройки рабочего окружения тестировщика, необходимые для выполнения ручного и автоматизированного тестирования программного обеспечения, а также сформировать понимание воспроизводимости тестового окружения.

Исходные условия

Тестирование выполняется на локальном компьютере обучающегося с использованием открытого программного обеспечения. Допускается использование операционной системы Linux или Windows (с поддержкой виртуализации/контейнеризации).

Задание

1. Настройка операционной среды

1. Установить и настроить операционную систему:
 - Linux (рекомендуется Ubuntu или Debian) **или**
 - Windows с поддержкой командной строки.
2. Проверить работу терминала и базовых системных утилит.
3. Настроить структуру рабочих каталогов для проектов тестирования.

2. Установка и настройка инструментов разработки

1. Установить систему контроля версий **Git**.
2. Создать локальный репозиторий для учебных заданий по тестированию.
3. Настроить пользовательские параметры Git (имя, e-mail).

3. Подготовка среды для автоматизированного тестирования

1. Установить интерпретатор **Python** (актуальной стабильной версии).
 2. Создать виртуальное окружение Python.
 3. Установить фреймворк **PyTest** в виртуальное окружение.
 4. Проверить корректность установки путём запуска тестового примера.
-

4. Инструменты для тестирования веб-приложений и API

1. Установить современный веб-браузер.
 2. Изучить и активировать инструменты разработчика браузера.
 3. Установить инструмент для тестирования API (например, Postman или аналог).
 4. Выполнить тестовый запрос к открытому API.
-

5. Контейнеризация тестового окружения (ознакомительно)

1. Установить **Docker**.
 2. Проверить работоспособность Docker.
 3. Запустить тестовый контейнер.
 4. Кратко описать преимущества контейнеризации для тестирования.
-

6. Документирование результата

Подготовить отчёт, включающий:

- перечень установленного ПО;
 - краткое описание этапов настройки;
 - скриншоты или вывод команд, подтверждающие успешную установку;
 - выводы о готовности рабочего места к тестированию ПО.
-

Формат сдачи

- отчёт в формате PDF / DOCX / Markdown;
 - структура отчёта должна соответствовать этапам задания;
 - допускается размещение материалов в Git-репозитории.
-

Критерии оценки

Критерий	Баллы
Корректность установки и настройки окружения	
Работоспособность инструментов	
Полнота и аккуратность отчёта	
Понимание назначения инструментов	

Ожидаемый результат

В результате выполнения домашней работы обучающийся:

- подготовит рабочее место тестировщика;
- освоит базовые инструменты тестирования ПО;
- поймёт роль окружения в качестве и воспроизводимости тестирования;
- будет готов к выполнению практических и лабораторных работ по дисциплине.

№ 9. Кейс: "Проектирование и реализация тестирования модуля сервиса EnergyMonitor".

КЕЙС Проектирование и реализация тестирования модуля сервиса EnergyMonitor (Эн+ Диджитал)

Цель работы

Закрепить навыки применения видов и уровней тестирования программного обеспечения в рамках учебного проекта, а также сформировать практические умения разработки и сопровождения автоматизированных тестов с использованием фреймворка PyTest.

Описание кейса

В рамках учебного проекта рассматривается программный модуль сервиса **EnergyMonitor**, предназначенного для анализа энергопотребления промышленных объектов компании **Эн+ Диджитал**. Вам необходимо спроектировать и реализовать систему тестирования данного модуля, обеспечивающую проверку корректности бизнес-логики и устойчивости работы функций.

Вы выступаете в роли **инженера-разработчика**, отвечающего за качество программного кода.

Задание

Часть 1. Анализ и проектирование тестирования

1. Кратко опишите функциональное назначение тестируемого модуля.
2. Определите:
 - применимые **уровни тестирования**;
 - соответствующие **виды тестирования**.
3. Сформируйте перечень тестируемых сценариев, включая:
 - корректные входные данные;
 - граничные случаи;
 - потенциально ошибочные ситуации.
4. Опишите риски для системы при отсутствии или неполном тестировании.

Часть 2. Реализация автоматизированных тестов

5. Создайте структуру проекта для тестирования (исходный код и каталог с тестами).

6. Реализуйте:

- модульные тесты для отдельных функций;
- не менее одного интеграционного теста для проверки взаимодействия функций.

7. Используйте возможности PyTest:

- assert-проверки;
- параметризацию тестов;
- осмысленные имена тестовых функций.

Часть 3. Анализ и оформление результатов

8. Запустите тесты и зафиксируйте результаты выполнения.

9. Проанализируйте:

- какие дефекты выявляются тестами;
- какие сценарии требуют дополнительного покрытия.

10. Сформулируйте вывод о готовности модуля к использованию и дальнейшему развитию.

Требования к оформлению и сдаче

- язык программирования: **Python**;
- фреймворк тестирования: **PyTest**;
- форма сдачи:
 - Git-репозиторий **или**
 - архив с проектом и отчётом;
- отчёт оформляется в формате PDF / DOCX / Markdown и включает:
 - описание проекта;
 - обоснование выбора видов и уровней тестирования;
 - краткие выводы.

Критерии оценки

- корректность применения понятий «виды тестирования» и «уровни тестирования»;
- полнота и качество тестовых сценариев;
- работоспособность и читаемость тестового кода;

- использование возможностей PyTest;
- аргументированность аналитической части.

№ 10. Кейс. Проектирование тест-дизайна и управление дефектами.

Домашняя работа по проекту

Тема

КЕЙС. Проектирование тест-дизайна и управление дефектами для программного модуля (Эн+ Диджитал)

Цель домашней работы

Закрепить практические навыки применения техник тест-дизайна, разработки автоматизированных тестов и оформления дефектов в системе управления задачами (Jira) при тестировании программного обеспечения, используемого в промышленной ИТ-компании.

Описание проекта

В рамках учебного проекта вы работаете с программным модулем сервиса **EnergyMonitor**, предназначенного для анализа и обработки данных об энергопотреблении. Модуль реализован на языке Python и содержит как корректные участки логики, так и потенциальные дефекты.

Вы выполняете роль **инженера-разработчика / QA-инженера** компании **Эн+ Диджитал**, отвечающего за качество программного продукта.

Задание

Часть 1. Тест-дизайн

1. На основе исходного кода модуля:
 - проанализируйте его функциональность;
 - определите ключевые точки риска.
2. Разработайте **чек-лист или набор тест-кейсов**, используя техники тест-дизайна:
 - эквивалентное разбиение;
 - анализ граничных значений;
 - негативное тестирование.
3. Для каждого пункта чек-листа / тест-кейса укажите:
 - тестируемую функцию;
 - входные данные;
 - ожидаемый результат.

Часть 2. Реализация автоматизированных тестов

4. Реализуйте автоматизированные тесты на **Python с использованием PyTest**:
 - модульные тесты для отдельных функций;
 - не менее одного интеграционного теста.
5. В тестах обязательно:

-
- использовать `assert`;
- применить параметризацию;
- покрыть позитивные и негативные сценарии.

Часть 3. Управление дефектами

6. По результатам выполнения тестов:

-
- выявите дефекты в программном коде;
- для каждого дефекта оформите **баг-репорт**.

7. Каждый баг-репорт должен содержать:

-
- заголовок (Summary);
- шаги воспроизведения (Steps to Reproduce);
- ожидаемый и фактический результат;
- окружение;
- приоритет и критичность.

8. Оформите баг-репорты в **Jira** (или в виде макета баг-репорта, если доступ к Jira отсутствует).

Часть 4. Анализ и выводы

9. Кратко опишите:

-
- какие дефекты были выявлены с помощью тест-дизайна;
- какие дефекты удалось обнаружить только при автоматизированном тестировании;
- почему корректное оформление баг-репортов важно для командной разработки.

Требования к оформлению и сдаче

- язык программирования: **Python**;
- фреймворк тестирования: **PyTest**;
- формат сдачи:
 -
 - Git-репозиторий с кодом и тестами **или**
 - архив с проектом;
- отчёт (1–2 страницы) в формате PDF / DOCX / Markdown, включающий:
 -
 - описание тест-дизайна;
 - примеры баг-репортов;
 - выводы.

Критерии оценки

- корректность применения техник тест-дизайна;

- полнота и качество тестового покрытия;
- корректность и читаемость автоматизированных тестов;
- качество оформления баг-репортов;
- аргументированность выводов.

2.3.6. Практические задания для оценки компетенции «ПК-2 ППК-Р2.3»

№ 11. Подготовка рабочего места.

Домашняя работа

Тема: Подготовка рабочего места для тестирования программного обеспечения

Цель работы

Освоить базовые навыки настройки рабочего окружения тестировщика, необходимые для выполнения ручного и автоматизированного тестирования программного обеспечения, а также сформировать понимание воспроизводимости тестового окружения.

Исходные условия

Тестирование выполняется на локальном компьютере обучающегося с использованием открытого программного обеспечения. Допускается использование операционной системы Linux или Windows (с поддержкой виртуализации/контейнеризации).

Задание

1. Настройка операционной среды

1. Установить и настроить операционную систему:
 - Linux (рекомендуется Ubuntu или Debian) **или**
 - Windows с поддержкой командной строки.
2. Проверить работу терминала и базовых системных утилит.
3. Настроить структуру рабочих каталогов для проектов тестирования.

2. Установка и настройка инструментов разработки

1. Установить систему контроля версий **Git**.
2. Создать локальный репозиторий для учебных заданий по тестированию.
3. Настроить пользовательские параметры Git (имя, e-mail).

3. Подготовка среды для автоматизированного тестирования

1. Установить интерпретатор **Python** (актуальной стабильной версии).
 2. Создать виртуальное окружение Python.
 3. Установить фреймворк **PyTest** в виртуальное окружение.
 4. Проверить корректность установки путём запуска тестового примера.
-

4. Инструменты для тестирования веб-приложений и API

1. Установить современный веб-браузер.
2. Изучить и активировать инструменты разработчика браузера.
3. Установить инструмент для тестирования API (например, Postman или аналог).
4. Выполнить тестовый запрос к открытому API.

5. Контейнеризация тестового окружения (ознакомительно)

1. Установить **Docker**.
2. Проверить работоспособность Docker.
3. Запустить тестовый контейнер.
4. Кратко описать преимущества контейнеризации для тестирования.

6. Документирование результата

Подготовить отчёт, включающий:

- перечень установленного ПО;
- краткое описание этапов настройки;
- скриншоты или вывод команд, подтверждающие успешную установку;
- выводы о готовности рабочего места к тестированию ПО.

Формат сдачи

- отчёт в формате PDF / DOCX / Markdown;
- структура отчёта должна соответствовать этапам задания;
- допускается размещение материалов в Git-репозитории.

Критерии оценки

Критерий	Баллы
Корректность установки и настройки окружения	
Работоспособность инструментов	
Полнота и аккуратность отчёта	
Понимание назначения инструментов	

Ожидаемый результат

В результате выполнения домашней работы обучающийся:

- подготовит рабочее место тестировщика;
- освоит базовые инструменты тестирования ПО;
- поймёт роль окружения в качестве и воспроизводимости тестирования;
- будет готов к выполнению практических и лабораторных работ по дисциплине.

№ 12. Кейс: "Проектирование и реализация тестирования модуля сервиса EnergyMonitor".

КЕЙС Проектирование и реализация тестирования модуля сервиса EnergyMonitor (Эн+ Диджитал)

Цель работы

Закрепить навыки применения видов и уровней тестирования программного обеспечения в рамках учебного проекта, а также сформировать практические умения разработки и сопровождения автоматизированных тестов с использованием фреймворка PyTest.

Описание кейса

В рамках учебного проекта рассматривается программный модуль сервиса **EnergyMonitor**, предназначенного для анализа энергопотребления промышленных объектов компании **Эн+ Диджитал**. Вам необходимо спроектировать и реализовать систему тестирования данного модуля, обеспечивающую проверку корректности бизнес-логики и устойчивости работы функций.

Вы выступаете в роли **инженера-разработчика**, отвечающего за качество программного кода.

Задание

Часть 1. Анализ и проектирование тестирования

1. Кратко опишите функциональное назначение тестируемого модуля.
 2. Определите:
 - применимые **уровни тестирования**;
 - соответствующие **виды тестирования**.
 3. Сформируйте перечень тестируемых сценариев, включая:
 - корректные входные данные;
 - граничные случаи;
 - потенциально ошибочные ситуации.
 4. Опишите риски для системы при отсутствии или неполном тестировании.
-

Часть 2. Реализация автоматизированных тестов

5. Создайте структуру проекта для тестирования (исходный код и каталог с тестами).
6. Реализуйте:
 - модульные тесты для отдельных функций;
 - не менее одного интеграционного теста для проверки взаимодействия функций.
7. Используйте возможности PyTest:
 -

- assert-проверки;
 - параметризацию тестов;
 - осмысленные имена тестовых функций.
-

Часть 3. Анализ и оформление результатов

8. Запустите тесты и зафиксируйте результаты выполнения.
 9. Проанализируйте:
 -
 - какие дефекты выявляются тестами;
 - какие сценарии требуют дополнительного покрытия.
 10. Сформулируйте вывод о готовности модуля к использованию и дальнейшему развитию.
-

Требования к оформлению и сдаче

- язык программирования: **Python**;
 - фреймворк тестирования: **PyTest**;
 - форма сдачи:
 -
 - Git-репозиторий **или**
 - архив с проектом и отчётом;
 - отчёт оформляется в формате PDF / DOCX / Markdown и включает:
 -
 - описание проекта;
 - обоснование выбора видов и уровней тестирования;
 - краткие выводы.
-

Критерии оценки

- корректность применения понятий «виды тестирования» и «уровни тестирования»;
- полнота и качество тестовых сценариев;
- работоспособность и читаемость тестового кода;
- использование возможностей PyTest;
- аргументированность аналитической части.

№ 13. Кейс. Проектирование тест-дизайна и управление дефектами.

Домашняя работа по проекту

Тема

КЕЙС. Проектирование тест-дизайна и управление дефектами для программного модуля (Эн+ Диджитал)

Цель домашней работы

Закрепить практические навыки применения техник тест-дизайна, разработки автомат-

изированных тестов и оформления дефектов в системе управления задачами (Jira) при тестировании программного обеспечения, используемого в промышленной ИТ-компании.

Описание проекта

В рамках учебного проекта вы работаете с программным модулем сервиса **EnergyMonitor**, предназначенного для анализа и обработки данных об энергопотреблении. Модуль реализован на языке Python и содержит как корректные участки логики, так и потенциальные дефекты.

Вы выполняете роль **инженера-разработчика / QA-инженера** компании **Эн+Диджитал**, отвечающего за качество программного продукта.

Задание

Часть 1. Тест-дизайн

1. На основе исходного кода модуля:
 - проанализируйте его функциональность;
 - определите ключевые точки риска.
 2. Разработайте **чек-лист или набор тест-кейсов**, используя техники тест-дизайна:
 - эквивалентное разбиение;
 - анализ граничных значений;
 - негативное тестирование.
 3. Для каждого пункта чек-листа / тест-кейса укажите:
 - тестируемую функцию;
 - входные данные;
 - ожидаемый результат.
-

Часть 2. Реализация автоматизированных тестов

4. Реализуйте автоматизированные тесты на **Python с использованием PyTest**:
 - модульные тесты для отдельных функций;
 - не менее одного интеграционного теста.
5. В тестах обязательно:
 - использовать `assert`;
 - применить параметризацию;
 - покрыть позитивные и негативные сценарии.

Часть 3. Управление дефектами

6. По результатам выполнения тестов:
 - выявите дефекты в программном коде;
 - для каждого дефекта оформите **баг-репорт**.

7. Каждый баг-репорт должен содержать:

- заголовок (Summary);
- шаги воспроизведения (Steps to Reproduce);
- ожидаемый и фактический результат;
- окружение;
- приоритет и критичность.

8. Оформите баг-репорты в **Jira** (или в виде макета баг-репорта, если доступ к Jira отсутствует).

Часть 4. Анализ и выводы

9. Кратко опишите:

- какие дефекты были выявлены с помощью тест-дизайна;
- какие дефекты удалось обнаружить только при автоматизированном тестировании;
- почему корректное оформление баг-репортов важно для командной разработки.

Требования к оформлению и сдаче

- язык программирования: **Python**;
- фреймворк тестирования: **PyTest**;
- формат сдачи:
 - Git-репозиторий с кодом и тестами **или**
 - архив с проектом;
- отчёт (1–2 страницы) в формате PDF / DOCX / Markdown, включающий:
 - описание тест-дизайна;
 - примеры баг-репортов;
 - выводы.

Критерии оценки

- корректность применения техник тест-дизайна;
- полнота и качество тестового покрытия;
- корректность и читаемость автоматизированных тестов;
- качество оформления баг-репортов;
- аргументированность выводов.

2.3.7. Практические задания для оценки компетенции «ПК-2 ППК-Р2.4»

№ 14. Подготовка рабочего места.

Домашняя работа

Тема: Подготовка рабочего места для тестирования программного обеспечения

Цель работы

Освоить базовые навыки настройки рабочего окружения тестировщика, необходимые для выполнения ручного и автоматизированного тестирования программного обеспечения, а также сформировать понимание воспроизводимости тестового окружения.

Исходные условия

Тестирование выполняется на локальном компьютере обучающегося с использованием открытого программного обеспечения. Допускается использование операционной системы Linux или Windows (с поддержкой виртуализации/контейнеризации).

Задание

1. Настройка операционной среды

1. Установить и настроить операционную систему:
—
Linux (рекомендуется Ubuntu или Debian) **или**
— Windows с поддержкой командной строки.
 2. Проверить работу терминала и базовых системных утилит.
 3. Настроить структуру рабочих каталогов для проектов тестирования.
-

2. Установка и настройка инструментов разработки

1. Установить систему контроля версий **Git**.
 2. Создать локальный репозиторий для учебных заданий по тестированию.
 3. Настроить пользовательские параметры Git (имя, e-mail).
-

3. Подготовка среды для автоматизированного тестирования

1. Установить интерпретатор **Python** (актуальной стабильной версии).
 2. Создать виртуальное окружение Python.
 3. Установить фреймворк **PyTest** в виртуальное окружение.
 4. Проверить корректность установки путём запуска тестового примера.
-

4. Инструменты для тестирования веб-приложений и API

1. Установить современный веб-браузер.
 2. Изучить и активировать инструменты разработчика браузера.
 3. Установить инструмент для тестирования API (например, Postman или аналог).
 4. Выполнить тестовый запрос к открытому API.
-

5. Контейнеризация тестового окружения (ознакомительно)

1. Установить **Docker**.
2. Проверить работоспособность Docker.

3. Запустить тестовый контейнер.
4. Кратко описать преимущества контейнеризации для тестирования.

6. Документирование результата

Подготовить отчёт, включающий:

- перечень установленного ПО;
- краткое описание этапов настройки;
- скриншоты или вывод команд, подтверждающие успешную установку;
- выводы о готовности рабочего места к тестированию ПО.

Формат сдачи

- отчёт в формате PDF / DOCX / Markdown;
- структура отчёта должна соответствовать этапам задания;
- допускается размещение материалов в Git-репозитории.

Критерии оценки

Критерий	Баллы
Корректность установки и настройки окружения	
Работоспособность инструментов	
Полнота и аккуратность отчёта	
Понимание назначения инструментов	

Ожидаемый результат

В результате выполнения домашней работы обучающийся:

- подготовит рабочее место тестировщика;
- освоит базовые инструменты тестирования ПО;
- поймёт роль окружения в качестве и воспроизводимости тестирования;
- будет готов к выполнению практических и лабораторных работ по дисциплине.

№ 15. Кейс: "Проектирование и реализация тестирования модуля сервиса EnergyMonitor".

КЕЙС Проектирование и реализация тестирования модуля сервиса EnergyMonitor (Эн+ Диджитал)

Цель работы

Закрепить навыки применения видов и уровней тестирования программного обеспечения в рамках учебного проекта, а также сформировать практические умения разработки и сопровождения автоматизированных тестов с использованием фреймворка PyTest.

Описание кейса

В рамках учебного проекта рассматривается программный модуль сервиса **EnergyMonitor**, предназначенного для анализа энергопотребления промышленных объектов компании **Эн+ Диджитал**. Вам необходимо спроектировать и реализовать систему тестирования данного модуля, обеспечивающую проверку корректности бизнес-логики и устойчивости работы функций.

Вы выступаете в роли **инженера-разработчика**, отвечающего за качество программного кода.

Задание

Часть 1. Анализ и проектирование тестирования

1. Кратко опишите функциональное назначение тестируемого модуля.
 2. Определите:
 - применимые **уровни тестирования**;
 - соответствующие **виды тестирования**.
 3. Сформируйте перечень тестируемых сценариев, включая:
 - корректные входные данные;
 - граничные случаи;
 - потенциально ошибочные ситуации.
 4. Опишите риски для системы при отсутствии или неполном тестировании.
-

Часть 2. Реализация автоматизированных тестов

5. Создайте структуру проекта для тестирования (исходный код и каталог с тестами).
 6. Реализуйте:
 - модульные тесты для отдельных функций;
 - не менее одного интеграционного теста для проверки взаимодействия функций.
 7. Используйте возможности PyTest:
 - assert-проверки;
 - параметризацию тестов;
 - осмысленные имена тестовых функций.
-

Часть 3. Анализ и оформление результатов

8. Запустите тесты и зафиксируйте результаты выполнения.
9. Проанализируйте:

-
- какие дефекты выявляются тестами;
- какие сценарии требуют дополнительного покрытия.

10. Сформулируйте вывод о готовности модуля к использованию и дальнейшему развитию.

Требования к оформлению и сдаче

- язык программирования: **Python**;
- фреймворк тестирования: **PyTest**;
- форма сдачи:
 -
 - Git-репозиторий **или**
 - архив с проектом и отчётом;
- отчёт оформляется в формате PDF / DOCX / Markdown и включает:
 -
 - описание проекта;
 - обоснование выбора видов и уровней тестирования;
 - краткие выводы.

Критерии оценки

- корректность применения понятий «виды тестирования» и «уровни тестирования»;
- полнота и качество тестовых сценариев;
- работоспособность и читаемость тестового кода;
- использование возможностей PyTest;
- аргументированность аналитической части.

№ 16. Кейс. Проектирование тест-дизайна и управление дефектами.

Домашняя работа по проекту

Тема

КЕЙС. Проектирование тест-дизайна и управление дефектами для программного модуля (Эн+ Диджитал)

Цель домашней работы

Закрепить практические навыки применения техник тест-дизайна, разработки автоматизированных тестов и оформления дефектов в системе управления задачами (Jira) при тестировании программного обеспечения, используемого в промышленной ИТ-компании.

Описание проекта

В рамках учебного проекта вы работаете с программным модулем сервиса **EnergyMonitor**, предназначенного для анализа и обработки данных об энергопотреблении. Модуль реализован на языке Python и содержит как корректные участки логики, так и потенциальные дефекты.

Вы выполняете роль **инженера-разработчика / QA-инженера** компании **Эн+ Диджитал**, отвечающего за качество программного продукта.

Задание

Часть 1. Тест-дизайн

1. На основе исходного кода модуля:
 - проанализируйте его функциональность;
 - определите ключевые точки риска.
 2. Разработайте **чек-лист или набор тест-кейсов**, используя техники тест-дизайна:
 - эквивалентное разбиение;
 - анализ граничных значений;
 - негативное тестирование.
 3. Для каждого пункта чек-листа / тест-кейса укажите:
 - тестируемую функцию;
 - входные данные;
 - ожидаемый результат.
-

Часть 2. Реализация автоматизированных тестов

4. Реализуйте автоматизированные тесты на **Python с использованием PyTest**:
 - модульные тесты для отдельных функций;
 - не менее одного интеграционного теста.
5. В тестах обязательно:
 - использовать `assert`;
 - применить параметризацию;
 - покрыть позитивные и негативные сценарии.

Часть 3. Управление дефектами

6. По результатам выполнения тестов:
 - выявите дефекты в программном коде;
 - для каждого дефекта оформите **баг-репорт**.
7. Каждый баг-репорт должен содержать:
 - заголовок (Summary);
 - шаги воспроизведения (Steps to Reproduce);
 - ожидаемый и фактический результат;
 - окружение;
 - приоритет и критичность.
8. Оформите баг-репорты в **Jira** (или в виде макета баг-репорта, если доступ к Jira

отсутствует).

Часть 4. Анализ и выводы

9. Кратко опишите:

- какие дефекты были выявлены с помощью тест-дизайна;
- какие дефекты удалось обнаружить только при автоматизированном тестировании;
- почему корректное оформление баг-репортов важно для командной разработки.

Требования к оформлению и сдаче

- язык программирования: **Python**;
- фреймворк тестирования: **PyTest**;
- формат сдачи:
 - Git-репозиторий с кодом и тестами **или**
 - архив с проектом;
- отчёт (1–2 страницы) в формате PDF / DOCX / Markdown, включающий:
 - описание тест-дизайна;
 - примеры баг-репортов;
 - выводы.

Критерии оценки

- корректность применения техник тест-дизайна;
- полнота и качество тестового покрытия;
- корректность и читаемость автоматизированных тестов;
- качество оформления баг-репортов;
- аргументированность выводов.

2.3.8. Практические задания для оценки компетенции «ПК-8 ППК-Р7.1»

№ 17. Подготовка рабочего места.

Домашняя работа

Тема: Подготовка рабочего места для тестирования программного обеспечения

Цель работы

Освоить базовые навыки настройки рабочего окружения тестировщика, необходимые для выполнения ручного и автоматизированного тестирования программного обеспечения, а также сформировать понимание воспроизводимости тестового окружения.

Исходные условия

Тестирование выполняется на локальном компьютере обучающегося с использованием открытого программного обеспечения. Допускается использование операционной системы Linux или Windows (с поддержкой виртуализации/контейнеризации).

Задание

1. Настройка операционной среды

1. Установить и настроить операционную систему:
—
Linux (рекомендуется Ubuntu или Debian) **или**
— Windows с поддержкой командной строки.
2. Проверить работу терминала и базовых системных утилит.
3. Настроить структуру рабочих каталогов для проектов тестирования.

2. Установка и настройка инструментов разработки

1. Установить систему контроля версий **Git**.
2. Создать локальный репозиторий для учебных заданий по тестированию.
3. Настроить пользовательские параметры Git (имя, e-mail).

3. Подготовка среды для автоматизированного тестирования

1. Установить интерпретатор **Python** (актуальной стабильной версии).
2. Создать виртуальное окружение Python.
3. Установить фреймворк **PyTest** в виртуальное окружение.
4. Проверить корректность установки путём запуска тестового примера.

4. Инструменты для тестирования веб-приложений и API

1. Установить современный веб-браузер.
2. Изучить и активировать инструменты разработчика браузера.
3. Установить инструмент для тестирования API (например, Postman или аналог).
4. Выполнить тестовый запрос к открытому API.

5. Контейнеризация тестового окружения (ознакомительно)

1. Установить **Docker**.
2. Проверить работоспособность Docker.
3. Запустить тестовый контейнер.
4. Кратко описать преимущества контейнеризации для тестирования.

6. Документирование результата

Подготовить отчёт, включающий:

- перечень установленного ПО;
- краткое описание этапов настройки;
- скриншоты или вывод команд, подтверждающие успешную установку;
- выводы о готовности рабочего места к тестированию ПО.

Формат сдачи

- отчёт в формате PDF / DOCX / Markdown;
 - структура отчёта должна соответствовать этапам задания;
 - допускается размещение материалов в Git-репозитории.
-

Критерии оценки

Критерий	Баллы
Корректность установки и настройки окружения	
Работоспособность инструментов	
Полнота и аккуратность отчёта	
Понимание назначения инструментов	

Ожидаемый результат

В результате выполнения домашней работы обучающийся:

- подготовит рабочее место тестировщика;
- освоит базовые инструменты тестирования ПО;
- поймёт роль окружения в качестве и воспроизводимости тестирования;
- будет готов к выполнению практических и лабораторных работ по дисциплине.

№ 18. Кейс. Проектирование тест-дизайна и управление дефектами.

Домашняя работа по проекту

Тема

КЕЙС. Проектирование тест-дизайна и управление дефектами для программного модуля (Эн+ Диджитал)

Цель домашней работы

Закрепить практические навыки применения техник тест-дизайна, разработки автоматизированных тестов и оформления дефектов в системе управления задачами (Jira) при тестировании программного обеспечения, используемого в промышленной ИТ-компании.

Описание проекта

В рамках учебного проекта вы работаете с программным модулем сервиса **EnergyMonitor**, предназначенного для анализа и обработки данных об энергопотреблении. Модуль реализован на языке Python и содержит как корректные участки логики, так и потенциальные дефекты.

Вы выполняете роль **инженера-разработчика / QA-инженера** компании **Эн+ Диджитал**, отвечающего за качество программного продукта.

Задание

Часть 1. Тест-дизайн

1. На основе исходного кода модуля:

- проанализируйте его функциональность;

- определите ключевые точки риска.
 - 2. Разработайте **чек-лист или набор тест-кейсов**, используя техники тест-дизайна:
 - эквивалентное разбиение;
 - анализ граничных значений;
 - негативное тестирование.
 - 3. Для каждого пункта чек-листа / тест-кейса укажите:
 - тестируемую функцию;
 - входные данные;
 - ожидаемый результат.
-

Часть 2. Реализация автоматизированных тестов

- 4. Реализуйте автоматизированные тесты на **Python с использованием PyTest**:
 - модульные тесты для отдельных функций;
 - не менее одного интеграционного теста.
- 5. В тестах обязательно:
 - использовать `assert`;
 - применить параметризацию;
 - покрыть позитивные и негативные сценарии.

Часть 3. Управление дефектами

- 6. По результатам выполнения тестов:
 - выявите дефекты в программном коде;
 - для каждого дефекта оформите **баг-репорт**.
- 7. Каждый баг-репорт должен содержать:
 - заголовок (Summary);
 - шаги воспроизведения (Steps to Reproduce);
 - ожидаемый и фактический результат;
 - окружение;
 - приоритет и критичность.
- 8. Оформите баг-репорты в **Jira** (или в виде макета баг-репорта, если доступ к Jira отсутствует).

Часть 4. Анализ и выводы

- 9. Кратко опишите:
 - какие дефекты были выявлены с помощью тест-дизайна;

- какие дефекты удалось обнаружить только при автоматизированном тестировании;

- почему корректное оформление баг-репортов важно для командной разработки.

Требования к оформлению и сдаче

- язык программирования: **Python**;

- фреймворк тестирования: **PyTest**;

- формат сдачи:

-

- Git-репозиторий с кодом и тестами **или**

- архив с проектом;

- отчёт (1–2 страницы) в формате PDF / DOCX / Markdown, включающий:

-

- описание тест-дизайна;

- примеры баг-репортов;

- выводы.

Критерии оценки

- корректность применения техник тест-дизайна;

- полнота и качество тестового покрытия;

- корректность и читаемость автоматизированных тестов;

- качество оформления баг-репортов;

- аргументированность выводов.

2.3.9. Практические задания для оценки компетенции «ПК-8 ППК-Р7.2»

№ 19. Подготовка рабочего места.

Домашняя работа

Тема: Подготовка рабочего места для тестирования программного обеспечения

Цель работы

Освоить базовые навыки настройки рабочего окружения тестировщика, необходимые для выполнения ручного и автоматизированного тестирования программного обеспечения, а также сформировать понимание воспроизводимости тестового окружения.

Исходные условия

Тестирование выполняется на локальном компьютере обучающегося с использованием открытого программного обеспечения. Допускается использование операционной системы Linux или Windows (с поддержкой виртуализации/контейнеризации).

Задание

1. Настройка операционной среды

1. Установить и настроить операционную систему:

-

Linux (рекомендуется Ubuntu или Debian) **или**

— Windows с поддержкой командной строки.

2. Проверить работу терминала и базовых системных утилит.
3. Настроить структуру рабочих каталогов для проектов тестирования.

2. Установка и настройка инструментов разработки

1. Установить систему контроля версий **Git**.
2. Создать локальный репозиторий для учебных заданий по тестированию.
3. Настроить пользовательские параметры Git (имя, e-mail).

3. Подготовка среды для автоматизированного тестирования

1. Установить интерпретатор **Python** (актуальной стабильной версии).
2. Создать виртуальное окружение Python.
3. Установить фреймворк **PyTest** в виртуальное окружение.
4. Проверить корректность установки путём запуска тестового примера.

4. Инструменты для тестирования веб-приложений и API

1. Установить современный веб-браузер.
2. Изучить и активировать инструменты разработчика браузера.
3. Установить инструмент для тестирования API (например, Postman или аналог).
4. Выполнить тестовый запрос к открытому API.

5. Контейнеризация тестового окружения (ознакомительно)

1. Установить **Docker**.
2. Проверить работоспособность Docker.
3. Запустить тестовый контейнер.
4. Кратко описать преимущества контейнеризации для тестирования.

6. Документирование результата

Подготовить отчёт, включающий:

- перечень установленного ПО;
- краткое описание этапов настройки;
- скриншоты или вывод команд, подтверждающие успешную установку;
- выводы о готовности рабочего места к тестированию ПО.

Формат сдачи

- отчёт в формате PDF / DOCX / Markdown;
- структура отчёта должна соответствовать этапам задания;

— допускается размещение материалов в Git-репозитории.

Критерии оценки

Критерий	Баллы
Корректность установки и настройки окружения	
Работоспособность инструментов	
Полнота и аккуратность отчёта	
Понимание назначения инструментов	

Ожидаемый результат

В результате выполнения домашней работы обучающийся:

- подготовит рабочее место тестировщика;
- освоит базовые инструменты тестирования ПО;
- поймёт роль окружения в качестве и воспроизводимости тестирования;
- будет готов к выполнению практических и лабораторных работ по дисциплине.

№ 20. Кейс: "Проектирование и реализация тестирования модуля сервиса *EnergyMonitor*".

КЕЙС Проектирование и реализация тестирования модуля сервиса *EnergyMonitor* (Эн+ Диджитал)

Цель работы

Закрепить навыки применения видов и уровней тестирования программного обеспечения в рамках учебного проекта, а также сформировать практические умения разработки и сопровождения автоматизированных тестов с использованием фреймворка PyTest.

Описание кейса

В рамках учебного проекта рассматривается программный модуль сервиса **EnergyMonitor**, предназначенного для анализа энергопотребления промышленных объектов компании **Эн+ Диджитал**. Вам необходимо спроектировать и реализовать систему тестирования данного модуля, обеспечивающую проверку корректности бизнес-логики и устойчивости работы функций.

Вы выступаете в роли **инженера-разработчика**, отвечающего за качество программного кода.

Задание

Часть 1. Анализ и проектирование тестирования

1. Кратко опишите функциональное назначение тестируемого модуля.
2. Определите:

—

- применимые **уровни тестирования**;
- соответствующие **виды тестирования**.

3. Сформируйте перечень тестируемых сценариев, включая:

-
- корректные входные данные;
- граничные случаи;
- потенциально ошибочные ситуации.

4. Опишите риски для системы при отсутствии или неполном тестировании.

Часть 2. Реализация автоматизированных тестов

5. Создайте структуру проекта для тестирования (исходный код и каталог с тестами).

6. Реализуйте:

-
- модульные тесты для отдельных функций;
- не менее одного интеграционного теста для проверки взаимодействия функций.

7. Используйте возможности PyTest:

- - assert-проверки;
 - параметризацию тестов;
 - осмысленные имена тестовых функций.
-

Часть 3. Анализ и оформление результатов

8. Запустите тесты и зафиксируйте результаты выполнения.

9. Проанализируйте:

-
- какие дефекты выявляются тестами;
- какие сценарии требуют дополнительного покрытия.

10. Сформулируйте вывод о готовности модуля к использованию и дальнейшему развитию.

Требования к оформлению и сдаче

- язык программирования: **Python**;
- фреймворк тестирования: **PyTest**;
- форма сдачи:
 -
 - Git-репозиторий **или**
 - архив с проектом и отчётом;
- отчёт оформляется в формате PDF / DOCX / Markdown и включает:
 -

- описание проекта;
 - обоснование выбора видов и уровней тестирования;
 - краткие выводы.
-

Критерии оценки

- корректность применения понятий «виды тестирования» и «уровни тестирования»;
- полнота и качество тестовых сценариев;
- работоспособность и читаемость тестового кода;
- использование возможностей PyTest;
- аргументированность аналитической части.

№ 21. Кейс. Проектирование тест-дизайна и управление дефектами.

Домашняя работа по проекту

Тема

КЕЙС. Проектирование тест-дизайна и управление дефектами для программного модуля (Эн+ Диджитал)

Цель домашней работы

Закрепить практические навыки применения техник тест-дизайна, разработки автоматизированных тестов и оформления дефектов в системе управления задачами (Jira) при тестировании программного обеспечения, используемого в промышленной ИТ-компании.

Описание проекта

В рамках учебного проекта вы работаете с программным модулем сервиса **EnergyMonitor**, предназначенного для анализа и обработки данных об энергопотреблении. Модуль реализован на языке Python и содержит как корректные участки логики, так и потенциальные дефекты.

Вы выполняете роль **инженера-разработчика / QA-инженера** компании **Эн+ Диджитал**, отвечающего за качество программного продукта.

Задание

Часть 1. Тест-дизайн

1. На основе исходного кода модуля:
 - проанализируйте его функциональность;
 - определите ключевые точки риска.
2. Разработайте **чек-лист или набор тест-кейсов**, используя техники тест-дизайна:
 - эквивалентное разбиение;
 - анализ граничных значений;
 - негативное тестирование.
3. Для каждого пункта чек-листа / тест-кейса укажите:
 - тестируемую функцию;

- входные данные;
 - ожидаемый результат.
-

Часть 2. Реализация автоматизированных тестов

4. Реализуйте автоматизированные тесты на **Python** с использованием **PyTest**:

- модульные тесты для отдельных функций;
- не менее одного интеграционного теста.

5. В тестах обязательно:

- использовать `assert`;
- применить параметризацию;
- покрыть позитивные и негативные сценарии.

Часть 3. Управление дефектами

6. По результатам выполнения тестов:

- выявите дефекты в программном коде;
- для каждого дефекта оформите **баг-репорт**.

7. Каждый баг-репорт должен содержать:

- заголовок (Summary);
- шаги воспроизведения (Steps to Reproduce);
- ожидаемый и фактический результат;
- окружение;
- приоритет и критичность.

8. Оформите баг-репорты в **Jira** (или в виде макета баг-репорта, если доступ к Jira отсутствует).

Часть 4. Анализ и выводы

9. Кратко опишите:

- какие дефекты были выявлены с помощью тест-дизайна;
- какие дефекты удалось обнаружить только при автоматизированном тестировании;
- почему корректное оформление баг-репортов важно для командной разработки.

Требования к оформлению и сдаче

- язык программирования: **Python**;
- фреймворк тестирования: **PyTest**;
- формат сдачи:

- Git-репозиторий с кодом и тестами **или**
 - архив с проектом;
- отчёт (1–2 страницы) в формате PDF / DOCX / Markdown, включающий:
 - описание тест-дизайна;
 - примеры баг-репортов;
 - выводы.

Критерии оценки

- корректность применения техник тест-дизайна;
- полнота и качество тестового покрытия;
- корректность и читаемость автоматизированных тестов;
- качество оформления баг-репортов;
- аргументированность выводов.

2.3.10. Практические задания для оценки компетенции «ПК-8 ППК-Р7.3»

№ 22. Подготовка рабочего места.

Домашняя работа

Тема: Подготовка рабочего места для тестирования программного обеспечения

Цель работы

Освоить базовые навыки настройки рабочего окружения тестировщика, необходимые для выполнения ручного и автоматизированного тестирования программного обеспечения, а также сформировать понимание воспроизводимости тестового окружения.

Исходные условия

Тестирование выполняется на локальном компьютере обучающегося с использованием открытого программного обеспечения. Допускается использование операционной системы Linux или Windows (с поддержкой виртуализации/контейнеризации).

Задание

1. Настройка операционной среды

1. Установить и настроить операционную систему:
 - Linux (рекомендуется Ubuntu или Debian) **или**
 - Windows с поддержкой командной строки.
 2. Проверить работу терминала и базовых системных утилит.
 3. Настроить структуру рабочих каталогов для проектов тестирования.
-

2. Установка и настройка инструментов разработки

1. Установить систему контроля версий **Git**.
2. Создать локальный репозиторий для учебных заданий по тестированию.

3. Настроить пользовательские параметры Git (имя, e-mail).

3. Подготовка среды для автоматизированного тестирования

1. Установить интерпретатор **Python** (актуальной стабильной версии).
2. Создать виртуальное окружение Python.
3. Установить фреймворк **PyTest** в виртуальное окружение.
4. Проверить корректность установки путём запуска тестового примера.

4. Инструменты для тестирования веб-приложений и API

1. Установить современный веб-браузер.
2. Изучить и активировать инструменты разработчика браузера.
3. Установить инструмент для тестирования API (например, Postman или аналог).
4. Выполнить тестовый запрос к открытому API.

5. Контейнеризация тестового окружения (ознакомительно)

1. Установить **Docker**.
2. Проверить работоспособность Docker.
3. Запустить тестовый контейнер.
4. Кратко описать преимущества контейнеризации для тестирования.

6. Документирование результата

Подготовить отчёт, включающий:

- перечень установленного ПО;
- краткое описание этапов настройки;
- скриншоты или вывод команд, подтверждающие успешную установку;
- выводы о готовности рабочего места к тестированию ПО.

Формат сдачи

- отчёт в формате PDF / DOCX / Markdown;
- структура отчёта должна соответствовать этапам задания;
- допускается размещение материалов в Git-репозитории.

Критерии оценки

Критерий	Баллы
Корректность установки и настройки окружения	
Работоспособность инструментов	
Полнота и аккуратность отчёта	
Понимание назначения инструментов	

Ожидаемый результат

В результате выполнения домашней работы обучающийся:

- подготовит рабочее место тестировщика;
- освоит базовые инструменты тестирования ПО;
- поймёт роль окружения в качестве и воспроизводимости тестирования;
- будет готов к выполнению практических и лабораторных работ по дисциплине.

№ 23. Кейс: "Проектирование и реализация тестирования модуля сервиса *EnergyMonitor*".

КЕЙС Проектирование и реализация тестирования модуля сервиса *EnergyMonitor* (Эн+ Диджитал)

Цель работы

Закрепить навыки применения видов и уровней тестирования программного обеспечения в рамках учебного проекта, а также сформировать практические умения разработки и сопровождения автоматизированных тестов с использованием фреймворка PyTest.

Описание кейса

В рамках учебного проекта рассматривается программный модуль сервиса **EnergyMonitor**, предназначенного для анализа энергопотребления промышленных объектов компании **Эн+ Диджитал**. Вам необходимо спроектировать и реализовать систему тестирования данного модуля, обеспечивающую проверку корректности бизнес-логики и устойчивости работы функций.

Вы выступаете в роли **инженера-разработчика**, отвечающего за качество программного кода.

Задание

Часть 1. Анализ и проектирование тестирования

1. Кратко опишите функциональное назначение тестируемого модуля.
2. Определите:
 - применимые **уровни тестирования**;
 - соответствующие **виды тестирования**.
3. Сформируйте перечень тестируемых сценариев, включая:
 - корректные входные данные;
 - граничные случаи;
 - потенциально ошибочные ситуации.
4. Опишите риски для системы при отсутствии или неполном тестировании.

Часть 2. Реализация автоматизированных тестов

5. Создайте структуру проекта для тестирования (исходный код и каталог с тестами).

6. Реализуйте:

- модульные тесты для отдельных функций;
- не менее одного интеграционного теста для проверки взаимодействия функций.

7. Используйте возможности PyTest:

- assert-проверки;
- параметризацию тестов;
- осмысленные имена тестовых функций.

Часть 3. Анализ и оформление результатов

8. Запустите тесты и зафиксируйте результаты выполнения.

9. Проанализируйте:

- какие дефекты выявляются тестами;
- какие сценарии требуют дополнительного покрытия.

10. Сформулируйте вывод о готовности модуля к использованию и дальнейшему развитию.

Требования к оформлению и сдаче

- язык программирования: **Python**;
- фреймворк тестирования: **PyTest**;
- форма сдачи:
 - Git-репозиторий **или**
 - архив с проектом и отчётом;
- отчёт оформляется в формате PDF / DOCX / Markdown и включает:
 - описание проекта;
 - обоснование выбора видов и уровней тестирования;
 - краткие выводы.

Критерии оценки

- корректность применения понятий «виды тестирования» и «уровни тестирования»;
- полнота и качество тестовых сценариев;
- работоспособность и читаемость тестового кода;

- использование возможностей PyTest;
- аргументированность аналитической части.

2.3.11. Практические задания для оценки компетенции «ПК-8 ППК-Р7.4»

№ 24. Подготовка рабочего места.

Домашняя работа

Тема: Подготовка рабочего места для тестирования программного обеспечения

Цель работы

Освоить базовые навыки настройки рабочего окружения тестировщика, необходимые для выполнения ручного и автоматизированного тестирования программного обеспечения, а также сформировать понимание воспроизводимости тестового окружения.

Исходные условия

Тестирование выполняется на локальном компьютере обучающегося с использованием открытого программного обеспечения. Допускается использование операционной системы Linux или Windows (с поддержкой виртуализации/контейнеризации).

Задание

1. Настройка операционной среды

1. Установить и настроить операционную систему:
 - Linux (рекомендуется Ubuntu или Debian) **или**
 - Windows с поддержкой командной строки.
2. Проверить работу терминала и базовых системных утилит.
3. Настроить структуру рабочих каталогов для проектов тестирования.

2. Установка и настройка инструментов разработки

1. Установить систему контроля версий **Git**.
2. Создать локальный репозиторий для учебных заданий по тестированию.
3. Настроить пользовательские параметры Git (имя, e-mail).

3. Подготовка среды для автоматизированного тестирования

1. Установить интерпретатор **Python** (актуальной стабильной версии).
2. Создать виртуальное окружение Python.
3. Установить фреймворк **PyTest** в виртуальное окружение.
4. Проверить корректность установки путём запуска тестового примера.

4. Инструменты для тестирования веб-приложений и API

1. Установить современный веб-браузер.
2. Изучить и активировать инструменты разработчика браузера.

3. Установить инструмент для тестирования API (например, Postman или аналог).
4. Выполнить тестовый запрос к открытому API.

5. Контейнеризация тестового окружения (ознакомительно)

1. Установить **Docker**.
2. Проверить работоспособность Docker.
3. Запустить тестовый контейнер.
4. Кратко описать преимущества контейнеризации для тестирования.

6. Документирование результата

Подготовить отчёт, включающий:

- перечень установленного ПО;
- краткое описание этапов настройки;
- скриншоты или вывод команд, подтверждающие успешную установку;
- выводы о готовности рабочего места к тестированию ПО.

Формат сдачи

- отчёт в формате PDF / DOCX / Markdown;
- структура отчёта должна соответствовать этапам задания;
- допускается размещение материалов в Git-репозитории.

Критерии оценки

Критерий	Баллы
Корректность установки и настройки окружения	
Работоспособность инструментов	
Полнота и аккуратность отчёта	
Понимание назначения инструментов	

Ожидаемый результат

В результате выполнения домашней работы обучающийся:

- подготовит рабочее место тестировщика;
- освоит базовые инструменты тестирования ПО;
- поймёт роль окружения в качестве и воспроизводимости тестирования;
- будет готов к выполнению практических и лабораторных работ по дисциплине.

2.3.12. Практические задания для оценки компетенции «ПК-6 ИП-2.1»

№ 25. Кейс: "Проектирование и реализация тестирования модуля сервиса *EnergyMonitor*".

КЕЙС Проектирование и реализация тестирования модуля сервиса EnergyMonitor (Эн+ Диджитал)

Цель работы

Закрепить навыки применения видов и уровней тестирования программного обеспечения в рамках учебного проекта, а также сформировать практические умения разработки и сопровождения автоматизированных тестов с использованием фреймворка PyTest.

Описание кейса

В рамках учебного проекта рассматривается программный модуль сервиса **EnergyMonitor**, предназначенного для анализа энергопотребления промышленных объектов компании **Эн+ Диджитал**. Вам необходимо спроектировать и реализовать систему тестирования данного модуля, обеспечивающую проверку корректности бизнес-логики и устойчивости работы функций.

Вы выступаете в роли **инженера-разработчика**, отвечающего за качество программного кода.

Задание

Часть 1. Анализ и проектирование тестирования

1. Кратко опишите функциональное назначение тестируемого модуля.
 2. Определите:
 - применимые **уровни тестирования**;
 - соответствующие **виды тестирования**.
 3. Сформируйте перечень тестируемых сценариев, включая:
 - корректные входные данные;
 - граничные случаи;
 - потенциально ошибочные ситуации.
 4. Опишите риски для системы при отсутствии или неполном тестировании.
-

Часть 2. Реализация автоматизированных тестов

5. Создайте структуру проекта для тестирования (исходный код и каталог с тестами).
6. Реализуйте:
 - модульные тесты для отдельных функций;
 - не менее одного интеграционного теста для проверки взаимодействия функций.
7. Используйте возможности PyTest:
 -

- assert-проверки;
 - параметризацию тестов;
 - осмысленные имена тестовых функций.
-

Часть 3. Анализ и оформление результатов

8. Запустите тесты и зафиксируйте результаты выполнения.
 9. Проанализируйте:
 -
 - какие дефекты выявляются тестами;
 - какие сценарии требуют дополнительного покрытия.
 10. Сформулируйте вывод о готовности модуля к использованию и дальнейшему развитию.
-

Требования к оформлению и сдаче

- язык программирования: **Python**;
 - фреймворк тестирования: **PyTest**;
 - форма сдачи:
 -
 - Git-репозиторий **или**
 - архив с проектом и отчётом;
 - отчёт оформляется в формате PDF / DOCX / Markdown и включает:
 -
 - описание проекта;
 - обоснование выбора видов и уровней тестирования;
 - краткие выводы.
-

Критерии оценки

- корректность применения понятий «виды тестирования» и «уровни тестирования»;
- полнота и качество тестовых сценариев;
- работоспособность и читаемость тестового кода;
- использование возможностей PyTest;
- аргументированность аналитической части.

№ 26. Кейс. Проектирование тест-дизайна и управление дефектами.

Домашняя работа по проекту

Тема

КЕЙС. Проектирование тест-дизайна и управление дефектами для программного модуля (Эн+ Диджитал)

Цель домашней работы

Закрепить практические навыки применения техник тест-дизайна, разработки автомат-

изированных тестов и оформления дефектов в системе управления задачами (Jira) при тестировании программного обеспечения, используемого в промышленной ИТ-компании.

Описание проекта

В рамках учебного проекта вы работаете с программным модулем сервиса **EnergyMonitor**, предназначенного для анализа и обработки данных об энергопотреблении. Модуль реализован на языке Python и содержит как корректные участки логики, так и потенциальные дефекты.

Вы выполняете роль **инженера-разработчика / QA-инженера** компании **Эн+Диджитал**, отвечающего за качество программного продукта.

Задание

Часть 1. Тест-дизайн

1. На основе исходного кода модуля:
 - проанализируйте его функциональность;
 - определите ключевые точки риска.
 2. Разработайте **чек-лист или набор тест-кейсов**, используя техники тест-дизайна:
 - эквивалентное разбиение;
 - анализ граничных значений;
 - негативное тестирование.
 3. Для каждого пункта чек-листа / тест-кейса укажите:
 - тестируемую функцию;
 - входные данные;
 - ожидаемый результат.
-

Часть 2. Реализация автоматизированных тестов

4. Реализуйте автоматизированные тесты на **Python с использованием PyTest**:
 - модульные тесты для отдельных функций;
 - не менее одного интеграционного теста.
5. В тестах обязательно:
 - использовать `assert`;
 - применить параметризацию;
 - покрыть позитивные и негативные сценарии.

Часть 3. Управление дефектами

6. По результатам выполнения тестов:
 - выявите дефекты в программном коде;
 - для каждого дефекта оформите **баг-репорт**.

7. Каждый баг-репорт должен содержать:

- заголовок (Summary);
- шаги воспроизведения (Steps to Reproduce);
- ожидаемый и фактический результат;
- окружение;
- приоритет и критичность.

8. Оформите баг-репорты в **Jira** (или в виде макета баг-репорта, если доступ к Jira отсутствует).

Часть 4. Анализ и выводы

9. Кратко опишите:

- какие дефекты были выявлены с помощью тест-дизайна;
- какие дефекты удалось обнаружить только при автоматизированном тестировании;
- почему корректное оформление баг-репортов важно для командной разработки.

Требования к оформлению и сдаче

- язык программирования: **Python**;
- фреймворк тестирования: **PyTest**;
- формат сдачи:
 - Git-репозиторий с кодом и тестами **или**
 - архив с проектом;
- отчёт (1–2 страницы) в формате PDF / DOCX / Markdown, включающий:
 - описание тест-дизайна;
 - примеры баг-репортов;
 - выводы.

Критерии оценки

- корректность применения техник тест-дизайна;
- полнота и качество тестового покрытия;
- корректность и читаемость автоматизированных тестов;
- качество оформления баг-репортов;
- аргументированность выводов.

2.3.13. Практические задания для оценки компетенции «ПК-6 ИП-2.2»

№ 27. Кейс: *"Проектирование и реализация тестирования модуля сервиса EnergyMonitor"*.

КЕЙС Проектирование и реализация тестирования модуля сервиса EnergyMonitor (Эн+ Диджитал)

Цель работы

Закрепить навыки применения видов и уровней тестирования программного обеспечения в рамках учебного проекта, а также сформировать практические умения разработки и сопровождения автоматизированных тестов с использованием фреймворка PyTest.

Описание кейса

В рамках учебного проекта рассматривается программный модуль сервиса **EnergyMonitor**, предназначенного для анализа энергопотребления промышленных объектов компании **Эн+ Диджитал**. Вам необходимо спроектировать и реализовать систему тестирования данного модуля, обеспечивающую проверку корректности бизнес-логики и устойчивости работы функций.

Вы выступаете в роли **инженера-разработчика**, отвечающего за качество программного кода.

Задание

Часть 1. Анализ и проектирование тестирования

1. Кратко опишите функциональное назначение тестируемого модуля.
 2. Определите:
 - применимые **уровни тестирования**;
 - соответствующие **виды тестирования**.
 3. Сформируйте перечень тестируемых сценариев, включая:
 - корректные входные данные;
 - граничные случаи;
 - потенциально ошибочные ситуации.
 4. Опишите риски для системы при отсутствии или неполном тестировании.
-

Часть 2. Реализация автоматизированных тестов

5. Создайте структуру проекта для тестирования (исходный код и каталог с тестами).
6. Реализуйте:
 - модульные тесты для отдельных функций;
 - не менее одного интеграционного теста для проверки взаимодействия функций.
7. Используйте возможности PyTest:
 -

- assert-проверки;
 - параметризацию тестов;
 - осмысленные имена тестовых функций.
-

Часть 3. Анализ и оформление результатов

8. Запустите тесты и зафиксируйте результаты выполнения.
 9. Проанализируйте:
 -
 - какие дефекты выявляются тестами;
 - какие сценарии требуют дополнительного покрытия.
 10. Сформулируйте вывод о готовности модуля к использованию и дальнейшему развитию.
-

Требования к оформлению и сдаче

- язык программирования: **Python**;
 - фреймворк тестирования: **PyTest**;
 - форма сдачи:
 -
 - Git-репозиторий **или**
 - архив с проектом и отчётом;
 - отчёт оформляется в формате PDF / DOCX / Markdown и включает:
 -
 - описание проекта;
 - обоснование выбора видов и уровней тестирования;
 - краткие выводы.
-

Критерии оценки

- корректность применения понятий «виды тестирования» и «уровни тестирования»;
- полнота и качество тестовых сценариев;
- работоспособность и читаемость тестового кода;
- использование возможностей PyTest;
- аргументированность аналитической части.

№ 28. Кейс. Проектирование тест-дизайна и управление дефектами.

Домашняя работа по проекту

Тема

КЕЙС. Проектирование тест-дизайна и управление дефектами для программного модуля (Эн+ Диджитал)

Цель домашней работы

Закрепить практические навыки применения техник тест-дизайна, разработки автомат-

изированных тестов и оформления дефектов в системе управления задачами (Jira) при тестировании программного обеспечения, используемого в промышленной ИТ-компании.

Описание проекта

В рамках учебного проекта вы работаете с программным модулем сервиса **EnergyMonitor**, предназначенного для анализа и обработки данных об энергопотреблении. Модуль реализован на языке Python и содержит как корректные участки логики, так и потенциальные дефекты.

Вы выполняете роль **инженера-разработчика / QA-инженера** компании **Эн+Диджитал**, отвечающего за качество программного продукта.

Задание

Часть 1. Тест-дизайн

1. На основе исходного кода модуля:
 - проанализируйте его функциональность;
 - определите ключевые точки риска.
 2. Разработайте **чек-лист или набор тест-кейсов**, используя техники тест-дизайна:
 - эквивалентное разбиение;
 - анализ граничных значений;
 - негативное тестирование.
 3. Для каждого пункта чек-листа / тест-кейса укажите:
 - тестируемую функцию;
 - входные данные;
 - ожидаемый результат.
-

Часть 2. Реализация автоматизированных тестов

4. Реализуйте автоматизированные тесты на **Python с использованием PyTest**:
 - модульные тесты для отдельных функций;
 - не менее одного интеграционного теста.
5. В тестах обязательно:
 - использовать `assert`;
 - применить параметризацию;
 - покрыть позитивные и негативные сценарии.

Часть 3. Управление дефектами

6. По результатам выполнения тестов:
 - выявите дефекты в программном коде;
 - для каждого дефекта оформите **баг-репорт**.

7. Каждый баг-репорт должен содержать:

- заголовок (Summary);
- шаги воспроизведения (Steps to Reproduce);
- ожидаемый и фактический результат;
- окружение;
- приоритет и критичность.

8. Оформите баг-репорты в **Jira** (или в виде макета баг-репорта, если доступ к Jira отсутствует).

Часть 4. Анализ и выводы

9. Кратко опишите:

- какие дефекты были выявлены с помощью тест-дизайна;
- какие дефекты удалось обнаружить только при автоматизированном тестировании;
- почему корректное оформление баг-репортов важно для командной разработки.

Требования к оформлению и сдаче

- язык программирования: **Python**;
- фреймворк тестирования: **PyTest**;
- формат сдачи:
 - Git-репозиторий с кодом и тестами **или**
 - архив с проектом;
- отчёт (1–2 страницы) в формате PDF / DOCX / Markdown, включающий:
 - описание тест-дизайна;
 - примеры баг-репортов;
 - выводы.

Критерии оценки

- корректность применения техник тест-дизайна;
- полнота и качество тестового покрытия;
- корректность и читаемость автоматизированных тестов;
- качество оформления баг-репортов;
- аргументированность выводов.

2.3.14. Задачи для оценки компетенции «ПК-2 ППК-Р2.1»

№ 29. Инструменты и автоматизация тестирования.

Задача 1. Разработка тестовых наборов данных и планирование тестирования
(Индикатор: *Разрабатывает тестовые наборы данных*)

Цель: научиться создавать системные тестовые данные, включая позитивные, негативные и граничные случаи.

Задание

Проанализируйте предоставленный код модуля `LogAnalyzer` (его сигнатуры методов) и спецификацию формата `Common Log Format`.

Создайте директорию `test_data` и внутри неё:

`valid_access.log` – файл с корректными строками логов (минимум 1000 строк, сгенерируйте с помощью скрипта или онлайн-генератора).

`invalid_format.log` – файл, содержащий строки с нарушенным форматом (пропущенные кавычки, неверное количество полей, некорректные даты).

`empty.log` – пустой файл.

`huge_file.log` – ссылку на сгенерированный файл размером >100 МБ (можно не загружать, но описать, как его создать).

Напишите краткий план тестирования (Test Plan) в формате Markdown, описывающий, какие аспекты модуля будут проверяться какими тестовыми данными и почему.

Задача 2. Проверка работоспособности и создание автоматизированных тестов (Индикатор: *Проверяет работоспособность*)

Цель: написать модульные и интеграционные тесты с использованием фреймворков (pytest/JUnit).

Задание

Используя созданные в Задаче 1 тестовые данные, напишите модульные тесты (минимум 5) с использованием `pytest` для ключевых функций модуля (например, парсинг строки, подсчет статистики).

Напишите 2-3 интеграционных теста, которые запускают основной сценарий работы модуля на файлах `valid_access.log` и `invalid_format.log`, проверяя не только результат, но и корректную обработку ошибок (например, логирование проблемных строк без падения).

Создайте конфигурационный файл для CI (например, `.gitlab-ci.yml` или GitHub Actions), который будет автоматически запускать ваши тесты при пуше в репозиторий.

Задача 3. Исправление дефектов и работа с Bug Tracker (Индикатор: *Исправляет дефекты... зафиксированных в базе данных дефектов*)

Цель: потренировать workflow исправления багов, заведенных в системе.

Задание

Преподаватель (или заранее подготовленный скрипт) создает в issue tracker (GitHub Issues, YouTrack, Jira) 3-4 конкретных дефекта на основе вашего же кода (например, «Утечка памяти при обработке большого файла», «Некорректный парсинг User-Agent с пробелами», «Падение при файле в кодировке UTF-8 с BOM»).

Студент должен для каждого дефекта:

Воспроизвести ошибку на своих тестовых данных.

Создать отдельную ветку в git для исправления (`fix/issue-<номер>`).

Исправить код.

Написать или обновить тест, который теперь покрывает этот исправленный кейс.

Создать Pull Request (Merge Request), указав в описании `Closes #<номер_issue>`.

Важно: К каждому PR должен быть приложен результат запуска тестов и статического анализатора (см. след. задачу).

Задача 4. Инспекция и рефакторинг кода (Индикатор: *Выполняет рефакторинг и инспекцию программного кода*)

Цель: навыки работы со статическими анализаторами и осмысленного рефакторинга.

Задание

Проведите статический анализ предоставленного/исправленного кода с помощью pylint (или sonarlint, checkstyle для Java). Сформируйте отчет.

Проведите код-ревью кода своего одноклассника (обменяйтесь Pull Request из Задачи 3). Оформите ревью в виде комментариев в PR, используя принципы конструктивной критики.

На основе результатов анализа и ревью выполните рефакторинг своего собственного кода, не меняя его внешнего поведения. Цели: улучшение читаемости, устранение code smells (дублирование, слишком длинные методы), соблюдение PEP8/Google Java Style. Рефакторинг оформите отдельным коммитом с пояснениями.

Задача 5. Мониторинг производительности и оптимизация (Индикаторы: *Выполняет мониторинг производительности, Выполняет оптимизацию программного кода*)

Цель: навыки профилирования и оптимизации «узких мест».

Задание.:

Напишите скрипт-бенчмарк, который замеряет время выполнения и потребление памяти вашего модуля на файле `huge_file.log` (или его уменьшенной копии).

Используя профилировщик (cProfile + snakeviz для Python, async-profiler/VisualVM для Java), найдите 2-3 основных «узких места» в коде (например, неоптимальные структуры данных, конкатенация строк в цикле).

Выполните оптимизацию кода на основе данных профилировщика. Добейтесь измеримого улучшения (минимум 20% по времени или памяти на бенчмарке). Предоставьте графики/отчеты «до» и «после».

Задача 6. Использование ИИ для анализа и оптимизации кода (Индикаторы: *Использует ИИ для анализа и отладки кода, Оптимизирует код с помощью ИИ*)

Цель: навыки эффективного взаимодействия с ИИ (ChatGPT, GitHub Copilot, Claude) как с инженерным ассистентом.

Задание

Анализ и отладка: Выберите один неочевидный дефект из Задачи 3 или создайте новый. Сымитируйте ситуацию, когда вы не понимаете причину. С помощью ИИ-помощника (предоставьте ему код, стек-трейч ошибки, контекст) проведите сессию отладки. Предоставьте полный диалог (prompts и responses) и итоговое решение.

Оптимизация с ИИ: Сравните свои оптимизации из Задачи 5 с рекомендациями ИИ. Дайте ИИ задачу: «Проанализируй этот код на Python для парсинга логов. Предложи 3 конкретные оптимизации для повышения скорости работы и снижения потребления памяти. Объясни, почему каждая оптимизация будет работать». Оцените предложения ИИ: какие были полезны, какие нет, и почему.

Написание тестов с ИИ: Используя ИИ, сгенерируйте дополнительные юнит-тесты с высокой покрывающей способностью (boundary value, edge cases) для самого сложного метода вашего модуля. Проанализируйте сгенерированный код на адекватность.

Задача 7. Руководство процессом проверки (Индикатор: *Руководит проверкой*)

работоспособности)

Цель: освоить роль тест-лида или ответственного за качество.

Задание (проектное, может быть групповым)

Вам назначена роль руководителя проверки работоспособности для всего финального проекта (LogAnalyzer).

Составьте итоговый **Отчет о качестве (Quality Report)** по модулю, включив в него:

Сводку по выполненным тестам (процент покрытия, результаты прогона).

Сводку по исправленным дефектам (статистика из Issue Tracker).

Результаты статического анализа и код-ревью (ключевые замечания и их статус).

Результаты нагрузочного тестирования и оптимизации (метрики «до/после»).

Рекомендацию о готовности к релизу: на основе собранных данных обоснуйте, можно ли выпускать текущую версию модуля, или нужны доработки. Если нужны, то какие именно (приоритет и критичность).

2.3.15. Задачи для оценки компетенции «ПК-2 ППК-Р2.2»

№ 30. Инструменты и автоматизация тестирования.

Задача 1. Разработка тестовых наборов данных и планирование тестирования (Индикатор: Разрабатывает тестовые наборы данных)

Цель: научиться создавать системные тестовые данные, включая позитивные, негативные и граничные случаи.

Задание

Проанализируйте предоставленный код модуля LogAnalyzer (его сигнатуры методов) и спецификацию формата Common Log Format.

Создайте директорию `test_data` и внутри неё:

`valid_access.log` – файл с корректными строками логов (минимум 1000 строк, сгенерируйте с помощью скрипта или онлайн-генератора).

`invalid_format.log` – файл, содержащий строки с нарушенным форматом (пропущенные кавычки, неверное количество полей, некорректные даты).

`empty.log` – пустой файл.

`huge_file.log` – ссылку на сгенерированный файл размером >100 МБ (можно не загружать, но описать, как его создать).

Напишите краткий план тестирования (Test Plan) в формате Markdown, описывающий, какие аспекты модуля будут проверяться какими тестовыми данными и почему.

Задача 2. Проверка работоспособности и создание автоматизированных тестов (Индикатор: Проверяет работоспособность)

Цель: написать модульные и интеграционные тесты с использованием фреймворков (pytest/JUnit).

Задание

Используя созданные в Задаче 1 тестовые данные, напишите модульные тесты (минимум 5) с использованием `pytest` для ключевых функций модуля (например, парсинг строки, подсчет статистики).

Напишите 2-3 интеграционных теста, которые запускают основной сценарий работы модуля на файлах `valid_access.log` и `invalid_format.log`, проверяя не только результат,

но и корректную обработку ошибок (например, логирование проблемных строк без падения).

Создайте конфигурационный файл для CI (например, `.gitlab-ci.yml` или GitHub Actions), который будет автоматически запускать ваши тесты при пуше в репозиторий.

Задача 3. Исправление дефектов и работа с Bug Tracker (Индикатор: *Исправляет дефекты... зафиксированных в базе данных дефектов*)

Цель: потренировать workflow исправления багов, заведенных в системе.

Задание

Преподаватель (или заранее подготовленный скрипт) создает в issue tracker (GitHub Issues, YouTrack, Jira) 3-4 конкретных дефекта на основе вашего же кода (например, «Утечка памяти при обработке большого файла», «Некорректный парсинг User-Agent с пробелами», «Падение при файле в кодировке UTF-8 с BOM»).

Студент должен для каждого дефекта:

Воспроизвести ошибку на своих тестовых данных.

Создать отдельную ветку в git для исправления (`fix/issue- $\langle$ номер $\rangle$`).

Исправить код.

Написать или обновить тест, который теперь покрывает этот исправленный кейс.

Создать Pull Request (Merge Request), указав в описании `Closes # $\langle$ номер_issue $\rangle$` .

Важно: К каждому PR должен быть приложен результат запуска тестов и статического анализатора (см. след. задачу).

Задача 4. Инспекция и рефакторинг кода (Индикатор: *Выполняет рефакторинг и инспекцию программного кода*)

Цель: навыки работы со статическими анализаторами и осмысленного рефакторинга.

Задание

Проведите статический анализ предоставленного/исправленного кода с помощью `pylint` (или `sonarlint`, `checkstyle` для Java). Сформируйте отчет.

Проведите код-ревью кода своего одногруппника (обменяйтесь Pull Request из Задачи 3). Оформите ревью в виде комментариев в PR, используя принципы конструктивной критики.

На основе результатов анализа и ревью выполните рефакторинг своего собственного кода, не меняя его внешнего поведения. Цели: улучшение читаемости, устранение code smells (дублирование, слишком длинные методы), соблюдение PEP8/Google Java Style. Рефакторинг оформите отдельным коммитом с пояснениями.

Задача 5. Мониторинг производительности и оптимизация (Индикаторы: *Выполняет мониторинг производительности, Выполняет оптимизацию программного кода*)

Цель: навыки профилирования и оптимизации «узких мест».

Задание.:

Напишите скрипт-бенчмарк, который замеряет время выполнения и потребление памяти вашего модуля на файле `huge_file.log` (или его уменьшенной копии).

Используя профилировщик (`cProfile` + `snakeviz` для Python, `async-profiler/VisualVM` для Java), найдите 2-3 основных «узких места» в коде (например, неоптимальные структуры данных, конкатенация строк в цикле).

Выполните оптимизацию кода на основе данных профилировщика. Добейтесь

измеримого улучшения (минимум 20% по времени или памяти на бенчмарке). Предоставьте графики/отчеты «до» и «после».

Задача 6. Использование ИИ для анализа и оптимизации кода (Индикаторы: *Использует ИИ для анализа и отладки кода, Оптимизирует код с помощью ИИ*)

Цель: навыки эффективного взаимодействия с ИИ (ChatGPT, GitHub Copilot, Claude) как с инженерным ассистентом.

Задание

Анализ и отладка: Выберите один неочевидный дефект из Задачи 3 или создайте новый. Сымитируйте ситуацию, когда вы не понимаете причину. С помощью ИИ-помощника (предоставьте ему код, стек-трейч ошибки, контекст) проведите сессию отладки. Предоставьте полный диалог (prompts и responses) и итоговое решение.

Оптимизация с ИИ: Сравните свои оптимизации из Задачи 5 с рекомендациями ИИ. Дайте ИИ задачу: «Проанализируй этот код на Python для парсинга логов. Предложи 3 конкретные оптимизации для повышения скорости работы и снижения потребления памяти. Объясни, почему каждая оптимизация будет работать». Оцените предложения ИИ: какие были полезны, какие нет, и почему.

Написание тестов с ИИ: Используя ИИ, сгенерируйте дополнительные юнит-тесты с высокой покрывающей способностью (boundary value, edge cases) для самого сложного метода вашего модуля. Проанализируйте сгенерированный код на адекватность.

Задача 7. Руководство процессом проверки (Индикатор: *Руководит проверкой работоспособности*)

Цель: освоить роль тест-лида или ответственного за качество.

Задание (проектное, может быть групповым)

Вам назначена роль руководителя проверки работоспособности для всего финального проекта (LogAnalyzer).

Составьте итоговый **Отчет о качестве (Quality Report)** по модулю, включив в него:

Сводку по выполненным тестам (процент покрытия, результаты прогона).

Сводку по исправленным дефектам (статистика из Issue Tracker).

Результаты статического анализа и код-ревью (ключевые замечания и их статус).

Результаты нагрузочного тестирования и оптимизации (метрики «до/после»).

Рекомендацию о готовности к релизу: на основе собранных данных обоснуйте, можно ли выпускать текущую версию модуля, или нужны доработки. Если нужны, то какие именно (приоритет и критичность).

2.3.16. Задачи для оценки компетенции «ПК-2 ППК-Р2.3»

№ 31. Инструменты и автоматизация тестирования.

Задача 1. Разработка тестовых наборов данных и планирование тестирования (Индикатор: *Разрабатывает тестовые наборы данных*)

Цель: научиться создавать системные тестовые данные, включая позитивные, негативные и граничные случаи.

Задание

Проанализируйте предоставленный код модуля LogAnalyzer (его сигнатуры методов) и спецификацию формата Common Log Format.

Создайте директорию `test_data` и внутри неё:

`valid_access.log` – файл с корректными строками логов (минимум 1000 строк, сгенерируйте с помощью скрипта или онлайн-генератора).

`invalid_format.log` – файл, содержащий строки с нарушенным форматом (пропущенные кавычки, неверное количество полей, некорректные даты).

`empty.log` – пустой файл.

`huge_file.log` – ссылку на сгенерированный файл размером >100 МБ (можно не загружать, но описать, как его создать).

Напишите краткий план тестирования (Test Plan) в формате Markdown, описывающий, какие аспекты модуля будут проверяться какими тестовыми данными и почему.

Задача 2. Проверка работоспособности и создание автоматизированных тестов (Индикатор: *Проверяет работоспособность*)

Цель: написать модульные и интеграционные тесты с использованием фреймворков (pytest/JUnit).

Задание

Используя созданные в Задаче 1 тестовые данные, напишите модульные тесты (минимум 5) с использованием `pytest` для ключевых функций модуля (например, парсинг строки, подсчет статистики).

Напишите 2-3 интеграционных теста, которые запускают основной сценарий работы модуля на файлах `valid_access.log` и `invalid_format.log`, проверяя не только результат, но и корректную обработку ошибок (например, логирование проблемных строк без падения).

Создайте конфигурационный файл для CI (например, `.gitlab-ci.yml` или GitHub Actions), который будет автоматически запускать ваши тесты при пуше в репозиторий.

Задача 3. Исправление дефектов и работа с Bug Tracker (Индикатор: *Исправляет дефекты... зафиксированных в базе данных дефектов*)

Цель: потренировать workflow исправления багов, заведенных в системе.

Задание

Преподаватель (или заранее подготовленный скрипт) создает в issue tracker (GitHub Issues, YouTrack, Jira) 3-4 конкретных дефекта на основе вашего же кода (например, «Утечка памяти при обработке большого файла», «Некорректный парсинг User-Agent с пробелами», «Падение при файле в кодировке UTF-8 с BOM»).

Студент должен для каждого дефекта:

Воспроизвести ошибку на своих тестовых данных.

Создать отдельную ветку в git для исправления (`fix/issue-<номер>`).

Исправить код.

Написать или обновить тест, который теперь покрывает этот исправленный кейс.

Создать Pull Request (Merge Request), указав в описании `Closes #<номер_issue>`.

Важно: К каждому PR должен быть приложен результат запуска тестов и статического анализатора (см. след. задачу).

Задача 4. Инспекция и рефакторинг кода (Индикатор: *Выполняет рефакторинг и инспекцию программного кода*)

Цель: навыки работы со статическими анализаторами и осмысленного рефакторинга.

Задание

Проведите статический анализ предоставленного/исправленного кода с помощью `pylint` (или `sonarlint`, `checkstyle` для Java). Сформируйте отчет.

Проведите код-ревью кода своего одногруппника (обменяйтесь Pull Request из Задачи 3). Оформите ревью в виде комментариев в PR, используя принципы конструктивной критики.

На основе результатов анализа и ревью выполните рефакторинг своего собственного кода, не меняя его внешнего поведения. Цели: улучшение читаемости, устранение code smells (дублирование, слишком длинные методы), соблюдение PEP8/Google Java Style. Рефакторинг оформите отдельным коммитом с пояснениями.

Задача 5. Мониторинг производительности и оптимизация
(Индикаторы: *Выполняет мониторинг производительности, Выполняет оптимизацию программного кода*)

Цель: навыки профилирования и оптимизации «узких мест».

Задание.:

Напишите скрипт-бенчмарк, который замеряет время выполнения и потребление памяти вашего модуля на файле `huge_file.log` (или его уменьшенной копии).

Используя профилировщик (`cProfile` + `snakeviz` для Python, `async-profiler`/`VisualVM` для Java), найдите 2-3 основных «узких места» в коде (например, неоптимальные структуры данных, конкатенация строк в цикле).

Выполните оптимизацию кода на основе данных профилировщика. Добейтесь измеримого улучшения (минимум 20% по времени или памяти на бенчмарке). Предоставьте графики/отчеты «до» и «после».

Задача 6. Использование ИИ для анализа и оптимизации кода
(Индикаторы: *Использует ИИ для анализа и отладки кода, Оптимизирует код с помощью ИИ*)

Цель: навыки эффективного взаимодействия с ИИ (ChatGPT, GitHub Copilot, Claude) как с инженерным ассистентом.

Задание

Анализ и отладка: Выберите один неочевидный дефект из Задачи 3 или создайте новый. Сымитируйте ситуацию, когда вы не понимаете причину. С помощью ИИ-помощника (предоставьте ему код, стек-трейс ошибки, контекст) проведите сессию отладки. Предоставьте полный диалог (prompts и responses) и итоговое решение.

Оптимизация с ИИ: Сравните свои оптимизации из Задачи 5 с рекомендациями ИИ. Дайте ИИ задачу: «Проанализируй этот код на Python для парсинга логов. Предложи 3 конкретные оптимизации для повышения скорости работы и снижения потребления памяти. Объясни, почему каждая оптимизация будет работать». Оцените предложения ИИ: какие были полезны, какие нет, и почему.

Написание тестов с ИИ: Используя ИИ, сгенерируйте дополнительные юнит-тесты с высокой покрывающей способностью (boundary value, edge cases) для самого сложного метода вашего модуля. Проанализируйте сгенерированный код на адекватность.

Задача 7. Руководство процессом проверки (Индикатор: *Руководит проверкой работоспособности*)

Цель: освоить роль тест-лида или ответственного за качество.

Задание (проектное, может быть групповым)

Вам назначена роль руководителя проверки работоспособности для всего финального проекта (LogAnalyzer).

Составьте итоговый **Отчет о качестве (Quality Report)** по модулю, включив в него:

Сводку по выполненным тестам (процент покрытия, результаты прогона).

Сводку по исправленным дефектам (статистика из Issue Tracker).

Результаты статического анализа и код-ревью (ключевые замечания и их статус).

Результаты нагрузочного тестирования и оптимизации (метрики «до/после»).

Рекомендацию о готовности к релизу: на основе собранных данных обоснуйте, можно ли выпускать текущую версию модуля, или нужны доработки. Если нужны, то какие именно (приоритет и критичность).

2.3.17. Задачи для оценки компетенции «ПК-2 ППК-Р2.4»

№ 32. Инструменты и автоматизация тестирования.

Задача 1. Разработка тестовых наборов данных и планирование тестирования (Индикатор: Разрабатывает тестовые наборы данных)

Цель: научиться создавать системные тестовые данные, включая позитивные, негативные и граничные случаи.

Задание

Проанализируйте предоставленный код модуля LogAnalyzer (его сигнатуры методов) и спецификацию формата Common Log Format.

Создайте директорию `test_data` и внутри неё:

`valid_access.log` – файл с корректными строками логов (минимум 1000 строк, сгенерируйте с помощью скрипта или онлайн-генератора).

`invalid_format.log` – файл, содержащий строки с нарушенным форматом (пропущенные кавычки, неверное количество полей, некорректные даты).

`empty.log` – пустой файл.

`huge_file.log` – ссылку на сгенерированный файл размером >100 МБ (можно не загружать, но описать, как его создать).

Напишите краткий план тестирования (Test Plan) в формате Markdown, описывающий, какие аспекты модуля будут проверяться какими тестовыми данными и почему.

Задача 2. Проверка работоспособности и создание автоматизированных тестов (Индикатор: Проверяет работоспособность)

Цель: написать модульные и интеграционные тесты с использованием фреймворков (pytest/JUnit).

Задание

Используя созданные в Задаче 1 тестовые данные, напишите модульные тесты (минимум 5) с использованием `pytest` для ключевых функций модуля (например, парсинг строки, подсчет статистики).

Напишите 2-3 интеграционных теста, которые запускают основной сценарий работы модуля на файлах `valid_access.log` и `invalid_format.log`, проверяя не только результат, но и корректную обработку ошибок (например, логирование проблемных строк без падения).

Создайте конфигурационный файл для CI (например, `.gitlab-ci.yml` или GitHub

Actions), который будет автоматически запускать ваши тесты при пуше в репозиторий.

Задача 3. Исправление дефектов и работа с Bug Tracker (Индикатор: *Исправляет дефекты... зафиксированных в базе данных дефектов*)

Цель: потренировать workflow исправления багов, заведенных в системе.

Задание

Преподаватель (или заранее подготовленный скрипт) создает в issue tracker (GitHub Issues, YouTrack, Jira) 3-4 конкретных дефекта на основе вашего же кода (например, «Утечка памяти при обработке большого файла», «Некорректный парсинг User-Agent с пробелами», «Падение при файле в кодировке UTF-8 с BOM»).

Студент должен для каждого дефекта:

Воспроизвести ошибку на своих тестовых данных.

Создать отдельную ветку в git для исправления (`fix/issue-<номер>`).

Исправить код.

Написать или обновить тест, который теперь покрывает этот исправленный кейс.

Создать Pull Request (Merge Request), указав в описании `Closes #<номер_issue>`.

Важно: К каждому PR должен быть приложен результат запуска тестов и статического анализатора (см. след. задачу).

Задача 4. Инспекция и рефакторинг кода (Индикатор: *Выполняет рефакторинг и инспекцию программного кода*)

Цель: навыки работы со статическими анализаторами и осмысленного рефакторинга.

Задание

Проведите статический анализ предоставленного/исправленного кода с помощью pylint (или sonarlint, checkstyle для Java). Сформируйте отчет.

Проведите код-ревью кода своего одногруппника (обменяйтесь Pull Request из Задачи 3). Оформите ревью в виде комментариев в PR, используя принципы конструктивной критики.

На основе результатов анализа и ревью выполните рефакторинг своего собственного кода, не меняя его внешнего поведения. Цели: улучшение читаемости, устранение code smells (дублирование, слишком длинные методы), соблюдение PEP8/Google Java Style. Рефакторинг оформите отдельным коммитом с пояснениями.

Задача 5. Мониторинг производительности и оптимизация (Индикаторы: *Выполняет мониторинг производительности, Выполняет оптимизацию программного кода*)

Цель: навыки профилирования и оптимизации «узких мест».

Задание.:

Напишите скрипт-бенчмарк, который замеряет время выполнения и потребление памяти вашего модуля на файле `huge_file.log` (или его уменьшенной копии).

Используя профилировщик (`cProfile` + `snakeviz` для Python, `async-profiler/VisualVM` для Java), найдите 2-3 основных «узких места» в коде (например, неоптимальные структуры данных, конкатенация строк в цикле).

Выполните оптимизацию кода на основе данных профилировщика. Добейтесь измеримого улучшения (минимум 20% по времени или памяти на бенчмарке). Предоставьте графики/отчеты «до» и «после».

Задача 6. Использование ИИ для анализа и оптимизации кода

(Индикаторы: *Использует ИИ для анализа и отладки кода, Оптимизирует код с помощью ИИ*)

Цель: навыки эффективного взаимодействия с ИИ (ChatGPT, GitHub Copilot, Claude) как с инженерным ассистентом.

Задание

Анализ и отладка: Выберите один неочевидный дефект из Задачи 3 или создайте новый. Сымитируйте ситуацию, когда вы не понимаете причину. С помощью ИИ-помощника (предоставьте ему код, стек-трейч ошибки, контекст) проведите сессию отладки. Предоставьте полный диалог (prompts и responses) и итоговое решение.

Оптимизация с ИИ: Сравните свои оптимизации из Задачи 5 с рекомендациями ИИ. Дайте ИИ задачу: «Проанализируй этот код на Python для парсинга логов. Предложи 3 конкретные оптимизации для повышения скорости работы и снижения потребления памяти. Объясни, почему каждая оптимизация будет работать». Оцените предложения ИИ: какие были полезны, какие нет, и почему.

Написание тестов с ИИ: Используя ИИ, сгенерируйте дополнительные юнит-тесты с высокой покрывающей способностью (boundary value, edge cases) для самого сложного метода вашего модуля. Проанализируйте сгенерированный код на адекватность.

Задача 7. Руководство процессом проверки (Индикатор: *Руководит проверкой работоспособности*)

Цель: освоить роль тест-лида или ответственного за качество.

Задание (проектное, может быть групповым)

Вам назначена роль руководителя проверки работоспособности для всего финального проекта (LogAnalyzer).

Составьте итоговый **Отчет о качестве (Quality Report)** по модулю, включив в него:

Сводку по выполненным тестам (процент покрытия, результаты прогона).

Сводку по исправленным дефектам (статистика из Issue Tracker).

Результаты статического анализа и код-ревью (ключевые замечания и их статус).

Результаты нагрузочного тестирования и оптимизации (метрики «до/после»).

Рекомендацию о готовности к релизу: на основе собранных данных обоснуйте, можно ли выпускать текущую версию модуля, или нужны доработки. Если нужны, то какие именно (приоритет и критичность).

3. Промежуточная аттестация

3.1. Методические материалы, определяющие процедуру оценивания знаний, умений, навыков и опыта деятельности

Зачет является заключительным этапом процесса формирования компетенций обучающегося при изучении дисциплины и имеет целью проверку и оценку знаний обучающегося по теории, и применению полученных знаний, умений и навыков при решении практических задач.

Зачет проводится по расписанию, сформированному учебно-методическим управлением, в сроки, предусмотренные календарным учебным графиком. Зачет принимается преподавателем, ведущим лекционные занятия.

Зачет проводится только при предъявлении обучающимся зачетной книжки и при условии выполнения всех контрольных мероприятий, предусмотренных учебным планом и

рабочей программой дисциплины. Обучающимся на зачету представляется право выбрать один из билетов. Время подготовки к ответу составляет 30 минут. По истечении установленного времени обучающийся должен ответить на вопросы билета. Результаты зачета оцениваются по четырехбалльной системе и заносятся в зачетно-экзаменационную ведомость и зачетную книжку. В зачетную книжку заносятся только положительные оценки. Подписанный преподавателем экземпляр ведомости сдаётся не позднее следующего дня в деканат.

В случае неявки обучающегося на зачет в зачетно-экзаменационную ведомость делается отметка «не явка». Обучающиеся, не прошедшие промежуточную аттестацию по дисциплине, должны ликвидировать академическую задолженность в установленном локальными нормативными актами порядке.

3.2. Вопросы к зачету

№	Вопрос	Код компетенции
1.	Теоретический вопрос: опишите жизненный цикл дефекта (bug lifecycle) в системе отслеживания. Какие статусы он проходит? Что должно содержаться в хорошем отчете об ошибке? Практическое задание В проекте в Issue Tracker заведен дефект #CONV-101: «При конвертации очень больших сумм (например, 1e30) возвращается некорректный результат (Infinity или ошибка), а не сообщение об ограничении». Воспроизведение: Напишите простой скрипт или используйте Postman, чтобы воспроизвести ошибку. Зафиксируйте факт ошибки (скриншот/логи). Исправление: Найдите в коде сервиса место, отвечающее за проверку входных данных. Исправьте дефект, добавив валидацию максимальной суммы (например, не более 1e12). Исправление оформите через git: создайте ветку fix/CONV-101, коммит, а затем без создания Merge Request. Тест на исправление: Дополните или создайте модульный тест, который проверяет новое поведение (успешная конвертация до лимита, ошибка — сверх лимита).	ПК-2 ППК-Р2.3
2.	Теоретический вопрос: что такое «запах кода» (code smell)? Приведите три примера «запахов», относящихся к методам/функциям, и по одному возможному способу рефакторинга для каждого. Практическое задание Вам предоставлен файл CurrencyService.java (или .py) с методом convert, содержащим «запахи» кода. Инспекция: запустите статический анализатор (SonarLint в IDE или через CLI) на этом файле. Сохраните топ-3 наиболее критических замечания (не связанных с форматированием). Код-ревью: в качестве ревьюера изучите предоставленный фрагмент кода (другого студента или заранее подготовленный). В формате комментариев укажите две конкретные проблемы: одну — касающуюся потенциальной ошибки (например, отсутствие обработки null), вторую — касающуюся читаемости/архитектуры (например, слишком длинный метод). Рефакторинг: выполните рефакторинг метода convert, устранив один выявленный «запах» (например, «Длинный метод» через выделение метода валидации). Код после рефакторинга должен проходить все существующие тесты.	ПК-2 ППК-Р2.4

№	Вопрос	Код компетенции
3.	Теоретический вопрос: опишите стратегию создания тестовых данных (Test Data Design). Какие основные категории/классы эквивалентности данных вы выделите для метода API, который конвертирует сумму из одной валюты в другую? Объясните на примерах. Практическое задание Вам дан эндпоинт POST /api/convert. Параметры: amount (число), from_currency (строка, код ISO), to_currency (строка, код ISO). Создайте структурированный файл test_data.json, содержащий массив JSON-объектов для тестирования этого эндпоинта. Включите не менее 10 тестовых случаев, покрывающих: Валидные данные (положительные, отрицательные, нулевая сумма). Невалидные данные (несуществующие коды валют, некорректный тип amount, отсутствующие поля). Граничные/особые значения (очень большое число, валюта из трех букв, одинаковые from и to). Кратко обоснуйте (в комментариях к файлу) выбор каждого тестового случая.	ПК-2 ППК-Р2.1
4.	Теоретический вопрос: в чем разница между модульным, интеграционным и сквозным (E2E) тестированием? Какой тип тестирования наиболее релевантен для проверки REST API и почему? Практическое задание (ПК-2.2): Используя готовый набор данных из Пакета 1 или создав свой, выполните: Автоматизация в Postman: Создайте коллекцию Postman для эндпоинта /api/convert. Настройте автоматические проверки (assertions) в тестах Tab для: Статус-кода ответа. Структуры JSON-ответа (наличие полей result, rate). Корректности вычисления ($result \approx amount * rate$ для успешных случаев). Интеграционный тест: Напишите один интеграционный тест на Python (pytest + requests) или Java (JUnit + RestAssured), который: Проверяет успешную конвертацию. Проверяет обработку ошибки при несуществующей валюте. Запустите тест и предоставьте отчет о прохождении.	ПК-2 ППК-Р2.2
5.	Теоретический вопрос: опишите жизненный цикл дефекта (bug lifecycle) в системе отслеживания. Какие статусы он проходит? Что должно содержаться в хорошем отчете об ошибке? Практическое задание В проекте в Issue Tracker заведен дефект #CONV-101: «При конвертации очень больших сумм (например, 1e30) возвращается некорректный результат (Infinity или ошибка), а не сообщение об ограничении». Воспроизведение: Напишите простой скрипт или используйте Postman, чтобы воспроизвести ошибку. Зафиксируйте факт ошибки (скриншот/логи). Исправление: Найдите в коде сервиса место, отвечающее за проверку входных данных. Исправьте дефект, добавив валидацию максимальной суммы (например, не более 1e12). Исправление оформите через git: создайте ветку fix/CONV-101, коммит, а затем без создания Merge Request. Тест на исправление: Дополните или создайте модульный тест, который проверяет новое поведение (успешная конвертация до лимита, ошибка — сверх лимита).	ПК-2 ППК-Р2.3

№	Вопрос	Код компетенции
6.	Теоретический вопрос: что такое «запах кода» (code smell)? Приведите три примера «запахов», относящихся к методам/функциям, и по одному возможному способу рефакторинга для каждого. Практическое задание Вам предоставлен файл CurrencyService.java (или .py) с методом convert, содержащим «запахи» кода. Инспекция: запустите статический анализатор (SonarLint в IDE или через CLI) на этом файле. Сохраните топ-3 наиболее критических замечания (не связанных с форматированием). Код-ревью: в качестве ревьюера изучите предоставленный фрагмент кода (другого студента или заранее подготовленный). В формате комментариев укажите две конкретные проблемы: одну — касающуюся потенциальной ошибки (например, отсутствие обработки null), вторую — касающуюся читаемости/архитектуры (например, слишком длинный метод). Рефакторинг: выполните рефакторинг метода convert, устранив один выявленный «запах» (например, «Длинный метод» через выделение метода валидации). Код после рефакторинга должен проходить все существующие тесты.	ПК-2 ППК-Р2.4
7.	Мониторинг производительности Теоретический вопрос: какие метрики являются ключевыми для оценки производительности веб-сервиса? Дайте определение времени отклика (response time), пропускной способности (throughput) и процентилям (p95, p99). Практическое задание Настройка мониторинга: используя инструмент k6, напишите скрипт нагрузочного теста для эндпоинта /api/convert. Сценарий: 15 виртуальных пользователей (VUs) в течение 30 секунд отправляют запросы на конвертацию случайных (но валидных) сумм. Сбор метрик: запустите тест. Сохраните сводный отчет (stdout). Обратите внимание на ключевые метрики: http_req_duration (среднее, p95), http_reqs (общее количество), iterations. Анализ: на основе полученных данных определите, удовлетворяет ли сервис требованиям производительности (Требование: p95 времени ответа < 500 мс). Сформулируйте краткий вывод.	ПК-6 ИП-2.1

№	Вопрос	Код компетенции
8.	Оптимизация кода Теоретический вопрос: Объясните разницу между оптимизацией «до того как» (premature optimization) и обоснованной оптимизацией. На каких данных должна основываться обоснованная оптимизация? Практическое задание В ходе нагрузочного теста выявлено «узкое место» — метод <code>getExchangeRate(from, to)</code>, который каждый раз делает «дорогой» синхронный вызов к внешнему API или читает весь файл с курсами. Профилирование (демо): изучите предоставленный искусственный отчет профилировщика, где видно, что >80% времени тратится в этом методе. Оптимизация: не меняя сигнатуру метода, предложите и реализуйте одну оптимизацию для устранения узкого места. Варианты: добавление in-memory кэша (например, <code>HashMap/dict</code>) на 5 минут, ленивая загрузка данных и т.д. Верификация: запустите упрощенный нагрузочный тест (например, 10 последовательных вызовов) и замерьте среднее время до и после оптимизации. Сделайте вывод об эффективности.	ПК-6 ИП-2.2
9.	Использование ИИ для анализа и отладки Теоретический вопрос: каковы основные преимущества и риски использования ИИ (например, больших языковых моделей) в процессе отладки и анализа кода? Практическое задание. В проекте есть неочевидная ошибка: при определенной последовательности запросов (например, конвертация USD->EUR, затем EUR->JPY) курс для одной из пар может рассчитаться неверно. Постановка задачи ИИ: смоделируйте диалог с ИИ-помощником (например, в ChatGPT). Предоставьте ему: Фрагмент кода ключевого метода. Пример failing-теста или описание симптомов. Попросите проанализировать возможные причины. Анализ ответа: на основе ответа ИИ сформулируйте гипотезу о потенциальной причине ошибки (например, «проблема округления», «неверное обновление кэша»). Локализация: используя гипотезу, найдите в реальном коде потенциально проблемную строку. Укажите ее номер и объясните, почему она может быть причиной.	ПК-8 ППК-Р7.2

№	Вопрос	Код компетенции
10.	Оптимизация кода с помощью ИИ Теоретический вопрос: Как можно использовать ИИ не для генерации кода «с нуля», а для его улучшения (оптимизации)? Приведите пример конкретной задачи по оптимизации, которую можно поручить ИИ. Практическое задание. Вам дан метод <code>validateCurrencyCode(String code)</code>, который работает корректно, но написан неэффективно (например, использует цепочку <code>if-else</code> или регулярное выражение при каждом вызове). Запрос к ИИ: сформулируйте и выполните запрос к ИИ-инструменту (например, GitHub Copilot в чате): «Проанализируй этот метод валидации кода валюты [вставь код]. Предложи две альтернативные, более производительные реализации. Объясни, почему они быстрее». Оценка предложений: из предложенных ИИ вариантов выберите один, который, на ваш взгляд, является лучшим по соотношению производительность/читаемость. Реализация и проверка: замените текущую реализацию метода на выбранный оптимизированный вариант. Убедитесь, что существующие тесты проходят. Кратко аргументируйте свой выбор.	ПК-8 ППК-Р7.3
11.	Руководство проверкой работоспособности Теоретический вопрос: каковы ключевые обязанности и зона ответственности руководителя проверки работоспособности (Test Lead/QA Lead) на этапе подготовки к релизу? Практическое задание. Вы — руководитель проверки для сервиса Currency Converter перед выпуском версии v1.1. Анализ артефактов. Вам предоставлены: Отчет о выполнении тестов (85% passed, 15% failed). Dashboard SonarQube (качество кода: А, есть 2 критические уязвимости). Отчет о нагрузочном тесте (p95 = 620 мс). Список открытых дефектов в Jira (3 штуки, приоритет Medium). Принятие решения: Напишите краткое, но структурированное заключение (Approval/Rejection) на имя менеджера продукта. В заключении: Резюмируйте текущее состояние. Укажите, является ли какая-либо из перечисленных проблем блокирующей для релиза. Почему? Дайте четкую рекомендацию: Выпускать или Отложить выпуск. Если отложить, укажите обязательные условия для снятия блокировки.	ПК-14 РГ-1 .2

№	Вопрос	Код компетенции
12.	Руководство интеграцией. Теоретический вопрос: что такое непрерывная интеграция (CI)? Опишите типичный пайплайн CI для проекта на Python/Java. Какие этапы должны в него входить для обеспечения качества? Практическое задание. Ваша задача — настроить процесс интеграции для нового модуля NotificationService, который должен быть подключен к основному сервису Currency Converter. Проектирование пайплайна: напишите конфигурационный файл CI (.gitlab-ci.yml или GitHub Actions) для проекта Currency Converter. Пайплайн должен включать последовательные стадии: build — сборка/установка зависимостей. static_analysis — запуск линтера и анализатора уязвимостей. unit_test — запуск модульных тестов с генерацией отчета о покрытии. integration_test — запуск интеграционных тестов (с поднятием тестовой БД и зависимостей в Docker). Интеграция модуля: в коде основного сервиса есть заглушка для отправки уведомления. Ваше задание — не код, а планирование. Опишите шаги (checklist) для безопасного включения реального NotificationService в пайплайн интеграции. Включите пункты о: тестировании в изоляции, feature-флагах, мониторинге ошибок после деплоя.	ПК-14 РГ-1 .3

3.3. Тематика курсовых работ

По данной дисциплине выполнение курсовых проектов (работ) не предусматривается.

3.4. Материалы для компьютерного тестирования обучающихся

Общие критерии оценивания

Процент правильных ответов	Оценка
91% – 100%	5 (отлично)
81% – 90%	4 (хорошо)
71% – 80%	3 (удовлетворительно)
Менее 70%	2 (неудовлетворительно)

Соответствие вопросов теста индикаторам формируемых и оцениваемых компетенций

№ вопроса в тесте	Код индикатора компетенции
1	ПК-2 ППК-Р2.4
2	ПК-2 ППК-Р2.2
3	ПК-2 ППК-Р2.3
4	ПК-8 ППК-Р7.3
5	ПК-2 ППК-Р2.1
6	ПК-2 ППК-Р2.2
7	ПК-14 РГ-1 .3
8	ПК-6 ИП-2.2
9	ПК-8 ППК-Р7.2

№ вопроса в тесте	Код индикатора компетенции
10	ПК-6 ИП-2.1
11	ПК-14 РГ-1 .2

Ключ ответов

№ вопроса в тесте	Номер ответа (или ответ, или соответствие)
1	d
2	b
3	a
4	a
5	b
6	d
7	b
8	c
9	d
10	c
11	d

Перечень тестовых вопросов

№ 1. Задание с единичным выбором. Выберите один правильный ответ.

Какое из перечисленных действий относится к рефакторингу, а не к исправлению дефекта?

- a. Добавление проверки входных данных
- b. Изменение условия ветвления для корректной работы
- c. Исправление деления на ноль
- d. Переименование переменных для улучшения читаемости без изменения логики

№ 2. Задание с единичным выбором. Выберите один правильный ответ.

Какое действие относится непосредственно к проверке работоспособности программного обеспечения?

- a. Оптимизация алгоритма
- b. Запуск программы с тестовыми данными и анализ фактического результата
- c. Рефакторинг кода
- d. Написание технического задания

№ 3. Задание с единичным выбором. Выберите один правильный ответ.

Какой порядок действий является наиболее корректным при исправлении дефекта, зафиксированного в системе баг-трекинга?

- a. Проанализировать баг → воспроизвести → исправить → проверить → закрыть
- b. Исправить код → закрыть баг → написать тест
- c. Исправить код без воспроизведения дефекта
- d. Закрыть баг → исправить код → проверить

№ 4. Задание с единичным выбором. Выберите один правильный ответ.

Какой результат свидетельствует об эффективном использовании ИИ для оптимизации

кода?

- a. Снижено время выполнения алгоритма при сохранении корректности
- b. ИИ переписал код без объяснений
- c. Код стал короче, но работает медленнее
- d. Добавлены комментарии

№ 5. Задание с единичным выбором. Выберите один правильный ответ.

Какой набор входных данных наиболее корректно использовать для проверки функции, вычисляющей среднее значение массива чисел?

- a. Только массив положительных чисел
- b. Набор данных, включающий нормальные, граничные и некорректные значения
- c. Массив с одинаковыми значениями
- d. Пустой массив

№ 6. Задание с единичным выбором. Выберите один правильный ответ.

Какое действие относится непосредственно к проверке работоспособности программного обеспечения?

- a. Оптимизация алгоритма
- b. Рефакторинг кода
- c. Написание технического задания
- d. Запуск программы с тестовыми данными и анализ фактического результата

№ 7. Задание с единичным выбором. Выберите один правильный ответ.

Какое решение должен принять руководитель при интеграции нескольких программных модулей?

- a. Передать интеграцию одному разработчику без проверки
- b. Назначить интеграционное тестирование и определить порядок объединения модулей
- c. Отложить тестирование до релиза
- d. Проверить каждый модуль только отдельно

№ 8. Задание с единичным выбором. Выберите один правильный ответ.

Какое действие является примером оптимизации программного кода?

- a. Добавление логирования
- b. Добавление комментариев
- c. Замена линейного поиска на хеш-таблицу
- d. Переименование функций

№ 9. Задание с единичным выбором. Выберите один правильный ответ.

В каком случае использование ИИ наиболее оправдано для анализа и отладки кода?

- a. Для запуска программы
- b. Для форматирования кода
- c. Для поиска синтаксических ошибок компилятором
- d. Для анализа стектрейса и предложения возможной причины ошибки

№ 10. Задание с единичным выбором. Выберите один правильный ответ.

Какой показатель наиболее непосредственно используется при мониторинге производи-

тельности программного обеспечения?

- a. Количество строк кода
- b. Стиль оформления кода
- c. Время отклика и использование ресурсов
- d. Количество комментариев

№ 11. Задание с единственным выбором. Выберите один правильный ответ.

Какое действие относится к руководству проверкой работоспособности программного обеспечения, а не к индивидуальному тестированию?

- a. Исправление одного дефекта
- b. Запуск модульного теста
- c. Написание одного тест-кейса
- d. Планирование этапов тестирования и распределение задач в команде