МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«МИРЭА – РОССИЙСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

Отчет по практической работе 3

по дисциплине «Тестирование и верификация ПО»

Выполнили: Карасев Е.В.
Студенты группы **ИКБО-43-23**

Проверил:
Ильичев Г.П.

2025 г.

СОДЕРЖАНИЕ

1. Цель и задачи работы

Цель работы — получение практических навыков применения современных методологий разработки программного обеспечения, основанных на тестировании, для создания надежного и качественного продукта.

Задачи: изучить теоретические основы методологий TDD, ATDD, BDD, SDD.

Разработать приложение «Онлайн форум» (вариант №27), реализующее базовую логику: создание темы, публикация ответов, поиск по темам.

Последовательно применить все четыре методологии для разработки и верификации функциональности.

Проанализировать вклад каждого подхода в итоговое качество продукта.

Подготовить отчет, документирующий все этапы работы.

Приложение представляет собой простую консольную викторину. Основные сущности: Topic — тема форума (содержит заголовок, автора, описание и список ответов) и Post — сообщение (ответ пользователя в теме).

Функциональные требования:

- Система должна предоставлять возможность создания новой темы, указав её заголовок, автора и описание.
- Пользователь может добавлять ответы к уже существующим темам.
- После публикации ответ должен сохраняться и отображаться внутри соответствующей темы.
- После публикации ответ должен сохраняться и отображаться внутри соответствующей темы.
- В случае попытки публикации ответа в несуществующую тему система должна выдавать ошибку.
- При запуске приложение работает в консольном режиме: принимает команды (create;, reply;, find;, exit) и выводит результаты операций пользователю

2. Практическая часть

2.1 Этап 1. Разработка на основе тестов (TDD)

На первом этапе каждый участник команды разработал свой программный модуль, содержащий преднамеренную логическую ошибку для последующего обнаружения в процессе тестирования.

Шаг 1: создание тестов (Красный этап)

Первым делом были написаны юнит-тесты, которые описывают желаемое поведение системы. Использовался встроенный фреймворк unittest.

test_main.py

```python
# tests/test_forum_tdd.py
import pytest
from main import create_topic, add_reply, get_topic, search_topics,  _reset_state_for_tests

@pytest.fixture(autouse=True)
def _isolate():
    _reset_state_for_tests()

def test_create_topic_success():
    t = create_topic(title="Как учить TDD?", author="sonya", description="Опыт и грабли")
    assert t["id"] > 0
    assert t["title"] == "Как учить TDD?"
    assert t["author"] == "sonya"
    assert t["posts"] == []

def test_add_reply_success():
    t = create_topic("Python vs JS", "makar", None)
    p = add_reply(t["id"], author="kate", content="Зависит от задач")
    assert p["id"] > 0 and p["topic_id"] == t["id"] and p["author"] == "kate"
    assert len(get_topic(t["id"])["posts"]) == 1

def test_search_topics_case_insensitive():
    create_topic("Docker и тесты", "ivan", None)
    create_topic("tdd в продакшне", "dasha", None)
    create_topic("Про велосипед", "olga", None)
    results = search_topics("TDD")
    assert {t["title"] for t in results} == {"tdd в продакшне"}

def test_validation_and_errors():
    with pytest.raises(ValueError):
        create_topic("", author="anna", description=None)
    t = create_topic("Вопросы по БД", "anna", None)
    with pytest.raises(ValueError):
        add_reply(t["id"], author="", content="ok")
    with pytest.raises(ValueError):
        add_reply(t["id"], author="max", content="")
```

Рисунок 1 – Листинг юнит-тестов test_main.py

Шаг 2: запуск тестов и ожидаемый провал

При запуске тестов на данном этапе возникла ошибка ModuleNotFoundError, так как файл main.py и классы в нем еще не существовали. Это ожидаемое поведение для «красного» этапа.

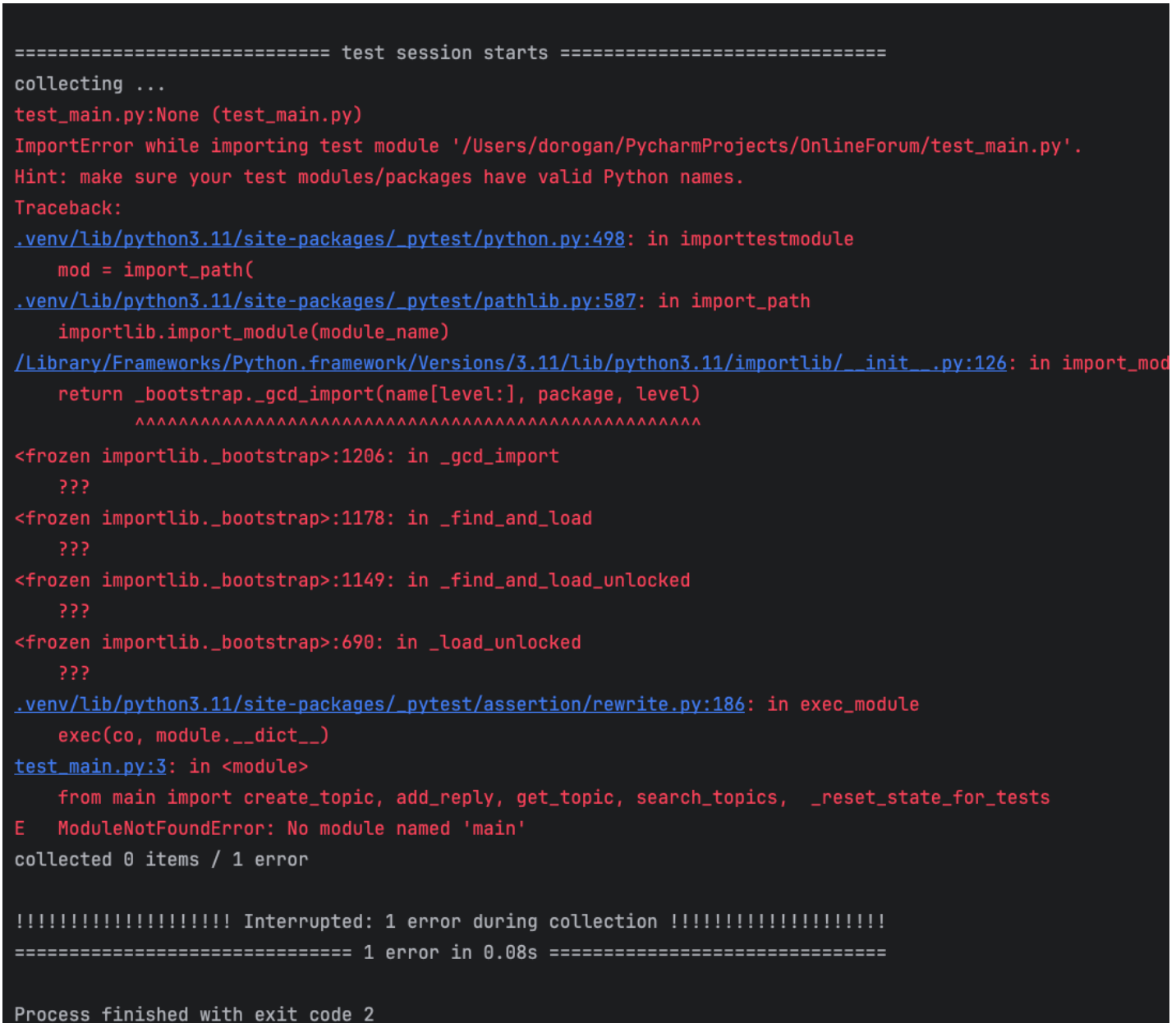

Рисунок 2– Первая ошибка

Шаг 3: реализация минимального функционала (Зеленый этап)

Был написан минимально необходимый код в файле main.py для того, чтобы тесты прошли успешно.

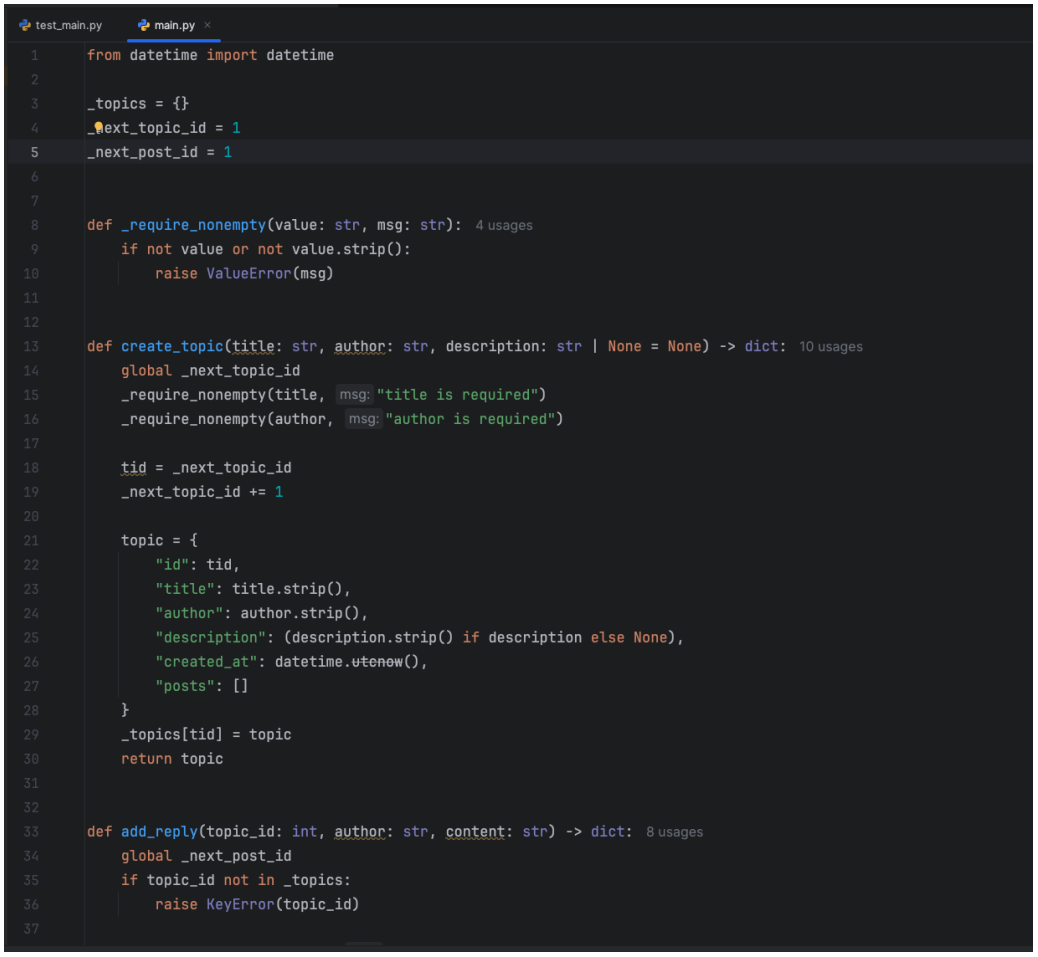

Рисунок 3 – Код main.py

Шаг 4: повторный запуск тестов

После реализации кода тесты были запущены повторно.

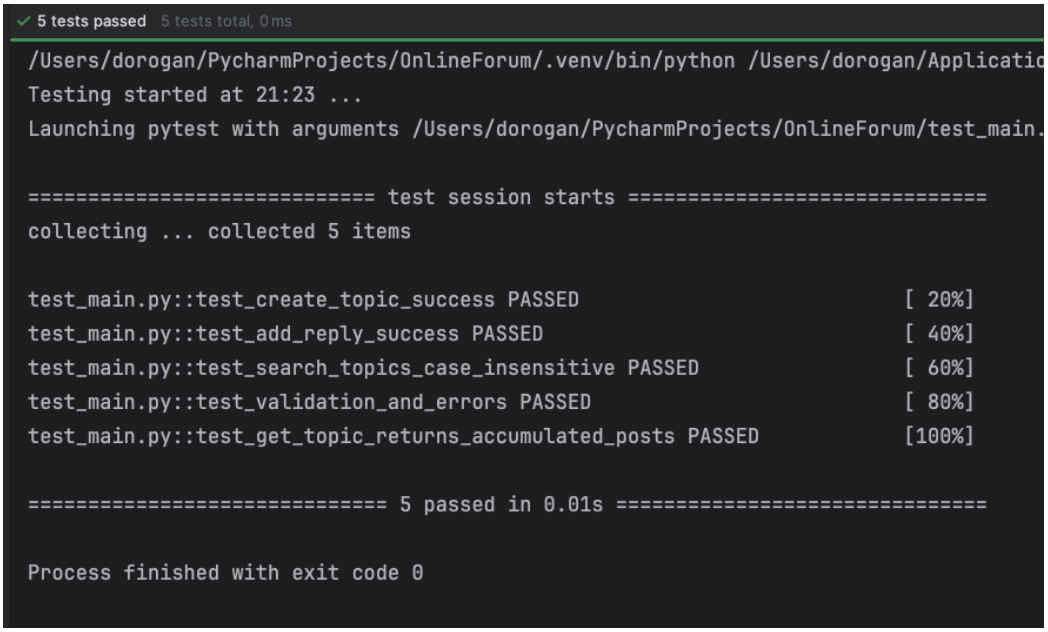

Рисунок 4 – Успешное прохождение тестов

3. Этап 2. Разработка на основе приемочных тестов (ATDD)

ATDD фокусируется на требованиях пользователя. На этом этапе мы определили ключевые сценарии использования и автоматизировали их.

Определение приемочных критериев

Совместно с условным «заказчиком» были сформулированы следующие сценарии:

Сценарий «Создание темы»: пользователь вводит заголовок и автора, после чего тема появляется в списке и доступна по уникальному идентификатору.

Сценарий «Публикация ответа»: пользователь добавляет ответ в существующую тему, и количество сообщений в ней увеличивается.

Создание и запуск автоматизированных приемочных тестов

```
def test_acceptance_add_reply():
    t = create_topic( title: "Python vs JS",  author: "makar",  description: None)
    before = len(get_topic(t["id"])["posts"])
    add_reply(t["id"],  author: "kate",  content: "Берите по задаче")
    after = len(get_topic(t["id"])["posts"])
    assert after == before + 1
```

Рисунок 5 – Автоматизированный приемочный тест

```
test_main.py::test_acceptance_add_reply PASSED                    [100%]
```

Рисунок 6 – Автоматизированный приемочный тест успешно выполненный

Этот тест успешно выполняется, подтверждая соответствие продукта требованиям.

4. Этап 3: Разработка через поведение (BDD)

На данном эта для описания функциональности с точки зрения пользователя был применен подход BDD. Он использует естественный язык для создания сценариев, которые одновременно являются и документацией, и автоматизированными тестами. Для реализации был использован фреймворк Behave для Python.

Требования к поведению викторины были сформулированы в виде сценариев в файле features/forum.feature.

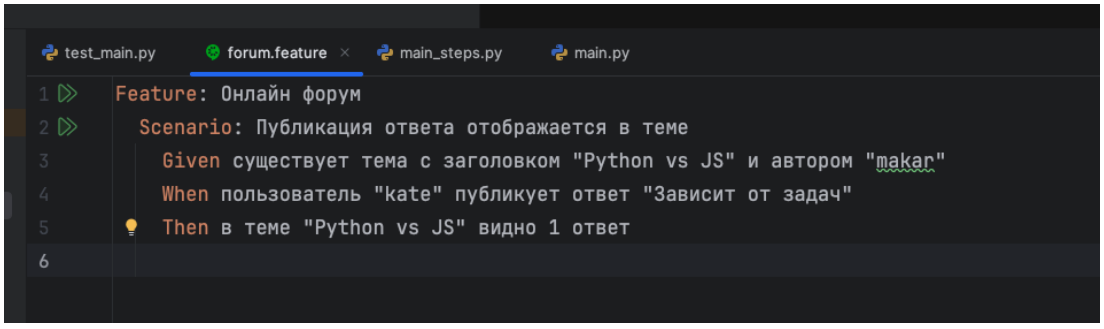

Рисунок 7 – *Пример сценария на языке Gherkin*

Реализация шагов (Step Definitions)

Для выполнения сценариев были написаны шаги с использованием фреймворка behave. Код шагов использует уже существующую логику из main.py. Для каждого шага, описанного в Gherkin, была написана соответствующая реализация на Python в файле features/steps/main_steps.py.

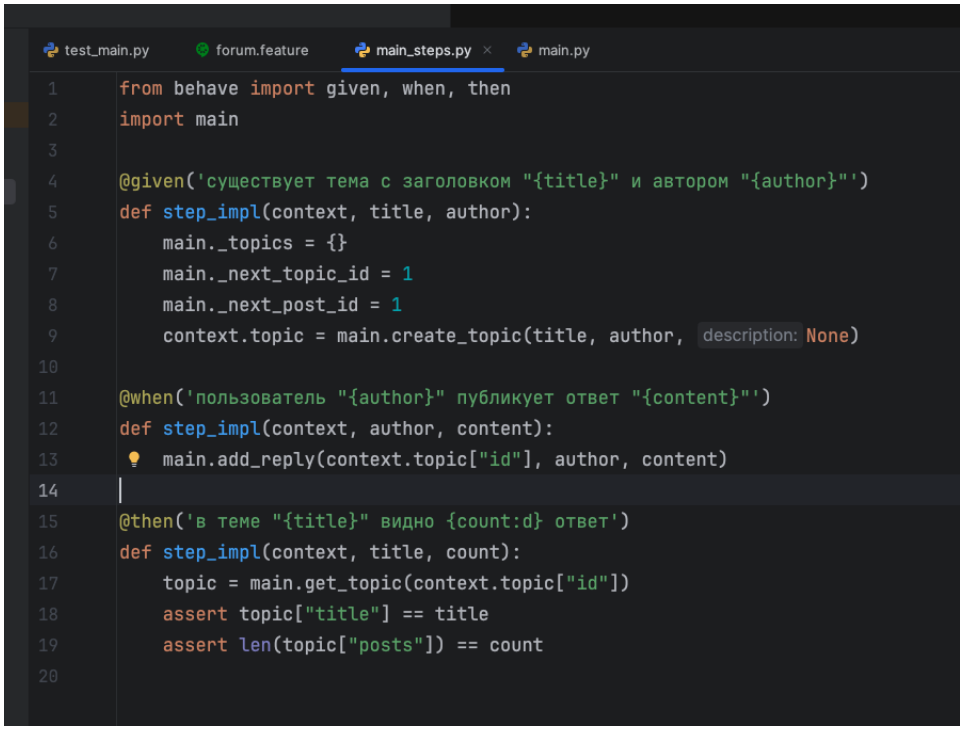

Рисунок 8 – *Код для behave*

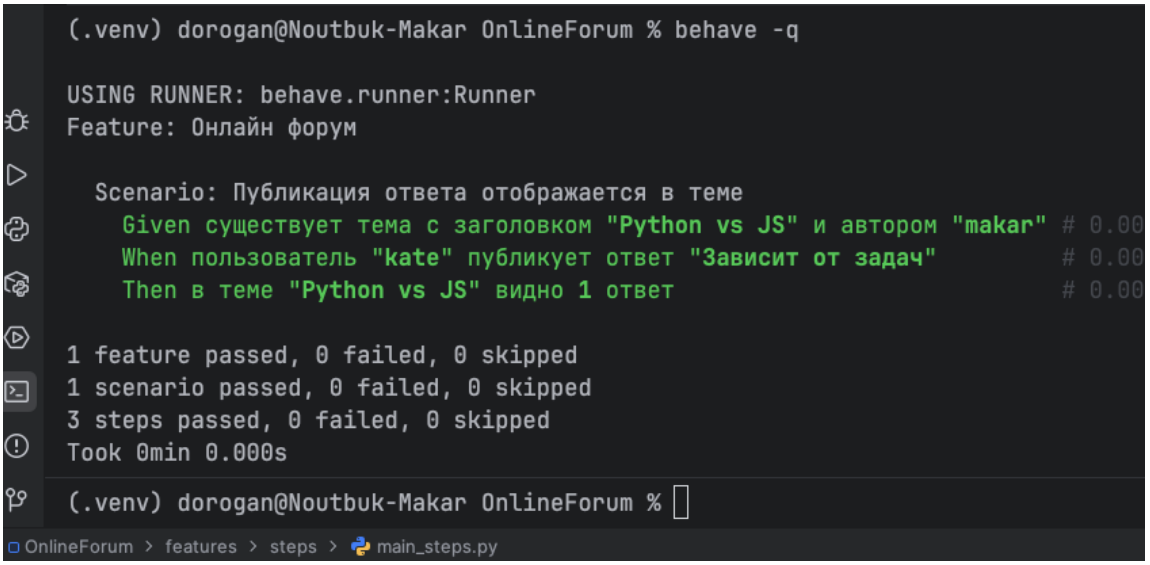

Рисунок 9 – *Успешное прохождение тестов*

Успешное прохождение всех сценариев подтверждает, что поведение приложения полностью соответствует требованиям, описанным на естественном языке.

5. Этап 4. Спецификация на примерах (SDD)

Методология Specification by Example (SDD) направлена на устранение двусмысленности в требованиях путем совместной разработки спецификаций с конкретными примерами. Эти примеры затем становятся основой для автоматизированных тестов и "живой" документации.

Для ключевых функций приложения — создания темы, публикации ответов и поиска по темам— была создана детализированная спецификация в виде таблицы. Она однозначно описывает, как должен изменяться счет в зависимости от ответа пользователя.

Таблица 1. Спецификация функций онлайн форума

№	Действие пользователя	Исходное состояние	Ожидаемый результат
0	Создание темы «Python vs JS» автором makar	Форум пуст	Тема появляется в списке тем
1	Публикация ответа пользователем kate	В теме нет ответов	В теме появляется 1 ответ
2	Публикация второго ответа пользователем ivan	Уже 1 ответ	Количество ответов = 2
3	Поиск по запросу vs	Форум содержит темы	Найдена только тема «Python vs JS»

Эта таблица служит ясным и недвусмысленным техническим заданием для разработчика и тестировщика.

Ценность подхода SDD заключается в том, что эти примеры напрямую преобразуются в тесты. В нашем случае, юнит-тесты, созданные на **Этапе 1 (TDD)**, уже полностью покрывают все сценарии из приведенной выше спецификации:

- test_create_topic_success проверяет первую строку таблицы.
- test_add_reply_success проверяет вторую и третью строки.
- test_search_topics_case_insensitive проверяет четвёртую строку.

ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы были успешно изучены и применены четыре методологии разработки через тестирование: TDD, ATDD, BDD и SDD. На примере создания приложения «Викторина» была продемонстрирована их синергия.

- TDD позволил создать надежную и полностью протестированную кодовую базу для основной логики.
- ATDD помог определить высокоуровневые требования с точки зрения пользователя и убедиться, что продукт решает его задачи.
- BDD улучшил понимание требований для всех участников процесса, предоставив сценарии на естественном языке, которые также служат автоматизированными тестами.
- SDD с помощью конкретных примеров устранил любую двусмысленность в бизнес-логике (правила начисления очков).

Комплексное использование этих подходов позволило создать качественный программный продукт, который не только корректно работает, но и хорошо документирован, легко поддерживается и полностью соответствует изначальным требованиям.