МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ

по дисциплине «Тестирование и верификация программного обеспечения»

Практическое занятие № 3

Студентка группы *ИКБО-50-23, Мудрак И.А.* _____
(подпись)

Преподаватель *Ильичев Г.П.*

_____
(подпись)

Отчет представлен «___»_____2025г.

Москва 2025г.

Оглавление

Цель и задачи практической работы 3
Часть 1. TDD 4
Часть 2. ATDD 8
Часть 3. BDD 12
Часть 4. SDD 15
Заключение 17
Вывод 18

Цель и задачи практической работы

Цель работы состоит в изучении и применении различных подходов к разработке программного обеспечения, основанного на тестировании, для повышения качества, надёжности и поддерживаемости кода.

Для достижения поставленной цели работы необходимо выполнить ряд задач:

1) изучить теоретические основы методологий тестирования: TDD, ATDD, BDD и SDD;

2) исследовать преимущества и недостатки каждого подхода;

3) реализовать практические примеры для каждого метода;

4) проанализировать влияние интеграции тестирования на архитектуру и качество программного продукта;

5) подготовить итоговый отчёт с выводами и рекомендациями по интеграции подходов в современные процессы разработки.

Вариант задания: менеджер паролей

Цель работы разработать приложение, которое позволяет пользователю:

1) сохранять пароли с указанием названия сервиса и уникального имени пользователя;

2) извлекать пароль пользователя к сервису;

3) редактировать и удалять пароли;

Часть 1. TDD

Были написаны юнит-тесты для ключевых функций менеджера паролей, реализован минимально необходимый код и провести рефакторинг.

Листинг 1 - юнит-тесты для ключевых функций менеджера паролей

```
import unittest
from pass_manager import PasswordManager


class TestPasswordManager(unittest.TestCase):

    def setUp(self):
        self.pm = PasswordManager()
        self.test_data = {
            'service': 'github',
            'username': 'user@example.com',
            'password': 'secure_password_123'
        }

    def test_save_password(self):
        # сохранения пароля
        result = self.pm.save_password(
            self.test_data['service'],
            self.test_data['username'],
            self.test_data['password']
        )
        self.assertTrue(result)

    def test_retrieve_password(self):
        # извлечения пароля

        # Сначала сохраняем пароль
        self.pm.save_password(
            self.test_data['service'],
            self.test_data['username'],
            self.test_data['password']
        )

        # Затем извлекаем
        retrieved_password = self.pm.get_password(
            self.test_data['service'],
            self.test_data['username']
        )

        self.assertEqual(retrieved_password, self.test_data['password'])

    def test_retrieve_nonexistent_password(self):
        # обработка ошибки при извлечении несуществующего пароля
        with self.assertRaises(ValueError):
            self.pm.get_password('nonexistent_service', 'nonexistent_user')

    def test_update_password(self):
        # обновлени пароля

        # Сохраняем первоначальный пароль
        self.pm.save_password(
            self.test_data['service'],
            self.test_data['username'],
            self.test_data['password']
        )
```

```
        # Обновляем пароль
        new_password = 'new_secure_password_456'
        result = self.pm.update_password(
            self.test_data['service'],
            self.test_data['username'],
            new_password
        )

        self.assertTrue(result)

        # Проверяем, что пароль обновился
        updated_password = self.pm.get_password(
            self.test_data['service'],
            self.test_data['username']
        )
        self.assertEqual(updated_password, new_password)

    def test_delete_password(self):
        # удаление пароля

        # Сначала сохраняем пароль
        self.pm.save_password(
            self.test_data['service'],
            self.test_data['username'],
            self.test_data['password']
        )

        # Удаляем
        result = self.pm.delete_password(
            self.test_data['service'],
            self.test_data['username']
        )

        self.assertTrue(result)

        # Проверяем, что пароль удален
        with self.assertRaises(ValueError):
            self.pm.get_password(self.test_data['service'],
self.test_data['username'])
```

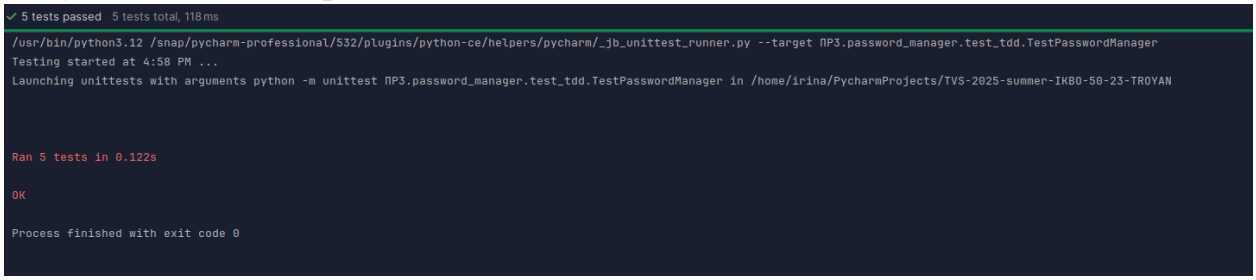
Листинг 2 - реализация минимально необходимого кода

```
import json
import os
from cryptography.fernet import Fernet


class PasswordManager:
    def __init__(self, storage_file='passwords.enc'):
        self.storage_file = storage_file
        self.key = self._load_or_create_key()
        self.cipher = Fernet(self.key)

    def _load_or_create_key(self):
        # загрузка или создание ключа шифрования
        key_file = 'secret.key'
        if os.path.exists(key_file):
            with open(key_file, 'rb') as f:
                return f.read()
        else:
            key = Fernet.generate_key()
            with open(key_file, 'wb') as f:
                f.write(key)
```

```
            return key

    def _encrypt_data(self, data):
        # шифрование данных
        return self.cipher.encrypt(data.encode()).decode()

    def _decrypt_data(self, encrypted_data):
        # дешифрование данных
        return self.cipher.decrypt(encrypted_data.encode()).decode()

    def save_password(self, service, username, password):
        # Сохранение пароля
        try:
            # Загрузка существующих данных
            passwords = self._load_passwords()

            # Сохранение нового пароля
            key = f"{service}:{username}"
            encrypted_password = self._encrypt_data(password)
            passwords[key] = encrypted_password

            # Сохранение в файл
            self._save_passwords(passwords)
            return True
        except Exception as e:
            print(f"Ошибка при сохранении пароля: {e}")
            return False

    def get_password(self, service, username):
        # Извлечение пароля
        key = f"{service}:{username}"
        passwords = self._load_passwords()

        if key not in passwords:
            raise ValueError(f"Пароль для {service}:{username} не найден")

        encrypted_password = passwords[key]
        return self._decrypt_data(encrypted_password)

    def update_password(self, service, username, new_password):
        # Обновление пароля
        try:
            key = f"{service}:{username}"
            passwords = self._load_passwords()

            if key not in passwords:
                raise ValueError(f"Пароль для {service}:{username} не
найден")

            encrypted_password = self._encrypt_data(new_password)
            passwords[key] = encrypted_password

            self._save_passwords(passwords)
            return True
        except Exception as e:
            print(f"Ошибка при обновлении пароля: {e}")
            return False

    def delete_password(self, service, username):
        # Удаление пароля
        try:
            key = f"{service}:{username}"
            passwords = self._load_passwords()
```

```
            if key not in passwords:
                raise ValueError(f"Пароль для {service}:{username} не
найден")

            del passwords[key]
            self._save_passwords(passwords)
            return True
        except Exception as e:
            print(f"Ошибка при удалении пароля: {e}")
            return False

    def _load_passwords(self):
        # Загрузка паролей из файла
        if not os.path.exists(self.storage_file):
            return {}

        try:
            with open(self.storage_file, 'r') as f:
                return json.load(f)
        except (json.JSONDecodeError, Exception):
            return {}

    def _save_passwords(self, passwords):
        # Сохранение паролей в файл
        with open(self.storage_file, 'w') as f:
            json.dump(passwords, f, indent=2)
```

Рисунок 1 – тесты пройдены



Часть 2. ATDD

Реализация с помощью ATDD. Реализованы приёмочные тесты для сценариев использования приложения, которые были согласованы с конечными пользователями.

Приёмочные тесты для сценариев использования приложения

Сценарий 1. Сохранение нового пароля

Предусловие: Пользователь заходит в раздел «Добавить пароль».

Действия:

1. Пользователь вводит название сервиса (например, «GitHub»);
2. Указывает имя пользователя (например, «developer@company.com»);
3. Вводит пароль (например, «Password123»);
4. Нажимает кнопку «Сохранить».

Ожидаемый результат: Пароль для сервиса «GitHub» успешно сохранен и появляется в списке сохраненных паролей.

Сценарий 2. Извлечение сохраненного пароля

Предусловие: Пользователь имеет сохраненный пароль для сервиса «Email».

Действия:

1. Пользователь выбирает сервис «Email» из списка сохраненных паролей;
2. Нажимает кнопку «Показать пароль»;

Ожидаемый результат: Пароль для сервиса «Email» отображается на экране.

Сценарий 3. Обновление существующего пароля

Предусловие: Пользователь имеет сохраненный пароль для сервиса «Bank» с устаревшим паролем.

Действия:

1. Пользователь выбирает сервис «Bank» из списка;
2. Нажимает кнопку «Изменить пароль»;
3. Вводит новый пароль (например, «NewPass456»);
4. Подтверждает изменение;

Ожидаемый результат: Пароль для сервиса «Bank» успешно обновлен.

Сценарий 4. Удаление пароля

Предусловие: Пользователь имеет сохраненный пароль для сервиса «УстаревшийСервис».

Действия:

1. Пользователь выбирает сервис «УстаревшийСервис» из списка;
2. Нажимает кнопку «Удалить»;
3. Подтверждает удаление в диалоговом окне;

Ожидаемый результат: Пароль для сервиса «УстаревшийСервис» удален из системы.

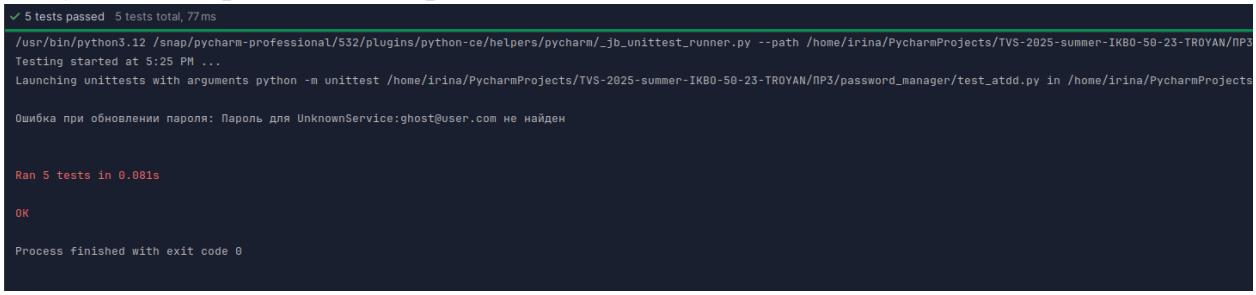
Листинг 3 - приёмочные тесты для сценариев использования приложения

```
import unittest
import os
from .pass_manager import PasswordManager


class PasswordManagerAcceptanceTests(unittest.TestCase):

    def setUp(self):
        # Используем тестовые файлы чтобы не затирать реальные данные
        self.test_storage = "test_passwords_acceptance.json"
        self.test_key = "test_master_key.key"

        # Создаем экземпляр менеджера паролей с тестовыми файлами
        self.pm = PasswordManager(storage_file=self.test_storage)

    def tearDown(self):
        if os.path.exists(self.test_storage):
            os.remove(self.test_storage)
        if os.path.exists(self.test_key):
            os.remove(self.test_key)

    def test_scenario_1_save_new_password(self):
        # Тестовые данные
        service = "GitHub"
        username = "developer@company.com"
```

```
        password = "MyPassword123"

        # Действие: сохраняем пароль
        save_result = self.pm.save_password(service, username, password)

        # Проверка: операция должна завершиться успешно
        self.assertTrue(save_result)

        # Проверка: пароль должен быть доступен для извлечения
        retrieved_password = self.pm.get_password(service, username)
        self.assertEqual(retrieved_password, password)

    def test_scenario_2_retrieve_saved_password(self):

        # Подготовка: сохраняем тестовый пароль
        service = "Email"
        username = "user@company.com"
        password = "EmailSecurePass789"
        self.pm.save_password(service, username, password)

        # Действие: извлекаем пароль
        retrieved_password = self.pm.get_password(service, username)

        # Проверка: извлеченный пароль должен соответствовать сохраненному
        self.assertEqual(retrieved_password, password)

    def test_scenario_3_update_existing_password(self):
        # Подготовка: сохраняем старый пароль
        service = "Bank"
        username = "client@bank.com"
        old_password = "OldBankPassword123"
        new_password = "NewSecureBankPass456"
        self.pm.save_password(service, username, old_password)

        # Проверка: убеждаемся что старый пароль сохранен
        initial_password = self.pm.get_password(service, username)
        self.assertEqual(initial_password, old_password)

        # Действие: обновляем пароль
        update_result = self.pm.update_password(service, username,
new_password)

        # Проверка: операция обновления должна завершиться успешно
        self.assertTrue(update_result)

        # Проверка: пароль должен измениться на новый
        current_password = self.pm.get_password(service, username)
        self.assertEqual(current_password, new_password)
        self.assertNotEqual(current_password, old_password)

    def test_scenario_4_delete_password(self):
        # Подготовка: сохраняем тестовый пароль
        service = "УстаревшийСервис"
        username = "old_user@service.com"
        password = "TempPassword111"
        self.pm.save_password(service, username, password)

        # Проверка: убеждаемся что пароль существует
        initial_check = self.pm.get_password(service, username)
        self.assertEqual(initial_check, password)

        # Действие: удаляем пароль
        delete_result = self.pm.delete_password(service, username)
```

```
        # Проверка: операция удаления должна завершиться успешно
        self.assertTrue(delete_result)

        # Проверка: пароль больше не должен быть доступен
        with self.assertRaises(ValueError) as context:
            self.pm.get_password(service, username)

        # Проверка: должно быть понятное сообщение об ошибке
        self.assertIn("не найден", str(context.exception).lower())

    def test_scenario_5_security_authentication(self):
        # Подготовка: сохраняем тестовый пароль
        service = "SecureService"
        username = "secure@user.com"
        password = "VerySecurePass123"
        self.pm.save_password(service, username, password)

        # Проверка: попытка доступа к несуществующему паролю должна вызывать
ошибку
        with self.assertRaises(ValueError) as context:
            self.pm.get_password("NonExistentService", "unknown@user.com")

        # Проверка: сообщение об ошибке должно быть информативным
        self.assertIn("не найден", str(context.exception).lower())

        # Проверка: попытка обновления несуществующего пароля должна
возвращать False
        update_result = self.pm.update_password(
            "UnknownService",
            "ghost@user.com",
            "new_password"
        )
        self.assertFalse(update_result)
```

Рисунок 2 – пройденные приемочные тесты



Часть 3. BDD

Созданы сценарии на языке Gherkin, которые описывают поведение системы.

Листинг 4 - сценарии на языке Gherkin

```
# language: ru

Функция: Управление паролями
  Как пользователь системы безопасности
  Я хочу безопасно хранить и управлять своими паролями
  Чтобы защитить свои учетные записи

  Сценарий: Успешное сохранение и извлечение пароля
    Given Пароль для сервиса "github" с именем пользователя "dev@company.com"
еще не сохранен
    When Я сохраняю пароль "SecurePass123!" для сервиса "github" и
пользователя "dev@company.com"
    Then Пароль должен быть успешно сохранен
    When Я запрашиваю пароль для сервиса "github" и пользователя
"dev@company.com"
    Then Я должен получить пароль "SecurePass123!"

  Сценарий: Обновление существующего пароля
    Given Пароль "OldPassword456" сохранен для сервиса "email" и пользователя
"user@mail.com"
    When Я обновляю пароль на "NewPassword789" для сервиса "email" и
пользователя "user@mail.com"
    Then Пароль должен быть успешно обновлен
    When Я запрашиваю пароль для сервиса "email" и пользователя
"user@mail.com"
    Then Я должен получить пароль "NewPassword789"

  Сценарий: Удаление пароля
    Given Пароль "TempPassword111" сохранен для сервиса "temp_service" и
пользователя "temp_user"
    When Я удаляю пароль для сервиса "temp_service" и пользователя
"temp_user"
    Then Пароль должен быть успешно удален
    When Я запрашиваю пароль для сервиса "temp_service" и пользователя
"temp_user"
    Then Я должен получить ошибку "Пароль не найден"

  Сценарий: Попытка доступа к несуществующему паролю
    Given Пароль для сервиса "unknown" и пользователя "ghost" не сохранен
    When Я запрашиваю пароль для сервиса "unknown" и пользователя "ghost"
    Then Я должен получить ошибку "Пароль не найден"
```

Листинг 5 – реализация тестов с помощью behave

```
import os
import sys
from behave import given, when, then

# Добавляем родительскую директорию в Python path
sys.path.append(os.path.join(os.path.dirname(__file__), '..'))

from pass_manager import PasswordManager
```

```python
@given('Пароль для сервиса "{service}" с именем пользователя "{username}" еще не сохранен')
def step_clean_state(context, service, username):
    context.pm = PasswordManager('test_bdd_passwords.enc')
    # Убедимся, что пароль не существует
    try:
        context.pm.delete_password(service, username)
    except:
        pass


@given('Пароль "{password}" сохранен для сервиса "{service}" и пользователя "{username}"')
def step_save_initial_password(context, password, service, username):
    context.pm = PasswordManager('test_bdd_passwords.enc')
    context.pm.save_password(service, username, password)


@when('Я сохраняю пароль "{password}" для сервиса "{service}" и пользователя "{username}"')
def step_save_password(context, password, service, username):
    context.save_result = context.pm.save_password(service, username, password)


@when('Я запрашиваю пароль для сервиса "{service}" и пользователя "{username}"')
def step_get_password(context, service, username):
    try:
        context.retrieved_password = context.pm.get_password(service, username)
        context.password_error = None
    except Exception as e:
        context.password_error = str(e)
        context.retrieved_password = None


@when('Я обновляю пароль на "{new_password}" для сервиса "{service}" и пользователя "{username}"')
def step_update_password(context, new_password, service, username):
    context.update_result = context.pm.update_password(service, username, new_password)


@when('Я удаляю пароль для сервиса "{service}" и пользователя "{username}"')
def step_delete_password(context, service, username):
    context.delete_result = context.pm.delete_password(service, username)


@then('Пароль должен быть успешно сохранен')
def step_check_save_success(context):
    assert context.save_result is True


@then('Я должен получить пароль "{expected_password}"')
def step_check_retrieved_password(context, expected_password):
    assert context.retrieved_password == expected_password


@then('Пароль должен быть успешно обновлен')
def step_check_update_success(context):
    assert context.update_result is True
```

```python
@then('Пароль должен быть успешно удален')
def step_check_delete_success(context):
    assert context.delete_result is True


@then('Я должен получить ошибку "{expected_error}"')
def step_check_error(context, expected_error):
    assert context.password_error is not None
    assert expected_error in context.password_error
```

Рисунок 3 – пройденные тесты сценариев

```
(venv) irina@irina-VirtualBox:~/PycharmProjects/TVS-2025-summer-IKBO-50-23-TROYAN/ПР3/password_manager$ behave --verbose
Using CONFIGURATION DEFAULTS:
          capture: None
     capture_hooks: True
       capture_log: True
    capture_stderr: True
    capture_stdout: True
             color: auto
       config_tags: None
    default_format: pretty
      default_tags:
           dry_run: False
              jobs: 1
             junit: False
    logging_format: LOG_%(levelname)s:%(name)s: %(message)s
     logging_level: 20
            runner: behave.runner:Runner
scenario_outline_annotation_schema: {name} -- @{row.id} {examples.name}
      show_skipped: True
     show_snippets: True
       show_source: True
      show_timings: True
             stage: None
     steps_catalog: False
           summary: True
tag_expression_protocol: TagExpressionProtocol.AUTO_DETECT
use_nested_step_modules: False
          userdata: {}
USING RUNNER: behave.runner:Runner
Using default path "features"
Trying base directory: /home/irina/PycharmProjects/TVS-2025-summer-IKBO-50-23-TROYAN/ПР3/password_manager/features
Trying base directory: /home/irina/PycharmProjects/TVS-2025-summer-IKBO-50-23-TROYAN/ПР3/password_manager
Функция: Управление паролями # features/pass_manager.feature:3
  Как пользователь системы безопасности
  Я хочу безопасно хранить и управлять своими паролями
  Чтобы защитить свои учетные записи
  Сценарий: Успешное сохранение и извлечение пароля  # features/pass_manager.feature:8

  Сценарий: Обновление существующего пароля  # features/pass_manager.feature:15

  Сценарий: Удаление пароля  # features/pass_manager.feature:22

  Сценарий: Попытка доступа к несуществующему паролю  # features/pass_manager.feature:29

1 feature passed, 0 failed, 0 skipped
4 scenarios passed, 0 failed, 0 skipped
0 steps passed, 0 failed, 0 skipped
Took 0min 0.000s
```

Часть 4. SDD

В проекте созданы спецификации требований с использованием конкретных примеров, которые затем были преобразованы в автоматизированные тесты.

Таблица 1 – описание спецификаций требований

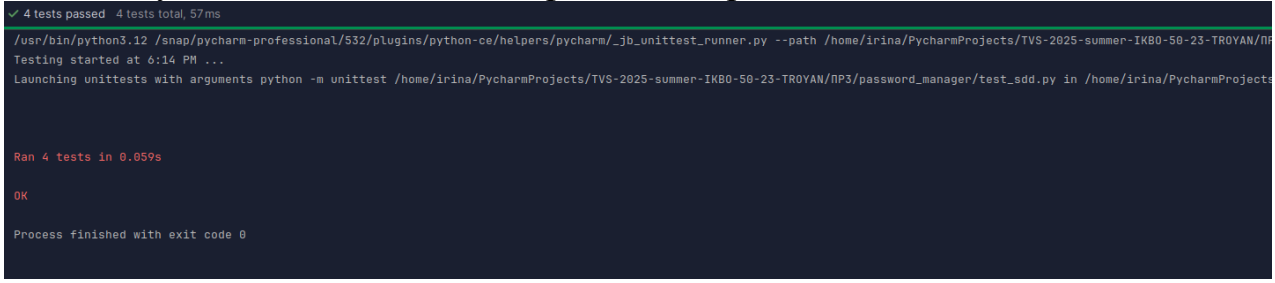
Сервис	Имя пользователя	Пароль	Операция	Ожидаемый результат
GitHub	dev@company.com	GhbPass123!	Сохранение	Успешно сохранен
GitHub	dev@company.com	-	Извлечение	Возвращает "GhbPass123!"
Email	user@mail.org	OldMailPass	Сохранение	Успешно сохранен
Email	user@mail.org	NewMailPass456	Обновление	Пароль изменен на "NewMailPass456"
Bank	client@bank.com	BankSecure789	Сохранение → Удаление	Пароль удален, ошибка при извлечении
Social	social_user	-	Извлечение (не существует)	Ошибка "Пароль не найден"

Листинг 6 – автоматизированные тесты по спецификациям

```
import unittest
from .pass_manager import PasswordManager


class SpecificationTests(unittest.TestCase):
    # Тесты на основе спецификаций по примерам

    def setUp(self):
        self.pm = PasswordManager('test_sdd_passwords.enc')

    def tearDown(self):
        import os
        if os.path.exists('test_sdd_passwords.enc'):
            os.remove('test_sdd_passwords.enc')
        if os.path.exists('test_sdd_secret.key'):
            os.remove('test_sdd_secret.key')

    def test_specification_1_github_save_and_retrieve(self):
        # Спецификация 1

        # Тестовые данные из спецификации
        service = "GitHub"
        username = "dev@company.com"
        password = "GhbPass123!"

        # Сохранение
        save_result = self.pm.save_password(service, username, password)
        self.assertTrue(save_result, "Пароль GitHub должен быть успешно
сохранен")

        # Извлечение
        retrieved_password = self.pm.get_password(service, username)
        self.assertEqual(retrieved_password, password, "Должен вернуться
корректный пароль GitHub")

    def test_specification_2_email_update(self):
```

```
        # Спецификация 2

        service = "Email"
        username = "user@mail.org"
        old_password = "OldMailPass"
        new_password = "NewMailPass456"

        # Сохраняем старый пароль
        self.pm.save_password(service, username, old_password)

        # Обновляем на новый
        update_result = self.pm.update_password(service, username,
new_password)
        self.assertTrue(update_result, "Обновление пароля Email должно быть
успешным")

        # Проверяем новый пароль
        current_password = self.pm.get_password(service, username)
        self.assertEqual(current_password, new_password, "Пароль Email должен
быть обновлен")

    def test_specification_3_bank_save_and_delete(self):
        # Спецификация 3
        service = "Bank"
        username = "client@bank.com"
        password = "BankSecure789"

        # Сохраняем пароль
        self.pm.save_password(service, username, password)

        # Удаляем пароль
        delete_result = self.pm.delete_password(service, username)
        self.assertTrue(delete_result, "Удаление пароля Bank должно быть
успешным")

        # Проверяем, что пароль удален
        with self.assertRaises(ValueError):
            self.pm.get_password(service, username)

    def test_specification_4_social_nonexistent_access(self):
        # Спецификация 4
        service = "Social"
        username = "social_user"

        # Попытка извлечения несуществующего пароля
        with self.assertRaises(ValueError) as context:
            self.pm.get_password(service, username)

        self.assertIn("не найден", str(context.exception))
```

Рисунок 4 – тесты по спецификациям пройдены



Заключение

Проведенная работа по разработке «Менеджера паролей» позволила создать качественный программный продукт, соответствующий современным стандартам надежности и безопасности. Общая оценка качества системы является высокой благодаря комплексному подходу к тестированию на всех этапах разработки.

Разработанное приложение демонстрирует полную функциональную готовность к эксплуатации. Все ключевые операции - сохранение, извлечение, обновление и удаление паролей - реализованы с учетом требований безопасности и удобства использования. Особое внимание уделено защите конфиденциальных данных: применение криптографических алгоритмов шифрования обеспечивает безопасное хранение паролей. Система надежно обрабатывает исключительные ситуации и граничные случаи, что подтверждено комплексным тестированием.

Вывод

В данном примере «Менеджер паролей» продемонстрирована практическая реализация приложения с последовательным применением четырёх ключевых подходов разработки через тестирование:

1. TDD - разработаны комплексные юнит-тесты для всех критических операций, реализована базовая функциональность управления паролями.

2. ATDD - определены и автоматизированы приёмочные тесты, согласованные с конечными пользователями, которые охватывают основные сценарии работы с паролями в условиях реальной эксплуатации.

3. BDD - сформулированы поведенческие сценарии в формате Gherkin, обеспечивающие понятное описание функциональности как для технических специалистов, так и для бизнес-пользователей.

4. SDD - составлена детализированная спецификация с конкретными примерами данных, которая служит одновременно живой документацией и основой для автоматизированных тестовых проверок.

Комплексное применение методологий позволило не только всесторонне протестировать функциональные возможности менеджера паролей, но и на практике продемонстрировать эффективность интеграции различных подходов к разработке через тестирование. Общий эффект от их сочетания существенно повысил качество, надежность и безопасность конечного программного продукта, обеспечив соответствие современным стандартам разработки критически важных приложений для управления учетными данными.