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»

Практическая работа № 3
ПОДХОДЫ ДЛЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Студент группы _____

(подпись)

Преподаватель _____

(подпись)

Отчет представлен «___»_____2025 г.

Москва 2025 г.

1. ЦЕЛЬ И ЗАДАЧИ

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 студентам 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.

2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1. TDD (Test-Driven Development)

Test-Driven Development — это методология разработки программного обеспечения, которая подразумевает создание тестов для функциональности ПО до того, как эта функциональность будет фактически реализована. TDD представляет собой циклический процесс, который помогает разработчикам создавать высококачественное, надежное ПО.

Этапы разработки по методологии TDD:

1. **Создание тестов** — разработчик создает тесты, которые описывают ожидаемое поведение новой функциональности или компонента на основе спецификаций, требований или ожиданий от заказчика.

2. **Запуск тестов** — поскольку код, реализующий требуемую функциональность, еще не написан, все тесты должны провалиться, что ожидаемо.

3. **Реализация функциональности** — разработчик начинает писать код, который будет реализовывать функциональность. Основная цель — написать минимальный объем кода, который позволит пройти тесты.

4. **Повторный запуск тестов** — после написания некоторой части кода разработчик снова запускает тесты для проверки их прохождения.

5. **Рефакторинг** — после успешного прохождения тестов разработчик может провести рефакторинг кода с целью улучшить читаемость, поддерживаемость и эффективность, но при этом не изменять функциональность.

Если приложение сложное, то процесс разработки будет циклическим — разработчик добавляет новые тесты, запускает их, реализует функциональность, запускает тесты снова, проводит рефакторинг и так далее. Цель — гарантировать, что каждая добавленная или измененная часть кода проходит тесты и не ломает то, что было реализовано ранее.

TDD является одним из ключевых инструментов для повышения качества программного обеспечения и ускорения процесса разработки.

2.2. ATDD (Acceptance Test-Driven Development)

Acceptance Test-Driven Development — это методология, при которой разработка программного обеспечения начинается с формулирования приёмочных тестов, отражающих требования заказчика. Эти тесты пишутся совместно с участниками проекта (бизнес-аналитиками, заказчиками, тестировщиками и разработчиками) и определяют, какую функциональность должен предоставить продукт.

Этапы разработки по методологии ATDD:

1. **Обсуждение требований** — все заинтересованные стороны собираются для обсуждения требований и составляют список сценариев или тестовых случаев, описывающих, как должна работать система с точки зрения потребителя.

2. **Создание приёмочных сценариев** — требования оформляются в виде приёмочных сценариев, которые преобразуются в автоматизированные тесты. Поскольку функциональность ещё не реализована, все тесты будут «падать».

3. **Реализация функциональности** — разработчики пишут код, обеспечивающий выполнение требований, зафиксированных в приёмочных тестах.

4. **Проверка соответствия** — после реализации кода тесты запускаются повторно. Если все тесты проходят, это свидетельствует о том, что функциональность соответствует ожиданиям.

5. **Оптимизация и рефакторинг** — при необходимости проводится рефакторинг, оптимизация кода и улучшение архитектуры с повторным запуском тестов.

ATDD позволяет команде разработки, тестировщикам и бизнес-аналитикам совместно формировать требования к системе в виде приёмочных тестов ещё до написания кода. Это способствует лучшему пониманию ожидаемого поведения системы, повышает прозрачность разработки и снижает количество ошибок.

2.3. BDD (Behavior-Driven Development)

Behavior-Driven Development — это методология разработки программного обеспечения, которая фокусируется на описании ожидаемого поведения системы с использованием понятного всем участникам процесса

естественного языка. BDD является развитием подхода Test-Driven Development (TDD) и направлена на улучшение взаимодействия между разработчиками, тестировщиками и бизнес-аналитиками.

Этапы разработки по методологии BDD:

1. **Формулирование сценариев** — требования к системе формулируются в виде сценариев на естественном языке с использованием структуры «Given–When–Then»:
   - Given (Дано) — начальные условия или состояние системы;
   - When (Когда) — действие пользователя или событие;
   - Then (Тогда) — ожидаемый результат.
2. **Автоматизация сценариев** — сценарии превращаются в автоматизированные тесты с помощью инструментов. Поскольку функциональность ещё не реализована, тесты проваливаются.
3. **Реализация поведения** — разработчики пишут код, реализующий описанное поведение системы.
4. **Проверка поведения** — сценарии запускаются снова для проверки соответствия системы описанным ожиданиям.
5. **Оптимизация** — при необходимости производится оптимизация кода с повторным запуском тестов.

Методология BDD способствует улучшению коммуникации внутри команды и созданию общего понимания требований.

2.4. SDD (Specification by Example)

Specification by Example — это подход, который использует конкретные примеры для описания требований и превращает их в автоматизированные тесты. Вместо абстрактных описаний бизнес-требований используются реальные примеры, позволяющие избежать неоднозначностей в понимании требований между бизнесом и разработчиками.

Этапы разработки по методологии SDD:

1. **Сбор примеров** — вместе с бизнес-аналитиками и заказчиками собираются конкретные примеры использования системы, отражающие различные сценарии работы продукта.

2. **Создание спецификаций** — примеры превращаются в спецификации, которые описывают входные данные, ожидаемые результаты и условия работы.

3. **Автоматизация тестов** — спецификации с примерами используются для написания автоматизированных тестов. На этом этапе тесты проваливаются, поскольку функциональность ещё не реализована.

4. **Реализация функциональности** — разработчики пишут код, обеспечивающий выполнение спецификаций.

5. **Проверка и рефакторинг** — после реализации функциональности тесты повторно запускаются для проверки корректности работы. При необходимости проводится рефакторинг кода.

2.5. Сравнительный анализ подходов

Все четыре подхода — TDD, ATDD, BDD и SDD — интегрируют тестирование в процесс разработки, но делают это с разными акцентами:

- TDD базируется на тестах;
- ATDD фокусируется на согласовании приёмочных критериев с бизнесом и заказчиками до начала реализации;
- BDD использует сценарии на естественном языке для описания поведения системы;
- SDD строит спецификации на конкретных примерах, позволяющих устранить двусмысленность требований.

Каждая из методологий помогает обеспечить качество кода, уменьшить количество ошибок и улучшить коммуникацию между участниками проекта, выбирается в зависимости от специфики задачи и потребностей команды.

3. ПРАКТИЧЕСКАЯ ЧАСТЬ

3.1. Личный вариант

26. Социальная сеть (мини-версия)

Функции — регистрация, добавление друзей, публикация статусов.

TDD — тесты для проверки регистрации, поиска пользователей и обновления статусов.

ATDD — приёмочные тесты для сценариев взаимодействия пользователей.

BDD — сценарий «Given зарегистрирован пользователь, When добавляю друга, Then друг отображается в списке».

SDD — спецификации с примерами входных данных (имена, email) и результатами.

3.2. Реализованная функциональность

3.2.1. Основные классы и методы

Класс User: хранение данных пользователя

```
class User:  15 usages
    def __init__(self, username, email, password):
        self.username = username
        self.email = email
        self.password = password
        self.friends = []
        self.statuses = []
        self.registered_at = datetime.now()
```

Рисунок 1 - Код класса User

Класс SocialNetwork: управление социальной сетью

```
class SocialNetwork:  8 usages
    def __init__(self):
        self.users = {}
```

Рисунок 2 - Код класса SocialNetwork

Методы:

- register_user() - регистрация пользователя

```python
def register_user(self, user):  15 usages
    if user.email in self.users:
        raise ValueError(f"Пользователь с электронной почтой {user.email} уже существует")
    self.users[user.email] = user
```

Рисунок 3 - Код метода register_user

- add_friend() - добавление в друзья

```python
def add_friend(self, user_email, friend_email):  3 usages (1 dynamic)
    if user_email not in self.users:
        raise ValueError(f"Пользователь {user_email} не найден")
    if friend_email not in self.users:
        raise ValueError(f"Пользователь {friend_email} не найден")
    if user_email == friend_email:
        raise ValueError("Нельзя добавить самого себя в друзья")
    user = self.users[user_email]
    friend = self.users[friend_email]
    if friend_email not in user.friends:
        user.friends.append(friend_email)
    if user_email not in friend.friends:
        friend.friends.append(user_email)
```

Рисунок 4 - Код метода add_friend

- post_status() - публикация статуса

```python
def post_status(self, user_email, text):  2 usages
    if user_email not in self.users:
        raise ValueError(f"Пользователь {user_email} не найден")
    if not text.strip():
        raise ValueError("Текст статуса не может быть пустым")
    status = {
        "text": text,
        "timestamp": datetime.now(),
        "likes": 0
    }
    self.users[user_email].statuses.append(status)
```

Рисунок 5 - Код метода post_status

- find_users_by_username() - поиск пользователей

```python
def find_users_by_username(self, username):  2 usages
    return [user for user in self.users.values() if username.lower() in user.username.lower()]
```

Рисунок 6 - Код метода find_users_by_username

3.2.2. Обработка ошибок

Дублирование электронной почты при регистрации

Добавление несуществующего пользователя в друзья

Добавление самого себя в друзья

Публикация пустого статуса

3.3. Реализации методов

3.3.1. TDD

```python
import unittest
from social_network import User, SocialNetwork


class TestSocialNetworkTDD(unittest.TestCase):

    def setUp(self):
        self.network = SocialNetwork()
        self.user1 = User( username: "ivan",  email: "ivan@mail.com",  password: "password123")
        self.user2 = User( username: "maria",  email: "maria@mail.com",  password: "password456")

    def test_user_registration(self):
        self.network.register_user(self.user1)
        self.assertIn( member: "ivan@mail.com", self.network.users)

    def test_duplicate_registration(self):
        self.network.register_user(self.user1)
        with self.assertRaises(ValueError):
            self.network.register_user(User( username: "ivan2",  email: "ivan@mail.com",  password: "pass"))

    def test_post_status(self):
        self.network.register_user(self.user1)
        self.network.post_status( user_email: "ivan@mail.com",  text: "Мой первый пост!")
        ivan = self.network.users["ivan@mail.com"]
        self.assertEqual(len(ivan.statuses),  second: 1)
        self.assertEqual(ivan.statuses[0]["text"],  second: "Мой первый пост!")

```

Рисунок 7 - Код реализации TDD (часть 1)

```python
    def test_find_users_by_username(self):
        self.network.register_user(self.user1)
        self.network.register_user(self.user2)

        found_users = self.network.find_users_by_username("ivan")
        self.assertEqual(len(found_users),  second: 1)
        self.assertEqual(found_users[0].email,  second: "ivan@mail.com")

    def test_find_users_partial_match(self):
        user3 = User( username: "ivanov",  email: "ivanov@mail.com",  password: "pass")
        self.network.register_user(self.user1)
        self.network.register_user(user3)

        found_users = self.network.find_users_by_username("iva")
        self.assertEqual(len(found_users),  second: 2)


if __name__ == "__main__":
    unittest.main()

```

Рисунок 8 - Код реализации TDD (часть 2)

3.3.2. ATDD

```python
import unittest
from social_network import SocialNetwork, User


class AcceptanceTestsATDD(unittest.TestCase):

    def test_friendship_flow(self):
        network = SocialNetwork()

        user1 = User( username: "alice",  email: "alice@mail.com",  password: "pass1")
        user2 = User( username: "bob",  email: "bob@mail.com",  password: "pass2")
        network.register_user(user1)
        network.register_user(user2)

        network.add_friend( user_email: "alice@mail.com",  friend_email: "bob@mail.com")

        alice = network.users["alice@mail.com"]
        bob = network.users["bob@mail.com"]

        self.assertIn( member: "bob@mail.com", alice.friends)
        self.assertIn( member: "alice@mail.com", bob.friends)


if __name__ == "__main__":
    unittest.main()

```

Рисунок 9 - Код реализации ATDD

3.3.3. BDD

```
# language: ru

Функционал: Социальная сеть

  Сценарий: Добавление друга в социальной сети
    Дано зарегистрирован пользователь с электронной почтой "user1@mail.com"
    И зарегистрирован пользователь с электронной почтой "user2@mail.com"
    Когда пользователь "user1@mail.com" добавляет в друзья "user2@mail.com"
    Тогда пользователь "user2@mail.com" появляется в списке друзей пользователя "user1@mail.com"
    И пользователь "user1@mail.com" также появляется в списке друзей пользователя "user2@mail.com"

```

Рисунок 10 - Сценарий BDD

```python
from behave import given, when, then
from social_network import SocialNetwork, User

@given('зарегистрирован пользователь с электронной почтой "{email}"')
def step_impl(context, email):
    if not hasattr(context, 'network'):
        context.network = SocialNetwork()
    username = email.split('@')[0]
    user = User(username, email, password: "password")
    context.network.register_user(user)

@when('пользователь "{user_email}" добавляет в друзья "{friend_email}"')
def step_impl(context, user_email, friend_email):
    context.network.add_friend(user_email, friend_email)

@then('пользователь "{friend_email}" появляется в списке друзей пользователя "{user_email}"')
def step_impl(context, user_email, friend_email):
    user = context.network.users[user_email]
    assert friend_email in user.friends

@then('пользователь "{user_email}" также появляется в списке друзей пользователя "{friend_email}"')
def step_impl(context, user_email, friend_email):
    friend = context.network.users[friend_email]
    assert user_email in friend.friends

```

Рисунок 11 - Код реализации BDD

3.3.4. SDD

№	Описание тестового случая	Входные данные	Ожидаемый результат
1	Успешная регистрация пользователя Петр	username: "Петр" email: "petr@mail.com" password: "123456"	Успех (пользователь добавлен в систему)
2	Ошибка при регистрации с существующей электронной почтой	username: "Дубликат" email: "ivan@mail.com" password: "654321"	Ошибка (исключение ValueError)

№	Описание тестового случая	Входные данные	Ожидаемый результат
1	Успешное добавление друга: Иван и Мария становятся взаимными друзьями	user_email: "ivan@mail.com" friend_email: "maria@mail.com"	Успех (взаимная дружба установлена)

№	Описание тестового случая	Входные данные	Ожидаемый результат
1	Успешная публикация статуса "Привет всем!" пользователем Иван	user_email: "ivan@mail.com" text: "Привет всем!"	Успех (статус опубликован)

```python
import unittest
from social_network import SocialNetwork, User


SPECIFICATIONS = {
    "registration": [
        {
            "input": {"username": "Петр", "email": "petr@mail.com", "password": "123456"},
            "expected": "success",
            "description": "Успешная регистрация пользователя Петр"
        },
        {
            "input": {"username": "Дубликат", "email": "ivan@mail.com", "password": "654321"},
            "expected": "error",
            "description": "Ошибка: электронная почта ivan@mail.com уже существует"
        }
    ],
    "friendship": [
        {
            "input": {"user_email": "ivan@mail.com", "friend_email": "maria@mail.com"},
            "expected": "success",
            "description": "Успешное добавление друга: Иван и Мария становятся взаимными друзьями"
        }
    ],
    "status": [
        {
            "input": {"user_email": "ivan@mail.com", "text": "Привет всем!"},
            "expected": "success",
            "description": "Успешная публикация статуса 'Привет всем!' пользователем Иван"
        }
```

Рисунок 12 - Код реализации SDD (часть 1)

```python
    ]
}


class SpecificationTestsSDD(unittest.TestCase):
    def setUp(self):
        self.network = SocialNetwork()
        self.network.register_user(User( username: "Иван",  email: "ivan@mail.com",  password: "123456"))
        self.network.register_user(User( username: "Мария",  email: "maria@mail.com",  password: "qwerty"))

    def test_registration_examples(self):
        for spec in SPECIFICATIONS["registration"]:
            with self.subTest(description=spec["description"]):
                if spec["expected"] == "success":
                    user = User(**spec["input"])
                    self.network.register_user(user)
                    self.assertIn(spec["input"]["email"], self.network.users)
                else:
                    with self.assertRaises(ValueError):
                        user = User(**spec["input"])
                        self.network.register_user(user)

    def test_friendship_examples(self):
        for spec in SPECIFICATIONS["friendship"]:
            with self.subTest(description=spec["description"]):
                self.network.add_friend(**spec["input"])
                user = self.network.users[spec["input"]["user_email"]]
                friend = self.network.users[spec["input"]["friend_email"]]
                self.assertIn(spec["input"]["friend_email"], user.friends)
                self.assertIn(spec["input"]["user_email"], friend.friends)
```

Рисунок 13 - Код реализации SDD (часть 2)

```
61
62      def test_status_examples(self):
63          for spec in SPECIFICATIONS["status"]:
64              with self.subTest(description=spec["description"]):
65                  self.network.post_status(**spec["input"])
66                  user = self.network.users[spec["input"]["user_email"]]
67                  self.assertEqual(len(user.statuses), second: 1)
68                  self.assertEqual(user.statuses[0]["text"], spec["input"]["text"])
69
70
71  if __name__ == "__main__":
72      unittest.main()
73
```

Рисунок 14 - Код реализации SDD (часть 3)

3.4. Результаты тестирования

3.4.1. TDD

```
(.venv) PS C:\Users\wrada\PycharmProjects\prac3> python -m unittest tests.tdd
.....
----------------------------------------------------------------------
Ran 5 tests in 0.001s

OK
```

Рисунок 15 - Результаты TDD

3.4.2. ATDD

```
(.venv) PS C:\Users\wrada\PycharmProjects\prac3> python -m unittest tests.atdd
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Рисунок 16 - Результаты ATDD

3.4.3. BDD

```
(.venv) PS C:\Users\wrada\PycharmProjects\prac3> behave tests/bdd.feature
USING RUNNER: behave.runner:Runner
Функционал: Социальная сеть # tests/bdd.feature:3

    И пользователь "user1@mail.com" также появляется в списке друзей пользователя "user2@mail.com" # tests/steps/bdd.py:21 0.000s
    И пользователь "user1@mail.com" также появляется в списке друзей пользователя "user2@mail.com" # tests/steps/bdd.py:21
1 feature passed, 0 failed, 0 skipped
1 scenario passed, 0 failed, 0 skipped
5 steps passed, 0 failed, 0 skipped
Took 0min 0.001s
```

Рисунок 17 - Результаты BDD

3.4.4. SDD

```
(.venv) PS C:\Users\wrada\PycharmProjects\prac3> python -m unittest tests.sdd
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```

Рисунок 18 - Результаты SDD

4. ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы был успешно изучен и применен комплекс современных подходов к разработке через тестирование (TDD, ATDD, BDD, SDD) на примере создания социальной сети (мини-версии), что позволило не только реализовать полнофункциональное приложение с регистрацией пользователей, системой дружбы и публикацией статусов, но и обеспечить высокое качество кода за счет полного тестового покрытия, раннего выявления ошибок и четкой спецификации требований, продемонстрировав тем самым эффективность интеграции различных методологий тестирования в современный процесс разработки программного обеспечения.