МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ

по дисциплине «Тестирование и верификация программного обеспечения»

Практическая работа №3

Студенты группы *ИКБО-50-23*, *Иващенко А.В.* _____

(подпись)

Преподаватель *Ильичев Г. П.*

_____

(подпись)

Отчет представлен «___»_____202__г.

Москва 2025 г.

1. Цели и задачи практической работы

Цель работы: изучение и применение 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. Теоретический материал

2.1 Test-Driven Development (TDD)

TDD – 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. Разработка делится на 5 этапов:

1. На первой стадии создания программного продукта разработчик создает тесты, которые описывают ожидаемое поведение новой функциональности или компонента. Они обычно создаются на основе спецификаций, требований или ожиданий от заказчика.

2. На втором этапе после создания тестов разработчик запускает их. Поскольку код, реализующий требуемую функциональность, еще не написан, все тесты должны провалиться, что ожидаемо.

3. На третьем этапе разработчик начинает писать код, который будет реализовывать функциональность. Основная цель — написать минимальный объем кода, который позволит пройти тесты.

4. После написания некоторой части кода, разработчик снова запускает тесты, из которых один или несколько должны пройти, а некоторые могут оставаться проваленными. После успешного прохождения тестов разработчик может провести рефакторинг кода. Его цель — улучшить читаемость, поддерживаемость и эффективность, но при этом не изменять функциональность будущего программного продукта.

5. Если приложение сложное, то процесс разработки будет циклическим, т.е. 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 раннее.

2.2 Acceptance Test-Driven Development (ATDD)

ATDD – 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.

Разработка делится на 5 этапов:

1. На начальном этапе все заинтересованные стороны собираются для обсуждения требований. Вместе они составляют список сценариев или тестовых случаев, описывающих, как должна работать система с точки зрения потребителя контента.

2. После того, как участники проекта (например, заказчик, бизнес-аналитик, тестировщики и разработчики) согласовали, как должна работать новая функциональность, требования оформляются в виде приёмочных сценариев. Далее они преобразуются в автоматизированные тесты, которые описывают, каким именно должно быть поведение системы при выполнении заданных условий. Поскольку в этот момент функциональность ещё не реализована, все тесты будут «падать».

3. Разработчики пишут код, обеспечивающий выполнение требований, зафиксированных в приёмочных тестах. Основная цель — реализовать функциональность так, чтобы при запуске тесты стали успешными.

4. После реализации кода тесты запускаются повторно. Если все тесты проходят, это свидетельствует о том, что функциональность соответствует ожиданиям, согласованным с заказчиком и бизнес-аналитиками.

5. При необходимости проводится рефакторинг, оптимизация кода и улучшение архитектуры, при этом тесты повторно запускаются, чтобы убедиться, что все приёмочные критерии сохранены.

2.3 Behavior-Driven Development (BDD)

Behavior-Driven Development (BDD) – 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. В рамках BDD требования к системе формулируются в виде сценариев, описывающих конкретные примеры использования, что способствует созданию общего понимания и повышению качества разрабатываемого продукта. Разработка делится на 5 этапов:

1) Формулирование сценариев на естественном языке. Они описываются с использованием структуры «Given–When–Then», где:

- Given (Дано) — начальные условия или состояние системы;
- When (Когда) — действие пользователя или событие;
- Then (Тогда) — ожидаемый результат.

2) Сценарии превращаются в автоматизированные тесты с помощью инструментов. На этом этапе сценарии запускаются — поскольку функциональность ещё не реализована, тесты проваливаются.

3) Разработчики пишут код, реализующий описанное поведение системы так, чтобы сценарии проходили успешно.

4) Сценарии запускаются снова, чтобы проверить, что система ведёт себя в соответствии с описанными ожиданиями.

5) При необходимости производится оптимизация кода, сопровождаемая повторным запуском тестов для подтверждения корректности изменений.

2.4 Specification-Driven Development (SDD)

Specification-Driven Development (SDD) — 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.

Вместе с бизнес-аналитиками и заказчиками собираются конкретные примеры использования системы, которые отражают различные сценарии работы продукта. Примеры могут оформляться в виде таблиц, диаграмм или описаний.

Примеры превращаются в спецификации, которые описывают входные данные, ожидаемые результаты и условия работы. Эти спецификации служат основой для автоматизированных тестов.

Спецификации с примерами используются для написания автоматизированных тестов. На этом этапе тесты запускаются и проваливаются, поскольку функциональность ещё не реализована.

Разработчики пишут код, обеспечивающий выполнение спецификаций, чтобы все примеры, оформленные в виде тестов, проходили успешно.

После реализации функциональности тесты повторно запускаются для проверки корректности работы. Если требуется, проводится рефакторинг кода.

3. Ход работы

3.1 Личный вариант – 13 Отдел кадров

Функции — добавление сотрудника, поиск, обновление информации, удаление записи.

TDD — тесты для CRUD-операций с данными сотрудника.

ATDD — приёмочные тесты для сценариев управления сотрудниками.

BDD — сценарий «Given сотрудник добавлен, When ищу по фамилии, Then нахожу нужного сотрудника».

SDD — спецификации с примерами входных данных и ожидаемых результатов поиска.

3.2 Подходы к тестированию

3.2.1 TDD

Листинг 1 – TDD – Test-Driven Development

```
import unittest
import tempfile, pathlib, os, uuid

class TestEmployeeCRUD(unittest.TestCase):
def setUp(self):
fd, path = tempfile.mkstemp(prefix="tdd_", suffix=".json")
os.close(fd)
self.temp = pathlib.Path(path)
self.temp.write_text("[]", encoding="utf-8")
self.mgr = EmployeeManagment(path=self.temp)

def tearDown(self):
try:
self.temp.unlink()
except PermissionError:
pass

def test_create(self):
emp = DataEmployee(
id=str(uuid.uuid4()),
first_name="Drake",
last_name="Drobov",
position="Engineer"
)
self.mgr.add(emp)
found = self.mgr.get(emp.id)
self.assertIsNotNone(found)
self.assertEqual(found.first_name, "Drake")
self.assertEqual(found.last_name, "Drobov")

def test_update(self):
emp = DataEmployee(
id=str(uuid.uuid4()),
first_name="Grigory",
last_name="Grobov",
position="Analyst"
)
self.mgr.add(emp)
updated = self.mgr.update(emp.id, position="Analyst")
self.assertTrue(updated)
self.assertEqual(self.mgr.get(emp.id).position, "Analyst")
def test_delete(self):
emp = DataEmployee(
id=str(uuid.uuid4()),
```

```
first_name="Lena",
last_name="Varkinovna",
position="Designer"
)
self.mgr.add(emp)
deleted = self.mgr.delete(emp.id)
self.assertTrue(deleted)
self.assertIsNone(self.mgr.get(emp.id))

def test_find_by_last_name(self):
emp1 = DataEmployee(
id=str(uuid.uuid4()),
first_name="Alexander",
last_name="Alexandrov",
position="Manager"
)
emp2 = DataEmployee(
id=str(uuid.uuid4()),
first_name="Igor",
last_name="Igorev",
position="Developer"
)
self.mgr.add(emp1)
self.mgr.add(emp2)
results = self.mgr.find_by_last_name("Alexandrov")
self.assertEqual(len(results),1)
self.assertEqual(results[0].first_name, "Alexander")

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

```
========== RESTART: C:\Users\Alexander\Downloads\tdd_test.py ==========
EEEE
======================================================================
ERROR: test_create (__main__.TestEmployeeCRUD.test_create)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\Alexander\Downloads\tdd_test.py", line 10, in setUp
    self.mgr = EmployeeManagment(path=self.temp)
               ^^^^^^^^^^^^^^^^^
NameError: name 'EmployeeManagment' is not defined
```

Рисунок 1 – Запуск тестирования

Теперь для нашего тестирования пропишем функционал, который в последствии будем использовать для остальных тестирований.

Создадим отдельно дата-класс для хранения данных сотрудников.

```
@dataclass
class DataEmployee:
    id: str
    first_name: str
    last_name: str
    position: str
    email: str = ""
    phone: str = ""
    hired_date: str = ""
```

Рисунок 2 – Дата-класс DataEmployee

Для работы с множеством сотрудников и реализации основного функционала создадим основной класс управления сотрудниками.

```
class EmployeeManagment:
    def __init__(self, path="employees.json"):
        self.path = path
        self._employees = []
        self.load()
```

Рисунок 3 – Класс EmployeeManagment

Реализуем требуемые для функционала методы:

```
def add(self, emp: DataEmployee):
    self._employees.append(emp)
    self.save()
```

Рисунок 4 – метод add для добавления данных сотрудника

```
def find_by_last_name(self, last_name):
    result = []
    for emp in self._employees:
        if emp.last_name.lower() == last_name.lower():
            result.append(emp)
    return result
```

Рисунок 5 – метод find_by_last_name для поиска сотрудника по фамилии

```
def update(self, emp_id, **fields):
    emp = self.get(emp_id)
    if not emp:
        return False
    for i, j in fields.items():
        if hasattr(emp, i):
            setattr(emp, i, j)
    self.save()
    return True
```

Рисунок 6 – метод update для обновления данных сотрудников

```
def delete(self, emp_id):
    length = len(self._employees)
    self._employees = []
    for employee in self._employees:
        if employee.id != emp_id:
            self._employees.append(employee)
    if (len(self._employees) < length):
        self.save()
        return True
    return False
```

Рисунки 7 – метод delete для удаления данных сотрудника

Также пропишем дополнительные методы, не требуемые по варианту, для гибкой работы с кодом.

```
def save(self):
    data = []
    for emp in self._employees:
        data.append(asdict(emp))
    with open(self.path, "w", encoding="utf-8") as f:
        json.dump(data, f, ensure_ascii=False, indent=2)

def load(self):
    emps = []
    if not self.path.exists():
        self._employees = []
        return
    with open(self.path, "r", encoding="utf-8") as f:
        inf = json.load(f)
    for emp in inf:
        emps.append(DataEmployee(**emp))
    self._employees = emps
```

Рисунок 8 – методы save и load для сохранения данных в файл и загрузки этих данных из файла

```
def get(self, emp_id):
    for emp in self._employees:
        if emp.id == emp_id:
            return emp
    return None
```

Рисунок 9 – метод get(emp_id) для получения данных по сотруднику по его id

После добавления функционала проведем тестирование повторно.

```
============== RESTART: C:\Users\Alexander\Downloads\tdd_test.py ==============
....
----------------------------------------------------------------------
Ran 4 tests in 0.015s

OK
```

Рисунок 10 – Результат Тестирования

3.2.2 ATDD

Листинг 2 – ATDD – Acceptance Test-Driven Development

```
import tempfile, pathlib, os, uuid

def create_temp_manage():
    fd, path = tempfile.mkstemp(prefix="atdd_", suffix=".json")
    os.close(fd)
    temp = pathlib.Path(path)
    temp.write_text("[]", encoding="utf-8")
    mgr = EmployeeManagment(path=temp)
    return temp, mgr

def test_add_and_find_employee_acceptance():
    temp, mgr = create_temp_manage()
    try:
        emp = DataEmployee(
            id=str(uuid.uuid4()),
            first_name="Frax",
            last_name="Klak",
            position="Manager"
        )
        mgr.add(emp)

        results = mgr.find_by_last_name("Klak")
        assert len(results) == 1
        assert results[0].first_name == "Frax"
    finally:
        try:
            temp.unlink()
        except PermissionError:
            pass

def test_update_and_delete_employee_acceptance():
    temp, mgr = create_temp_manage()
    try:
        emp = DataEmployee(
            id=str(uuid.uuid4()),
            first_name="Rodion",
            last_name="Raskolnikov",
            position="Student"
        )
        mgr.add(emp)
        updated = mgr.update(emp.id, position="Student")
        assert updated
        assert mgr.get(emp.id).position == "Student"
        deleted = mgr.delete(emp.id)
        assert deleted
        assert mgr.get(emp.id) is None
```

```
    finally:
        try:
            temp.unlink()
        except PermissionError:
            pass
```

```
C:\Users\Alexander\Downloads>python -m pytest atdd_test.py
============================================= test session starts =============================================
platform win32 -- Python 3.12.10, pytest-8.3.4, pluggy-1.5.0
rootdir: C:\Users\Alexander\Downloads
collected 2 items

atdd_test.py FF                                                                                      [100%]

================================================== FAILURES ===================================================
_____________________________________ test_add_and_find_employee_acceptance ______________________________________

    def test_add_and_find_employee_acceptance():
>       temp, mgr = create_temp_manage()

atdd_test.py:12:
```

Рисунок 11 – Запуск тестирования

Воспользуемся функционалом из работы с TDD для работы с ATDD и запустим тестирование снова.

```
C:\Users\Alexander\Downloads>python -m pytest atdd_test.py
============================================= test session starts =============================================
platform win32 -- Python 3.12.10, pytest-8.3.4, pluggy-1.5.0
rootdir: C:\Users\Alexander\Downloads
collected 2 items

atdd_test.py ..                                                                                      [100%]

============================================== 2 passed in 0.03s ==============================================
```

Рисунок 12 – Результат тестирования

3.2.3 BDD

```
Feature: Поиск сотрудника по фамилии
  Scenario: Успешный поиск по фамилии
    Given сотрудник "Ivan Ivanov" с должностью "Engineer" добавлен в систему
    When я выполняю поиск сотрудника по фамилии "Ivanov"
    Then я нахожу одного сотрудника с именем "Ivan"
```

Рисунок 13 – Сценарий на языке Gherkin

Листинг 3 – BDD – Behavior-Driven Development

```
import sys, os, json, tempfile, pathlib, uuid
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__
file__), "../../")))

from behave import given, when, then

@given('сотрудник "{first_name} {last_name}" с должностью
"{position}" добавлен в систему')
def step_add_employee(context, first_name, last_name,
position):
    fd, path = tempfile.mkstemp(prefix="bdd_", suffix=".json")
    os.close(fd)
    context.temp = pathlib.Path(path)
    context.temp.write_text("[]", encoding="utf-8")
    context.mgr = EmployeeManagment(path=context.temp)
    emp = DataEmployee(
        id=str(uuid.uuid4()),
        first_name=first_name,
        last_name=last_name,
        position=position
    )
    context.mgr.add(emp)

@when('я выполняю поиск сотрудника по фамилии "{last_name}"')
def step_search(context, last_name):
    context.results = context.mgr.find_by_last_name(last_name)

@then('я нахожу одного сотрудника с именем "{expected}"')
def step_check(context, expected):
    assert len(context.results) == 1
    assert context.results[0].first_name == expected
    try:
        context.temp.unlink(missing_ok=True)
    except PermissionError:
        pass
```

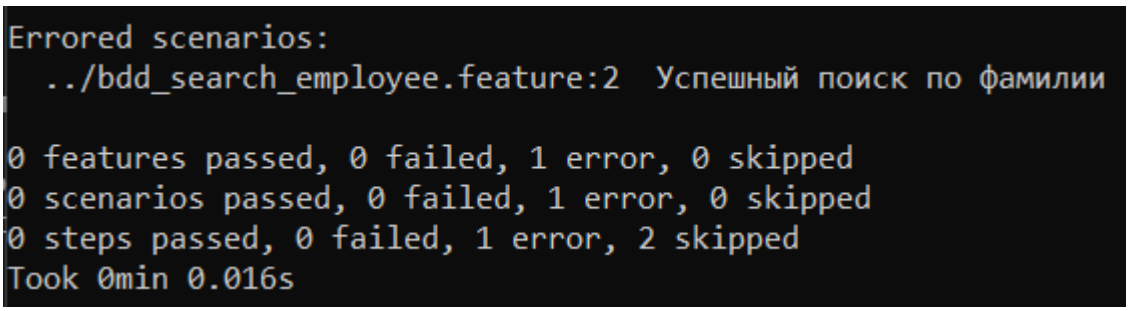

Рисунок 13 – Запуск тестирования

Воспользуемся функционалом из работы с TDD для работы с BDD и запустим тестирование снова.

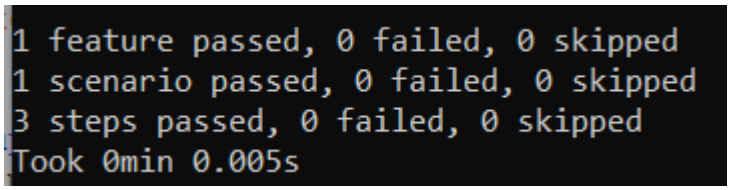

Рисунок 14 – Результат тестирования

3.2.4 SDD

Таблица 1 – Описание тестирований

Действие	Входные данные	Ожидаемый результат
Добавление сотрудника (add)	first_name: "Ivan", last_name: "Ivanov", position: "Engineer", email: "", phone: ""	– (Информация по сотруднику добавлена)
Поиск сотрудника по фамилии (find_by_last_name)	last_name: "Ivanov"	Ivan (Список сотрудников с заданной фамилией)
Обновление информации по сотруднику (update)	fields={position: "Senior Engineer"}	– (Обновлена информация о сотруднике)
Удаление сотрудника (delete)	emp_id=1	True (Сотрудник удален, при его наличии)

Листинг 4 – SDD – Specification-Driven Development

```
from dataclasses import dataclass, asdict
import tempfile, pathlib, os, uuid
import pytest

SDD_SPECIFICATIONS = [
    {
        "action": "add",
        "input": {"first_name": "Ivan", "last_name": "Ivanov",
"position": "Engineer", "email": "", "phone": ""},
        "expected": {
            "fields": ["id", "first_name", "last_name",
"position", "email", "phone", "hired_date"],
            "first_name": "Ivan",
            "last_name": "Ivanov"
        }
    },
    {
        "action": "find_by_last_name",
        "input": {"last_name": "Ivanov"},
        "expected": {"count": 1, "first_name": "Ivan"}
    },
```

```
    {
        "action": "update",
        "input": {"fields": {"position": "Senior Engineer"}},
        "expected": {"updated": True, "position": "Senior
Engineer"}
    },
    {
        "action": "delete",
        "expected": {"deleted": True}
    }
]

@pytest.mark.parametrize("spec", SDD_SPECIFICATIONS)
def test_sdd_new_version(spec):
    fd, path = tempfile.mkstemp(prefix="sdd_", suffix=".json")
    os.close(fd)
    db_path = pathlib.Path(path)
    db_path.write_text("[]", encoding="utf-8")
    mgr = EmployeeManagment(path=db_path)
    try:
        if spec["action"] == "add":
            inp = spec["input"]
            emp = DataEmployee(id=str(uuid.uuid4()), **inp,
hired_date="")
            mgr.add(emp)
            for field in spec["expected"]["fields"]:
                assert hasattr(emp, field)
            assert emp.first_name ==
spec["expected"]["first_name"]
            assert emp.last_name ==
spec["expected"]["last_name"]

        elif spec["action"] == "find_by_last_name":
            emp = DataEmployee(id=str(uuid.uuid4()),
first_name="Ivan", last_name="Ivanov", position="Engineer")
            mgr.add(emp)
            results =
mgr.find_by_last_name(spec["input"]["last_name"])
            assert len(results) == spec["expected"]["count"]
            if results:
                assert results[0].first_name ==
spec["expected"]["first_name"]

        elif spec["action"] == "update":
            emp = DataEmployee(id=str(uuid.uuid4()),
first_name="Ivan", last_name="Ivanov", position="Engineer")
            mgr.add(emp)
            updated = mgr.update(emp.id,
```

```
**spec["input"]["fields"])
            assert updated == spec["expected"]["updated"]
            for k, v in spec["input"]["fields"].items():
                assert getattr(emp, k) == v

        elif spec["action"] == "delete":
            emp = DataEmployee(id=str(uuid.uuid4()),
first_name="Ivan", last_name="Ivanov", position="Engineer")
            mgr.add(emp)
            deleted = mgr.delete(emp.id)
            assert deleted == spec["expected"]["deleted"]
            assert mgr.get(emp.id) is None

    finally:
        try:
            db_path.unlink()
        except PermissionError:
            pass

if __name__ == "__main__":
    import sys
    sys.exit(pytest.main([__file__]))
```

```
C:\Users\Alexander\Downloads>python -m pytest sdd_test.py
============================= test session starts =============================
platform win32 -- Python 3.12.10, pytest-8.3.4, pluggy-1.5.0
rootdir: C:\Users\Alexander\Downloads
collected 4 items

sdd_test.py FFFF                                                         [100%]

================================== FAILURES ===================================
_____________________________ test_sdd_new_version[spec0] _____________________
```

Рисунок 15 – Запуск тестирования

Воспользуемся функционалом из работы с TDD для работы с BDD и запустим тестирование снова.

```
C:\Users\Alexander\Downloads>python -m pytest sdd_test.py
============================= test session starts =============================
platform win32 -- Python 3.12.10, pytest-8.3.4, pluggy-1.5.0
rootdir: C:\Users\Alexander\Downloads
collected 4 items

sdd_test.py ....                                                         [100%]

============================== 4 passed in 0.04s ==============================
```

Рисунок 16 – Результат тестирования

4. Заключение

В ходе практической работы были изучены и освоены подходы к разработке через тестирование: TDD, ATDD, BDD, SDD; на основе выданного варианта. При помощи тестирований мы смогли в полной мере реализовать по заданному функционалу код.