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»
Команда № 6

Состав: Зенцова Е. Д, Каушина А. В.

Практическое занятие № 2

Студент группы *ИКБО-50-23, Каушина А.В., Зенцова Е.Д.* _____ (подпись)

Преподаватель *Ильичев Г.П.* _____ (подпись)

Отчет представлен « » сентября 2025 г.

Москва 2025 г.

Цели и задачи

Цель работы: познакомить студентов с процессом модульного и мутационного тестирования, включая разработку, проведение тестов, исправление ошибок, анализ тестового покрытия, а также оценку эффективности тестов путём применения методов мутационного тестирования.

Для достижения поставленной цели работы студентам необходимо выполнить ряд **задач**:

− изучить основы модульного тестирования и его основные принципы;

− освоить использование инструментов для модульного тестирования (pytest для Python, JUnit для Java и др.);

− разработать модульные тесты для программного продукта и проанализировать их покрытие кода; − изучить основы мутационного тестирования и освоить инструменты для его выполнения (MutPy, PIT, Stryker);

− применить мутационное тестирование к программному продукту, оценить эффективность тестов;

− улучшить существующий набор тестов, ориентируясь на результаты мутационного тестирования;

− оформить итоговый отчёт с результатами проделанной работы

Практическая часть

1. Разработка модуля

Программа для работы с массивами представляет собой консольное приложение с интерактивным меню. Пользователь выбирает действие через меню, вводя соответствующую цифру. Программа обрабатывает некорректный ввод и предоставляет возможность повторно ввести массив.

Основные функции (выбор пользователя)

1. find_max_value(arr): находит максимальное значение в переданном массиве чисел. Параметры: arr — список чисел (list), в котором нужно найти максимум. Возвращает максимальный элемент списка, если список не пуст,

None, если список пуст.

2. find_sum(arr): вычисляет сумму всех элементов в массиве. Параметры: arr — список чисел, элементы которого нужно просуммировать. Возвращает сумму всех чисел в списке.

3. calculate_average(arr): вычисляет среднее арифметическое элементов массива. Параметры: arr — список чисел, для которых нужно вычислить среднее значение. Возвращает среднее значение, если массив не пуст, 0, если массив пуст.

4. bubble_sort(arr): сортирует массив методом пузырька по возрастанию. Параметры: arr — список чисел, который нужно отсортировать. Возвращает новый отсортированный список, не изменяя исходный.

5. check_unique(arr): проверяет, уникальны ли все элементы в массиве. Параметры: arr — список чисел, элементы которого нужно проверить. Возвращает True, если все элементы уникальны, иначе False.

6. enter_arr(): организует ввод массива пользователем через консоль. Параметры: не принимает аргументов. Возвращает список чисел, введённых пользователем.

В разработке программы была допущена ошибка – функция bubble_sort(arr): сортирует не по возрастанию, как указано в программе, а по убыванию.

Листинг 1 – исходный код программы

```
def find_max_value(arr):
    """Нахождение максимального значения в массиве"""
    if not arr:
        return None
    max_value = arr[0]
    for num in arr:
        if num > max_value:
            max_value = num
    return max_value


def find_sum(arr):
    """Нахождение суммы всех элементов массива"""
    return sum(arr)


def calculate_average(arr):
    """Нахождение среднего значения элементов массива"""
    if not arr:
        return 0
```

```
    return sum(arr) / len(arr)


def bubble_sort(arr):
    """Сортировка массива методом пузырька по возрастанию"""
    n = len(arr)
    sorted_arr = arr.copy()
    for i in range(n):
        for j in range(0, n - i - 1):
            if sorted_arr[j] < sorted_arr[j + 1]:
                sorted_arr[j], sorted_arr[j + 1] = sorted_arr[j + 1],
sorted_arr[j]
    return sorted_arr


def check_unique(arr):
    """Проверка уникальности элементов в массиве"""
    unique_set = set(arr)
    return len(unique_set) == len(arr)


def enter_arr():
    """Ввод нового массива"""
    while True:
        try:
            input_str = input("Введите элементы массива через пробел: ")
            arr = [float(x) for x in input_str.split()]
            return arr
        except ValueError:
            print("Ошибка ввода, введите числа через пробел:")


def main():
    arr = enter_arr()
    while True:
        print("Выберите действие:\n1. Найти максимальное значение\n2. Найти
сумму элементов\n"
              "3. Вычислить среднее значение\n4. Отсортировать массив по
возрастанию\n"
              "5. Проверить уникальность элементов\n6. Ввести новый
массив\n0. Выход")
        choice = input("Введите номер действия (0-6): ")

        if choice == '1':
            result = find_max_value(arr)
            print(f"Максимальное значение: {result if result is not None else
'Массив пуст'}")
        elif choice == '2':
            result = find_sum(arr)
            print(f"Сумма элементов: {result}")
        elif choice == '3':
            result = calculate_average(arr)
            print(f"Среднее значение: {result}")
        elif choice == '4':
            result = bubble_sort(arr)
            print(f"Отсортированный массив: {result}")
        elif choice == '5':
            result = check_unique(arr)
            print(f"Уникальны ли элементы: {result}")
        elif choice == '6':
            arr = enter_arr()
            print("Массив успешно введен")
        elif choice == '0':
            print("Выход")
```

```
            return
        else:
            print("Неверный выбор, введите номер действия (0-6): ")


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

2. Модульное тестирование

Проверка функции generate_password():

test_generate_password_length: убеждается, что длина сгенерированного пароля соответствует запрошенному количеству символов.

test_generate_password_content: проверяет, что при включении всех опций пароль содержит хотя бы один символ каждого типа.

test_generate_password_no_uppercase, test_generate_password_no_numbers, test_generate_password_no_special — контролируют ситуации, когда некоторые категории символов должны быть исключены.

Проверка функции check_password_strength():

test_check_password_strength_simple: проверяет, что короткие и однотипные пароли определяются как простые.

test_check_password_strength_hard: убеждается, что пароли, содержащие разные типы символов и достаточную длину, определяются как сложные.

Проверка функции save_password():

test_save_password_writes_password: проверяет, что при сохранении нового пароля он корректно записывается в файл и содержимое соответствует ожидаемому.

test_save_password_new_line: проверяет, что при многократном вызове функции новые пароли добавляются построчно, а не перезаписывают старые данные.

Проверка функции add_symbol_to_password():

test_add_symbol_to_password_valid: имитирует ввод корректного символа (одного знака) и проверяет, что новый символ действительно добавляется к паролю.

test_add_symbol_to_password_invalid: проверяет обработку ошибочной ситуации, когда пользователь вводит более одного символа.

Проверка функции check_repeated_chars():

test_check_repeated_chars_true: использует пароли, содержащие повторяющиеся символы, и проверяет, что функция корректно их обнаруживает, возвращая True.

test_check_repeated_chars_false: проверяет пароли без повторов.

Листинг 2 – модульные тесты

```
import os, pytest
from unittest.mock import patch

from password_mutant import generate_password, check_password_strength,
add_symbol_to_password, check_repeated_chars, \
    save_password


# проверка генерации пароля
def test_generate_password_length():
    """Проверка на корректное количество символов в пароле"""
    password = generate_password(10, True, True, True)
    assert len(password) == 10
    password = generate_password(8, False, False, False)
    assert len(password) == 8


def test_generate_password_content():
    """Проверка на включение необходимых символов в пароль"""
    password = generate_password(8, True, True, True)
    assert any(c.islower() for c in password)
    assert any(c.isupper() for c in password)
    assert any(c.isdigit() for c in password)
    assert any(c in "!@#$%" for c in password)


def test_generate_password_no_uppercase():
    """Проверка генерации без заглавных букв"""
    password = generate_password(8, False, True, True)
    assert any(c.islower() for c in password)
    assert not any(c.isupper() for c in password)
    assert any(c.isdigit() for c in password)
    assert any(c in "!@#$%" for c in password)


def test_generate_password_no_numbers():
    """Проверка генерации без цифр"""
    password = generate_password(8, True, False, True)
    assert any(c.islower() for c in password)
    assert any(c.isupper() for c in password)
    assert not any(c.isdigit() for c in password)
    assert any(c in "!@#$%" for c in password)


def test_generate_password_no_special():
    """Проверка генерации без специальных символов"""
    password = generate_password(8, True, True, False)
    assert any(c.islower() for c in password)
```

```
    assert any(c.isupper() for c in password)
    assert any(c.isdigit() for c in password)
    assert not any(c in "!@#$%" for c in password)


# проверка оценки сложности пароля
def test_check_password_strength_simple():
    """Проверка сложности простых паролей"""
    password = "abcde"
    assert check_password_strength(password) == "Простой"
    password = "12345"
    assert check_password_strength(password) == "Простой"


def test_check_password_strength_hard():
    """Проверка сложности сложных паролей"""
    password = "Ab1!"
    assert check_password_strength(password) == "Сложный"
    password = "qwerty123!"
    assert check_password_strength(password) == "Сложный"


# проверка сохранения в файл
def test_save_password_writes_password(tmp_path):
    """Проверка сохранения пароля в файл"""
    test_file = tmp_path / "passwords.txt"
    os.chdir(tmp_path)
    save_password("password123")
    content = test_file.read_text().strip()
    assert content == "password123"


def test_save_password_new_line(tmp_path):
    """Проверка записи новых паролей в один файл"""
    test_file = tmp_path / "passwords.txt"
    os.chdir(tmp_path)
    save_password("first_pw")
    save_password("second_pw")
    lines = test_file.read_text().splitlines()
    assert lines == ["first_pw", "second_pw"]


# проверка добавления символа в пароль
@patch('builtins.input', return_value='@')
def test_add_symbol_to_password_valid(mock_input):
    """Проверка добавления символа в пароль"""
    password = "abc123"
    new_password = add_symbol_to_password(password)
    assert new_password == password + "@"


@patch('builtins.input', return_value='abc')
def test_add_symbol_to_password_invalid(mock_input):
    """Проверка ошибки добавления символа в пароль"""
    password = "abc123"
    new_password = add_symbol_to_password(password)
    assert new_password == password


# проверка на уникальность элементов
def test_check_repeated_chars_true():
    """Проверка на наличие повторяющихся символов"""
    password = "aabbccdd"
    assert check_repeated_chars(password) is True
```

```
    password = "qw123rt456yu1"
    assert check_repeated_chars(password) is True


def test_check_repeated_chars_false():
    """Проверка на отсутствие повторяющихся символов"""
    password = "abcdefgh"
    assert check_repeated_chars(password) is False
    password = "lowkeysam"
    assert check_repeated_chars(password) is False
```

Рисунок 1 - результат тестирования

Описание найденной ошибки: «Деление на ноль в функции проверки сложности пароля»

Статус ошибки: открыта («Open»)

Категория ошибки: критическая («Critical»)

Тестовый случай: «Проверка алгоритма оценки сложности пароля»

Описание ошибки:

1. Вызвать функцию check_password_strength() с любым паролем
2. В качестве параметра передать строку с паролем
3. Полученный результат: исключение ZeroDivisionError
4. Ожидаемый результат: строка "Сложный" или "Простой" в зависимости от анализа пароля

Исправленная программа

Листинг 3 – исправленная программа

```
import random


def generate_password(length, use_big_letters=True, use_numbers=True,
use_special=True):
    small_letters = "abcdefghijklmnopqrstuvwxyz"
    big_letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" if use_big_letters else ""
    numbers = "0123456789" if use_numbers else ""
    special_chars = "!@#$%" if use_special else ""

    all_chars = small_letters + big_letters + numbers + special_chars
```

```
    password = ""
    password += random.choice(small_letters)
    if use_big_letters:
        password += random.choice(big_letters)
    if use_numbers:
        password += random.choice(numbers)
    if use_special:
        password += random.choice(special_chars)

    for _ in range(length - len(password)):
        password += random.choice(all_chars)

    return (password)


def check_password_strength(password):
    length = len(password)
    has_big = any(c.isupper() for c in password)
    has_small = any(c.islower() for c in password)
    has_num = any(c.isdigit() for c in password)
    has_spec = any(c in "!@#$%" for c in password)

    score = length + (has_big + has_small + has_num + has_spec) * 10

    if score > 15:
        return "Сложный"
    return "Простой"


def save_password(password):
    with open("passwords.txt", "a") as f:
        f.write(password + "\n")
    print("Пароль сохранён в файл passwords.txt")


def add_symbol_to_password(password):
    new_symbol = input("Введите символ, который хотите добавить: ")
    if len(new_symbol) != 1:
        print("Можно добавить только один символ!")
        return password
    password += new_symbol
    print("Новый пароль:", password)
    return password


def check_repeated_chars(password):
    repeats = []
    for i in range(len(password)):
        for j in range(i + 1, len(password)):
            if password[i] == password[j] and password[i] not in repeats:
                repeats.append(password[i])
    if repeats:
        print(f"Повторяющиеся символы найдены: {' '.join(repeats)}")
        return True
    else:
        print("Повторяющихся символов не найдено.")
        return False


if __name__ == "__main__":
    while True:
        try:
            length = int(input("Введите длину пароля (минимум 8): "))
```

```
            if length < 8:
                print("Длина пароля должна быть не менее 8 символов!")
                continue
        except ValueError:
            print("Введите число!")
            continue

        while True:
            bigletter = input("Использовать заглавные буквы? (да/нет):
").strip().lower()
            if bigletter in ["да", "нет"]:
                bigletter = bigletter == "да"
                break
            print("Пожалуйста, введите только 'да' или 'нет'!")

        while True:
            num = input("Использовать цифры? (да/нет): ").strip().lower()
            if num in ["да", "нет"]:
                num = num == "да"
                break
            print("Пожалуйста, введите только 'да' или 'нет'!")

        while True:
            spec = input("Использовать специальные символы? (да/нет):
").strip().lower()
            if spec in ["да", "нет"]:
                spec = spec == "да"
                break
            print("Пожалуйста, введите только 'да' или 'нет'!")

        password = generate_password(length, bigletter, num, spec)
        if not password:
            continue

        print("\nВаш пароль:", password)

        while True:
            print("\nВыберите действие:\n1 — Проверить сложность пароля\n"
                  "2 — Проверить на повторяющиеся символы\n3 — Добавить новый
символ в пароль"
                  "\n4 — Сохранить пароль в файл\n"
                  "0 — Сгенерировать новый пароль / выйти")
            choice = input("Ваш выбор: ").strip()

            if choice == "1":
                strength = check_password_strength(password)
                print("Сложность пароля:", strength)
            elif choice == "2":
                repeated = check_repeated_chars(password)
            elif choice == "3":
                password = add_symbol_to_password(password)
            elif choice == "4":
                save_password(password)
            elif choice == "0":
                while True:
                    again = input("Хотите сгенерировать новый пароль?
(да/нет): ").strip().lower()
                    if again in ["да", "нет"]:
                        if again == "да":
                            break
                        else:
                            print("До свидания!")
                            exit()
                    print("Пожалуйста, введите только 'да' или 'нет'!")
```

```
            else:
                print("Неверный выбор, попробуйте снова.")
```

3. Мутационное тестирование

Рисунок 2 - результат тестирования

Описание мутантов

1. generate_password: строчные и заглавные буквы перепутаны местами; количество символов в пароле вычисляется неверно.

Мутант выжил в 3 из 5 тестов. Тесты проверяют наличие символов, но не обнаруживают проблему с длиной.

Листинг 4 – мутант

```
def generate_password(length, use_big_letters=True, use_numbers=True,
use_special=True):
    small_letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    big_letters = "abcdefghijklmnopqrstuvwxyz" if use_big_letters else ""
    numbers = "0123456789" if use_numbers else ""
    special_chars = "!@#$%" if use_special else ""

    all_chars = small_letters + big_letters + numbers + special_chars

    password = ""
    password += random.choice(small_letters)
    if use_big_letters:
        password += random.choice(big_letters)
    if use_numbers:
        password += random.choice(numbers)
    if use_special:
        password += random.choice(special_chars)

    for _ in range(length - 1):
        password += random.choice(all_chars)

    return password
```

2. check_password_strength: изменен порог с 15 до 25, а также логика условия была перевернута (< вместо >).

Все тесты оценки сложности правильно обнаружили инвертированную логику.

Листинг 5 – мутант

```
def check_password_strength(password):
    length = len(password)
    has_big = any(c.isupper() for c in password)
    has_small = any(c.islower() for c in password)
    has_num = any(c.isdigit() for c in password)
    has_spec = any(c in "!@#$%" for c in password)

    score = length + (has_big + has_small + has_num + has_spec) * 10

    if score < 25:
        return "Сложный"
    return "Простой"
```

3. save_password: изменен режим записи – с дозаписи в файл на перезапись.

Базовый тест записи прошел (так как для единственной записи дозапись и перезапись одинаковы), но тест на множественную запись провалился.

Листинг 6 – мутант

```
def save_password(password):
    with open("passwords.txt", "w") as f:
        f.write(password + "\n")
    print("Пароль сохранён в файл passwords.txt")
```

4. add_symbol_to_password: условие проверки корректности введенного символа ослаблено.

Тест с правильным вводом прошел, так как правильный ввод не подразумевает попадание в мутацию. Тест с неправильным вводом прошел (мутант выжил), так как условие добавления только одного символа не выполнено.

Листинг 7 – мутант

```
def add_symbol_to_password(password):
    new_symbol = input("Введите символ, который хотите добавить: ")
    if len(new_symbol) > 10:
        print("Можно добавить только один символ!")
        return password
    password += new_symbol
    print("Новый пароль:", password)
    return password
```

5. check_repeated_chars: проверка только соседних символов на повторение.

Листинг 8 – мутант

```
def check_repeated_chars(password):
    repeats = []
    for i in range(len(password)):
```

```
        for j in range(i + 1, min(i+2, len(password))):
            if password[i] == password[j] and password[i] not in repeats:
                repeats.append(password[i])
    if repeats:
        print(f"Повторяющиеся символы найдены: {' '.join(repeats)}")
        return True
    else:
        print("Повторяющихся символов не найдено.")
        return False
```

Мутант выжил частично: тест с повторяющимися символами падает, так как не всегда повторяются соседние элементы, а тест с уникальными символами проходит, так как повторы и так отсутствуют.

По результатам мутационного тестирования был добавлен тест для проверки полной генерации пароля.

Листинг 9 – добавленные тесты

```
def test_generate_password_full():
    """Проверка генерации с корректной длиной и необходимыми символами"""
    password = generate_password(10, True, False, True)
    assert len(password) == 10
    assert any(c.islower() for c in password)
    assert not any(c.isupper() for c in password)
    assert any(c.isdigit() for c in password)
    assert any(c in "!@#$%" for c in password)
```

Заключение

В ходе проведённого тестирования модуля было реализовано 14 тестов. Тесты охватывают все ключевые функции программы: генерацию паролей, проверку их сложности, сохранение в файл, добавление символа и анализ на повторяющиеся символы. Для оценки эффективности тестирования применялась методика мутационного анализа, в ходе которой в исходный код были намеренно внесены логические ошибки (мутации). Из пяти внедрённых мутантов тестами были обнаружены все, что можно считать высоким показателем качества тестового покрытия.

Были выявлены недостатки в текущем наборе тестов и логике программы: не хватает тестов на точную проверку длины пароля при изменении параметров генерации. В некоторых мутациях изменение цикла не вызывало сбоя теста, что говорит о необходимости добавить дополнительную проверку или отдельный тест на проверку полной генерации.