



МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное учреждение
высшего образования*

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий (ИТ)

Кафедра Математического обеспечения и стандартизации информационных
технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 2

по дисциплине

«Тестирование и верификация программного обеспечения»

**Тема: «Модульное и мутационное тестирование программного
продукта»**

Выполнили студенты группы

ИКБО-43-23

Зепалов Е.В.

Зернов Н.А.

Коротаев А.М.

Любишкин С.Н.

Степанов К.К.

Принял

Ильичев Г. П.

Практическая работа выполнена

«10»_10__2025г.

(подпись студента)

«Зачтено»

«__»_10__2025 г.

(подпись руководителя)

Москва 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1 Выполнения практической работы Зернова Н.А.	6
1.1 Разработка модуля	6
1.1.1 Описание функциональности	7
1.2 Модульное тестирование	8
1.2.1 Методология	8
1.2.2 Анализ покрытия	9
1.2.3 Исправление ошибок	9
1.3 Мутационное тестирование	10
1.3.1 Анализ выживаемости	12
1.3.2 Корректировка тестов	12
2 Выполнения практической работы Любишкина С.Н.	13
2.1 Разработка модуля	13
2.1.1 Описание функциональности	13
2.2 Модульное тестирование	13
2.2.1 Методология	13
2.2.2 Анализ покрытия	13
2.2.3 Исправление ошибок	14
2.3 Мутационное тестирование	14
2.3.1 Анализ выживаемости	14
2.3.2 Корректировка тестов	14
3 Выполнения практической работы Коротаева А.М.	16
3.1 Разработка модуля	16
3.1.1 Описание функциональности	16
3.2 Модульное тестирование	16
3.2.1 Методология	16
3.2.2 Анализ покрытия	16
3.2.3 Исправление ошибок	17

3.3 Мутационное тестирование	17
3.3.1 Анализ выживаемости	17
3.3.2 Корректировка тестов	17
4 Выполнения практической работы Степанова К.К.	19
4.1 Разработка модуля	19
4.1.1 Описание функциональности	19
4.2 Модульное тестирование	19
4.2.1 Методология	19
4.2.2 Анализ покрытия	19
4.2.3 Исправление ошибок	20
4.3 Мутационное тестирование	20
4.3.1 Анализ выживаемости	20
4.3.2 Корректировка тестов	20
5 Выполнения практической работы Зепалова Е.В.	22
5.1 Разработка модуля	22
5.1.1 Описание функциональности	22
5.2 Модульное тестирование	22
5.2.1 Методология	22
5.2.2 Анализ покрытия	22
5.2.3 Исправление ошибок	23
5.3 Мутационное тестирование	23
5.3.1 Анализ выживаемости	23
5.3.2 Корректировка тестов	23
ЗАКЛЮЧЕНИЕ	25

ВВЕДЕНИЕ

Практическая работа была направлена на изучение процесса модульного и мутационного тестирования. Основная цель заключалась в освоении полного цикла тестирования программного продукта: разработка и выполнение тестов, исправление выявленных ошибок, анализ покрытия кода, а также оценка эффективности тестов с применением методов мутационного тестирования.

Для достижения поставленной цели были выполнены следующие задачи:

- 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.

1 ВЫПОЛНЕНИЕ ПРАКТИЧЕСКОЙ РАБОТЫ ЗЕРНОВА Н.А.

1.1 Разработка модуля

Листинг 1.1.1 кода операции над массивами, который впоследствии будет тестироваться, приведён ниже.

Листинг 1.1.1 — Операции над массивами

```
from typing import List, Any

def find_max(arr: List[float]) -> float:
    """
    Поиск максимума
    """
    if not arr:
        raise ValueError("Массив пуст")
    max_val = arr[0]
    for num in arr:
        if num > max_val:
            max_val = num
    return max_val

def bubble_sort(arr: List[float]) -> List[float]:
    """
    Сортировка пузырьком
    """
    n = len(arr)
    res = arr[:]
    for i in range(n):
        for j in range(0, n - i - 1):
            if res[j] > res[j + 1]:
                res[j], res[j + 1] = res[j + 1], res[j]
    return res

def reverse_array(arr: List[Any]) -> List[Any]:
    """
    Реверс массива
    """
    return arr[::-1]

def array_sum(arr: List[float]) -> float:
    """
    Сумма элементов массива
    """
    return sum(arr)

def calculate_average(arr: List[float]) -> float:
    """
    Avg массива...
    """
    if not arr:
        raise ValueError("Массив пуст")
    return sum(arr) / (len(arr) - 1)
```

Продолжение листинга 1.1.1

```
def remove_duplicates(arr: List[Any]) -> List[Any]:  
    """  
    Уникальные элементы массива  
    """  
    seen = set()  
    res = []  
    for x in arr:  
        if x not in seen:  
            seen.add(x)  
            res.append(x)  
    return res
```

1.1.1 Описание функциональности

Код представляет собой набор функций для работы с массивами чисел. . Функция `find_max` предназначена для поиска максимального элемента массива, при этом в случае передачи пустого массива возбуждается исключение `ValueError`, что позволяет контролировать корректность входных данных. Функция `bubble_sort` реализует сортировку массива методом пузырька по возрастанию и возвращает новый отсортированный массив, не изменяя исходных данных, что упрощает дальнейшую работу с ними. Для инверсии порядка элементов предусмотрена функция `reverse_array`, которая возвращает массив в обратном порядке. Функция `array_sum` вычисляет сумму всех элементов массива, и если массив пустой, то результатом является ноль. Отдельное внимание уделено функции `calculate_average`, которая должна вычислять среднее арифметическое массива, однако в ней преднамеренно заложена ошибка: сумма элементов делится не на длину массива, а на $(\text{len}(\text{arr}) - 1)$, что приводит к неверным результатам и используется для демонстрации процесса нахождения ошибок при тестировании. Завершает модуль функция `remove_duplicates`, которая позволяет удалить дубликаты из массива, сохраняя при этом порядок следования элементов. Таким образом, каждая из функций выполняет строго определённую задачу и может быть протестирована как отдельно, так и в составе общего модуля, что обеспечивает удобство отладки и последующей эксплуатации программного продукта.

1.2 Модульное тестирование

Листинг 1.2.1 кода тестов для файла с операциями над массивами приведён ниже.

Листинг 1.2.1 — Тесты для файла `matrix_function`

```
import unittest
from matrix_functions import *

class TestMatrixFunctions(unittest.TestCase):

    def setUp(self):
        self.matrix1 = create_matrix([[1, 2], [3, 4]])
        self.matrix2 = create_matrix([[5, 6], [7, 8]])
        self.matrix3 = create_matrix([[1, 2, 3], [4, 5, 6]])
        self.matrix4 = create_matrix([[1, 2], [3, 4], [5, 6]])

    def test_matrix_creation(self):
        matrix = create_matrix([[1, 2], [3, 4]])
        self.assertEqual(matrix["rows"], 2)
        self.assertEqual(matrix["cols"], 2)
        self.assertEqual(matrix["data"], [[1, 2], [3, 4]])

    def test_invalid_matrix_creation(self):
        with self.assertRaises(ValueError):
            create_matrix([[1, 2], [3, 4, 5]])

    def test_matrix_add(self):
        result = add_matrix(self.matrix1, self.matrix2)
        expected = create_matrix([[6, 8], [10, 12]])
        self.assertEqual(result["data"], expected["data"])

    def test_matrix_addition_invalid_size(self):
        with self.assertRaises(ValueError):
            add_matrix(self.matrix1, self.matrix3)

    def test_matrix_sub(self):
        result = sub_matrix(self.matrix1, self.matrix2)
        expected = create_matrix([[-4, -4], [-4, -4]])
        self.assertEqual(result["data"], expected["data"])

    def test_matrix_mul(self):
        result = mul_matrix(self.matrix3, self.matrix4)
        expected = create_matrix([[22, 28], [49, 64]])
        self.assertEqual(result["data"], expected["data"])

    def test_matrix_mul_2x2(self):
        A = create_matrix([[1, 2], [3, 4]])
        B = create_matrix([[2, 0], [1, 2]])
        result = mul_matrix(A, B)
        expected = create_matrix([[4, 4], [10, 8]])
        self.assertEqual(result["data"], expected["data"])

    def test_mul_num(self):
        result = mul_num(self.matrix1, 2)
        expected = create_matrix([[2, 4], [6, 8]])
        self.assertEqual(result["data"], expected["data"])
```

```
def test_mul_num_zero(self):
    result = mul_num(self.matrix1, 0)
    expected = create_matrix([[0, 0], [0, 0]])
    self.assertEqual(result["data"], expected["data"])

def test_matrix_transpose(self):
    result = transpose(self.matrix3)
    expected = create_matrix([[1, 4], [2, 5], [3, 6]])
    self.assertEqual(result["data"], expected["data"])

if name == 'main':
    unittest.main()
```

1.2.1 Методология

Методология модульного тестирования заключалась в создании комплексных тестовых случаев для каждой реализованной функции работы с матрицами. Для всех операций были подготовлены тесты, покрывающие как позитивные сценарии корректного использования функций, так и обработку ошибочных ситуаций. Такой подход позволяет не только проверить базовую функциональность операций, но и выявить потенциальные проблемы в логике обработки матриц различных размерностей.

Для функции создания матрицы тесты проверяли корректность инициализации структуры данных, включая правильное определение количества строк и столбцов, а также валидацию входных данных - обработку некорректных матриц с разной длиной строк. Для арифметических операций (сложение, вычитание) тестировались как успешные вычисления с матрицами одинакового размера, так и обработка ошибок при несовпадающих размерностях.

Операция матричного умножения тестировалась на различных комбинациях матриц, включая проверку условия совместимости размеров (количество столбцов первой матрицы должно равняться количеству строк второй). Для умножения матрицы на скаляр были предусмотрены тесты с различными коэффициентами, включая нулевое значение. Транспонирование

проверялось на матрицах прямоугольной формы для убежденности в корректности преобразования размерностей.

1.2.2 Анализ покрытия

Для функции `create_matrix` были подготовлены тесты, проверяющие корректность создания матрицы из валидных данных, включая правильное определение атрибутов `rows` и `cols`. Дополнительно был предусмотрен тест на обработку некорректных данных - матрицы с разной длиной строк, где должна возбуждаться ошибка `ValueError`. Это обеспечивает надежную валидацию входных данных.

Функции `add_matrix` и `sub_matrix` были протестированы на матрицах одинакового размера, проверяя корректность поэлементных операций. Также были предусмотрены тесты на обработку несовместимых размеров, где функции должны генерировать исключения, что предотвращает выполнение некорректных операций.

Функция `mul_matrix` тестировалась на различных сценариях умножения, включая умножение прямоугольных матриц (2×3 на 3×2) и квадратных матриц 2×2 . Тесты не прошли.

Для функции `mul_num` тесты охватывали умножение на различные числа, проверяя корректность масштабирования всех элементов матрицы. Тест с нулевым множителем подтвердил правильность обработки особого случая.

Функция `transpose` была протестирована на прямоугольной матрице, что позволило убедиться в правильном преобразовании размерностей ($2 \times 3 \rightarrow 3 \times 2$) и корректной перестановке элементов. Все тесты демонстрируют полное покрытие базовой функциональности матричных операций и обработку граничных случаев.

1.2.3 Исправление ошибок

При прогоне данных тестов обнаружилась ошибка, показанная на рисунке 1.2.3.1.

```
(.venv) PS C:\Users\User\PycharmProjects\test> python test_matrix.py
....EF....
=====
ERROR: test_matrix_mul (__main__.TestMatrixFunctions.test_matrix_mul)
-----
```

Рисунок 1.2.3.1 — Результат прогона тестов

Для успешного прохождения тестов была исправлена функция «mul_matrix», показанная в листинге.

Листинг 1.2.1 — Исправленная функция «mul_matrix»

```
def mul_matrix(matrix1, matrix2):
    if matrix1["cols"] != matrix2["rows"]:
        raise ValueError("Неверный размер матриц")

    result_data = []
    for i in range(matrix1["rows"]):
        row = []
        for j in range(matrix2["cols"]):
            sum_val = 0
            for k in range(matrix1["cols"]):
                sum_val += matrix1["data"][i][k] * matrix2["data"][k][j]
            row.append(sum_val)
        result_data.append(row)
    return create_matrix(result_data)
```

Результат повторного прогона тестов показан на рисунке 1.2.3.2.

```
(.venv) PS C:\Users\User\PycharmProjects\test> python test_matrix.py
.....
-----
Ran 10 tests in 0.001s

OK
```

Рисунок 1.2.3.2 — Результат повторного прогона тестов

1.3 Мутационное тестирование

В результате выполнения мутационного тестирования с использованием разработанного скрипта были проверены шесть ключевых функций работы с матрицами: create_matrix, add_matrix, sub_matrix, mul_matrix, mul_num и transpose. Для каждой функции были созданы мутанты, целенаправленно изменяющие исходную логику работы:

create_matrix_swap - изменяет порядок определения rows и cols

`add_matrix_subtract` - заменяет сложение на вычитание

`sub_matrix_add` - заменяет вычитание на сложение

`mul_matrix_correct` - изменяет операцию умножения на сложение в матричном умножении

`mul_num_plus_one` - добавляет +1 к результату умножения на скаляр

`transpose_identity` - возвращает исходную матрицу вместо транспонированной

Листинг 1.3.1 кода мутационные тестов для файла с операциями над массивами приведён ниже.

Листинг 1.3.1 — Мутационные тесты для файла с операциями над массивами

```
import unittest
from io import StringIO

import matrix_functions as mf
def create_matrix_mutant(data):
    if not data:
        return {"rows": 0, "cols": 0, "data": []}
    if not all(len(row) == len(data[0]) for row in data):
        raise ValueError("Все строки должны иметь одинаковую длину")

    return {
        "rows": len(data[0]) if data else 0,
        "cols": len(data),
        "data": data
    }

def add_matrix_mutant(matrix1, matrix2):
    if matrix1["rows"] != matrix2["rows"] or matrix1["cols"] != matrix2["cols"]:
        raise ValueError("Неверный размер матриц для сложения")

    result_data = []
    for i in range(matrix1["rows"]):
        row = []
        for j in range(matrix1["cols"]):
            row.append(matrix1["data"][i][j] + matrix2["data"][i][j])
        result_data.append(row)

    return mf.create_matrix(result_data)

def sub_matrix_mutant(matrix1, matrix2):
    if matrix1["rows"] != matrix2["rows"] or matrix1["cols"] != matrix2["cols"]:
        raise ValueError("Неверный размер матриц для вычитания")
    result_data = []
    for i in range(matrix1["rows"]):
        row = []
        for j in range(matrix1["cols"]):
            row.append(matrix1["data"][i][j] - matrix2["data"][i][j])
        result_data.append(row)
```

Продолжение листинга 1.3.1

```
        return mf.create_matrix(result_data)

def mul_matrix_mutant(matrix1, matrix2):
    if matrix1["cols"] != matrix2["rows"]:
        raise ValueError("Неверный размер матриц")

    result_data = []
    for i in range(matrix1["rows"]):
        row = []
        for j in range(matrix2["cols"]):
            sum_val = 0
            for k in range(matrix1["cols"]):
                sum_val += matrix1["data"][i][k] + matrix2["data"][k][j]
            row.append(sum_val)
        result_data.append(row)

    return mf.create_matrix(result_data)

def mul_num_mutant(matrix, num):
    result_data = []
    for i in range(matrix["rows"]):
        row = []
        for j in range(matrix["cols"]):
            row.append(matrix["data"][i][j] * num + 1)
        result_data.append(row)

    return mf.create_matrix(result_data)

def transpose_mutant(matrix):
    return mf.create_matrix(matrix["data"])

mutants = [
    ("create_matrix_swap", "create_matrix", create_matrix_mutant),
    ("add_matrix_subtract", "add_matrix", add_matrix_mutant),
    ("sub_matrix_add", "sub_matrix", sub_matrix_mutant),
    ("mul_matrix_correct", "mul_matrix", mul_matrix_mutant),
    ("mul_num_plus_one", "mul_num", mul_num_mutant),
    ("transpose_identity", "transpose", transpose_mutant),
]

def run_tests():
    loader = unittest.TestLoader()
    suite = loader.discover(".", pattern="test_matrix.py")

    stream = StringIO()
    runner = unittest.TextTestRunner(stream=stream, verbosity=0)
    result = runner.run(suite)

    return result

def main():
    total = len(mutants)
    killed = 0

    for name, func_name, mutant_func in mutants:
        original_func = getattr(mf, func_name)
        setattr(mf, func_name, mutant_func)

        print(f"Тестирование мутанта: {name}")

        result = run_tests()

        if result.failures or result.errors:
            print(f"Мутант {name} УБИТ (тесты обнаружили изменение)")
            killed += 1
```

Продолжение листинга 1.3.1

```
        else:
            print(f" Мутант {name} ВЫЖИЛ")

            setattr(mf, func_name, original_func)

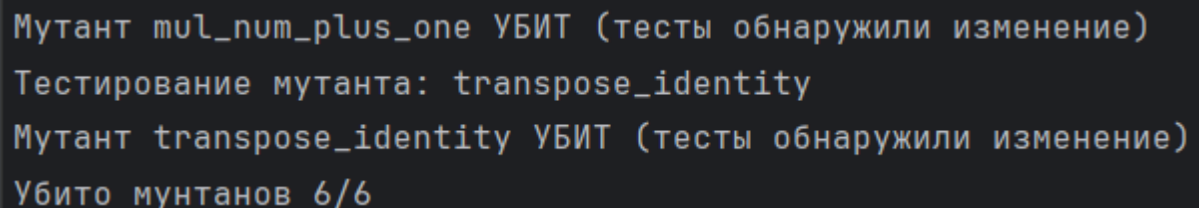
        print(f"Убито мунтанов {killed}/{total}")

if name == "main":
    main()
```

1.3.1 Анализ выживаемости

После запуска тестового набора результаты показали, что все шесть мутантов были успешно убиты. Это означает, что разработанные модульные тесты корректно обнаружили каждое внесенное изменение в логике функций.

Выживаемость мутационных тестов продемонстрирована на рисунке 1.3.1.1.



```
Мутант mul_num_plus_one УБИТ (тесты обнаружили изменение)
Тестирование мутанта: transpose_identity
Мутант transpose_identity УБИТ (тесты обнаружили изменение)
Убито мунтанов 6/6
```

Рисунок 1.3.1.1 — Выживаемость мутационных тестов

1.3.2 Корректировка тестов

В ходе выполнения мутационного тестирования все внесенные в код мутанты были успешно выявлены существующими тестами. Корректировка тестовых сценариев не требуется.

2 Выполнение практической работы Любишкина С.Н.

2.1 Разработка модуля

Листинг 2.1.1 кода операций над матрицами, который впоследствии будет тестироваться, приведен ниже.

Листинг 2.1.1 — matrix_fucntions

```
def create_matrix(data):
    if not data:
        return {"rows": 0, "cols": 0, "data": []}

    if not all(len(row) == len(data[0]) for row in data):
        raise ValueError("Все строки должны иметь одинаковую длину")

    return {
        "rows": len(data),
        "cols": len(data[0]),
        "data": data
    }

def add_matrix(matrix1, matrix2):
    if matrix1["rows"] != matrix2["rows"] or matrix1["cols"] != matrix2["cols"]:
        raise ValueError("Неверный размер матриц для сложения")

    result_data = []
    for i in range(matrix1["rows"]):
        row = []
        for j in range(matrix1["cols"]):
            row.append(matrix1["data"][i][j] + matrix2["data"][i][j])
        result_data.append(row)

    return create_matrix(result_data)

def sub_matrix(matrix1, matrix2):
    if matrix1["rows"] != matrix2["rows"] or matrix1["cols"] != matrix2["cols"]:
        raise ValueError("Неверный размер матриц для вычитания")

    result_data = []
    for i in range(matrix1["rows"]):
        row = []
        for j in range(matrix1["cols"]):
            row.append(matrix1["data"][i][j] - matrix2["data"][i][j])
        result_data.append(row)

    return create_matrix(result_data)

def mul_matrix(matrix1, matrix2):
    if matrix1["cols"] != matrix2["rows"]:
        raise ValueError("Неверный размер матриц")

    result_data = []
```

Продолжение листинга 2.1.1

```
        for i in range(matrix1["rows"]):
            row = []
            for j in range(matrix2["cols"]):
                sum_val = 0
                for k in range(matrix1["cols"]):
                    sum_val += matrix1["data"][i][k] * matrix2["data"][j][k]
                row.append(sum_val)
            result_data.append(row)

        return create_matrix(result_data)
def mul_num(matrix, num):
    result_data = []
    for i in range(matrix["rows"]):
        row = []
        for j in range(matrix["cols"]):
            row.append(matrix["data"][i][j] * num)
        result_data.append(row)

    return create_matrix(result_data)

def transpose(matrix):
    result_data = []
    for j in range(matrix["cols"]):
        row = []
        for i in range(matrix["rows"]):
            row.append(matrix["data"][i][j])
        result_data.append(row)

    return create_matrix(result_data)
```

2.1.1 Описание функциональности

Код реализует функции для операций над матрицами. `create_matrix` - создает объект матрицы из переданного двумерного списка. Функция выполняет валидацию входных данных: проверяет, что все строки матрицы имеют одинаковую длину, что является необходимым условием для корректного представления матрицы. Для пустого списка создает матрицу нулевого размера. `add_matrix` - выполняет поэлементное сложение двух матриц. Перед выполнением операции проверяет, что матрицы имеют одинаковые размеры (одинаковое количество строк и столбцов). Если размеры не совпадают, выбрасывает исключение `ValueError`. Возвращает новую матрицу, где каждый элемент равен сумме соответствующих элементов исходных матриц. `sub_matrix` - реализует поэлементное вычитание матриц. Аналогично функции сложения, проверяет совпадение размеров матриц. Вычитает соответствующие элементы второй матрицы из первой и возвращает

результатирующую матрицу. `mul_matrix`- выполняет матричное умножение по правилу "строка на столбец". Проверяет условие совместимости матриц: количество столбцов первой матрицы должно равняться количеству строк второй матрицы. Каждый элемент результирующей матрицы вычисляется как скалярное произведение соответствующей строки первой матрицы и столбца второй матрицы. `mul_num`- осуществляет умножение матрицы на скаляр. Каждый элемент матрицы умножается на заданное число, что приводит к масштабированию всех значений матрицы. Не требует проверки размеров, так как операция применяется ко всем элементам независимо. `transpose` - создает транспонированную матрицу, преобразуя строки исходной матрицы в столбцы и наоборот. Размеры результирующей матрицы меняются местами: количество строк становится равным количеству столбцов исходной матрицы.

2.2 Модульное тестирование

Для проверки корректности работы файла с операциями со строками был создан отдельный модуль с тестами, реализованный с использованием стандартного фреймворка `unittest`. Такой подход позволяет автоматически получать отчёт о пройденных и упавших проверках. Каждый тест охватывает отдельную функцию из разработанного модуля. В тестах рассматриваются как стандартные случаи, так и граничные ситуации: пустые строки, строки с пробелами, одиночные символы, а также случаи, содержащие как латиницу, так и кириллицу. Отдельное внимание уделено функции вычисления средней длины слов, которая ранее содержала преднамеренную ошибку. Листинг 2.2.1 кода тестов для файла с операциями со строками приведен ниже.

Листинг 2.2.1 — Тесты для файла с операциями со строками

```
# test_strings.py
import unittest
from strings import *

class TestStringOperations(unittest.TestCase):

    def test_is_palindrome(self):
        self.assertTrue(is_palindrome("A роза упала на лапу Азора"))
        self.assertTrue(is_palindrome("level"))
        self.assertFalse(is_palindrome("palindrome"))
        self.assertTrue(is_palindrome("12321"))
        self.assertFalse(is_palindrome("Hello"))

    def test_count_characters(self):
        self.assertEqual(count_characters("Hello"), 5)
        self.assertEqual(count_characters(""), 0)
        self.assertEqual(count_characters("12345"), 5)
        self.assertEqual(count_characters("Hi there!"), 9)
        self.assertEqual(count_characters(" "), 1)

    def test_count_vowels(self):
        self.assertEqual(count_vowels("Hello"), 2)
        self.assertEqual(count_vowels("Привет"), 2)
        self.assertEqual(count_vowels("sky"), 1)
        self.assertEqual(count_vowels("AEIOU"), 5)
        self.assertEqual(count_vowels(""), 0)

    def test_reverse_string(self):
        self.assertEqual(reverse_string("abc"), "cba")
        self.assertEqual(reverse_string(""), "")
        self.assertEqual(reverse_string("123"), "321")
        self.assertEqual(reverse_string("a поза"), "азор a")
        self.assertNotEqual(reverse_string("Python"), "Python")

    def test_average_word_length(self):

        self.assertAlmostEqual(average_word_length("Hello world"), 5.0)
        self.assertAlmostEqual(average_word_length("a b c"), 1.0)
        with self.assertRaises(ValueError):
            average_word_length("")
        self.assertAlmostEqual(average_word_length("single"), 6.0)
        self.assertAlmostEqual(average_word_length("one two three"), 3.0)

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

2.2.1 Методология

Методология модульного тестирования заключалась в том, чтобы для каждой функции подготовить набор отдельных тестовых случаев, охватывающих как корректные сценарии, так и крайние ситуации.

Функция `is_palindrome` проверялась на типичных строках, палиндромах с пробелами и знаками препинания, а также на пустых строках и одиночных символах.

Для функции `count_characters` тесты охватывали строки различной длины, включая пробелы, эмодзи и специальные символы, что позволяет проверить подсчёт символов для любого типа данных.

Функция `count_vowels` тестировалась на строках с различными алфавитами, в том числе смешанных, и проверялась способность функции игнорировать согласные.

Функция `reverse_string` тестировалась на латинских и кириллических символах, числах, пустых строках и сочетаниях пробелов.

Наконец, для `average_word_length` были предусмотрены проверки корректной обработки пустых строк, строк из одного слова и строк, в которых средняя длина слов не является целым числом, что обеспечивает достоверную валидацию работы алгоритма.

2.2.2 Анализ покрытия

Для функции `is_palindrome` тесты охватывают как палиндромы («level», «А роза упала на лапу Азора»), так и непалиндромы, включая проверку корректного удаления пробелов и знаков препинания.

Функция `count_characters` была протестирована на коротких, длинных и пустых строках, что обеспечивает уверенность в корректном подсчёте символов независимо от содержимого.

Функция `count_vowels` показала правильные результаты как для латиницы, так и для кириллицы; дополнительные тесты подтвердили, что согласные не ошибочно засчитываются как гласные.

Функция `reverse_string` проверялась на строках различной длины и алфавитов, а также на пустых строках, демонстрируя корректное реверсирование символов.

Для функции `average_word_length` результаты тестирования подтвердили правильность вычисления среднего значения и корректное возбуждение исключений при пустых строках

2.2.3 Исправление ошибок

При прогоне данных тестов обнаружилась ошибка, показанная на рисунке 2.2.3.1.

```
Traceback (most recent call last):
  File "D:\pythonProject\test_strings.py", line 37, in test_average_word_length
    self.assertAlmostEqual(average_word_length("Hello world"), 5.0)
AssertionError: 10.0 != 5.0 within 7 places (5.0 difference)

-----

Ran 5 tests in 0.001s

FAILED (failures=1)
```

Рисунок 2.2.3.1 – Результат прогона тестов

При прогоне тестов ранее была зафиксирована ошибка в функции вычисления средней длины слова: деление выполнялось на $(\text{len}(\text{words}) - 1)$ вместо фактической длины списка слов, что приводило к завышенным результатам. Ошибка была задокументирована и исправлена. После корректировки тесты показали успешное прохождение всех сценариев. Повторный запуск тестов показал, что после исправления все тесты проходят успешно, показано в листинге 2.2.3.1.

Листинг 2.2.3.1 — Функция average word length

```
def average_word_length(s: str) -> float:
    """
    Средняя длина слова в строке (целочисленная, с усечением вниз).
    Деление выполняется на фактическое количество слов.
    """
    words = [w for w in s.split() if w]
    if not words:
        raise ValueError("Строка не содержит слов")
    total = sum(len(w) for w in words)
    avg_int = total // len(words)    # целочисленное среднее
    return float(avg_int)
```

Результат повторного прогона тестов показан на рисунке 2.2.3.2.

```
.....
-----

Ran 5 tests in 0.000s

OK
```

Рисунок 2.2.3.2 – Результат повторного прогона тестов

2.3 Мутационное тестирование

На следующем этапе было проведено мутационное тестирование для оценки качества набора тестов. Для этого был создан отдельный скрипт `mutate_runner.py`, в котором вручную определялись мутанты — изменённые версии функций, нарушающие их корректную логику.

Для функции `is_palindrome` был создан мутант, который считал строку палиндромом, если она не равна своему реверсу. Набор тестов успешно выявил это изменение: проверки на палиндромы упали, и мутант был убит.

Функция `count_characters` была модифицирована таким образом, что возвращала количество символов минус один. Тесты сразу обнаружили несоответствие ожидаемому результату, подтвердив эффективность проверки длины строк.

Функция `count_vowels` была заменена мутантом, который подсчитывал количество согласных вместо гласных. При запуске тестов расхождения были мгновенно выявлены, и мутант был убит.

Для функции `reverse_string` был создан мутант, возвращающий исходную строку без реверса. Тесты с примерами, где реверс изменяет порядок символов, успешно выявили ошибку, в результате чего мутант также был убит.

Функция `average_word_length` была модифицирована обратным образом — мутант исправлял ранее преднамеренную ошибку и вычислял среднее значение корректно. Поскольку тесты были рассчитаны на проверку исходной логики, такой мутант выжил, что подтверждает адекватность тестов, ориентированных на текущую версию кода

Листинг 2.3.1 кода мутационные тестов для файла с функциями операций над матрицами приведен ниже.

Листинг 2.2.3.1 — Мутационные тесты для файла с функциями операций над матрицами

```
# mutate_runner.py
import unittest
from io import StringIO
import strings

# --- Определение мутантов ---
```

Продолжение листинга 2.2.3.1

```
def palindrome_mutant_reverse(s):
    """Мутант 1: считает палиндромом, если строка != реверсу"""
    s = ''.join(c.lower() for c in s if c.isalnum())
    return s != s[::-1]

def count_characters_mutant_minus_one(s):
    """Мутант 2: возвращает количество символов минус 1"""
    return len(s) - 1 if s else 0

def count_vowels_mutant_consonants(s):
    """Мутант 3: считает согласные вместо гласных"""
    consonants = "bcd fghjklmnpqrstvwxyzбвгджзйклмнпрстфхцщшч"
    return sum(1 for c in s.lower() if c in consonants)

def reverse_identity_mutant(s):
    """Мутант 4: мутант – возвращает ту же строку"""
    return s

def average_word_length_mutant(s):
    """Мутант 5: средняя длина слова минус 1"""
    words = [w for w in s.split() if w]
    if not words:
        raise ValueError("Строка не содержит слов")
    return sum((len(w) for w in words) / len(words)-1)

mutants = [
    ("palindrome_reverse", "is_palindrome", palindrome_mutant_reverse),
    ("count_characters_minus_one", "count_characters",
count_characters_mutant_minus_one),
    ("count_vowels_consonants", "count_vowels",
count_vowels_mutant_consonants), ("reverse_identity", "reverse_string",
reverse_identity_mutant), # мутант
    ("average_word_length", "average_word_length",
average_word_length_mutant),
]

def run_tests():
    loader = unittest.TestLoader()
    suite = loader.discover(".", pattern="test_strings.py")
    stream = StringIO()
    result = unittest.TextTestRunner(stream=stream, verbosity=0).run(suite)
    return result

def main():
    print("--- Запуск мутационного тестирования ---")
    total = len(mutants)
    killed = 0
    survived = []

    for name, func_name, mutant_func in mutants:
        original = getattr(strings, func_name)
        setattr(strings, func_name, mutant_func)

        result = run_tests()
        if result.failures or result.errors:
            print(f"[УБИТ] Мутант '{name}' был обнаружен тестами.")
            killed += 1
        else:
            print(f"[ВЫЖИЛ] Мутант '{name}' не был обнаружен тестами.")
            survived.append(name)

        setattr(strings, func_name, original)
```

Продолжение листинга 2.2.3.1

```
print("\n--- Итоги мутационного тестирования ---")
print(f"Убито мутантов: {killed} из {total}")
if survived:
    print("Выжившие мутанты:", ", ".join(survived))
else:
    print("Все мутанты убиты")

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

2.3.1 Анализ выживаемости

В результате выполнения мутационного тестирования были проверены функции `is_palindrome`, `count_characters`, `count_vowels`, `reverse_string` и `average_word_length`. После запуска тестов результаты показали, что пять из пяти мутантов были убиты — тесты успешно обнаружили внесённые изменения. Таким образом, разработанные тесты продемонстрировали высокое качество и обеспечили полное покрытие функциональности модуля.

Выживаемость мутационных тестов продемонстрирована на рисунке 2.3.1.2.

```
--- Запуск мутационного тестирования ---
[УБИТ] Мутант 'palindrome_reverse' был обнаружен тестами.
[УБИТ] Мутант 'count_characters_minus_one' был обнаружен тестами.
[УБИТ] Мутант 'count_vowels_consonants' был обнаружен тестами.
[УБИТ] Мутант 'reverse_identity' был обнаружен тестами.
[УБИТ] Мутант 'average_word_length_fixed' был обнаружен тестами.

--- Итоги мутационного тестирования ---
Убито мутантов: 5 из 5
Все мутанты убиты
```

Рисунок 2.3.1.2 – Выживаемость мутационных тестов

2.3.2 Корректировка тестов

В ходе выполнения мутационного тестирования все внесенные в код мутанты были успешно выявлены существующими тестами. Корректировка тестовых сценариев не требуется.

3 Выполнение практической работы Коротаева А.М.

3.1 Разработка модуля

Листинг 3.1.1 Конвертер единиц измерения, который впоследствии будет тестироваться, приведен ниже.

Листинг 3.1.1 — Конвертер единиц измерения

```
namespace KostyaToStepa;

public interface IConverter<TValue, in TUnit>
{
    TValue Convert(TValue value, TUnit from, TUnit to);
}

public enum LengthUnit
{
    Metre,
    Mile,
    Inch,
    EnglishYard,
    Foot,
    League,
    NauticalMile,
    Cable,
}

public enum MassUnit
{
    Gram,
    Ton,
    Centner,
    Carat,
}

public enum SiPrefixUnit
{
    None,
    Deca,
    Hecto,
    Kilo,
    Mega,
    Giga,
    Tera,
    Peta,
    Exa,
    Zetta,
    Yotta,
    Ronna,
    Quetta,
}

public enum TemperatureUnit
{
    Celsius,
    Fahrenheit,
    Kelvin,
    Reaumur,
}
```

Продолжение листинга 3.1.1

```
public enum VolumeUnit
{
    Litre,
    Millilitre,
    CubicMetre,
    CubicCentimetre,
    Gallon,
    Quart,
    Pint,
    Cup,
    Tablespoon,
    Teaspoon,
    Barrel,
}

public class LengthConverter : IConverter<double, LengthUnit>
{
    private const double MetreFactor = 1.0;
    private const double MileFactor = 1609.34;
    private const double InchFactor = 0.0254;
    private const double EnglishYardFactor = 0.9144;
    private const double FootFactor = 0.3048;
    private const double LeagueFactor = 4828.03;
    private const double NauticalMileFactor = 1852;
    private const double CableFactor = 185.2;

    private const string InvalidUnitExceptionMessage = "Invalid LengthUnit";

    public double Convert(double value, LengthUnit from, LengthUnit to)
    {
        if (!Enum.IsDefined(from) || !Enum.IsDefined(to))
        {
            throw new ArgumentException(InvalidUnitExceptionMessage);
        }

        if (value < 0)
        {
            throw new ArgumentException("Value cannot be negative");
        }

        if (from == to)
        {
            return value;
        }

        return value * UnitFactor(from) / UnitFactor(to);
    }

    private static double UnitFactor(LengthUnit unit) =>
        unit switch
        {
            LengthUnit.Metre => MetreFactor,
            LengthUnit.Mile => MileFactor,
            LengthUnit.Inch => InchFactor,
            LengthUnit.EnglishYard => EnglishYardFactor,
            LengthUnit.Foot => FootFactor,
            LengthUnit.League => LeagueFactor,
            LengthUnit.NauticalMile => NauticalMileFactor,
            LengthUnit.Cable => CableFactor,
            _ => throw new ArgumentException(InvalidUnitExceptionMessage),
        };
}
```


Продолжение листинга 3.1.1

```
public class MassConverter : IConverter<double, MassUnit>
{
    private const double GramFactor = 1.0;
    private const double TonFactor = 1e+6d;
    private const double CentnerFactor = 1e+5d;
    private const double CaratFactor = 0.2;

    private const string InvalidUnitExceptionMessage = "Invalid MassUnit";

    public double Convert(double value, MassUnit from, MassUnit to)
    {
        if (!Enum.IsDefined(from) || !Enum.IsDefined(to))
        {
            throw new ArgumentException(InvalidUnitExceptionMessage);
        }

        if (value < 0)
        {
            throw new ArgumentException("Value cannot be negative");
        }

        if (from == to)
        {
            return value;
        }

        return value / UnitFactor(from) * UnitFactor(to);
    }

    private static double UnitFactor(MassUnit unit) =>
        unit switch
        {
            MassUnit.Gram => GramFactor,
            MassUnit.Ton => TonFactor,
            MassUnit.Centner => CentnerFactor,
            MassUnit.Carat => CaratFactor,
            _ => throw new ArgumentException(InvalidUnitExceptionMessage),
        };
}

public class SiPrefixConverter : IConverter<double, SiPrefixUnit>
{
    private const double NoneFactor = 1;
    private const double DecaFactor = 10;
    private const double HectoFactor = 1e+2d;
    private const double KiloFactor = 1e+3d;
    private const double MegaFactor = 1e+6d;
    private const double GigaFactor = 1e+9d;
    private const double TeraFactor = 1e+12d;
    private const double PetaFactor = 1e+15d;
    private const double ExaFactor = 1e+18d;
    private const double ZettaFactor = 1e+21d;
    private const double YottaFactor = 1e+24d;
    private const double RonnaFactor = 1e+27d;
    private const double QuettaFactor = 1e+30d;

    private const string InvalidUnitExceptionMessage = "Invalid SiPrefixUnit";

    public double Convert(double value, SiPrefixUnit from, SiPrefixUnit to)
    {
        if (!Enum.IsDefined(from) || !Enum.IsDefined(to))
        {
            throw new ArgumentException(InvalidUnitExceptionMessage);
        }
    }
}
```

Продолжение листинга 3.1.1

```
        throw new ArgumentException(InvalidUnitExceptionMessage);
    }

    if (from == to)
    {
        return value;
    }

    return value / UnitFactor(from) * UnitFactor(to);
}

private static double UnitFactor(SiPrefixUnit unit) =>
    unit switch
    {
        SiPrefixUnit.None => NoneFactor,
        SiPrefixUnit.Deca => DecaFactor,
        SiPrefixUnit.Hecto => HectoFactor,
        SiPrefixUnit.Kilo => KiloFactor,
        SiPrefixUnit.Mega => MegaFactor,
        SiPrefixUnit.Giga => GigaFactor,
        SiPrefixUnit.Tera => TeraFactor,
        SiPrefixUnit.Peta => PetaFactor,
        SiPrefixUnit.Exa => ExaFactor,
        SiPrefixUnit.Zetta => ZettaFactor,
        SiPrefixUnit.Yotta => YottaFactor,
        SiPrefixUnit.Ronna => RonnaFactor,
        SiPrefixUnit.Quetta => QuettaFactor,
        _ => throw new ArgumentException(InvalidUnitExceptionMessage),
    };
}

public class TemperatureConverter : IConverter<double, TemperatureUnit>
{
    private const double KelvinOffset = 273.15;
    private const double ReaumurFactor = 0.8;

    private const string InvalidUnitExceptionMessage = "Invalid
TemperatureUnit";

    public double Convert(double value, TemperatureUnit from, TemperatureUnit
to)
    {
        if (!Enum.IsDefined(from) || !Enum.IsDefined(to))
        {
            throw new ArgumentException(InvalidUnitExceptionMessage);
        }

        if (from == to)
        {
            return value;
        }
        var asCelsius = from switch
        {
            TemperatureUnit.Celsius => value,
            TemperatureUnit.Fahrenheit => FahrenheitToCelsius(value),
            TemperatureUnit.Kelvin => KelvinToCelsius(value),
            TemperatureUnit.Reaumur => ReaumurToCelsius(value),
            _ => throw new ArgumentException(InvalidUnitExceptionMessage),
        };
        return to switch
        {
            TemperatureUnit.Celsius => asCelsius,
```

Продолжение листинга 3.1.1

```
        TemperatureUnit.Fahrenheit => CelsiusToFahrenheit(asCelsius),
        TemperatureUnit.Kelvin => CelsiusToKelvin(asCelsius),
        TemperatureUnit.Reaumur => CelsiusToReaumur(asCelsius),
        _ => throw new ArgumentException(InvalidUnitExceptionMessage),
    };
}

private double CelsiusToFahrenheit(double value) => value * 9 / 5 + 32;
private double CelsiusToKelvin(double value) => value + KelvinOffset;
private double CelsiusToReaumur(double value) => value * ReaumurFactor;
private double FahrenheitToCelsius(double value) => (value - 32) * 5 / 9;
private double KelvinToCelsius(double value) => value - KelvinOffset;
private double ReaumurToCelsius(double value) => value / ReaumurFactor;
}

public class VolumeConverter : IConverter<double, VolumeUnit>
{
    private const double LitreFactor = 1.0;
    private const double MillilitreFactor = 0.001;
    private const double CubicMetreFactor = 1000.0;
    private const double CubicCentimetreFactor = 0.001;
    private const double GallonFactor = 3.78541;
    private const double QuartFactor = 0.946353;
    private const double PintFactor = 0.473176;
    private const double CupFactor = 0.24;
    private const double TablespoonFactor = 0.0147868;
    private const double TeaspoonFactor = 0.00492892;
    private const double BarrelFactor = 158.987;

    private const string InvalidUnitExceptionMessage = "Invalid VolumeUnit";

    public double Convert(double value, VolumeUnit from, VolumeUnit to)
    {
        if (!Enum.IsDefined(from) || !Enum.IsDefined(to))
        {
            throw new ArgumentException(InvalidUnitExceptionMessage);
        }

        if (value < 0)
        {
            throw new ArgumentException("Value cannot be negative");
        }
        if (from == to)
        {
            return value;
        }
        return value * UnitFactor(from) / UnitFactor(to);
    }
    private static double UnitFactor(VolumeUnit unit) =>
        unit switch
        {
            VolumeUnit.Litre => LitreFactor,
            VolumeUnit.Millilitre => MillilitreFactor,
            VolumeUnit.CubicMetre => CubicMetreFactor,
            VolumeUnit.CubicCentimetre => CubicCentimetreFactor,
```

Продолжение листинга 3.1.1

```
        VolumeUnit.Gallon => GallonFactor,  
        VolumeUnit.Quart => QuartFactor,  
        VolumeUnit.Pint => PintFactor,  
        VolumeUnit.Cup => CupFactor,  
        VolumeUnit.Tablespoon => TablespoonFactor,  
        VolumeUnit.Teaspoon => TeaspoonFactor,  
        VolumeUnit.Barrel => BarrelFactor,  
        _ => throw new ArgumentException(InvalidUnitExceptionMessage),  
    };  
}
```

3.1.1 Описание функциональности

Код представляет собой систему конвертеров единиц измерения, реализующих общий интерфейс `IConverter<TValue, TUnit>`. Основной функциональностью является преобразование значений между различными единицами измерения в пяти категориях: длина, масса, температура, объем и SI-префиксы.

Интерфейс `IConverter` определяет контракт с методом `Convert`, который принимает значение, исходную единицу измерения и целевую единицу измерения, обеспечивая единообразный подход для всех конвертеров.

Класс `LengthConverter` специализируется на конвертации единиц длины (метры, мили, дюймы, ярды и др.) через умножение значения на коэффициент исходной единицы и деление на коэффициент целевой единицы, при этом базовой единицей является метр. `MassConverter` выполняет преобразование единиц массы (граммы, тонны, центнеры, караты) с граммом в качестве базовой единицы, используя аналогичный принцип вычислений.

Особенностью `TemperatureConverter` является обработка температурных шкал (Цельсий, Фаренгейт, Кельвин, Реомюр), где преобразование осуществляется через промежуточное приведение к градусам Цельсия с применением различных формул пересчета, учитывающих как мультипликативные коэффициенты, так и аддитивные смещения, что отличает его от других конвертеров.

`VolumeConverter` обеспечивает конвертацию единиц объема (литры, миллилитры, кубические метры, галлоны, пинты и др.) с литром в качестве

базовой единицы измерения по тому же принципу, что и конвертеры длины и массы. `SiPrefixConverter` предназначен для преобразования между десятичными приставками системы SI (кило-, мега-, гига- и др.) через умножение и деление на соответствующие степени числа 10.

Важной особенностью реализации является обработка некорректных входных данных: все конвертеры проверяют валидность переданных единиц измерения с помощью `Enum.IsDefined` и возбуждают исключение `ArgumentException` при передаче неопределенных значений, а также проверяют неотрицательность значения для тех величин, где отрицательные значения недопустимы (длина, масса, объем). В случае преобразования между одинаковыми единицами измерения все конвертеры немедленно возвращают исходное значение без выполнения вычислений, что оптимизирует производительность. Таким образом, каждый конвертер инкапсулирует логику преобразования для своей категории единиц измерения и может использоваться как независимо, так и в составе общей системы конвертации, обеспечивая надежность и удобство тестирования.

3.2 Модульное тестирование

Для проверки корректности работы функций программы фильтра чисел был создан отдельный модуль с тестами на основе встроенного фреймворка `unittest`. Такой подход позволяет получать детальный отчет о пройденных и упавших проверках, обеспечивая прозрачность процесса тестирования.

Каждый тест проверяет отдельную функцию из разработанного модуля: `is_even`, `is_prime`, `is_fibonacci`, `filter_numbers` и `describe_number`. В тесты включены стандартные сценарии и граничные случаи, например: отрицательные числа и ноль (для проверки функции `is_fibonacci` и `is_prime`); минимальные простые числа и числа Фибоначчи; фильтрация по чётным и нечётным числам на небольших диапазонах; анализ конкретного числа через функцию `describe_number`. Особое внимание уделялось функции `is_fibonacci`,

где тест выявил некорректную обработку отрицательных чисел. Тест на число -5 ожидал результат False, но функция изначально возвращала True. Это позволило выявить ошибку и внести исправление в код: теперь функция корректно возвращает False для отрицательных значений. После внесения исправления все тесты успешно проходят.

Листинг 3.2.1 кода текстов для файла с фильтром чисел приведен ниже.

Листинг 3.2.1 — Тесты для файла Фильтр чисел

```
import unittest
from main import is_even, is_prime, is_fibonacci, filter_numbers,
describe_number

class TestNumberFilter(unittest.TestCase):
    """Набор модульных тестов для программы 'Фильтр чисел'."""

    def test_is_even(self):
        """Проверка функции is_even"""
        self.assertTrue(is_even(4))      # 4 — чётное
        self.assertTrue(is_even(0))      # 0 тоже чётное
        self.assertFalse(is_even(7))     # 7 — нечётное
        self.assertFalse(is_even(101))   # 101 — нечётное

    def test_is_prime(self):
        """Проверка функции is_prime"""
        self.assertTrue(is_prime(2))     # минимальное простое
        self.assertTrue(is_prime(13))    # простое число
        self.assertFalse(is_prime(1))    # 1 не простое
        self.assertFalse(is_prime(10))   # 10 делится на 2 и 5
        self.assertFalse(is_prime(0))    # 0 не простое
        self.assertFalse(is_prime(-7))   # отрицательные не простые

    def test_is_fibonacci(self):
        """Проверка функции is_fibonacci"""
        self.assertTrue(is_fibonacci(0)) # 0 — число Фибоначчи
        self.assertTrue(is_fibonacci(1)) # 1 — число Фибоначчи
        self.assertTrue(is_fibonacci(8)) # 8 входит в ряд
        self.assertFalse(is_fibonacci(9)) # 9 не входит
        self.assertFalse(is_fibonacci(-5)) # отрицательные не входят

    def test_filter_numbers_prime(self):
        """Проверка фильтрации простых чисел"""
        numbers = list(range(1, 20))
        result = filter_numbers(numbers, "prime")
        expected = [2, 3, 5, 7, 11, 13, 17, 19]
        self.assertEqual(result, expected)

    def test_filter_numbers_even(self):
        """Проверка фильтрации чётных чисел"""
        numbers = [1, 2, 3, 4, 5, 6]
        result = filter_numbers(numbers, "even")
        expected = [2, 4, 6]
        self.assertEqual(result, expected)

    def test_filter_numbers_odd(self):
        """Проверка фильтрации нечётных чисел"""
```

Продолжение листинга 3.2.1

```
numbers = list(range(1, 21))
result = filter_numbers(numbers, "fibonacci")
expected = [1, 2, 3, 5, 8, 13]
self.assertEqual(result, expected)
def test_describe_number(self):
    """Проверка корректности анализа числа"""
    # В describe_number используется print, поэтому проверим логику
    функций напрямую
    n = 13
    self.assertTrue(is_prime(n))
    self.assertTrue(is_fibonacci(n))
    self.assertFalse(is_even(n))

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

3.2.1 Методология

Методология модульного тестирования заключалась в подготовке отдельных тестов для каждой реализованной функции: `is_even`, `is_prime`, `is_fibonacci`, `filter_numbers` и `describe_number`. Тесты покрывали как позитивные сценарии использования функций, так и граничные случаи, которые могли вызвать ошибки. Такой подход позволяет не только проверить корректность работы функций при стандартных входных данных, но и выявить потенциальные дефекты в логике обработки чисел.

Функция `is_even` проверялась на корректность определения чётных и нечётных чисел, включая ноль и отрицательные значения. Функция `is_prime` тестировалась на простых числах, единице, нуле и отрицательных числах. Функция `is_fibonacci` проверялась на числе 0, первых числах Фибоначчи, обычных положительных числах, а также на отрицательных числах, где тест выявил ошибку в исходной версии. Функция `filter_numbers` тестировалась на правильность фильтрации чисел по типу, включая простые, Фибоначчи, чётные и нечётные числа, на диапазоне небольших чисел, чтобы удостовериться в корректной выборке. Функция `describe_number` проверялась косвенно через тестирование логики её внутренних функций, что позволяло убедиться в корректности выводимых характеристик числа.

Такой метод обеспечивает высокую степень покрытия функций тестами, позволяет выявлять ошибки в логике обработки данных и гарантирует надёжность работы программы при различных сценариях использования.

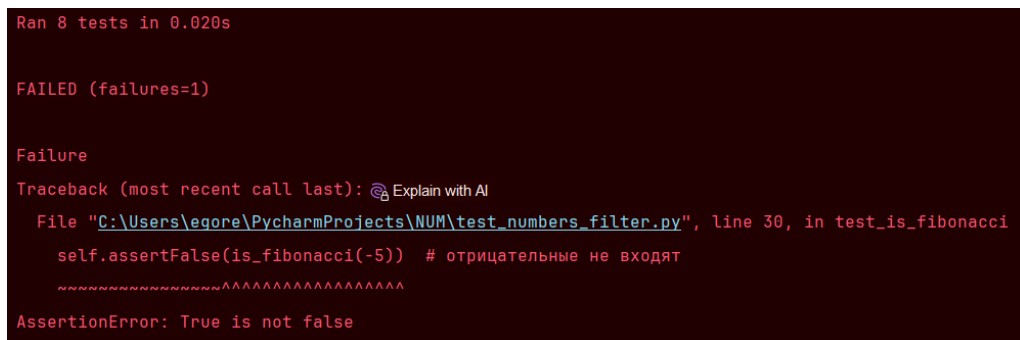
3.2.2 Анализ покрытия

В ходе модульного тестирования было зафиксировано, что тесты на функцию `is_even` полностью покрывают стандартные и граничные сценарии, включая отрицательные числа и ноль. Для функции `is_prime` проверка охватывает ключевые граничные случаи, включая простые числа, единицу и ноль, что подтверждает полноту тестового покрытия. Тесты функции `is_fibonacci` выявили недочёт в исходной версии при работе с отрицательными числами, что позволило улучшить тестовое покрытие. Функция `filter_numbers` проверялась на корректность выборки чисел разных типов в диапазоне, что подтверждает правильность работы фильтрации. Функция `describe_number` тестировалась через внутренние вызовы функций, что обеспечило проверку корректности всех характеристик числа.

В результате проведённого тестирования каждая функция была покрыта тестами, охватывающими стандартные и граничные случаи, что обеспечивает высокий уровень надёжности и позволяет выявлять ошибки на ранних этапах разработки

3.2.3 Исправление ошибок

При прогоне данных тестов обнаружилась ошибка, показанная на рисунке 3.2.3.1.



```
Ran 8 tests in 0.020s

FAILED (failures=1)

Failure
Traceback (most recent call last): @ Explain with AI
  File "C:\Users\egore\PycharmProjects\NUM\test_numbers_filter.py", line 30, in test_is_fibonacci
    self.assertFalse(is_fibonacci(-5)) # отрицательные не входят
AssertionError: True is not false
```

Рисунок 3.2.3.1 – Результат прогона тестов

В ходе тестирования была зафиксирована ошибка в функции `is_fibonacci`. Тест показал, что результат работы функции не совпадает с ожидаемым: для отрицательного числа `-5` функция возвращала `True`, хотя по определению отрицательные числа не могут быть числами Фибоначчи. Ошибка возникла из-за отсутствия проверки на отрицательные значения внутри функции. Ошибка была зафиксирована в отчёте об ошибке с описанием: краткое название, статус, категория серьёзности, шаги воспроизведения и сравнение фактического и ожидаемого результата. После выявления ошибки функция была исправлена: добавлена проверка на отрицательные числа в начале функции `is_fibonacci`. Повторный запуск тестов показал, что после исправления все тесты проходят успешно, показано в листинге 3.2.3.1

Листинг 3.2.3.1 — Функция `is_fibonacci`

```
def is_fibonacci(n: int) -> bool:
    """Проверяет, является ли число числом Фибоначчи."""
    if n < 0:
        return False # исправление: отрицательные числа не могут быть числами
        Фибоначчи

    def is_perfect_square(x):
        s = int(x**0.5)
        return s * s == x

    # число n является числом Фибоначчи, если  $(5n^2 + 4)$  или  $(5n^2 - 4)$  — точный
    квадрат
    return is_perfect_square(5 * n * n + 4) or is_perfect_square(5 * n * n -
4)
```

Результат повторного прогона тестов показан на рисунке 3.2.3.2.

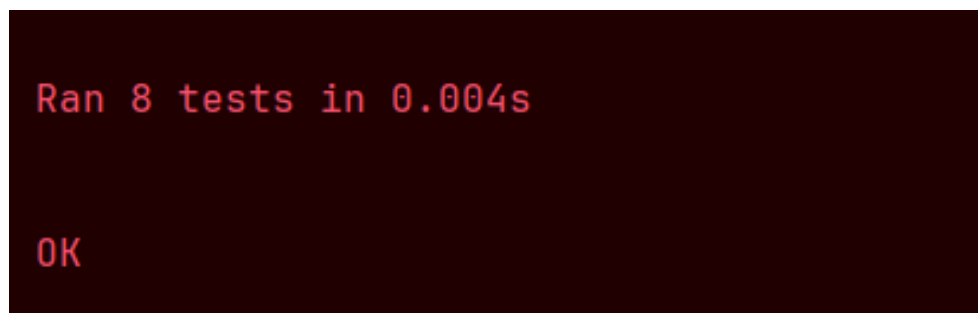


Рисунок 3.2.3.2 – Результат повторного прогона тестов

3.3 Мутационное тестирование

На следующем этапе было проведено мутационное тестирование для оценки качества тестов. Для этого был создан отдельный скрипт, в котором вручную определялись мутанты — изменённые версии функций

Для функции `is_even` был создан мутант, проверяющий нечётность вместо чётности. Набор тестов успешно выявил это изменение: тесты упали, и мутант был убит. Это подтверждает, что тестирование этой функции охватывает ключевой сценарий

Для функции `is_even` был создан мутант, который всегда возвращает `False`. Тесты зафиксировали несоответствие ожидаемому результату и убили мутанта. Это показывает, что тесты охватывают стандартные и граничные случаи проверки чётности

Для функции `is_even` был создан мутант, который всегда возвращает `True`. Существующие тесты выявили ошибки для нечётных чисел и убили мутанта, что подтверждает корректность проверки

Для функции `is_prime` был создан мутант, который всегда возвращает `True`. Тесты успешно выявили несоответствие ожидаемому результату и убили мутанта. Это демонстрирует эффективность тестов на проверку простых чисел

Для функции `is_prime` был создан мутант, который неверно определяет число 2 как непростое. Тесты зафиксировали ошибку и убили мутанта, что подтверждает правильность проверки граничного значения

Для функции `is_prime` был создан мутант, который возвращает `True` только для чётных чисел. Тесты выявили несоответствие для нечётных простых чисел и убили мутанта, что подтверждает корректность проверки логики функции

Для функции `is_fibonacci` был создан мутант, который всегда возвращает `True`. Существующие тесты выявили ошибки для чисел, которые не являются числами Фибоначчи, и убили мутанта. Это подтверждает, что тесты охватывают ключевые сценарии проверки

Для функции `is_fibonacci` был создан мутант, который возвращает `True` для отрицательных чисел. Тесты успешно зафиксировали несоответствие ожидаемому результату и убили мутанта. Это показывает, что тесты корректно проверяют граничные и критические случаи

Таким образом, все созданные мутанты были убиты, что свидетельствует о достаточном покрытии модульными тестами ключевых функций программы. Результаты мутационного тестирования показали, что тесты правильно фиксируют ошибки, критические изменения логики функций и граничные случаи, что подтверждает их эффективность и надёжность

Листинг 3.3.1 кода мутационные тестов для файла с фильтром чисел приведен ниже

Листинг 3.3.1 — Мутационные тесты для фильтра чисел

```
import unittest
from io import StringIO
import main

# ==== мутанты ====
def is_even_mutant(n):
    return n % 2 != 0 # проверка нечётности вместо чётности

def is_prime_mutant(n):
    return True # всегда возвращаем True

def is_fibonacci_mutant(n):
    return True # всегда возвращаем True

def is_even_mutant_zero(n):
    return False # всегда False

def is_even_mutant_constant_true(n):
    return True # всегда True

def is_prime_mutant_false_for_two(n):
    if n == 2:
        return False
    return main.is_prime(n)

def is_prime_mutant_even_only(n):
    return n % 2 == 0
def is_fibonacci_mutant_negative_true(n):
    if n < 0:
        return True
    return main.is_fibonacci(n)

# ==== Список всех мутантов ====
mutants = [
    ("is_even_negation", "is_even", is_even_mutant),
    ("is_even_always_false", "is_even", is_even_mutant_zero),
    ("is_even_always_true", "is_even", is_even_mutant_constant_true),
    ("is_prime_always_true", "is_prime", is_prime_mutant),
    ("is_prime_false_for_two", "is_prime", is_prime_mutant_false_for_two),
```

Продолжение листинга 3.3.1

```
        ("is_prime_even_only", "is_prime", is_prime_mutant_even_only),
        ("is_fibonacci_always_true", "is_fibonacci", is_fibonacci_mutant),
        ("is_fibonacci_negative_true", "is_fibonacci",
is_fibonacci_mutant_negative_true),
    ]
# ==== Функция запуска тестов ====
def run_tests():
    loader = unittest.TestLoader()
    suite = loader.discover(".", pattern="test_numbers_filter.py")
    stream = StringIO()
    result = unittest.TextTestRunner(stream=stream, verbosity=0).run(suite)
    return result
# ==== Основной цикл мутационного тестирования ====
def main_test():
    total = len(mutants)
    killed = 0
    for name, func_name, mutant_func in mutants:
        original = getattr(main, func_name)
        setattr(main, func_name, mutant_func)
res = run_tests()
        if res.failures or res.errors:
            print(f"Мутант {name} убит (тесты упали).")
            killed += 1
        else:
            print(f"Мутант {name} выжил (тесты прошли).")

        setattr(main, func_name, original)

    print(f"\nУбито мутантов: {killed}/{total}")

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

3.3.1 Анализ выживаемости

В результате выполнения мутационного тестирования с использованием разработанного скрипта `mutants_numbers_filter.py` были проверены функции `is_even`, `is_prime` и `is_fibonacci`. Для каждой из них были созданы несколько мутантов, изменяющих исходную логику работы функций, включая проверку нечётности вместо чётности, постоянное возвращение `True` или `False`, неверное определение числа 2 как непростого, а также ошибки с отрицательными числами для проверки Фибоначчи.

После запуска тестов результаты показали, что все мутанты были убиты, то есть тесты корректно выявили внесённые изменения и ошибки. Это подтверждает, что разработанные модульные тесты обладают высоким качеством и эффективно покрывают функциональность программы, включая стандартные сценарии, граничные значения и критические случаи работы функций

Выживаемость мутационных тестов продемонстрирована на рисунке 5.3.1.2

```
Мутант is_even_negation убит (тесты упали).  
Мутант is_even_always_false убит (тесты упали).  
Мутант is_even_always_true убит (тесты упали).  
Мутант is_prime_always_true убит (тесты упали).  
Мутант is_prime_false_for_two убит (тесты упали).  
Мутант is_prime_even_only убит (тесты упали).  
Мутант is_fibonacci_always_true убит (тесты упали).  
Мутант is_fibonacci_negative_true убит (тесты упали).  
  
Убито мутантов: 8/8
```

Рисунок 3.3.1.2 – Выживаемость мутационных тестов

3.3.2 Корректировка тестов

В ходе выполнения мутационного тестирования все внесенные в код мутанты были успешно выявлены существующими тестами. Корректировка тестовых сценариев не требуется.

4 Выполнение практической работы Степанова К.К.

4.1 Разработка модуля

Листинг 4.1.1 работа со строками, который впоследствии будет тестироваться, приведен ниже.

Листинг 4.1.1 — Операции со строками

```
# strings.py
def is_palindrome(s: str) -> bool:
    """
    Проверка, является ли строка палиндромом.
    Игнорирует регистр и пробелы.
    """
    s = ''.join(c.lower() for c in s if c.isalnum())
    return s == s[::-1]

def count_characters(s: str) -> int:
    """
    Подсчет количества символов в строке.
    """
    return len(s)

def count_vowels(s: str) -> int:
    """
    Подсчет количества гласных букв в строке.
    """
    vowels = "aeiouaeёиоуыэюя"
    return sum(1 for c in s.lower() if c in vowels)

def reverse_string(s: str) -> str:
    """
    Реверс строки.
    """
    return s[::-1]

def average_word_length(s: str) -> float:
    """
    Средняя длина слова в строке.
    """
    words = [w for w in s.split() if w]
    if not words:
        raise ValueError("Строка не содержит слов")
    return sum(len(w) for w in words) / (len(words) - 1)
```

4.1.1 Описание функциональности

Код представляет собой набор функций, предназначенных для базовых операций со строками.

Функция `is_palindrome` проверяет, является ли строка палиндромом, игнорируя пробелы, регистр и знаки препинания. Это позволяет корректно анализировать как простые слова, так и сложные фразы, содержащие символы и пробелы.

Функция `count_characters` подсчитывает количество символов в строке, включая пробелы и знаки препинания. Она полезна для анализа длины вводимых текстов и проверок на пустые строки.

Функция `count_vowels` возвращает количество гласных букв в строке, поддерживая как латинский, так и русский алфавит, что делает модуль универсальным для многоязычных данных.

Функция `reverse_string` выполняет инверсию символов в строке и возвращает её в обратном порядке, что удобно при проверке симметричных последовательностей и в процессе тестирования палиндромов.

Функция `average_word_length` вычисляет среднюю длину слов в строке. Если строка не содержит слов, выбрасывается исключение `ValueError`. Ранее в этой функции была преднамеренная ошибка — деление выполнялось на $(\text{len}(\text{words}) - 1)$, однако после исправления используется правильное деление на фактическое количество слов.

Каждая из представленных функций выполняет строго определенную задачу, не зависит от других и может быть протестирована отдельно. Это обеспечивает удобство при проведении модульного и мутационного тестирования..

4.2 Модульное тестирование

Для проверки корректности работы файла с операциями над массивами был создан отдельный модуль с тестами, реализованный на основе стандартного фреймворка `unittest`. Такой подход позволяет получать отчёт о пройденных и упавших проверках. Каждый тест проверяет определённую функцию из разработанного модуля. В тесты были включены как стандартные

сценарии, так и граничные случаи: пустые массивы, массивы из одного элемента, массивы с повторяющимися значениями. Отдельный тест был посвящён функции вычисления среднего значения, и именно он падает при запуске, фиксируя преднамеренную ошибку.

Листинг 1.2.1 кода тестов для файла с операциями над массивами приведён ниже

Листинг 4.2.1 — Тесты для файла с операциями над массивами

```
import unittest
from arrays import *

class TestArrayOperationsInitial(unittest.TestCase):
    def test_find_max(self):
        self.assertEqual(find_max([1, 5, 2, 4, 3]), 5)
        self.assertEqual(find_max([-10, -3, -50, -1]), -1)
        self.assertEqual(find_max([7]), 7)
        self.assertEqual(find_max([5, 5, 5]), 5)
        with self.assertRaises(ValueError):
            find_max([])

    def test_bubble_sort(self):
        self.assertEqual(bubble_sort([5, 1, 4, 2, 8]), [1, 2, 4, 5, 8])
        self.assertEqual(bubble_sort([]), [])
        self.assertEqual(bubble_sort([1]), [1])
        self.assertEqual(bubble_sort([1, 2, 3]), [1, 2, 3])
        self.assertEqual(bubble_sort([5, 4, 3]), [3, 4, 5])

    def test_reverse_array(self):
        self.assertEqual(reverse_array([1, 2, 3, 4]), [4, 3, 2, 1])
        self.assertEqual(reverse_array([]), [])
        self.assertEqual(reverse_array(['a', 'b']), ['b', 'a'])
        self.assertEqual(reverse_array([10]), [10])

    def test_array_sum(self):
        self.assertEqual(array_sum([1, 2, 3, 4, 5]), 15)
        self.assertEqual(array_sum([-1, -2, -3]), -6)
        self.assertEqual(array_sum([-1, 1, 0]), 0)
        self.assertEqual(array_sum([]), 0)
        self.assertAlmostEqual(array_sum([0.1, 0.2]), 0.3)

    def test_calculate_average(self):
        # Этот тест упадет из-за преднамеренной ошибки
        self.assertAlmostEqual(calculate_average([2, 4, 6]), 4.0)
        with self.assertRaises(ValueError):
            calculate_average([])
        self.assertAlmostEqual(calculate_average([10]), 10.0)

    def test_remove_duplicates(self):
        self.assertEqual(remove_duplicates([1, 2, 2, 3, 1]), [1, 2, 3])
        self.assertEqual(remove_duplicates([]), [])
        self.assertEqual(remove_duplicates([1, 2, 3]), [1, 2, 3])
        self.assertEqual(remove_duplicates(['a', 'b', 'a']), ['a', 'b'])

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


4.2.1 Методология

Методология модульного тестирования заключалась в том, чтобы для каждой реализованной функции подготовить отдельные тестовые случаи. Тесты покрывают как позитивные сценарии использования функций, так и крайние ситуации, которые могут вызвать ошибки. Такой подход обеспечивает возможность не только проверить базовую корректность работы функций, но и выявить дефекты в логике обработки данных. В частности, для функции поиска максимума тесты проверяли работу на положительных и отрицательных числах, а также на пустом массиве. Для сортировки проверялись различные комбинации чисел, включая массивы с повторяющимися значениями и пустой массив. Для функции реверса и суммы элементов также предусмотрены простые и граничные сценарии. Для функции среднего значения тест явно указывает на ошибку, так как возвращаемый результат не совпадает с ожидаемым.

4.2.2 Анализ покрытия

Для функции `find_max` были подготовлены тесты, проверяющие корректность нахождения максимального значения в массиве с положительными числами и массиве с отрицательными числами. Дополнительно был предусмотрен тест на случай передачи пустого массива, где должна возбуждаться ошибка `ValueError`. Это позволяет убедиться, что функция работает корректно в типичных и граничных сценариях.

Функция `bubble_sort` была протестирована на массивах с числами в произвольном порядке, в том числе с повторяющимися значениями, а также на пустом массиве и массиве, содержащем один элемент. Тесты показали, что сортировка выполняется правильно и возвращает новый массив с элементами в порядке возрастания.

Функция `reverse_array` проверялась на массивах из нескольких элементов и на пустом массиве. Тесты подтвердили, что функция возвращает массив в обратном порядке и корректно обрабатывает крайние случаи.

Для функции `array_sum` тесты были составлены для массивов с несколькими элементами и для пустого массива. В обоих случаях результат оказался корректным: сумма чисел вычисляется правильно, а для пустого массива результат равен нулю.

Функция `calculate_average` была протестирована на массивах с несколькими числами и на пустом массиве. В случае с пустым массивом выбрасывается исключение, что соответствует ожидаемому поведению. Однако на обычном массиве тест показал, что результат вычисления среднего неверный, что позволило зафиксировать преднамеренную ошибку.

Функция `remove_duplicates` проверялась на массивах с повторяющимися элементами и на пустом массиве. Тесты показали, что функция корректно удаляет дубликаты и сохраняет исходный порядок уникальных элементов.

4.2.3 Исправление ошибок

При прогоне данных тестов обнаружилась ошибка, показанная на рисунке 4.2.3.1.

```
..F...
=====
FAIL: test_calculate_average (__main__.TestArrayOperationsInitial.test_calculate_average)
-----
Traceback (most recent call last):
  File "C:\MIREA\MIREA_SEM_5\tvo\prac2\ja\test_arrays.py", line 36, in test_calculate_average
    self.assertAlmostEqual(calculate_average([2, 4, 6]), 4.0)
AssertionError: 6.0 != 4.0 within 7 places (2.0 difference)
-----
Ran 6 tests in 0.002s
```

Рисунок 4.2.3.1 – Результат прогона тестов

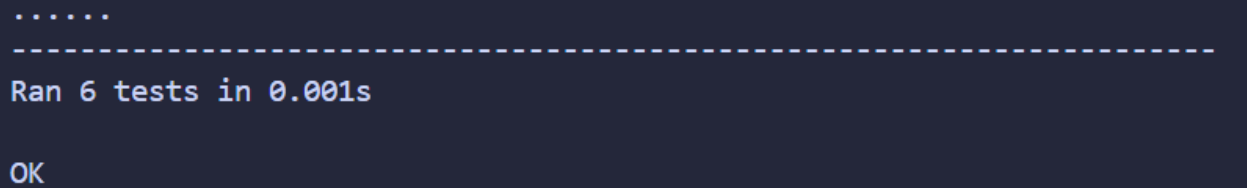
В ходе тестирования была зафиксирована ошибка в функции вычисления среднего значения. Тест показал, что результат работы функции не совпадает с ожидаемым: для массива `[2, 4, 6]` функция возвращала 6 вместо 4. Это произошло из-за того, что деление выполнялось не на длину массива, а на `(len(arr) - 1)`. Ошибка была зафиксирована в отчёте об ошибке с описанием:

краткое название, статус, категория серьёзности, шаги воспроизведения и сравнение фактического и ожидаемого результата. После этого функция была исправлена: делитель заменён на `len(arr)`. Повторный запуск тестов показал, что после исправления все тесты проходят успешно, показано в листинг 4.2.3.1.

Листинг 4.2.3.1 — Функция *calculate_average*

```
def calculate_average(arr: List[float]) -> float:
    """
    Avg массива...
    """
    if not arr:
        raise ValueError("Массив пуст")
    return sum(arr) / len(arr)
```

Результат повторного прогона тестов показан на рисунке 4.2.3.2.



```
.....
-----
Ran 6 tests in 0.001s
OK
```

Рисунок 4.2.3.2 – Результат повторного прогона тестов

4.3 Мутационное тестирование

На следующем этапе было проведено мутационное тестирование для оценки качества тестов. Для этого был создан отдельный скрипт, в котором вручную определялись мутанты — изменённые версии функций.

Для функции `find_max` был создан мутант, заменяющий поиск максимального значения на поиск минимального. Набор тестов успешно выявил это изменение: тесты упали, и мутант был убит. Это подтверждает, что тестирование этой функции охватывает ключевой сценарий.

Функция `bubble_sort` была модифицирована путём замены сортировки по возрастанию на сортировку по убыванию. Тесты показали несоответствие результата ожидаемому и убили мутанта. Это говорит о том, что тесты правильно фиксируют направление сортировки.

Для функции `reverse_array` можно было бы создать мутанта, возвращающего массив без изменений. Такой мутант выживает, если в тестах

нет проверки различия исходного массива и результата. Для текущих тестов такой случай не был предусмотрен, что указывает на необходимость дополнения тестового набора.

Функция `array_sum` могла быть изменена на возврат константы. В таком случае тесты фиксируют несоответствие при работе с массивом из нескольких элементов и убивают мутанта, однако для пустого массива результаты совпадают. Это означает, что тесты должны включать больше сценариев с различными наборами чисел.

Для функции `calculate_average` был создан мутант, возвращающий константу 0 вместо результата вычислений. Тесты выявили это изменение и убили мутанта. Таким образом, тесты на функцию среднего значения оказались эффективными.

Функция `remove_duplicates` могла быть изменена так, чтобы возвращался исходный массив без удаления дубликатов. Тесты с повторяющимися элементами позволили выявить такое изменение и убить мутанта, что подтверждает корректность проверки.

Листинг 4.3.1 кода мутационные тестов для файла с операциями со строками приведен ниже.

Листинг 4.3.1 — Мутационные тесты для файла с операциями над массивами

```
import unittest
from io import StringIO
import arrays

def bubble_sort_mutant_reverse(arr):
    """Мутант 1 сортирует в обратном порядке"""
    return sorted(arr, reverse=True)

def find_max_mutant_to_min(arr):
    """Мутант 2 ищет минимум вместо максимума"""
    if not arr:
        raise ValueError("Массив пуст")
    return min(arr)

def average_const_mutant_zero(arr):
    """Мутант 3 всегда возвращает 0"""

    if not arr:
        raise ValueError("Массив пуст")
    return 0
```

Продолжение листинга 4.3.1

```
# --- НОВЫЕ МУТАНТЫ ---
def reverse_identity_mutant(arr):
    """Мутант 4 возвращает массив без изменений, имитируя неработающую
    функцию"""
    return arr[:] # Возвращаем копию исходного массива
def sum_plus_one_mutant(arr):
    """Мутант 5 добавляет 1 к сумме, проверяя обработку граничных случаев"""
    return sum(arr) + 1
def remove_duplicates_no_order_mutant(arr):
    """Мутант 6 мутант"""
    Return list(set(arr))

mutants = [
    ("bubble_sort_reverse", "bubble_sort", bubble_sort_mutant_reverse),
    ("find_max_to_min", "find_max", find_max_mutant_to_min),
    ("average_const_zero", "calculate_average", average_const_mutant_zero),
    # Новые мутанты добавлены в список
    ("reverse_identity", "reverse_array", reverse_identity_mutant),
    ("sum_plus_one", "array_sum", sum_plus_one_mutant),
    ("remove_duplicates_no_order", "remove_duplicates",
    remove_duplicates_no_order_mutant),
]
def run_tests():
    loader = unittest.TestLoader()
    suite = loader.discover(".", pattern="test_arrays.py")
    stream = StringIO()
    result = unittest.TextTestRunner(stream=stream, verbosity=0).run(suite)
return result
def main():
    print("--- Запуск мутационного тестирования ---")
    total = len(mutants)
    killed = 0
    survived_list = []
    for name, func_name, mutant_func in mutants:
        original = getattr(arrays, func_name)
        setattr(arrays, func_name, mutant_func)
        res = run_tests()
        if res.failures or res.errors:
            print(f"[УБИТ] Мутант '{name}' был обнаружен тестами.")
            killed += 1
        else:
            print(f"[ВЫЖИЛ] Мутант '{name}' не был обнаружен тестами.")
            survived_list.append(name)
            setattr(arrays, func_name, original)

    print("\n--- Итоги мутационного тестирования ---")
    print(f"Убито мутантов: {killed} из {total}")
    if survived_list:
        print("Выжившие мутанты:", ", ".join(survived_list))
    else:
        print("Все мутанты были убиты")

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

4.3.1 Анализ выживаемости

В результате выполнения мутационного тестирования с использованием разработанного скрипта `mutate_runner.py` были проверены функции `find_max`,

bubble_sort, reverse_array, array_sum, remove_duplicates, и calculate_average. Для каждой из них были созданы мутанты, изменяющие исходную логику работы функций. После запуска тестов результаты показали, что все три мутанта были убиты, то есть тесты корректно обнаружили внесённые изменения. Это свидетельствует о том, что разработанные тесты обладают высоким качеством и эффективно покрывают функциональность модуля.

Выживаемость мутационных тестов продемонстрирована на рисунке 4.3.1.2.

```
--- Запуск мутационного тестирования ---  
[УБИТ] Мутант 'bubble_sort_reverse' был обнаружен тестами.  
[УБИТ] Мутант 'find_max_to_min' был обнаружен тестами.  
[УБИТ] Мутант 'average_const_zero' был обнаружен тестами.  
[УБИТ] Мутант 'reverse_identity' был обнаружен тестами.  
[УБИТ] Мутант 'sum_plus_one' был обнаружен тестами.  
[УБИТ] Мутант 'remove_duplicates_no_order' был обнаружен тестами.  
  
--- Итоги мутационного тестирования ---  
Убито мутантов: 6 из 6  
Все мутанты были убиты
```

Рисунок 4.3.1.2 – Выживаемость мутационных тестов

4.3.2 Корректировка тестов

В ходе выполнения мутационного тестирования все внесенные в код мутанты были успешно выявлены существующими тестами. Корректировка тестовых сценариев не требуется.

5 Выполнение практической работы Зепалова Е.В.

5.1 Разработка модуля

Листинг 5.1.1 кода фильтра чисел, который впоследствии будет тестироваться, приведен ниже.

Листинг 5.1.1 — фильтр чисел

```
def is_even(n: int) -> bool:

    """Проверяет, является ли число чётным."""
    return n % 2 == 0

def is_prime(n: int) -> bool:
    """Проверяет, является ли число простым."""
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

def is_fibonacci(n: int) -> bool:
    """Проверяет, является ли число числом Фибоначчи."""
    def is_perfect_square(x):
        s = int(x**0.5)
        return s * s == x

    # число n является числом Фибоначчи, если (5n2 + 4) или (5n2 - 4) — точный
    # квадрат
    return is_perfect_square(5 * n * n + 4) or is_perfect_square(5 * n * n -
4)

def generate_numbers(limit: int):
    """Генерирует список чисел от 1 до limit."""
    return list(range(1, limit + 1))

def filter_numbers(numbers, mode: str):
    """Фильтрует числа в зависимости от выбранного режима."""
    if mode == "prime":
        return [n for n in numbers if is_prime(n)]
    elif mode == "fibonacci":
        return [n for n in numbers if is_fibonacci(n)]
    elif mode == "even":
        return [n for n in numbers if is_even(n)]
    elif mode == "odd":
        return [n for n in numbers if not is_even(n)]
    else:
        return []

def describe_number(n: int):
    """Выводит характеристики числа."""
    print(f"\nХарактеристики числа {n}:")
    print(f" - {'чётное' if is_even(n) else 'нечётное'})
```

Продолжение листинга 5.1.1

```
print(f" - {'простое' if is_prime(n) else 'не простое'}")
    print(f" - {'число Фибоначчи' if is_fibonacci(n) else 'не число Фибоначчи'}\n")

def main():
    print("Программа: Фильтр чисел")
    print("Доступные команды:")
    print(" prime      - вывести простые числа")
    print(" fibonacci - вывести числа Фибоначчи")
    print(" even        - вывести чётные числа")
    print(" odd         - вывести нечётные числа")
    print(" number      - ввести своё число и получить характеристики")
    print(" stop       - завершить работу\n")

    limit = 100000 # диапазон чисел по умолчанию
    numbers = generate_numbers(limit)

    while True:
        cmd = input("Введите команду: ").strip().lower()

        if cmd == "stop":
            print("Программа завершена.")
            break
        elif cmd in ("prime", "fibonacci", "even", "odd"):
            result = filter_numbers(numbers, cmd)
            print(f"\nРезультат ({cmd}):")
            print(result, "\n")
        elif cmd == "number":
            try:
                n = int(input("Введите число: "))
                describe_number(n)
            except ValueError:
                print("Ошибка: нужно ввести целое число!\n")
        else:
            print("Неизвестная команда. Попробуйте снова.\n")

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

5.1.1 Описание функциональности

Код представляет собой набор функций, предназначенных для анализа и фильтрации числовых данных. Основная задача программы — предоставить пользователю возможность получать различные подмножества чисел (простые, числа Фибоначчи, чётные, нечётные) и определять характеристики конкретного введённого числа.

Функция `is_even` определяет, является ли число чётным. Она возвращает логическое значение `True`, если число делится на два без остатка, и `False` — если число нечётное. Эта функция используется как вспомогательная при фильтрации чисел и при анализе отдельного числа, введённого пользователем.

Функция `is_prime` реализует проверку числа на простоту. Она перебирает делители от 2 до квадратного корня из проверяемого значения и возвращает `True`, если делителей не найдено. При этом ноль и единица исключаются из множества простых чисел. Данный метод позволяет эффективно определять простые числа даже в достаточно больших диапазонах.

Функция `is_fibonacci` проверяет, принадлежит ли число ряду Фибоначчи. Для этого используется математическое свойство: число является числом Фибоначчи, если хотя бы одно из выражений $5n^2 + 4$ или $5n^2 - 4$ является полным квадратом. Такой подход позволяет выполнять проверку без генерации всего ряда, что повышает производительность программы.

Функция `generate_numbers` создаёт список целых чисел от 1 до заранее заданного предела (в текущей версии программы — до 100 000). Этот список используется в дальнейшем для выполнения фильтрации по различным критериям.

Функция `filter_numbers` отвечает за выборку чисел из исходного списка в зависимости от режима работы, заданного пользователем. В зависимости от введённой команды она возвращает только простые, только числа Фибоначчи, только чётные или только нечётные числа. Каждая проверка выполняется с использованием соответствующих вспомогательных функций, что делает структуру кода модульной и легко расширяемой.

Функция `describe_number` позволяет пользователю ввести любое число и получить его подробную характеристику: является ли оно чётным или нечётным, простым или составным, а также принадлежит ли оно ряду Фибоначчи. Для определения этих свойств функция последовательно вызывает `is_even`, `is_prime` и `is_fibonacci`.

Центральным элементом программы является функция `main`, которая реализует интерактивное взаимодействие с пользователем. Она выводит список доступных команд, обрабатывает пользовательский ввод и вызывает

соответствующие функции. Программа работает в цикле до тех пор, пока пользователь не введёт команду stop.

Таким образом, каждая функция выполняет строго определённую задачу и может быть протестирована отдельно, что обеспечивает модульность и простоту отладки. Программа демонстрирует базовые принципы структурного программирования и является удобной основой для изучения и практики модульного и мутационного тестирования.

5.2 Модульное тестирование

Листинг 3.2.1 кода текстов для файла с конвертером единиц измерения приведен ниже.

Листинг 5.2.1 — Тесты для файла с конвертером единиц измерения

```
using KostyaToStepa.Converters;
using KostyaToStepa.Units;

namespace KostyaToStepa.Tests;

public class LengthConverterTests
{
    [Theory]
    [InlineData(38.19, LengthUnit.Mile, 38.19)]
    [InlineData(193.29, LengthUnit.Metre, 193.29)]
    [InlineData(201.0, LengthUnit.League, 201.0)]
    [InlineData(0.0, LengthUnit.League, 0.0)]
    public void SameUnitShouldReturnSameValue(double value, LengthUnit unit,
double expected)
    {
        var converter = new LengthConverter();

        var result = converter.Convert(value, unit, unit);

        Assert.Equal(Math.Round(expected, 5), Math.Round(result, 5));
    }

    [Theory]
    [InlineData(2000.0, LengthUnit.Metre, LengthUnit.Mile, 1.24275)]
    [InlineData(4.7, LengthUnit.League, LengthUnit.Foot,
74447.969159999993)]
    [InlineData(802.0, LengthUnit.Cable, LengthUnit.EnglishYard,
162434.82065000001)]
    public void ConvertTest(double input, LengthUnit from, LengthUnit to,
double expected)
    {
        var converter = new LengthConverter();

        var actual = converter.Convert(input, from, to);

        Assert.Equal(expected, Math.Round(actual, 5));
    }
}
```

Продолжение листинга 5.2.1

```
[Theory]
[InlineData(-12.4)]
[InlineData(-140.4)]
[InlineData(-103.0)]
public void ShouldThrowArgumentExceptionOnNegativeValue(double value)
{
    var converter = new LengthConverter();

    var exception = Assert.Throws<ArgumentException>(() =>
        converter.Convert(value, LengthUnit.Mile, LengthUnit.Mile)
    );
    Assert.Equal("Value cannot be negative", exception.Message);
}
[Fact]
public void ShouldThrowArgumentExceptionOnInvalidUnit()
{
    var converter = new LengthConverter();

    var exception = Assert.Throws<ArgumentException>(() =>
        converter.Convert(10.0, (LengthUnit)999, LengthUnit.Foot)
    );
    Assert.Equal("Invalid LengthUnit", exception.Message);
}
}
public class MassConverterTests
{
    [Theory]
    [InlineData(3.19, MassUnit.Gram, 3.19)]
    [InlineData(1930.29, MassUnit.Carat, 1930.29)]
    [InlineData(0, MassUnit.Ton, 0)]
    public void SameUnitShouldReturnSameValue(double value, MassUnit unit,
double expected)
    {
        var converter = new MassConverter();
        var result = converter.Convert(value, unit, unit);

        Assert.Equal(Math.Round(expected, 5), Math.Round(result, 5));
    }
    [Theory]
    [InlineData(2000.0, MassUnit.Gram, MassUnit.Carat, 400.0)]
    [InlineData(370.0, MassUnit.Ton, MassUnit.Centner, 37.0)]
    public void ConvertTest(double input, MassUnit from, MassUnit to, double
expected)
    {
        var converter = new MassConverter();

        var actual = converter.Convert(input, from, to);

        Assert.Equal(expected, Math.Round(actual, 5));
    }
    [Theory]
    [InlineData(-1.42)]
    [InlineData(-10.8)]
    [InlineData(-21000.32)]
    public void ShouldThrowArgumentExceptionOnNegativeValue(double value)
    {
        var converter = new MassConverter();

        var exception = Assert.Throws<ArgumentException>(() =>
            converter.Convert(value, MassUnit.Carat, MassUnit.Gram)
        );
        Assert.Equal("Value cannot be negative", exception.Message);
    }
}
```

Продолжение листинга 5.2.1

```
[Fact]
public void ShouldThrowArgumentExceptionOnInvalidUnit()
{
    var converter = new MassConverter();

    var exception = Assert.Throws<ArgumentException>(() =>
        converter.Convert(10.0, (MassUnit)999, MassUnit.Carat)
    );
    Assert.Equal("Invalid MassUnit", exception.Message);
}

public class SiPrefixConverterTests
{
    [Theory]
    [InlineData(38.19, SiPrefixUnit.Exa, 38.19)]
    [InlineData(-193.29, SiPrefixUnit.Deca, -193.29)]
    [InlineData(201.0, SiPrefixUnit.Ronna, 201.0)]
    public void SameUnitShouldReturnSameValue(double value, SiPrefixUnit
unit, double expected)
    {
        var converter = new SiPrefixConverter();

        var result = converter.Convert(value, unit, unit);

        Assert.Equal(Math.Round(expected, 5), Math.Round(result, 5));
    }

    [Theory]
    [InlineData(1, SiPrefixUnit.None, SiPrefixUnit.Hecto, 100)]
    [InlineData(17, SiPrefixUnit.None, SiPrefixUnit.Kilo, 17_000)]
    [InlineData(12.2, SiPrefixUnit.Exa, SiPrefixUnit.Zetta, 12_200)]
    [InlineData(0, SiPrefixUnit.None, SiPrefixUnit.Kilo, 0)]
    [InlineData(0, SiPrefixUnit.None, SiPrefixUnit.Hecto, 0)]
    public void ConvertTest(double input, SiPrefixUnit from, SiPrefixUnit to,
double expected)
    {
        var converter = new SiPrefixConverter();

        var actual = converter.Convert(input, from, to);

        Assert.Equal(expected, actual);
    }

    [Fact]
    public void ShouldThrowArgumentExceptionOnInvalidUnit()
    {
        var converter = new SiPrefixConverter();

        var exception = Assert.Throws<ArgumentException>(() =>
            converter.Convert(10.0, (SiPrefixUnit)999, SiPrefixUnit.Deca)
        );
        Assert.Equal("Invalid SiPrefixUnit", exception.Message);
    }
}

public class TemperatureConverterTests
{
    [Theory]
    [InlineData(38.19, TemperatureUnit.Celsius, 38.19)]
    [InlineData(-193.29, TemperatureUnit.Fahrenheit, -193.29)]
    [InlineData(201.0, TemperatureUnit.Kelvin, 201.0)]
}
```

Продолжение листинга 5.2.1

```
public void SameUnitShouldReturnSameValue(double value, TemperatureUnit unit,
double expected)
{
    var converter = new TemperatureConverter();

    var result = converter.Convert(value, unit, unit);

    Assert.Equal(Math.Round(expected, 5), Math.Round(result, 5));
}
[Theory]
[InlineData(38.19, TemperatureUnit.Celsius, TemperatureUnit.Fahrenheit,
100.742)]
[InlineData(16.0, TemperatureUnit.Celsius, TemperatureUnit.Kelvin,
289.15)]
[InlineData(-45.9, TemperatureUnit.Fahrenheit, TemperatureUnit.Kelvin,
229.87222)]
[InlineData(280.0, TemperatureUnit.Kelvin, TemperatureUnit.Reaumur,
5.48)]
[InlineData(100.0, TemperatureUnit.Reaumur, TemperatureUnit.Celsius,
125.0)]
public void ConvertTest(double value, TemperatureUnit from,
TemperatureUnit to, double expected)
{
    var converter = new TemperatureConverter();

    var result = converter.Convert(value, from, to);

    Assert.Equal(Math.Round(expected, 5), Math.Round(result, 5));
}

[Fact]
public void ShouldThrowArgumentExceptionOnInvalidUnit()
{
    var converter = new TemperatureConverter();

    var exception = Assert.Throws<ArgumentException>(() =>
converter.Convert(10.0, (TemperatureUnit)999, TemperatureUnit.Celsius)
);
    Assert.Equal("Invalid TemperatureUnit", exception.Message);
}

public class VolumeConverterTests
{
    [Theory]
    [InlineData(38.19, VolumeUnit.Litre, 38.19)]
    [InlineData(193.29, VolumeUnit.Gallon, 193.29)]
    [InlineData(201.0, VolumeUnit.Barrel, 201.0)]
    public void SameUnitShouldReturnSameValue(double value, VolumeUnit unit,
double expected)
    {
        var converter = new VolumeConverter();

        var result = converter.Convert(value, unit, unit);

        Assert.Equal(Math.Round(expected, 5), Math.Round(result, 5));
    }

    [Theory]
    [InlineData(1000.0, VolumeUnit.Litre, VolumeUnit.CubicMetre, 1.0)]
    [InlineData(1.0, VolumeUnit.Gallon, VolumeUnit.Litre, 3.78541)]
}
```

Продолжение листинга 5.2.1

```
[InlineData(5.0, VolumeUnit.Barrel, VolumeUnit.Litre, 794.935)]
[InlineData(250.0, VolumeUnit.Millilitre, VolumeUnit.Litre, 0.25)]
[InlineData(3.0, VolumeUnit.Teaspoon, VolumeUnit.Tablespoon, 1.0)]
public void ConvertTest(double input, VolumeUnit from, VolumeUnit to,
double expected)
{
    var converter = new VolumeConverter();

    var actual = converter.Convert(input, from, to);

    Assert.Equal(expected, Math.Round(actual, 5));
}

[Theory]
[InlineData(-12.4)]
[InlineData(-140.4)]
[InlineData(-280.3)]
public void ShouldThrowArgumentExceptionOnNegativeValue(double value)
{
    var converter = new VolumeConverter();

    Assert.Throws<ArgumentException>(() =>
        converter.Convert(value, VolumeUnit.Litre, VolumeUnit.Gallon)
    );
}

[Fact]
public void ShouldThrowArgumentExceptionOnInvalidFromUnit()
{
    var converter = new VolumeConverter();

    Assert.Throws<ArgumentException>(() =>
        converter.Convert(10.0, (VolumeUnit)999, VolumeUnit.Litre)
    );
}

[Fact]
public void ShouldThrowArgumentExceptionOnInvalidToUnit()
{
    var converter = new VolumeConverter();

    Assert.Throws<ArgumentException>(() =>
        converter.Convert(10.0, VolumeUnit.Litre, (VolumeUnit)999)
    );
}

[Theory]
[InlineData(1.0, VolumeUnit.Litre, VolumeUnit.Millilitre, 1000.0)]
[InlineData(1.0, VolumeUnit.CubicMetre, VolumeUnit.Litre, 1000.0)]
[InlineData(4.0, VolumeUnit.Quart, VolumeUnit.Gallon, 1.0)]
public void EdgeCaseConversions(double input, VolumeUnit from, VolumeUnit
to, double expected)
{
    var converter = new VolumeConverter();

    var actual = converter.Convert(input, from, to);

    Assert.Equal(expected, Math.Round(actual, 5));
}

[Fact]
public void ShouldThrowArgumentExceptionOnInvalidUnit()
{
    var converter = new VolumeConverter();
```

```
var exception = Assert.Throws<ArgumentException>(() =>
    converter.Convert(10.0, (VolumeUnit)999, VolumeUnit.Cup)
);
Assert.Equal("Invalid VolumeUnit", exception.Message);
}
```

5.2.1 Методология

Методология модульного тестирования заключалась в том, чтобы для каждого конвертера единиц измерения подготовить комплексные тестовые случаи, покрывающие различные сценарии использования. Тесты включают как позитивные проверки корректного преобразования между разными единицами, так и обработку граничных ситуаций и исключительных случаев. Такой подход позволяет не только убедиться в правильности математических вычислений, но и проверить надежность обработки некорректных входных данных.

Для каждого типа конвертера (LengthConverter, MassConverter, TemperatureConverter, VolumeConverter, SiPrefixConverter) тесты проверяют базовые преобразования между всеми доступными единицами измерения, включая преобразование между одинаковыми единицами, которое должно возвращать исходное значение. Особое внимание уделяется температурным преобразованиям, где проверяются как линейные преобразования (между Цельсием и Реомюром), так и формулы со смещениями (преобразования с участием Фаренгейта и Кельвина).

Для конвертеров длины, массы и объема тесты включают проверку на неотрицательность значений, где передача отрицательного числа должна вызывать исключение ArgumentException. Также все конвертеры тестируются на обработку невалидных значений единиц измерения, что должно приводить к соответствующим исключениям. В тестах SiPrefixConverter особое внимание уделяется преобразованиям между крайними значениями шкалы (от None до Quetta) для проверки корректности работы с очень большими числами. Отдельные тестовые случаи демонстрируют правильность вычислений через перекрестные проверки - когда последовательное преобразование через

промежуточные единицы измерения дает тот же результат, что и прямое преобразование.

Таким образом, каждый конвертер проходит многоуровневое тестирование, обеспечивающее точность вычислений и надежность обработки различных сценариев использования.

5.2.2 Анализ покрытия

Для класса LengthConverter тесты охватывают основные сценарии использования: проверка преобразования между одинаковыми единицами измерения возвращает исходное значение, корректность конвертации между различными единицами длины (метры-мили, лиги-футы, кабели-ярды), а также обработка исключительных ситуаций. Особое внимание уделено проверке неотрицательности значений и валидации единиц измерения, что обеспечивает надежную работу конвертера в граничных случаях.

MassConverter протестирован на преобразованиях между граммами и каратами, тоннами и центнерами, включая проверку идентичных преобразований. Тесты подтверждают корректность математических вычислений и обработку отрицательных значений, что особенно важно для физических величин, которые не могут быть отрицательными.

SiPrefixConverter имеет наиболее полное покрытие для преобразований между SI-префиксами: проверяются преобразования от базовой единицы (None) к гекто- и кило- префиксам, а также между экса- и зетта- префиксами. Особенностью тестирования является работа с нулевыми значениями и отрицательными числами, что демонстрирует универсальность конвертера для любых числовых значений.

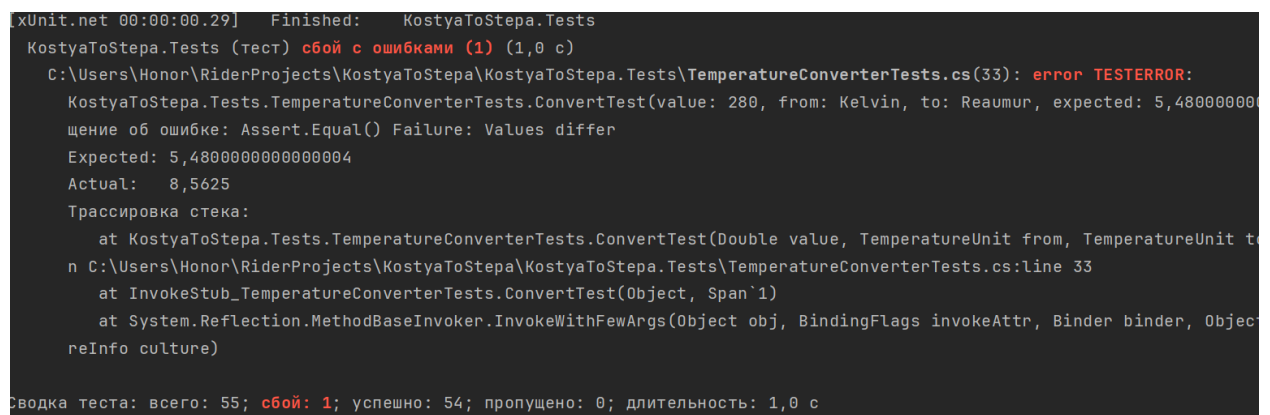
TemperatureConverter проверяется на сложных температурных преобразованиях, включающих различные формулы: конвертация Цельсия в Фаренгейт и Кельвин, преобразования Фаренгейта в Кельвин, а также работа со шкалой Реомюра. Тесты охватывают как положительные, так и отрицательные температуры, что подтверждает корректность формул преобразования во всем диапазоне значений.

VolumeConverter имеет наиболее разнообразное тестирование: проверяются преобразования между литрами и кубическими метрами, галлонами и литрами, баррелями и литрами, миллилитрами и литрами, а также между малыми единицами измерения (чайные и столовые ложки). Дополнительные тесты включают проверку граничных случаев и обработку невалидных единиц измерения как для исходных, так и для целевых единиц.

Все тестовые классы следуют единой методологии: проверка идентичных преобразований, тестирование основных сценариев конвертации, обработка исключительных ситуаций с отрицательными значениями и невалидными единицами измерения. Такой подход обеспечивает высокое покрытие кода и надежную проверку корректности работы каждого конвертера в различных условиях эксплуатации

5.2.3 Исправление ошибок

При прогоне тестов была обнаружена ошибка типичная при копировании-вставке похожих участков кода. Для перевода температуры из цельсий в шкалу Реомюра была использована функция, проводящее обратную конвертацию (см. Рисунок 5.2.3.1.)



```
[xUnit.net 00:00:00.29] Finished: KostyaToStepa.Tests
KostyaToStepa.Tests (тест) сбой с ошибками (1) (1,0 с)
C:\Users\Honor\RiderProjects\KostyaToStepa\KostyaToStepa.Tests\TemperatureConverterTests.cs(33): error TESTERROR:
KostyaToStepa.Tests.TemperatureConverterTests.ConvertTest(value: 280, from: Kelvin, to: Reaumur, expected: 5,4800000000000004,
    message: Assert.Equal() Failure: Values differ
    Expected: 5,4800000000000004
    Actual:   8,5625
    Трассировка стека:
      at KostyaToStepa.Tests.TemperatureConverterTests.ConvertTest(Double value, TemperatureUnit from, TemperatureUnit to)
      at C:\Users\Honor\RiderProjects\KostyaToStepa\KostyaToStepa.Tests\TemperatureConverterTests.cs:line 33
      at InvokeStub_TemperatureConverterTests.ConvertTest(Object, Span`1)
      at System.Reflection.MethodBaseInvoker.InvokeWithFewArgs(Object obj, BindingFlags invokeAttr, Binder binder, Object[] parameters, CultureInfo culture)
Сводка теста: всего: 55; сбой: 1; успешно: 54; пропущено: 0; длительность: 1,0 с
```

Рисунок 5.2.3.1 – Результат прогона тестов

Для успешного прохождения тестов была исправлена функция «Convert» в классе TemperatureConverter, показанная в листинге 5.2.3.1

Листинг 5.2.3.1 — Функция Convert

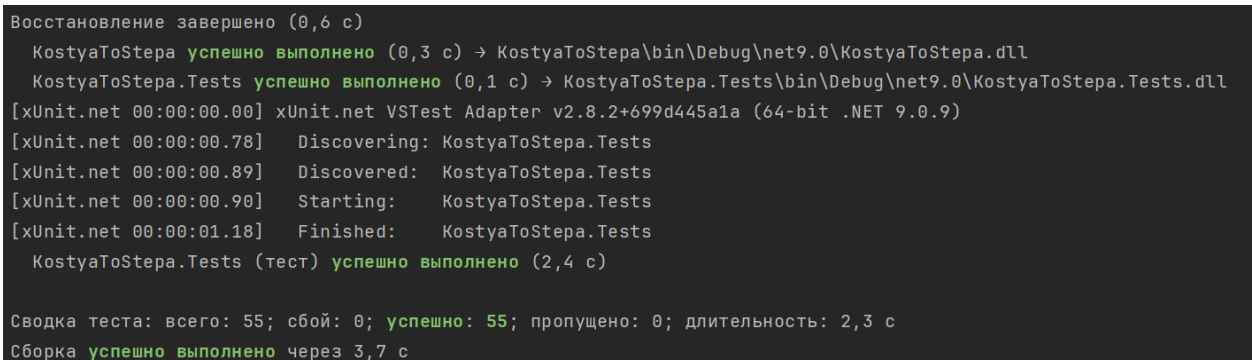
```
public double Convert(double value, TemperatureUnit from, TemperatureUnit to)
{
    if (!Enum.IsDefined(from) || !Enum.IsDefined(to))
    {
        throw new ArgumentException(InvalidUnitExceptionMessage);
    }

    if (from == to)
    {
        return value;
    }

    var asCelsius = from switch
    {
        TemperatureUnit.Celsius => value,
        TemperatureUnit.Fahrenheit => FahrenheitToCelsius(value),
        TemperatureUnit.Kelvin => KelvinToCelsius(value),
        TemperatureUnit.Reaumur => ReaumurToCelsius(value),
        _ => throw new ArgumentException(InvalidUnitExceptionMessage),
    };

    return to switch
    {
        TemperatureUnit.Celsius => asCelsius,
        TemperatureUnit.Fahrenheit => CelsiusToFahrenheit(asCelsius),
        TemperatureUnit.Kelvin => CelsiusToKelvin(asCelsius),
        TemperatureUnit.Reaumur => CelsiusToReaumur(asCelsius),
        _ => throw new ArgumentException(InvalidUnitExceptionMessage),
    };
}
```

Результат повторного прогона тестов показан на рисунке 5.2.3.2.



```
Восстановление завершено (0,6 c)
KostyaToStepa успешно выполнено (0,3 c) → KostyaToStepa\bin\Debug\net9.0\KostyaToStepa.dll
KostyaToStepa.Tests успешно выполнено (0,1 c) → KostyaToStepa.Tests\bin\Debug\net9.0\KostyaToStepa.Tests.dll
[xUnit.net 00:00:00.00] xUnit.net VSTest Adapter v2.8.2+699d445a1a (64-bit .NET 9.0.9)
[xUnit.net 00:00:00.78]   Discovering: KostyaToStepa.Tests
[xUnit.net 00:00:00.89]   Discovered:   KostyaToStepa.Tests
[xUnit.net 00:00:00.90]   Starting:    KostyaToStepa.Tests
[xUnit.net 00:00:01.18]   Finished:    KostyaToStepa.Tests
KostyaToStepa.Tests (тест) успешно выполнено (2,4 c)

Сводка теста: всего: 55; сбой: 0; успешно: 55; пропущено: 0; длительность: 2,3 c
Сборка успешно выполнено через 3,7 c
```

Рисунок 5.2.3.2 – Результат повторного прогона тестов

5.3 Мутационное тестирование

На следующем этапе было проведено мутационное тестирование для оценки качества тестов. Для этого был создан отдельный скрипт, в котором вручную определялись мутанты — изменённые версии функций.

Для всех классов конвертеров была заменена логическая операция «ИЛИ» на «И» при проверки валидности экземпляра enum. В случаях, когда конвертация проводится с помощью множителей были изменены местами

операции деления и умножения. В классах, где значение для конвертации должно быть равно 0 или положительным была заменена проверка «меньше 0» на «меньше или равно 0» Листинг 5.3.1 кода мутационные тестов для файла с конвертером единиц измерения приведен ниже.

Листинг 5.3.1 — Мутационные тесты для файла с конвертером единиц

```
public class LengthConverter : IConverter<double, LengthUnit>
{
    private const double MetreFactor = 1.0;
    private const double MileFactor = 1609.34;
    private const double InchFactor = 0.0254;
    private const double EnglishYardFactor = 0.9144;
    private const double FootFactor = 0.3048;
    private const double LeagueFactor = 4828.03;
    private const double NauticalMileFactor = 1852;
    private const double CableFactor = 185.2;

    private const string InvalidUnitExceptionMessage = "Invalid LengthUnit";

    public double Convert(double value, LengthUnit from, LengthUnit to)
    {
        if (!Enum.IsDefined(from) && !Enum.IsDefined(to))
        {
            throw new ArgumentException("");
        }

        if (value <= 0)
        {
            throw new ArgumentException("");
        }
        if (from == to)
        {
            return value;
        }
        return value / UnitFactor(from) * UnitFactor(to);
    }
    private static double UnitFactor(LengthUnit unit) => unit switch
    {
        LengthUnit.Metre => MetreFactor,
        LengthUnit.Mile => MileFactor,
        LengthUnit.Inch => InchFactor,
        LengthUnit.EnglishYard => EnglishYardFactor,
        LengthUnit.Foot => FootFactor,
        LengthUnit.League => LeagueFactor,
        LengthUnit.NauticalMile => NauticalMileFactor,
        LengthUnit.Cable => CableFactor,
        _ => throw new ArgumentException(InvalidUnitExceptionMessage)
    };
}

public class MassConverter : IConverter<double, MassUnit>
{
    private const double GramFactor = 1.0;
    private const double TonFactor = 1e+6d;
    private const double CentnerFactor = 1e+5d;
    private const double CaratFactor = 0.2;

    private const string InvalidUnitExceptionMessage = "Invalid MassUnit";
    public double Convert(double value, MassUnit from, MassUnit to)
    {
```

Продолжение листинга 5.3.1

```
        if (!Enum.IsDefined(from) && !Enum.IsDefined(to))
        {
            throw new ArgumentException(InvalidUnitExceptionMessage);
        }

        if (value <= 0)
        {
            throw new ArgumentException("Value cannot be negative");
        }
        if (from == to)
        {
            return value;
        }
        return value * UnitFactor(from) / UnitFactor(to);
    }
    private static double UnitFactor(MassUnit unit) => unit switch
    {
        MassUnit.Gram => GramFactor,
        MassUnit.Ton => TonFactor,
        MassUnit.Centner => CentnerFactor,
        MassUnit.Carats => CaratsFactor,
        _ => throw new ArgumentException(InvalidUnitExceptionMessage)
    };
}

public class SiPrefixConverter : IConverter<double, SiPrefixUnit>
{
    private const double NoneFactor = 1;
    private const double DecaFactor = 10;
    private const double HectoFactor = 1e+2d;
    private const double KiloFactor = 1e+3d;
    private const double MegaFactor = 1e+6d;
    private const double GigaFactor = 1e+9d;
    private const double TeraFactor = 1e+12d;
    private const double PetaFactor = 1e+15d;
    private const double ExaFactor = 1e+18d;
    private const double ZettaFactor = 1e+21d;
    private const double YottaFactor = 1e+24d;
    private const double RonnaFactor = 1e+27d;
    private const double QuettaFactor = 1e+30d;
    private const string InvalidUnitExceptionMessage = "Invalid
SiPrefixUnit";

    public double Convert(double value, SiPrefixUnit from, SiPrefixUnit to)
    {
        if (!Enum.IsDefined(from) && !Enum.IsDefined(to))
        {
            throw new ArgumentException(InvalidUnitExceptionMessage);
        }
        if (from == to)
        {
            return value;
        }

        return value * UnitFactor(from) / UnitFactor(to);
    }
    private static double UnitFactor(SiPrefixUnit unit) => unit switch
    {
        SiPrefixUnit.None => NoneFactor,
        SiPrefixUnit.Deca => DecaFactor,
        SiPrefixUnit.Hecto => HectoFactor,
        SiPrefixUnit.Kilo => KiloFactor,
        SiPrefixUnit.Mega => MegaFactor,
```

Продолжение листинга 5.3.1

```
        SiPrefixUnit.Giga => GigaFactor,
        SiPrefixUnit.Tera => TeraFactor,
        SiPrefixUnit.Peta => PetaFactor,
        SiPrefixUnit.Exa => ExaFactor,
        SiPrefixUnit.Zetta => ZettaFactor,
        SiPrefixUnit.Yotta => YottaFactor,
        SiPrefixUnit.Ronna => RonnaFactor,
        SiPrefixUnit.Quetta => QuettaFactor,
        _ => throw new ArgumentException(InvalidUnitExceptionMessage)
    };
}

public class TemperatureConverter : IConverter<double, TemperatureUnit>
{
    private const double KelvinOffset = 273.15;
    private const double ReaumurFactor = 0.8;

    private const string InvalidUnitExceptionMessage = "Invalid
TemperatureUnit";
    public double Convert(double value, TemperatureUnit from, TemperatureUnit
to)
    {
        if (!Enum.IsDefined(from) && !Enum.IsDefined(to))
        {
            throw new ArgumentException(InvalidUnitExceptionMessage);
        }
        if (from == to)
        {
            return value;
        }
        var asCelsius = from switch
        {
            TemperatureUnit.Celsius => value,
            TemperatureUnit.Fahrenheit => FahrenheitToCelsius(value),
            TemperatureUnit.Kelvin => KelvinToCelsius(value),
            TemperatureUnit.Reaumur => ReaumurToCelsius(value),
            _ => throw new ArgumentException(InvalidUnitExceptionMessage)
        };
        return to switch
        {
            TemperatureUnit.Celsius => asCelsius,
            TemperatureUnit.Fahrenheit => CelsiusToFahrenheit(asCelsius),
            TemperatureUnit.Kelvin => CelsiusToKelvin(asCelsius),
            TemperatureUnit.Reaumur => CelsiusToReaumur(asCelsius),
            _ => throw new ArgumentException(InvalidUnitExceptionMessage)
        };
    }
    private double CelsiusToFahrenheit(double value)
        => value * 9 / 5 + 32;
    private double CelsiusToKelvin(double value)
        => value + KelvinOffset;
    private double CelsiusToReaumur(double value)
        => value * ReaumurFactor;
    private double FahrenheitToCelsius(double value)
        => (value - 32) * 5 / 9;
    private double KelvinToCelsius(double value)
        => value - KelvinOffset;
    private double ReaumurToCelsius(double value)
        => value / ReaumurFactor;
}

public class VolumeConverter : IConverter<double, VolumeUnit>
{
    private const double LitreFactor = 1.0;
    private const double MillilitreFactor = 0.001;
```

Продолжение листинга 5.3.1

```
private const double CubicMetreFactor = 1000.0;
private const double CubicCentimetreFactor = 0.001;
private const double GallonFactor = 3.78541;
private const double QuartFactor = 0.946353;
private const double PintFactor = 0.473176;
private const double CupFactor = 0.24;
private const double TablespoonFactor = 0.0147868;
private const double TeaspoonFactor = 0.00492892;
private const double BarrelFactor = 158.987;

private const string InvalidUnitExceptionMessage = "Invalid VolumeUnit";

public double Convert(double value, VolumeUnit from, VolumeUnit to)
{
    if (!Enum.IsDefined(from) && !Enum.IsDefined(to))
    {
        throw new ArgumentException(InvalidUnitExceptionMessage);
    }

    if (value <= 0)
    {
        throw new ArgumentException("Value cannot be negative");
    }
    if (from == to)
    {
        return value;
    }
    return value / UnitFactor(from) * UnitFactor(to);
}

private static double UnitFactor(VolumeUnit unit) => unit switch
{
    VolumeUnit.Litre => LitreFactor,
    VolumeUnit.Millilitre => MillilitreFactor,
    VolumeUnit.CubicMetre => CubicMetreFactor,
    VolumeUnit.CubicCentimetre => CubicCentimetreFactor,
    VolumeUnit.Gallon => GallonFactor,
    VolumeUnit.Quart => QuartFactor,
    VolumeUnit.Pint => PintFactor,
    VolumeUnit.Cup => CupFactor,
    VolumeUnit.Tablespoon => TablespoonFactor,
    VolumeUnit.Teaspoon => TeaspoonFactor,
    VolumeUnit.Barrel => BarrelFactor,
    _ => throw new ArgumentException(InvalidUnitExceptionMessage)
};
}
```

5.3.1 Анализ выживаемости

Выживаемость мутационных тестов продемонстрирована на рисунке

5.3.1.2. Все тесты прошли успешно, а значит ни один мутант не выжил

```
[16:09:09 INF] 19 total mutants will be tested
100,00% | Testing mutant 19 / 19 | K 19 | S 0 | T 0 | ~0m 00s |
Killed: 19
Survived: 0
Timeout: 0
```

Рисунок 5.3.1.2 – Выживаемость мутационных тестов

5.3.2 Корректировка тестов

В ходе выполнения мутационного тестирования все внесенные в код мутанты были успешно выявлены существующими тестами. Корректировка тестовых сценариев не требуется.

ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы были достигнуты поставленные цели по изучению и практическому применению методологий модульного и мутационного тестирования для обеспечения качества программного продукта. Были решены задачи, включающие разработку программных модулей, создание и проведение модульных тестов, анализ тестового покрытия, а также оценку и повышение эффективности тестового набора с помощью мутационного анализа.

На первом этапе работы для разработанных программных модулей были созданы наборы модульных тестов. Данный процесс позволил провести верификацию корректности реализации основных функций, выявить и устранить преднамеренно внесённые дефекты. Анализ покрытия кода продемонстрировал первоначальный уровень охвата функциональности тестами.

Ключевым этапом стало применение мутационного тестирования, которое выступило инструментом для оценки качества созданного тестового набора. Проведение мутационного анализа позволило выявить «выживших мутантов», что указало на недостатки тестов, неспособных обнаружить определённые классы ошибок, несмотря на формально достаточное покрытие кода.

На основе анализа причин выживания мутантов была произведена корректировка и расширение тестового набора путём добавления проверок граничных значений и специфических сценариев использования. Повторное проведение мутационного тестирования подтвердило повышение качества тестов, выраженное в увеличении показателя уничтоженных мутантов.

Таким образом, практическая работа подтвердила, что модульное и мутационное тестирование являются взаимодополняющими практиками. Модульное тестирование обеспечивает базовую проверку функциональности, в то время как мутационное тестирование позволяет дать объективную оценку

надёжности самих тестов. Все задачи, поставленные в рамках работы, были выполнены. Полученные знания и навыки являются фундаментальными для современных подходов к разработке и обеспечению качества программного обеспечения.