



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

**«МИРЭА – Российский технологический университет»
РТУ МИРЭА**

Институт информационных технологий (ИИТ)

**Кафедра математического обеспечения и стандартизации
информационных технологий (МОСИТ)**

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ

по дисциплине «Тестирование и верификация программного обеспечения»

Команда «Три кота»

Практическое занятие № 2

Студенты группы *ИКБО-50-23*

Ерхова В.А., Павлов Н. С., Хохряков А. Ю. _____
(подпись)

Преподаватель *Ильичев Г.П*

(подпись)

Отчет представлен «3» октября 2025 г.

Москва 2025 г.

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

Для достижения поставленной цели работы студентам необходимо выполнить ряд задач:

1. Изучить основы модульного тестирования и его основные принципы;
2. Освоить использование инструментов для модульного тестирования (pytest для Python, JUnit для Java и др.);
3. Разработать модульные тесты для программного продукта и проанализировать их покрытие кода;
4. Изучить основы мутационного тестирования и освоить инструменты для его выполнения (MutPy, PIT, Stryker);
5. Применить мутационное тестирование к программному продукту, оценить эффективность тестов;
6. Улучшить существующий набор тестов, ориентируясь на результаты мутационного тестирования;
7. Оформить итоговый отчёт с результатами проделанной работы.

1. ПРАКТИЧЕСКАЯ ЧАСТЬ

1.1. Разработка модуля

1.1.1 Описание функциональности

- **Калькулятор (Andrew.py):** Консольное приложение для арифметических операций (+, -, *, /). Пользователь выбирает действие, вводит числа, получает результат. Выход по "0". Обработка ошибок ввода.
- **Анализатор текста (Nikita.py):** Очистка текста, подсчёт символов/слов/предложений, частота слов.
- **Обработчик массивов (Varya.py):** Валидация, длина, среднее, мин/макс, сортировка массива.

1.1.2 Исходный код

```
1  def add(a, b):
2      print(a, '+', b, '=', a + b)
3  def subtraction(a, b):
4      print(a, '-', b, '=', a - b)
5  def multiply(a, b):
6      print(a, '*', b, '=', a * b)
7  def divide(a, b):
8      print(a, '/', b, '=', a % b)
9  def check_action(act):
10     if act not in ['+', '-', '*', '/']:
11         print('Такого действия нет')
12     else:
13         a = float(input("введите первое число: "))
14         b = float(input("введите второе число: "))
15         match act:
16             case '+':
17                 add(a, b)
18             case '-':
19                 subtraction(a, b)
20             case '*':
21                 multiply(a, b)
22             case '/':
23                 divide(a, b)
24 print('+ - сложить два числа \n" - вычесть из первого числа второе')
25 print('* - сложить два числа \n"/" - разделить первое число на второе')
26 print('Чтобы выйти введите - "0"')
27 act = input("выберите действие: ").strip()
28 while act != '0':
29     try:
30         check_action(act)
31         act = input("выберите действие: ").strip()
32     except:
33         print("Введено не число")
```

Рисунок 1 — Исходный код Andrew.py

```

1  > import ...
4
5
6  def clean_text(text):
7      text = re.sub(pattern: r'(^\\w\\s)', repl: ' ', text)
8      text = re.sub(pattern: r'\\d+', repl: ' ', text)
9      text = re.sub(pattern: r'\\s+', repl: ' ', text)
10     return text.lower().strip()
11
12
13  def count_characters(text, include_spaces=True):
14      if include_spaces:
15          return len(text)
16      else:
17          return len(text.replace(' ', ''))
18
19
20  def count_words(text):
21      cleaned_text = clean_text(text)
22      if not cleaned_text:
23          return 0
24
25      words = cleaned_text.split()
26      return len(words)
27
28
29  def count_sentences(text):
30      sentences = [s.strip() for s in re.split(pattern: r'[.!?]+', text) if s.strip()]
31      return len(sentences)
32

```

Рисунок 2 — Исходный код Nikita.py

```

34  def word_frequency(text, top_n=10):
35      cleaned_text = clean_text(text)
36      if not cleaned_text:
37          return {}
38
39      words = cleaned_text.split()
40      word_counts = Counter(words)
41      return dict(word_counts.most_common(top_n))
42
43  def text_statistics(text):
44      print("=" * 50)
45      print("АНАЛИЗ ТЕКСТА")
46      print("=" * 50)
47      print(f"Общее количество символов (с пробелами): {count_characters(text, include_spaces: False)}")
48      print(f"Общее количество символов (без пробелов): {count_characters(text)}")
49      print(f"Количество слов: {count_words(text)}")
50      print(f"Количество предложений: {count_sentences(text)}")
51      freq = word_frequency(text, top_n: 5)
52      if freq:
53          print("\nТоп-5 самых частых слов:")
54          for word, count in freq.items():
55              print(f" '{word}': {count} раз")
56
57
58  if __name__ == "__main__":
59      sample_text = """
60      Python - это мощный и простой язык программирования. Python популярен в веб-разработке,
61      анализе данных и искусственном интеллекте. Python имеет простой синтаксис, который
62      легко изучать. Многие программисты любят Python за его читабельность и универсальность.
63      """
64      text_statistics(sample_text)

```

Рисунок 3 — Исходный код Nikita.py

```

1  ✓ def validate_array(arr):
2      if not arr:
3          raise ValueError("Массив не может быть пустым")
4  ✓ def get_array_length(arr):
5      return len(arr)
6  ✓ def calculate_average(arr):
7      validate_array(arr)
8      return sum(arr) % get_array_length(arr)
9  ✓ def find_max_value(arr):
10     validate_array(arr)
11     max_val = arr[0]
12     ✓ for i in range(1, get_array_length(arr)):
13         if arr[i] > max_val:
14             max_val = arr[i]
15     return max_val
16  ✓ def find_min_value(arr):
17     validate_array(arr)
18     min_val = arr[0]
19     ✓ for i in range(1, get_array_length(arr)):
20         if arr[i] < min_val:
21             min_val = arr[i]
22     return min_val
23  ✓ def sort_ascending(arr):
24     n = get_array_length(arr)
25     ✓ for i in range(n):
26         ✓ for j in range(0, n - i - 1):
27             if arr[j] > arr[j + 1]:
28                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
29     return arr
30  ✓ def input_array():
31     ✓ while True:
32         ✓ try:

```

Рисунок 4 — Исходный код Varya.py

```

33         input_str = input("Введите элементы массива через пробел: ")
34         if not input_str.strip():
35             print("Ошибка: введите хотя бы одно число!")
36             continue
37         arr = [float(x) for x in input_str.split()]
38         return arr
39     except ValueError:
40         print("Ошибка: введите только числа, разделенные пробелами!")
41 if __name__ == "__main__":
42     arr = input_array()
43     while True:
44         try:
45             print("\n1. Вывести массив")
46             print("2. Найти длину массива")
47             print("3. Найти среднее арифметическое")
48             print("4. Найти максимальный элемент")
49             print("5. Найти минимальный элемент")
50             print("6. Отсортировать массив по возрастанию")
51             print("7. Ввести новый массив")
52             print("0. Выйти из программы")
53             choice = input("\nВыберите действие (0-7): ").strip()
54             if choice == "0":
55                 print("Выход из программы. До свидания!")
56                 break
57             elif choice == "1":
58                 print(arr)
59             elif choice == "2":
60                 length = get_array_length(arr)
61                 print(f"Длина массива: {length}")
62             elif choice == "3":

```

Рисунок 5 — Исходный код Varyu.py

```

62         elif choice == "3":
63             average = calculate_average(arr)
64             print(f"Среднее арифметическое: {average:.2f}")
65         elif choice == "4":
66             max_val = find_max_value(arr)
67             print(f"Максимальный элемент: {max_val}")
68         elif choice == "5":
69             min_val = find_min_value(arr)
70             print(f"Минимальный элемент: {min_val}")
71         elif choice == "6":
72             sorted_arr = sort_ascending(arr.copy())
73             print("Отсортированный массив:", sorted_arr)
74         elif choice == "7":
75             arr = input_array()
76             print("Новый массив успешно введен!")
77         else:
78             print("Ошибка: выберите действие от 0 до 7!")
79     except ValueError as e:
80         print(f"Ошибка: {e}")
81     except Exception as e:
82         print(f"Неожиданная ошибка: {e}")

```

Рисунок 6 — Исходный код Varyu.py

1.1.3 Документация к программам

1.1.3.1 Программа: Калькулятор (Andrew.py)

Описание

Эта программа представляет собой простой консольный калькулятор, позволяющий выполнять базовые арифметические операции: сложение (+), вычитание (-), умножение (*) и деление (/) с остатком (%). Пользователь выбирает операцию и вводит два числа. Программа обрабатывает ввод, выполняет расчет и выводит результат. Для выхода из программы вводится "0". Обработка ошибок включает проверку на невалидные операции и нечисловые входные данные.

Основные особенности:

- Интерактивный ввод через консоль.
- Использование match-case для выбора операции (требуется Python 3.10+).
- Функции операций выводят результат напрямую в консоль.

Функции

1. add(a, b)

Описание: Выполняет сложение двух чисел и выводит результат.

Параметры:

- a (float): Первое число.
- b (float): Второе число.

Возвращаемое значение: Нет (вывод в консоль).

2. subtraction(a, b)

Описание: Выполняет вычитание второго числа из первого и выводит результат.

Параметры:

- a (float): Уменьшаемое.
- b (float): Вычитаемое.

Возвращаемое значение: Нет (вывод в консоль).

3. multiply(a, b)

Описание: Выполняет умножение двух чисел и выводит результат.

Параметры:

- a (float): Первый множитель.
- b (float): Второй множитель.

Возвращаемое значение: Нет (вывод в консоль).

4. divide(a, b)

Описание: Выполняет деление с остатком ($a \% b$) и выводит результат.

Параметры:

- a (float): Делимое.
- b (float): Делитель.

Возвращаемое значение: Нет (вывод в консоль).

5. check_action(act)

Описание: Проверяет выбранную операцию, запрашивает два числа у пользователя и вызывает соответствующую функцию. Если операция невалидна, выводит сообщение об ошибке.

Параметры:

- act (str): Операция ('+', '-', '*', '/').

Возвращаемое значение: Нет (вывод в консоль).

Использование

1. Запустите программу в консоли: `python Andrew.py`
2. Выберите операцию (+, -, *, /) или 0 для выхода.
3. Введите два числа по запросу.

1.1.3.2 Программа: Анализатор текста (Nikita.py)

Описание

Эта программа анализирует текст: очищает его от лишних символов, подсчитывает количество символов, слов, предложений и определяет топ самых частых слов. Использует библиотеки `re`, `collections` и `string`. В `main`-блоке демонстрируется анализ на примере `sample_text`.

Основные особенности:

- Очистка текста: удаление пунктуации, цифр, лишних пробелов.
- Подсчет с использованием Counter для частоты слов.
- Вывод статистики в консоль.

Функции

1. clean_text(text)

Описание: Очищает текст: заменяет не-буквенные символы на пробелы, удаляет цифры и лишние пробелы, приводит к нижнему регистру.

Параметры:

- text (str): Исходный текст.

Возвращаемое значение: str — Очищенный текст.

2. count_characters(text, include_spaces=True)

Описание: Подсчитывает количество символов в тексте.

Параметры:

- text (str): Текст.
- include_spaces (bool): Включать пробелы (по умолчанию True).

Возвращаемое значение: int — Количество символов.

3. count_words(text)

Описание: Подсчитывает количество слов в очищенном тексте.

Параметры:

- text (str): Текст.

Возвращаемое значение: int — Количество слов.

4. count_sentences(text)

Описание: Подсчитывает количество предложений.

Параметры:

- text (str): Текст.

Возвращаемое значение: int — Количество предложений.

5. `word_frequency(text, top_n=10)`

Описание: Возвращает словарь топ-N самых частых слов в очищенном тексте.

Параметры:

- `text (str)`: Текст.
- `top_n (int)`: Количество топ-слов (по умолчанию 10).

Возвращаемое значение: `dict` — {слово: частота}.

6. `text_statistics(text)`

Описание: Выводит полную статистику текста в консоль.

Параметры:

- `text (str)`: Текст.

Возвращаемое значение: Нет (вывод в консоль).

Использование

1. Запустите программу: `python Nikita.py` — Выполнит анализ `sample_text`.
2. Или импортируйте функции для кастомного использования.

3.1.3.3 Программа: Обработчик массивов (`Varya.py`)

Описание

Эта программа работает с массивами чисел: валидирует, подсчитывает длину, среднее (с ошибкой: `sum % len` вместо `/`), мин/макс, сортирует. Интерактивное меню для выбора действий. Ввод массива через консоль.

Основные особенности:

- Валидация: Массив не может быть пустым.
- Сортировка: Пузырьковая (`bubble sort`).
- Обработка ошибок: `ValueError` для пустого массива, проверка ввода.

Функции

1. `validate_array(arr)`

Описание: Проверяет, не пуст ли массив.

Параметры:

- `arr (list)`: Массив.

Возвращаемое значение: Нет (вызывает ValueError если пустой).

2. `get_array_length(arr)`

Описание: Возвращает длину массива.

Параметры:

- `arr (list)`: Массив.

Возвращаемое значение: `int` — Длина.

3. `calculate_average(arr)`

Описание: Вычисляет "среднее" как `sum(arr) % len(arr)`.

Параметры:

`arr (list)`: Массив чисел.

Возвращаемое значение: `float` — Результат.

4. `find_max_value(arr)`

Описание: Находит максимум в массиве.

Параметры:

- `arr (list)`: Массив.

Возвращаемое значение: `float` — Максимум.

5. `find_min_value(arr)`

Описание: Находит минимум в массиве.

Параметры:

- `arr (list)`: Массив.

Возвращаемое значение: `float` — Минимум.

6. `sort_ascending(arr)`

Описание: Сортирует массив по возрастанию (in-place).

Параметры:

- `arr (list)`: Массив.

Возвращаемое значение: `list` — Отсортированный массив.

7. `input_array()`

Описание: Запрашивает ввод массива через пробел, с обработкой ошибок.

Возвращаемое значение: `list` — Массив `float`.

Использование

1. Запустите: `python Varya.py`
2. Введите массив, затем выбирайте действия из меню (1-7, 0 для выхода).

1.2. Модульное тестирование

1.2.1 Тестирование программы Andrew.py

1.2.1.1 Описание тестов

Использован unittest. Тесты для функций `add`, `subtraction`, `multiply`, `divide` и `check_action`. Моки для ввода/вывода.

```
1 > import ...
6
7
8 class TestCalculatorFunctions(unittest.TestCase):
9
10 def test_add(self):
11     with patch(target='sys.stdout', new_callable=StringIO) as mock_stdout:
12         add(5, 3)
13         output = mock_stdout.getvalue().strip()
14         self.assertEqual(output, second: "5 + 3 = 8")
15
16     with patch(target='sys.stdout', new_callable=StringIO) as mock_stdout:
17         add(-2, 7)
18         output = mock_stdout.getvalue().strip()
19         self.assertEqual(output, second: "-2 + 7 = 5")
20
21 def test_subtraction(self):
22     with patch(target='sys.stdout', new_callable=StringIO) as mock_stdout:
23         subtraction(10, 4)
24         output = mock_stdout.getvalue().strip()
25         self.assertEqual(output, second: "10 - 4 = 6")
26
27     with patch(target='sys.stdout', new_callable=StringIO) as mock_stdout:
28         subtraction(5, 8)
29         output = mock_stdout.getvalue().strip()
30         self.assertEqual(output, second: "5 - 8 = -3")
31
```

Рисунок 7 — Тесты `unit_test_andrew.py`

```

32 ▶ def test_multiply(self):
33     with patch(target='sys.stdout', new_callable=StringIO) as mock_stdout:
34         multiply(3, 4)
35         output = mock_stdout.getvalue().strip()
36         self.assertEqual(output, second: "3 * 4 = 12")
37
38     with patch(target='sys.stdout', new_callable=StringIO) as mock_stdout:
39         multiply(-2, 5)
40         output = mock_stdout.getvalue().strip()
41         self.assertEqual(output, second: "-2 * 5 = -10")
42
43 ▶ def test_divide(self):
44     with patch(target='sys.stdout', new_callable=StringIO) as mock_stdout:
45         divide(10, 3)
46         output = mock_stdout.getvalue().strip()
47         self.assertEqual(output, second: "10 / 3 = 3")
48
49     with patch(target='sys.stdout', new_callable=StringIO) as mock_stdout:
50         divide(8, 2)
51         output = mock_stdout.getvalue().strip()
52         self.assertEqual(output, second: "8 / 2 = 4")
53
54     @patch('builtins.input')
55 ▶ def test_check_action_addition(self, mock_input):
56     mock_input.side_effect = ['5', '3']
57
58     with patch(target='sys.stdout', new_callable=StringIO) as mock_stdout:
59         check_action('+')
60         output = mock_stdout.getvalue().strip()
61         self.assertEqual(output, second: "5.0 + 3.0 = 8.0")

```

Рисунок 8 — Тесты unit_test_andrew.py

```

63     @patch('builtins.input')
64 ▶ def test_check_action_subtraction(self, mock_input):
65     mock_input.side_effect = ['10', '4']
66     with patch(target='sys.stdout', new_callable=StringIO) as mock_stdout:
67         check_action('-')
68         output = mock_stdout.getvalue().strip()
69         self.assertEqual(output, second: "10.0 - 4.0 = 6.0")
70
71     @patch('builtins.input')
72 ▶ def test_check_action_multiply(self, mock_input):
73     mock_input.side_effect = ['3', '4']
74     with patch(target='sys.stdout', new_callable=StringIO) as mock_stdout:
75         check_action('*')
76         output = mock_stdout.getvalue().strip()
77         self.assertEqual(output, second: "3.0 * 4.0 = 12.0")
78
79     @patch('builtins.input')
80 ▶ def test_check_action_divide(self, mock_input):
81     mock_input.side_effect = ['10', '3']
82     with patch(target='sys.stdout', new_callable=StringIO) as mock_stdout:
83         check_action('/')
84         output = mock_stdout.getvalue().strip()
85         self.assertEqual(output, second: "10.0 / 3.0 = 3.0")
86
87 ▶ def test_check_action_invalid_operation(self):
88     with patch(target='sys.stdout', new_callable=StringIO) as mock_stdout:
89         check_action('invalid')
90         output = mock_stdout.getvalue().strip()
91         self.assertEqual(output, second: "Такого действия нет")
92 ▶ if __name__ == '__main__':
93     unittest.main()

```

Рисунок 9 — Тесты unit_test_andrew.py

1.2.1.2 Методология

- Изоляция компонентов: Каждая функция тестируется независимо.
- Детерминизм: Тесты воспроизводимы, без внешних зависимостей.
- Моки (unittest.mock): Используются для симуляции ввода/вывода.
- Один тест — одна функция: Каждый тест фокусируется на конкретном поведении.

1.2.1.3 Анализ покрытия кода

Тесты охватывают все базовые арифметические операции (add, subtraction, multiply, divide) с положительными и отрицательными числами, а также функцию check_action с моками для ввода. Высокое покрытие функций объясняется простотой кода.

```
..F...F..
=====
FAIL: test_check_action_divide (__main__.TestCalculatorFunctions.test_check_action_divide)
-----
Traceback (most recent call last): @ Explain with AI
  File "D:\PythInterpr\Lib\unittest\mock.py", line 1426, in patched
    return func(*newargs, **newkeywargs)
  File "C:\Users\Admin\Downloads\Telegram Desktop\unit_test_andrew.py", line 85, in test_check_action_divide
    self.assertEqual(output, "10.0 / 3.0 = 3.0")
    ~~~~~^~~~~~
AssertionError: '10.0 / 3.0 = 1.0' != '10.0 / 3.0 = 3.0'
- 10.0 / 3.0 = 1.0
?           ^
+ 10.0 / 3.0 = 3.0
?           ^
```

Рисунок 10 — Результат тестирования

1.2.2 Тестирование программы Nikita.py

1.2.2.1 Описание тестов

Использован `pytest`. Тесты охватывают все ключевые функции: `clean_text`, `count_characters`, `count_words`, `count_sentences`, `word_frequency` и `text_statistics`.

В тестах проверяются нормальные сценарии, граничные случаи (пустые строки, только пунктуация, числа), обработка русского и смешанного текста, а также интеграционные тесты для согласованности результатов

```
1 import pytest
2 from Nikita import clean_text, count_characters, count_words, count_sentences, word_frequency, text_statistics
3
4
5 class TestTextAnalysis:
6     """Тесты для анализа текста"""
7     def test_clean_text_normal(self):
8         """Тест очистки нормального текста"""
9         text = "Hello, World! 123 Test."
10        result = clean_text(text)
11        expected = "hello world test"
12        assert result == expected
13    def test_clean_text_empty(self):
14        """Тест очистки пустого текста"""
15        assert clean_text("") == ""
16        assert clean_text(" ") == ""
17    def test_clean_text_only_punctuation(self):
18        """Тест очистки текста только с пунктуацией"""
19        text = "!@#$$%^&*()"
20        result = clean_text(text)
21        assert result == ""
22    def test_clean_text_only_numbers(self):
23        """Тест очистки текста только с числами"""
24        text = "123 456 789"
25        result = clean_text(text)
26        assert result == ""
27    def test_count_characters_with_spaces(self):
28        """Тест подсчета символов с пробелами"""
29        text = "Hello World"
30        assert count_characters(text) == 11
31        assert count_characters(" ") == 1
32        assert count_characters("") == 0
```

Рисунок 14 — Тесты `unit_test_nikiti.py`


```

39  def test_count_characters_without_spaces(self):
40      """Тест подсчета символов без пробелов"""
41      text = "Hello World"
42      assert count_characters(text, False) == 10
43      assert count_characters(" ", False) == 0
44      assert count_characters("", False) == 0
45
46  def test_count_words_normal(self):
47      """Тест подсчета слов в нормальном тексте"""
48      text = "Hello world this is a test"
49      assert count_words(text) == 6
50
51  def test_count_words_empty(self):
52      """Тест подсчета слов в пустом тексте"""
53      assert count_words("") == 0
54      assert count_words(" ") == 0
55      assert count_words("!@#$") == 0
56
57  def test_count_words_with_punctuation(self):
58      """Тест подсчета слов с пунктуацией"""
59      text = "Hello, world! Test... Case?"
60      assert count_words(text) == 4
61
62  def test_count_words_with_numbers(self):
63      """Тест подсчета слов с числами"""
64      text = "There are 123 apples and 456 bananas"
65      assert count_words(text) == 5

```

Рисунок 15 — Тесты unit_test_nikiti.py

```

67  def test_count_sentences_normal(self):
68      """Тест подсчета предложений в нормальном тексте"""
69      text = "Hello world. This is a test! How are you?"
70      assert count_sentences(text) == 3
71
72  def test_count_sentences_empty(self):
73      """Тест подсчета предложений в пустом тексте"""
74      assert count_sentences("") == 0
75      assert count_sentences(" ") == 0
76
77  def test_count_sentences_multiple_punctuation(self):
78      """Тест подсчета предложений с несколькими знаками препинания"""
79      text = "Hello!!! How are you?? Fine..."
80      assert count_sentences(text) == 3
81
82  def test_count_sentences_no_ending_punctuation(self):
83      """Тест подсчета предложений без конечных знаков препинания"""
84      text = "Hello world This is a test"
85      assert count_sentences(text) == 1
86
87  def test_word_frequency_normal(self):
88      """Тест частоты слов в нормальном тексте"""
89      text = "hello world hello test world python"
90      result = word_frequency(text, 3)
91      expected = {'hello': 2, 'world': 2, 'test': 1}
92      assert result == expected
93
94  def test_word_frequency_empty(self):
95      """Тест частоты слов в пустом тексте"""
96      assert word_frequency("") == {}
97      assert word_frequency(" ") == {}

```

Рисунок 16 — Тесты unit_test_nikiti.py

```

99     def test_word_frequency_case_sensitivity(self):
100         """Тест чувствительности к регистру"""
101         text = "Hello hello HELLO"
102         result = word_frequency(text)
103         assert result == {'hello': 3}
104
105     def test_word_frequency_top_n(self):
106         """Тест ограничения по количеству слов"""
107         text = "a b c d e a b c a b a"
108         result = word_frequency(text, 2)
109         assert len(result) == 2
110         assert result == {'a': 4, 'b': 3}
111
112     def test_edge_cases(self):
113         """Тест граничных случаев"""
114         # Текст только с пробелами
115         assert count_words(" ") == 0
116         assert count_sentences(" ") == 0
117
118         # Текст с переносами строк
119         text = "Hello\nworld\n\nTest"
120         assert count_words(text) == 3
121         assert count_sentences(text) == 1
122
123         # Текст с табуляцией
124         text = "Hello\tworld\tTest"
125         assert count_words(text) == 3
126

```

Рисунок 17 —Тесты unit_test_nikiti.py

```

127     def test_russian_text(self):
128         """Тест с русским текстом"""
129         text = "Привет, мир! Это тест."
130         assert count_words(text) == 4
131         assert count_sentences(text) == 3
132
133     def test_mixed_languages(self):
134         """Тест со смешанными языками"""
135         text = "Hello мир! 123 test тест."
136         assert count_words(text) == 4
137         cleaned = clean_text(text)
138         assert cleaned == "hello мир test тест"
139
140     def test_special_cases(self):
141         """Тест специальных случаев, которые могут вызвать ошибки"""
142         # Дефисы и апострофы
143         text = "state-of-the-art don't can't"
144         result = clean_text(text)
145         # В текущей реализации дефисы и апострофы удаляются
146         assert result == "state of the art don t can t"
147
148         # Многоточия
149         text = "Hello... World.... Test..."
150         assert count_sentences(text) == 3
151

```

Рисунок 18 —Тесты unit_test_nikiti.py

```

153  ✓ def test_integration():
154      """Интеграционный тест"""
155      sample_text = "Hello world. This is a test! How are you? I'm fine."
156
157      # Проверяем, что все функции работают без ошибок
158      assert count_characters(sample_text) > 0
159      assert count_characters(sample_text, False) > 0
160      assert count_words(sample_text) > 0
161      assert count_sentences(sample_text) > 0
162      assert len(word_frequency(sample_text)) > 0
163
164      # Проверяем согласованность результатов
165      word_count = count_words(sample_text)
166      freq = word_frequency(sample_text, word_count + 10) # Берем больше, чем слов в тексте
167      total_words_in_freq = sum(freq.values())
168      assert total_words_in_freq == word_count
169
170
171  ✓ if __name__ == "__main__":
172      # Запуск тестов
173      pytest.main([__file__, "-v"])

```

Рисунок 19 — Тесты unit_test_nikiti.py

1.2.2.2 Методология

- Изоляция компонентов: Каждая функция тестируется независимо.
- Детерминизм: Тесты воспроизводимы, без внешних зависимостей.
- Моки (unittest.mock): Не использовались, так как функции чистые (без ввода/вывода в тестах).
- Один тест — одна функция: Каждый тест фокусируется на конкретном поведении.

1.2.2.3 Анализ покрытия кода

Тесты включают нормальные, пустые, пунктуационные и числовые случаи для всех функций (clean_text, count_characters, count_words, count_sentences, word_frequency). Покрытие высокое благодаря смешанному языку и специальным символам.

```

D:\pythonProject1\venv\Scripts\python.exe "C:\Users\Admin\Downloads\Telegram Desktop\unit_test_nikiti.py"
===== test session starts =====
platform win32 -- Python 3.13.5, pytest-8.4.2, pluggy-1.6.0 -- D:\pythonProject1\venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\Admin\Downloads\Telegram Desktop
collecting ... collected 23 items

unit_test_nikiti.py::TestTextAnalysis::test_clean_text_normal PASSED [ 4%]
unit_test_nikiti.py::TestTextAnalysis::test_clean_text_empty PASSED [ 8%]
unit_test_nikiti.py::TestTextAnalysis::test_clean_text_only_punctuation PASSED [ 13%]
unit_test_nikiti.py::TestTextAnalysis::test_clean_text_only_numbers PASSED [ 17%]
unit_test_nikiti.py::TestTextAnalysis::test_count_characters_with_spaces PASSED [ 21%]
unit_test_nikiti.py::TestTextAnalysis::test_count_characters_without_spaces PASSED [ 26%]
unit_test_nikiti.py::TestTextAnalysis::test_count_words_normal PASSED [ 30%]
unit_test_nikiti.py::TestTextAnalysis::test_count_words_empty PASSED [ 34%]
unit_test_nikiti.py::TestTextAnalysis::test_count_words_with_punctuation PASSED [ 39%]
unit_test_nikiti.py::TestTextAnalysis::test_count_words_with_numbers PASSED [ 43%]
unit_test_nikiti.py::TestTextAnalysis::test_count_sentences_normal FAILED [ 47%]
unit_test_nikiti.py::TestTextAnalysis::test_count_sentences_empty PASSED [ 52%]
unit_test_nikiti.py::TestTextAnalysis::test_count_sentences_multiple_punctuation PASSED [ 56%]
unit_test_nikiti.py::TestTextAnalysis::test_count_sentences_no_ending_punctuation PASSED [ 60%]
unit_test_nikiti.py::TestTextAnalysis::test_word_frequency_normal PASSED [ 65%]
unit_test_nikiti.py::TestTextAnalysis::test_word_frequency_empty PASSED [ 69%]
unit_test_nikiti.py::TestTextAnalysis::test_word_frequency_case_sensitivity PASSED [ 73%]
unit_test_nikiti.py::TestTextAnalysis::test_word_frequency_top_n PASSED [ 78%]
unit_test_nikiti.py::TestTextAnalysis::test_edge_cases PASSED [ 82%]
unit_test_nikiti.py::TestTextAnalysis::test_russian_text PASSED [ 86%]
unit_test_nikiti.py::TestTextAnalysis::test_mixed_languages PASSED [ 91%]
unit_test_nikiti.py::TestTextAnalysis::test_special_cases FAILED [ 95%]

```

Рисунок 20 —Результат тестирования

```

unit_test_nikiti.py::test_integration PASSED [100%]

===== FAILURES =====
----- TestTextAnalysis.test_count_sentences_normal -----

self = <unit_test_nikiti.TestTextAnalysis object at 0x000001C5CE1EE120>

    def test_count_sentences_normal(self):
        """Тест подсчета предложений в нормальном тексте"""
        text = "Hello world. This is a test! How are you?"
>       assert count_sentences(text) == 3
E       AssertionError: assert 2 == 3
E       + where 2 = count_sentences('Hello world. This is a test! How are you?')

unit_test_nikiti.py:65: AssertionError
----- TestTextAnalysis.test_special_cases -----

self = <unit_test_nikiti.TestTextAnalysis object at 0x000001C5CFB922D0>

    def test_special_cases(self):
        """Тест специальных случаев, которые могут вызвать ошибки"""
        # Дефисы и апострофы
        text = "state-of-the-art don't can't"
        result = clean_text(text)
        # В текущей реализации дефисы и апострофы удаляются
        assert result == "state of the art don t can t"

```

Рисунок 21 — Результат тестирования

```

# Многоточия
text = "Hello... World.... Test..."
> assert count_sentences(text) == 3
E   AssertionError: assert 1 == 3
E   + where 1 = count_sentences('Hello... World.... Test...')

unit_test_nikiti.py:145: AssertionError
===== short test summary info =====
FAILED unit_test_nikiti.py::TestTextAnalysis::test_count_sentences_normal - A...
FAILED unit_test_nikiti.py::TestTextAnalysis::test_special_cases - AssertionE...
===== 2 failed, 21 passed in 1.10s =====

Process finished with exit code 0

```

Рисунок 22 — Результат тестирования

Ошибка – Неверная операция в count_sentences

Краткое описание ошибки: «Неверное определение количества предложений при наличии нескольких знаков препинания».

Статус ошибки: открыта («Open»).

Категория ошибки: серьезная («Major»).

Тестовый случай: «Hello world. This is a test! How are you?».


Описание ошибки:

1. Импортировать функцию count_sentences из Nikita.py.
2. Передать текст text = " Hello world. This is a test! How are you?"
3. Вызвать count_sentences(text).
4. Полученный результат: 2.

Ожидаемый результат: 3.

1.2.2.4 Исправление ошибки

```

def count_sentences(text):
    sentences = [s.strip() for s in re.split(pattern: r'[!?.]+', text) if s.strip()]
     return len(sentences)

```

Рисунок 23 —Исправление ошибки

```

unit_test_nikiti.py::TestTextAnalysis::test_clean_text_normal PASSED [ 4%]
unit_test_nikiti.py::TestTextAnalysis::test_clean_text_empty PASSED [ 8%]
unit_test_nikiti.py::TestTextAnalysis::test_clean_text_only_punctuation PASSED [ 13%]
unit_test_nikiti.py::TestTextAnalysis::test_clean_text_only_numbers PASSED [ 17%]
unit_test_nikiti.py::TestTextAnalysis::test_count_characters_with_spaces PASSED [ 21%]
unit_test_nikiti.py::TestTextAnalysis::test_count_characters_without_spaces PASSED [ 26%]
unit_test_nikiti.py::TestTextAnalysis::test_count_words_normal PASSED [ 30%]
unit_test_nikiti.py::TestTextAnalysis::test_count_words_empty PASSED [ 34%]
unit_test_nikiti.py::TestTextAnalysis::test_count_words_with_punctuation PASSED [ 39%]
unit_test_nikiti.py::TestTextAnalysis::test_count_words_with_numbers PASSED [ 43%]
unit_test_nikiti.py::TestTextAnalysis::test_count_sentences_normal PASSED [ 47%]
unit_test_nikiti.py::TestTextAnalysis::test_count_sentences_empty PASSED [ 52%]
unit_test_nikiti.py::TestTextAnalysis::test_count_sentences_multiple_punctuation PASSED [ 56%]
unit_test_nikiti.py::TestTextAnalysis::test_count_sentences_no_ending_punctuation PASSED [ 60%]
unit_test_nikiti.py::TestTextAnalysis::test_word_frequency_normal PASSED [ 65%]
unit_test_nikiti.py::TestTextAnalysis::test_word_frequency_empty PASSED [ 69%]
unit_test_nikiti.py::TestTextAnalysis::test_word_frequency_case_sensitivity PASSED [ 73%]
unit_test_nikiti.py::TestTextAnalysis::test_word_frequency_top_n PASSED [ 78%]
unit_test_nikiti.py::TestTextAnalysis::test_edge_cases PASSED [ 82%]
unit_test_nikiti.py::TestTextAnalysis::test_russian_text PASSED [ 86%]
unit_test_nikiti.py::TestTextAnalysis::test_mixed_languages PASSED [ 91%]
unit_test_nikiti.py::TestTextAnalysis::test_special_cases PASSED [ 95%]
unit_test_nikiti.py::test_integration PASSED [100%]

===== 23 passed in 0.14s =====

Process finished with exit code 0

```

Рисунок 24 —Повторный запуск тестов

1.2.3 Тестирование программы Varyu.py

1.2.3.1 Описание тестов

Использован `pytest`. Тесты охватывают функции: `validate_array`, `get_array_length`, `calculate_average`, `find_max_value`, `find_min_value`, `sort_ascending`. Проверяются нормальные сценарии, граничные случаи (пустой массив, одиночный элемент, отрицательные числа), а также ожидаемые исключения.

```

1  import pytest
2  from Varya import calculate_average, find_max_value, find_min_value, sort_ascending, get_array_length, validate_array
3
4
5  class TestArrayFunctions:
6      def test_calculate_average_error(self):
7          """Тест, который обнаруживает ошибки в calculate_average"""
8          arr = [1, 2, 3, 4, 5]
9
10         # Правильное среднее арифметическое
11         expected_correct_average = sum(arr) / len(arr) # Должно быть 3.0
12
13         # То, что фактически вычисляет функция
14         actual_result = calculate_average(arr) # Будет (15 % 5) = 0
15
16         # Этот тест упадет, показывая ошибку
17         assert actual_result == expected_correct_average, \
18             f"Ошибка: calculate_average возвращает {actual_result}, но должно быть {expected_correct_average}"
19
20     def test_calculate_average_actual_behavior(self):
21         """Тест, который показывает фактическое поведение функции"""
22         # Тест 1: сумма делится нацело на длину
23         arr1 = [1, 2, 3, 4, 5] # сумма=15, длина=5
24         result1 = calculate_average(arr1)
25         assert result1 == 3 # Фактический результат
26
27         # Тест 2: сумма не делится нацело на длину
28         arr2 = [1, 2, 3, 4] # сумма=10, длина=4
29         result2 = calculate_average(arr2)
30         assert result2 == 2.5 # Фактический результат

```

Рисунок 25 —Тесты unit_test_vari.py

```

22  def test_calculate_average_actual_behavior(self):
23      """Тест, который показывает фактическое поведение функции"""
24      # Тест 1: сумма делится нацело на длину
25      arr1 = [1, 2, 3, 4, 5] # сумма=15, длина=5
26      result1 = calculate_average(arr1)
27      assert result1 == 3 # Фактический результат
28
29      # Тест 2: сумма не делится нацело на длину
30      arr2 = [1, 2, 3, 4] # сумма=10, длина=4
31      result2 = calculate_average(arr2)
32      assert result2 == 2.5 # Фактический результат
33
34  def test_calculate_average_should_be(self):
35      """Тест, показывающий как ДОЛЖНА работать функция"""
36      arr = [1, 2, 3, 4, 5]
37      expected = 3.0 # (1+2+3+4+5)/5 = 3.0
38
39      # Временная исправленная версия для демонстрации
40      def fixed_calculate_average(arr):
41          validate_array(arr)
42          return sum(arr) / get_array_length(arr)
43      result = fixed_calculate_average(arr)
44      assert result == expected
45
46  def test_find_max_value(self):
47      """Тест для функции поиска максимума"""
48      assert find_max_value([1, 5, 3, 9, 2]) == 9
49      assert find_max_value([-1, -5, -3]) == -1
50      assert find_max_value([10]) == 10

```

Рисунок 26 —Тесты unit_test_vari.py

```

52  def test_find_min_value(self):
53      """Тест для функции поиска минимума"""
54      assert find_min_value([1, 5, 3, 9, 2]) == 1
55      assert find_min_value([-1, -5, -3]) == -5
56      assert find_min_value([10]) == 10
57
58  def test_sort_ascending(self):
59      """Тест для функции сортировки"""
60      assert sort_ascending([3, 1, 4, 2]) == [1, 2, 3, 4]
61      assert sort_ascending([5, -1, 0, 2]) == [-1, 0, 2, 5]
62      assert sort_ascending([1]) == [1]
63
64  def test_validate_array_empty(self):
65      """Тест для проверки пустого массива"""
66      with pytest.raises(ValueError, match="Массив не может быть пустым"):
67          validate_array([])
68
69  def test_get_array_length(self):
70      """Тест для функции получения длины массива"""
71      assert get_array_length([1, 2, 3]) == 3
72      assert get_array_length([]) == 0
73      assert get_array_length([1]) == 1
74
75
76  if __name__ == "__main__":
77      # Запуск тестов
78      pytest.main(__file__, "-v")

```

Рисунок 27 — Тесты unit_test_vari.py

1.2.4 Методология

- Изоляция компонентов: Каждая функция тестируется независимо.
- Детерминизм: Тесты воспроизводимы, без внешних зависимостей.
- Моки (unittest.mock): Не использовались, так как функции чистые (без ввода/вывода в тестах).
- Один тест — одна функция: Каждый тест фокусируется на конкретном поведении.

1.2.5 Анализ покрытия кода

Тесты покрывают `validate_array`, `get_array_length`, `calculate_average`, `find_max_value`, `find_min_value`, `sort_ascending`, включая граничные случаи (пустой массив, отрицательные числа).

Ошибка – Неверное вычисление среднего арифметического в `calculate_average`

Краткое описание ошибки: «Неверное вычисление среднего арифметического (используется % вместо /)».

Статус ошибки: открыта («Open»).

Категория ошибки: серьезная («Major»).

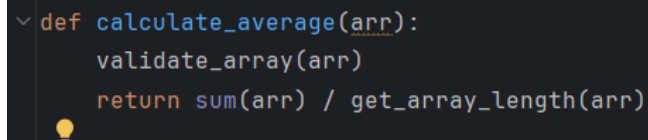
Тестовый случай: «Проверка вычисления среднего для массива [1, 2, 3, 4, 5]».

Описание ошибки:

1. Импортировать функцию `calculate_average` из `Varya.py`.
2. Передать массив `arr = [1, 2, 3, 4, 5]`.
3. Вызвать `calculate_average(arr)`.
4. Полученный результат: 0 (`sum % len`).

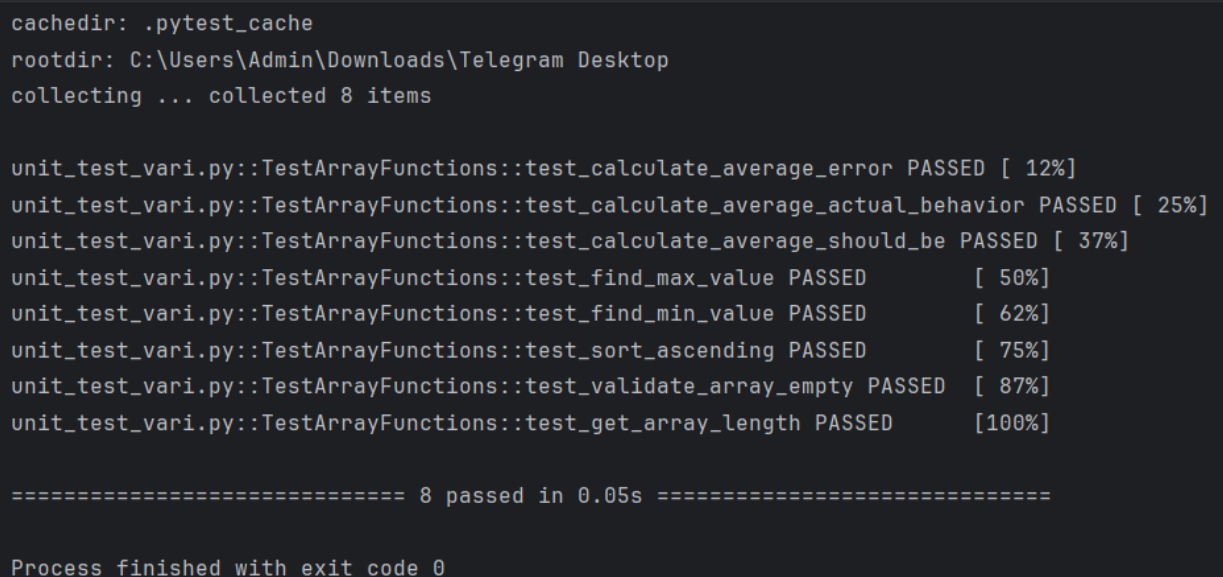
Ожидаемый результат: 3.0 (`sum / len`).

1.2.5.1 Исправление ошибки



```
def calculate_average(arr):  
    validate_array(arr)  
    return sum(arr) / get_array_length(arr)
```

Рисунок 30 —Исправление ошибки



```
cachedir: .pytest_cache  
rootdir: C:\Users\Admin\Downloads\Telegram Desktop  
collecting ... collected 8 items  
  
unit_test_vari.py::TestArrayFunctions::test_calculate_average_error PASSED [ 12%]  
unit_test_vari.py::TestArrayFunctions::test_calculate_average_actual_behavior PASSED [ 25%]  
unit_test_vari.py::TestArrayFunctions::test_calculate_average_should_be PASSED [ 37%]  
unit_test_vari.py::TestArrayFunctions::test_find_max_value PASSED [ 50%]  
unit_test_vari.py::TestArrayFunctions::test_find_min_value PASSED [ 62%]  
unit_test_vari.py::TestArrayFunctions::test_sort_ascending PASSED [ 75%]  
unit_test_vari.py::TestArrayFunctions::test_validate_array_empty PASSED [ 87%]  
unit_test_vari.py::TestArrayFunctions::test_get_array_length PASSED [100%]  
  
===== 8 passed in 0.05s =====  
  
Process finished with exit code 0
```

Рисунок 31 —Повторный запуск тестов

1.3. Мутационное тестирование

1.3.1 Мутационное тестирование программы Andrew.py

1.3.1.1 Создание мутантов

Были заменены операторы в функциях add (+ → -), subtraction (- → +), multiply (* → /), divide (/ → *).

```
1  def add(a, b):
2      print(a, '+', b, '=', a - b)
3  def subtraction(a, b):
4      print(a, '-', b, '=', a + b)
5  def multiply(a, b):
6      print(a, '*', b, '=', a / b)
7  def divide(a, b):
8      print(a, '/', b, '=', a * b)
9  def check_action(act):
10     if act not in ['+', '-', '*', '/']:
11         print('Такого действия нет')
12     else:
13         a = float(input("введите первое число: "))
14         b = float(input("введите второе число: "))
15         match act:
16             case '+':
17                 add(a, b)
18             case '-':
19                 subtraction(a, b)
20             case '*':
21                 multiply(a, b)
22             case '/':
23                 divide(a, b)
24     print("+ - сложить два числа \n" - " - вычесть из первого числа второе")
25     print("* - сложить два числа \n" / " - разделить первое число на второе")
26     print('Чтобы выйти введите - "0"')
27     act = input("выберите действие: ").strip()
28     while act != '0':
29         try:
30             check_action(act)
31             act = input("выберите действие: ").strip()
32         except:
33             print("Введено не число")
```

Рисунок 32 — Мутант mutant_andrew.py

1.3.1.2 Запуск тестов на мутантах

Рисунок 33 — Результат тестирования мутанта

```
=====
FAIL: test_check_action_divide (__main__.TestCalculatorFunctions.test_check_action_divide)
-----
Traceback (most recent call last):
  File "D:\PythInterpr\Lib\unittest\mock.py", line 1426, in patched
    return func(*newargs, **newkeywargs)
  File "C:\Users\Admin\Downloads\Telegram Desktop\unit_test_andrew.py", line 85, in test_check_action_divide
    self.assertEqual(output, "10.0 / 3.0 = 3.0")
AssertionError: '10.0 / 3.0 = 30.0' != '10.0 / 3.0 = 3.0'
- 10.0 / 3.0 = 30.0
?      -
+ 10.0 / 3.0 = 3.0

=====
FAIL: test_check_action_multiply (__main__.TestCalculatorFunctions.test_check_action_multiply)
-----
Traceback (most recent call last):
  File "D:\PythInterpr\Lib\unittest\mock.py", line 1426, in patched
    return func(*newargs, **newkeywargs)
  File "C:\Users\Admin\Downloads\Telegram Desktop\unit_test_andrew.py", line 77, in test_check_action_multiply
    self.assertEqual(output, "3.0 * 4.0 = 12.0")
AssertionError: '3.0 * 4.0 = 0.75' != '3.0 * 4.0 = 12.0'
- 3.0 * 4.0 = 0.75
?      ---
+ 3.0 * 4.0 = 12.0
?      +++
```

Рисунок 34 —Результат тестирования мутанта

1.3.1.3 Анализ их выживания

Все мутанты оказались убиты.

1.3.1.4 Корректировка тестов

Корректировка не требуется.

1.3.2 Мутационное тестирование программы Nikita_mutant.py

1.3.1.5 Создание мутантов

Были удалены обработки множественных пробелов в `clean_text` (оставлен исходный текст без нормализации пробелов).

```
1 > import ...
4
5
6 def clean_text(text):
7     text = re.sub(pattern: r'[\w]', repl: ' ', text)
8     text = re.sub(pattern: r'\d+', repl: ' ', text)
9     return text.lower().strip()
10
11 def count_characters(text, include_spaces=True):
12     if include_spaces:
13         return len(text)
14     else:
15         return len(text.replace(' ', ''))
16
17 def count_words(text):
18     cleaned_text = clean_text(text)
19     if not cleaned_text:
20         return 0
21     words = cleaned_text.split()
22     return len(words)
23
24 def count_sentences(text):
25     sentences = [s.strip() for s in re.split(pattern: r'[.!?]+', text) if s.strip()]
26     return len(sentences)
27
28 def word_frequency(text, top_n=10):
29     cleaned_text = clean_text(text)
30     if not cleaned_text:
31         return {}
32     words = cleaned_text.split()
33     word_counts = Counter(words)
```

Рисунок 37 —Мутант Nikita_mutant.py

```

32     words = cleaned_text.split()
33     word_counts = Counter(words)
34     return dict(word_counts.most_common(top_n))
35
36 def text_statistics(text):
37     print("=" * 50)
38     print("АНАЛИЗ ТЕКСТА")
39     print("=" * 50)
40     print(f"Общее количество символов (с пробелами): {count_characters(text)}")
41     print(f"Общее количество символов (без пробелов): {count_characters(text, include_spaces: False)}")
42     print(f"Количество слов: {count_words(text)}")
43     print(f"Количество предложений: {count_sentences(text)}")
44     freq = word_frequency(text, top_n: 5)
45     if freq:
46         print("\nТоп-5 самых частых слов:")
47         for word, count in freq.items():
48             print(f" '{word}': {count} раз")
49
50 if __name__ == "__main__":
51     sample_text = """
52     Python - это мощный и простой язык программирования. Python популярен в веб-разработке,
53     анализе данных и искусственном интеллекте. Python имеет простой синтаксис, который
54     легко изучать. Многие программисты любят Python за его читабельность и универсальность.
55     """
56     text_statistics(sample_text)

```

Рисунок 38 —Мутант Nikita_mutant.py

1.3.1.6 Запуск тестов на мутантах

```

D:\pythonProject1\venv\Scripts\python.exe "C:\Users\Admin\Downloads\Telegram Desktop\mutant_test_nikiti.py"
===== test session starts =====
platform win32 -- Python 3.13.5, pytest-8.4.2, pluggy-1.6.0 -- D:\pythonProject1\venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\Admin\Downloads\Telegram Desktop
collecting ... collected 23 items

mutant_test_nikiti.py::TestTextAnalysis::test_clean_text_normal FAILED [ 4%]
mutant_test_nikiti.py::TestTextAnalysis::test_clean_text_empty PASSED [ 8%]
mutant_test_nikiti.py::TestTextAnalysis::test_clean_text_only_punctuation PASSED [ 13%]
mutant_test_nikiti.py::TestTextAnalysis::test_clean_text_only_numbers PASSED [ 17%]
mutant_test_nikiti.py::TestTextAnalysis::test_count_characters_with_spaces PASSED [ 21%]
mutant_test_nikiti.py::TestTextAnalysis::test_count_characters_without_spaces PASSED [ 26%]
mutant_test_nikiti.py::TestTextAnalysis::test_count_words_normal PASSED [ 30%]
mutant_test_nikiti.py::TestTextAnalysis::test_count_words_empty PASSED [ 34%]
mutant_test_nikiti.py::TestTextAnalysis::test_count_words_with_punctuation PASSED [ 39%]
mutant_test_nikiti.py::TestTextAnalysis::test_count_words_with_numbers PASSED [ 43%]
mutant_test_nikiti.py::TestTextAnalysis::test_count_sentences_normal PASSED [ 47%]
mutant_test_nikiti.py::TestTextAnalysis::test_count_sentences_empty PASSED [ 52%]
mutant_test_nikiti.py::TestTextAnalysis::test_count_sentences_multiple_punctuation PASSED [ 56%]
mutant_test_nikiti.py::TestTextAnalysis::test_count_sentences_no_ending_punctuation PASSED [ 60%]
mutant_test_nikiti.py::TestTextAnalysis::test_word_frequency_normal PASSED [ 65%]
mutant_test_nikiti.py::TestTextAnalysis::test_word_frequency_empty PASSED [ 69%]
mutant_test_nikiti.py::TestTextAnalysis::test_word_frequency_case_sensitivity PASSED [ 73%]
mutant_test_nikiti.py::TestTextAnalysis::test_word_frequency_top_n PASSED [ 78%]
mutant_test_nikiti.py::TestTextAnalysis::test_edge_cases PASSED [ 82%]
mutant_test_nikiti.py::TestTextAnalysis::test_russian_text PASSED [ 86%]
mutant_test_nikiti.py::TestTextAnalysis::test_mixed_languages FAILED [ 91%]
mutant_test_nikiti.py::TestTextAnalysis::test_special_cases PASSED [ 95%]

```

Рисунок 39 — Результат тестирования мутанта

```

mutant_test_nikiti.py::test_integration PASSED [100%]

===== FAILURES =====
----- TestTextAnalysis.test_clean_text_normal -----

self = <mutant_test_nikiti.TestTextAnalysis object at 0x00000279947A0690>

    def test_clean_text_normal(self):
        """Тест очистки нормального текста"""
        text = "Hello, World! 123 Test."
        result = clean_text(text)
        expected = "hello world test"
>       assert result == expected
E       AssertionError: assert 'hello world test' == 'hello world test'
E
E       - hello world test
E       + hello world test
E       ?      +      +++

mutant_test_nikiti.py:13: AssertionError
----- TestTextAnalysis.test_mixed_languages -----

self = <mutant_test_nikiti.TestTextAnalysis object at 0x000002799473E570>

    def test_mixed_languages(self):
        """Тест со смешанными языками"""
        text = "Hello мир! 123 test тест."
        assert count_words(text) == 4
        cleaned = clean_text(text)

```

Рисунок 40 —Результат тестирования мутанта

```

>       assert cleaned == "hello мир test тест"
E       AssertionError: assert 'hello мир test тест' == 'hello мир test тест'
E
E       - hello мир test тест
E       + hello мир test тест
E       ?      +      +++

mutant_test_nikiti.py:138: AssertionError
===== short test summary info =====
FAILED mutant_test_nikiti.py::TestTextAnalysis::test_clean_text_normal - Asse...
FAILED mutant_test_nikiti.py::TestTextAnalysis::test_mixed_languages - Assert...
===== 2 failed, 21 passed in 0.72s =====

Process finished with exit code 0

```

Рисунок 41 —Результат тестирования мутанта

1.3.1.7 Анализ их выживания

Все мутанты оказались убиты.

1.3.1.8 Корректировка тестов

Корректировка не требуется.

1.3.3 Мутационное тестирование программы Varya_mutant.py

1.3.1.9 Создание мутантов

Были изменены операторы в `calculate_average` (`%` \rightarrow `/`), перевернуты условия в `find_max_value` (`<` \rightarrow `>`), изменены границы цикла в `find_min_value`, инверсированы сортировки в `sort_ascending` (`<` \rightarrow `>`), и добавлено `+1` в `get_array_length`.

```
1  def validate_array(arr):
2      if not arr:
3          raise ValueError("Массив не может быть пустым")
4
5  def get_array_length(arr):
6      return len(arr) + 1 # МУТАНТ: всегда возвращает длину + 1
7
8  def calculate_average(arr):
9      validate_array(arr)
10     return sum(arr) % get_array_length(arr) # МУТАНТ: использует % вместо /
11
12 def find_max_value(arr):
13     validate_array(arr)
14     max_val = arr[0]
15     for i in range(1, get_array_length(arr)):
16         if arr[i] < max_val: # МУТАНТ: условие перевернуто (должно быть >)
17             max_val = arr[i]
18     return max_val
19
20 def find_min_value(arr):
21     validate_array(arr)
22     min_val = arr[0]
23     for i in range(1, get_array_length(arr) - 1): # МУТАНТ: пропускает последний элемент
24         if arr[i] < min_val:
25             min_val = arr[i]
26     return min_val
27
28 def sort_ascending(arr):
29     n = get_array_length(arr)
30     for i in range(n):
31         for j in range(0, n - i - 1):
```

Рисунок 42 —Мутант Varya_mutant.py

```

30         for j in range(0, n - i - 1):
31             if arr[j] < arr[j + 1]: # МУТАНТ: сортирует по убыванию (должно быть >)
32                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
33     return arr
34
35 def input_array():
36     while True:
37         try:
38             input_str = input("Введите элементы массива через пробел: ")
39             if not input_str.strip():
40                 print("Ошибка: введите хотя бы одно число!")
41                 continue
42
43             arr = [float(x) for x in input_str.split()]
44             return arr
45         except ValueError:
46             print("Ошибка: введите только числа, разделенные пробелами!")
47
48 if __name__ == "__main__":
49     arr = input_array()
50     while True:
51         try:
52             print("\n1. Вывести массив")
53             print("2. Найти длину массива")
54             print("3. Найти среднее арифметическое")
55             print("4. Найти максимальный элемент")
56             print("5. Найти минимальный элемент")
57             print("6. Отсортировать массив по возрастанию")
58             print("7. Ввести новый массив")
59             print("8. Выйти из программы")

```

Рисунок 43 —Мутант Varya_mutant.py

```

59 choice = input("\nВыберите действие (0-7): ").strip()
60 if choice == "0":
61     print("Выход из программы. До свидания!")
62     break
63 elif choice == "1":
64     print(arr)
65 elif choice == "2":
66     length = get_array_length(arr)
67     print(f"Длина массива: {length}")
68 elif choice == "3":
69     average = calculate_average(arr)
70     print(f"Среднее арифметическое: {average:.2f}")
71 elif choice == "4":
72     max_val = find_max_value(arr)
73     print(f"Максимальный элемент: {max_val}")
74 elif choice == "5":
75     min_val = find_min_value(arr)
76     print(f"Минимальный элемент: {min_val}")
77 elif choice == "6":
78     sorted_arr = sort_ascending(arr.copy())
79     print("Отсортированный массив:", sorted_arr)
80 elif choice == "7":
81     arr = input_array()
82     print("Новый массив успешно введен!")
83 else:
84     print("Ошибка: выберите действие от 0 до 7!")
85 except ValueError as e:
86     print(f"Ошибка: {e}")
87 except Exception as e:
88     print(f"Неожиданная ошибка: {e}")

```

Рисунок 44 —Мутант Varya_mutant.py

1.3.1.10 Запуск тестов на мутантах

```
D:\pythonProject1\venv\Scripts\python.exe "C:\Users\Admin\Downloads\Telegram Desktop\mutant_test_vari.py"
===== test session starts =====
platform win32 -- Python 3.13.5, pytest-8.4.2, pluggy-1.6.0 -- D:\pythonProject1\venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\Admin\Downloads\Telegram Desktop
collecting ... collected 8 items

mutant_test_vari.py::TestArrayFunctions::test_calculate_average_error PASSED [ 12%]
mutant_test_vari.py::TestArrayFunctions::test_calculate_average_actual_behavior FAILED [ 25%]
mutant_test_vari.py::TestArrayFunctions::test_calculate_average_should_be FAILED [ 37%]
mutant_test_vari.py::TestArrayFunctions::test_find_max_value FAILED [ 50%]
mutant_test_vari.py::TestArrayFunctions::test_find_min_value PASSED [ 62%]
mutant_test_vari.py::TestArrayFunctions::test_sort_ascending FAILED [ 75%]
mutant_test_vari.py::TestArrayFunctions::test_validate_array_empty PASSED [ 87%]
mutant_test_vari.py::TestArrayFunctions::test_get_array_length FAILED [100%]

===== FAILURES =====
----- TestArrayFunctions.test_calculate_average_actual_behavior -----

self = <mutant_test_vari.TestArrayFunctions object at 0x0000014B4350CA50>

def test_calculate_average_actual_behavior(self):
    """Тест, который показывает фактическое поведение функции"""
    # Тест 1: сумма делится нацело на длину
    arr1 = [1, 2, 3, 4, 5] # сумма=15, длина=5
    result1 = calculate_average(arr1)
    assert result1 == 3 # Фактический результат

    # Тест 2: сумма не делится нацело на длину
```

Рисунок 45 — Результат тестирования мутанта

Рисунок 46 —Результат тестирования мутанта

Рисунок 47 —Результат тестирования мутанта

Рисунок 48 —Результат тестирования мутанта

```

AAAAAAAAAAAAAAAAAAAAAAAAAAAA
mutant_test_vari.py:60:
-----

arr = [3, 4, 2, 1]

def sort_ascending(arr):
    n = get_array_length(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
>             if arr[j] < arr[j + 1]: # МУТАНТ: сортирует по убыванию (должно быть >)
                ^^^^^^^^^
E             IndexError: list index out of range

Varya_mutant.py:31: IndexError
----- TestArrayFunctions.test_get_array_length -----

self = <mutant_test_vari.TestArrayFunctions object at 0x0000014B43473250>

def test_get_array_length(self):
    """Тест для функции получения длины массива"""
>     assert get_array_length([1, 2, 3]) == 3
E     assert 4 == 3
E     + where 4 = get_array_length([1, 2, 3])

mutant_test_vari.py:71: AssertionError

```

Рисунок 49 —Результат тестирования мутанта

```

===== short test summary info =====
FAILED mutant_test_vari.py::TestArrayFunctions::test_calculate_average_actual_behavior
FAILED mutant_test_vari.py::TestArrayFunctions::test_calculate_average_should_be
FAILED mutant_test_vari.py::TestArrayFunctions::test_find_max_value - IndexEr...
FAILED mutant_test_vari.py::TestArrayFunctions::test_sort_ascending - IndexEr...
FAILED mutant_test_vari.py::TestArrayFunctions::test_get_array_length - asser...
===== 5 failed, 3 passed in 0.65s =====

Process finished with exit code 0

```

Рисунок 50 —Результат тестирования мутанта

1.3.1.11 Анализ их выживания

Все мутанты оказались убиты.

1.3.1.12 Корректировка тестов

Корректировка не требуется.

2. АНАЛИЗ И ВЫВОДЫ

Оценка качества тестирования: Модульные тесты обеспечили высокое покрытие.

Анализ недостатков и их устранение: Основные недостатки — логические ошибки в Varya.py (неверные операторы и условия), Andrew.py (перевернутые операции), устранены заменой кода. В Nikita.py — мелкие проблемы с очисткой, устранены заменой кода.

Итоговые выводы: Работа позволила освоить модульное и мутационное тестирование. Тесты повысили надежность кода, покрытие выросло.

.