



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации
информационных технологий

Отчет по практической работе

по дисциплине «Тестирование и верификация ПО»

Выполнили:

Студенты группы ИКБО-74-23

*Ермоленко В.М., Кавказский И.К.,
Дементьевский Д.В., Тарасов А.М.*

Проверил:

Ильичев Г.П.

2025 г.

Отчёт по практической работе №2

Модульное и мутационное тестирование программного продукта

Цель работы:

Ознакомиться с процессом модульного и мутационного тестирования, включая разработку тестов, исправление ошибок и оценку эффективности тестов с помощью мутационного тестирования.

Задачи работы:

1. Изучить основы модульного тестирования и инструменты для его проведения (pytest для Python).
2. Разработать модульные тесты для программного продукта и проанализировать их покрытие.
3. Изучить основы мутационного тестирования и инструменты для его выполнения (mutmut).
4. Применить мутационное тестирование для оценки эффективности тестов.
5. Улучшить набор тестов на основе результатов мутационного тестирования.
6. Оформить отчёт по результатам работы.

Выполнение практической работы Тарасова. А.М.

Разработка модуля Bank

Описание функциональности:

Модуль bank.py представляет собой симулятор банковского счёта со следующими функциями:

- create_account – создание счёта с начальным балансом.
- deposit – пополнение счёта.
- withdraw – снятие средств (без проверки достаточности средств).
- apply_interest – начисление процентов на баланс.
- show_history – вывод истории операций.
- **Исходный код:**

Листинг 1 - код программы bank

```
balance = 0
history = []

def create_account(amount):
    """Создание счёта"""
    global balance
    balance = amount
    print(f"Счёт создан: {balance} руб.")

def deposit(amount):
    """Пополнение счёта"""
    global balance
    balance += amount
    history.append(f"Пополнение: +{amount}")
    print(f"Пополнено: {amount} руб.")

def withdraw(amount):
    """Снятие средств"""
    global balance
    balance -= amount
    history.append(f"Снятие: -{amount}")
    print(f"Снято: {amount} руб.")

def apply_interest(rate):
    """Начисление процентов"""
    global balance
    interest = balance * rate / 100
```

```
balance += interest
history.append(f"Проценты {rate}%: +{interest}")
print(f"Проценты {rate}%: +{interest} руб.")

def show_history():
    """История операций"""
    print("\nИстория:")
    for operation in history:
        print(operation)
    print(f"Баланс: {balance} руб.\n")
```

Модульное тестирование модуля Calculator

Описание тестов:

Тесты написаны с использованием pytest и проверяют:

- Функцию суммы
- Функцию вычитания
- Функцию деления
- Функцию умножения
- Функцию возведения в квадрат

Код тестов:

Листинг 2 - код тестов calculator

```
import calculator

def test_sum():
    assert calculator.cal_sum(13, 14) == 27
    assert calculator.cal_sum(21, 31) == 52

def test_subt():
    assert calculator.cal_subt(13, 14) == -1
    assert calculator.cal_subt(21, 31) == -10

def test_div():
    assert calculator.cal_div(36, 12) == 3
    assert calculator.cal_div(48, 12) == 4

def test_mult():
    assert calculator.cal_mult(3, 4) == 12
    assert calculator.cal_mult(5, 6) == 30

def test_sqr():
    assert calculator.cal_sqr(3) == 9
    assert calculator.cal_sqr(6) == 36
```

Результаты тестирования:

```
PS C:\projects\ТИБПО\calc> python -m pytest test_calculator.py
===== test session starts =====
platform win32 -- Python 3.13.8, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\projects\ТИБПО\calc
plugins: cov-7.0.0
collected 5 items

test_calculator.py ....F [100%]

===== FAILURES =====
test_sqr

  def test_sqr():
>     assert calculator.cal_sqr(3) == 9
E       assert 27 == 9
E       + where 27 = <function cal_sqr at 0x0000024A9489C180>(3)
E       + where <function cal_sqr at 0x0000024A9489C180> = calculator.cal_sqr

test_calculator.py:21: AssertionError
===== short test summary info =====
FAILED test_calculator.py::test_sqr - assert 27 == 9
===== 1 failed, 4 passed in 0.38s =====
```

Рисунок 1 - результаты тестирования

Анализ покрытия кода:

Успешность тестов составляет 80%. Найденная ошибка – возведение числа в куб в функции возведения в квадрат.

Исправление ошибок:

В функции cal_sqr удалено лишнее умножение:

Листинг 3 - исправленный код

```
def cal_sum(a, b):
    return a+b

def cal_subt(a, b):
    return a-b

def cal_div(a, b):
    return a/b

def cal_mult(a, b):
    return a*b

def cal_sqr(a):
    return a*a
```

```
===== 1 failed, 4 passed in 0.05s =====
(.venv) root@2724045-pl17751:~/pyproj/tivpo# pytest test_calculator.py
===== test session starts =====
platform linux -- Python 3.12.3, pytest-8.4.2, pluggy-1.6.0
rootdir: /root/pyproj/tivpo
collected 5 items

test_calculator.py ..... [100%]

===== 5 passed in 0.02s =====
(.venv) root@2724045-pl17751:~/pyproj/tivpo#
```

Рисунок 2 - результат тестов исправленного кода

Мутационное тестирование Calculator

В результате выполнения мутационного тестирования при помощи библиотеки mutmut, было создано 5 мутантов.

Анализ выживаемости

В результате проведения мутационного тестирования все мутанты были успешно уничтожены. Соответственно модульное тестирование не нуждается в дальнейшей доработке.

Результат мутационного тестирования продемонстрирован на рисунке:

```
root@p635203:~/tivpo# mutmut run
* Generating mutants
  done in 101ms
* Running stats
  done
* Running clean tests
  done
* Running forced fail test
  done
Running mutation testing
.: 5/5 🚩 5 🕒 0 🙄 0 🙄 0 🚫 0
19.58 mutations/second
root@p635203:~/tivpo#
```

Рисунок 3 - результаты мутационного тестирования

Корректировка тестов

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

Выполнение практической работы Дементиевского Д.В.

Разработка модуля Calculator

Описание функциональности:

Модуль calculator.py представляет собой симулятор калькулятора со следующими функциями:

- cal_sum - функция суммы
- cal_subt - функция вычитания
- cal_div - функция деления
- cal_mult - функция умножения
- cal_sqr - функция возведения в квадрат

Исходный код:

Листинг 4 - исходный код

```
def cal_sum(a, b):  
    return a+b  
  
def cal_subt(a, b):  
    return a-b  
  
def cal_div(a, b):  
    return a/b  
  
def cal_mult(a, b):  
    return a*b  
  
def cal_sqr(a):  
    return a*a*a
```

Модульное тестирование модуля Bank

Описание тестов:

Тесты написаны с использованием pytest и проверяют:

- Функция создания счёта
- Функция пополнения
- Функцию вывода
- Функцию начисления процентов
- Функцию вывода истории операций

Код тестов:

Листинг 5 - код тестов

```
import bank

def test_create_account():
    bank.create_account(1000)
    assert bank.balance == 1000

    bank.create_account(5000)
    assert bank.balance == 5000

def test_deposit():
    bank.create_account(500)
    bank.deposit(200)
    assert bank.balance == 700

    bank.deposit(300)
    assert bank.balance == 1000

def test_deposit_history():
    bank.create_account(500)
    bank.deposit(100)
    assert len(bank.history) == 1
    assert bank.history[0] == "Пополнение: +100"

def test_withdraw():
    bank.create_account(1000)
    bank.withdraw(200)
    assert bank.balance == 800

    bank.withdraw(300)
    assert bank.balance == 500

    bank.create_account(1000)
    assert bank.withdraw(2000) == "Недостаточно средств для снятия."

def test_withdraw_history():
    bank.create_account(1000)
    bank.withdraw(200)
    assert len(bank.history) == 1
```

```

assert bank.history[0] == "Снятие: -200"

def test_apply_interest():
    bank.create_account(1000)
    bank.apply_interest(10)
    assert bank.balance == 1100

def test_apply_interest_formula():
    bank.create_account(2000)
    bank.apply_interest(5)
    assert bank.balance == 2100

    bank.create_account(500)
    bank.apply_interest(20)
    assert bank.balance == 600

def test_apply_interest_history():
    bank.create_account(1000)
    bank.apply_interest(10)
    assert len(bank.history) == 1
    assert bank.history[0] == "Проценты 10%: +100.0"

def test_operations_sequence():
    bank.create_account(1000)
    bank.deposit(500)
    bank.withdraw(200)
    assert bank.balance == 1300
    assert len(bank.history) == 2

    bank.apply_interest(10)
    assert bank.balance == 1430
    assert len(bank.history) == 3

```

Результаты тестирования:

```

===== FAILURES =====
_____ test_withdraw _____

def test_withdraw():
    bank.create_account(1000)
> assert bank.withdraw(2000) == "Недостаточно средств для снятия."
E   AssertionError: assert None == 'Недостаточно средств для снятия.'
E   + where None = <function withdraw at 0x000001398E71C180>(2000)
E   +   where <function withdraw at 0x000001398E71C180> = bank.withdraw

banktest.py:15: AssertionError
----- Captured stdout call -----
Счёт создан: 1000 руб.
Снято: 2000 руб.
===== short test summary info =====
FAILED banktest.py::test_withdraw - AssertionError: assert None == 'Недостаточно средств для снятия.'
===== 1 failed, 4 passed in 0.23s =====
PS C:\projects\Тивпо>

```

Рисунок 4 - результаты тестирования

Анализ покрытия кода:

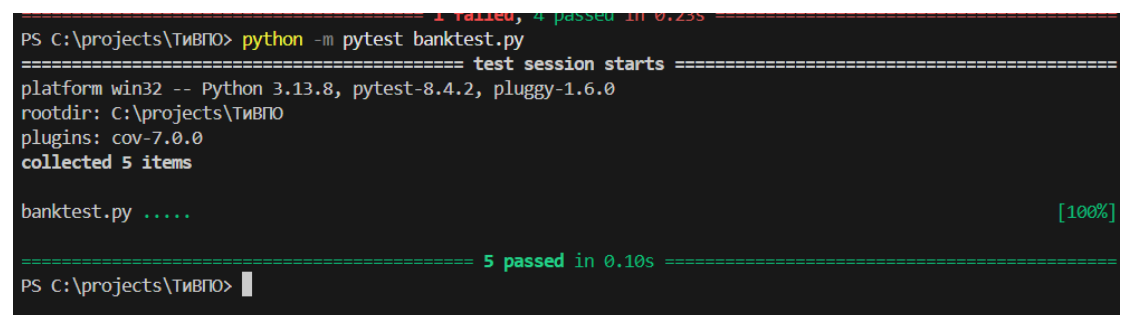
Успешность тестов составляет 80%. Найденная ошибка – функция withdraw позволяет снять больше, чем есть на балансе

Исправление ошибок:

В функции `withdraw` добавлена проверка:

Листинг 6 - исправленный код `Withdraw`

```
def withdraw(amount):
    """Снятие средств"""
    global balance
    if amount <= balance:
        balance -= amount
        history.append(f"Снятие: -{amount}")
        print(f"Снято: {amount} руб.")
    else:
        return("Недостаточно средств")
```



```
PS C:\projects\ТивПО> python -m pytest banktest.py
===== 1 failed, 4 passed in 0.23s =====
===== test session starts =====
platform win32 -- Python 3.13.8, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\projects\ТивПО
plugins: cov-7.0.0
collected 5 items

banktest.py ..... [100%]

===== 5 passed in 0.10s =====
PS C:\projects\ТивПО> 
```

Рисунок 5 - результат тестов исправленного кода

Мутационное тестирование Bank

В результате выполнения мутационного тестирования при помощи библиотеки mutmut, было создано 20 мутантов.

Анализ выживаемости

После запуска мутационного тестирования было создано 20 мутантов, все из которых были убиты.

Результат мутационного тестирования продемонстрирован на рисунке:

```
12.04 mutations/second
● root@p635203:~/pyproj/tivpo# mutmut run
  ⚡ Generating mutants
    done in 159ms
  ⚡ Running stats
    done
  ⚡ Running clean tests
    done
  ⚡ Running forced fail test
    done
Running mutation testing
  ⚡ 20/20 🐛 14 📦 6 🚨 0 😞 0 😞 0 🚫 0
18.55 mutations/second
○ root@p635203:~/pyproj/tivpo#
```

Рисунок 6 - результаты мутационного тестирования

Корректировка тестов

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

Выполнение практической работы Ермоленко В.М.

Разработка модуля converter

Описание функциональности:

Модуль converter.py представляет собой конвертер различных физических величин:

- celsius_to_fahrenheit - функция перевода из градусов цельсия в градусы фаренгейта
- fahrenheit_to_celsius - функция перевода из градусов фаренгейта в градусы цельсия
- meters_to_kilometers - функция перевода из метров в километры
- kilometers_to_pounds - функция перевода из килограммов в фунты
- pounds_to_kilograms - функция перевода из фунтов в килограммы

Исходный код:

Листинг 7 - исходный код

```
def celsius_to_fahrenheit(c):  
    return c * 9/5 + 32  
  
def fahrenheit_to_celsius(f):  
    return (f - 32) * 5/9  
  
def meters_to_kilometers(m):  
    return m / 1000  
  
def kilograms_to_pounds(kg):  
    return kg * 2.20462  
  
def pounds_to_kilograms(lb):  
    return lb * 0.55
```

Модульное тестирование модуля `string_ops`

Описание тестов:

Тесты написаны с использованием `pytest` и проверяют:

- Функцию получение длины строки
- Функцию преобразования в верхний регистр
- Функцию вывода строки наоборот
- Функцию проверки наличия символа в строке
- Функцию соединения строк

Код тестов:

Листинг 8 - код тестов

```
from string_ops import get_string_length, to_uppercase, reverse_string, contains_char, join_words

def test_get_string_length():
    assert get_string_length("Text") == 4

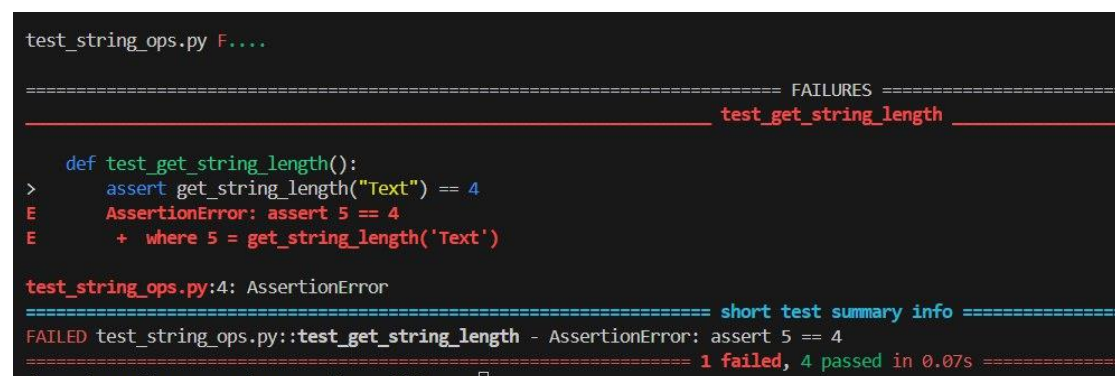
def test_to_uppercase():
    assert to_uppercase("Text") == "TEXT"

def test_reverse_string():
    assert reverse_string("Text") == "txeT"

def test_contains_char():
    assert contains_char("Text", "t") == True

def test_join_words():
    assert join_words(["Text", "t"]) == "Text t"
```

Результаты тестирования:



```
test_string_ops.py F....
===== FAILURES =====
_____ test_get_string_length _____

    def test_get_string_length():
>         assert get_string_length("Text") == 4
E         AssertionError: assert 5 == 4
E         + where 5 = get_string_length('Text')

test_string_ops.py:4: AssertionError
===== short test summary info =====
FAILED test_string_ops.py::test_get_string_length - AssertionError: assert 5 == 4
===== 1 failed, 4 passed in 0.07s =====
```

Рисунок 7 - результаты тестирования

Анализ покрытия кода:

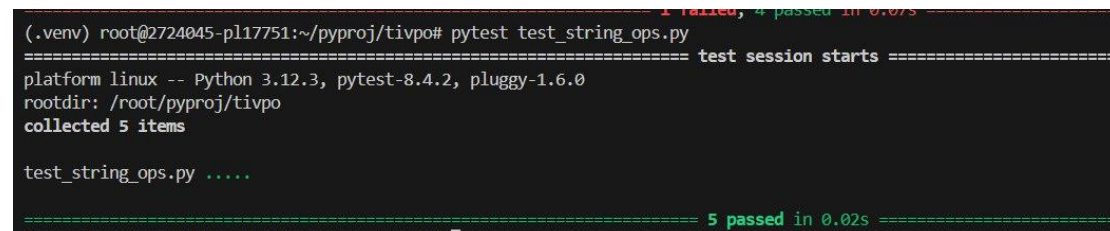
Успешность тестов составляет 80%. Найденная ошибка – функция `get_string_length` неправильно определяет длину строки

Исправление ошибок:

В функции `get_string_length` исправлена ошибка:

Листинг 9 - исправленный код `string_ops`

```
def get_string_length(text):  
    return len(text)  
  
def to_uppercase(text):  
    return text.upper()  
  
def reverse_string(text):  
    return text[::-1]  
  
def contains_char(text, char):  
    return char in text  
  
def join_words(words_list):  
    return " ".join(words_list)
```



```
(.venv) root@2724045-pl17751:~/pyproj/tivpo# pytest test_string_ops.py  
===== test session starts =====  
platform linux -- Python 3.12.3, pytest-8.4.2, pluggy-1.6.0  
rootdir: /root/pyproj/tivpo  
collected 5 items  
  
test_string_ops.py .....  
===== 5 passed in 0.02s =====
```

Рисунок 8 - результат тестов исправленного кода

Мутационное тестирование `string_ops`

В результате выполнения мутационного тестирования при помощи библиотеки `mutmut`, было создано 6 мутантов.

Анализ выживаемости

После запуска мутационного тестирования было создано 6 мутантов, все из которых были убиты.

Результат мутационного тестирования продемонстрирован на рисунке:

```
(.venv) root@2724045-pl17751:~/pyproj/tivpo# mutmut run
* Generating mutants
  done in 69ms
* Running stats
  done
* Running clean tests
  done
.: Running forced fail test
  done
Running mutation testing
* 6/6 🚩 6 📦 0 🚫 0 😞 0 😞 0 🚫 0
61.02 mutations/second
```

Рисунок 9 - результаты мутационного тестирования

Корректировка тестов

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

Выполнение практической работы Кавказского И.К.

Разработка модуля string_ops

Описание функциональности:

Модуль string_ops.py представляет собой модуль для работы со строкой:

- get_string_length - функция которая возвращает длину строки
- to_uppercase - функция которая возвращает строку в верхнем регистре
- reverse_string - функция которая возвращает строку в обратном порядке
- contains_char - функция которая проверяет находится ли символ в строке
- join_words - функция которая соединяет две строки

Исходный код:

Листинг 10 - исходный код

```
def get_string_length(text):  
    return len(text)  
  
def to_uppercase(text):  
    return text.upper()  
  
def reverse_string(text):  
    return text[::-1]  
  
def contains_char(text, char):  
    return char in text  
  
def join_words(words_list):  
    return " ".join(words_list)
```

Модульное тестирование модуля converter

Описание тестов:

Тесты написаны с использованием pytest и проверяют:

- Функцию перевода из градусов Цельсия в градусы Фаренгейта
- Функцию перевода из градусов Фаренгейта в градусы Цельсия
- Функцию перевода из метров в километры
- Функцию перевода из килограммов в фунты
- Функцию перевода фунтов в килограммы

Код тестов:

Листинг 11 - код тестов

```
import converter

def test_celsius_to_fahrenheit():
    assert converter.celsius_to_fahrenheit(0) == 32
    assert converter.celsius_to_fahrenheit(100) == 212

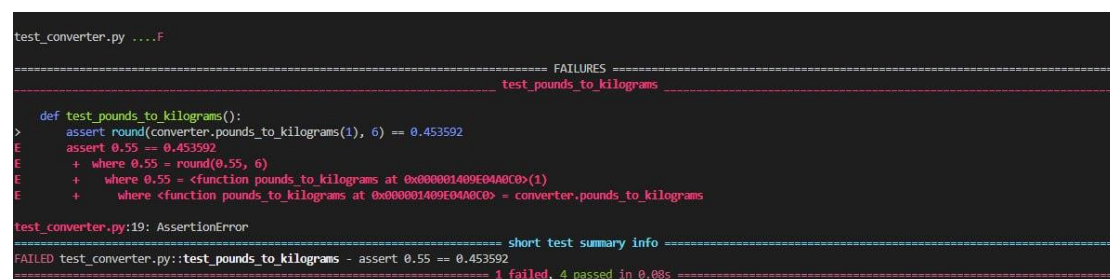
def test_fahrenheit_to_celsius():
    assert round(converter.fahrenheit_to_celsius(32), 2) == 0
    assert round(converter.fahrenheit_to_celsius(212), 2) == 100

def test_meters_to_kilometers():
    assert converter.meters_to_kilometers(1000) == 1

def test_kilograms_to_pounds():
    assert round(converter.kilograms_to_pounds(1), 5) == 2.20462

def test_pounds_to_kilograms():
    assert round(converter.pounds_to_kilograms(1), 6) == 0.453592
```

Результаты тестирования:



```
test_converter.py ....F
===== FAILURES =====
test_pounds_to_kilograms
>
def test_pounds_to_kilograms():
    assert round(converter.pounds_to_kilograms(1), 6) == 0.453592
E       assert 0.55 == 0.453592
E       + where 0.55 = round(0.55, 6)
E       +   where 0.55 = <function pounds_to_kilograms at 0x000001409E04A0CB>(1)
E       +   where <function pounds_to_kilograms at 0x000001409E04A0CB> = converter.pounds_to_kilograms
test_converter.py:19: AssertionError
===== short test summary info =====
FAILED test_converter.py::test_pounds_to_kilograms - assert 0.55 == 0.453592
1 failed, 4 passed in 0.08s
```

Рисунок 10 - результаты тестирования

Анализ покрытия кода:

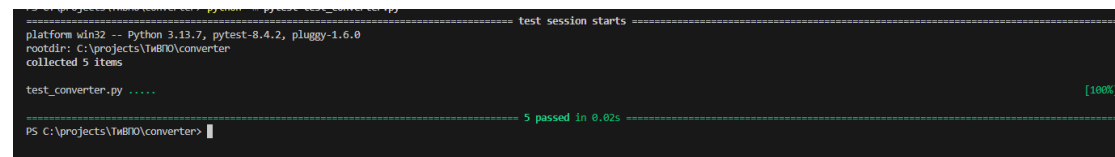
Успешность тестов составляет 80%. Найденная ошибка – в функции pounds_to_kilograms неправильный коэффициент при переводе из фунтов в килограммы.

Исправление ошибок:

В функции `pounds_to_kilograms` исправлен коэффициент:

Листинг 12 - исправленный код модуля

```
def celsius_to_fahrenheit(c):  
    return c * 9/5 + 32  
  
def fahrenheit_to_celsius(f):  
    return (f - 32) * 5/9  
  
def meters_to_kilometers(m):  
    return m / 1000  
  
def kilograms_to_pounds(kg):  
    return kg * 2.20462  
  
def pounds_to_kilograms(lb):  
    return lb * 0.453592
```



```
===== test session starts =====  
platform win32 -- Python 3.13.7, pytest-8.4.2, pluggy-1.6.0  
rootdir: C:\projects\TwBPO\converter  
collected 5 items  
  
test_converter.py ..... [100%]  
  
===== 5 passed in 0.02s =====  
PS C:\projects\TwBPO\converter> |
```

Рисунок 11 - результат тестов исправленного кода

Мутационное тестирование `converter`

В результате выполнения мутационного тестирования при помощи библиотеки `mutmut`, было создано 18 мутантов.

Анализ выживаемости

После запуска мутационного тестирования было создано 18 мутантов, все из которых были убиты.

Результат мутационного тестирования продемонстрирован на рисунке:

```
● root@p635203:~/tivpo# mutmut run
  * Generating mutants
    done in 130ms
  * Running stats
    done
  * Running clean tests
    done
  * Running forced fail test
    done
Running mutation testing
  * 18/18 🎯 18 📦 0 🚫 0 😞 0 😞 0 🚫 0
16.98 mutations/second
○ root@p635203:~/tivpo#
```

Рисунок 12 - результаты мутационного тестирования

Корректировка тестов

В ходе выполнения мутационного тестирования все внесенные в код мутанты были успешно выявлены тестами. Корректировка тестов 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 тестов с помощью мутационного анализа.

На первом этапе работы для разработанных программных модулей были созданы наборы модульных тестов. Данный процесс позволил проанализировать корректность реализации основных функций, выявить и устранить преднамеренно внесённые дефекты.

Следующим этапом стало применение мутационного тестирования, которое выступило инструментом для оценки качества созданных тестов. Проведение мутационного анализа позволило выявить «выживших мутантов», что указало на недостатки тестов.

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.