|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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 группыИКБО-74-23 | *Ермоленко В.М., Кавказский И.К., Дементиевский 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:**

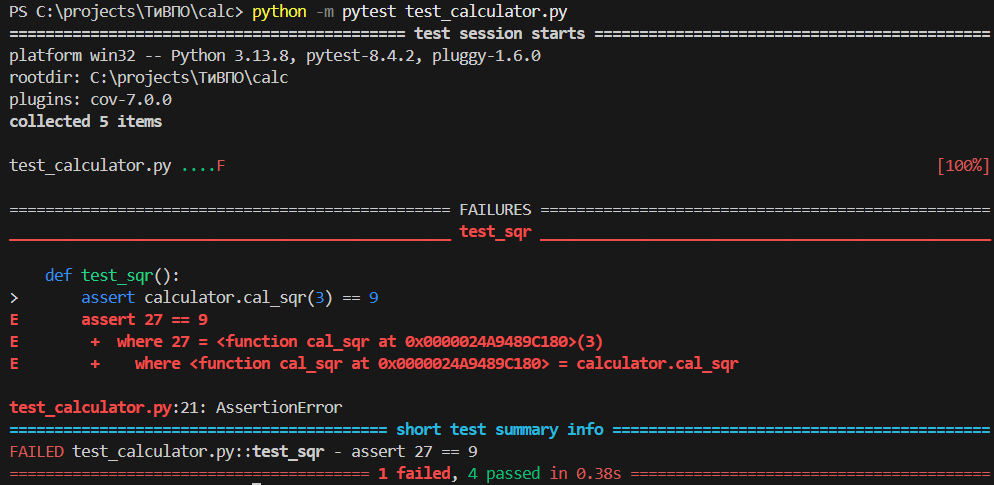


Рисунок 1 - результаты тестирования

**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 |

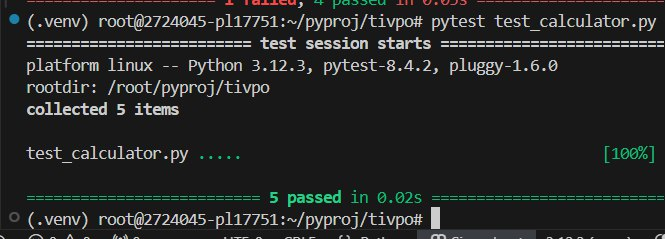


Рисунок 2 - результат тестов исправленного кода

### 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:**

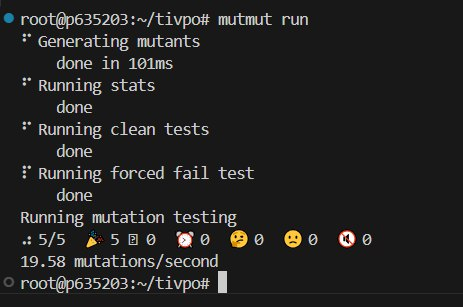


Рисунок 3 - результаты мутационного тестирования

**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:**

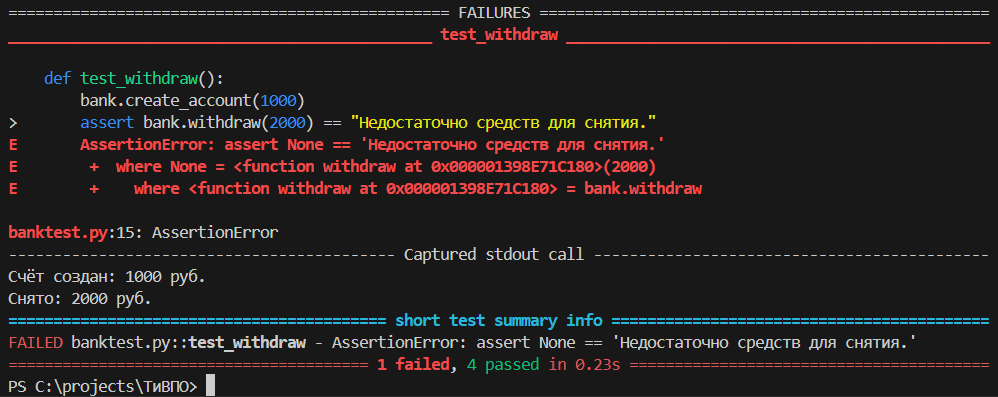


Рисунок 4 - результаты тестирования

**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") |

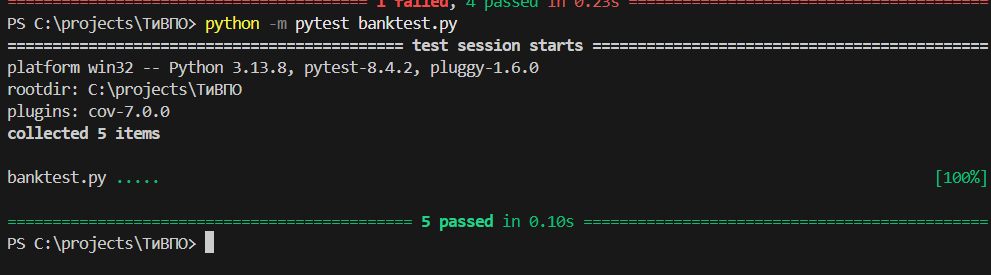


Рисунок 5 - результат тестов исправленного кода

### 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:**

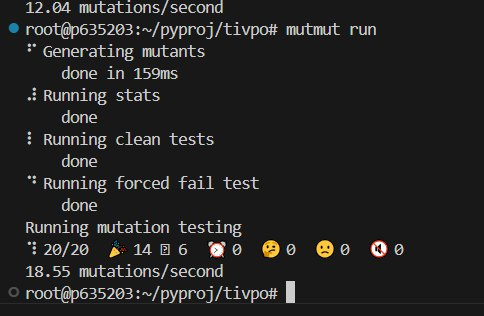


Рисунок 6 - результаты мутационного тестирования

**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:**

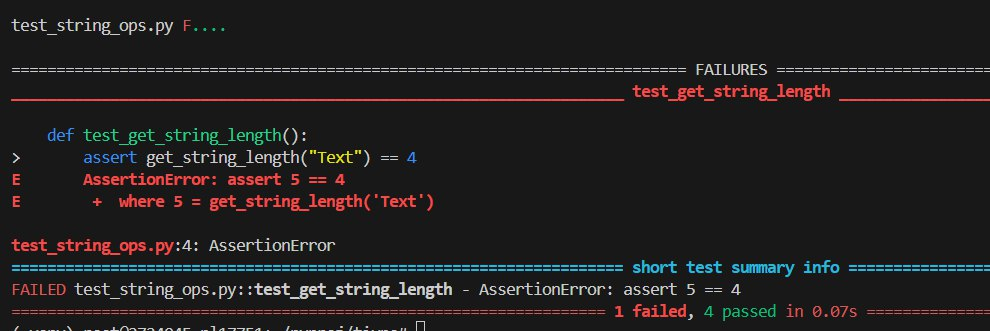


Рисунок 7 - результаты тестирования

**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) |

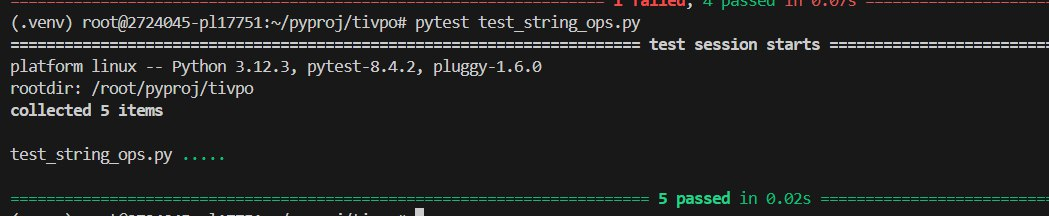


Рисунок 8 - результат тестов исправленного кода

### 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:**

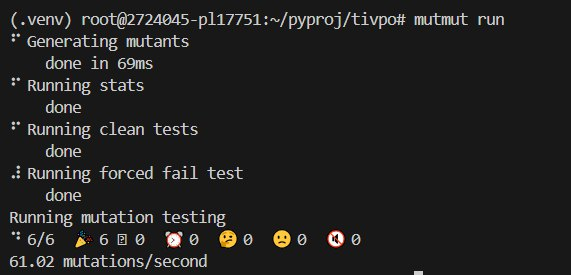


Рисунок 9 - результаты мутационного тестирования

**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:**

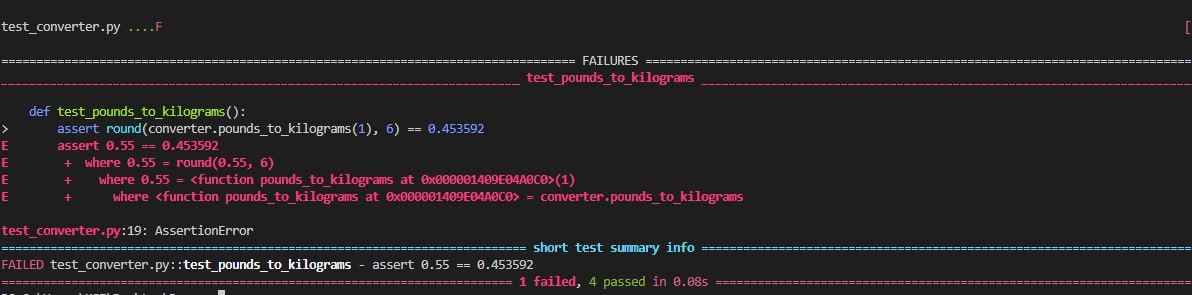


Рисунок 10 - результаты тестирования

**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 |

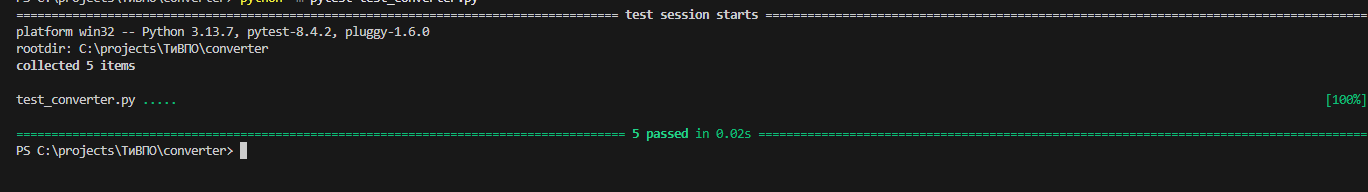


Рисунок 11 - результат тестов исправленного кода

### 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:**

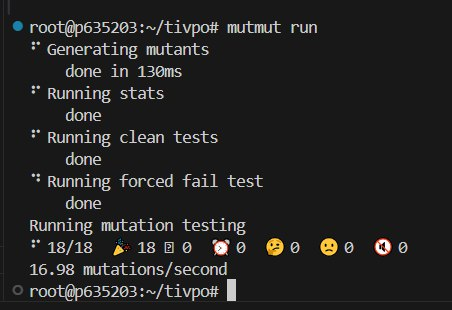


Рисунок 12 - результаты мутационного тестирования

**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.