



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»

Практическое занятие № 2

Студентки группы *ИКБО-50-23, Зернова К.А. Мудрак И. А.*

(подпись)

Преподаватель *Ильичев Г.П*

(подпись)

Отчет представлен « » сентября 2025 г.

Москва 2025 г.

Оглавление

Цель и задачи практической работы	3
Часть 1. Модульное тестирование	4
Часть 2. Мутационное тестирование.....	15
Вывод	18

Цель и задачи практической работы

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

Для достижения поставленной цели работы студентам необходимо выполнить ряд задач:

- изучить основы модульного тестирования и его основные принципы;
- освоить использование инструментов для модульного тестирования;
- разработать модульные тесты для программного продукта и проанализировать их покрытие кода;
- изучить основы мутационного тестирования и освоить инструменты для его выполнения (MutPy, PIT, Stryker);
- применить мутационное тестирование к программному продукту,
- оценить эффективность тестов;
- улучшить существующий набор тестов, ориентируясь на результаты мутационного тестирования;
- оформить итоговый отчёт с результатами проделанной работы.

Часть 1. Модульное тестирование

Разработка программного модуля

1. Калькулятор

Консольное приложение-калькулятор на Node.js с поддержкой основных математических операций. Калькулятор выполняет сложение, вычитание, умножение, деление, возведение в степень и вычисление факториала.

Приложение предоставляет интерактивное меню для выбора операций. Для бинарных операций требуется ввод двух чисел, для факториала - одного числа. Реализована валидация вводимых данных и обработка математических ошибок, таких как деление на ноль или вычисление факториала отрицательного числа.

После выполнения каждой операции приложение ожидает нажатия Enter для возврата в главное меню. Интерфейс на русском языке с четкими инструкциями для пользователя.

Листинг 1 - Основной класс приложения CalculatorApp

```
const readline = require('readline');
const Calculator = require('./calculator');

class CalculatorApp {
  constructor() {
    this.calculator = new Calculator();
    this.rl = readline.createInterface({
      input: process.stdin,
      output: process.stdout
    });
  }

  displayMenu() {
    console.log('\n=== КАЛЬКУЛЯТОР ===');
    console.log('1. Сложение (+)');
    console.log('2. Вычитание (-)');
    console.log('3. Умножение (*)');
    console.log('4. Деление (/)');
    console.log('5. Возведение в степень (^)');
    console.log('6. Факториал (!)');
    console.log('0. Выход');
    console.log('=====');
  }

  async getNumberInput(promptText) {
    return new Promise((resolve) => {
      this.rl.question(promptText, (input) => {
        const number = parseFloat(input);
        if (isNaN(number)) {
          console.log('Ошибка: Введите корректное число');
        }
        resolve(number);
      });
    });
  }
}
```

```

        resolve(this.getNumberInput(promptText));
    } else {
        resolve(number);
    }
    });
});
}

async performOperation(choice) {
    try {
        let result;

        switch (choice) {
            case '1':
                const a1 = await this.getNumberInput('Введите первое
число: ');
                const b1 = await this.getNumberInput('Введите второе
число: ');
                result = this.calculator.add(a1, b1);
                console.log(`Результат: ${a1} + ${b1} = ${result}`);
                break;

            case '2':
                const a2 = await this.getNumberInput('Введите первое
число: ');
                const b2 = await this.getNumberInput('Введите второе
число: ');
                result = this.calculator.subtract(a2, b2);
                console.log(`Результат: ${a2} - ${b2} = ${result}`);
                break;

            case '3':
                const a3 = await this.getNumberInput('Введите первое
число: ');
                const b3 = await this.getNumberInput('Введите второе
число: ');
                result = this.calculator.multiply(a3, b3);
                console.log(`Результат: ${a3} * ${b3} = ${result}`);
                break;

            case '4':
                const a4 = await this.getNumberInput('Введите первое
число: ');
                const b4 = await this.getNumberInput('Введите второе
число: ');
                result = this.calculator.divide(a4, b4);
                console.log(`Результат: ${a4} / ${b4} = ${result}`);
                break;

            case '5':
                const base = await this.getNumberInput('Введите
основание: ');
                const exponent = await this.getNumberInput('Введите
степень: ');
                result = this.calculator.power(base, exponent);
                console.log(`Результат: ${base}^${exponent} =
${result}`);
                break;

            case '6':
                const n = await this.getNumberInput('Введите число для
вычисления факториала: ');
                result = this.calculator.factorial(n);
                console.log(`Результат: ${n}! = ${result}`);

```

```

        break;

        default:
            console.log('Неверный выбор операции');
    }
} catch (error) {
    console.log(`Ошибка: ${error.message}`);
}
}

async run() {
    console.log('Добро пожаловать в калькулятор!');

    while (true) {
        this.displayMenu();

        const choice = await new Promise((resolve) => {
            this.rl.question('Выберите операцию (0-6): ', resolve);
        });

        if (choice === '0') {
            console.log('До свидания!');
            this.rl.close();
            break;
        }

        if (['1', '2', '3', '4', '5', '6'].includes(choice)) {
            console.log(1)
            await this.performOperation(choice);
        } else {
            console.log('Неверный выбор. Попробуйте снова.');
        }

        await new Promise((resolve) => {
            this.rl.question('\nНажмите Enter для продолжения...',
resolve);
        });
    }
}

// Запуск приложения
if (require.main === module) {
    const app = new CalculatorApp();
    app.run();
}

module.exports = CalculatorApp;

```

Листинг 2 - Класс Calculator с математическими операциями

```

class Calculator {
    add(a, b) {
        if (typeof a !== 'number' || typeof b !== 'number') {
            throw new Error('Оба аргумента должны быть числами');
        }
        return a + b;
    }

    subtract(a, b) {
        if (typeof a !== 'number' || typeof b !== 'number') {
            throw new Error('Оба аргумента должны быть числами');
        }
        return a - b;
    }
}

```

```

    }

    multiply(a, b) {
      if (typeof a !== 'number' || typeof b !== 'number') {
        throw new Error('Оба аргумента должны быть числами');
      }
      return a * b;
    }

    divide(a, b) {
      if (typeof a !== 'number' || typeof b !== 'number') {
        throw new Error('Оба аргумента должны быть числами');
      }
      return a / b;
    }

    power(base, exponent) {
      if (typeof base !== 'number' || typeof exponent !== 'number') {
        throw new Error('Оба аргумента должны быть числами');
      }
      return Math.pow(base, exponent);
    }

    factorial(n) {
      if (typeof n !== 'number') {
        throw new Error('Аргумент должен быть числом');
      }
      if (n < 0) {
        throw new Error('Факториал определен только для неотрицательных чисел');
      }
      if (!Number.isInteger(n)) {
        throw new Error('Факториал определен только для целых чисел');
      }

      if (n === 0 || n === 1) {
        return 1;
      }

      let result = 1;
      for (let i = 2; i <= n; i++) {
        result = i * result;
      }
      return result;
    }
  }

  module.exports = Calculator;

```

2. Фигуры

Библиотека для вычисления параметров геометрических фигур.

Предоставляет функциональность для работы с кругами и треугольниками, включая вычисление основных характеристик и специальных свойств.

Библиотека позволяет создавать круги и вычислять их основные параметры: площадь, длину окружности и диаметр. Дополнительно

поддерживается вычисление площади сектора круга по заданному углу и площади сегмента по длине хорды. Все вычисления автоматически проверяют корректность входных данных - радиус должен быть положительным, угол сектора должен быть в пределах от 0 до 360 градусов, а длина хорды не может превышать диаметр.

Для треугольников реализовано вычисление площади по формуле Герона и периметра. Библиотека автоматически проверяет возможность существования треугольника с заданными сторонами - все стороны должны быть положительными и удовлетворять неравенству треугольника. Дополнительно определяются специальные свойства треугольника: прямоугольность (проверка по теореме Пифагора), равносторонность и равнобедренность.

Все фигуры наследуются от абстрактного базового класса, что гарантирует единообразие интерфейса и возможность расширения библиотеки новыми фигурами. Каждая фигура обязательно реализует методы вычисления площади и периметра.

Листинг 3 – circle

```
import math
from figure import Figure

class Circle(Figure):
    def __init__(self, radius):
        self.radius = radius
        self._validate_radius()

    def _validate_radius(self):
        """Валидация радиуса с отдельными проверками."""
        Раздельные проверки вместо одного условия
        is_positive = self.radius > 0

        if not is_positive:
            raise ValueError("radius must be greater than 0")

    def area(self):
        """Вычисление площади круга."""
        Разбиваем вычисление на шаги
        radius_squared = self.radius * self.radius
        area_value = math.pi * radius_squared
        return area_value

    def perimeter(self):
```



```

        """Вычисление длины окружности."""
        Разбиваем вычисление на шаги
        circumference_value = 2 * math.pi * self.radius
        return circumference_value

    def diameter(self):
        """Вычисление диаметра."""
        return 2 * self.radius

```

Листинг 4 – figure

```

from abc import ABC, abstractmethod
class Figure(ABC):

    @abstractmethod
    def area(self) -> float:
        pass

    @abstractmethod
    def perimeter(self) -> float:
        pass

```

Листинг 5 – triangle

```

import math
from figure import Figure

class Triangle(Figure):
    def __init__(self, s1: float, s2: float, s3: float) -> None:
        self.side1 = s1
        self.side2 = s2
        self.side3 = s3
        self._validate_sides()

    def _validate_sides(self):
        """Валидация сторон треугольника с отдельными проверками."""
        Проверка положительности каждой стороны отдельно
        side1_valid = self.side1 > 0
        side2_valid = self.side2 > 0
        side3_valid = self.side3 > 0

        if not side1_valid:
            raise ValueError("Invalid triangle sides")
        if not side2_valid:
            raise ValueError("Invalid triangle sides")
        if not side3_valid:
            raise ValueError("Invalid triangle sides")

        Проверка неравенства треугольника
        sides = self.get_sides()
        a, b, c = sorted(sides)

        sum_shorter = a + b
        is_triangle_valid = sum_shorter > c

        if not is_triangle_valid:
            raise ValueError("Invalid triangle sides")

    def get_sides(self):

```

```

        """Возвращает кортеж сторон."""
        return (self.side1, self.side2, self.side3)

def area(self):
    """Вычисление площади по формуле Герона."""
    # Вычисление полупериметра
    p = self.perimeter()
    semi_perimeter = p / 2.0

    # Разности для формулы Герона
    diff1 = semi_perimeter - self.side1
    diff2 = semi_perimeter - self.side2
    diff3 = semi_perimeter - self.side3

    # Проверка возможности вычисления
    can_compute_area = diff1 > 0 and diff2 > 0 and diff3 > 0

    if not can_compute_area:
        return 0.0

    # Вычисление произведения
    product1 = semi_perimeter * diff1
    product2 = product1 * diff2
    product = product2 * diff3

    # Проверка на отрицательное значение
    if product <= 0:
        return 0.0

    area_value = math.sqrt(product)
    return round(area_value, 4)

def perimeter(self):
    """Вычисление периметра."""
    sum1 = self.side1 + self.side2
    perimeter_value = sum1 + self.side3
    return perimeter_value

def is_right(self):
    """Проверка, является ли треугольник прямоугольным."""
    # Сортируем стороны
    sides = self.get_sides()
    a, b, c = sorted(sides)

    # Вычисляем квадраты
    a_sq = a * a
    b_sq = b * b
    c_sq = c * c

    # Проверяем теорему Пифагора
    sum_of_squares = a_sq + b_sq
    difference = abs(sum_of_squares - c_sq)

    tolerance = 1e-7
    is_right_angled = difference < tolerance

    return is_right_angled

def is_equilateral(self) -> bool:
    """Проверка, является ли треугольник равносторонним."""
    a, b, c = self.get_sides()

    sides_equal_ab = abs(a - b) < 1e-10
    sides_equal_bc = abs(b - c) < 1e-10

```

```
        return sides_equal_ab and sides_equal_bc

def is_isosceles(self) -> bool:
    """Проверка, является ли треугольник равнобедренным."""
    a, b, c = self.get_sides()

    Проверяем все возможные пары равных сторон
    ab_equal = abs(a - b) < 1e-10
    bc_equal = abs(b - c) < 1e-10
    ac_equal = abs(a - c) < 1e-10

    return ab_equal or bc_equal or ac_equal
```

Модульное тестирование

1. Калькулятор

Тестирование математического модуля Calculator включает проверку всех шести операций: сложение, вычитание, умножение, деление, возведение в степень и факториал. Для каждой операции тестируются как корректные сценарии работы, так и обработка ошибок.

Используется подход модульного тестирования с изоляцией тестируемого класса. Для каждого тестового случая создается новый экземпляр Calculator через `beforeEach`, что обеспечивает изоляцию тестов. Тесты сгруппированы по операциям с использованием `describe`.

Тесты обеспечивают 100% покрытие всех методов класса Calculator:

- Позитивные сценарии: Проверка корректных вычислений
- Граничные случаи: Работа с нулем, отрицательными числами
- Обработка ошибок: Валидация входных параметров

В процессе тестирования выявлена и исправлена следующая проблема:

Отчет об ошибке: Деление на ноль

Краткое описание ошибки: «Некорректная обработка операции деления на ноль»

Статус ошибки: открыта («Open»)

Категория ошибки: критическая («Critical»)

Тестовый случай: «Проверка обработки математических ошибок»

Описание ошибки:

1. Загрузить программу калькулятора
2. Выбрать операцию «Деление» (пункт меню 4)
3. Ввести первое число: «10»
4. Ввести второе число: «0»
5. Нажать кнопку выполнения операции

Полученный результат: Программа завершает работу с фатальной ошибкой

Ожидаемый результат: Отображение сообщения «Ошибка: Деление на ноль невозможно» с возвратом в главное меню

Результат исправления

Была реализована комплексная обработка ошибки деления на ноль:

Листинг 6 – Исправленная функция “divide”

```
divide(a, b) {  
  if (typeof a !== 'number' || typeof b !== 'number') {  
    throw new Error('Оба аргумента должны быть числами');  
  }  
  if (b === 0) {  
    throw new Error('Деление на ноль невозможно');  
  }  
  return a / b;  
}
```

2. Фигуры

Тестирование модуля включает проверку двух основных классов: `Circle` и `Triangle`. Для каждого класса тестируются как корректные сценарии работы, так и обработка ошибок валидации входных параметров.

Тестирование класса Circle

Позитивные сценарии:

- Создание круга с валидным радиусом (положительные числа)
- Вычисление площади круга по формуле πr^2
- Вычисление длины окружности по формуле $2\pi r$
- Вычисление диаметра

Граничные случаи:

- Работа с большими радиусами (1000+)
- Работа с дробными радиусами
- Проверка точности вычислений (округление до 4 знаков)

Обработка ошибок:

- Создание круга с отрицательным радиусом
- Создание круга с нулевым радиусом
- Расчет сектора с углом $\leq 0^\circ$ или $\geq 360^\circ$
- Расчет сегмента с нулевой или отрицательной хордой

- Расчет сегмента с хордой, превышающей диаметр

Тестирование класса Triangle

Позитивные сценарии:

- Создание треугольника с валидными сторонами
- Вычисление периметра
- Вычисление площади по формуле Герона
- Проверка типа треугольника (прямоугольный, равносторонний, равнобедренный)

Граничные случаи:

- Работа с дробными сторонами
- Работа с большими числами
- Почти вырожденные треугольники

Обработка ошибок:

- Создание треугольника с отрицательными сторонами
- Создание треугольника с нулевыми сторонами
- Нарушение неравенства треугольника
- Вырожденные треугольники

Выявленные проблемы

Отчет об ошибке: Некорректное вычисление площади круга

Краткое описание ошибки: «Некорректное вычисление площади круга»

Статус ошибки: Открыта ("Open")

Категория: Критическая ("Critical")

Тестовый случай: "Тест вычисления площади круга"

Описание ошибки:

1. Создать круг с радиусом 2.0
2. Вызвать метод area()

3. Получить результат вычисления площади

Полученный результат: При вычислении площади круга радиус не возводится в квадрат в соответствии с формулой πr^2 . Вместо этого используется некорректное вычисление.

Ожидаемый результат: Площадь круга должна вычисляться по формуле: $\pi \times \text{radius}^2$. Для радиуса 2.0 ожидаемое значение: $\pi \times 4 \approx 12.5664$

Результат исправления

Была исправлена функция “area”:

Листинг 7 – исправленная функция


```
def area(self):
    """Вычисление площади круга."""
    Разбиваем вычисление на шаги
    radius_squared = self.radius * self.radius
    area_value = math.pi * radius_squared
    rounded_area = round(area_value, 4)
    return rounded_area
```

Тестовое покрытие составляет 100% всех методов классов Circle и Triangle, что обеспечивает надежное обнаружение подобных математических ошибок.

Часть 2. Мутационное тестирование

1. Калькулятор

Результат мутационного тестирования:

File / Directory	I	Mutation score	Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
js calculator.js		 91.09	91	9	1	0	0	0	0	92	9	101

Мутационное тестирование класса Calculator показало высокое качество тестового покрытия - 91.09% убитых мутантов.

Статистика мутаций

- Всего мутантов: 101
- Убито мутантов: 91
- Таймаутов: 1
- Выживших мутантов: 9
- Mutation Score: 91.09%

Все мутации не изменяют фактическое поведение программы при всех

ВОЗМОЖНЫХ ВХОДНЫХ ДАННЫХ.

В связи с этим тесты в доработке не нуждаются.

2. Фигуры

Результат мутационного тестирования:



Мутационный скор: 80.0% - показатель высокого качества тестового покрытия

Статистика мутационного тестирования:

- Всего сгенерировано мутантов: 108
- Успешно обнаружено (убито): 80 мутантов (74.1%)
- Выжило мутантов: 20 (18.5%)
- Некомпетентных мутантов: 8 (7.4%)
- Таймаутов: 0 (0.0%)

Успешно обнаруженные мутации:

- Арифметические операторы (AOR) - 100% эффективность - Тесты надежно обнаруживают изменения в математических вычислениях
- Мутации условий (COD/COI) - 100% эффективность - Валидация входных данных полностью протестирована
- Реляционные операторы (ROR) - высокая эффективность - Большинство критических мутаций убиты

Области с выжившими мутантами:

- Мутации контекста выражений (SCI) - 4 выживших мутанта (24, 25, 107, 108), указывают на возможные пробелы в тестировании граничных случаев

Результат мутационного тестирования в 80% демонстрирует высокое качество тестового покрытия и соответствует лучшим практикам разработки программного обеспечения.

Вывод

Ключевым результатом выполненной работы стало формирование комплексного подхода к обеспечению качества программного обеспечения, охватывающего все этапы жизненного цикла тестирования. В процессе работы были успешно применены теоретические знания и практические навыки модульного тестирования: от изучения фундаментальных принципов и освоения инструментария до непосредственной разработки набора тестов с использованием таких фреймворков, как `pytest` и `jest`. Разработанные модульные тесты прошли этап анализа покрытия кода, который выявил как сильные стороны созданного тестового набора, так и области, требующие дополнительного внимания.

Для углубленной оценки эффективности тестов было освоено и применено мутационное тестирование с помощью инструментов `MutPy` и `Stryker`. Данный метод позволил перейти от оценки количественного покрытия к анализу качественной способности тестов обнаруживать ошибки. Мутационное тестирование выступило в роли строгого критерия, выявившего места, где тесты, несмотря на высокие показатели покрытия, не могли обнаружить определенные классы дефектов. На основе полученных метрик, таких как процент убитых мутантов, был проведен целенаправленный процесс улучшения тестовой базы.