Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №4
по дисциплине «Надёжность программного обеспечения»
Тема: ***«Оценка надежности с использованием метрик»***

Выполнил:
Студент 3-го курса, ФЭИС
Группы ПО-11
Лесько М.И.
Проверил:
Козик И. Д.

Брест 2025

Цель работы: Изучить метрики надежности программного обеспечения и применить их для оценки качества кода.

Задание: Рассчитать метрики надежности (например, количество ошибок на строку кода, цикломатическую сложность). Сравнить метрики до и после улучшения кода. Сделать выводы о влиянии изменений на надежность программы.

Ход работы

Для проведения нагрузочного тестирования была разработана программа, которая анализирует файлы своего приложения и оценивает цикломатическую сложность каждого файла с расширением .java

Код:

Начальная программа:

```
def process_data(data):
    result = []
    for item in data:
        if item > 0:
            if item % 2 == 0:
                if item < 100:
                    result.append(item * 2)
                else:
                    result.append(item)
            else:
                if item < 50:
                    result.append(item * 3)
                else:
                    result.append(item)
        else:
            if item < -10:
                result.append(abs(item))
            else:
                result.append(0)
    return result

def validate_input(input_data):
    if not isinstance(input_data, list):
        return False
    for item in input_data:
        if not isinstance(item, (int, float)):
            return False
    return True

def main():
    data = [1, 2, 3, 4, 5, -1, -2, -3, 101, 102]
    if validate_input(data):
        result = process_data(data)
        print(result)
    else:
        print("Invalid input data")

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

Измененная программа:

```
def process_positive_number(item):
    if item >= 100:
        return item
    return item * 2 if item % 2 == 0 else item * 3 if item < 50 else item

def process_negative_number(item):
    return abs(item) if item < -10 else 0

def process_data(data):
    result = []
    for item in data:
        processed_value = process_positive_number(item) if item > 0 else process_negative_number(item)
        result.append(processed_value)
    return result

def validate_input(input_data):
    if not isinstance(input_data, list):
        return False
    return all(isinstance(item, (int, float)) for item in input_data)

def main():
    data = [1, 2, 3, 4, 5, -1, -2, -3, 101, 102]
    if validate_input(data):
        result = process_data(data)
        print(result)
    else:
```

```
        print("Invalid input data")

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

Исход работы программы:

Давайте сравним результаты и сделаем выводы:

1. **Цикломатическая сложность:**
- До улучшения:
- process_data: B (высокая сложность)
- Средняя сложность: 4.33
- После улучшения:
- Все функции имеют рейтинг A (низкая сложность)
- Средняя сложность: 2.8
2. **Улучшения в коде:**
- Разделение большой функции на меньшие, более специализированные функции
- Использование тернарных операторов вместо вложенных условий
- Улучшение читаемости кода
- Упрощение логики валидации с помощью функции all()
3. **Влияние на надежность программы:**
- Уменьшение цикломатической сложности означает:
- Меньше путей выполнения кода
- Легче тестировать
- Меньше вероятность ошибок
- Код стал более модульным, что облегчает его поддержку и отладку
- Каждая функция теперь отвечает за одну конкретную задачу (принцип единой ответственности)
4. **Дополнительные преимущества:**
- Улучшенная читаемость кода
- Легче добавлять новую функциональность
- Проще находить и исправлять ошибки
- Код стал более поддерживаемым

Таким образом, после рефакторинга мы значительно улучшили надежность программы, уменьшив её сложность и сделав код более структурированным и легким для понимания. Все метрики показывают улучшение качества кода.

```
PS C:\Users\user\.vscode\cli\NPO_4> python -m radon cc NPO_4.py -a
NPO_4.py
    F 1:0 process_data - B
    F 22:0 validate_input - A
    F 30:0 main - A

3 blocks (classes, functions, methods) analyzed.
Average complexity: A (4.333333333333333)
```

```
PS C:\Users\user\.vscode\cli\NPO_4> python -m radon cc NPO_4.py -a
NPO_4.py
    F 1:0 process_positive_number - A
    F 9:0 process_data - A
    F 16:0 validate_input - A
    F 6:0 process_negative_number - A
    F 21:0 main - A

5 blocks (classes, functions, methods) analyzed.
Average complexity: A (2.8)
```