**Финансово-экономический колледж**

**РГЭУ (РИНХ)**



**Практика**

Студент: ИС-202

**Орлов Илья Вадимович**

Преподаватель:

**Журавлёв Д.Г.**

**Ростов-на-Дону**

**Теоретическая часть.  
1.Определение и основные цели измерительных методов оценки программ**

Измерительные методы оценки программ – это подходы к количественному анализу характеристик программного обеспечения с использованием метрик. Они позволяют оценивать различные аспекты кода, такие как размер, сложность, качество и надежность.

**Основные цели применения:**

1.Оценка сложности кода для выявления потенциально проблемных участков.

2.Контроль качества программного продукта.

3.Прогнозирование трудозатрат на разработку и сопровождение.

4.Сравнение альтернативных решений (например, разных архитектур или алгоритмов).

5.Повышение эффективности тестирования (например, за счет анализа покрытия кода тестами).

**2.Примеры метрик и их назначение**

**Метрики размера:**

1.Количество строк кода (LOC – Lines of Code) – измеряет объем программы. Используется для оценки трудоемкости разработки и сопровождения.

2.Количество функций/методов – помогает оценить модульность кода. Большое число функций может указывать на избыточную детализацию, а малое – на недостаточную декомпозицию.

**Метрики сложности**

Цикломатическая сложность (McCabe) – измеряет количество линейно независимых путей в программе. Чем выше значение, тем сложнее код для тестирования и понимания. Используется для выявления переусложненных функций.

**Метрики качества**

1.Коэффициент сопровождения (Maintainability Index) – оценивает легкость поддержки кода на основе его объема, сложности и других факторов. Чем выше значение, тем проще сопровождать код.

2.Покрытие тестами (Test Coverage) – показывает процент кода, выполняемого при тестировании. Используется для оценки эффективности тестовой базы.

**3.Условия применения и ограничения**

**Когда применяются метрики?**

1.На этапе разработки – для контроля качества кода.

2.При рефакторинге – для выявления проблемных мест.

3.В процессе тестирования – для оценки полноты тестов.

4.При оценке трудозатрат – для прогнозирования сроков разработки.

**Ограничения измерительных методов:**

1.Метрики не всегда отражают реальное качество кода (например, высокое покрытие тестами не гарантирует отсутствие ошибок).

2.Некоторые метрики (например, LOC) могут вводить в заблуждение (короткий код не всегда лучше).

3.Требуется контекстная интерпретация – метрики не заменяют экспертной оценки.

4.Возможны ложные срабатывания (например, высокая цикломатическая сложность не всегда означает плохой код).

**Практическая часть**

**1. Метрики размера**

|  |  |  |
| --- | --- | --- |
| Файл | Кол-во строк кода | Кол-во функций |
| Test1.py | 70+ строк | 0 (скрипт без функции) |
| Test2.py | 15 строк | 3 (calculate\_sum, factorial, print\_numbers) |

**2. Оценка сложности кода**

**Формула:**

Для test1.py:

Основной скрипт содержит:

1.Один цикл while.

2.Два вложенных условия (if response.status\_code == 200 и if char\_code in cirrencies).

Итого: сложность ≈ 3

Для test2.py:

Цикломатическая сложность = Количество ветвлений + 1

|  |  |  |  |
| --- | --- | --- | --- |
| Файл | Функция | Ветвления (if, for while) | Сложность |
| Test2.py | calculate\_sum | 0 | 1 |
| Test2.py | factorial | 1 (if n == 0) + рекурсия | 2 |
| Test2.py | print\_numbers | 1(for) + 1(if I % 2 == 0) | 3 |

**3. Выводы**

**Самая сложная функция:**

1.print\_numbers (сложность = 3) из-за цикла и условного ветвления.

2.factorial (сложность = 2) из-за рекурсии и условия.

3.test1.py (скрипт) имеет сложность ~3, но это не функция, а линейный код с ветвлениями.

**Потенциальные ошибки:**

***В test1.py:***

Нет обработки ошибок при запросах к API (например, если сервер недоступен).

Возможны проблемы с форматом даты или кодировкой XML.

***В test2.py:***

factorial: не проверяется, что n — целое неотрицательное число.

print\_numbers: нет проверки, что n > 0.

**Рекомендации:**

1.Добавить обработку ошибок в test1.py.

2.В test2.py добавить валидацию входных данных для factorial и print\_numbers.

3.Для test1.py можно выделить логику в функции, чтобы уменьшить сложность.

**Итог:**

test1.py — большой монолитный скрипт с умеренной сложностью, но без функций.

test2.py — маленький, но print\_numbers имеет наибольшую цикломатическую сложность.