Отчет

Тема: Измерительные методы оценки программ: назначение, условия применения

Задание 1. Ответы на теоретические вопросы

1. Что такое измерительные методы оценки программ?

Измерительные методы оценки программ — это способы анализа программного обеспечения, основанные на количественном измерении его характеристик с помощью специальных показателей — метрик.

Эти методы позволяют объективно оценивать такие параметры, как:

- Сложность кода

- Размер программы

- Качество реализации

- Уровень тестирования

2. Какие основные цели их применения?

Основные цели использования измерительных методов:

- Определение качества ПО (читаемость, тестируемость, надежность)

- Выявление потенциально сложных или проблемных участков кода

- Упрощение сопровождения и рефакторинга

- Сравнение модулей или версий программ

- Повышение эффективности разработки и тестирования

3. Примеры метрик и их назначение

Метрики размера

- Количество строк кода (Lines of Code, LOC)\*\* — показывает объем программы.

- Число функций/методов— характеризует структуру и модульность кода.

> Используются для оценки масштаба проекта и трудозатрат при разработке и сопровождении.

Метрики сложности

- Цикломатическая сложность (Cyclomatic Complexity)— определяет количество независимых путей выполнения в функции.

Формула: `Сложность = количество условий + 1`.

> Чем выше значение, тем сложнее читать, тестировать и поддерживать функцию.

Метрики качества

- Покрытие тестами (%) — доля кода, проверяемого автоматическими тестами.

- Коэффициент сопровождаемости— отражает простоту изменения и понимания кода (учитывает читаемость, комментарии, повторяемость).

> Эти метрики помогают оценить надежность и долгосрочную жизнеспособность кода.

4. В каких случаях применяются метрики?

Метрики применяются:

- При код-ревью и анализе технического долга

- Для выбора приоритетов рефакторинга

- На этапах тестирования и контроля качества

- В системах CI/CDдля автоматизации анализа

- При сравнении производительности и качества разных решений

5. Какие ограничения могут быть у измерительных методов?

Основные ограничения:

- Не учитывают семантику кода(не находят логических ошибок)

- Могут давать завышенные значения для простых конструкций

- Зависят от стиля написания кода

- Не всегда применимы к языкам с динамической типизацией

- Требуют интерпретации— не дают однозначных выводов без контекста

Для анализа представленного Python-скрипта оценим метрики размера, цикломатическую сложность и дадим выводы.

Задание 2. Код test1.py

1. Метрики размера

Общее количество строк кода (без пустых строк и комментариев):

67 строк

> Примечание: подсчёт включает только исполняемый код, без пустых строк и комментариев. Использовались следующие правила:

> - Строки с `import` считаются.

> - Каждая строка с логикой (циклы, условия, вызовы функций) считается.

> - Комментарии (`#`) и пустые строки не учитываются.

Количество функций:

0 функций (def)

> В данном скрипте \*\*нет явно определённых функций через `def` — весь код написан в главном потоке выполнения.

2. Оценка сложности кода

Цикломатическая сложность рассчитывается по формуле:

> \*\*Сложность = количество ветвлений + 1\*\*

В данном случае будем учитывать:

- `if`

- `for`

- `while`

Циклы и ветвления:

#### А. `while current\_date <= end\_date:`

- Это основной цикл, который проходит по датам.

- Внутри него есть:

- `if response.status\_code == 200:` → 1 ветвление

- `for valute in root.findall('Valute'):` → не добавляет сложности, так как это перебор коллекции

- `if char\_code in currencies:` → ещё одно ветвление внутри

Итого для основного блока:

- 2 ветвления

- 1 цикл (`while`)

- Цикломатическая сложность = 2 + 1 + 1 = 4

> Пояснение: +1 за вход в функцию/блок, +1 за каждый оператор `if`, +1 за цикл `while`.

3. Выводы

Самым сложным участком является тело цикла `while current\_date <= end\_date`.

Это обусловлено тем, что:

- Он содержит несколько уровней вложенности.

- Выполняет сразу несколько задач: запрос данных, парсинг XML, фильтрация по списку валют, нормализация значений.

- Отсутствие выделения в отдельные функции делает его трудным для тестирования и повторного использования.

Цикломатическая сложность = 4, что находится на грани допустимого уровня для одного блока кода.

Задание 2. Код test2.py

Анализ программного кода

1. Метрики размера

Общее количество строк кода (без пустых строк и комментариев):

15 строк

> В расчете участвовали только строки с логикой, без пустых строк и комментариев.

Количество функций:

3 функции

- `calculate\_sum(a, b)`

- `factorial(n)`

- `print\_numbers(n)`

2. Оценка сложности кода (цикломатическая сложность)

Цикломатическая сложность рассчитывается по формуле:

> \*\*Сложность = количество ветвлений + 1\*\*

### 🧮 Функция `calculate\_sum(a, b)`

- Содержит: только возврат суммы.

- Ветвлений: \*\*0\*\*

- \*\*Цикломатическая сложность: 1\*\*

### 🧮 Функция `factorial(n)`

- Содержит: рекурсивный вызов и одно условие (`if n == 0:`).

- Ветвлений: \*\*1\*\*

- \*\*Цикломатическая сложность: 2\*\*

### 🧮 Функция `print\_numbers(n)`

- Содержит: цикл `for` и условие `if i % 2 == 0:`.

- Ветвлений: \*\*1\*\*

- \*\*Цикломатическая сложность: 2\*\*

## 3. Выводы

Самыми сложными являются функции `factorial(n)` и `print\_numbers(n)`, обе имеют цикломатическую сложность 2.

- У `factorial(n)` есть условие и рекурсия — это увеличивает риск переполнения стека при больших значениях `n`.

- У `print\_numbers(n)` есть цикл и условие, что делает её более подверженной изменениям и трудной для тестирования.

Функция `calculate\_sum` — самая простая, так как не содержит условий и имеет минимальную сложность.