Отчет

Тема: Метрики программного обеспечения: направления применения, метрики сложности, метрики стилистики

Теоретическая часть

Ответы на вопросы: Метрики программного обеспечения

1. Метрики ПО: Назначение и Применение

Метрика программного обеспечения — это количественная мера, которая используется для оценки различных характеристик программного кода: сложности, размера, качества, производительности и других параметров.

Ключевые направления применения метрик:

1. Оценка сложности кода

— Помогает определить, насколько сложно читать и тестировать функцию. Высокая сложность увеличивает риск ошибок.

2. Прогнозирование трудозатрат

— На основе объема кода (например, строк кода) можно оценить время разработки и необходимые ресурсы.

3. \*\*Контроль качества кода\*\*

— Метрики позволяют выявлять участки с высокой вероятностью багов, низкой тестируемостью или плохой структурой.

4. Улучшение стиля и читаемости

— Анализ стилистики помогает повысить читаемость и облегчить сопровождение кода другими разработчиками.

Пример: Как метрики могут помочь при ревью кода?

При проведении код-ревью метрики помогают:

- Быстро находить сложные или запутанные функции.

- Обнаруживать участки с недостатком комментариев.

- Указывать на функции с большим количеством параметров, которые стоит упростить.

- Определять места, требующие рефакторинга.

2. Метрики Сложности

Измерение сложности позволяет:

- Определить, насколько легко будет читать, тестировать и поддерживать код.

- Предупредить о потенциальных проблемах: высокая сложность → больше ошибок и трудозатрат.

Проблемы высокосложного кода:

- Сложно читать и понимать.

- Сложно тестировать (высокое покрытие тестами).

- Часто становится причиной ошибок.

- Трудно модифицировать без риска поломать логику.

Другие метрики сложности:

1. Глубина вложенности условий

— Чем больше уровней вложенности `if`, тем сложнее воспринимается логика.

2. Количество параметров функции

— Функция с 5+ параметрами обычно сложна в использовании и тестировании.

3. Метрики Стилистики (Качества Кода)

Метрики стилистики показывают, насколько код соответствует стандартам написания и как его легко будет читать и изменять. Хороший стиль:

- Упрощает понимание кода новыми разработчиками.

- Уменьшает количество ошибок при доработке.

- Улучшает качество документирования и автоматического анализа.

Примеры метрик стилистики:

|  |  |
| --- | --- |
| Метрика | Описание |
| Длина идентификаторов | Слишком короткие или слишком длинные имена переменных затрудняют понимание кода. В идеале — от 3 до 15 символов. |
| Количество комментариев | Отношение числа комментариев к числу строк кода. Недостаток комментариев делает код сложным для сопровождения. |
| Длина строки кода | Слишком длинные строки снижают читаемость. Рекомендуется ограничение в 80–120 символов. |
| Глубина вложенности конструкций | Глубокая вложенность `if`, `for`, `while` усложняет логику. Лучше использовать ранний выход (`return`) или выносить в отдельные функции. |
| Соблюдение кодстайла | Единый стиль именования, отступов и форматирования облегчает работу всей команды. |

Пример: Почему слишком короткие имена переменных (x, tmp) считаются плохой практикой?

- Они не несут смысловой нагрузки, что затрудняет понимание кода.

- При чтении кода возникает необходимость "расшифровывать" значение переменной.

- Это увеличивает риск ошибок при доработке и усложняет сопровождение.

Практическая часть

Отчет по анализу стилистики программного кода

1. Метрики стилистики

Идентификаторы

#### Список всех идентификаторов:

\*\*Функция `calc(a, b, op)`\*\*

- `calc` (функция)

- `a`, `b`, `op`, `r`

\*\*Функция `find\_max(arr)`\*\*

- `find\_max` (функция)

- `arr`, `m`, `x`

Оценка понятности:

|  |  |  |
| --- | --- | --- |
| Функция | Идентификаторы | Понятность (1–5) |
| `calc` | calc, a, b, op, r | 2 |
| `find\_max` | find\_max, arr, m, x | 4 |

Примечание:

- `calc` — слишком общее имя функции, не описывает её назначение.

- `a`, `b`, `op`, `r` — короткие имена без контекста.

- `find\_max` — понятное имя; переменные `arr`, `m`, `x` имеют смысл, но `x` можно улучшить.

Средняя длина идентификаторов:

|  |  |  |  |
| --- | --- | --- | --- |
| Функция | Количество идентификаторов | Общая длина символов | Средняя длина |
| `calc` | 5 | 1 + 1 + 2 + 1 + 1 = 6 | ~1.2 |
| `find\_max` | 3 | 8 + 3 + 1 = 12 | ~4.0 |

Комментарии

Процент строк с комментариями:

- Всего строк кода (без пустых): \*\*24\*\*

- Строк с комментариями: \*\*0\*\*

- Процент комментариев: 0%

Оценка достаточности комментариев: 1/5

> ❗ Нет ни одного комментария. Это затрудняет понимание логики, особенно для новых разработчиков.

Структура кода

Средняя длина строки кода:

- Примерный подсчёт: средняя длина строки ≈ \*\*35–45 символов\*\*

- Длинные строки: почти отсутствуют → структура читаема

Соблюдение PEP 8:

- Положительные моменты:

- Хорошая структура отступов.

- Последовательное оформление блоков `if/elif`.

- Нарушения:

1. Имена функций и переменных: `calc` — слишком краткое, лучше `calculate` или `perform\_operation`.

2. Отсутствие пробелов вокруг операторов:

Например, `r=a+b` → лучше `r = a + b`

3. Отсутствие docstring'ов у функций

2. Анализ и Рекомендации

Сильные стороны кода:

1. Чистый и последовательный стиль отступов.

2. Хорошая логическая структура функций, легко читается при наличии контекста.

Слабые стороны кода:

1. Слишком короткие и непонятные имена переменных и функций.

2. Отсутствие комментариев и документации.

Предложения по улучшению:

1. Улучшение имен:

- Переименовать `calc` в `calculate` или `perform\_operation`

- `a`, `b`, `op`, `r` → заменить на `num1`, `num2`, `operation`, `result`

- `x` → `element` или `item`

Пример:

```python

def calculate(num1, num2, operation):

result = None

if operation == '+':

result = num1 + num2

elif operation == '-':

result = num1 - num2

return result

```

2. Добавление комментариев и docstring'ов:

```python

def calculate(num1, num2, operation):

"""

Выполняет указанную арифметическую операцию над двумя числами.

Аргументы:

num1 (float): Первое число.

num2 (float): Второе число.

operation (str): Операция ('+', '-', '\*', '/').

Возвращает:

float или None: Результат операции или None, если операция неверна или деление на 0.

"""

result = None

...

```

Заключение

Метрики позволяют дать объективную оценку качества кода, выявляя сложные, плохо читаемые или потенциально ошибочные участки. Использование метрик помогает улучшить читаемость, поддерживаемость и надежность программного обеспечения на ранних этапах разработки.