**1. Метрики ПО: Назначение и Применение**

**Определение метрики программного обеспечения**

**Метрика ПО** — численная характеристика некоторого аспекта программного кода или процесса разработки, позволяющая объективно оценивать его свойства (сложность, качество, эффективность и т. д.).

**Ключевые направления применения метрик**

**Оценка сложности кода**

Помогает выявить "сложные" участки кода, которые трудно поддерживать (например, цикломатическая сложность).

**Пример:** Метрика McCabe показывает количество независимых путей выполнения в функции.

1. **Прогнозирование трудозатрат**

Метрики (например, количество строк кода — **SLOC**) помогают оценить время на разработку и тестирование.

1. **Контроль качества**

Измерение количества багов, покрытия тестами и других показателей надежности.

1. **Улучшение стиля кода**

Анализ именования переменных, длины методов, глубины вложенности для повышения читаемости.

**Пример: как метрики помогают при ревью кода?**

**Цикломатическая сложность > 10** сигнализирует, что функцию стоит разбить на части.

**Высокая глубина вложенности** указывает на потенциально запутанную логику.

**Мало комментариев**  - код может быть непонятен другим разработчикам.

**2. Метрики Сложности**

**Зачем измерять сложность кода?**

**Высокосложный** код приводит к:

Трудностям в понимании и поддержке.

Увеличению числа ошибок.

Замедлению разработки (сложнее вносить изменения).

**Примеры метрик сложности**

1. **Цикломатическая сложность (McCabe)** — количество линейно независимых путей выполнения в коде.
2. **Глубина вложенности** — максимальный уровень вложенности if/for/while (например, 4+ уровня — плохо).
3. **Количество параметров функции** — если параметров > 5-7, функцию трудно использовать и тестировать.

**3. Метрики Стилистики (Качества Кода)**

**Зачем нужны метрики стилистики?**

Они напрямую влияют на:

**Читаемость** — насколько легко понять код.

**Сопровождаемость** — как просто вносить изменения.

**Примеры метрик стилистики**

1. **Длина идентификаторов**

Слишком короткие (x, tmp) — неинформативные.

Слишком длинные (customerAccountBalanceCalculator) — усложняют чтение.

Оптимально: avg\_score, user\_list.

1. **Количество комментариев**

Отношение комментариев к коду (например, 20-30%).

Слишком много комментариев - код перегружен.

Слишком мало - код непонятен.

1. **Длина строки кода**

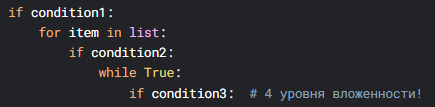
Рекомендуется ограничивать 80-120 символами.

Слишком длинные строки усложняют чтение.

1. **Глубина вложенности управляющих конструкций**

Рекомендуется не более 3-4 уровней.

Пример плохого кода:



**Соблюдение кодстайла**

Единые стандарты именования (snake\_case, CamelCase).

Правильные отступы (PEP 8 для Python).

Использование линтеров (pylint, flake8).

**Практическая часть**

1. **Метрики стилистики:**

Идентификаторы:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| calc | 3 | Слишком общее название. |
| find\_max | 5 | Понятно и точно |
| a,b,op,r | 2 | Неинформативно, лучше num1, operator, result |
| Arr,m,x | 3 | M и x слишком абстрактны, лучше max\_value, current |
| прочие | 5 | Хорошие названия (currency\_data, normalized\_value) etc. |

**Средняя длина идентификаторов:**

calc: (1 + 1 + 2 + 1) / 4 = 1.25 символа.

find\_max: (3 + 1 + 1) / 3 = 1.67 символа.

Основной код: ~7.5 символов (хорошо).

1. **Комментарии**

Общее число строк: ~90 (без пустых строк).

Комментарии: ~15 строк.

Процент: (15 / 90) \* 100 ~ 16.7% .

**Оценка достаточности (1-5):**

4 балла: Комментарии есть, но местами недостаточно пояснений (например, логика normalized\_value).

1. **Структура**

**Средняя длина строки кода:**

**Примеры строк:**

Короткие: currencies = ['DKK', 'CAD', 'NZD', 'PLN'] (30 символов).

Длинные: plt.plot(df['date'], df['value'], marker='o', linestyle='-', linewidth=2, markersize=6, color='#2c5aa0') (85 символов).

Средняя: ~50 символов (норма: до 79 по PEP 8).

**Анализ и рекомендации**

**Сильные стороны:**

1. Хорошие именования в основном коде:

currency\_data, normalized\_value, formatted\_date — понятные и информативные.

1. Оптимальное количество комментариев:

Описаны ключевые шаги (запросы API, парсинг XML).

**Слабые стороны:**

1. Слишком короткие имена в функциях:

a, b, r, m — непонятно без контекста.

Для оптимизации можно вынести парсинг XML в отдельную функцию