**Теоретическая часть**

1. Метрики ПО: Назначение и применение

***Определение***

- Метрика программного обеспечения — это количественная характеристика, позволяющая оценить определённые свойства или качество программного продукта, процессы разработки или его компонентов.

***Направления применения***

- Оценка сложности — помогает определить, насколько сложно понять, изменить или протестировать код, что влияет на трудозатраты и риск ошибок.

- Прогнозирование трудозатрат — на основе метрик можно предсказать объем работы, необходимый для разработки или сопровождения.

- Контроль качества — выявление потенциальных проблем, ошибок или слабых мест в коде, что способствует повышению его надежности.

- Улучшение стиля — контроль соблюдения стандартов кодирования, что повышает читаемость и сопровождаемость.

***Пример: ревью кода***

- Метрики помогают выявить сложные или запутанные участки кода, что облегчает их анализ и улучшение. Например, высокая глубина вложенности или длинные функции могут указывать на необходимость рефакторинга.

2. Метрики сложности

***Зачем измерять сложность кода?***

- Для оценки трудоемкости понимания, тестирования и сопровождения программных модулей.

***Высокая сложность вызывает проблемы:***

- Трудности в понимании и изменении кода.

- Повышенный риск ошибок при внесении изменений.

- Увеличение времени на тестирование и отладку.

***Дополнительные метрики сложности***

- Глубина вложенности — максимальный уровень вложенности управляющих конструкций (например, циклов, условий).

- Количество параметров функции — число входных параметров, что влияет на сложность использования и тестирования функции.

3. Метрики стилистики (качества кода)

***Зачем нужны?***

- Метрики стилистики способствуют повышению читаемости и понятности кода.

- Хороший стиль облегчает сопровождение, уменьшает вероятность ошибок и ускоряет обучение новых разработчиков.

***Примеры метрик стилистики***

- Длина идентификаторов — средняя или минимальная длина переменных и функций. Короткие имена (например, `x`, `tmp`) усложняют понимание, а слишком длинные — мешают быстрому восприятию.

- Количество комментариев — отношение комментариев к коду. Недостаток комментариев затрудняет понимание логики, а избыток может указывать на избыточность.

- Длина строки кода — максимальная или средняя длина строки. Слишком длинные строки ухудшают читаемость, особенно на небольших экранах.

- Глубина вложенности управляющих конструкций — чем меньше, тем проще понять логику программы.

- Кодстайл — соблюдение стандартов именования, отступов и форматирования. Нарушение стандартов снижает читаемость и вызывает трудности при командной работе.

***Почему короткие имена переменных считаются плохой практикой?***

- Они не дают информации о назначении переменной, что усложняет понимание кода. Например, `x` или `tmp` не отражают смысл, в отличие от `userAge` или `totalSum`.

**Практическая часть**

Метрики ПО: Назначение и применение

- Определение: Метрика программного обеспечения — это количественная характеристика, позволяющая оценить свойства или качество программного продукта, процессы разработки или его компонентов.

***Направления применения:***

- Оценка сложности: помогает определить, насколько трудно понять, изменить или протестировать код.

- Прогнозирование трудозатрат: на основе метрик можно предсказать объем работы.

- Контроль качества: выявление потенциальных ошибок и слабых мест.

- Улучшение стиля: контроль соблюдения стандартов кодирования.

- Пример: Метрики помогают при ревью кода выявлять сложные участки, требующие рефакторинга.

Метрики сложности

- Зачем измерять: для оценки трудоемкости понимания, тестирования и сопровождения.

- Проблемы высокого уровня сложности: трудности в понимании, риск ошибок, увеличение времени тестирования.

- Дополнительные метрики:

- Глубина вложенности управляющих конструкций.

- Количество параметров функции.

Метрики стилистики

Зачем нужны: для повышения читаемости и сопровождаемости кода.

***Примеры:***

- Длина идентификаторов.

- Количество комментариев.

- Длина строки кода.

- Глубина вложенности.

- Соблюдение стандартов кодстайла.

- Почему короткие имена плохие: они не дают информации о назначении переменной, усложняя понимание кода.

***Таблица с расчетами метрик для функций***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Функция | Кол-во строк | Средняя длина идентификаторов | % строк с комментариями | Средняя длина строки | Нарушения PEP 8 |
| calc | 15 | 3.75 | 20% | 78 | Отступы не всегда 4 пробела, нет пробелов вокруг операторов (a + b) |
| Find\_max | 8 | 4.33 | 0% | 66 | Нет комментариев, отступы в порядке |

***Анализ идентификаторов и комментариев***

*Идентификаторы:*

- Все имена функций и переменных понятны.

- Средняя длина — около 4 символов, что допустимо.

*Комментарии:*

- В функции `calc` есть комментарий, в `find\_max` — нет.

- Процент строк с комментариями — около 13%, что низко.

- Оценка достаточности комментариев — 2 балла (недостаточно для понимания логики).

***Нарушения стиля***

- В функции `calc` есть нарушения PEP 8:

- Отступы не всегда 4 пробела.

- В выражениях отсутствуют пробелы вокруг операторов (`a+b` вместо `a + b`).

- В функции `find\_max` отсутствуют комментарии, что ухудшает читаемость

Анализ и рекомендации

***Сильные стороны***

- Имена функций и переменных понятны и отражают смысл.

- Код структурирован и разделен на логические блоки.

***Слабые стороны***

- Недостаточное использование комментариев, особенно в сложных участках.

- Нарушения PEP 8, особенно в отступах и пробелах, снижают читаемость.

***Рекомендации***

- Добавить комментарии к сложным участкам кода, чтобы повысить его сопровождаемость.

- Использовать автоматические средства проверки стиля (например, `flake8`) и придерживаться стандартов PEP 8.

- Переименовать переменные для повышения читаемости, например, `a` и `b` в `operand1`, `operand2`.

- Упростить условие в функции `calc`, например, обработать деление на ноль более явно или использовать исключения.

***Заключение***

Метрики позволяют объективно оценить качество и стиль кода, выявить слабые места и определить направления для улучшения. Они помогают сделать код более понятным, поддерживаемым и надежным.