**Измерительные методы оценки программ**

**Цель работы**

Закрепить понимание измерительных методов оценки программного обеспечения, их назначения и условий применения. Научиться применять базовые метрики для анализа кода.

**Задание 1. Теоретическая часть**

**1. Определение измерительных методов оценки программ**

Измерительные методы оценки программ — это подходы к количественному анализу характеристик программного обеспечения с помощью метрик. Эти методы позволяют оценивать такие параметры, как сложность, размер, качество, надёжность и поддерживаемость программного кода.

**Основные цели применения:**

- Улучшение качества ПО;

- Снижение рисков при сопровождении и модификации кода;

- Оптимизация процессов разработки;

- Объективная оценка трудозатрат;

- Контроль за техническим долгом.

**2. Примеры метрик и их назначение**

**Метрики размера**:

- Количество строк кода (LOC) — показывает объём программы.

- Число функций/методов — позволяет оценить структуру и модульность.

Применение: используется для сравнения проектов, прогнозирования времени разработки, определения объёма работ.

**Метрики сложности:**

- Цикломатическая сложность (McCabe) — рассчитывается как количество линейно независимых путей в коде. Формула: M = E - N + 2P, где E — количество дуг, N — узлов, P — компонент связности графа. Упрощённо: M = количество ветвлений + 1.

*Применение:* помогает определить, насколько сложно тестировать или читать ту или иную функцию.

**Метрики качества:**

Коэффициент сопровождаемости — отношение числа изменений, требуемых для исправления ошибок, к общему числу изменений.

Покрытие тестами — процент кода, проверенного автоматическими тестами.

*Применение:* оценка тестируемости, простоты сопровождения, качества архитектуры.

**3. Условия применения**

**каких случаях применяются:**

- При рефакторинге и оптимизации кода;

- При управлении проектами (оценка трудозатрат);

- При аудите качества кода;

- Для выбора приоритетов при тестировании;

- Для контроля за уровнем технического долга.

**Ограничения:**

- Не все метрики применимы ко всем языкам и парадигмам программирования;

- Высокая сложность не всегда означает плохой код;

- Некоторые метрики игнорируют семантику программы;

- Требуется интерпретация данных, а не автоматическое принятие решений.

**Задание 2. Практическая часть**

python

def calculate\_sum(a, b):

return a + b

def factorial(n):

if n == 0:

return 1

else:

return n \* factorial(n - 1)

def print\_numbers(n):

for i in range(n):

if i % 2 == 0:

print(i, "is even")

else:

print(i, "is odd")

**1. Метрики размера**

|  |  |  |
| --- | --- | --- |
| Функция | Строки кода | Количество функций |
| calculate\_sum | 1 | 1 |
| factorial | 4 | 1 |
| print\_numbers | 6 | 1 |
| Итого | 11 | 3 |

**2. Метрики сложности**

|  |  |  |
| --- | --- | --- |
| Функция | Кол-во ветвлений | Цикломатическая сложность (M = ветвления + 1) |
| calculate\_sum | 0 | 1 |
| factorial | 1 (if n ==0) | 2 |
| print\_numbers | 2 (for и if внутри) | 3 |

**3. Выводы**

Самая сложная функция: print\_numbers с цикломатической сложностью 3. Это связано с наличием цикла for и внутреннего условия if.

Возможные места ошибок:

В функции factorial может возникнуть переполнение стека при больших значениях n из-за рекурсии.

В print\_numbers возможна проблема производительности при больших значениях n.

**Предложения по улучшению кода**

1. Функция calculate\_sum:

Уже простая, но можно добавить проверку типов (например, через assert или аннотации).

2. Функция factorial:

Рекомендуется заменить рекурсию на итеративную реализацию для повышения стабильности:

python

def factorial(n):

result = 1

for i in range(1, n+1):

result \*= i

return result

3. Функция print\_numbers:

Можно разделить логику вывода и проверки четности, чтобы повысить читаемость и тестируемость:

python

def is\_even(x):

return x % 2 == 0

def print\_numbers(n):

for i in range(n):

status = "even" if is\_even(i) else "odd"

print(f"{i} is {status}")

**Заключение**

Анализ программ с использованием измерительных метрик позволяет объективно оценить сложность и качество кода. На примере предоставленного фрагмента Python-кода были рассчитаны метрики размера и сложности. Полученные данные позволили выявить наиболее сложную функцию (`print\_numbers`) и предложить пути её улучшения. Такие метрики полезны на этапах рефакторинга, тестирования и управления проектами.