**ЗБ-ПИ20-2 Просветов Никита Экзамен   
Вариант 27**

**1 вопрос.** Критерии выделения маршрутов. Метрика Маккейба

Метрика Маккейба, также известная как цикломатическая сложность, — это метрика, используемая для оценки сложности программы на основе её управляющей структуры. Она измеряет количество независимых путей через программу. Каждый такой путь представляет собой уникальный путь выполнения, который может быть осуществлён в зависимости от условий и ветвлений в коде.

**Критерии выделения маршрутов по метрике Маккейба:**

1. **Ветвления и условные операторы**: Каждый раз, когда программа встречает условный оператор (например, if, while, for), добавляется новый независимый путь, поскольку код может пойти по разным направлениям в зависимости от выполнения этого условия.
2. **Циклы**: Циклы (for, while) также добавляют новые пути, так как они могут повторяться несколько раз или не выполняться вообще.
3. **Логические операторы**: Операторы, такие как ‘&&’ (и) и ‘**||’** (или), внутри условий могут создавать дополнительные пути, так как они могут разбивать проверку на несколько частей.
4. **Множественные выходы**: Наличие нескольких точек выхода из функции (например, несколько операторов return) увеличивает количество путей.

**Как рассчитывается цикломатическая сложность:**

Цикломатическая сложность V(G) графа потока управления программы может быть рассчитана с использованием следующей формулы:

Изображение выглядит как Шрифт, белый, текст, типография

Автоматически созданное описание

где:

* E — количество рёбер (переходов) в графе.
* N — количество узлов (вершин) в графе.
* P — количество компонент связности графа (обычно это 1 для одной функции или модуля).

Таким образом, метрика Маккейба помогает определить количество тестов, необходимых для покрытия всех независимых путей в программе, что способствует более полному тестированию и повышению надёжности кода.

**2 вопрос.** Программа находится в процессе испытаний 15 часов. При этом было выявлено 33 ошибок. Коэффициент сжатия тестов = 5. Первоначальное число ошибок в программе - 80. Заданная наработка на отказ - 2. Количество операторов в программе - 2000. V = 108; К = 4\*10-7; Найти надёжность по модели Муса?

Для определения надежности по модели Муса, необходимо выполнить несколько шагов расчетов с учетом предоставленных данных. Модель Муса - это метод оценки надежности программного обеспечения, который основывается на оценке ошибок, обнаруженных во время тестирования, и предполагаемом числе оставшихся ошибок.

import math

# Исходные данные

T = 15  # Время тестирования (часов)

E = 33  # Количество выявленных ошибок

lambda\_compression = 5  # Коэффициент сжатия тестов

N0 = 80  # Первоначальное число ошибок

MTTF\_required = 2  # Заданная наработка на отказ (часов)

S = 2000  # Количество операторов

V = 108  # Объем программы

K = 4 \* 10\*\*-7  # Коэффициент сложности

# Шаг 1: Расчет интенсивности отказов (lambda(t))

lambda\_t = lambda\_compression \* (E / T)

# Шаг 2: Расчет числа оставшихся ошибок (N(t))

N\_t = N0 \* math.exp(-lambda\_t \* T)

# Шаг 3: Оценка надежности (R(t))

R\_t = math.exp(-lambda\_t \* T)

# Шаг 4: Проверка по заданной наработке на отказ (MTTF)

MTTF\_calculated = 1 / lambda\_t

# Вывод результатов

print(f"Интенсивность отказов (λ(t)): {lambda\_t} ошибок в час")

print(f"Число оставшихся ошибок (N(t)): {N\_t}")

print(f"Надежность (R(t)): {R\_t}")

print(f"Расчетная наработка на отказ (MTTF): {MTTF\_calculated} часов")

# Проверка выполнения условия заданной наработки на отказ

if MTTF\_calculated >= MTTF\_required:

    print(f"Программа удовлетворяет требованию наработки на отказ (MTTF >= {MTTF\_required} часов)")

else:

    print(f"Программа не удовлетворяет требованию наработки на отказ (MTTF < {MTTF\_required} часов)")

Интенсивность отказов (λ(t)): 11.0 ошибок в час

Число оставшихся ошибок (N(t)): 1.7559028064114392e-70

Надежность (R(t)): 2.194878508014299e-72

Расчетная наработка на отказ (MTTF): 0.09090909090909091 часов

Программа не удовлетворяет требованию наработки на отказ (MTTF < 2 часов)

Этот код выполняет все необходимые расчеты и выводит результаты:

1. Интенсивность отказов (λ(t))
2. Число оставшихся ошибок (N(t))
3. Надежность (R(t))
4. Расчетную наработку на отказ (MTTF)

Также код проверяет, соответствует ли рассчитанная наработка на отказ заданной требуемой наработке на отказ.

Эта оценка показывает, что вероятность отказа очень мала, и программа достигла высокой надежности. Однако, если заданная наработка на отказ должна быть 2 часа, текущая надежность не удовлетворяет этому критерию, так как текущая оценка наработки на отказ *MTTF≈0.091* часа, что значительно меньше требуемых 2 часов.

**З вопрос.** Каким образом можно получать комплексную оценку качества на основе множества метрик качества (по типам историй, по типам работ)? Предложите вариант автоматизации принятия решения по матричной форме показателей успешности и бездефектности ПС.

Для получения комплексной оценки качества программного обеспечения (ПО) на основе множества метрик качества и автоматизации принятия решений можно использовать матричный подход. Этот подход предполагает создание матрицы, где строки представляют типы историй или работ, а столбцы - метрики качества. Каждая ячейка матрицы содержит значение метрики для конкретного типа истории или работы.

Для автоматизации процесса оценки качества и принятия решений на основе этой матрицы можно использовать специализированное программное обеспечение, которое будет автоматически собирать данные о качестве по каждой истории или работе, рассчитывать значения метрик и предоставлять результаты в виде матрицы. На основе этой матрицы можно проводить анализ качества ПО, выявлять слабые места и принимать решения о необходимости улучшения определенных аспектов разработки.

Пример автоматизации принятия решения на основе матричного подхода:

1. **Сбор данных**: Программное обеспечение собирает данные о качестве по каждой истории или работе, включая информацию о дефектах, времени выполнения, затраченных ресурсах и т.д.
2. **Расчет метрик**: Для каждой истории или работы рассчитываются значения метрик качества, таких как количество дефектов, время выполнения, стоимость и т.п.
3. **Создание матрицы**: Полученные значения метрик заносятся в матрицу, где строки представляют типы историй или работ, а столбцы - метрики качества.
4. **Анализ матрицы**: Анализируется матрица для выявления тенденций, закономерностей и слабых мест в качестве ПО.
5. **Принятие решений**: На основе анализа матрицы принимаются решения о необходимости улучшения определенных аспектов разработки, перераспределении ресурсов, изменении процессов и т.д.

Такой подход позволяет систематизировать процесс оценки качества, автоматизировать сбор и обработку данных, а также упростить принятие решений на основе полученной информации.