ЗБ-ПИ20-2 Петренко Артемий экзамен В25

**Вопрос №1:**

Метрики связности модулей. Метрики сцепления модулей.

**Ответ:**

Модуль, состоящий из компонент - методы классы, библиотека функций- может быть охарактеризован с точки зрения способов взаимодействия компонент между собой - связность и способа обращения к ним извне сцепление. Стив Макконнелл (автор книги «Совершенный код») объясняет понятие связности на примере класса (как частного случая модуля): «связность характеризует то, насколько хорошо все методы класса или все фрагменты метода соответствуют главной цели, — иначе говоря, насколько сфокусирован класс».

Виды связности обозначаются весами, отражающими степень связности компонент:

**Совпадение** (СВ=0) – компоненты не имеют ничего общего, кроме факта

нахождения в общем модуле;

**Логическая** (СВ=1) – компоненты в рамках общего функционала, но не связаны между собой, например, обработка ошибок различных типов;

**Временная** (СВ=3) - используются на одной фазе процесса, в один период

времени - инициализация, завершение;

**Процедурная** (СВ=5) – используются в рамках одного сценария, имеют

определенный порядок вызова, т.е. могут быть связаны между собой по

результату;

**Коммуникативная** (СВ=7) – работают с общей структурой данных;

**Последовательная** (СВ=9) – результат первого является входом второго;

**Функциональная** (СВ=10) – один модуль вызывает другой.

В ООП вводится объектная связность - в рамках класса с общими свойствами и функциональностью, она может быть любой из перечисленных, но для класса естественным выглядит использование трех последних:

Виды сцепления также обозначаются весами:

**Сцепление по данным** (СЦ=1): вызов с параметрами – примитивными типами данных;

**Сцепление по образцу** (СЦ=3): вызов с параметрами-объектами, структурами данных;

**Сцепление по управлению** (СЦ=4): модуль устанавливает флаги в другом

модуле, управляя его поведением;

**Сцепление по внешним ссылкам** (СЦ=5): модули используют один и тот же

глобальный элемент данных или ссылку на него;

**Сцепление по общей области** (СЦ=7): модули разделяют одну и ту же

глобальную структуру данных или используют ссылку на общий объект;

**Сцепление по содержанию** (СЦ=9): один модуль прямо исполняет часть кода

другого модуля.

Сама по себе степень связности или сцепления не является показателем хорошего или плохого кода. Они лишь являются индикаторами, сигнализирующими о сложностях с **управляемостью** кода.

Среди объектно-ориентированные метрик можно выделить следующие типы связностей:

1. Метрика связности (cohesion) **класса по данным**
2. Метрика связности класса по методам
3. Прагматические объектно-ориентированные метрики
4. Метрики Чидамбера и Кемерера предназначены для комплексной оценки качества класса
5. Метрика Мартина
6. Метрики Лоренца и Кидда

**Вопрос №2:**

Программа находится в процессе испытаний 15 часов. При этом было выявлено 40 ошибок. Коэффициент сжатия тестов = 6. Первоначальное число ошибок в программе -90. Заданная наработка на отказ – 4. Количество операторов в программе – 1300. V = 108; K = 3\*10-7; Найти надёжность по модели Муса.

**Ответ:**

Исправленное время испытаний () учитывает влияние коэффициента сжатия тестов (C), поэтому с учетом коэффициента сжатия выглядеть оно будет так:

- количество ошибок, найденных в процессе испытаний, является суммой первоначального количества ошибок () и ошибок, найденных во время испытаний ()

Интенсивность отказов (λ) - это среднее количество отказов в единицу времени:

В модели Муса надежность (R(t)) в момент времени t определяется как вероятность безотказной работы в течение этого времени.

Надежность ПС для периода эксплуатации t определяются по формуле:

**Таким образом**: на основании модели Муса, надежность программы после 4 часов работы составляет **36.79%.**

**Вопрос №3:**

Приведите выражения для оценки метрики времени выполнения проекта, оценки производительности (по типам историй, по типам работ) и уровня ошибок (по типам историй, по типам работ)

**Ответ:**

Метрики по времени выполнения и производительности:

1. Метрика **Burn Rate:** диаграмма сгорания задач показывает, сколько работы уже выполнено и сколько осталось. Данную метрику можно построить, для:
   1. – конкретного спринта — это называется **sprint burndown chart** или диаграмма сгорания задач за спринт;
   2. всего проекта — это называется **release burndown chart** или диаграмма сгорания задач до релиза

График представляет собой кривые, идущие вниз: они показывают динамику решения задач. По шкале X отмечают количество дней до окончания спринта или до релиза. По шкале Y — число задач. Одна из линий графика демонстрирует то, как быстро планировалось выполнить работу. Вторая линия — то, как работа идет на самом деле.

1. Метрика **Flow Efficiency**. В разработке программного обеспечения на заказ есть стадии активной работы и время ожидания: когда для того, чтобы приступить к задаче, не хватает информации или ресурсов. Соотношение времени работы ко всему времени с учетом простоя и называют эффективностью потока.

Например, если над задачей работали 3 дня, а еще 2 она «висела» в ожидании, когда разработчик для нее освободится, формула эффективности потока будет: **3/ (2+3) \*100%.**

1. Метрика **Team Velocity** демонстрирует, какой объем работы команда способна выполнить за спринт. Для вычисления показателя производительности в продуктовых командах используют стори поинты (story points) — с их помощью каждой задаче в бэклоге назначают вес в зависимости от ее сложности. Вместо стори поинтов оценка идет в часах: так всем проще и понятнее. Сумма всех стори поинтов или всех часов за спринт — это и есть производительность.
2. **Длительность итераций/спринтов:** средняя продолжительность итераций/спринтов в течение проекта.
3. **Время выполнения задач:** среднее время, необходимое для выполнения задач определенного типа.
4. **Соблюдение сроков:** процент задач, выполненных в срок.

Метрики по уровню ошибок:

1. Метрика **среднее количество багов** за спринт. Баги при разработке ПО возникают неизбежно. Чтобы их контролировать, можно подсчитывать количество в каждом спринте. На примере нескольких спринтов можно выявить нормальный показатель для конкретного проекта и в дальнейшем равняться на него. А существенные отклонения от средней цифры — повод задуматься.
2. **Test Coverage.** Плотность покрытия программного кода тестами — это процент кода, который проверили в ходе тестирования. Например, если этот показатель равен 78%, значит, 78% всего, что написали программисты, было проверено в ходе теста.
3. **Коэффициент дефектов:** процент историй/задач, в которых были найдены дефекты.
4. **Плотность дефектов:** количество дефектов на 1000 строк кода.
5. **Уровень серьезности дефектов:** распределение дефектов по уровню серьезности (критические, основные, мелкие).
6. **Время исправления дефектов:** среднее время, необходимое для исправления дефекта.
7. **Процент исправленных дефектов:** доля исправленных дефектов к определенному моменту времени.
8. **Профилактика дефектов:** количество дефектов, предотвращенных на этапе разработки.