

Проектирование и отбор тестов

Еникеев Р.Р.

Тестирование ПО – процесс, позволяющий удостовериться в том, что программный код

- делает все то, для чего он предназначался
- не делает того, для чего он не предназначался
- Исчерпывающее (exhaustive) тестирование – тестирование, которое позволяет обнаружить все ошибки

Отбор тестов

- Отбор тестов обусловлен ограниченностью ресурсов и невозможностью 100% тестирования
- Самый простой способ – оценка риска

Оценка риска (risk estimate)

- Оценка риска использует приоритеты (для бизнеса) для
 - создаваемых тестов во время генерирования тест-кейсов,
 - исполняемых и уже созданных тест-кейсов в момент выполнения тестирования.
- Шаги
 - Узнать необходимую информацию, в частности, приоритет
 - Оценить риск, используя полученные данные
- Информация и анализ приоритетов должны быть **объективными**, поэтому тестировщик
 - не должен предполагать
 - получает информацию
 - из объективных источников (статистика)
 - от компетентного специалиста/эксперта
- Оценка риска также может производиться с помощью эксперта
- Недостаток анализа риска - тестируются не все сценарии

Критерии тестирования

- Для проверки всех сценариев используются критерии тестирования
- Идеальный критерий должен быть:
 - **достаточным** – показывать, когда некоторое конечное множество тестов достаточно для тестирования данной программы.
 - **полным** – в случае ошибки должен существовать тест из множества тестов, который раскрывает ошибку.
 - **Надежным** – любые два множества тестов, удовлетворяющих ему, одновременно должны раскрывать или не раскрывать ошибки программы
 - **легко проверяемым** – например, вычисляемым на тестах

Критерии тестирования

- Исчерпывающее тестирование невозможно, поэтому
 - не существует полного критерия
 - наша задача – выбрать такое подмножество тестов, чтобы обеспечить максимальную вероятность обнаружения большинства ошибок
- Классы критериев
 - Структурные критерии («белого ящика»)
 - Функциональные критерии («черного ящика»)
 - Критерии стохастического тестирования
 - Мутационные критерии
- Данные критерии тестируют код по спекам
 - соответственно, не находят ошибки в спеках

Мутационный критерий

- Используется для написанных тест-кейсов
- Критерий
 - выполняется **мутационное** тестирование
 - в программу вносятся незначительные изменения

```
if (a + b < c) -> if (a + b <= c)
```

```
if (i < n - 1) -> if (i < n)
```

- проводится тестирование
 - если набор тестов не в состоянии обнаружить такие изменения (тесты по-прежнему проходят), то он рассматривается как недостаточный и нужно
 - расширить набор тестов
 - повторить тестирование

Стохастический критерии

- **Стохастическое** тестирование
 - используется, когда
 - когда набор тестов имеет громадную *мощность* или
 - необходимо построить тестовые наборы большой мощности
 - основано на случайной генерации входных данных
- Критерий формулируется в терминах проверки наличия заданных свойств у тестируемого приложения, средствами проверки некоторой статистической гипотезы
- Недостаток – малая вероятность получения оптимального тестового набора (с высокой обнаруживающей способностью)

Выполнение стохастического тестирования

- Случайная генерация последовательности входных данных $\{x\}$
- Выходные данные $\{y\}$
 1. Вычисление последовательности $\{y\}$
 2. Вычисление $\{y\}$ невозможно
- Запуск программы и проверка фактического выхода $\{y_{\text{ф}}\}$
 1. Проверка равенства $\{y_{\text{ф}}\}$ и $\{y\}$
 2. Сравнение соответствия $\{y_{\text{ф}}\}$ заранее известному распределению результатов $F(Y)$



Критерии стохастического тестирования

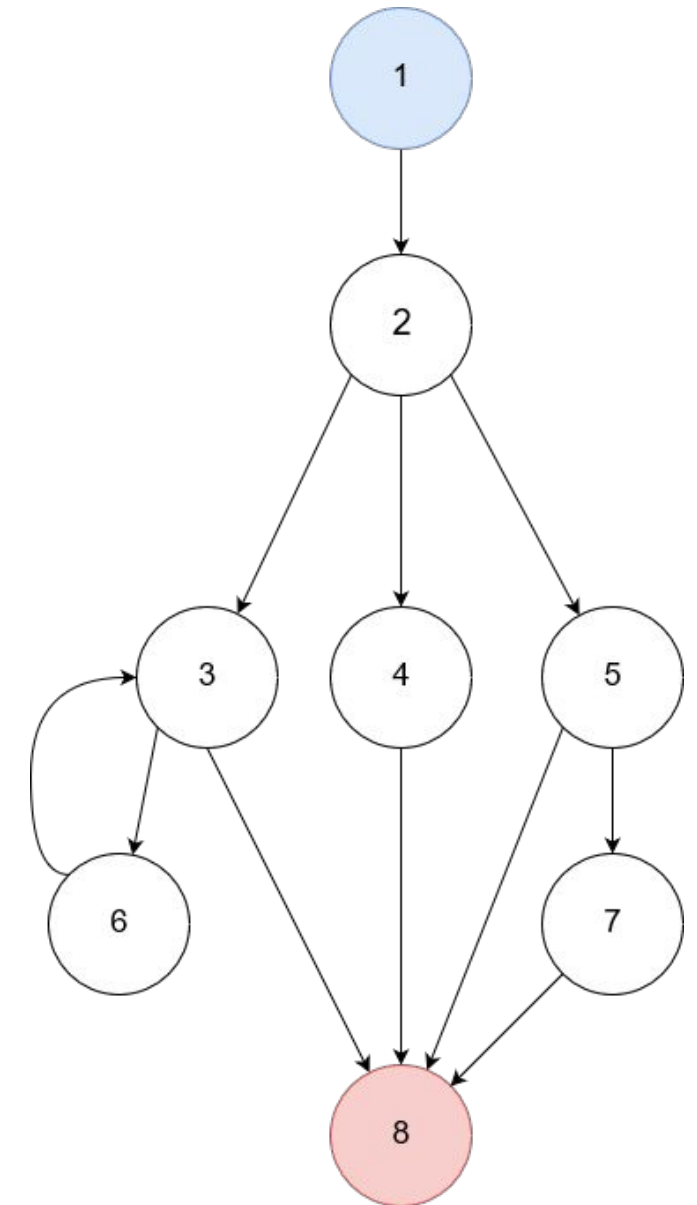
- **Статистические методы** окончания тестирования
 - стохастические методы принятия решений о совпадении гипотез о распределении случайных величин (метод Стьюдента, Хи-квадрат)
- Метод оценки скорости выявления ошибок – тестирование прекращается, если оцененный интервал времени между текущей ошибкой и следующей слишком велик для фазы тестирования приложения.
 - Чем дольше мы не находим баг, тем меньшая вероятность его найти

Тестирование методом «белого ящика»

- Тестирование, управляемое логикой программы (*logic-driven*)
 - Тесты покрывают логику программы
- Тестировщик подбирает тестовые/входные данные путем анализа логики программы
 - Не нужно забывать про спецификацию
 - Выходные данные не анализируются
- Недостатки
 - Структура программы может меняться
 - Применяется для уже написанной программы (нельзя разрабатывать тест-кейсы во время этапа «Кодирование»)

Граф потока управления (ГПУ)

- Граф потока управления:
 - Вершина – линейный блок кода
 -  - Входной блок (вход в функцию)
 -  - Выходной блок (выход из функции)
 - Дуга – инструкция перехода
- Путь (выполнения программы) – последовательность вершин из входного блока в выходной
 - (1, 2, 5, 8)
 - (1, 2, 3, 6, 3, 6, 3, 8)



Исчерпывающее тестирование «белого ящика»

- Исчерпывающее тестирование – исчерпывающее тестирование путей
 - Все пути выполнения программы покрыты тестами
 - Количество путей очень велико (циклы)
- Исчерпывающее тестирование путей \neq полное тестирование
 - Не проверяет соответствие программы спецификации (не проверяет выход)
 - Сортировка по убыванию вместо по возрастанию
 - Не обнаруживает отсутствующие пути
 - Не может найти ошибки, обусловленные чувствительностью алгоритма к данным
 - Ошибка в операторе `if (a + b < c)` вместо `if (a + b <= c)`

Структурный критерий («белый ящик»)

- Критерии – выполнение каждой
 - Покрытие команд/инструкций - каждая инструкция должна быть выполнена
 - необходимый критерий для последующих
 - Покрытие ветвлений – каждая ветвь должна быть выполнена хотя бы раз
 - каждое условие в программе должно принимать как true, так и false
 - Покрытие условий – каждое элементарное условие (без and, or) принимало и true, и false
 - Может не удовлетворять критерию покрытия ветвления `if (a && b)`
 - Покрытие ветвлений и условий
 - Комбинаторное покрытие условий – предыдущее + комбинируются всевозможные результаты элементарных условий

Тестирование «черным ящиком»

- Тестирование «черным ящиком» - тестирование, управляемое данными (data-driven)
- Исчерпывающее тестирование – исчерпывающее входное тестирование со всеми *возможными* (а не допустимыми) значениями
- Мы должны найти компромисс, добиваясь максимизации количества ошибок, обнаруживаемых конечным числом тестов
 - => нужно приблизительно представлять внутренний механизм работы программы и выдвигать разумные предположения/догадки

Критерии «Черного ящика»

- Эквивалентные классы (equivalent classes) – входные данные разбиваются на классы эквивалентностей по
 - входным данным
 - выходным данным
 - Достаточность – подать на вход входные данные из каждого класса
- Граничные значения (boundary values)
 - Из классов эквивалентностей берутся граничные значения ([1, 2, 3] для треугольника)
- Причинно-следственные связи
- Прогнозирование ошибок – выдвигаются предположения (интуитивные) о списке ошибок или ситуаций, в которых ошибки могут появиться

Прогнозирование ошибок (пример)

- Метод - требуется перечислить ситуации, которые могут быть не учтены
- Примеры ситуаций для «сортировка массива»
 - Пустой массив
 - Один элемент
 - Все значения в массиве одинаковые
 - Массив уже отсортирован

ИСТОЧНИКИ

- Савин Р. – Тестирование дот ком. с 188-203
- Маейрс и др. – Искусство тестирования программ – Главы 2 и 4
- https://ru.wikipedia.org/wiki/Мутационное_тестирование
- ИНТУИТ - Основы тестирования программного обеспечения
- Лекция 3