Тестирование ПО

Лекция 5. Методики тестирования

ШТАНЮК А.А., 2019

Позитивные и негативные тесты

Позитивные тесты

- Тесты, предназначенные для проверки, что программа выполняет свое основное предназначение
- Тесты на основании «правильных» входных данных
- Тестирование с целью проверки соответствий требованиям

Негативные тесты

Тесты для проверки устойчивости программы к негативным входным данным

Тесты на проверки устойчивости программы к ошибкам пользователя

Тесты на то что у программы нет неожиданных побочных эффектов

Тестирование с целью «сломаем это!»

«Черный ящик»

«Черный ящик»

- Не знаем/Игнорируем устройство тестируемого объекта
- Можем управлять входными параметрами
- Среда, в которой проводим эксперименты, может считаться входным параметром
- Можем измерять выходные параметры

Шаги

1. Изучение спецификаций и требований
2. Выбор входных значений
3. Определение ожидаемых выходных значений
4. Исполнение тестов
5. Сравнение полученных результатов с ожидаемыми

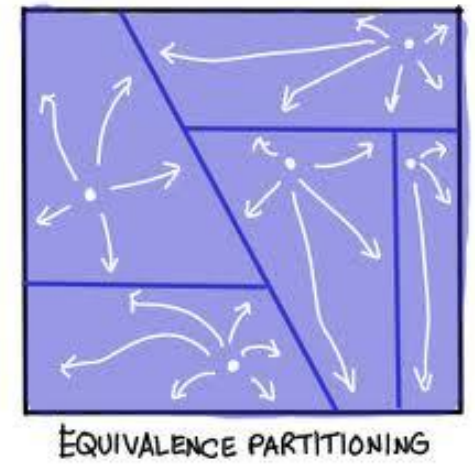
Стратегии

Число тестов определяется числом входов и диапазоном входных данных

Перебор всех вариантов (по диапазону), как правило, невозможен!

Стратегии уменьшения числа тестов:

1. Классы эквивалентности
2. Граничные условия



Классы эквивалентности

Если от двух тестов ожидается одинаковый результат – они эквивалентны
Группа тестов представляет **класс эквивалентности** если:

- Все тесты предназначены для выявление одной и той же ошибки
- Если один тест выявит ошибку, то и остальные это сделают
- Если один из тестов не выявит ошибку, то и остальные этого не сделают

Дополнительные практические критерии:

- Тесты включают значения одних и тех же входных данных
- Для проведения теста выполняются одни и те же операции программы
- В результате тестов формируются значения одних и тех же выходных данных
- Ни один из тестов не вызывает выполнения конкретного блока обработки ошибок либо выполнение этого блока вызывается всеми тестами

Классы эквивалентности

Программа классификации треугольников

Классы эквивалентности по корректным входным данным:
- Равнобедренные треугольники
- Равносторонние треугольники
- Прямоугольные треугольники
- Просто треугольники

Классы эквивалентности по некорректным входным данным:
- Отрезки не образуют треугольник
- Числа больше sizeof(int)
- Строка, содержащая буквы

Классы эквивалентности

Программа, говорящая дату следующего дня

Классы эквивалентности по корректным входным данным:

- День от 1 до 27
- Последний день месяца
- Последний день года
- 28 февраля високосного года

Классы эквивалентности по некорректным входным данным:

- Месяц > 12
- День > 31
- Неверная строка

Классы эквивалентности

Построение классов эквивалентности – субъективный процесс

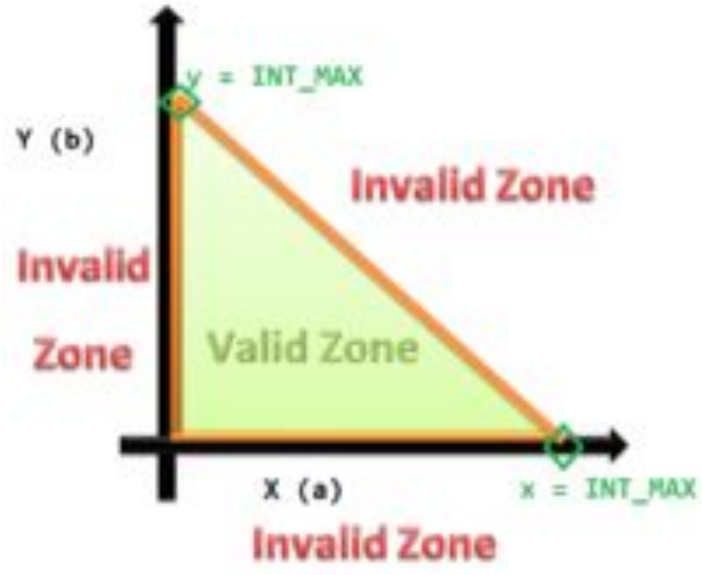
Общие рекомендации:

- Не забывайте о классах некорректных данных
- Формируйте классы в виде таблицы или плана
- Определите диапазоны числовых значений входных данных
- Проанализируйте варианты выбора из списков и меню
- Поищите переменные значения которых должны быть равными
- Поищите классы значений, зависящих от времени
- Выявите группы переменных, совместно участвующих в конкретных вычислениях
- Посмотрите на какие действия программа отвечает эквивалентными событиями
- Продумайте варианты среды тестирования

Граничное тестирование

Тестирование значений лежащих на границе классов эквивалентности, т. к. там выше вероятность возникновения ошибки

```
int safe_add( int a, int b )
{
    int c = a + b ;
    if ( a >= 0 && b >= 0 && c < 0 )
    {
       fprintf ( stderr, "Overflow!\n");
    }
    if ( a < 0 && b < 0 && c >= 0 )
    {
       fprintf ( stderr, "Underflow!\n");
    }
    return c;
}
```



Граничное тестирование

- Определяем границу класса эквивалентности
- Проверяем значения, лежащие ровно на границе
- Проверяем значения лежащие максимально близко к границе с обоих сторон

Пример:

При покупке более 100 единиц товара дается скидка 5%. Нужно проверить:

- 100
- 99
- 101

Преимущества и недостатки «ЧЯ»

Преимущества:

- Тестирование с точки зрения пользователя
- Не требует специальных знаний (например конкретного языка программирования)
- Позволяет найти проблемы в спецификациях
- Можно создавать тесты параллельно с кодом
- Тестировщик может быть отделен от разработчиков

Преимущества и недостатки «ЧЯ»

Недостатки:

Эффективность зависит от выбора конкретных тестовых значений

Необходимость наличия четких и полных спецификаций

Невозможность сконцентрироваться на особо сложных частях кода

Трудность локализации причины дефекта

Возможность не протестировать часть кода

«Белый ящик»

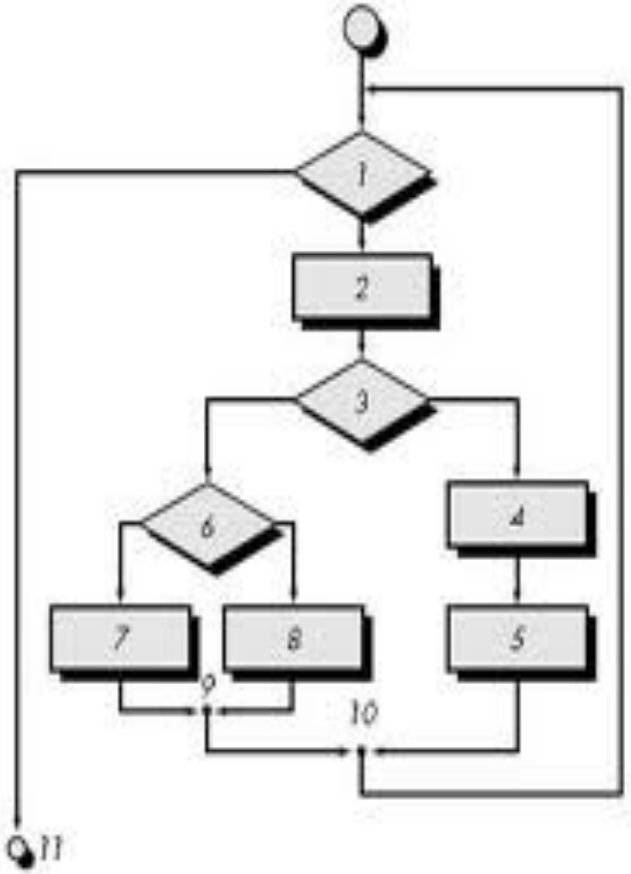
- Используем знание об устройстве тестируемого объекта
- В случае ПО – имеем полный доступ к тестируемому коду

Стадии применения:

- Unit-тестирование
- Интеграционное тестирование

Шаги

Представляем программу в виде графа



Шаги

Создаем тестовые сценарии чтобы:

- Попасть в каждое ветвление
- Пройти хоть раз через все вершины
- Пройти всеми возможными путями
- Пройти через вновь добавленные участки
- Пройти через известные проблемные участки

Метрики

Покрытие кода (code coverage) – мера измерения оттестированости имеющегося программного кода

Microsoft Visual Studio 2010(C++, C#)

DevPartner (C#, Java)

Codecov из Intel Compiler (C, C++, Fortran)

Jtest (Java)

Devel::Cover (Perl)

PHPUnit (PHP)

Coverage (Python)

CoverMe (Ruby)

Преимущества и недостатки

Преимущества:

- Позволяет найти «скрытые» в коде дефекты
- Позитивные побочные эффекты (например, обучение команды)
- Нахождение проблем производительности
- Более надежное разбиение на классы эквивалентности
- Как правило, ускорение цикла нахождение-исправление

Недостатки:

- Не найдем пропущенное в коде
- Дорого

Сравнение «ящиков»

Критерий	Черный Ящик	Белый Ящик
Основной уровень применимости	Приемочное тестирование	Юнит-тестирование
Ответственный	Независимый тестировщик	Разработчик
Знание программирования	Не обязательно	Необходимо
Знание реализации	Не обязательно	Необходимо
Знание сценариев использования	Необходимо	Не обязательно
Основа тестовых сценариев	Спецификации	Код

«Серый» ящик

Комбинация черного и белого ящиков:

- Знаем частично или полностью внутреннее устройство тестируемого объекта
- Тестировщик находится на уровне пользователя

Пример:

Зная особенности реализации модуля, создаем тестовые сценарии пользовательского уровня, которые покрывают потенциально проблемную область

Основная область применения: интеграционное тестирование

Выбор входных значений

Бессистемный выбор входных значений не позволит найти большое количество дефектов. Необходимо использование методов для выбора набора входных значений.

Основные методы выбора входных значений:

- Перебор всех возможных значений
- Случайные входные данные
- Предугадывание ошибки
- Построение графов «причина-следствие»
- Использование классов эквивалентности
- Исследование граничных значений

Метод перебора

Перебираем все возможные значения входных параметров

Последовательный перебор всех возможных комбинаций входных значений

Попарный перебор. Перебираем комбинации пары 2х входных параметров. Работаем в предположении что параметры попарно зависимы. На практике находит ~80% функциональных дефектов низкого уровня

Случайные входные данные

Генерируются случайные входные данные. Либо данные случайным образом выбираются из большого тестового набора, который не успеваем проверить целиком

- Часто используется в нагрузочном тестировании
- Необходимо иметь метод определения корректности выхода

Пример: программа подсчета числа вхождений символа в строку

Предугадывание

Составление тестовых сценариев на основании опыта предыдущего тестирования

Используйте знания о известных проблемных местах вашего продукта

Знайте распространенные ошибки программирования и пишите тесты для их поиска

- Некорректная работа с памятью: переполнение, чтение за пределами, утечки памяти
- Отсутствие обработки некорректных входных данных
- Ошибки работы с типами данных: переполнение, приведение, приближение
- Ошибки многопоточности: deadlock, data race
- Отсутствие инициализации/сброса переменных
- Недостаток привилегий, недоступность ресурсов
- ….