Тестирование белого ящика

Введение

- Модель программы в виде "белого ящика" предполагает знание исходного текста программы или спецификации программы в виде потокового графа управления.
- Структурная информация понятна и доступна разработчикам подсистем и модулей приложения, поэтому данный вид тестирования часто используется на этапах модульного и интеграционного тестирования (Unit testing, Integration testing).
- Структурные критерии тестирования базируются на основных элементах управляющего графа программы - операторах, ветвях и путях.

Управляющий граф программы

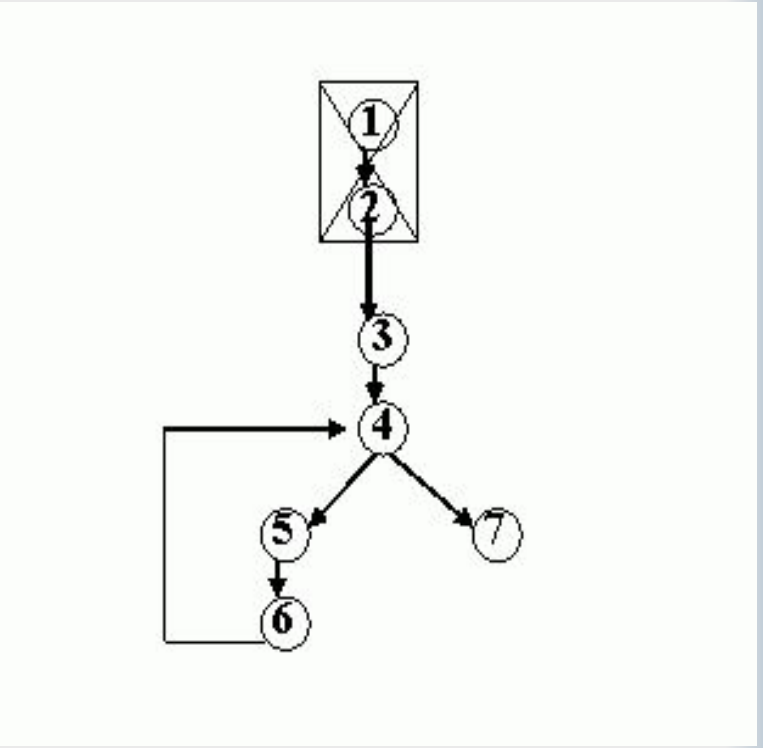
- ***Управляющий граф программы (УГП)*** отображает поток управления программы. Это граф G(V, A), где V(V1,… Vm) – множеств вершин (операторов), A(A1,… An) – множество дуг (управлений), соединяющих вершины.
- ***Путь*** – последовательность вершин и дуг УГП, в которой любая дуга выходит из вершины Vi и приходит в вершину Vj
- ***Ветвь*** – *путь*(V1, V2, … Vk), где V1 - либо первый, либо условный оператор, Vk - либо условный оператор, либо оператор выхода из программы, а все остальные операторы – безусловные.
- Число путей в программе может быть не ограничено (пути, различающиеся хотя бы числом прохождений цикла – разные). Ветви - линейные участки программы, их конечное число.
- Существуют реализуемые и нереализуемые пути в программе, в нереализуемые пути в обычных условиях попасть нельзя.

Реализуемые и нереализуемые пути

```
float Calc(float x, float y)  {
    float H;
1   if (x*x+y*y+2<=0)
2   H = 17;
3   else H = 64;
4   return H*H+x*x;  }
```

Пример

```
/* Функция вычисляет неотрицательную
   степень n числа x */
1  double Power(double x, int n){
2  double z=1; int i;
3  for ( i = 1;
4        i <= n;
5        i++ )
6          { z = z*x; }  /* Возврат в п.4 */
7   return z;
   }
```



Управляющий граф программы

примеры путей: (3,4,7), (3,4,5,6,4,5,6), (3,4), (3,4,5,6)
примеры ветвей: (3,4) (4,5,6,4) (4,7)

Примеры необозримого множества входных значений

1. Если программа P(x:int, y:int) реализована в машине с 64-разрядными словами, то мощность множества тестов ||(X,Y)||=2**64 (для перебора при 1Ггц потребуется ~ 3К лет)
2. Программа управления схватом робота, где интервал между моментами срабатывания схвата не определен (пример требует прогона бесконечного множества последовательностей входных значений).

Основные проблемы тестирования

- *Тестирование* программы на всех входных значениях невозможно.
- Невозможно *тестирование* и на всех *путях*.
- Следовательно, надо отбирать конечный *набор тестов*, позволяющий проверить программу **на основе интуитивных представлений**
- **Требование к тестам** - программа на любом из них должна останавливаться, т.е. не зацикливаться.
  В теории алгоритмов доказано, что не существует общего метода для решения этого вопроса, а также вопроса, достигнет ли программа на данном тесте заранее фиксированного оператора.
- Задача о *выборе конечного набора тестов* (X,Y) для проверки программы в общем случае неразрешима.

Требования к идеальному критерию тестирования

- **Критерий должен быть достаточным**, т.е. показывать, когда некоторое конечное множество тестов достаточно для тестирования данной программы.
- **Критерий должен быть полным**, т.е. в случае ошибки должен существовать тест из множества тестов, удовлетворяющих критерию, который раскрывает ошибку.
- **Критерий должен быть надежным**, т.е. любые два множества тестов, удовлетворяющих ему, одновременно должны обнаруживать или не обнаруживать ошибки программы
- **Критерий должен быть легко проверяемым**, например вычисляемым на тестах

Для нетривиальных классов программ в общем случае **не существует полного и надежного критерия**, зависящего от программ или спецификаций.

Классы критериев

- *Структурные критерии* используют информацию о структуре программы (критерии "белого ящика")
- *Функциональные критерии* формулируются в описании требований к программному изделию (критерии «черного ящика")
- Критерии *стохастического тестирования* формулируются в терминах проверки наличия заданных свойств у тестируемого приложения средствами проверки некоторой статистической гипотезы.
- *Мутационные критерии*.

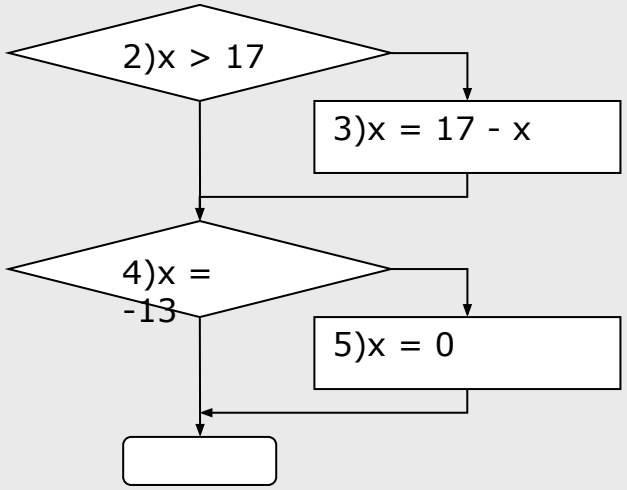
Структурные критерии

Используются на этапах модульного и интеграционного тестирования (Unit testing, Integration testing).

- **Тестирование команд** (критерий C0) - набор тестов в совокупности должен обеспечить прохождение каждой команды не менее одного раза.
- **Тестирование ветвей** (критерий C1) - набор тестов в совокупности должен обеспечить прохождение каждой ветви не менее одного раза.
- **Тестирование путей** (критерий C2) - набор тестов в совокупности должен обеспечить прохождение каждого пути не менее 1 раз. Если программа содержит цикл (в особенности с неявно заданным числом итераций), то число итераций ограничивается константой.
- **Тестирование условий** - покрытие всех булевских условий в программе. Критерии покрытия решений (ветвей - C1) и условий не заменяют друг друга, поэтому на практике используется комбинированный критерий покрытия условий/решений, совмещающий требования по покрытию и решений, и условий.

Пример

```
1  public void Method (ref int x) {
2  if (x>17)
3        x = 17-x;
4  if (x==-13)
5        x = 0;
6  }
```



критерий команд (**C0**):
(вх, вых) = {(30, 0)} - все операторы трассы 1-2-3-4-5-6

критерий ветвей (**C1**):
(вх, вых) = {(30, 0), (17, 17)}

критерий путей (**C2**):
(вх, вых) = {(30,0), (17,17), (-13,0), (21,-4)}

Условия операторов if

	(30,0)	(17,17)	(-13,0)	(21,-4)
`2 if (x>17)`	>	<=	<=	>
`4 if (x==-13)`	=	!=	=	!=

Недостаток структурных критериев

- Критерий ветвей C2 проверяет программу более тщательно, чем критерии - C1, однако даже если он удовлетворен, нет оснований утверждать, что программа реализована в соответствии со спецификацией.

Например, если спецификация задает условие, что|x|<100, невыполнимость которого можно подтвердить на тесте (-177,-177).

- ***Структурные критерии не проверяют соответствие спецификации,*** если оно не отражено в структуре программы. Поэтому при успешном тестировании программы по критерию C2 мы можем не заметить ошибку, связанную с невыполнением некоторых условий спецификации требований.

Функциональные критерии

- *Функциональный критерий* - важнейший для программной индустрии критерий тестирования. Он обеспечивает, прежде всего, контроль степени выполнения требований заказчика в программном продукте. Поскольку требования формулируются к продукту в целом, они отражают взаимодействие тестируемого приложения с окружением.
- При функциональном тестировании преимущественно используется модель "черного ящика" (см. остальную часть курса).
- Проблема функционального тестирования - это, прежде всего, трудоемкость.

Частные виды функциональных критериев

1. **Тестирование пунктов спецификации** - набор тестов в совокупности должен обеспечить проверку каждого тестируемого пункта не менее одного раза.
2. **Тестирование классов входных данных** - набор тестов в совокупности должен обеспечить проверку представителя каждого класса входных данных не менее одного раза.
3. **Тестирование правил** - набор тестов в совокупности должен обеспечить проверку каждого правила, если входные и выходные значения описываются набором правил некоторой грамматики.
4. **Тестирование классов выходных данных** - набор тестов в совокупности должен обеспечить проверку представителя каждого выходного класса
5. **Тестирование функций** - набор тестов в совокупности должен обеспечить проверку каждого действия, реализуемого тестируемым модулем, не менее одного раза ("полупрозрачный ящик«).
6. **Комбинированные критерии для программ и спецификаций**

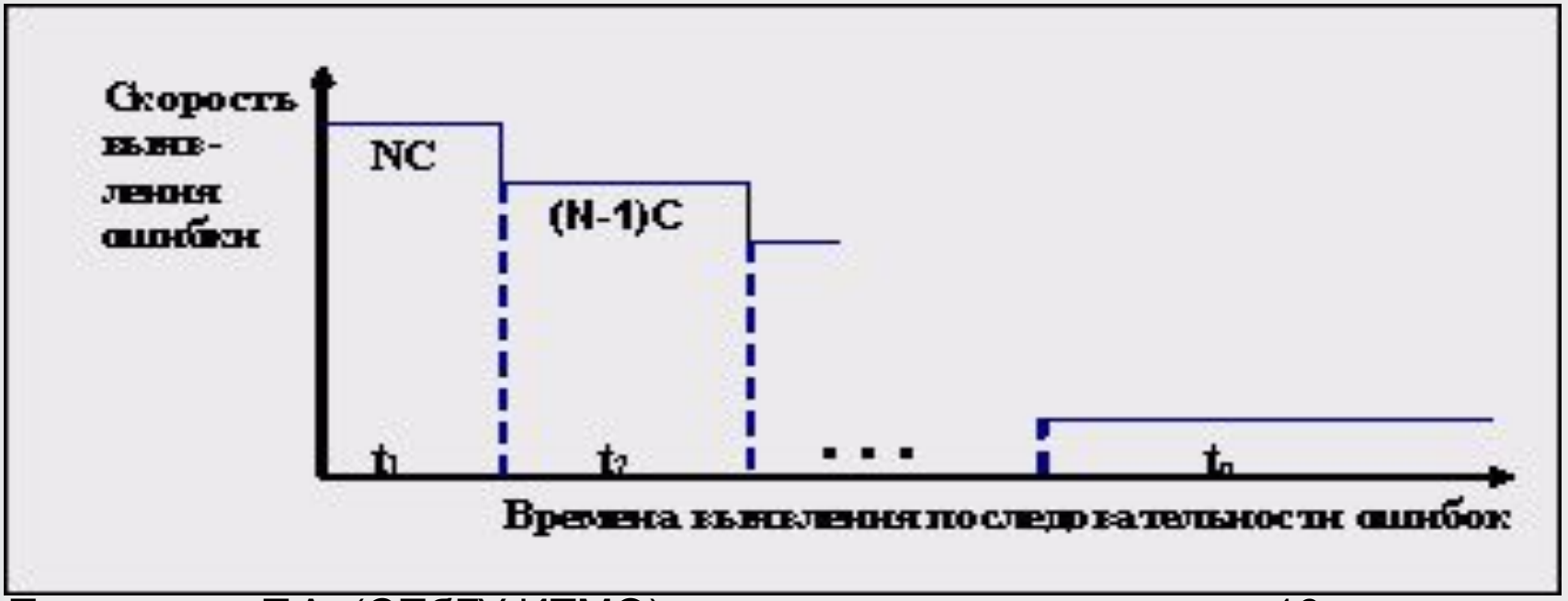
Стохастические критерии

Стохастическое тестирование применяется при тестировании сложных программных комплексов. Когда набор детерминированных тестов (X,Y) имеет громадную мощность и его невозможно разработать и исполнить, можно применить следующую методику:

- Разработать программы-имитаторы случайных последовательностей входных сигналов {x}.
- Вычислить независимым способом значения {y} для соответствующих входных сигналов {x} и получить тестовый набор (X,Y).
- Протестировать приложение на тестовом наборе (X,Y), используя два способа контроля результатов:
  - Детерминированный контроль - проверка соответствия вычисленного значения значению, полученному в результате прогона теста на наборе {x}.
  - Стохастический контроль - проверка соответствия множества значений, полученного в результате прогона тестов на наборе входных значений {x}, заранее известному распределению результатов F(Y). В этом случае множество Y неизвестно (его вычисление невозможно), но известен его закон распределения.

Статистические методы окончания тестирования

- Статистические методы окончания тестирования - стохастические методы принятия решений о совпадении гипотез о распределении случайных величин. К ним принадлежат: метод Стьюдента (St), метод Хи-квадрат (χ2) и т.п.
- Метод *оценки скорости выявления ошибок* - основан на модели скорости выявления ошибок, согласно которой тестирование прекращается, если оцененный интервал времени между текущей ошибкой и следующей слишком велик для фазы тестирования приложения.



Мутационный критерий

Подход базируется на следующих понятиях:

- ***Мутации*** - мелкие ошибки в программе.
- ***Мутанты*** - программы, отличающиеся друг от друга *мутациями*.

Метод мутационного тестирования - в разрабатываемую программу P вносят *мутации*, т.е. искусственно создают программы-мутанты P1, P2... Затем программа P и ее *мутанты* тестируются на одном и том же наборе тестов (X,Y).

- Если на наборе (X,Y) подтверждается правильность программы P и, кроме того, выявляются все внесенные в программы-мутанты ошибки, то **набор тестов (X,Y) соответствует** *мутационному критерию*, а тестируемая программа объявляется **правильной**.
- Если некоторые *мутанты* не выявили всех *мутаций*, то надо расширять набор тестов (X,Y) и продолжать тестирование.

Модульное тестирование (Unit testing)

- ***Модульное тестирование*** - это тестирование программы на уровне отдельно взятых модулей, функций или классов.
- Цель *модульного тестирования* состоит в выявлении локализованных в модуле ошибок в реализации алгоритмов, а также в определении степени готовности системы к переходу на следующий уровень разработки и тестирования.
- *Модульное тестирование* проводится по принципу "белого ящика".
- *Модульное тестирование* обычно подразумевает создание вокруг каждого модуля определенной среды

Принципы создания тестов

- На основе анализа ***потока управления***. В этом случае элементы, которые должны быть покрыты при прохождении тестов, определяются на основе структурных критериев тестирования C0, C1,C2. К ним относятся вершины, дуги, пути управляющего графа программы (УГП), условия, комбинации условий и т. п.
  К популярным критериям относятся *критерий покрытия функций* программы (каждая функция программы должна быть вызвана хотя бы один раз), и *критерий покрытия вызовов* (каждый вызов каждой функции в программе должен быть осуществлен хотя бы один раз).
- На основе анализа ***потока данных*** *(*элементы, которые должны быть покрыты, определяются на основе информационного графа программы).
  Этот вид направлен на выявление ссылок на неинициализи-рованные переменные и избыточные присваивания тестирования всех взаимосвязей, включающих в себя использование и определение переменной.
  Недостаток стратегии в том, что она не гарантирует покрытия решений.

Построение набора тестов

1. Конструирование УГП
2. Выбор тестовых путей:
   - Статические методы
   - Динамические методы
   - Методы реализуемых путей
3. Генерация тестов, соответствующих тестовым путям

Методы построения множества тестов

- **Статические методы.** Построение каждого пути посредством постепенного его удлинения за счет добавления дуг, пока не будет достигнута выходная вершина.
  Недостатки – не учитывается возможная нереализуемость построенных путей тестирования (непредсказуемый процент брака).
  - Трудоемкость (переход от покрывающего множества путей к полной системе тестов осуществляется вручную)
  Достоинство - сравнительно небольшое количество необходимых ресурсов
- **Динамические методы.** Построение полной системы тестов, удовлетворяющих заданному критерию, путем одновременного решения задачи построения покрывающего множества путей и тестовых данных. При этом можно автоматически учитывать реализуемость или нереализуемость ранее рассмотренных путей или их частей.
  Достоинство - некоторый качественный уровень - реализуемость путей.
- **Методы реализуемых путей.** Выделение из множества путей подмножества всех реализуемых путей, из которых строится покрывающее множество путей.

Сравнение методов

- Достоинство *статических методов* состоит в сравнительно небольшом количестве необходимых ресурсов. Однако их реализация может содержать непредсказуемый процент брака (нереализуемых путей). Кроме того, в этих системах переход от покрывающего множества путей к полной системе тестов пользователь должен осуществить вручную (трудоемко).
- *Динамические методы* требуют значительно больших ресурсов как при разработке, так и при эксплуатации, однако увеличение затрат происходит, в основном, за счет разработки и эксплуатации аппарата определения реализуемости пути (символический интерпретатор, решатель неравенств). Достоинство этих методов заключается в том, что их продукция имеет некоторый качественный уровень - реализуемость путей. Методы реализуемых путей дают самый лучший результат.

Методы отладки

- Результат выполнения теста ничего не говорит о том, где была допущена ошибка.
- Процедура исправления ошибки заключается в анализе протокола промежуточных вычислений с помощью следующих *методов*:
  - "Выполнение программы в уме" (deskchecking).
  - Вставка операторов протоколирования промежуточных результатов (logging).
  - Пошаговое выполнение программы.
  - Выполнение с заказанными *остановками (breakpoints)*, анализом трасс (traces) или состояний памяти - *дампов (dump)*.
  - обратное выполнение (reversible execution) – возможно в режиме off-line анализа при фиксации в log–файле всей истории выполнения трассы.