ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

РАЗДЕЛ 5

ОСНОВЫ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ ТЕСТИРОВАНИЯ

Программа – это аналог формулы в обычной математике.

Формула для функции **f, полученной суперпозицией функций** $f_1 * f_2 * f_3 * \ldots * f_n$ – выражение, описывающее эту суперпозицию. Если аналог $f_{1\ldots}\ f_n$ – операторы языка программирования, то их формула – **программа**.

Существует 2 метода обоснования истинности формул:

i) Формальный подход или **Доказательство** применяется, когда из исходных **формул-аксиом** с помощью **формальных процедур** (правил вывода) выводятся искомые формулы и утверждения (теоремы). Вывод осуществляется путем перехода от одних формул к другим по строгим правилам, которые позволяют свести процедуру перехода от формулы к формуле к **текстовой подстановке**:

A3 Ю A*A*A Ю A → R, R → A*R, R → A*R**

С помощью формального подхода удается избежать обращения к бесконечной области значений и на каждом шаге доказательства оперировать только с конечным множеством символов.

ii) Интерпретационный подход применяется, когда осуществляется **подстановка констант** в формулы, а затем интерпретация формул, как осмысленных утверждений в элементах конкретных множеств значений. На конечных множествах проверяется **истинность интерпретируемых формул на множестве значений**

Интерпретационный подход используется при экспериментальной проверке соответствия программы своей спецификации

ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ ТЕСТИРОВАНИЯ

Отладка (debug, debugging) – процесс поиска, локализации и исправления ошибок в программе [IEEE Std.610-12.1990].

Отладка состоит из Ю поиска ошибок (тестирования) + исправления ошибок

Тестирование обеспечивает констатацию наличия ошибок, если ошибки имеются.

Исправление (собственно отладка) обеспечивает локализацию ошибок, нахождение причин ошибок и соответствующую корректировку программы

Если программа не содержит синтаксических ошибок (прошла трансляцию) и следовательно может быть выполнена на компьютере, она обязательно вычисляет какую-либо функцию: осуществляет отображение входных данных в выходные

Следовательно компьютер на своих ресурсах доопределяет частично определенную программой функцию до тотальной определенности. Т.о. судить о правильности или неправильности результатов выполнения программы можно только, сравнивая спецификацию желаемой функции с результатами ее вычисления, что и осуществляется в процессе тестирования.

Тестирование разделяют на статическое и динамическое:

Статическое тестирование выявляет неверные конструкции (ошибки) без выполнения программы формальными методами статического (в т.ч. Синтаксического) анализа (CodeCheker)

Динамическое тестирование выявляет ошибки только в процессе выполнения программы

ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ ТЕСТИРОВАНИЯ

Организация тестирования. Тестирование осуществляется на заданном заранее множестве входных данных **X** и множестве предполагаемых результатов **Y** – (**X,Y**), которые задают график некоторой желаемой функции. Кроме того зафиксирована процедура **Оракул**, которая определяет соответствуют ли выходные данные – **Yв,** вычисленные по входным данным – **X,** желаемым результатам **Y,** т.е. принадлежит ли каждая вычисленная точка (**x,yв**) графику желаемой функции (**X,Y**).

Оракул дает заключение о факте появления неправильной пары (**x,yв**) и ничего не говорит о том, как она вычислялась или каков правильный алгоритм – он только сравнивает вычисленные и желаемые результаты. Оракулом м.б. даже Заказчик или программист производящий ручные вычисления, поскольку Оракулу нужен какой-либо другой способ получения функции (**X,Y**) для вычисления эталонных значений **yOY.**

В процессе тестирования Оракул последовательно получает элементы множества (**X,Y**) и соответствующие им результаты вычислений **Yв** для поиска несовпадений При выявлении (**x,yв**) П(**X,Y**) запускается процедура исправления ошибки, которая заключается в внимательном анализе (просмотре) протокола промежуточных вычислений, приведших к (**x,yв**), с помощью следующих средств:

ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ ТЕСТИРОВАНИЯ

1) «выполнение программы в уме» (**deskchecking**),
2) вставка операторов протоколирования (печати) промежуточных результатов (**logging**),
3) пошаговое выполнение программы (**single-step operation**),
4) выполние с заказанными остановками (**breakpoints**) и анализом трасс (**traces**) и дампов (**dump inspection**),
5) реверсивное (обратное) выполнение (**reversible execution**).

Сравнение промежуточных результатов с эталонными (полученными независимо) позволяет найти причины и место ошибки, исправить текст программы, провести повторную трансляцию и настройку на выполнение и продолжить тестирование.

Тестирование заканчивается, когда **достаточное** в соответствии с выбранным **критерием количество тестов** прошло (pass) успешно.

ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ ТЕСТИРОВАНИЯ

Тестирование - процесс функционирования системы или компоненты в заранее определенных условиях с записью и анализом результатов функционирования и оценкой свойств тестируемого объекта.
[IEEE Std.610-12.1990]
[Катков В.Л, Шимаров В.А.] - контролируемое выполнение программы на конечном множестве тестовых данных и анализ результатов этого выполнения для поиска ошибок.

Три фазы тестирования:

1) составление (генерация) тестов,

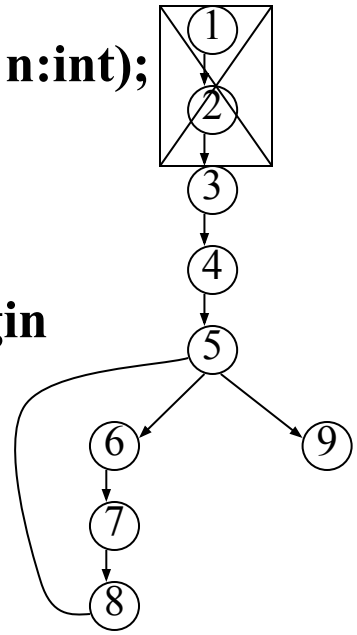
2) прогон программы на тестах,

3) оценка результатов выполнения программы на тестах

Основная проблема тестирования - определение достаточности множества тестов для истинности вывода о правильности реализации программы и нахождении этого множества тестов.

ГРАФОВАЯ МОДЕЛЬ ПРОГРАММЫ

```
1 function Z(x:real, n:int);
2 i:int
3 Z:=1
4 i:=1
5 M: if n>i then begin
6 Z:=Z*x
7 i:=i+1
8 goto M end
9 end
```



Граф программы - G(V,A), где $V(V_1,\dots V_m)$ – множеств вершин (операторов), $A(A_1,\dots A_n)$ – множество дуг (управлений), соединяющих операторы-вершины

Путь – последовательность вершин и дуг, в которой любая дуга выходит из вершины V_i и приходит в вершину V_j, например: (3,4,5,9) (3,4,5,6,7,8,5,6,7,8) (3,4,5) (3,4,5,6,7,8)

Ветвь – путь $(V_1, V_2, \dots V_k)$, где V_1 - либо первый либо условный оператор программы, V_k - либо условный оператор либо оператор выхода из программы, а все остальные операторы – безусловные, например: (3,4,5) (5,6,7,8) (5,9)

Пути, различающиеся хотя-бы числом прохождений цикла – разные пути, поэтому **число путей** в программе м.б. **не ограничено**. Ветви - линейные участки программы, **число ветвей конечно** .

ГРАФОВАЯ МОДЕЛЬ ПРОГРАММЫ

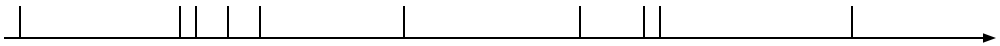
```
function H(x:real,y:real);
1 if x**2+y**2+2<=0
2 then H:=17
3 else H:=64
4 H:=H**2+x**2
```

Существуют **реализуемые и нереализуемые** пути в программе, в нереализуемые пути нельзя попасть в обычных условиях:
H(x.y): путь (1,3,4) реализуем, путь (1,2,4) нереализуем в условиях нормальной работы, но не при сбоях

ОСНОВНЫЕ ПРОБЛЕМЫ ТЕСТИРОВАНИЯ

i) Пусть программа **H(x:int, y:int)** реализована в машине с 64 разрядным словом, тогда мощность множества тестов
||(X,Y)||=2128 ~ 10**38 тестов**
(1GHz ; 1 тест=1ms ~ 10**32s ~ 25 лет) 1

ii) Программа работы схвата робота:
M1: ЧтДатчика (вкл:Bool); if вкл then goto M2 else M1;
M2: ОткрытьСхват; ЗакрытьСхват; goto M1
Это тривиальный пример, имеющий бесконечное множество последовательностей входных значений



Тестирование программы на всех возможных входных значениях невозможно, невозможно и тестирование на всех путях.
Следовательно надо отбирать конечный набор тестов, позволяющий хорошо проверить программу по нашим интуитивным представлениям

ОСНОВНЫЕ ПРОБЛЕМЫ ТЕСТИРОВАНИЯ

Требование к тестам - программа на любом из них должна останавливаться, т.е. не зацикливаться.

Можно ли заранее **гарантировать останов на любом тесте** ?

Теория алгоритмов: Не существует общего метода для решения этого вопроса, более того и вопроса достигнет ли программа на данном тесте заранее фиксированного оператора.

Задача о выборе конечного набора тестов (X,Y) для проверки программы **неразрешима в общем случае** Ю ищем частные случаи решения задачи.

КРИТЕРИИ ВЫБОРА ТЕСТОВ

Требования к идеальному критерию тестирования [Goodenough J.B., Gerhart S.L.] :

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

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

КРИТЕРИИ ВЫБОРА ТЕСТОВ

КЛАССЫ КРИТЕРИЕВ

i. Структурные критерии

ii. Функциональные критерии (опираются на описание задачи)

iii. Критерии стохастического тестирования

iv. Мутационные критерии

СТРУКТУРНЫЕ КРИТЕРИИ (класс i)

- **Тестирование команд (C0, P_1)-** набор тестов в совокупности должен обеспечить прохождение каждой команды (каждого оператора) не менее 1 раза (используется в больших программных системах, где другие критерии не применить)
- **Тестирование ветвей (C1, P_2)-** набор тестов в совокупности должен обеспечить прохождение каждой ветви не менее 1 раза (распространен в Авт.Системах Тестир.)
- **Тестирование путей (C2, P∞)-** набор тестов в совокупности должен обеспечить прохождение каждого пути не менее 1 раз. Если программа содержит цикл, то число итераций ограничено (часто - 2, или числом выходных классов путей)

ПРИМЕР
1 Proc PR(x:int); begin
2 If x > 17 then
3 x:=17-x;
4 If x= -13 then
5 x:=0;
6 end

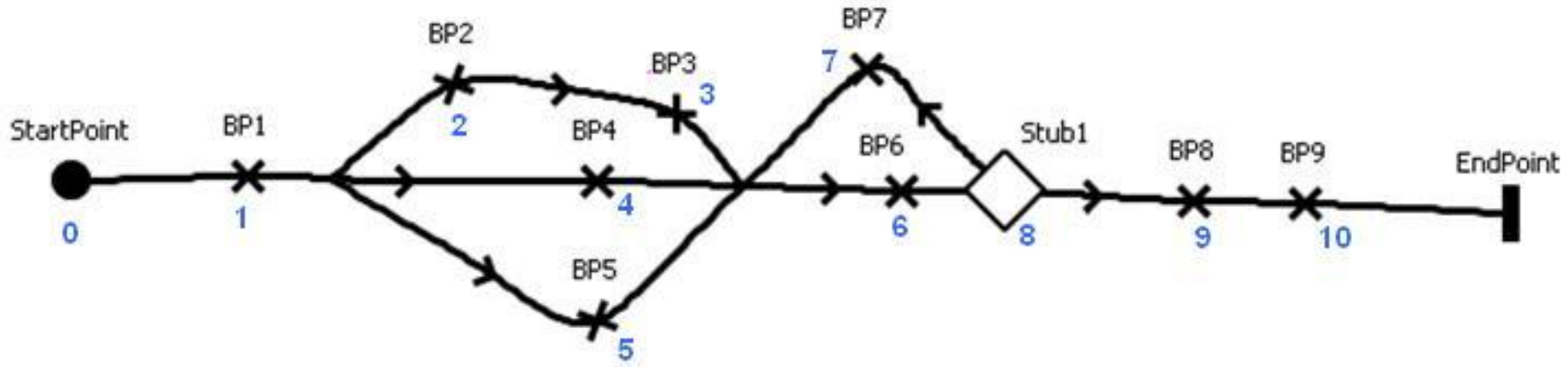
Критерий C0 (X,Y)={($x_{вх}$=30, $x_{вых}$=0)} 1-2-3-4-5-6
Критерий C1 (X,Y)={(30,0), (17,17)} +1-2-4-6
Критерий C2 (X,Y)={(30,0), (17,17), (-13,0), (21,-4)}

2 If > J J >
4 If = № = №

C2 проверяет программу более тщательно, чем C1,C2, но не проверяет соответствие со спецификациями, если оно не отражено в структуре программы.

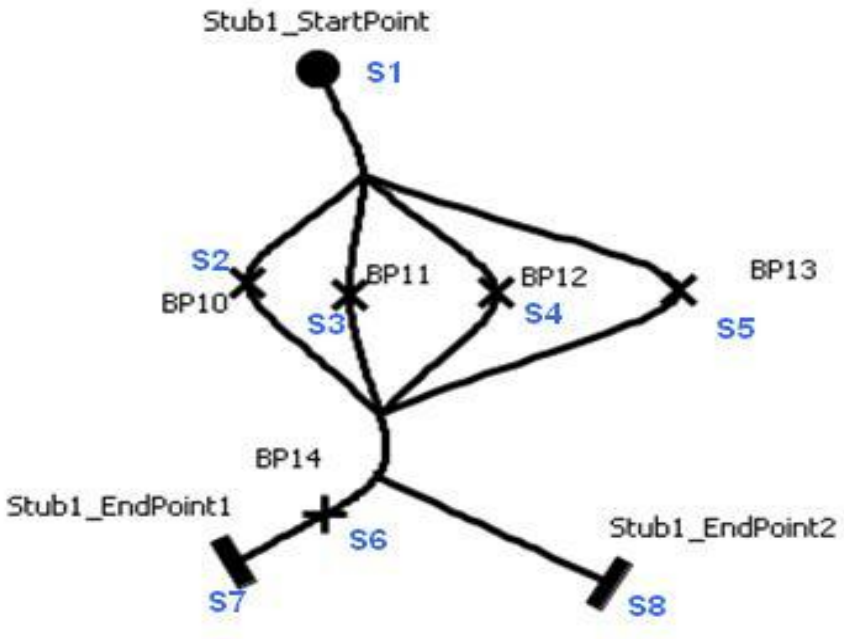
Если задано, что |x| J100, то операторы 3 и 5 на тесте (-177,-177) не изменят величину x=-177 и результат не соответствует спецификации.

Пример использования структурного критерия



3 трассы 1-2-3-6-8-9-10, 1-4-6-8-9-10, 1-4-6-8-7-6-8-9-10 и 1-5-6-8-9-10, построенные по графу (UCM диаграммы), гарантируют полное покрытие ветвей и частичное покрытие путей при условии однократного прохода цикла.

Структурный критерий для поддиаграмм



Если трассы содержат абстрактные фрагменты с описанием поведения, (например, Stub1 - 8), то трассы, представляющие поведение внутри элемента Stub, могут быть получены отдельно и затем добавлены в итоговую трассу с детальным поведением.

Например, поведение элемента, представленного диаграммой Stub1 описывается следующими трассами: S1-S2-S6-S7, S1-S3-S6-S7, S1-S4-S6-S7, S1-S5-S6-S7, а также идентичным набором трасс, оканчивающихся элементом S8 вместо последовательности элементов S6-S7. При подстановке детального поведения Stub1 в трассу, описывающую полное поведение системы, вход 6 в Stub1 соединяется с элементом S1, выход S7 – с элементом 9, выход S8 – с элементом 7 на рис. 3.

ФУНКЦИОНАЛЬНЫЕНЫЕ КРИТЕРИИ (класс ii)

- **Тестирование пунктов спецификаций** - набор тестов в совокупности должен обеспечить проверку каждого тестируемого пункта не менее 1 раза
- **Тестирование классов входных данных** - набор тестов в совокупности должен обеспечить проверку представителя каждого класса (нормальных/ошибочных типов)
  -- текстовые входные файлы из разных источников
- **Тестирование правил** - набор тестов в совокупности должен обеспечить проверку каждого правила, если входные значения описываются набором правил грамматики
  -- грамматика д.б. достаточно простой
- **Тестирование классов выходных данных** - набор тестов в совокупности должен обеспечить проверку представителя каждого выходного класса, если результаты заранее расклассифицированы
  -- проверяются в том числе ограничения на ресурсы, тайм-ауты
- **Тестирование функций** - набор тестов в совокупности должен обеспечить проверку каждого действия, реализуемого программой не менее 1 раза
- **Комбинированные критерии для программ и спецификаций** - набор тестов в совокупности должен обеспечить проверку всех комбинаций непротиворечивых условий программ или спецификаций.
  -- все комбинации условий надо проверить,а противоречивые обнаружить и убрать

Пример документа TRS, описываю щего требования

Identifier	*Requirements*
TRS_001	The Application shall provide the user with the test description before Camera Test execution. Приложение должно предоставить пользователю описание теста Камеры перед выполнением этого теста
TRS_002	During the Camera Test the Application shall prompt (спросить) the user to capture the image, then display the «current camera view» through the view finder (видоискатель). Во время теста Камеры Приложение должно побудить (спросить) пользователя захватить изображение, затем показать «текущее представление камеры» через отображение видоискателя на экране.
TRS_003	During the Camera Test as soon as the image has been captured, the Application shall present the image and prompt the user with the test question to confirm or reject the result of the test (e/n – наск хор камера снимает). Во время теста Камеры приложение должно побудить пользователя захватить изображение, а затем отбразить «текущее вид, наблюдаемый камерой» через видоискатель.
TRS_004	The user shall be able to confirm and reject the result of the Camera Test in response to the test question. Пользователь должен подтвердить или отклонять результат теста Камеры в ответ на соответствующий вопрос
TRS_005	The result of the Camera Test shall be evaluated by the Application as follows: confirmed by the user - the test has been passed, rejected by the user - the test has been failed. Результат теста Камеры должен быть оценен Приложением следующим образом: тест, подтвержденный пользователем, является успешным (passed) - тест отклонный пользователем – неуспещным (failed).
TRS_006	The user shall be able to cancel the ongoing Camera Test session while the Application is running in foreground (запущено). Пользователь может отменить незаконченную сессию теста Камеры, во время исполнения приложение, которое отображается на переднем плане экрана дисплея
TRS_007	If the cancellation of the Camera Test session has been invoked by the user, the Application shall confirm with the user prior to actual session cancellation (действительно хочешь cansel). Если отмена (concellation) сессии теста Камеры была вызвана пользователем, то Приложение должно получить подтверждение на Concel от пользователя еще до фактической отмены сессии.
TRS_008	If the cancellation hasn't been confirmed by the user, previous screen shall appear. Если отмена (cancellation) не была подтверждена пользователем, предыдущий экран должен появиться.

СТОХАСТИЧЕСКИЕ КРИТЕРИИ (класс iii)

-**Тестирование сложных программных комплексов (Кпрг) -** когда набор тестов (X,Y) имеет громадную мощность. Разрабатываем программы - имитаторы последовательностей входных сигналов {x}, для которых вычисляем {y}. Контолируем:

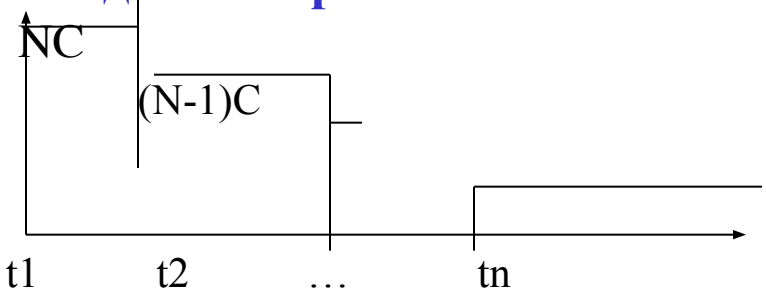
1) соответствие $y=y_{эталонному}$ - детерминированный случай,

2) соответствие {y} - известному распределению F(y) - стохастический случай

-- **Критерии прекращения стохастического тестирования**

-- **статистические методы принятия решений** о совпадении гипотез о распределении случайных величин - St, χ^2

-- **модель скорости выявления ошибок** [P.B.Moranda]



N - исходное число ошибок в Кпрг перед тестированием

C - константа снижения скорости выявления ошибок за счет нахождения очередной ошибки

$t_1,t_2,\ldots t_n$ - кортеж возрастающих интервалов обнаружения ошибок, если за T выявлено n ошибок

СТОХАСТИЧЕСКИЕ КРИТЕРИИ (класс iii) - продолжение

Пусть за T выявлено n ошибок:

(1) $(N-i+1)*C*t_i = 1$ – (скорость NC)*(время_выявления_i_ошибки) есть 1 по определению

(2) $NCt_1+(N-1)Ct_2+\ldots+(N-n+1)Ct_n=n$

$NC(t_1+t_2+\ldots+t_n) - C* e (i-1)t_i = n$

$NCT - C* e (i-1)t_i = n$

(3) $e\ 1/(N-i+1) = TC$ -- из (1) определено $C*t_i$ и просуммировано от 1 до n

(4) $C = n/(NT - e (i-1)t_i)$ -- выражаем из (2)

(5) $e\ \ 1/(N-i+1) = n/(N - 1/T* e\ \ (i-1)t_i)$ -- подставляем C в (3)

Оцениваем величину N приблизительно (методы оценки числа ошибок в программе), наблюдаем $t_1,t_2\ldots t_n$ и по (5) находим t_{n+1}.

Если t_{n+1} > Тдопустимого, то тестирование заканчиваем.

Наблюдая $t_1,t_2\ldots t_n$, T= e t_i, можно прогнозировать время следующего шага и уточнять величину C.

МУТАЦИОННЫЙ КРИТЕРИЙ (класс iv)

Мутации - мелкие ошибки в программе
Мутанты - программы, отличающиеся друг от друга мутациями
Метод мутационного тестирования - в разрабатываемую программу P вносят мутации, т.е. создают мутантов P_1,P_2... Затем программа P и ее мутанты тестируются на одном наборе тестов (X,Y).

Если на (X,Y) подтверждена правильность программы и выявлены все ошибки мутантов, то НАБОР ТЕСТОВ (X,Y) СООТВЕТСТВУЕТ мутационному критерию, а тестируемая программа - ПРАВИЛЬНА

Если некоторые мутанты не выявили всех мутаций, то надо расширять набор тестов (X,Y)