Тестирование

Технологии разработки программного обеспечения

Виноградова М.В.
МГТУ им. Н.Э. Баумана
Кафедра СОИУ (ИУ5)

Тестирование ПО

- Процесс выполнения ПО с целью обнаружения ошибок.
- Шаги процесса задаются тестами.
- Тест (Тестовый вариант):
  - набор исходных данных + условия запуска ПО,
  - набор ожидаемых результатов работы ПО.

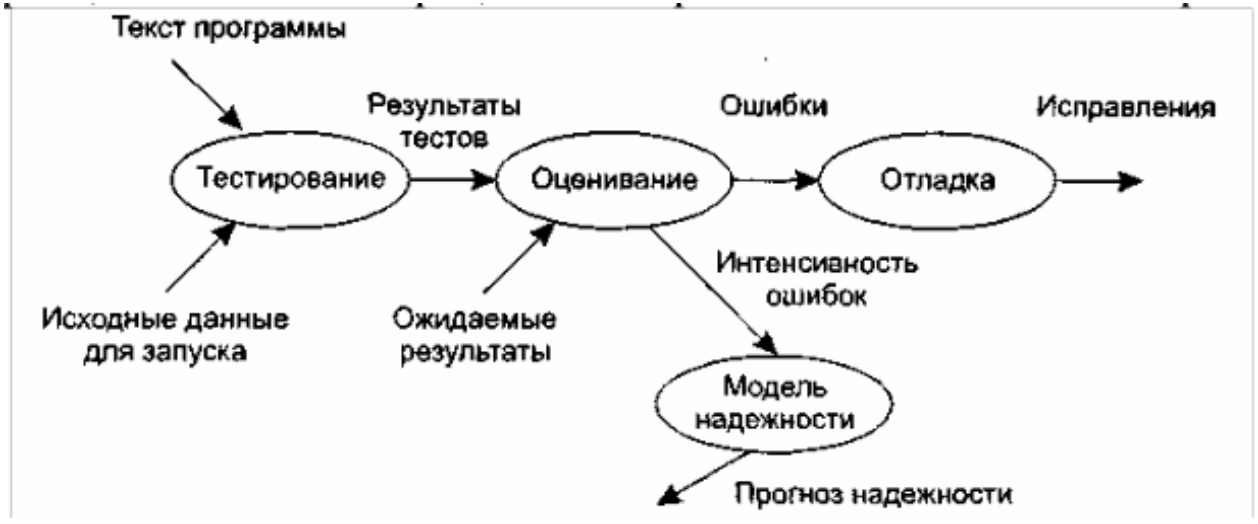
- Исчерпывающее тестирование
  - проведение всех возможных тестовых вариантов (все исходные данные, все варианты обработки);
  - невозможно из-за больших ресурсов.

- Хороший тестовый вариант
  - с большой вероятностью обнаружение новой ошибки.
- Успешный тест – нашел новую ошибку.

- Цель разработки тестов
  - систематическое обнаружение различных классов ошибок при min затратах времени и стоимости.

Возможности тестирования

- Тестирование может:
  - Найти ошибку;
  - Установить соответствие функции по ее назначению;
  - Демонстрировать реализацию требований к характеристикам ПО;
  - Демонстрировать качество ПО (через надежность).
- Тестирование не может:
  - Доказать отсутствие ошибок.
- Много ошибок -> усилить тестирование.
- Мало ошибок -> а) надежное ПО; б) плохие тесты.

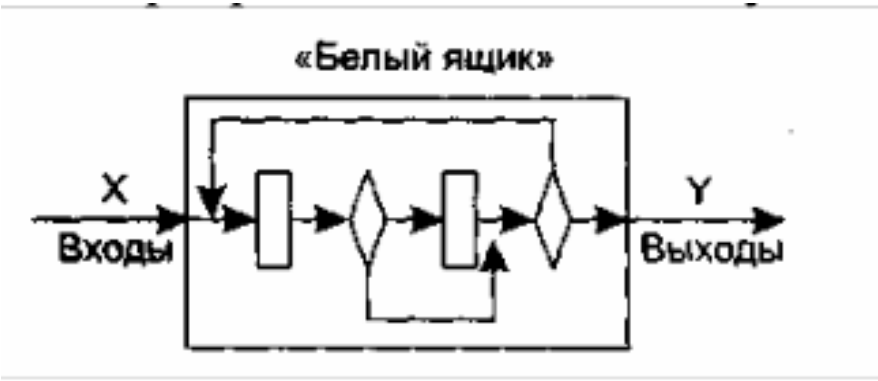


Принципы тестирования

- Функциональное – черный ящик
  - Позднее тестирование, модульное и ПО.
  - Известно: функции ПО.
  - Исследование: работа каждой функции на ее области определения.
- Структурное – белый ящик
  - Раннее тестирование, модульное.
  - Известно: внутренняя структура ПО.
  - Исследование: внутренние элементы ПО и связи между ними.

Структурное тестирование

- Объект тестирования:
  - внутреннее поведение ПО;
  - корректность построения всех элементов ПО и правильность их взаимодействия;
- Анализ управляющих и информационных (реже) потоков.
- Характеризуется степенью покрытия логики (исходного кода) ПО.
- Исчерпывающее тестирование затруднительно.



Особенности тестирования «Белого ящика»

- Анализ управляющей структуры ПО.
- ПО полностью проверено <= исчерпывающее тестирование путей (маршрутов) ее графа управления.

- Тестовые варианты:
  - Гарантируется проверка всех независимых маршрутов ПО;
  - Проходятся ветви True, False для всех логических решений.
  - Выполняются все циклы (в пределах их границ).
  - Анализируется правильность внутренних структур данных.

Результаты «Белого ящика»

- Плюсы - Учитывает особенности ошибок:
  - (Алгоритм → ПО) => синтаксические и семантические ошибки
  - Предположения о вероятности потока управления/ данных могут быть неправильными => типовой путь слабо проработан.
  - Количество ошибок min в центре, max на границе ПО;
  - Результаты ПО могут зависеть не от входных данных, а от внутренних состояний ПО.

- Минусы:
  - Много независимых путей;
  - Цикл – k (=20) раз; в цикле n (=5) ветвлений → путей = 1014 (3170 лет)
  - Исчерпывающее тестирование не гарантирует выполнение функций ПО;
  - Могут быть пропущены пути в ПО;
  - Нельзя найти ошибки, которые зависят от исходных данных.

Тестирование базового пути

- Том Мак Кейб (1976 г.)
- Цель:
  - оценка комплексной сложности ПО => количество тестовых вариантов (для тестирования базового множества путей).
- Гарантия
  - однократного выполнения каждого оператора ПО при тестировании;
  - выполнение каждого условия по True/False ветви.

Потоковый граф

Отображает управленческую структуру ПО:

- Узел (вершина) = линейный участок ПО, >= 1 оператора.
- Дуга = передача управления.
- Операторный узел – линейный код.
- Предикатный узел - простое условие (T/F).
- Составное условие делится на простые, отображается в N предикатных узлов.
- Закрывающие скобки условия/цикла задают операторы (фиктивные).
- Замкнутая область из дуг/узлов = регион.
- Окружающая среда = дополнительный регион.

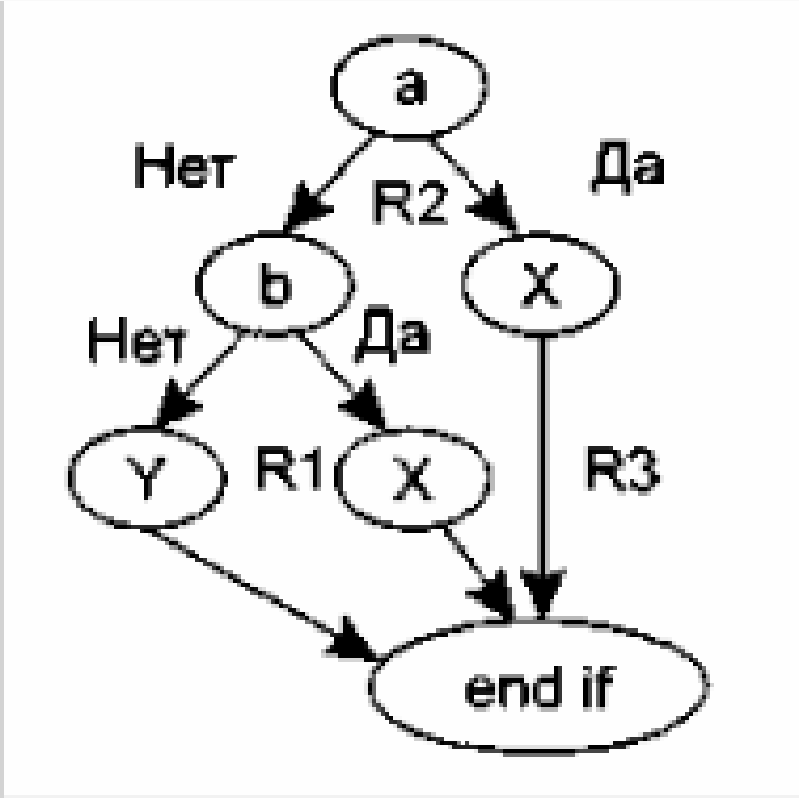
Пример потокового графа

If a OR b then x

else y

end if

R1, R2, R3 – регионы.



Мощность базового множества

- Мощность базового множества = <u>цикломатической сложности потокового графа:</u>
  - а) Количество независимых путей;
  - б) однократное выполнение всех операторов (верхняя граница);
  - в) логическая сложность ПО.
- Вычисление:
  - а) Количеству регионов потокового графа.
  - б) V = E - N+2, где E-количество дуг, N- количество вершин.
  - в) V = p+1, где p – количество предикатных узлов в потоковом графе.

Пример кода

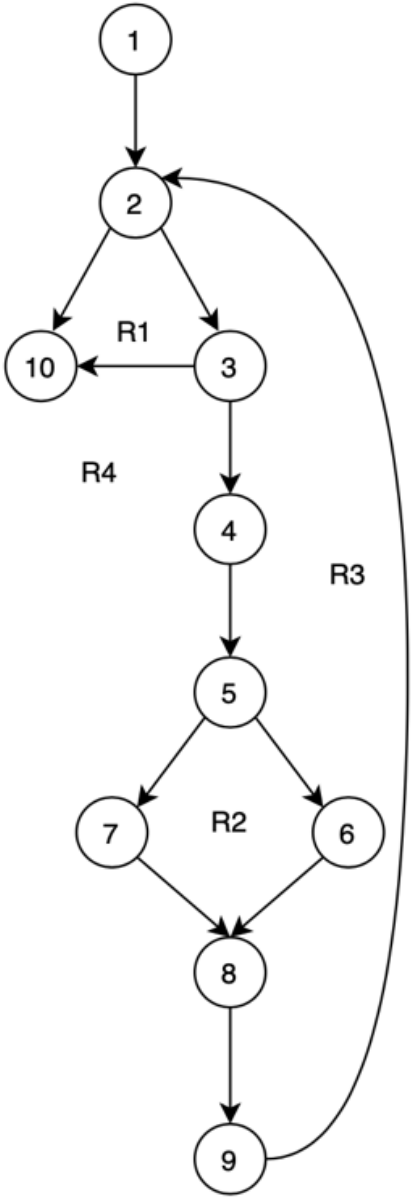
```
i = 0
s = 0
b = 0

Пока (i<10) AND
          (bb(i)≠0)
    i++
    if bb(i)>0
          s++
    else
          n++
     end if
 end пока

вывод(n)
вывод (s)
```

```
i = 0                (1)
s = 0                (1)
b = 0                (1)
          (2)          (3)
Пока (i<10) AND (bb(i)≠0)
    i++              (4)
    if bb(i)>0       (5)
          s++        (6)
    else
          n++        (7)
     end if          (8)
 end пока            (9)

вывод(n)             (10)
вывод (s)            (10)
```



Методика тестирования базового пути

- Шаг 1. Создается потоковый граф
  - Нумерация операторов.
  - Отображение операторов в граф.
- Шаг 2. Цикломатическая сложность
  - а) V(G) = 4 региона
  - б) 12-10+2 = 4
  - в) 3+1 = 4
- Шаг 3. Базовое множество независимых путей
  - а) 1-2-10 (обр. 11-го)
  - б) 1-2-3-10 (i=0, bb=[0])
  - в) 1-2-3-4-5-6-8-9-2… (i>0, bb[i]>0)
  - г) 1-2-3-4-5-7-8-9-2… (i>0, bb[i]<0)

Методика тестирования базового пути - 2

- Подготовка тестовых вариантов для каждого независимого пути:
- а)
  - ИД: bb(1,2,3,4,-5, 6,7,-8,9,10, 11)
  - ОЖ.РЕЗ.: n = 2, S = 8, i = 10
- б)
  - ИД: bb(0)
  - ОЖ.РЕЗ.: n = 0, S=0, i = 0
- в)
  - ИД: bb (1,2,3,4,-5,0)
  - ОЖ.РЕЗ.: n = 1, S = 4, i = 5
- г)
  - ИД: bb (-1,-2,-3,4,-5,0)
  - ОЖ.РЕЗ.: n = 4, S = 1, i = 5

- Существуют независимые пути, являющиеся частью другого пути.

Тестирование условий

- Цель:
  - проверка логических условий ПО (охват всех ветвей ПО).
- Простое условие (ПУ)
  - булевы переменные или выражение отношений.
- Выражения отношений
  - E1 <оператор отношения> E2
  - E1, E2 – арифметические выражения .
  - Оператор отношения: >,=,<,
- Составное условие (СУ) состоит из объединения:
  - простых условий;
  - булевых операторов (AND, OR, NOT);
  - ( ).
- Условие без выражения отношения - это булевое выражение.

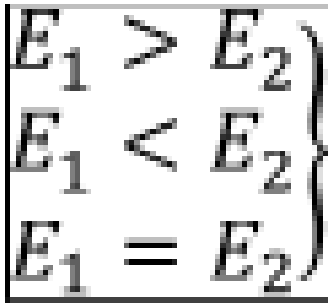
Ошибки в условиях:

- Виды ошибок:
  - Булевого оператора (некорректный/отсутствующий/избыточный).
  - Булевой переменной.
  - Булевой ( ).
  - Оператора отношения.
  - Арифметического выражения.

- Тест каждого условия в ПО. Плюсы:
  - оценка тестового покрытия условия,
  - генерация дополнительных тестов.
- Ошибки в условии => другие ошибки в ПО.

Методики тестирования условий

- Тестирование ветвей
  - Для составного условия С проверка:
    - а) каждое простое условие в нем
    - б) TRUE (ветви)
    - в) FALSE (ветви)
- Тестирование области определения:
  - Для E1 <оператор отношения> E2:

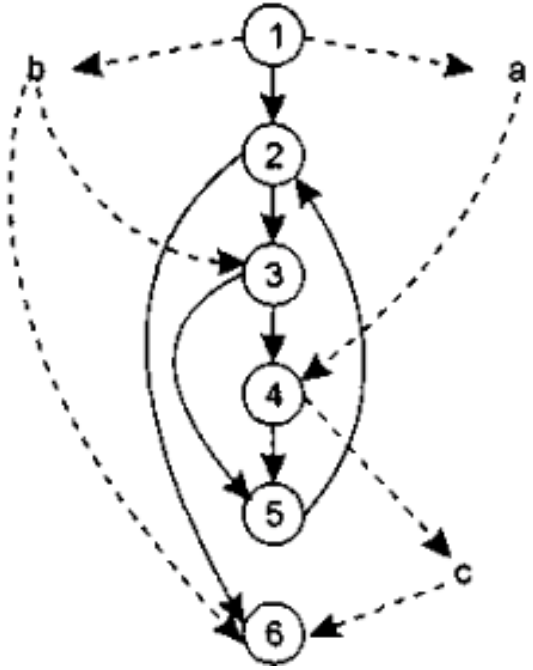


$\Rightarrow$ ошибка в операторе (при корр. E1, E2) | при min(|E1-E2|)

n переменных=> 2 в степени n тестов. Эффективно при малом n.

Тестирование потоков данных

- Анализ информационной структуры ПО.
- ПО = обработка потока данных от входа до выхода.
- Потоковый граф + информационные связи



в 1 определяется значение a, b
в 6 используются значения b, c

Множество <u>определений данных</u>:

DEF(i) = { x, i -> x }

Изменение данных: x = f (…)

Множество <u>использований данных</u>:

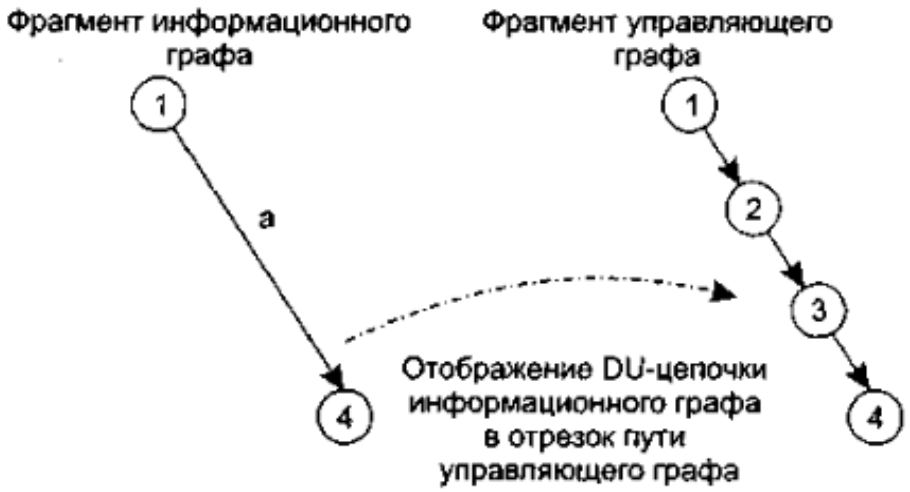
use(i) = { x, i <- x }

обращение без изменения данных:

[ ] = f(x)

DU тестирование

- DU цепочка
  - цепочка определения-использования - [x, i, j]
- DU-цепочки - пример:
  - [a,1,4],[b,1,3],[b,1,6],[c,4,6]
- DU тестирование
  - охват всех DU-цепочек(анализ ЖЦ всех данных ПО)
  - => необходимы пути => max покрытие DU-цепочек



Шаги DU тестирования

1. Построить управляющий граф ПО (УГ);
2. Построить информационный граф (ИГ);
3. Определить полный набор DU-цепочек;
4. Отобразить DU-цепочки в отрезки путей управляющего графа;
5. Построить пути на УГ, покрывающие набор отрезков путей;
6. Определить тестовые варианты.

Особенности DU тестирования

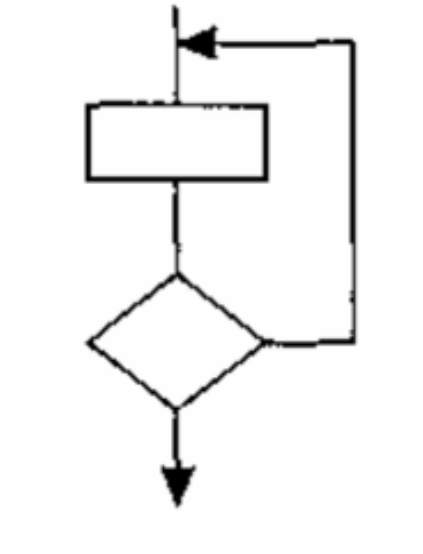
- Плюсы:
  - Простой анализ ПО;
  - Простота автоматизации.
- Минусы:
  - Трудно выбрать min количество max эффективных тестов.
- Область использования:
  - ПО с вложенными условиями и циклами.

Тестирование циклов

- Виды циклов:
  - Простые
  - Вложенные
  - Объединенные
  - Неструктурированные

Тестирование простых циклов

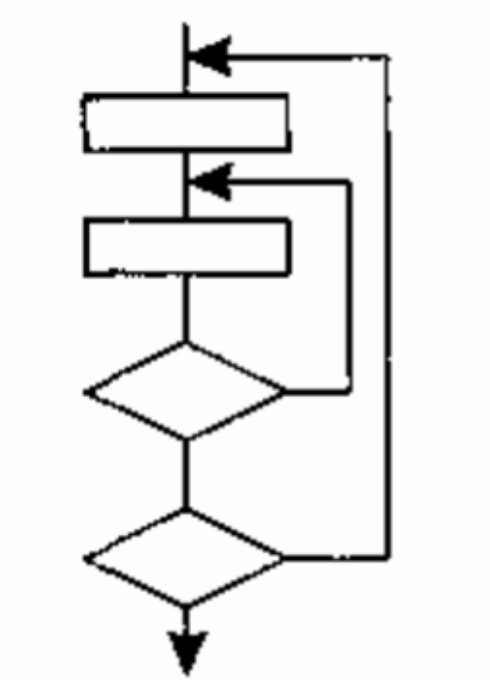
- n – количество повторов.



- Выбирают 1 из методов:
  - Прогон всего цикла,
  - 1 проход,
  - 2 прохода,
  - M<n проходов,
  - (n-1), n, (n+1) проходов.

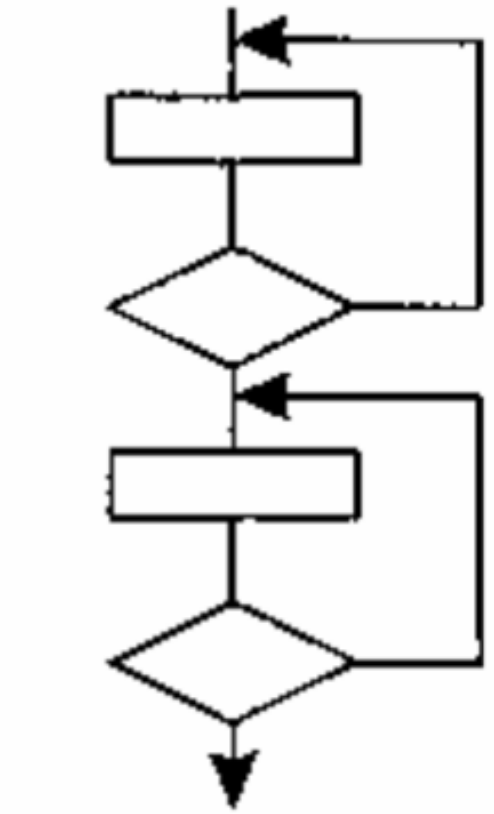
Тестирование вложенных циклов

1. Внутрений цикл – min параметры для остальных циклов.
2. Для внутреннего цикла тесты простого + тесты для исключенных значений и значений за рабочим диапазоном.
3. Следующий объемлющий цикл. Его тест при Min значениях параметров для внешних циклов; типовые значения для вложенных циклов.
4. И так далее, от внутренных к вшешним.



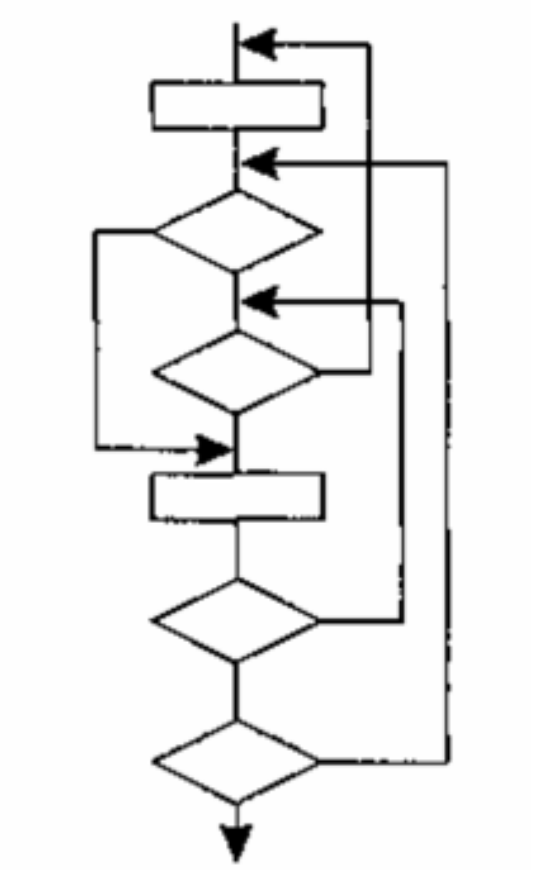
Тестирование объединенных циклов

- Если циклы независимы, то тест простого цикла,
- иначе методика вложенных.



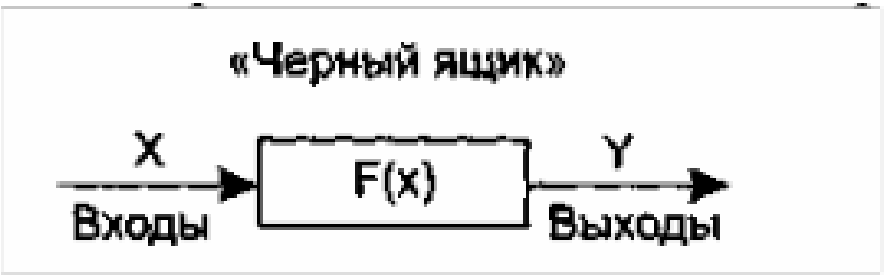
Тестирование неструктурированных циклов

- Не тестируются, должны быть переделаны.



Функциональное тестирование

- Демонстрирует:
  - Как выполняются функции ПО;
  - Как принимаются исходные данные;
  - Как вырабатываются результаты;
  - Как сохраняется целостность внешней информации.
- Рассматриваются системные характеристики ПО.
- Игнорируется внутренняя логическая структура.
- Исчерпывающее тестирование невозможно (10 входных величин, по 10 значений →1010 тестовых вариантов).
- Не реагирует на многие особенности ошибок.



Особенности тестирования «Черного ящика»

- Комбинации входных данных для полной проверки всех функциональных требований к ПО.
- Необходимо (желательно):
  - Набор входных данных => аномалии поведения ПО (IT).
  - Набор выходных данных => дефекты ПО (OT).
- Цель:
  - Определить входные данные, которые с высокой вероятностью относятся к IT.
  - Определить ожидаемые результаты, которые с высокой вероятностью относятся к OT.
- Поиск тестов:
  - Опыт инженеров тестирования;
  - Систематический подход.

Результаты тестирования «Черного ящика»

- Выявляет ошибки (не появляющиеся в «Белом ящике»):
  - Некорректных/ отсутствующих функций;
  - Ошибки интерфейса;
  - Ошибки во внешних структурах данных/ доступе к внешней БД;
  - Ошибки характеристик (емкость ОП и т.д.);
  - Ошибки инициализации и завершения.
- Плюсы:
  - уменьшение количества тестов (проверяется динамика, а не структура ПО);
  - Выявление классов, а не отдельных ошибок.

Разбиение по эквивалентности

- Входная область данных ПО делится на классы эквивалентности.
- Для каждого класса эквивалентности – 1 тестовый вариант.
- Классы эквивалентности:
  - Набор данных с общими свойствами;
  - Обрабатываются одинаково;
  - Для каждого набора из классов эквивалентности один набор операторов/связей ПО.
- Классы эквивалентности определяются по спецификации на программу.

Виды условий ввода для классов эквивалентности

- Определенное значение;
- Диапазон значений;
- Множество конкретных величин;
- Булево условие.

Правила формирования классов эквивалентности - 1

- Если условие ввода задает конкретное значение a, то определяется один допустимый и два недопустимых класса эквивалентности.
  - V_Class={ a };
  - 2_Class1={ x | для любого x: x < a};
  - 2_Class2={ y | для любого y: y < a}.
- Если условия ввода задает диапазон n…m, то определяются один допустимый и два недопустимых по условиям ввода.
  - V_Class = {n…m} – допустимый класс эквивалентности;
  - 2_Class1={x | x < n} – первый недопустимый класс;
  - 2_Class2={x |x > m} – второй недопустимый класс.

Правила формирования классов эквивалентности - 2

- Если условие ввода задает множество значений {a,b,c}, то определяется один допустимый и один недопустимый класс эквивалентности.
  - V_Class={ a,b,c };
  - 2_Class={ x | для любого x: (x<>a) & (x<>b) &(x<>c)}.
- Если условие ввода задает булево значение, например true, то определяются один допустимый и один недопустимый класс эквивалентности.
  - V_Class={true};
  - 2_Class={false}.
- Класс эквивалентности => тестовые варианты
  - Каждый тестовый вариант должен покрывать (проверять) max количество свойств класса эквивалентности.

Анализ граничных значений

- max количество ошибок на границе области ввода => тесты анализа границ.
- Дополняет разбиение по эквивалентности.

- Отличия от классов эквивалентности
  - Тесты проверяют только ребра классов эквивалентности;
  - Тесты учитывают не только условия ввода, но и область вывода.

Правила анализа граничных значений - 1

- Если условия ввода задают диапазон n…m, то тесты для
  - Значений n и m (+1,0 ; -1,0)
  - Значений чуть < n и чуть > m (+1,001; -1,001)
- Если условия ввода задают дискретное множество, то тесты для
  - Min и max из значений (1; 255)
  - Чуть < min и чуть > max (0; 256)
- 1) и 2) применяют к области вывода
  - (вывод min таблицы, вывод max таблицы)

Правила анализа граничных значений - 2

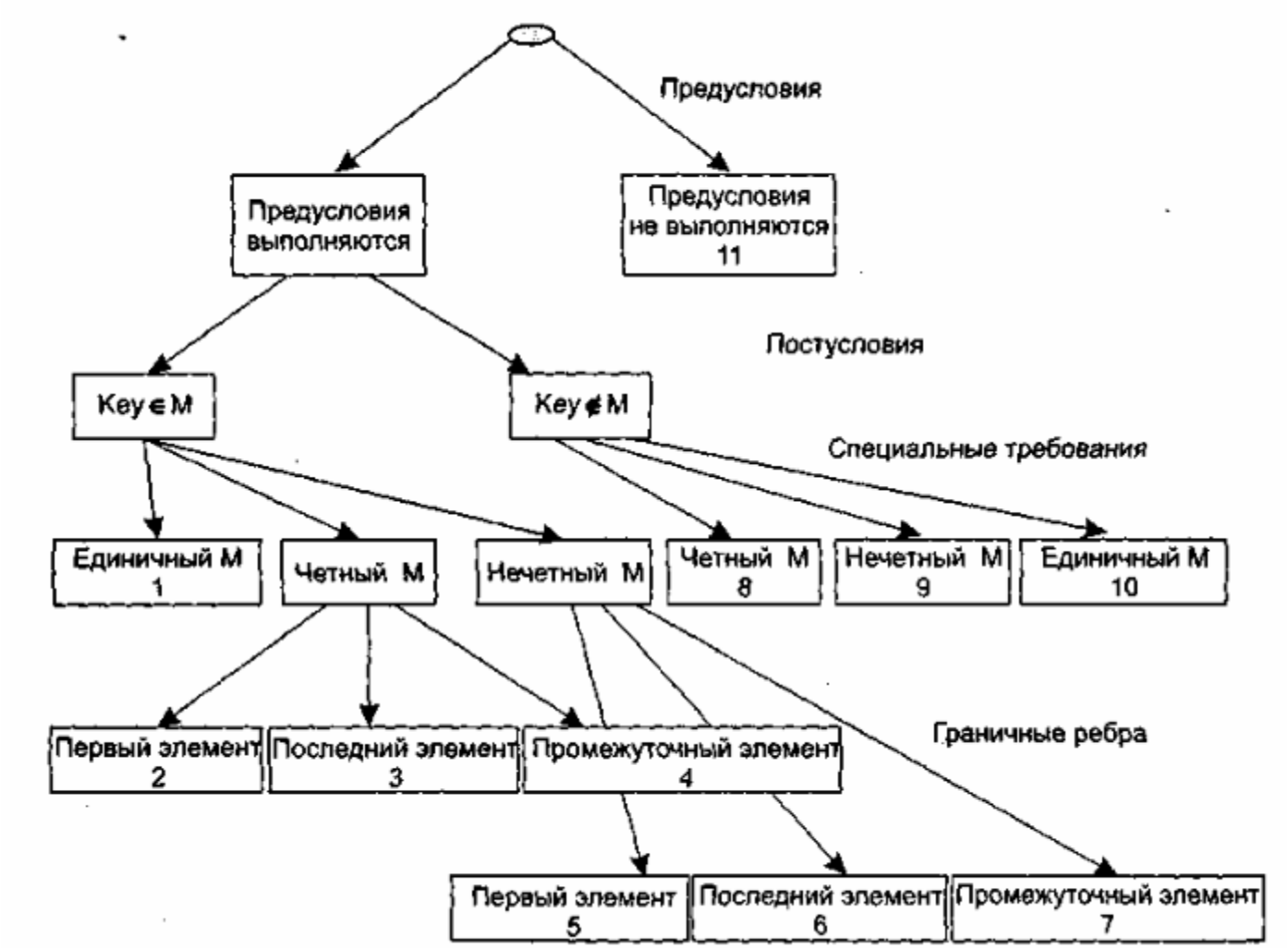
- Если внутренние структуры ПО имеют границы, то тесты
  - для проверки структур на границах.

- Если входные/выходные данные являются упорядоченным множеством, то тесты
  - обработки первого и последнего элементов множеств.

Пример анализа граничных значений

- Поиск в упорядоченном массиве по ключу, возвращает индекс элемента I.
- Предусловие:
  - Массив должен быть упорядочен.
  - Массив >= 1 элемента.
  - Нижние границы <= верхней границы массивы.
- Постусловия:
  - Найден => res = true, I = № элемента
  - Нет => res = false, I = null
- Разбиение на КЭ:
  - 1й уровень – выполнимость предусловий,
  - 2й уровень – выполнимость постусловий,
  - 3й уровень – специальные требования (чет/нечет/1 элемент),
  - 4й уровень – анализ ребер (1-й эл. массива/ последний/ средний элемент).

Пример анализа граничных значений - дерево тестов



Пример тестовых вариантов

- Каждый лист задает отдельный тестовый вариант:
- ТВ1:
  - ИД: M = (15), K = 15
  - ОР: R = True, I = 1.
- ТВ2:
  - ИД: M = (15,20,21,40), K = 15
  - ОР: R = True, I = 1.
- ТВ9:
  - ИД: M = (15,20,21), K = 100
  - ОР: R = False, I = null.
- ТВ11:
  - ИД: M = (1,3,2,5,10,7), K = 1
  - ОР: сообщение об ошибке.