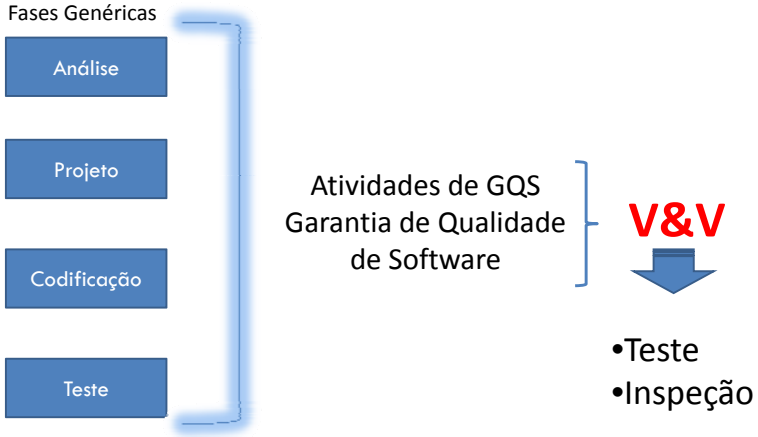
TESTE DE SOFTWARE

Profa. Sandra C. Pinto Ferraz Fabbri

Sumário

- Entendendo o que é Teste
  - Teste & Ciclo de Vida de Desenvolvimento
  - O que é V&V?
  - Você sabe responder?
- O Plano de Teste
- Critérios & Técnicas de Teste
- Teste Funcional
  - Particionamento de Equivalência
  - Análise do Valor Limite

Teste & Ciclo de Vida de Desenvolvimento



Teste & Ciclo de Vida de Desenvolvimento

- Atividades de Garantia de Qualidade de Software
  - É um conjunto de atividades técnicas aplicadas durante todo o processo de desenvolvimento.
  - O objetivo é garantir que tanto o processo de desenvolvimento quanto o produto de software atinjam níveis de qualidade especificados.

V&V – Verificação e Validação

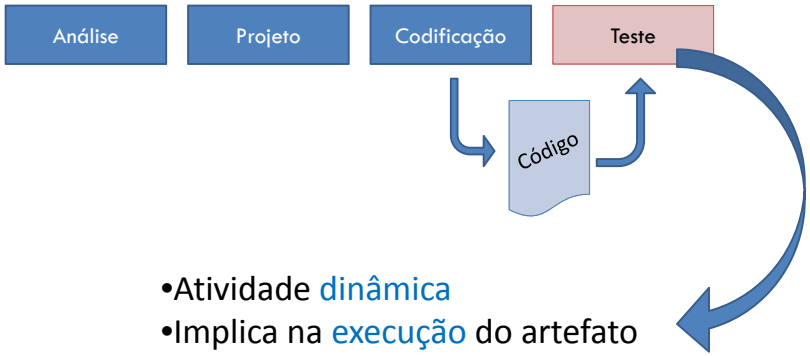
O que é V&V ?

- Verificação – Estou construindo o produto corretamente?
- Validação – Estou construindo o produto correto?

TESTE – você sabe responder ?

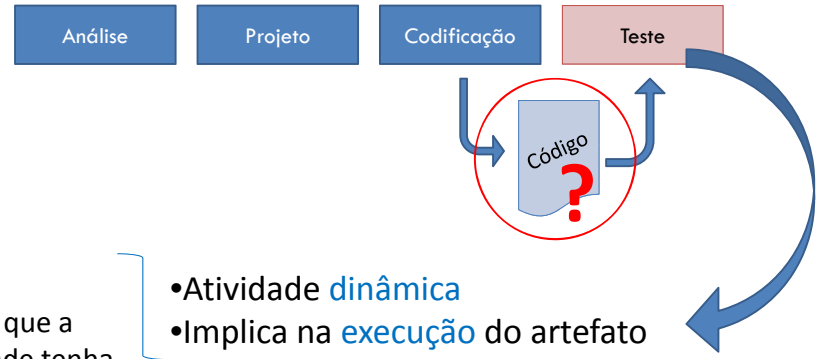
- Quando o teste ocorre?
- Qual o artefato necessário para que o teste possa existir?
- Qual o objetivo da atividade de teste?
- Depois de realizado o teste, é possível afirmar que o programa está correto?
- Quando é possível afirmar que o programa está correto?
- O que compõe a atividade de teste?
- Quando parar a atividade de teste?

Quando o teste ocorre?



- Atividade dinâmica
- Implica na execução do artefato que está sendo testado
- Avalia o comportamento

Qual o artefato necessário para que o teste possa ocorrer?



Não...
Desde que a atividade tenha essas características...

- Atividade dinâmica
- Implica na execução do artefato que está sendo testado
- Avalia o comportamento

Qual o objetivo da atividade de teste?

É uma atividade para mostrar que o programa está correto?

( ) SIM ( ) NÃO

Qual o objetivo da atividade de teste?

É uma atividade para mostrar que o programa está correto?

( ) SIM (**X**) NÃO

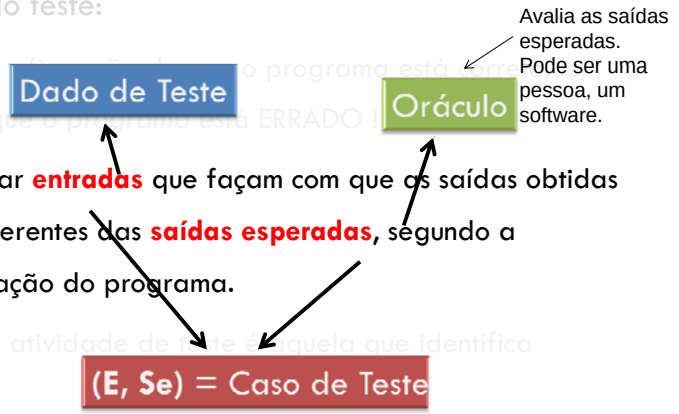
Mas então, qual é o objetivo da atividade de teste?

Qual o objetivo da atividade de teste?

- Objetivo do teste:
  - Refutar a afirmação de que o programa está correto, ie, mostrar que o programa está ERRADO !
  - Determinar entradas que façam com que as saídas obtidas sejam diferentes das saídas esperadas, segundo a especificação do programa.
  - Uma boa atividade de teste é aquela que identifica defeitos.

Qual o objetivo da atividade de teste?

- Objetivo do teste:
  - Refutar a afirmação de que o programa está correto, ie, mostrar que o programa está ERRADO !
  - Determinar **entradas** que façam com que as saídas obtidas sejam diferentes das **saídas esperadas**, segundo a especificação do programa.
  - Uma boa atividade de teste é aquela que identifica defeitos.



Depois de realizado o teste é possível afirmar que o programa está correto?

13

( ) SIM ( ) NÃO

Depois de realizado o teste é possível afirmar que o programa está correto?

14

( ) SIM (**X**) NÃO

- É possível aumentar a confiança de que isso seja verdade!
- ....a não ser que se aplique o TESTE EXAUSTIVO

Quando é possível afirmar que o programa está correto?

15

Quando se aplica

TESTE EXAUSTIVO = testar com TODAS as possíveis entradas

Ex: Suponha um programa que receba 2 números inteiros entre 0 e 10 e calcule a média aritmética

Se for aplicado o teste exaustivo, quantas são as entradas?

121

Quando é possível afirmar que o programa está correto?

16

TESTE EXAUSTIVO = testar com TODAS as possíveis entradas

Ex: Suponha um programa que receba 2 números reais, entre 0.0 e 10.0, com uma casa decimal e calcule a média aritmética

Se for aplicado o teste exaustivo, quantas são as entradas?

... e se forem duas casas decimais???

Quando é possível afirmar que o programa está correto?

Resumindo ……

Em geral, NÃO é possível aplicar o TESTE EXAUSTIVO

NÃO é possível afirmar que o programa está correto.

O que compõe a atividade de teste?

- Planejamento da atividade
- Determinação dos casos de teste
- Execução dos casos de teste
- Avaliação dos resultados

Plano de Teste

Quando parar a atividade de teste?

Terminou o tempo

Terminou o recurso financeiro

O **plano de teste** foi cumprido

O Plano de Teste

- Deve conter todas as informações sobre a atividade de teste:
  - Planejamento
    - Equipe de teste
    - Software de suporte
    - Cronograma das atividades
  - Determinação dos casos de teste
  - Execução dos casos de teste
  - Avaliação dos resultados
  - ……

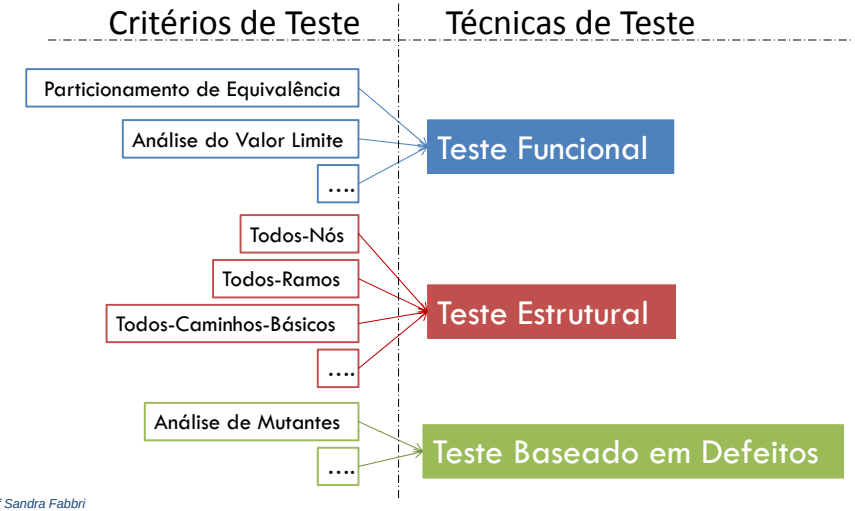
Técnicas de Teste
➢Critérios de Teste

Critérios de Teste – o que são e porque são precisos?

Critérios de Teste – o que são e porque são precisos?

- Maneira sistemática e planejada de elaborar os casos de teste (c.t.)
- Podem ser usados de duas formas:
  - Para seleção dos casos de teste: quando os c.t. são criados para satisfazer os requisitos do critério de teste
  - Para adequação dos casos de teste: quando os c.t. são criados, por ex, aleatoriamente e então verifica-se se eles atendem os requisitos do critério de teste.

Critérios de Teste & Técnicas de Teste



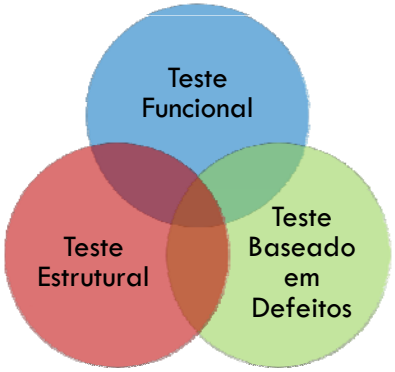
Técnicas de Teste

Teste Funcional
(Teste Caixa Preta)
Os requisitos de teste são extraídos da especificação do programa

Teste Estrutural
(Teste Caixa Branca)
Os requisitos de teste são extraídos da estrutura do código do programa

Teste Baseado em Defeitos
Os requisitos de teste são extraídos dos defeitos típicos e comuns cometidos durante a programação

Técnicas de Teste



São complementares pois identificam tipos diferentes de defeitos

Defeito, Erro e Falha

- Defeito ➡ Erro ➡ Falha
  - Defeito: deficiência algorítmica que, se ativada, pode levar a uma falha
  - Erro: estado de execução inconsistente
  - Falha: evento observável que mostra que o programa violou suas especificações

Teste Funcional

- Os requisitos de teste são extraídos da especificação do software.
- Aborda o software de um ponto de vista macroscópico. Por isso é também chamado **Teste Caixa Preta**.
- Problema:
  - Dificuldade em quantificar a atividade de teste - não se pode garantir que partes essenciais ou críticas do software foram executadas
- Critérios:
  - Particionamento de Equivalência
  - Análise do Valor Limite
  - Grafo Causa-Efeito

Teste Funcional – Particionamento de Equivalência

- Divide o domínio de entrada em *classes* ou *partições de equivalência* que, de acordo com a especificação do programa, são tratadas da mesma maneira.
- Observar também a saída do programa e verificar se, com base na saída, é possível estabelecer classes no domínio de entrada que ajudem a avaliar se a saída está sendo produzida corretamente.

Teste Funcional – Particionamento de Equivalência

- As classes, que podem ser *válidas* ou *inválidas*
- Diretrizes para definição das classes:
  - se a condição de entrada especifica um intervalo, são definidas uma classe válida e duas inválidas
  - se a condição de entrada exige um valor específico, são definidas uma classe válida e duas inválidas
  - se a condição de entrada especifica um membro de um conjunto, são definidas uma classe válida e uma inválida
  - se a condição de entrada for booleana, são definidas uma classe válida e uma inválida

Exemplo

Especificação:

O programa solicita do usuário um inteiro positivo no intervalo entre 1 e 20. Em seguida, solicita uma cadeia de caracteres desse comprimento.
Após isso, o programa solicita um caracter e retorna a posição na cadeia em que o caracter é encontrado pela primeira vez, ou uma mensagem indicando que o caracter não está presente na cadeia.
O usuário tem a opção de procurar por vários caracteres sequencialmente.

Aplicação – Particionamento de Equivalência

- Analisando a especificação do ponto de vista das entradas:
  - um inteiro positivo (entre 1 e 20 ⇒ 3 partições)
  - uma cadeia de caracteres
  - um caracter a ser procurado
  - a opção por procurar por mais caracteres (duas partições: uma para "y" e outra para "n")
- O domínio de saída consiste de duas respostas, que levam a outra divisão do domínio de entrada:
  - a posição na qual o caracter foi encontrado na string (caracter de entrada pertencente à string ⇒ 2 partições)
  - uma mensagem declarando que ele não foi encontrado (caracter de entrada não pertencente à string ⇒ 2 partições)

Aplicação – Particionamento de Equivalência

Casos de Teste criados para satisfazer o critério Particionamento de Equivalência:

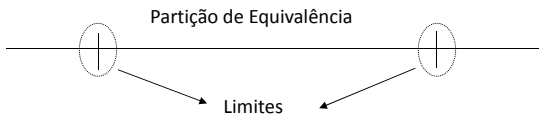
Entrada				Saída Esperada
Tamanho da cadeia	Cadeia	Caracter	Resposta do usuário	
34				Digite um inteiro entre 1 e 20
0				Digite um inteiro entre 1 e 20
3	abc	b		O caracter *b* aparece na posição 2
			y	
		g		O caracter *g* não aparece na cadeia
			n	

Aplicação – Particionamento de Equivalência

- Observação:
  - reduz o tamanho do domínio de entrada
  - concentra-se em criar casos de teste baseados unicamente na especificação
  - é especialmente adequado para aplicações em que as variáveis de entrada podem ser facilmente identificadas e podem ter valores distintos
  - problemas:
    - embora a especificação possa sugerir que um grupo de dados seja processado de forma idêntica, isso pode não ocorrer
    - a técnica não fornece um guia para a determinação dos dados de teste

Teste Funcional – Análise do Valor Limite

- Complementa o Particionamento de Equivalência
- Coloca sua atenção em uma fonte propícia a defeitos – os limites de uma classe ou partição de equivalência



Aplicação – Análise do Valor Limite

- Os valores limites das classes é que devem ser selecionados: 0, 1, 20 e 21
- Os caracteres a serem encontrados devem estar na 1a. e na última posição.

Aplicação – Análise do Valor Limite

Casos de Teste criados para satisfazer o critério Análise do Valor Limite:

	Entrada				Saída Esperada
	Tamanho da cadeia	Cadeia	Caracter	Resposta do usuário	
1a. execução	21				Digite um inteiro entre 1 e 20
	0				Digite um inteiro entre 1 e 20
	1	a	a		O caracter a aparece na posição 1
				y	
			g		O caracter g não aparece na cadeia
				n	
2a. execução	20	abcdefghijklmnopqrst	a		O caracter a aparece na posição 1
				y	
			t		O caracter a aparece na posição 20
				n	

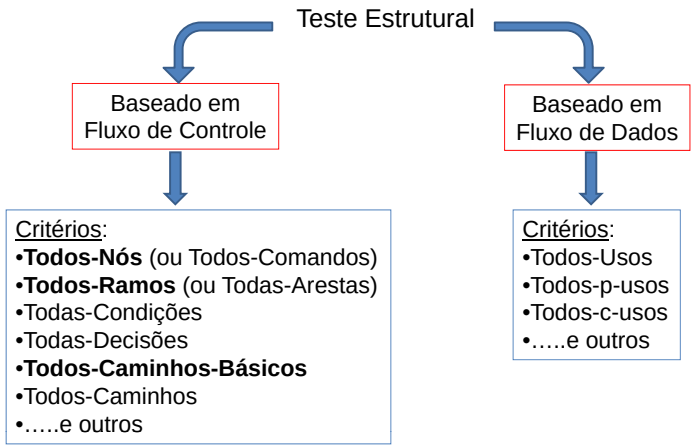
Sumário

- Teste Estrutural
  - Todos-Nós (ou Todos-Comandos)
  - Todos-Ramos (ou Todas-Arestas)
  - Todos-Caminhos-Básicos
- Teste Baseado em Defeitos
  - Análise de Mutantes

Teste Estrutural

- Os requisitos de teste são extraídos de uma implementação em particular.
- Teste dos detalhes procedimentais. Por isso é também chamado **Teste Caixa Branca.**
- A maioria dos critérios dessa técnica utiliza uma representação de programa conhecida como grafo de programa ou grafo de fluxo de controle.
- Permite uma avaliação de *cobertura* do código.
- Problema: caminhos não executáveis.

Teste Estrutural



Teste Estrutural

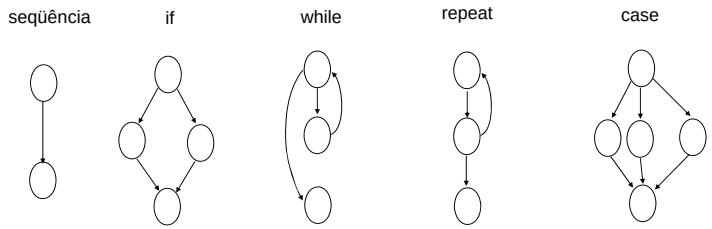
Grafo de Fluxo de Controle ou Grafo de Programa

- consiste de *nós* conectados por *arcos* com setas que mostram sua direção
- os *nós* representam blocos de comandos
  - bloco de comando: é um conjunto de comandos tal que se o primeiro comando for executado, então todos os comandos subseqüentes também o serão
- os *arcos* indicam precedência, ou transferência de controle
- essa representação permite que o programa seja examinado independentemente de sua função

Teste Estrutural

41

Construções Básicas do Grafo de Fluxo

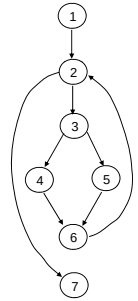


Teste Estrutural

42

Exemplo de Construção do Grafo de Fluxo de Controle

```
início                                          } 1
        leia nro
        enquanto nro ≠ 0                        } 2
            se nro > 0                          } 3
                    raiz = raiz-quadrada(nro)   } 4
                    escreva raiz
            senão
                    escreva mensagem de erro    } 5
            fim-se
            leia nro                            } 6
        fim-enqto
fim                                             } 7
```



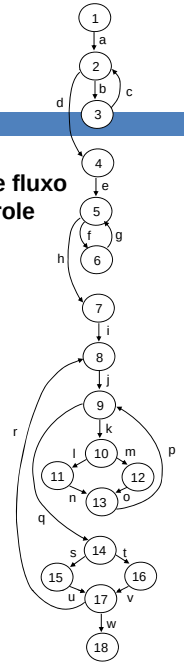
Teste Estrutural

43

Programa com blocos marcados

```
1  program exemplo(input, output);
2  var  Cadeia: array[1..20] of char;
3    Tam, i: integer;
4    Carac, resposta: char;
5    achou: boolean;
6  begin
7    writeln ('entre com um inteiro entre 1 e 20');                    1
8    readln (Tam);
9    while (Tam < 1) or (Tam > 20) do                                  2
10       begin
11         writeln ('digite um inteiro entre 1 e 20');                 3
12         readln (Tam)
13       end;
14   writeln ('digite', Tam, 'caracteres');                            4
15   for i := 1 to Tam do                                              5
16             read (Cadeia[ i ]);                                     6
17   readln;                                                           7
18   repeat
19         writeln ('digite o caracter a ser pesquisado: ');
20           readln (Carac);                                           8
21           achou := FALSE;
22           i := 1;
23         while (not(achou)) and (i <= Tam) do                        9
24           begin
25             If Cadeia[ i ] = Carac                                  10
26             then   achou := TRUE                                    11
27             else
28                    i := i + 1                                       12
29           end;                                                      13
30           if achou                                                  14
31           then  writeln ('o caracter ', Carac, ' aparece na posicao', i)   15
32           else
33               writeln ('o caracter ', Carac, ' não aparece na cadeia');    16
34           writeln;
35           writeln ('deseja fazer a procura para outro caracter? [s/n]');
36           readln (Resposta);                                        17
37   until (Resposta = 'n') or (Resposta = 'N');
38 end.                                                              18
```

Grafo de fluxo de controle



Teste Estrutural – Todos-Nós (ou Todos-Comandos)

44

- Estabelece como requisito de teste que sejam executados todos os comandos do programa ao menos uma vez

Aplicação – Todos-Nós

45

- é necessário concentrar-se nos comandos que são controlados por condições
- fornecer um valor para x que esteja fora do intervalo para forçar a execução dos comandos no loop (linhas 11-12)
- quando x está dentro do intervalo, ele será no mínimo 1 e o comando dentro do for (linha 16) será executado
- o comando *if* (linha 25) precisa ser executado, com as alternativas *then* e *else* (linhas 26 e 28) ⇒ procurar por um caracter que esteja na cadeia e que force a procura dentro dela
- a entrada no *while* (linha 23) é garantida
- o comando *if* (linha 30) precisa ser executado, com as alternativas *then* e *else* (linhas 31 e 33) ⇒ um caracter que pertença e outro que não pertença à cadeia (combinando com o *if* anterior ⇒ uma cadeia de caracteres *a*, de um caracter e valores para a variável *Carac* que ocorra e que não ocorra em *Cadeia*)
- por fim, é preciso terminar o *loop* para executar o comando *end*

Aplicação – Todos-Nós

46

Casos de Teste criados para satisfazer o critério Todos-Nós:

Entrada				Saída Esperada
Tamanho da cadeia	Cadeia	Caracter	Resposta do usuário	
25				Digite um inteiro entre 1 e 20
1	a	a		O caracter a aparece na posição 1
			y	
		g		O caracter g não aparece na cadeia
			n	

Obs: é o nível mínimo de cobertura esperado no teste estrutural.

Teste Estrutural – Todos-Ramos (ou Todas-Arestas)

47

- Estabelece como requisito de teste que sejam executadas todas as saídas *verdadeiro* e *falso* de todas as decisões.

Aplicação – Todos-Ramos

48

- observando o grafo, é necessário gerar dados de teste que causem as duas saídas *verdadeiro* e *falso* que ocorrem nos nós 2, 5, 9, 10, 14 e 17
- para o nó 2, um valor da variável Tam menor que 1 ou maior que 20 causa a saída pelo ramo verdadeiro e um valor de Tam dentro desse intervalo causa a saída pelo ramo falso
- o nó 5, comando *for* (linha 15), terá as saídas verdadeiro e falso, desde que o valor de Tam seja ao menos 1 (e terá que ser para chegar nesse ponto)
- para o nó 9, loop *while* (linha 23), a saída verdadeiro (arco k) é garantida devido aos comandos das linhas 21 e 22; a saída falso (arco q) é garantida ou quando o caracter que está sendo procurado é encontrado ou quando o final da cadeia é encontrado
- o nó 10, *if* (linha 25), requer uma comparação que encontre o caracter que está sendo procurado (arcos l, n) e uma outra que cause o incremento de i (arcos m, o)
- o nó 14 (linha 30) precisa de um caracter que seja encontrado e um outro que não seja encontrado
- o nó 17 requer pelo menos mais uma iteração do loop *repeat* (arco r) antes que o final do programa seja encontrado (arco w)

Aplicação – Todos-Ramos

Casos de Teste criados para satisfazer o critério Todos-Ramos:

Entrada				Saída Esperada
Tamanho da cadeia	Cadeia	Caracter	Resposta do usuário	
25				Digite um inteiro entre 1 e 20
1	a	a		O caracter a aparece na posição 1
			y	
		g		O caracter g não aparece na cadeia
			n	

Obs:
•o conjunto de dados de testes é o mesmo do Teste de Comandos, mas neste, o caracter *g* poderia ser fornecido através de outra execução do programa.
•já no Teste de Ramos, a execução do loop *repeat* é obrigatória .

Teste Estrutural – Todos-Caminhos-Básicos

- Esse critério determina um conjunto básico de caminhos linearmente independentes, de modo que executando-os garante-se a execução de todos os ramos ao menos uma vez.
- O número de caminhos é determinado pela fórmula da Complexidade Ciclomática de Mc'Cabe

  $V(G) = a - n + 2$ ou
  $V(G) = P + 1$ ou
  $V(G) = nº\text{ de regiões}$

  sendo:
  - G: um grafo direcionado
  - a: arestas (ramos)
  - n: nós
  - P: no. de nós predicativos
- Um caminho linearmente independente é aquele que contém ao menos um novo nó

Aplicação – Todos-Caminhos-Básicos

- Cálculo de V(G):
  - V(G) = 23 - 18 + 2 = 7
  - V(G) = 6 +1 = 7
  - V(G) = no. de regiões = 7
- A partir desse número deve-se identificar 7 caminhos linearmente independentes do grafo:
  1. 1-2-4-5-7-8-9-14-15-17-18
  2. 1-2-4-5-7-8-9-14-16-17-18
  3. 2-3-2
  4. 5-6-5
  5. 9-10-11-13-9
  6. 9-10-12-13-9
  7. 8-9-14-15-17-8

Aplicação – Todos-Caminhos-Básicos

- Esse conjunto é conhecido como conjunto base a partir do qual qualquer outro caminho pode ser construído
- Por exemplo, o caminho:

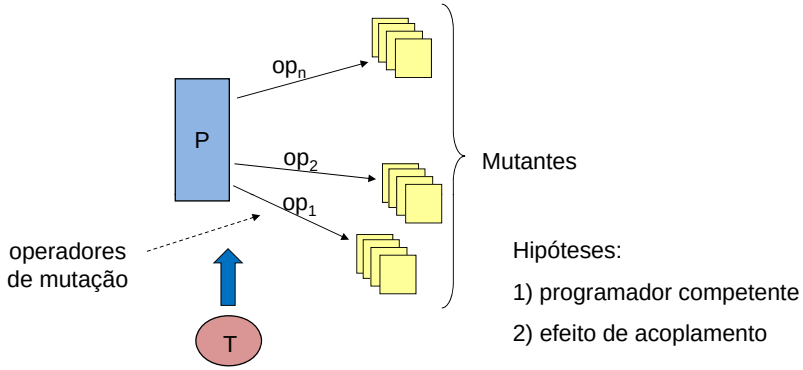
  1-2-3-2-4-5-6-5-6-5-7-8-9-10-11-13-9-14-15-17-18

  é uma combinação dos caminhos 1, 3, 4 (2 vezes) e 5
- Neste caso, o conjunto de casos de teste é o mesmo do Todos-Ramos

Teste Baseado em Defeitos

- Os requisitos de teste são estabelecidos com base nos defeitos típicos e comuns cometidos durante o desenvolvimento do software.

Teste Baseado em Defeitos – Análise de Mutantes



$$\text{escore de mutação} = \frac{\#\text{ mutantes mortos}}{\#\text{ mutantes gerados não equivalentes}}$$

Teste Baseado em Defeitos – Análise de Mutantes

- Hipótese do programador competente
  - *Programadores experientes escrevem programas corretos ou muito próximos do correto.*
- Efeito do acoplamento
  - *Casos de teste capazes de revelar erros simples são tão sensíveis que, implicitamente, também são capazes de revelar erros mais complexos.*

Teste Baseado em Defeitos – Análise de Mutantes

- Os operadores de mutação determinam o tipo de alteração sintática que deve ser feita para a criação dos mutantes
  - Introduzir pequenas alterações semânticas através de pequenas alterações sintáticas que representam defeitos típicos
- Operadores dependem da linguagem alvo
  - FORTRAN (22 operadores) C (71 operadores)

Teste Baseado em Defeitos – Análise de Mutantes

- Exemplos de operadores de mutação
  - Retira um comando de cada vez do programa
  - Troca operador relacional por operador relacional
  - Troca o comando while por do-while
  - Interrompe a execução do laço após duas execuções
  - Troca constante por constante
  - Requer valor negativo, positivo e zero para cada referência escalar

Teste Baseado em Defeitos – Análise de Mutantes

- Dados **P** e **T**
- Passos para a aplicação da Análise de Mutantes
  - **P** é executado com os casos de teste de **T**
  - Mutantes são gerados
  - Mutantes são executados com os casos de teste de **T**
  - Mutantes são analisados

Aplicação – Análise de Mutantes

- Considere o trecho do programa e o conjunto de teste obtido para o teste Todos-Ramos

```
:
21    achou := FALSE;
22    i := 1;
23    while (not(achou)) and (i <= Tam) do
24    begin
25       if  Cadeia[ i ] = Carac then
26          achou := TRUE
27       else
28          i := i + 1
29    end;
:
```

Entrada				Saída Esperada
Tamanho da cadeia	Cadeia	Caracter	Resposta do usuário	
25				Digite um inteiro entre 1 e 20
1	a	a		O caracter a aparece na posição 1
			y	
		g		O caracter g não aparece na cadeia
			n	

Aplicação – Análise de Mutantes

- Mutante 1: alteração na linha 21
  - de: achou := FALSE
  - para: achou := TRUE
    - considere agora que o mutante seja re-executado com os casos de teste obtidos para o critério Todos-Ramos
    - a saída gerada seria: "o caracter a aparece na posicao 1" em vez de: "o caracter a não aparece na cadeia"
    - o mutante seria morto por esse conjunto de casos de teste

Teste Baseado em Defeitos – Análise de Mutantes

- Observação:
  - Teste de Mutação consegue mostrar a ausência de defeitos particulares, pois ao matar os mutantes está sendo mostrado que o programa original não possui aquele defeito.
  - Esse critério força o testador a analisar cuidadosamente o programa, uma vez que ele precisa criar casos de teste que exponham os defeitos introduzidos.
  - desvantagem:
    - ele é computacionalmente caro devido ao grande número de mutantes gerados e o tempo e recurso usados para compilar e executar todos eles

Exercícios para fazer em sala

1. O programa deve ler 3 variáveis inteiras, *a*, *b* e c e escrevê-las nessa sequência. Após isso, os valores devem ser ordenados crescentemente, de forma que em *a* fique o menor valor e em c fique o maior valor. As variáveis devem ser escritas novamente nessa mesma sequência (*a*, *b*, c).

2. O programa deve ler um valor máximo (inteiro) e deve escrever a sequência de Fibonacci de forma a não ultrapassar esse valor, ou então escrever uma mensagem de erro caso a sequência não possa ser gerada de acordo com esse valor máximo. A sequência de Fibonacci é: 1 1 2 3 5 8 13 …..

Exercícios para fazer em sala


```
package exerciciosteste;

import java.util.Scanner;

/**
 * Exercicios
1- Ordenacao crescente de 3 valores inteiros
2- Geracao da sequencia de Fibonacci de acordo com um valor maximo
*/

public class Main {

    public static void main(String[] args) {
        crescente();
        fibonacci();
    }
```


Exercícios para fazer em sala


```
public static void crescente() {

1       int a, b, c, aux;
2
3       Scanner leitor = new Scanner(System.in);
4       a = leitor.nextInt();
5       b = leitor.nextInt();
6       c = leitor.nextInt();
7       System.out.println("Valores na ordem lida: " + a + " " + b + " " + c);
8       if(a > b) {
9           aux = a;
10          a = b;
11          b = aux;
12      }
13      if(a > c) {
14          aux = a;
15          a = c;
16          c = aux;
17      }
18      if(b > c) {
19          aux = b;
20          b = c;
21          c = aux;
22      }
23      System.out.println("Valores ordenados: " + a + " " + b + " " + c);
24  }
```

Exercícios para fazer em sala

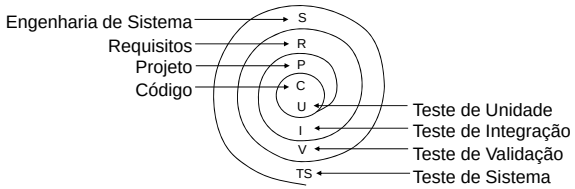

```
public static void fibonacci() {
1    int t1, t2, tnovo, max;
2    Scanner leitor = new Scanner(System.in);
3    max= leitor.nextInt();
4    if(max >= 1) {
5       System.out.println("A sequência de Fibonacci até o valor" + max + " é ");
6       t1 = 1;
7       t2 = 1;
8       System.out.println(t1 + " " + t2 + " ");
9       tnovo = t1 + t2;
10      while(tnovo <= max) {
11         System.out.println(tnovo + " ");
12         t1 = t2;
13         t2 = tnovo;
14         tnovo = t1 + t2;
15      }
16   }
17   else {
18      System.out.println("Não existe sequência de Fibonacci para o valor : " + max);
19   }
20   }// fim do metodo
```


Estratégias de Teste

- Aspectos genéricos das Estratégias de Teste
  - a atividade de teste inicia-se no nível de módulos e caminha na direção da integração de todo o sistema
  - diferentes técnicas de teste são apropriadas para diferentes situações
  - a atividade de teste, em geral, é realizada pela equipe de desenvolvimento e, no caso de grandes projetos, por um grupo de teste independente
  - as atividades de teste e depuração são atividades diferentes, mas a depuração é necessária em qualquer estratégia de teste

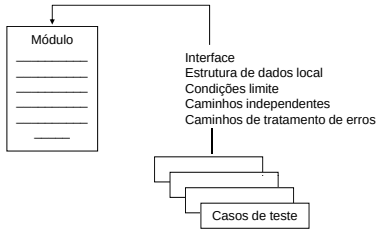
Estratégias de Teste

Relação entre o processo de desenvolvimento e uma estratégia de teste



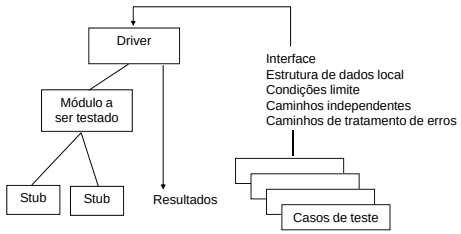
Teste de Unidade

- concentra-se no módulo
- utiliza a técnica de teste estrutural
- pode ser realizado em paralelo para vários módulos
- aspectos considerados:



Teste de Unidade

- Geralmente, um programa não é um módulo único, mas formado de diversos módulos que, para efeito do teste de unidade devem ser testados separadamente



driver : é um "programa principal" que aceita dados de casos de teste, passa esses dados para o módulo a ser testado e imprime os dados relevantes que ele recebe do módulo

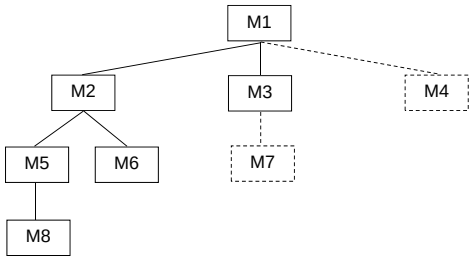
stub : são módulos que servem para substituir outros módulos que estejam subordinados, isto é, que são chamados pelo módulo testado; ele usa a interface do módulo subordinado, faz o mínimo de manipulação de dados, imprime uma verificação da entrada e retorna

Teste de Integração

- constrói-se, de uma forma sistemática, a estrutura do programa realizando, ao mesmo tempo, testes para detectar erros de interface
- embora os módulos, depois do teste de unidade, funcionem corretamente de forma isolada, o teste de integração é necessário pois quando colocados juntos, várias situações inesperadas podem acontecer
- integração dos módulos é feita através de uma *abordagem incremental*
  - *integração top-down*
  - *integração bottom-up*

Teste de Integração

- Integração Top-down
  - a integração dos módulos é feita de cima para baixo
  - pode ser realizada de duas maneiras:
    - por *profundidade* (M2, M5 e M8 ou M2, M5, M6 e M8)
    - por *largura* (M2, M3 e M4)



Teste de Integração

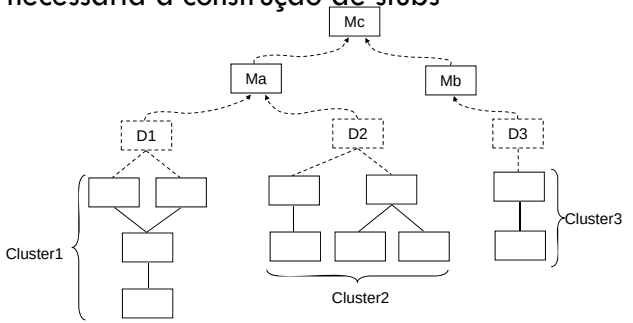
- Integração Top-down
  - o processo de integração é feito através de cinco passos:
    - 1. o módulo de controle principal é usado como um *driver* e substitui-se por stubs todos os módulos reais diretamente subordinados ao módulo principal;
    - 2. dependendo da abordagem de integração a ser utilizada – por profundidade ou largura – os stubs são substituídos pelos módulos reais, um de cada vez;
    - 3. são realizados testes para cada módulo que seja integrado;
    - 4. quando um teste é concluído, outro stub é substituído pelo módulo real;
    - 5. teste de regressão, isto é, repetição de todos ou alguns dos testes já realizados pode ser aplicado novamente para garantir que novos erros não tenham sido introduzidos.

Teste de Integração

- Integração Top-down
  - por profundidade: permite que uma função específica do módulo principal possa ser testada por completo
  - nem sempre a construção de um stub é uma tarefa fácil, pois se a função do módulo real que ele representa for complexa, o stub tem que tratar os aspectos principais desse módulo para que o teste seja significativo

Teste de Integração

- Integração Bottom-up
  - a integração dos módulos é feita de baixo para cima
  - quando os módulos de níveis superiores vão ser testados, os módulos subordinados já estão prontos e portanto, não se torna necessária a construção de stubs



Teste de Integração

- Integração Bottom-up
  - o processo de integração é feito através de quatro passos:
    - 1. os módulos de nível mais baixo são combinados em *clusters* que executam funções específicas do módulo principal;
    - 2. para cada cluster é elaborado um driver que coordena a entrada e saída dos casos de teste;
    - 3. o cluster é testado;
    - 4. os drivers são substituídos pelos clusters que passam a interagir com os módulos acima, na estrutura do programa.

Teste de Integração

- a escolha por uma dessas duas abordagens de integração depende do tipo de software e, às vezes, do cronograma do projeto
- em geral, uma integração combinada - *sanduíche* - é mais aconselhável:
  - módulos superiores → abordagem top-down
  - módulos mais inferiores → abordagem bottom-up
- top-down
  - vantagem: testar logo no início as funções principais do software
  - desvantagem: os stubs e a dificuldade de teste quando eles são usados
- bottom-up
  - vantagem: não se precisa de stubs
  - desvantagem:o módulo principal não existe enquanto todos módulos não estiverem testados

Teste de Validação

- o software está montado como um pacote e a validação do mesmo é realizada através de uma série de testes caixa preta
- finalidade:
  - demonstrar a conformidade aos requisitos funcionais e de desempenho
  - verificar se a documentação está correta
- duas possibilidades:
  - aceito
  - não está totalmente de acordo com os requisitos: negociar com o usuário

Teste de Validação

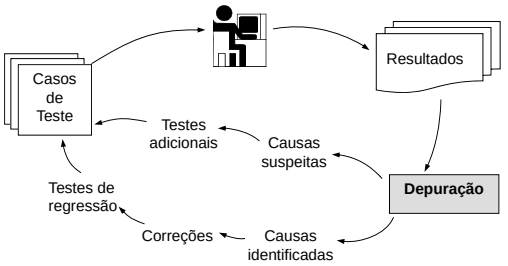
- engloba o *Teste de Aceitação*: realizado pelo próprio usuário
- no caso de software desenvolvido para vários usuários:
  - *teste alfa*: realizado pelo usuário no ambiente do desenvolvedor
  - *teste beta*: realizado pelo usuário em seu próprio ambiente

Teste de Sistema

- considera o software dentro do seu ambiente mais amplo (todos os aspectos de interação com ele, como outro hardware, software, pessoas, etc.)
- corresponde a uma série de testes que tem por objetivo verificar se todos os elementos do sistema foram integrados adequadamente e realizam corretamente suas funções
  - *teste de segurança*: tem por objetivo verificar se todos os mecanismos de proteção protegem realmente o software de acessos indevidos.
  - *teste de estresse*: tem por objetivo confrontar os programas com situações anormais de freqüência, volume ou recursos em quantidade.
  - *teste de desempenho*: tem por objetivo testar o tempo de resposta do sistema e é aplicado, geralmente, para sistemas de tempo real

Depuração

Processo de Depuração



Depuração

- abordagens para se conduzir a depuração:
  - *força bruta*: quando o erro não é analisado para se descobrir a causa, tentando que o próprio computador a descubra, inserindo, por exemplo, vários comandos de escrita no programa.
  - *backtracking*: inicia-se no local em que o sintoma foi detectado e rastreia-se para trás, manualmente, até que o local da causa seja encontrado.
  - *eliminação da causa*: supõe-se uma causa e elaboram-se casos de teste para comprovar ou refutar essa hipótese

Pontos-Chaves

- o objetivo do teste é encontrar erros e se eles não forem detectados, o teste não pode afirmar sua ausência
- testar tudo é impossível
- as técnicas de teste são complementares, isto é, devem aplicadas conjuntamente
- a execução do teste é criativa e difícil, pois para testar com eficiência é preciso conhecer o software a fundo

FIM