

Engenharia de Software

Testes de Unidade



Marcello Thiry
marcello.thiry@gmail.com

LQPS
<http://www.univali.br/lqps>

Teste de unidade

- ☐ Unidade é a menor parte de um sistema que pode ser testada:
 - ☐ Programa individual, procedimento, função
 - ☐ Classe, objeto, método
 - ☐ Componente (COTS – *commercial-off-the-shelf*)
- ☐ **Objetivos:**
 - ☐ Indicar se as unidades de software funcionam de acordo com a especificação (requisitos, design)
 - ☐ Testar uma unidade de código de forma isolada do resto do sistema de software (que talvez ainda não esteja pronto)
- ☐ Usualmente, é realizado pelo desenvolvedor durante a implementação da unidade

Teste de unidade

- ☐ As unidades são testadas individualmente e de modo independente de outras unidades
- ☐ Parte do processo de implementação, onde a unidade gerada deve estar em conformidade com sua especificação
- ☐ **Exemplos:**
 - ☐ O método retorna um valor correto quando é executado com uma entrada válida?
 - ☐ O método retorna uma exceção quando é executado com uma entrada inválida?

Porque técnicas de teste?

- ☐ Testes exaustivos (todas as possíveis entradas e condições) são geralmente impossíveis
- ☐ É usado um subconjunto de todos os possíveis casos de teste:
 - ☐ o subconjunto deve ter alta probabilidade de detectar defeitos
- ☐ Deve-se selecionar os casos de teste de modo adequado:
 - ☐ Pessoas diferentes têm a mesma probabilidade de detectar defeitos
 - ☐ Independência das habilidades individuais de pessoas
 - ☐ Efetividade dos testes: detectar um número maior de defeitos
 - ☐ Eficiência dos testes: detectar defeitos com menos esforço

Testes aleatórios

- ❑ Selecionar aleatoriamente casos de teste
- ❑ Criação e execução simultânea de um teste
- ❑ Também conhecidos como testes ad-hoc
- ❑ Características:
 - ❑ Usualmente, envolve muitas pessoas
 - ❑ Usualmente, depende da experiência dos testadores
 - ❑ Não há idéia sobre o grau da cobertura
 - ❑ Permitem explorar situações que não foram previamente identificadas
 - ❑ Devem ser utilizados em parceria com outras técnicas

Testes aleatórios não são suficientes ...

Exemplo:
Maior entre 2 números inteiros

```
...  
int maior;  
if (x>y) then  
    maior=x;  
else  
    maior=x;  
...
```

Testes aleatórios não são suficientes ...

Exemplo:

Maior entre 2 números inteiros

```
...  
int maior;  
if (x>y) then  
    maior=x;  
else  
    maior=x;  
...
```

Casos de teste:

x = 3 ; y = 2

x = 4 ; y = 3

x = 5 ; y = 1

x = 6 ; y = 4



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

7

Testes aleatórios não são suficientes ...

Exemplo:

Maior entre 2 números inteiros

```
...  
int maior;  
if (x>y) then  
    maior=x;  
else  
    maior=x;  
...
```

Casos de teste:

x = 3 ; y = 2 ☒

x = 4 ; y = 3 ☒

x = 5 ; y = 1 ☒

x = 6 ; y = 4 ☒



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

8

Testes aleatórios não são suficientes ...

Exemplo:
Maior entre 2 números inteiros

```
...  
int maior;  
if (x>y) then  
    maior=x;  
else  
    maior=x;  
...
```

Casos de teste:

x = 3 ; y = 2 ✓

x = 4 ; y = 3 ✓

x = 5 ; y = 1 ✓

x = 6 ; y = 4 ✓

Então, tudo Ok?

Testes aleatórios não são suficientes ...

Exemplo:
Maior entre 2 números inteiros

```
...  
int maior;  
if (x>y) then  
    maior=x;  
else  
    maior=x;  
...
```

Casos de teste:

x = 3 ; y = 2

x = 4 ; y = 3

x = 5 ; y = 1

x = 6 ; y = 4

Defeito não detectado

Casos de teste:

x = 3 ; y = 2

x = 2 ; y = 3

Defeito detectado

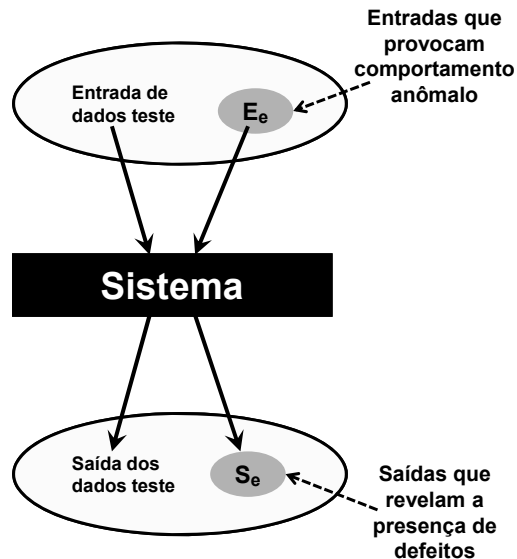
Técnicas de teste

- ☐ Procedimento para selecionar ou projetar testes
 - ☐ Baseado no modelo funcional ou estrutural do sistema
 - ☐ Deve permitir a detecção de defeitos
- ☐ Mecanismo para gerenciar testes
- ☐ Mecanismo para medir grau da cobertura dos testes
- ☐ Técnicas de teste:
 - ☐ Caixa preta (*Black box*)
 - ☐ Caixa branca (*White box*)

Testes de caixa preta

- ☐ Abordagem onde os testes são **derivados da especificação** do programa ou do componente
- ☐ O sistema é uma “**caixa preta**” cujo comportamento somente pode ser determinado estudando-se suas entradas e saídas relacionadas
- ☐ Também chamado de **teste funcional**:
 - ☐ O testador se preocupa somente com a funcionalidade e não com a implementação do software

Modelo de teste de caixa preta



- ☐ Igualmente aplicável a sistemas que são organizados em funções ou por objetos
- ☐ O testador apresenta as entradas ao componente ou ao sistema e examina as saídas correspondentes
- ☐ Uma saída não prevista representa um **teste com sucesso**

Importância dos requisitos

- ☐ Detecção de falhas depende da especificação/análise de requisitos
 - ☐ especificações ambíguas, incompletas, incorretas e inconsistentes são perigosas para os testes
 - ☐ especificações devem ser validadas
- ☐ Uma especificação deve ser **testável**, isto é, deve conter informações suficientes para os testes

Técnicas de teste de caixa preta

- ☐ Partição de equivalência
- ☐ Análise de valor limite
- ☐ Grafo causa-efeito
- ☐ Testes de transição de estados
- ☐ ...



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

15

Partição de equivalência

- ☐ Usualmente, os **dados de entrada** de um programa se dividem em **diferentes classes com características comuns**:
 - ☐ Números positivos, números negativos, strings sem “brancos”
 - ☐ **Se um programa deve aceitar um número negativo como entrada, então testar com um único número negativo deve ser suficiente**
- ☐ Estas classes são chamadas de **partições de equivalência** ou **classes de equivalência**
- ☐ Os membros de uma classe provocam um **comportamento equivalente** nos programas
 - ☐ Logo, dados pertencentes a uma mesma partição têm capacidade de revelar os mesmos defeitos



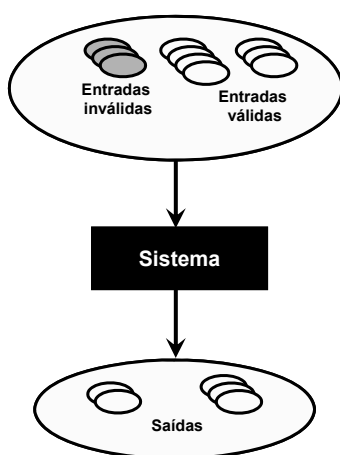
Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

16

Partição de equivalência

- ❑ A abordagem é identificar todas as partições de equivalência que serão utilizadas por um programa
- ❑ Os **casos de teste** devem ser projetados para que as entradas e saídas fiquem **dentro destas partições**
- ❑ **Objetivos:**
 - ❑ Minimizar números de casos de teste por partições de equivalência
 - ❑ Maximizar cobertura do domínio de entrada/saída ⇒ maximizar o número de defeitos detectados!

Partição de equivalência



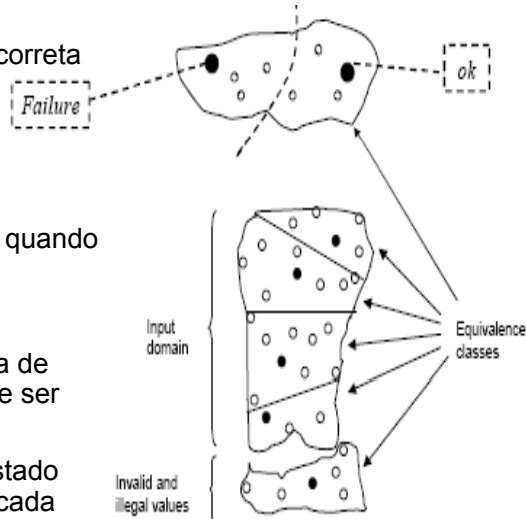
- ❑ **Partições de equivalência de entradas** são conjuntos de dados em que todos os membros do conjunto devem ser processados de maneira equivalente
- ❑ **Partições de equivalência de saídas** são saídas de programa que têm características comuns
- ❑ Uma boa diretriz é escolher casos de teste nos limites das partições e também aqueles próximos ao ponto médio da partição

Hipótese de teste de caixa preta

- ❑ Cada valor de uma partição de equivalência resulta na execução correta quando usado como entrada ao sistema

OU

- ❑ Cada valor de uma partição de equivalência resulta em uma falha quando usado como entrada ao sistema
- ❑ Geração de casos de teste:
 - ⇒ Um valor representativo de entrada de cada partição de equivalência pode ser suficiente
- ❑ **Na prática**, o sistema deve ser testado com vários valores de entrada de cada partição de equivalência



Exemplos

- ❑ **Faixa de valores** ⇒ 1 partição de equivalência válida e 2 inválidas
 - ❑ “número inteiro x deve estar entre 100 e 200, inclusive”
 - ❑ $\{\text{inteiro } x \mid 100 \leq x \leq 200\}$
 - ❑ $\{\text{inteiro } x \mid x < 100\}$
 - ❑ $\{\text{inteiro } x \mid x > 200\}$
- ❑ **Um valor específico dentro de uma faixa** ⇒ 1 partição de equivalência válida e 2 inválidas
 - ❑ “número inteiro x deve ser 100”
 - ❑ $\{\text{inteiro } x \mid x = 100\}$
 - ❑ $\{\text{inteiro } x \mid x < 100\}$
 - ❑ $\{\text{inteiro } x \mid x > 100\}$

Exemplos

- ❑ **Conjunto de valores** \Rightarrow uma partição de equivalência válida e uma inválida
 - ❑ “dia da semana x deve ser um dia de trabalho”
 - ❑ $x \in \{\text{Segunda, Terça, Quarta, Quinta, Sexta}\}$
 - ❑ $x \in \{\text{Sábado, Domingo}\}$
- ❑ **Valor Booleano** \Rightarrow uma partição de equivalência válida e uma inválida
 - ❑ “condição x deve ser *true*”
 - ❑ $x = \text{true}$
 - ❑ $x = \text{false}$



Valores ilegais

- ❑ Deve-se ter uma ou mais partições de equivalência para valores ilegais: valores que são incompatíveis com o tipo do parâmetro
 - ❑ “valores inteiros x ”
 - ❑ $\{\text{real } x\}$
 - ❑ $\{\text{string } x\}$
- ❑ Para cada valor de entrada válido, inválido ou ilegal:
 - ❑ Se o sistema **deve tratar de modo diferente** \rightarrow
então **uma partição de equivalência deve ser criada**



Passos para encontrar as partições

1. Decompor o programa em funções
2. Identificar as variáveis que determinam o comportamento de cada função
3. Particionar os valores de cada variável em classes de equivalência (válidas e inválidas)
4. Especificar os casos de teste:
 - a) eliminar as classes impossíveis ou os casos desinteressantes
 - b) selecionar casos de testes cobrindo as classes válidas das diferentes variáveis
 - c) para cada classe inválida escolha um caso de teste que cubra 1 e somente 1 de cada vez



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

23

Exercício

- ☐ Considere o teste do procedimento:
`boolean validaNovaSenha(String senha)`
- ☐ As regras de validação são:
 - ☐ Uma senha deve ter de 6 a 10 caracteres
 - ☐ O primeiro caractere dever ser alfa-numérico ou o símbolo “?”
 - ☐ Os demais caracteres só não podem ser caracteres de controle
 - ☐ A senha não pode existir em um dicionário



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)


24

Solução do exercício		
Variáveis	Válidas	Inválidas
tamanho	P1. tamanho $\in [6, 10]$	P7. tamanho < 6 P8. tamanho > 10
1º caractere (Car1)		
outros caracteres (Outro)		
valor		

Análise de valores-limite

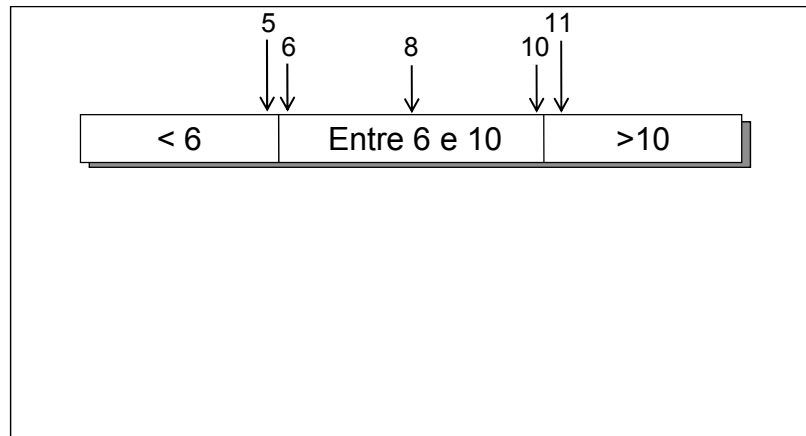
“Bugs lurk in corners and congregate at boundaries ...”
 Boris Beizer

- ☐ Critério de seleção que identifica valores nos limites das partições de equivalência
- ☐ Exemplos:
 - ☐ valor mínimo (máximo) igual ao mínimo (máximo) válido
 - ☐ uma unidade abaixo do mínimo
 - ☐ uma unidade acima do máximo
 - ☐ arquivo vazio
 - ☐ arquivo maior ou igual à capacidade máxima de armazenamento
 - ☐ cálculo que pode levar a *overflow* ou *underflow*
 - ☐ erro no primeiro ou no último registro



Exemplo

Tamanho da senha (parâmetro de entrada)



Limitação

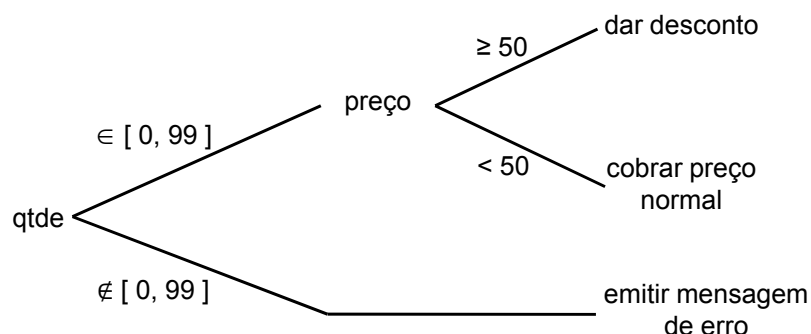
- ☐ Testes baseados em partições de equivalência ou análise de valores-limite consideram cada valor de entrada isoladamente
- ☐ E se existirem combinações de valores que constituam situações interessantes a serem testadas?

Análise causa-efeito

- ❑ Necessária quando se deseja testar combinações de entradas
- ❑ Utiliza tabelas de decisão e árvores de decisão
 - ❑ grafo causa-efeito como modelo auxiliar
- ❑ **Causas:** condições de entrada (valor lógico)
- ❑ **Efeitos:** ações realizadas em resposta às diferentes condições de entrada

Exemplo: Árvore de decisão

- ❑ **Causa:** $\text{preço} \geq 50$ e $0 \leq \text{qtd} \leq 99$
- ❑ **Efeito:** fornecer 5% de desconto



Exemplo: Tabela de decisão

Condições de entrada (causas)	preço ≥ 50	V	F	X
	$0 \leq \text{qtd} \leq 99$	V	V	F
Ações (efeitos)	dar desconto	X		
	cobrar preço normal		X	
	emitir msg de erro			X

regra

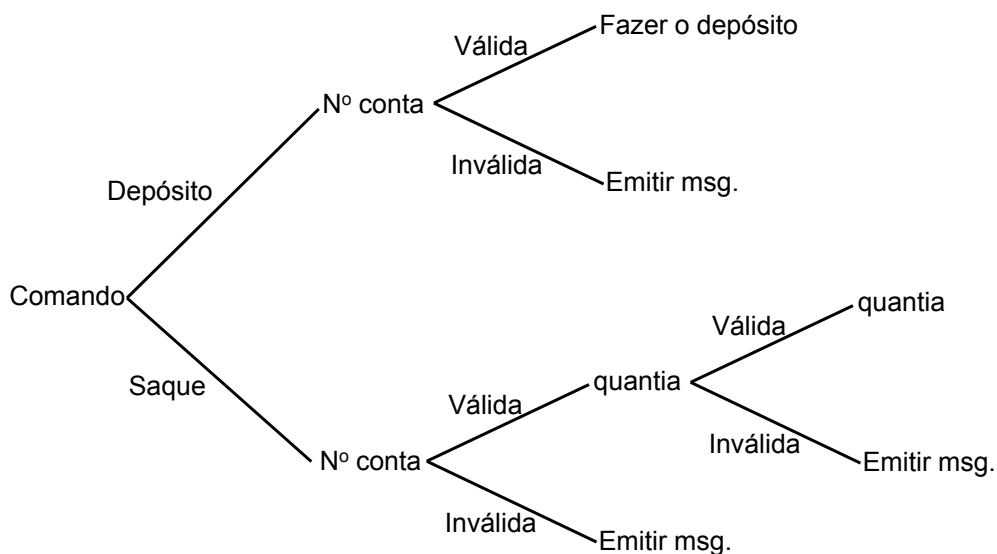
Geração de testes

- ❑ Tabela de decisão
 - ❑ critério: exercitar cada regra pelo menos 1 vez
- ❑ Árvore de decisão
 - ❑ critério: exercitar cada caminho da raiz até a folha pelo menos 1 vez
- ❑ Exemplos de regras para os casos de teste:
 1. preço ≥ 50 e $0 \leq \text{qtd} \leq 99$
 2. preço > 50 e $0 \leq \text{qtd} \leq 99$
 3. preço ≥ 50 e qtd = 100
- ❑ Limitação das tabelas de decisão
 - ❑ Número de regras a serem testadas: 2^N , onde N é o nº de causas
 - ❑ Será que todas interessam?

Exercício

- ❑ Considere um sistema bancário que trate somente duas transações:
 - ❑ Depósito: nº da conta, quantia
 - ❑ Saque: nº da conta, quantia
- ❑ Requisitos:
 - ❑ Se o comando for **depósito** e o **nº da conta** é válido então a **quantia** é depositada
 - ❑ Se o comando for **saque** e o **nº da conta** é válido e a **quantia** é válida ($0 < \text{quantia} \leq \text{saldo}$) então a **quantia** é sacada
 - ❑ Se o comando ou **nº da conta** ou a **quantia** é inválida, então exibir mensagem de erro apropriada

Solução do exercício



Testes caixa-branca

- ❑ Também conhecido como **teste estrutural**
- ❑ Testes baseados na **estrutura do programa**:
 - ❑ São baseados no código
 - ❑ Visam exercitar estruturas de controle (instruções) e de dados de um programa (não todas as combinações de caminhos)
 - ❑ Especificação do sistema não é considerada
 - ❑ Usualmente para **testes de unidade (pequenas partes)**
- ❑ Modelos de teste:
 - ❑ Grafo de fluxo de controle
 - ❑ Grafo de fluxo de dados



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

38

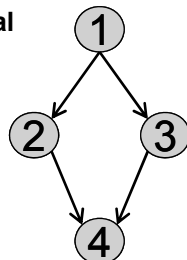
Grafo de fluxo de controle

nó = bloco de comandos seqüenciais
aresta ou ramo = transferência de controle

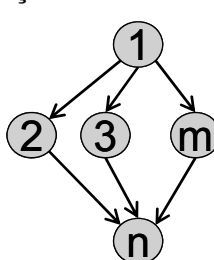
Bloco Seqüencial



Seleção

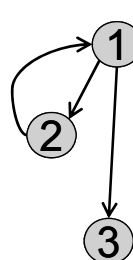


if-then-else

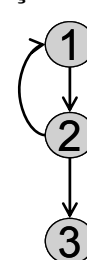


case (switch)

Repetição



while



repeat-until



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

39

Exemplo

Cálculo de x^y

```
1. read x, y;  
2. if y < 0  
3.   then p := 0 - y  
4.   else p := y;  
5. z := 1.0;  
6. while p ≠ 0 do  
7. begin  
   z := z * x; p := p - 1;  
end;  
8. if y < 0  
9.   then z := 1 / z;  
10. write z;  
    end;
```



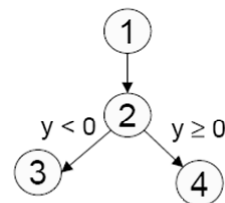
Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

40

Exemplo

Cálculo de x^y

```
1. read x, y;  
2. if y < 0  
3.   then p := 0 - y  
4.   else p := y;  
5. z := 1.0;  
6. while p ≠ 0 do  
7. begin  
   z := z * x; p := p - 1;  
end;  
8. if y < 0  
9.   then z := 1 / z;  
10. write z;  
    end;
```



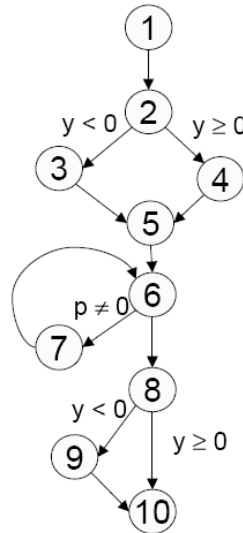
Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

41

Exemplo

Cálculo de x^y

```
1. read x, y;  
2. if y < 0  
3.   then p := 0 - y  
4.   else p := y;  
5. z := 1.0;  
6. while p ≠ 0 do  
7.   begin  
8.     z := z * x; p := p - 1;  
9.   end;  
10. write z;  
end;
```



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

42

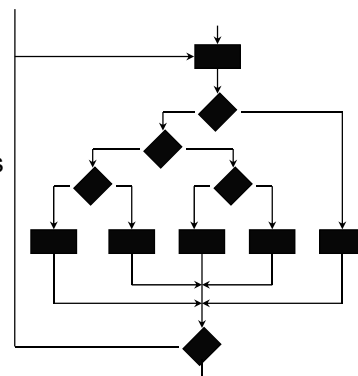
Seleção de casos de teste

- ❑ Critério de cobertura de testes
- ❑ Objetivos:
 - ❑ geração de testes: determinação dos dados de teste
 - ❑ avaliação final: indicação de término dos testes
- ❑ Todos os caminhos?

Inviável!

Exemplo: repetição

- Corpo da repetição tem 5 caminhos
- Se a repetição é executada 20 vezes
⇒ $5^{20} = 95.367.431.640.625$ caminhos diferentes



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

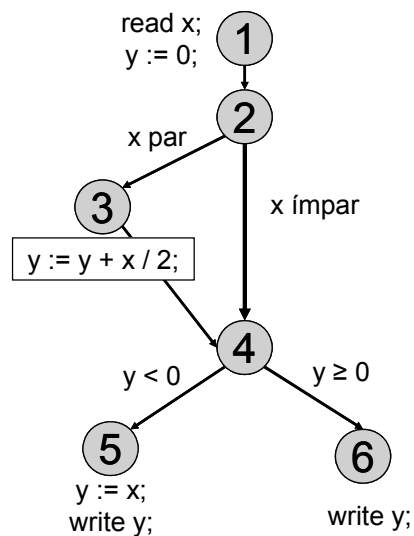
43

Critério de cobertura de testes

- ❑ Tipos:
 - ❑ cobertura de instruções
 - ❑ cobertura de decisões/condições
 - ❑ cobertura de caminhos

Testes de instruções

```
1. read x; y:=0;  
2. if x é par then  
3.   y := y + x/2;  
   endif  
4. if y < 0 then  
5.   y := x;  
   write y;  
6. else  
   write y;  
   endif  
end;
```



Testes de instruções

```

graph TD
    1((1)) --> 2((2))
    2 -- "x par" --> 3((3))
    2 -- "x ímpar" --> 4((4))
    3 --> 4
    4 -- "y < 0" --> 5((5))
    4 -- "y ≥ 0" --> 6((6))
    5 --> 6
    
```

Critério: cada instrução deve ser executada pelo menos 1 vez

nós	predicados	dados
{1,2,3,4,5}	x par negativo	-2
{1,2,3,4,6}	x par positivo	2

UNIVALI

Prof. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

46

Testes de instruções

```

graph TD
    1((1)) --> 2((2))
    2 -- "x par" --> 3((3))
    2 -- "x ímpar" --> 4((4))
    3 --> 4
    4 -- "y < 0" --> 5((5))
    4 -- "y ≥ 0" --> 6((6))
    5 --> 6
    
```

Critério: cada instrução deve ser executada pelo menos 1 vez

nós	predicados	dados
{1,2,3,4,5}	x par negativo	-2
{1,2,3,4,6}	x par positivo	2

O ramo “2 → 4” não é executado

UNIVALI

Prof. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

47

Exercício

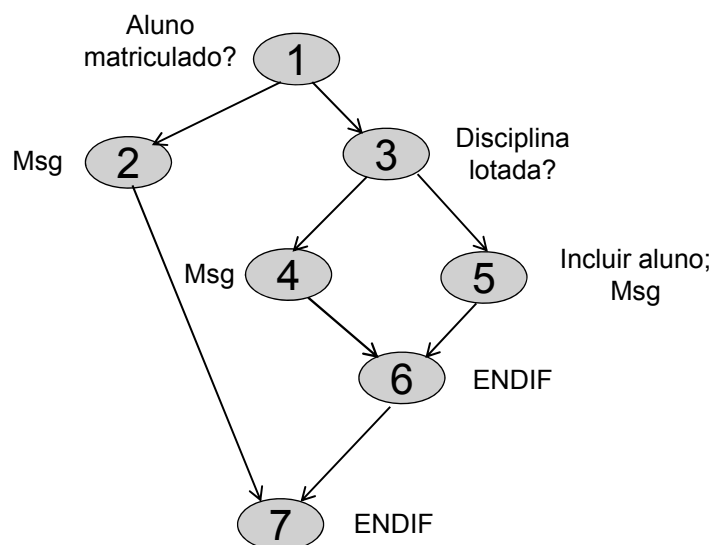
```
matricular(aluno, disciplina)
{
  1. IF aluno ∈ disciplina
  2.   THEN msg "já matriculado"
  3.   ELSE IF disciplina lotada
  4.     THEN msg "disciplina lotada"
  5.     ELSE incluir aluno na disciplina;
           msg "aluno incluído"
  6.   END
  7. END
}
```



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

48

Exercício



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

49

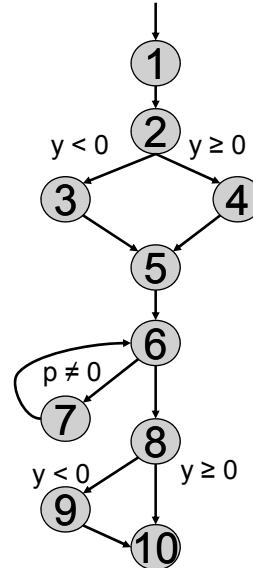
Testes de decisões

Cálculo de x^y

```

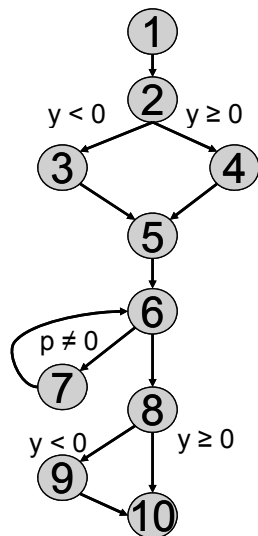
1. read x, y;
2. if y < 0
3.   then p := 0 - y
4.   else p := y;
5. z := 1.0;
6. while p ≠ 0 do
7.   begin
8.     z := z * x; p := p - 1;
9.   end;
10. if y < 0
11.   then z := 1 / z;
12. write z;
13. end;

```



Testes de decisões

Critério: cada ramo deve ser percorrido pelo menos 1 vez



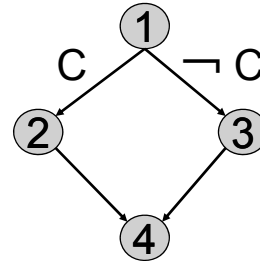
ramos	predicados	dados
{(1,2), (2,3), (3,5), (5,6), (6,7), (7,6), (6,8), (8,9), (9,10)}	$\forall x, y < 0$	(4, -1)
{(1,2), (2,4), (4,5), (5,6), (6,7), (7,6), (6,8), (8, 10)}	$\forall x, y = 0$	(4, 0)

Teste de decisões

```

1. if a >= 0 and a <= 200
2.   then m := 1
3.   else m := 3
4. ...
    
```

Supor que o
correto seria
 $a < 200$



Ramo	Dados
{(1,2), (2,4)}	a = 5
{(1,3), (3,4)}	a = -5



O critério é satisfeito, mas o defeito não é detectado

Teste de decisões

- ☐ Auxilia a detectar ramos que não podem ser executados
- ☐ Engloba completamente a cobertura de instruções
- ☐ ..., mas não considera todas as dependências entre os ramos
⇒ cobertura de caminhos
- ☐ Insuficiente para o teste de condições complexas e compostas
- ☐ Não detecta diretamente ramos faltantes
- ☐ É considerado o critério mínimo para testes

Testes de caminhos básicos

- ☐ **Critério:** todos os caminhos possíveis do grafo devem ser percorridos pelo menos 1 vez
- ☐ **Dificuldades:**
 - ☐ executabilidade: nem todos os caminhos no grafo são executáveis pelo programa
 - ☐ grafos com ciclos: nº de caminhos pode ser indeterminado ou infinito
 - ☐ necessidade de critérios que permitam limitar o nº de caminhos:
 - ☐ testes de caminhos básicos
 - ☐ testes de laços



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

54

Testes de caminhos básicos

- ☐ Em vez de todos os caminhos, busca-se os caminhos independentes:
 - ☐ Caminho independente → contém pelo menos 1 nova aresta do grafo de controle
 - ☐ O nº de caminhos independentes é dado pela complexidade ciclomática $V(G)$ de McCabe (1976):
$$V(G) = \text{nº de predicados} + 1 \quad // \text{ sem gotos}$$
 - ☐ Os predicados são as condições existentes no programa
 - ☐ Uma vez que todos os outros caminhos do grafo são combinações dos caminhos independentes, $V(G)$ é o limite inferior do nº de testes de caminhos



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

55

Testes de caminhos básicos

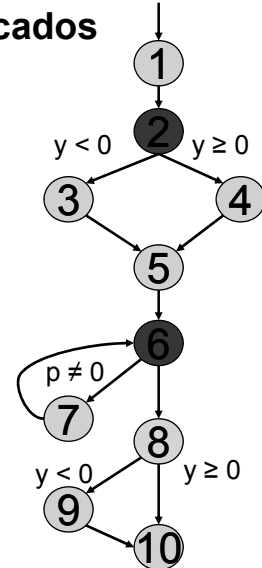
Cálculo de x^y

```

1. read x, y;
2. if y < 0
3.   then p := 0 - y
4.   else p := y;
5. z := 1.0;
6. while p ≠ 0 do
7.   begin
8.     z := z * x; p := p - 1;
9.   end;
10. if y < 0
11.   then z := 1 / z;
12.   write z;
13. end;

```

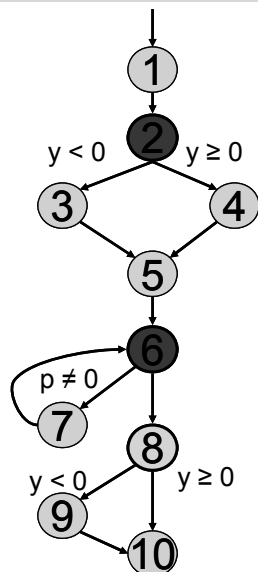
2 predicados



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

56

Testes de caminhos básicos



Critério: cada caminho independente deve ser percorrido pelo menos 1 vez

$$V(G) = n^{\circ} \text{ nós predicados} + 1 = 3$$

nós	predicados	dados
{1-2-4-5-6-8-10}	$\forall x, y = 0$	(0, 0)
{1-2-4-5- <u>6-7-6</u> -8-10}	$\forall x, y > 0$	(4, 2)
{1-2-3-5-6-7-6-7- <u>8-9</u> -10}	$\forall x, y < 0$	(-4, -2)



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

57

Exercício

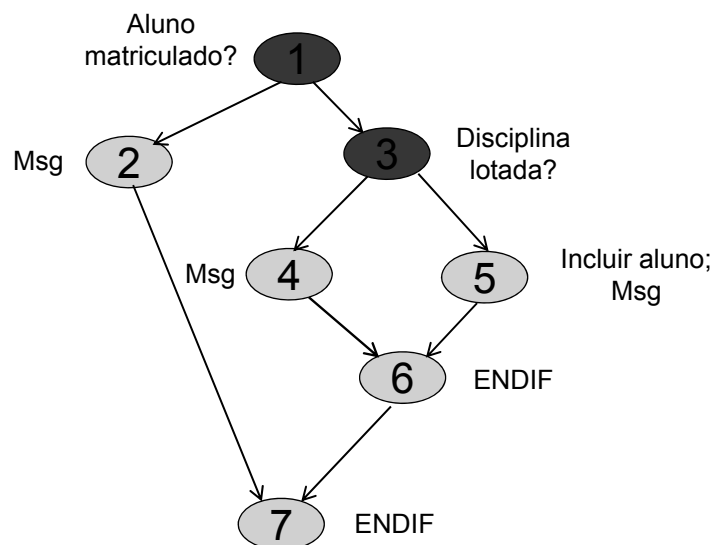
```
matricular(aluno, disciplina)
{
  1. IF aluno ∈ disciplina
  2.   THEN msg "já matriculado"
  3.   ELSE IF disciplina lotada
  4.     THEN msg "disciplina lotada"
  5.     ELSE incluir aluno na disciplina;
           msg "aluno incluído"
  6.   END
  7. END
}
```



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

58

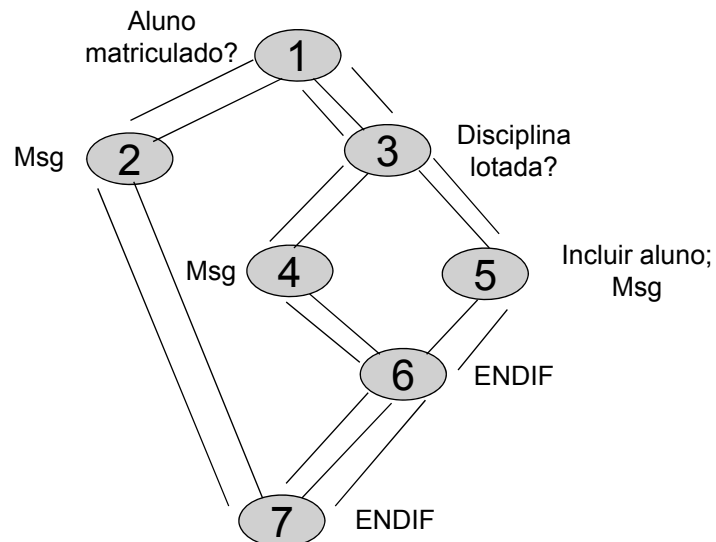
Exercício



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

59

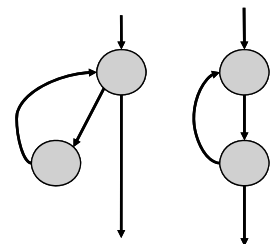
Exercício



Teste de laços

❑ Procedimento:

1. Pule o laço
2. Passe pelo laço 1 vez
3. Passe pelo laço n vezes, onde $n < \max$
4. Passe pelo laço $\max-1$ vezes
5. Passe pelo laço \max vezes
6. Tente passar pelo laço $\max+1$ vezes

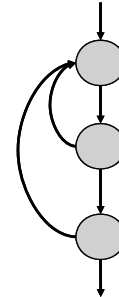


Laços Simples

Teste de laços

☐ Procedimento:

1. Inicie pelo laço mais interno; fixe os outros nos valores mínimos
2. Realize testes para laços simples
3. Caminhe para fora, realizando testes no laço seguinte e mantendo os demais nos valores mínimos
4. Continue até que todos tenham sido testados



Laços Aninhados

Comparação

☐ Teste caixa preta:

- ☐ Explosão combinatorial do número de casos de teste (dados válidos e inválidos)
- ☐ Frequentemente, não fica claro se os casos de teste selecionados descobrirão um determinado defeito ou não
- ☐ Não detecta *features*/casos de uso faltantes

☐ Teste caixa branca:

- ☐ Potencial número infinito de caminhos que precisam ser testados
- ☐ Frequentemente, testa-se o que é feito ao invés de o que deverá ser feito
- ☐ Não detecta *features*/casos de uso faltantes
- ☐ Permite detectar código inútil

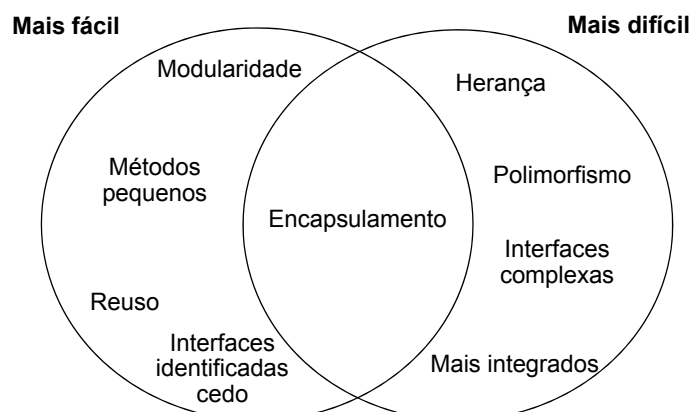
... e na prática?

- ❑ Combinação de testes caixa-preta seguidos por testes caixa-branca
- ❑ Alguns resultados:
 - ❑ Alta cobertura em si não garante detecção de defeitos
 - ❑ Detecção de defeitos aumenta significativamente com cobertura $\geq 95\%$
 - ❑ Testes de caminho são significativamente mais efetivos do que casos de teste aleatórios

[Hutchins et al. "Experiments on the Effectiveness of Dataflow- and Controlflow-Based Test Adequacy Criteria". ICST, May 1994]

Testando sistemas OO

Testes em sistemas OO são mais fáceis?



Níveis de testes em sistemas OO

- ☐ Teste de unidade:
 - ☐ testes de método
 - ☐ testes de classe
- ☐ Teste de integração:
 - ☐ testes de grupos de classes
- ☐ Testes de sistema ...

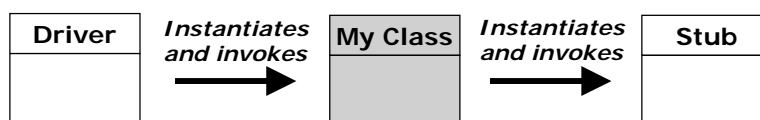


Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

66

Testes de classe

- ☐ Casos de teste de classe são criados com base na especificação da classe
- ☐ Cobertura total de teste de uma classe envolve:
 - ☐ Testar todas as operações/métodos da classe
 - ☐ Escrever (set) e ler (get) todos os atributos/variáveis da classe
 - ☐ Executar o objeto em todos os possíveis estados
- ☐ O teste de uma classe pode envolver outras classes
 - ☐ Escrever um **driver** para cada chamada (*simula quem chama*)
 - ☐ Escrever um **stub** para cada classe chamada (*simula classes invocadas*)



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

67

Testes aleatórios de classes

- ☐ Identificar operações aplicáveis a uma classe
- ☐ Identificar restrições para o seu uso
- ☐ Identificar seqüências mínimas de história de vida a serem testados
- ☐ Gerar uma variedade de seqüências de testes aleatoriamente (mas válidas)



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

68

Testes de particionamento de classes

- ☐ Basicamente, particionamento em classes de equivalência
- ☐ Particionamento baseado em estados
 - ☐ Categorizar e testar operações com base na sua habilidade de mudar o estado de uma classe
- ☐ Particionamento baseado em atributos
 - ☐ Categorizar e testar operações com base nos atributos usados
- ☐ Particionamento baseado em categorias
 - ☐ Categorizar e testar operações com base nas funções genéricas, p.ex:
 - ☐ inicialização
 - ☐ mudança de estado
 - ☐ buscas
 - ☐ finalização



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

69

Outras abordagens: Testes exploratórios

- ☐ Definir e executar testes simultaneamente
- ☐ Testes não são previamente definidos
- ☐ O testador exploratório escreve idéias de teste e as aplica (de forma sistemática)
- ☐ O testador pode continuamente aprender/conhecer melhor o sistema
- ☐ *"thinking-while-testing"*
- ☐ Usar quando:
 - ☐ Não há (ou só pouca) especificação dos requisitos
 - ☐ Não há (ou só pouco) conhecimento de domínio

Outras abordagens: Testes estatísticos

- ☐ Permitem o uso de técnicas de inferência estatística para calcular aspectos probabilísticos do processo de teste como confiabilidade, tempo médio entre falhas, etc
- ☐ Entradas são aleatoriamente definidas com base em uma distribuição de probabilidade, representando o uso esperado do software

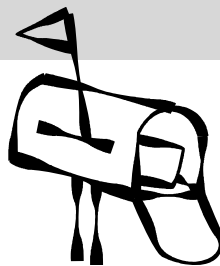
Bibliografia

- ❑ G.J.Myers. *The Art of Software Testing*. John Wiley & Sons, 1979.
- ❑ B.Beizer. *Software Testing Techniques*. International Thomson Computer
- ❑ R.Binder. *Testing OO Systems*. Addison Wesley, 2000.
- ❑ P.Jalote. *An Integrated Approach to Software Engineering*, 2ª ed. Springer, 1997.
- ❑ E.Martins. *Verificação e Validação de Software*. Notas de Curso, UNICAMP, 2001.
- ❑ J.F. Peters, W. Pedrycz. *Engenharia de Software: Teoria e Prática*.
- ❑ W. de Pádua Paula Fº. *Engenharia de Software*. Ed. LTC, 2ª ed., 2002.
- ❑ R.S.Pressman. *Software Engineering: A Practitioner's Approach*. 6ª ed, McGraw-Hill Science, 2004.
- ❑ I. Sommerville. *"Software Engineering"*. 7ª ed, Addison Wesley, 2004.



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

Contato



Marcello Thiry
marcello.thiry@gmail.com

LQPS
<http://www.univali.br/lqps>



Profs. Marcello Thiry e Christiane von Wangenheim – ES 2169 (Testes de Unidade)

73