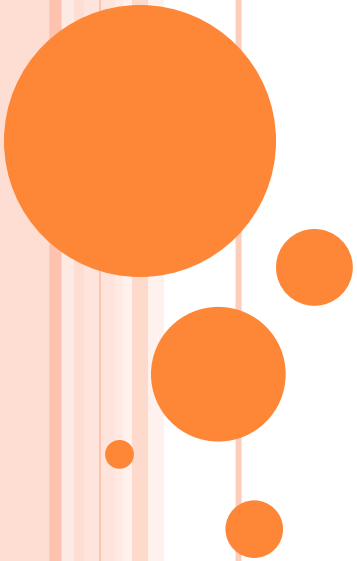


# Testes de Caixa branca

Paulyne Juca (paulyne@ufc.br)

# Definição

- Teste que leva em consideração o funcionamento interno de um sistema ou componente. Planejar o teste com conhecimento da estrutura e da implementação do software.

- Também chamado de:
  - Teste estrutural
  - Teste de caixa de vidro
  - Teste de caixa clara
  - Teste baseado em código

# Objetivo

- Garantir que as funcionalidades desenvolvidas funcionam corretamente
- Fornecer uma boa cobertura de testes:
  - Cobertura de comandos (% dos comandos existentes que foram testados)
    - Garantir que todos os comandos foram executados pelo menos uma vez
  - Cobertura de funções
  - Cobertura de caminhos (% de caminhos possíveis foi testado)
- Geralmente usado para testes unitários e de integração

# Vantagens

- Como conhece o código, fica mais fácil identificar que entrada ajudam efetivamente no teste
- Ajuda a otimizar o código
  - Se tem muitos caminhos, talvez a solução seja simplificar o código e não aumentar a quantidade de casos de teste
- Ajuda a encontrar códigos que nunca são executados e removê-los

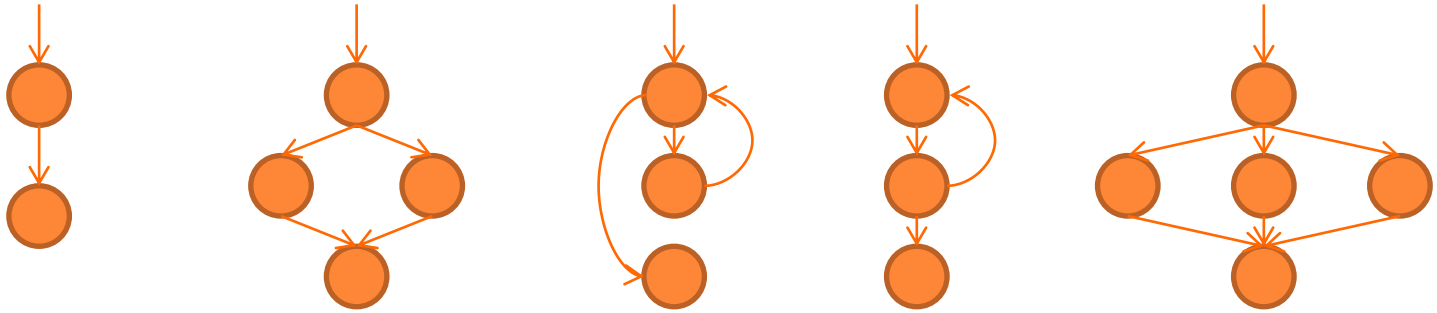
# Limitações

- O número de caminhos possíveis pode ser muito grande
- O caso de teste selecionado pode não revelar o defeito. Por exemplo:
  - y = x * 2; // deveria ser y = x ** 2
  - funciona corretamente para x = 0, y=0 e x = 2, y = 4.
- Caminhos inexistentes (não implementados) não podem ser testados
- Pode ser difícil de praticar em códigos grandes em projetos que não usaram TDD
- Os testes podem ser complexos e exigir profissionais mais qualificados
- Não garante conformidade com a especificação, apenas funcionamento sem erro.

# Passos

- Gerar o grafo do código a ser avaliado
- Calcular a complexidade ciclomática
- Identificar os casos de teste baseado em:
  - Comandos
  - Caminhos
- Calcular a cobertura

# Notação para o grafo



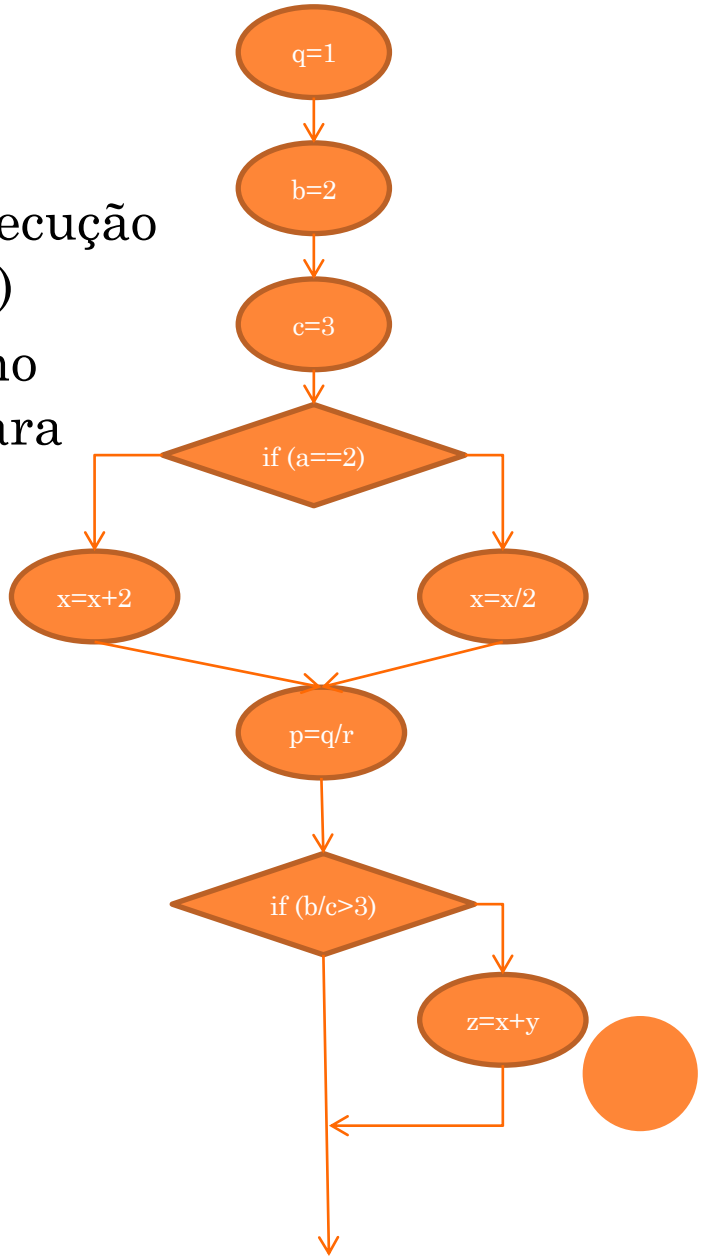
sequencia if while until case

# Grafo

- Sempre começa em um nó único
- Cada nó representa um comando de execução ou condição lógica (if, while, until, case)
  - Podemos representar comandos como círculo e condições como losangos para facilitar

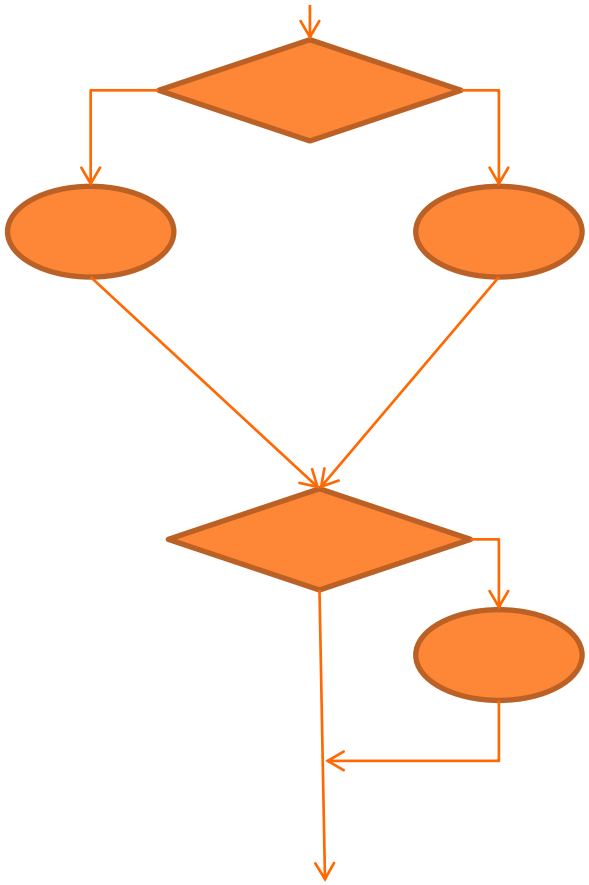
Ex:

```
q = 1 ;
b = 2 ;
c = 3 ;
i f ( a ==2) {
  x = x + 2 ;
} e l s e {
  x = x / 2 ;
}
p = q / r ;
i f ( b/ c>3) {
  z = x + y ;
}
```



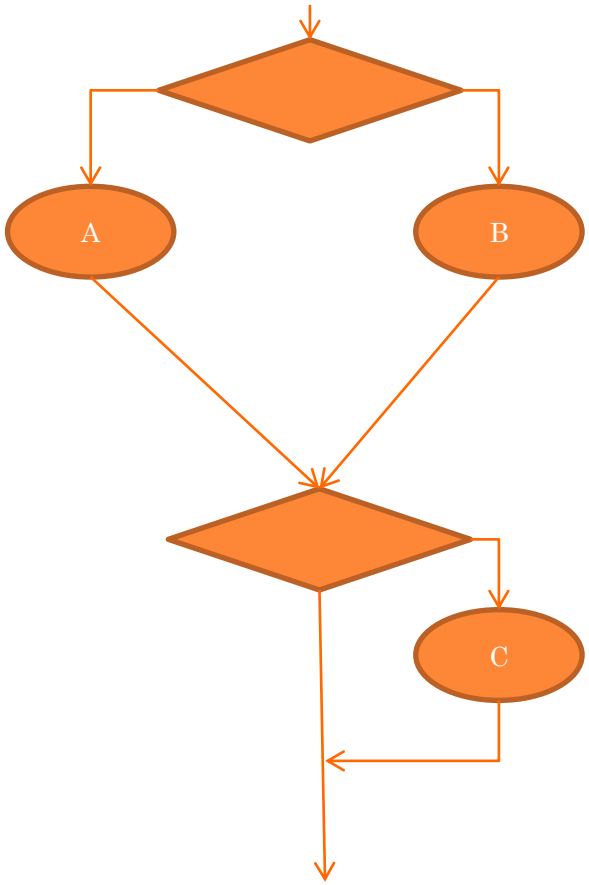
# Cobertura: Começando do começo

- If condicao1
  - Then comandoA
  - Else comandoB
- endIf
- If condicao2
  - Then comandoC
- endIF



# Cobertura: Começando do começo

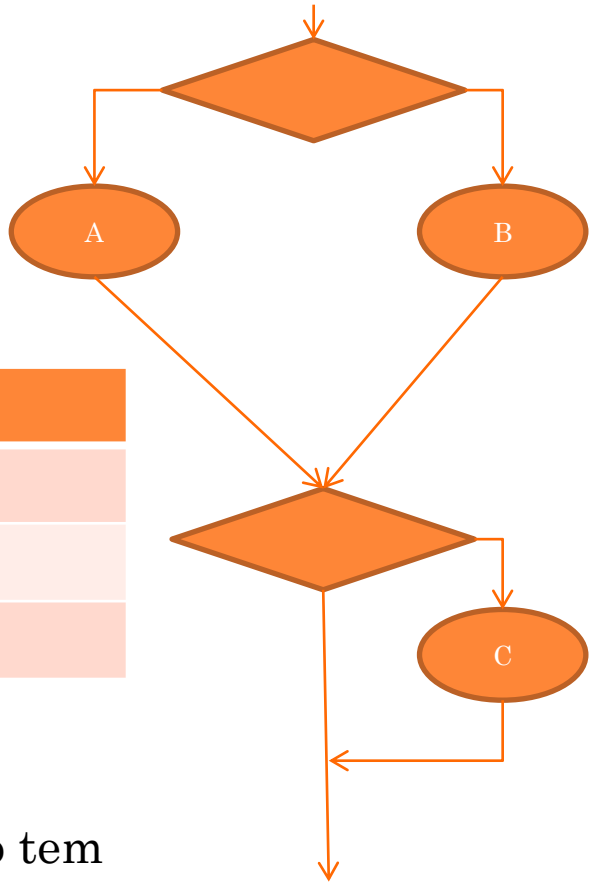
- If condicao1
  - Then comandoA
  - Else comandoB
- endIf
- If condicao2
  - Then comandoC
- endIF




As duas condições são independentes, certo?

# Cobertura

- Cobertura de comandos



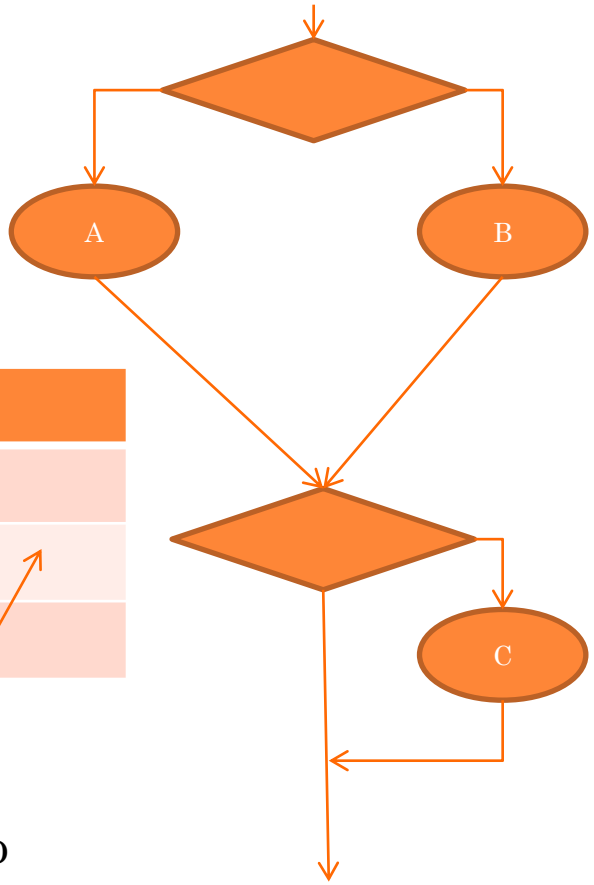

| Caso_teste | condicao1 | condicao2 |
|---|---|---|
| CT01 | TRUE | Não importa |
| CT02 | FALSE | Não importa |
| CT03 | Não importa | TRUE |

Condicao2 não tem comando para else, por isso não importa o FALSE na cobertura por comandos

# Cobertura

- Cobertura de comandos

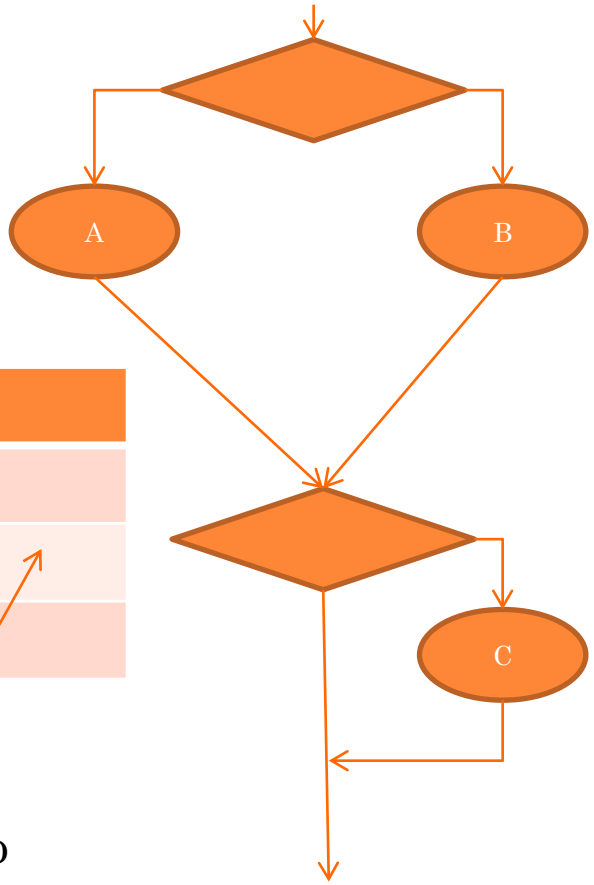
| Caso_teste | condicao1 | condicao2 |
|---|---|---|
| CT01 | TRUE | Não importa |
| CT02 | FALSE | Não importa |
| CT03 | Não importa | TRUE |




Condicao2 se o false não importa, podemos substituir o Não importa para TRUE

# Cobertura

- Cobertura de comandos




| Caso_teste | condicao1 | condicao2 |
|---|---|---|
| CT01 | TRUE | TRUE |
| CT02 | FALSE | TRUE |
| | | |

Condicao2 se o false não importa, podemos substituir o Não importa para TRUE. CT03 deixa de existir

100% de cobertura de comandos

# É uma boa cobertura?

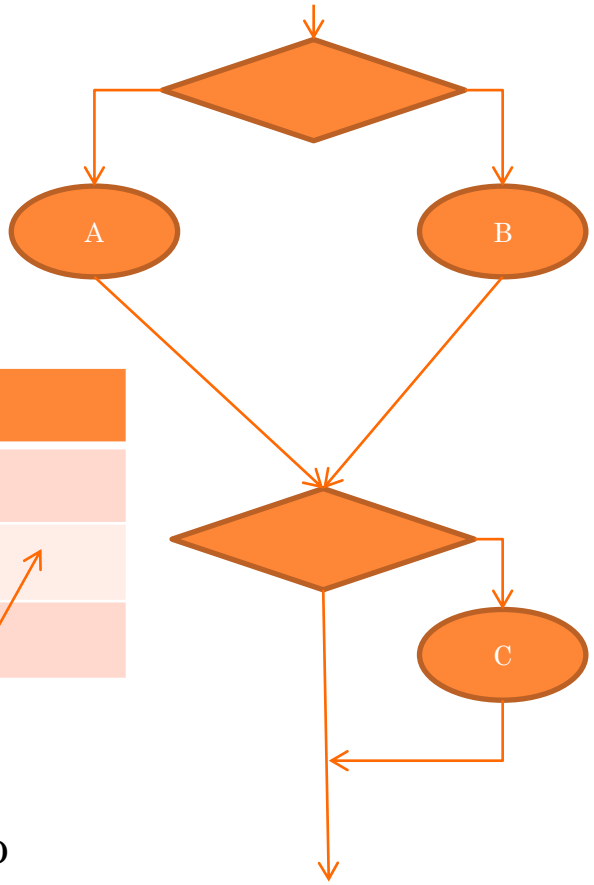
# É uma boa cobertura?

**Não totalmente. O que ficou de fora?**

# Cobertura

- Cobertura de caminhos

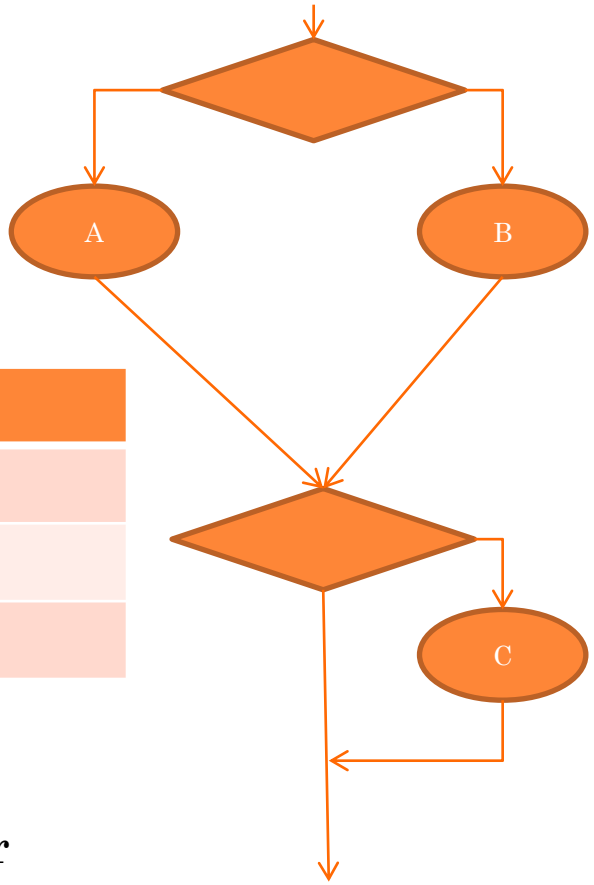
| Caso_teste | condicao1 | condicao2 |
|---|---|---|
| CT01 | TRUE | TRUE |
| CT02 | FALSE | TRUE |
| | | |




Condicao2 se o false nunca é testado.

# Cobertura

- Cobertura de caminhos




| Caso_teste | condicao1 | condicao2 |
|---|---|---|
| CT01 | TRUE | TRUE |
| CT02 | FALSE | FALSE |
| | | |

E se trocar por false qual minha cobertura?

100% de cobertura de comandos e 100% de cobertura de caminhos

# É uma boa cobertura?

# É UMA BOA COBERTURA?

**Bem melhor. Testamos todas as condições?**

# É UMA BOA COBERTURA?

**Bem melhor. Testamos todas as condições?**

**Sim. Faltou alguma coisa?**

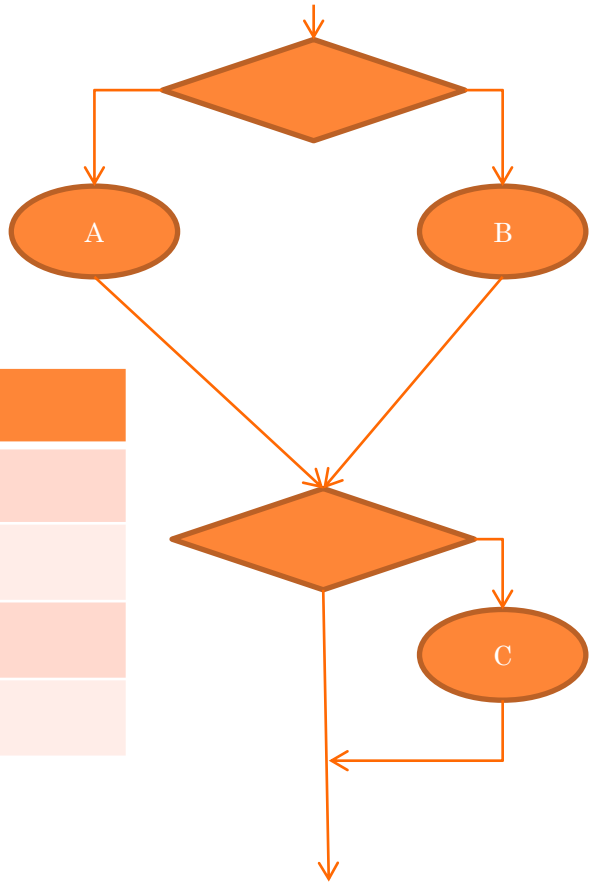
# Tipos de Condições

- Existem diferentes tipos de condições
  - Com um único operador, chamadas de atômicas
    - Ex: a >= b
    - Não usam and, or , xor
  - Expressões booleanas mais complexas (a and b or not c)

- Para condições atômicas nossos casos de teste resolvem bem.
  - Mas será que pode melhorar?

# Cobertura

- Conjunto completo de caminhos




| Caso_teste | condicao1 | condicao2 |
|---|---|---|
| CT01 | TRUE | TRUE |
| CT02 | FALSE | FALSE |
| CT03 | FALSE | TRUE |
| CT04 | TRUE | FALSE |

100% de cobertura de comandos e 100% de cobertura de caminhos

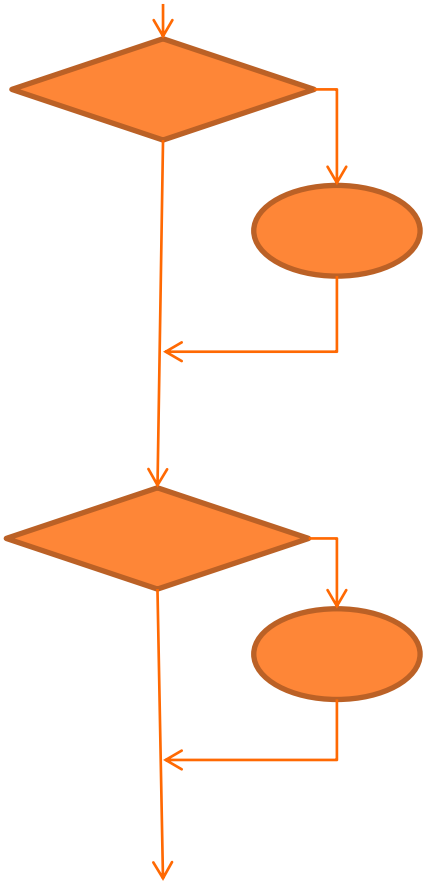
# É UMA BOA COBERTURA?

**Sim!!! Para o caso demonstrado, sim.**

# Cobertura

- Mas e se o comando alterar o valor e influenciar a condição?

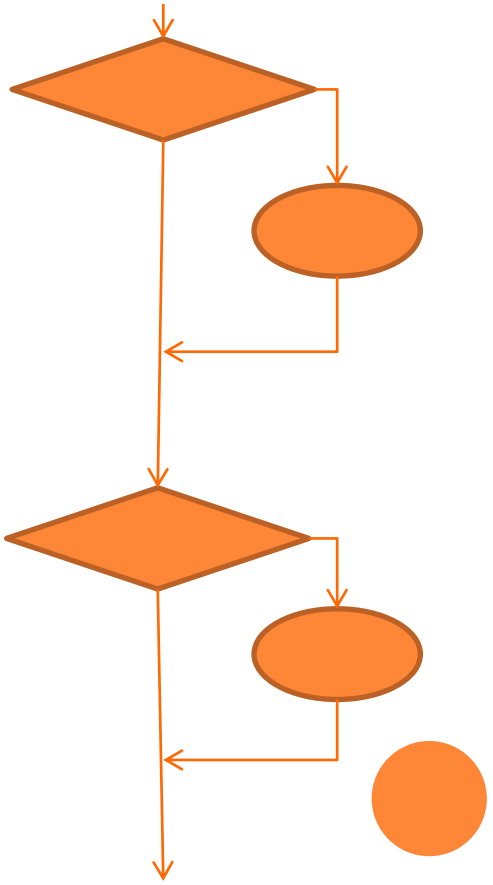
- If (x>1 and y=0)
  - Then z=z/x
- endIf
- If (x=2 or z>1)
  - Then z=z+1
- endIF



# Cobertura

- Para atingir os dois comandos com TRUE nas duas condições:
  - x = 2
  - y = 0
  - z = 4

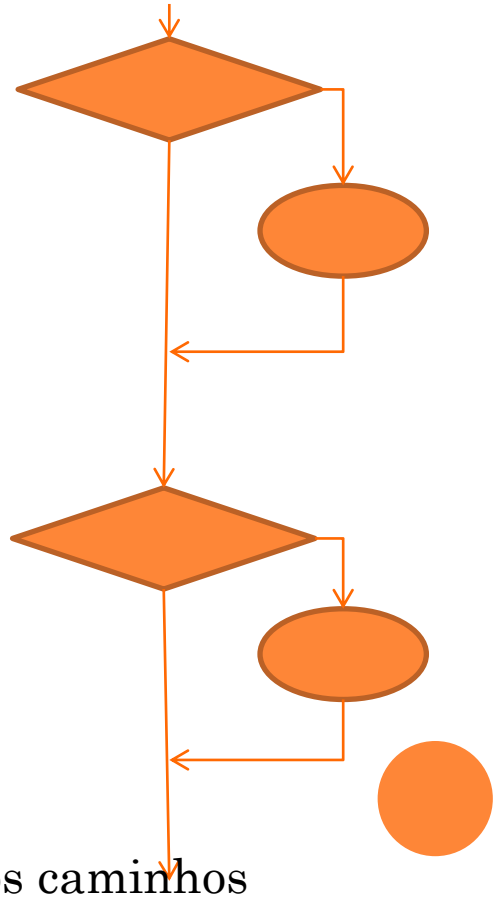
- If (x>1 and y=0)
  - Then z=z/x
- endIf
- If (x=2 or z>1)
  - Then z=z+1
- endIF



100% de cobertura de comandos

# Cobertura

- Para atingir os dois comandos com FALSE nas duas condições:
  - x = 1
  - y = 1
  - z = 1
- If (x>1 and y=0)
  - Then z=z/x
- endIf
- If (x=2 or z>1)
  - Then z=z+1
- endIF



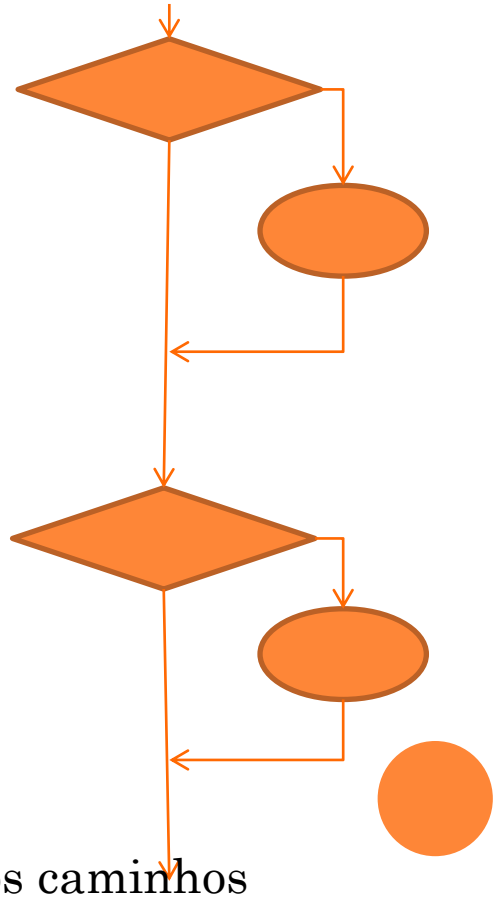
100% de cobertura de comandos e 100% de cobertura dos caminhos

# Cobertura

- Os dois outros caminhos possíveis são:
  - x = 4
  - y = 0
  - z = 4

  - x = 2
  - y = 1
  - z = 1

- If (x>1 and y=0)
  - Then z=z/x
- endIf
- If (x=2 or z>1)
  - Then z=z+1
- endIF



100% de cobertura de comandos e 100% de cobertura dos caminhos

# Cobertura de Caminhos

- Vantagem
  - Cobertura forte
- Desvantagem
  - Muito custoso

# Complexidade ciclomática

- Complexidade ciclomática: métrica que fornece uma medida quantitativa da complexidade do programa.
  - Ajuda a identificar a quantidade de caminhos independentes que existem no sistema que fornecem o número de casos de testes necessários para garantir que todos os comandos tenham sido executados pelo menos 1 vez.
  - Um caminho independente é qualquer caminho do programa que introduz pelo menos 1 novo comando ou condição
- Para o cálculo da complexidade ciclomática usaremos (não é o único modo)
  - Número de decisões lógicas + 1
  - Para condições compostas (a and b or c) somar um para cada condição

# Complexidade ciclomática

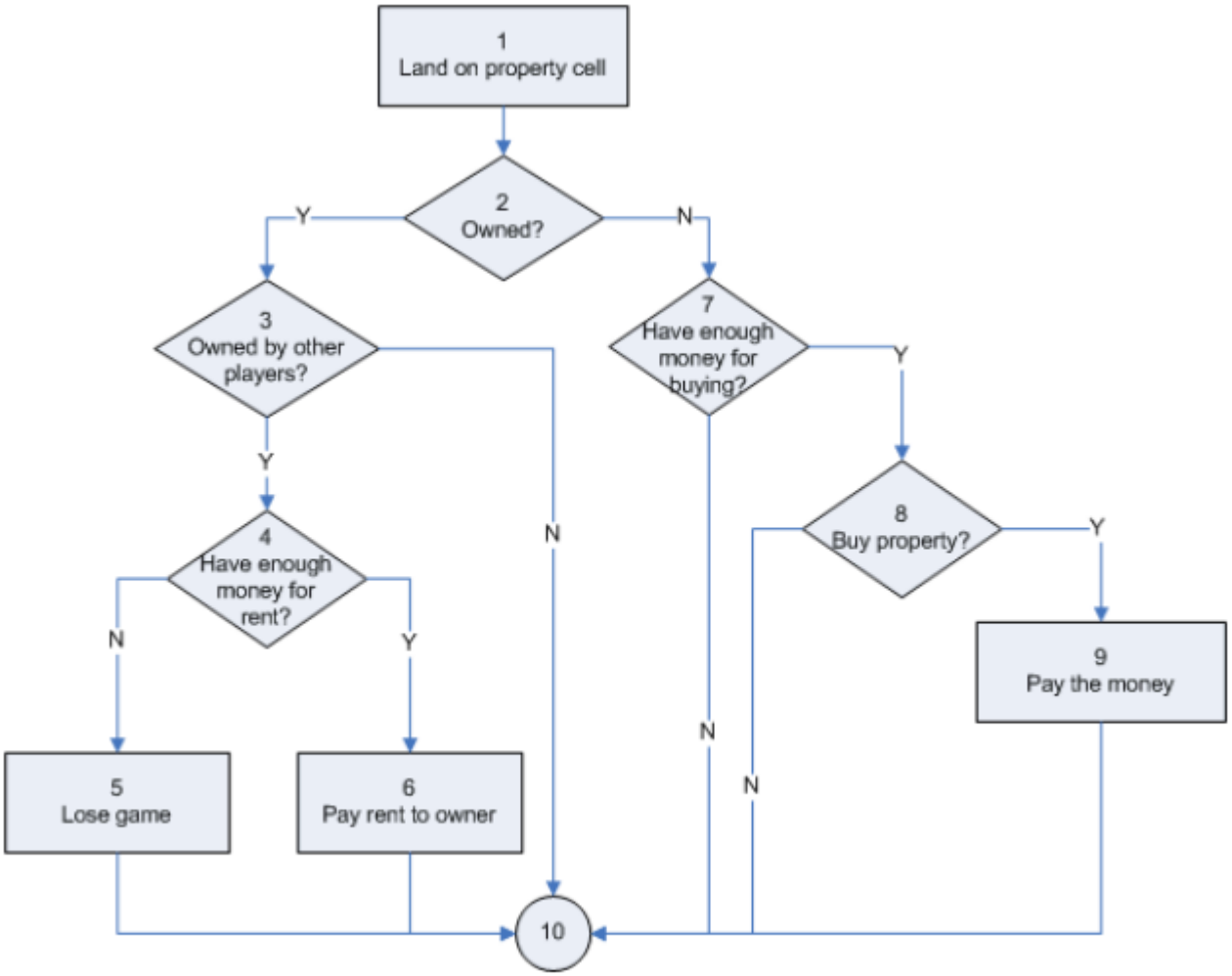



**Figure 1: Flowgraph of purchasing property**

# Complexidade ciclomática

- Complexidade 6. Caminhos possíveis:
  - 1-2-3-4-5-10 (property owned by others, no money for rent)
  - 1-2-3-4-6-10 (property owned by others, pay rent)
  - 1-2-3-10 (property owned by the player)
  - 1-2-7-10 (property available, don't have enough money)
  - 1-2-7-8-10 (property available, have money, don't want to buy it)
  - 1-2-7-8-9-10 (property available, have money, and buy it)

- Quanto mais alta a complexidade, mais difícil é testar esse sistema
  - Especialmente a quantidade de casos de teste para cobertura de caminhos cresce rápido

# Referências


- Engenharia de software – Sommerville – Cap 23
- Teste e análise de software – Pezzè
- Engenharia de software – Pressman
- http://www.youtube.com/watch?v=wLINA-Gj7eA
- http://softwaretestingfundamentals.com/white-box-testing/
- http://www.testinggeek.com/white-box-testing
- http://agile.csc.ncsu.edu/SEMaterials/WhiteBox.pdf
- http://www.buzzle.com/editorials/4-10-2005-68350.asp
- http://www.codeproject.com/Articles/37111/White-Box-Testing-Technique
- http://www.softwaretestinghelp.com/white-box-testing/
- http://www.inf.ufg.br/~auri/curso/arquivos/estrutural01.pdf
- http://www.inf.ufg.br/~auri/curso/arquivos/estrutural02.pdf
- http://www.devmedia.com.br/uma-visao-da-tecnica-de-teste-de-caixa-branca/15610