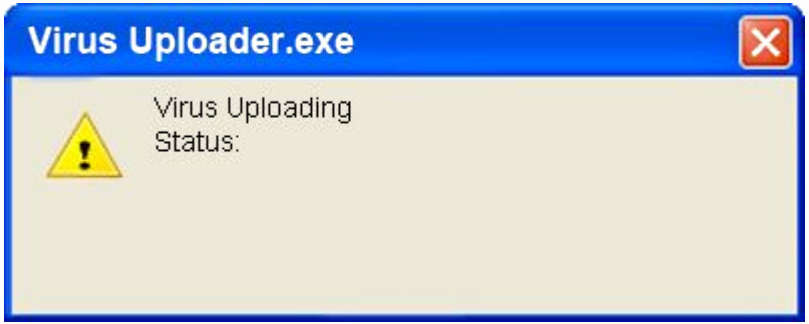


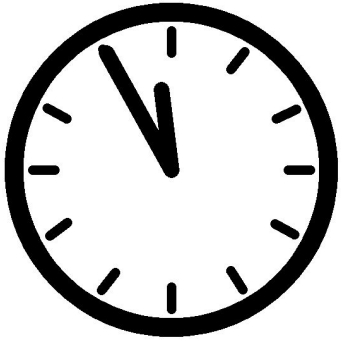
UniCesumar

Projeto, implementação e Teste de Software

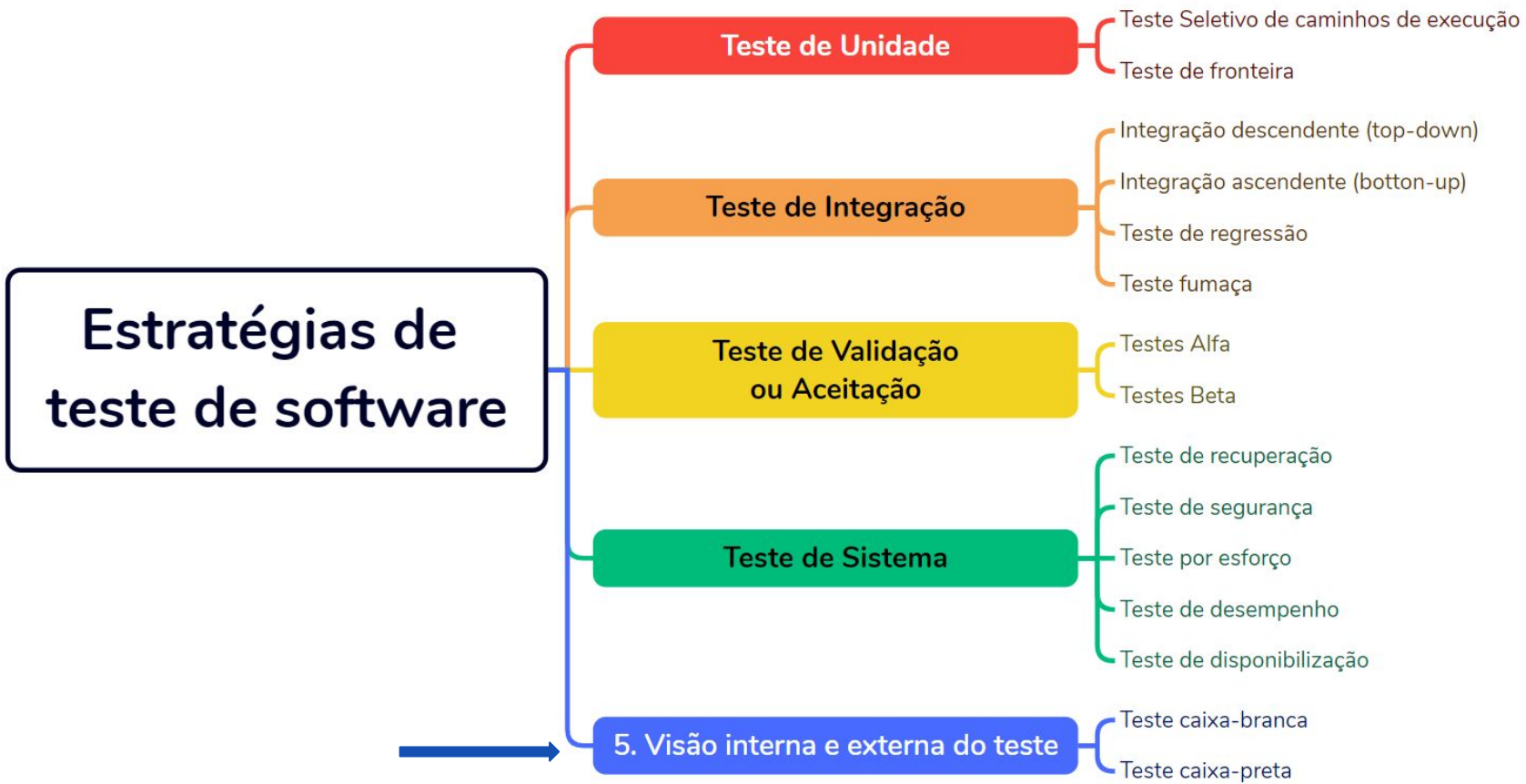
Artefatos de Teste de Software

Prof. Esp. Dacio F. Machado

Anteriormente em Teste de Software



- Visão de Técnicas de Teste de Software
- Testes Funcionais


Estratégias de teste de software
Teste de Unidade
Teste Seletivo de caminhos de execução
Teste de fronteira
Teste de Integração
Integração descendente (top-down)
Integração ascendente (botton-up)
Teste de regressão
Teste fumaça
Teste de Validação ou Aceitação
Testes Alfa
Testes Beta
Teste de Sistema
Teste de recuperação
Teste de segurança
Teste por esforço
Teste de desempenho
Teste de disponibilização
5. Visão interna e externa do teste
Teste caixa-branca
Teste caixa-preta

Níveis de Teste

- **Teste de Unidade / Unitário**
- **Teste de Componentes**
- **Teste de Integração**
- **Teste de Sistema**
- **Teste de Regressão**
- **Teste de Aceitação**

Técnicas de Teste

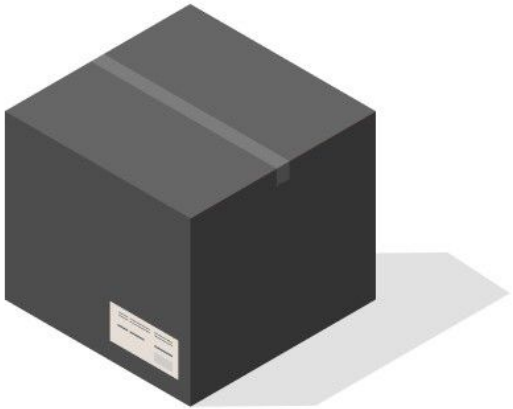
- **Teste Caixa-Branca / *White-Box* / Estrutural**
- **Teste Caixa-Preta / *Black-Box* / Funcional**

Tipos de Teste

- **Teste de Usabilidade**
- **Teste de Desempenho / *Performance***
- **Teste de Carga**
- **Teste de Estresse / Esforço**
- **Teste de Segurança**

Visão de Teste

Segundo Pressman (2011), qualquer produto de engenharia pode ser testado a partir de duas perspectivas diferentes:



Black box - we do not know anything

White box - we know everything

Visão de Teste

(1) a lógica interna do programa é exercitada usando técnicas de projeto de caso de teste “caixa branca”;

(2) os requisitos de software são exercitados usando técnicas de projeto de casos de teste “caixa preta”.

Caixa Branca

A Primeira abordagem requer uma visão interna e é chamada de teste caixa-branca.

• Fundamenta-se em um exame rigoroso do detalhe procedimental.

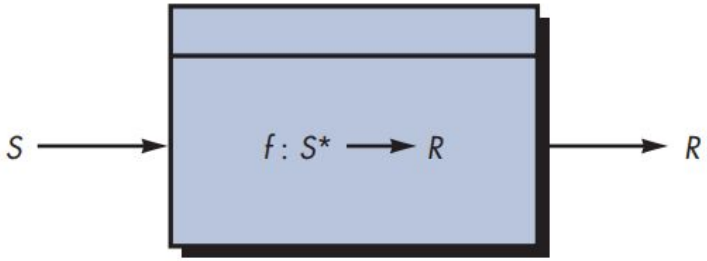
• Os caminhos lógicos do software e as colaborações entre componentes são testados.

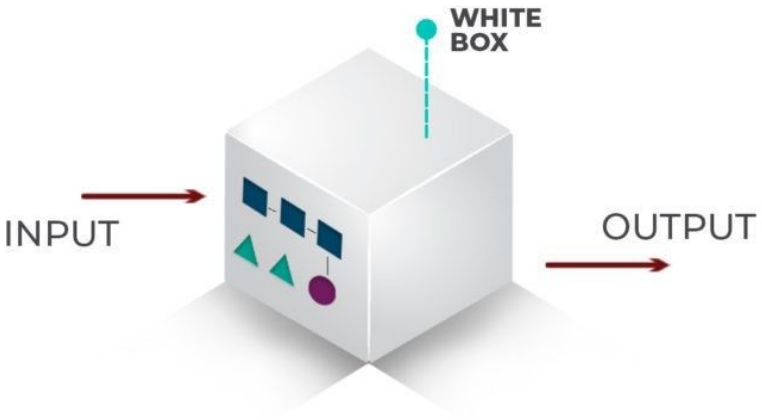
Uma abordagem para testes de programas onde os testes são baseados no conhecimento da estrutura do programa e de seus componentes.

Acesso ao código-fonte é essencial para testes de caixa branca.

FIGURA 21.3

Uma especificação caixa-preta





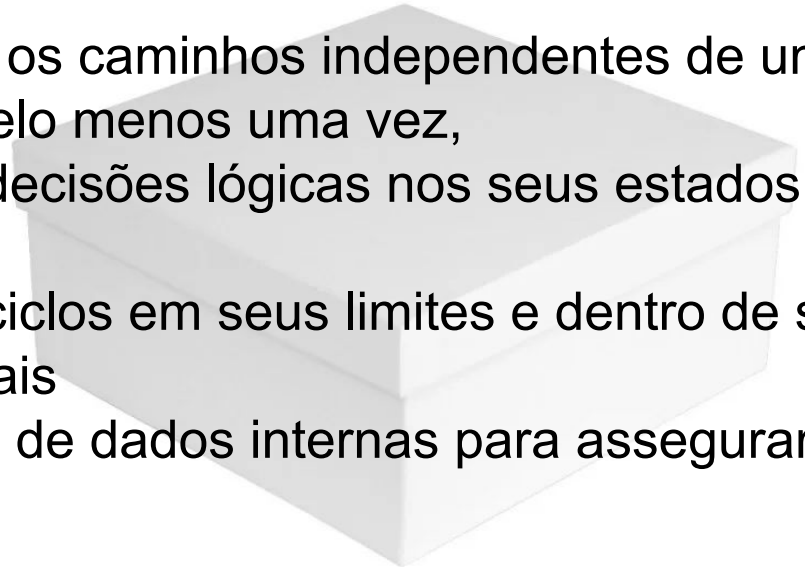
TESTE ESTRUTURAL

Teste CAIXA BRANCA

Teste CAIXA BRANCA

Usando métodos de teste caixa-branca, o engenheiro de software pode criar casos de teste que:

1. garantam que todos os caminhos independentes de um módulo foram exercitados pelo menos uma vez,
2. exercitam todas as decisões lógicas nos seus estados verdadeiro e falso,
3. executam todos os ciclos em seus limites e dentro de suas fronteiras operacionais
4. exercitam estruturas de dados internas para assegurar a sua validade.



Teste Estrutural

Objetivo principal é garantir que o código-fonte seja testado de maneira abrangente, com ênfase na cobertura de todas as partes do código

- Testar:
  - Instruções
  - Caminhos de execução
  - Ramificações condicionais

Teste Estrutural

Objetivo principal é garantir que o código-fonte seja testado de maneira abrangente, com ênfase na cobertura de todas as partes do código

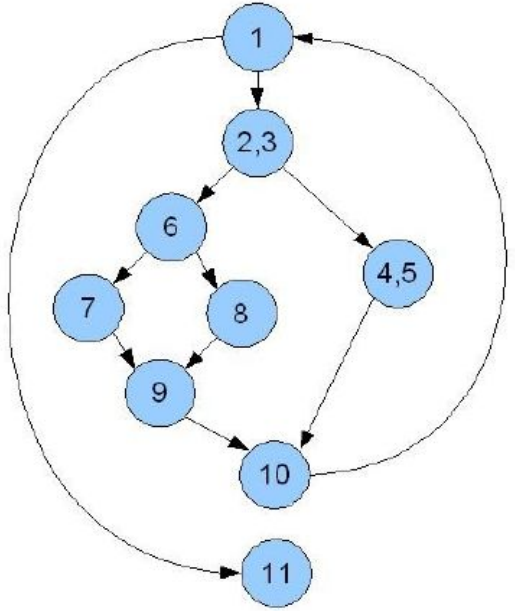
- Testar:
  - Teste de cobertura de código
  - Teste de caminho
  - Teste de ramificação
  - Teste de mutação

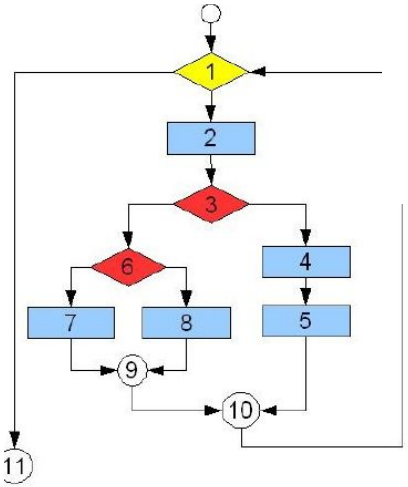
Critério de Teste	Critério	Elementos Requeridos
Propriedades que devem ser avaliadas no teste	1. Baseados em complexidade 2. Baseados em fluxo de controle 3. Baseados em fluxo de dados	Todo critério de teste é composto por um conjunto requisitos de teste • Caminhos, laços de repetição, definição e uso de variáveis

Teste Estrutural - Grafo de Fluxo de Controle

Geralmente o teste estrutural é representado utilizando um Grafo de Fluxo de Controle.

- Um nó corresponde a uma instrução
- As arestas denotam o potencial fluxo de controle entre as instruções

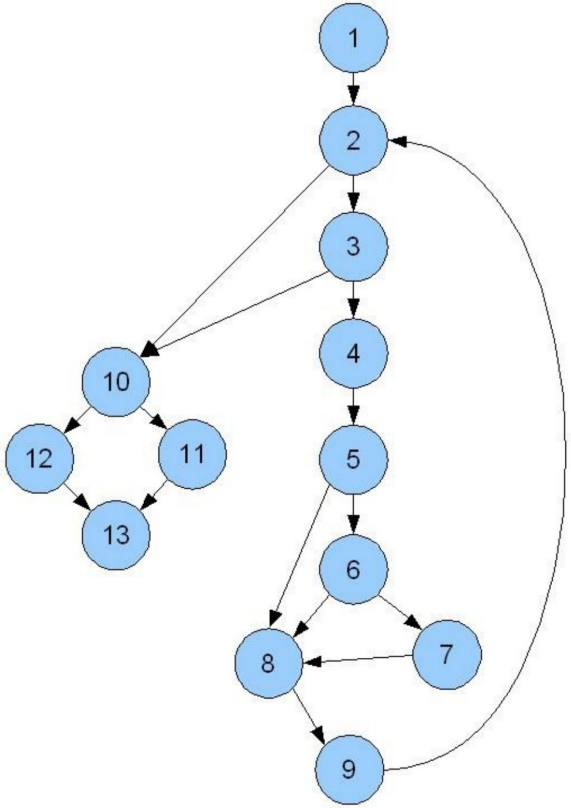




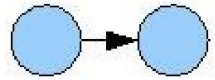
Teste Estrutural - Fluxo de Controle

É a sequência de passos que o computador segue para executar as operações do programa:

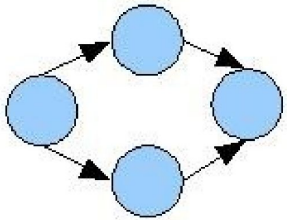
- Sequência
- Condicionais
- Estruturas de repetição



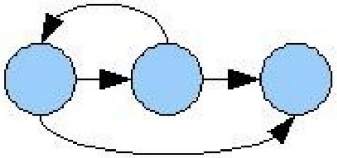
As Construções Estruturadas em forma de grafo de fluxo



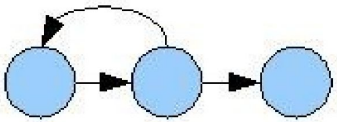
Seqüência



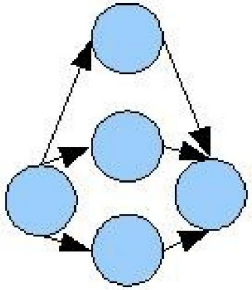
Se-então-senão



Enquanto

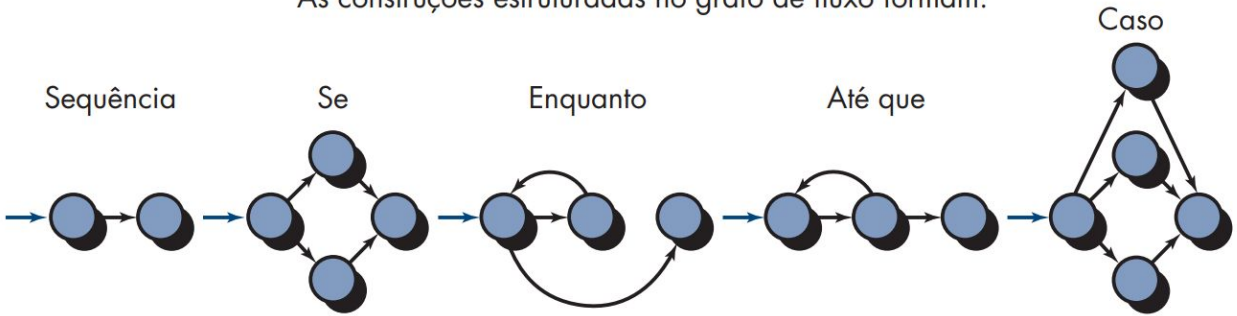


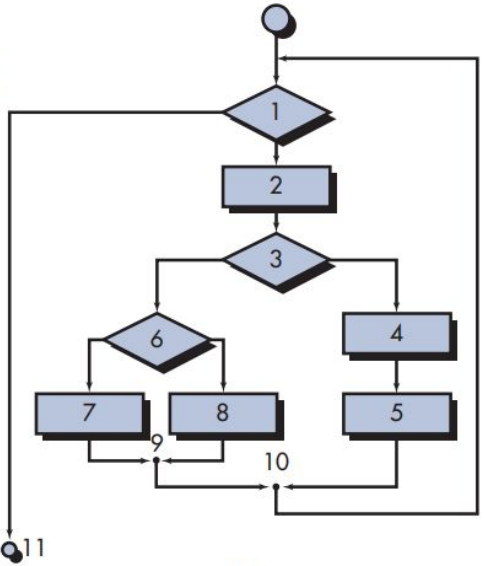
Repita



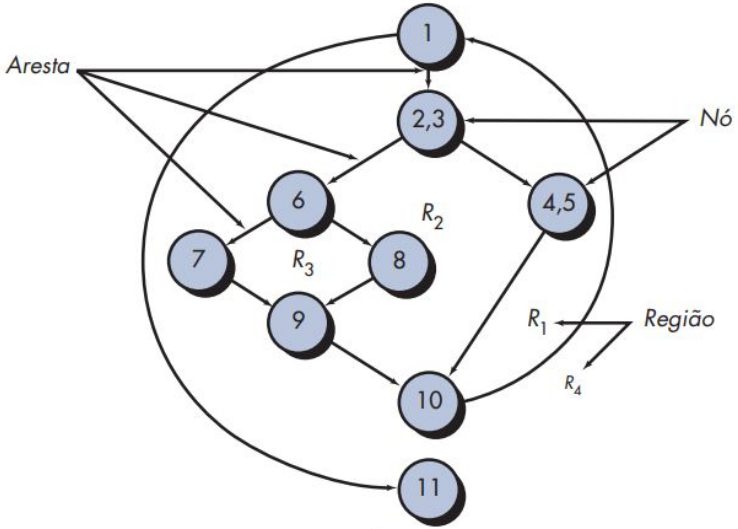
Caso

As construções estruturadas no grafo de fluxo formam:





(a)



(b)

Diretrizes	
Caminho	seqüência de vértices conectados por arestas.
Caminho Simples	Caminho em que um nó não se repete, exceto o primeiro e último.
Caminho Livre de Laço	Caminho em que um nó não se repete.
Caminho Completo	caminho que inicia no nó de entrada e termina em um nó de saída.
Caminho não executável	Se existe algum nó ou vertice não executado de acordo com um dado de entrada.
Caminho Livre de Definição	Não contém redefinição uma varíavel ao longo do caminho
Definição Global	Ou contém um caminho livre de definição para um nó ou existe C-USO ou P-USO da varíavel em um arco.

Teste Estrutural - Fluxo de Controle

- Todos-Nós exige que a execução do programa passe, ao menos uma vez, em cada vértice do grafo de fluxo, ou seja, que cada comando do programa seja executado pelo menos uma vez.
- Todos-Arcos requer que cada aresta do grafo, ou seja, cada desvio de fluxo de controle do programa, seja exercitada pelo menos uma vez.
- Todos-Caminhos requer que todos os caminhos possíveis do programa sejam executados.

Teste Estrutural - Fluxo de Controle

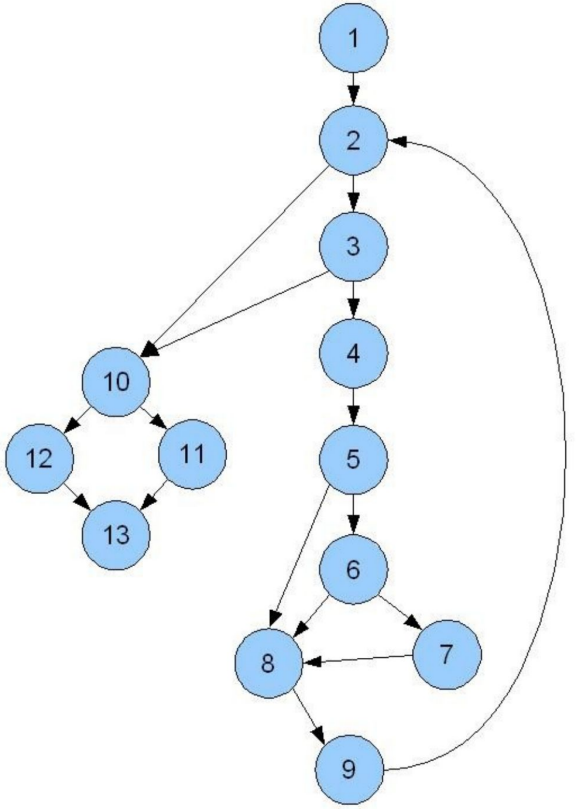
- Todas-Definições: requer que cada definição de variável seja exercitada pelo menos uma vez, não importa se por um c-uso ou por um p-uso.
- Todos-Usos: requer que todas as associações entre uma definição de variável e seus subseqüentes usos (c-usos e p-usos) sejam exercitadas pelos casos de teste, através de pelo menos um caminho livre de definição, ou seja, um caminho onde a variável não é redefinida

Teste Estrutural - Fluxo de Dados

O fluxo de dados descreve como os dados são lidos, processados e transmitidos

Teste de fluxo de dados possui intenção de revelar defeitos em decorrência de valores incorretos na codificação.

Princípio da definição dos critérios: sequência das ações realizadas sobre as variáveis mais onde elas são definidas e utilizadas.

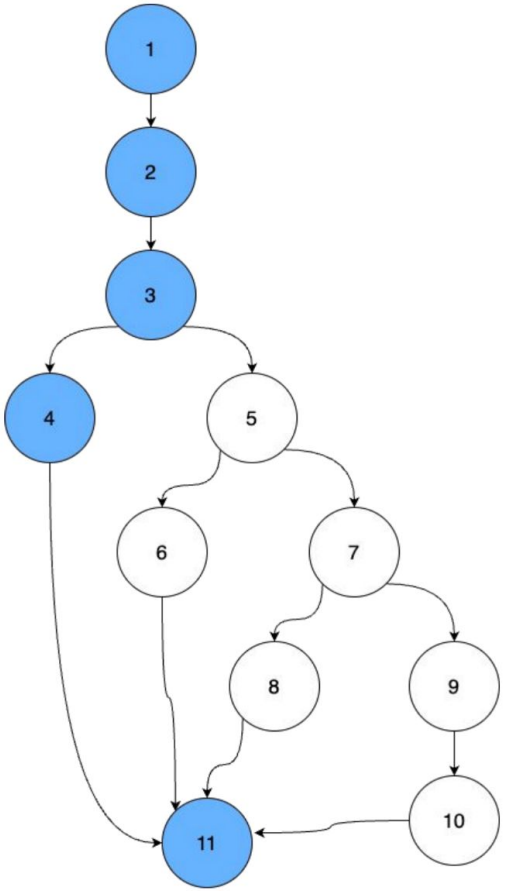


Anomalias do Fluxo de Dados
Uso de variável não inicializada.
Atribuição de valor a uma variável mais de uma vez sem que tenha havido uma referência a essa variável entre essas atribuições.
Liberação ou reinicialização de uma variável antes que ela tenha sido criada ou inicializada.
Liberação ou reinicialização de uma variável antes que ela tenha sido usada.
Atribuir novo valor a um ponteiro sem que a variável tenha sido liberada.

Notação	Significado
~	Não existe variável
d	Definição da variável
u	Uso da variável
K	Destruição da variável
~d	Variável não existe e é definida (correto)
~u	Variável não existe e é usada (incorreto)
~k	Variávelnão existe e é destruída
dd	Definida e redefinida (incorreto se global)
du	Definida e usada (correto)
dk	Definida e destruída (incorreto)
ud	Usada e definida (aceitável)
uu	Usada e reusada (aceitável)

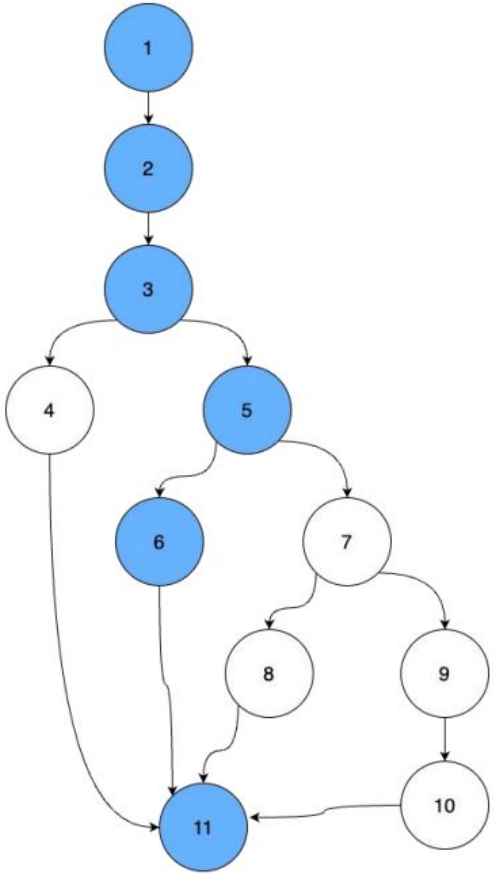
```
1. public class PnE {
2.   public static void calcularValor(int valor) {
3.       if (valor < 0) {
4.               System.out.println("Valor negativo");
5.       } else if (valor > 100) {
6.           System.out.println("Valor maior que 100");
7.       } else if (valor >= 0 && valor <= 100) {
8.           System.out.println("Valor entre 0 e 100");
9.       } else {
10.          System.out.println("Esta linha nunca será executada.");
11.      }
12.  }

13.      public static void main(String[] args) {
14.              calcularValor(-50);
15.      }
16. }
```



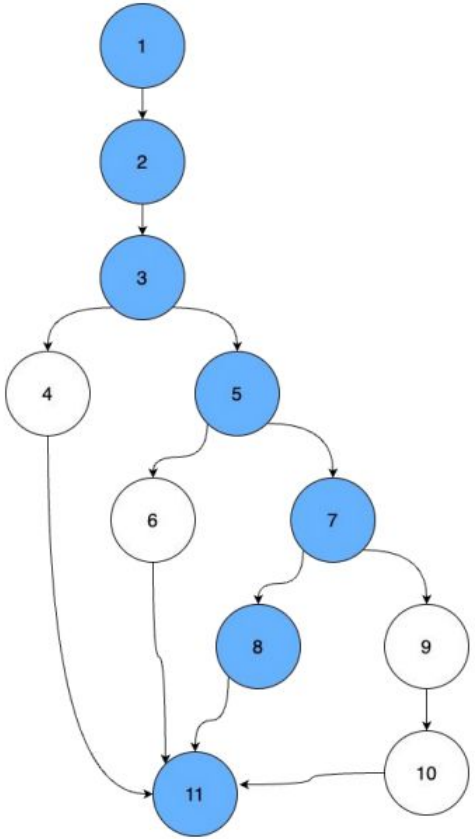
```
1. public class PnE {
2.   public static void calcularValor(int valor) {
3.       if (valor < 0) {
4.               System.out.println("Valor negativo");
5.       } else if (valor > 100) {
6.           System.out.println("Valor maior que 100");
7.       } else if (valor >= 0 && valor <= 100) {
8.           System.out.println("Valor entre 0 e 100");
9.       } else {
10.          System.out.println("Esta linha nunca será executada.");
11.      }
12.  }

13.      public static void main(String[] args) {
14.              calcularValor(101);
15.      }
16. }
```



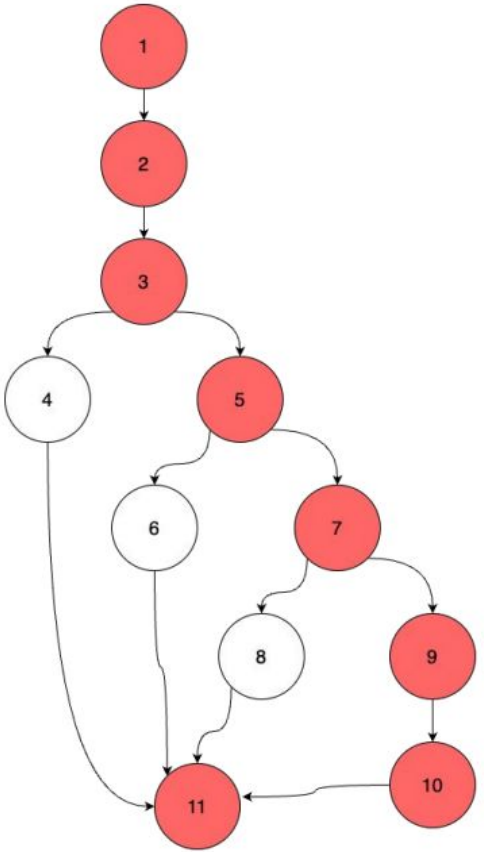
```
1. public class PnE {
2.   public static void calcularValor(int valor) {
3.       if (valor < 0) {
4.               System.out.println("Valor negativo");
5.       } else if (valor > 100) {
6.           System.out.println("Valor maior que 100");
7.       } else if (valor >= 0 && valor <= 100) {
8.           System.out.println("Valor entre 0 e 100");
9.       } else {
10.          System.out.println("Esta linha nunca será executada.");
11.      }
12.  }

13.      public static void main(String[] args) {
14.              calcularValor(75);
15.      }
16. }
```



```
1. public class PnE {
2.   public static void calcularValor(int valor) {
3.       if (valor < 0) {
4.               System.out.println("Valor negativo");
5.       } else if (valor > 100) {
6.           System.out.println("Valor maior que 100");
7.       } else if (valor >= 0 && valor <= 100) {
8.           System.out.println("Valor entre 0 e 100");
9.       } else {
10.          System.out.println("Esta linha nunca será executada.");
11.      }
12.  }

13.      public static void main(String[] args) {
14.              calcularValor(??);
15.      }
16. }
```



Teste de Caminho Básico

O teste de caminho básico é uma técnica de teste caixa-branca.

- Permite derivar uma medida da complexidade lógica de um projeto e usar essa medida como guia para definir um conjunto base de caminhos de execução.
- Casos de teste criados para exercitar o conjunto básico executam com certeza todas as instruções de um programa pelo menos uma vez durante o teste (Pressman, 2011).

Teste de Caminho Básico

A ideia por trás do caminho básico é identificar e testar caminhos que percorrem diferentes partes do código, como instruções, decisões condicionais e loops.

Os caminhos básicos ajudam a garantir que todas as partes do código sejam testadas pelo menos uma vez.

Teste de Caminho Básico

- Identificação de caminhos:
  - identificar todos os caminhos possíveis de execução no código-fonte.
  - Isso inclui caminhos que passam por instruções simples, estruturas de controle de fluxo, como condicionais (if/else) e loops (for/while), e qualquer outra estrutura de decisão.

Teste de Caminho Básico

- Simplificação:
  - eliminar caminhos redundantes ou irrelevantes.
- Desenvolvimento de casos de teste:
  - Com os caminhos básicos identificados, são criados casos de teste que sigam esses caminhos.
  - Cada caso de teste visa testar um caminho específico, fornecendo entradas e condições de teste apropriadas.
- Execução de testes:
  - os casos de teste são executados no programa, e os resultados são avaliados

Teste de Caminho Básico

O cálculo da complexidade ciclomática fornece a resposta. Para calcular a complexidade ciclomática de McCabe, você pode usar a fórmula a seguir:

$$V(G) = E - N + 2$$

Onde:
V(G) é a complexidade ciclomática.
E é o número de arestas no grafo de fluxo de controle.
N é o número de nós no grafo de fluxo de controle.

OBRIGADO

dacio.francisco@unicesumar.edu.br