

# **Engenharia de Software II / Qualidade e Teste de Software**

## **Aula 08: Testes Estruturais (Caixa-Branca)**

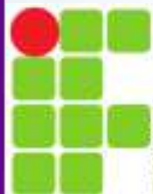
**Breno Lisi Romano**

**<http://sites.google.com/site/blromano>**

**Instituto Federal de São Paulo – IFSP São João da Boa Vista**

**Bacharelado em Ciência da Computação – BCC (ENSC6)**

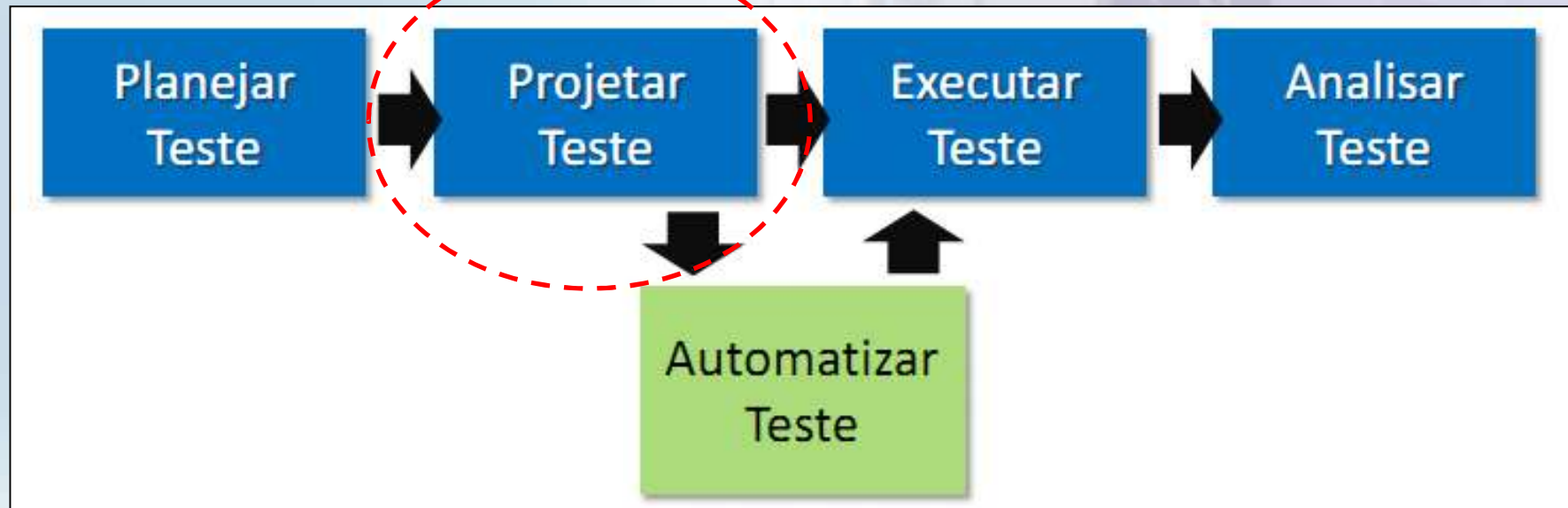
**Tecnologia em Sistemas para Internet – TSI (QTSI6)**



**INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Campus São João da Boa Vista**



# Revisão: Processo Básico de Teste de Software





# Revisão: V&V

- **Verificação:**

- Processo de avaliação de um sistema ou componente para determinar se os **artefatos** produzidos **satisfazem** às **especificações** determinadas no início da fase
- “**Você construiu corretamente?**”

- **Validação:**

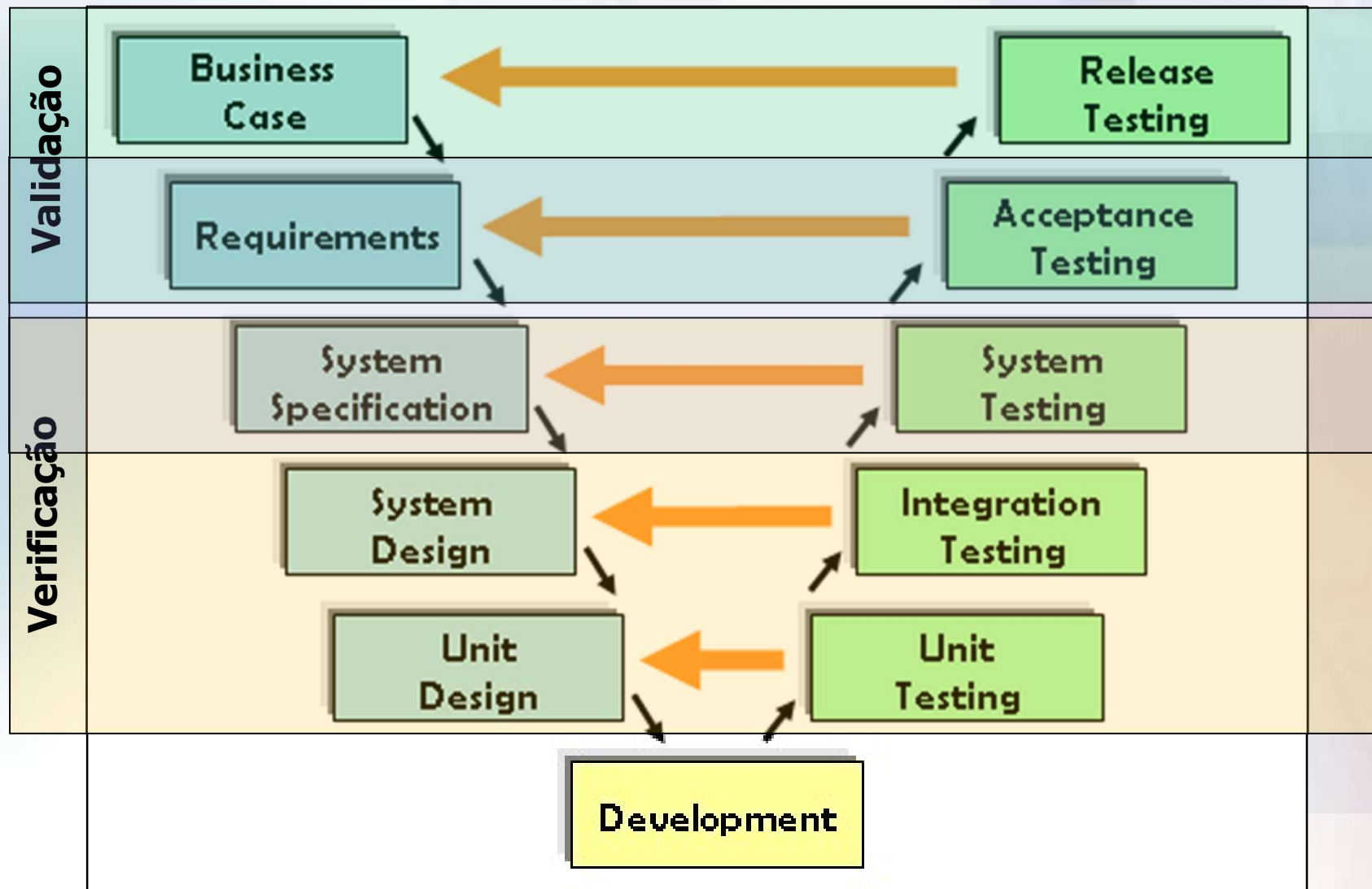
- Processo de avaliação para determinar se o **sistema atende** as **necessidades** e **requisitos** dos usuários
- “**Você construiu o sistema correto?**”

- **Testes:**

- Processo de exercitar um sistema ou componente para **validar** que este **satisfaz** os **requisitos** e para **verificar** para identificar defeitos

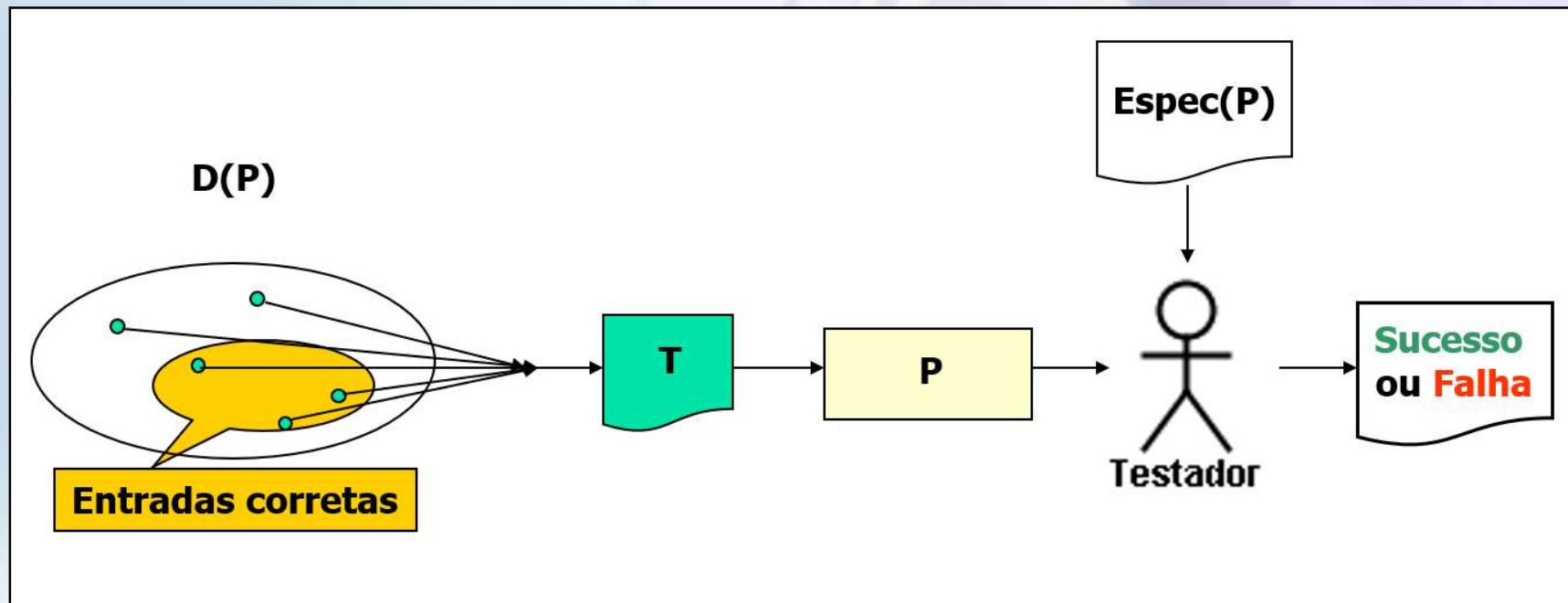


# Revisão: Modelo V





# Revisão: Cenário Típico da Atividades de Teste





# Revisão: Teste de Subdomínios

- A identificação dos subdomínios é feita com base em **critérios de teste**
- Dependendo dos critérios estabelecidos, são obtidos subdomínios diferentes.
- Tipos principais de **critérios de teste**:
  - Funcionais \*
  - Estruturais \*
  - Baseados em Modelos
  - Baseados em Defeitos \*
- O que diferencia cada um deles é o tipo de informação utilizada para estabelecer os subdomínios (Delamaro et al., 2007) → Técnicas de Teste.



# Revisão: Funcional – Particionamento de Equivalência

- Passos:
  1. **Identificar as Classes de Equivalência**, tomando por base:
    - a) Especificação
    - b) As entradas possíveis
    - c) As saídas possíveis
    - d) Identificar as Classes de Equivalência Válidas e Inválidas
  2. **Gerar Casos de Teste selecionando um elemento de cada classe e uma saída possível**, de modo a ter o **menor número possível de casos de teste** (Delamaro et al., 2007)
  3. Casos de Testes adicionais podem ser criados, para que exista um sentimento de conforto com os resultados dos testes
  4. Dificilmente defeitos adicionais serão descobertos com estes casos de teste





# Revisão: Funcional – Análise do Valor Limite

- Critério usado **em conjunto com o particionamento de equivalência**
- **Premissa: Casos de Teste que exploram condições limites têm maior probabilidade de encontrar defeitos**
- Tais **condições** estão **exatamente sobre ou imediatamente acima ou abaixo dos limitantes** das Classes de Equivalência (Delamaro et al., 2007)





# Revisão: Funcional – Sistemático

- **Resumindo:**

1. Testar ao menos dois elementos de uma mesma Classe de Equivalência
2. Testar tipos ilegais de valores para o problema em questão (Ex.: integer, float, double, long, ...)
3. Testar entradas válidas que podem ser consideradas ilegais (Caracteres Especiais)
4. Se preocupar com as saídas dos testes

# Testes Estruturais



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Campus São João da Boa Vista

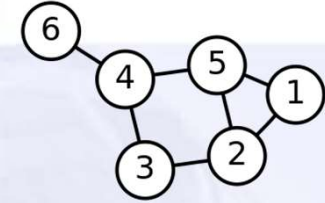


# Teste Estrutural (1)

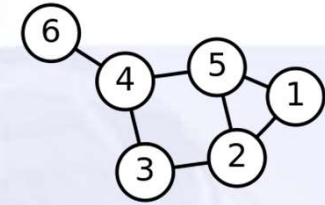
- Baseia-se no conhecimento da estrutura interna do software
- Caminhos lógicos são testados, estabelecendo Casos de Teste que põem à prova:
  - Condições, Laços, Definições e Usos de variáveis
- Principais critérios (Delamaro et al., 2007):
  - Critérios baseados na complexidade
  - Critérios baseados no fluxo de controle
  - Critérios baseados no fluxo de dados



## Teste Estrutural (2)



- A maioria dos critérios estruturais utiliza uma representação de **Grafo de Fluxo de Controle (GFC)** ou Grafo de Programa, no qual blocos disjuntos de comandos são considerados nós
- A execução do primeiro comando do bloco acarreta a execução de todos os outros comandos desse bloco, na ordem dada
- Todos os comandos em um bloco têm um único predecessor (possivelmente com exceção do primeiro) e um único sucessor (possivelmente com exceção do último)



## Teste Estrutural (3)

- A representação de um software como um GFC estabelece uma correspondência entre nós e blocos de comandos e indica possíveis fluxos de controle entre blocos por meio de arcos:
  - Um GFC é um grafo orientado com um único nó de entrada e um único nó de saída
- **Importantíssimo:** Cada nó representa um bloco indivisível (sequencial) de comandos e cada arco representa um possível desvio de um bloco para outro
- A partir do GFC, podem ser escolhidos os elementos que devem ser executados

# Grafos



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Campus São João da Boa Vista



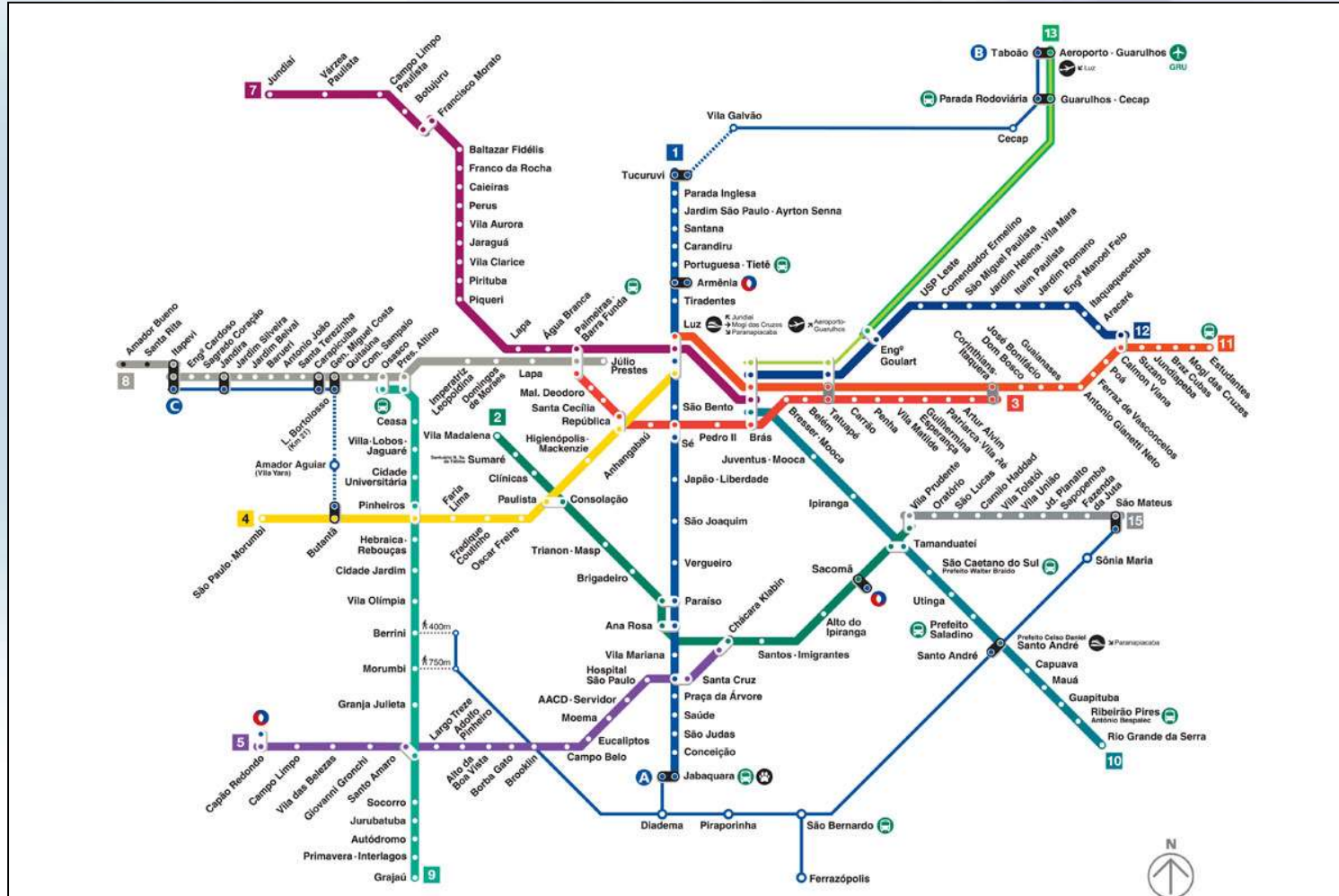
# Exemplos de Grafos – Mapa Rodoviário do Estado de SP







# Exemplos de Grafos – Estações de Metrô da cidade de São Paulo





# Exemplos de Grafos – Mapa Hidrográfico do Rio Amazonas



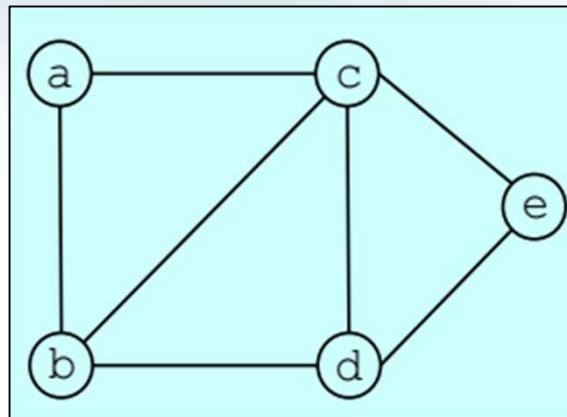


# Grafos: Definição

- **Definição Formal:**

- **Grafo é um par  $G = (V, A)$ , em que:**

- **$V$  é um conjunto finito com  $n$  vértices ou nós  $\rightarrow V = \{a, b, c, d, e\}$**
- **$A$  conjunto finito de pares não-ordenados de vértices chamados de arestas ou arcos  $\rightarrow A = \{(a, b), (a, c), (b, c), (b, d), (c, d), (c, e), (d, e)\}$**

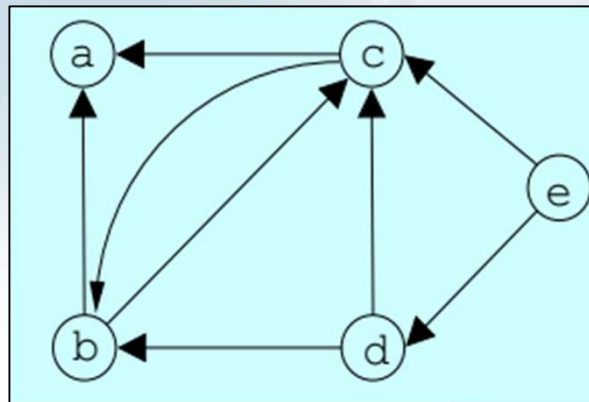




# Grafos Orientados

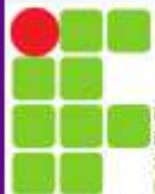
- Um **grafo orientado (direcionado)** é definido de forma semelhante, com a diferença que as **arestas consistem de pares ordenados de vértices**

- **Exemplo:**



- As vezes, para enfatizar, dizemos grafo não-orientado em vez de simplesmente grafo

# Grafos de Fluxo de Controle (GFC)



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Campus São João da Boa Vista




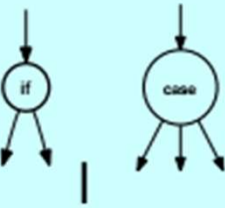



# Grafo de Fluxo de Controle – GFC (1)

- **Grafo Orientado:** Notação gráfica utilizada para abstrair o Fluxo de Controle Lógico de um software
- Composto de **nós** e **arcos**:
  - Um **nó** representa uma ou mais instruções as quais são sempre executadas em sequência, ou seja, uma vez executada a primeira instrução de um nó, todas as demais instruções daquele nó também são executadas
  - Um **arco**, também chamado de ramo ou aresta, representa o fluxo de controle entre blocos de comandos (nós)



## Grafo de Fluxo de Controle – GFC (2)

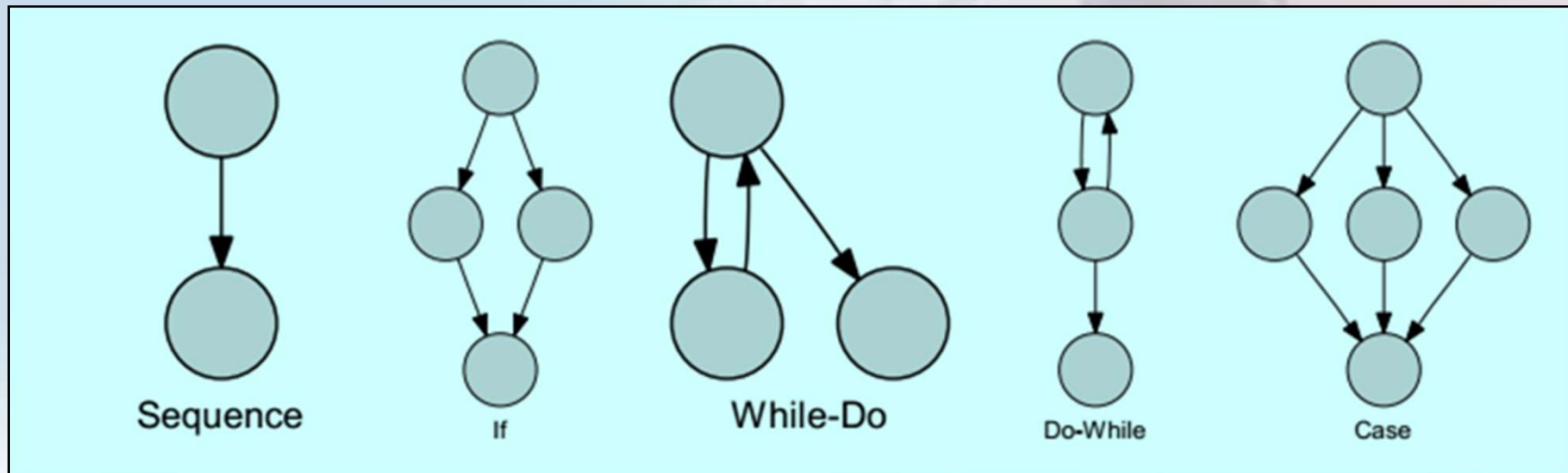
	grupo de comandos executados seqüencialmente do início ao fim.
	ponto de mudança de fluxo de controle.
	ponto no qual fluxos de controle são unidos.





## Grafo de Fluxo de Controle – GFC (3)

- Para representar uma função / método com um GFC, as seguintes construções podem ser utilizadas:





# Exemplo: Fibonacci (1)

```
public class Fibonacci {  
  
    public long fibonacci (int n) {  
  
1        int i = 1;  
2        int a = 0;  
3        int b = 1;  
  
4        if (n >= 1) {  
5            while (i < n) {  
6                long temp = b;  
7                b = b + a;  
8                a = temp;  
9                i = i + 1;  
10           }  
11        }  
12        return (b+a);  
    }  
}
```

## Exemplos de Aplicação:

F(0)	F(1)	F(2)	F(3)	F(4)	F(5)	F(6)	.....
1	1	2	3	5	8	13	.....

## Aplicando para n=5:

i	a	b	Temp
1	0	1	-



## Exemplo: Fibonacci (2)

```
public class Fibonacci {  
  
    public long fibonacci (int n) {  
  
1        int i = 1;  
2        int a = 0;  
3        int b = 1;  
  
4        if (n >= 1) {  
5            while (i < n) {  
6                long temp = b;  
7                b = b + a;  
8                a = temp;  
9                i = i + 1;  
10           }  
11        }  
12        return (b+a);  
    }  
}
```

### Exemplos de Aplicação:

F(0)	F(1)	F(2)	F(3)	F(4)	F(5)	F(6)	.....
1	1	2	3	5	8	13	.....

### Aplicando para n=5:

i	a	b	Temp
1	0	1	-
2	1	1	1
3	1	2	1
4	2	3	2
5	3	5	3



## Exemplo: Fibonacci (3)

```
public class Fibonacci {  
    public long fibonacci (int n) {
```

```
1      int i = 1;  
2      int a = 0;  
3      int b = 1;
```

**Nó 01**

```
4      if (n >= 1) {
```

```
5          while (i < n) {
```

**Nó 02**

```
6              long temp = b;  
7              b = b + a;  
8              a = temp;  
9              i = i + 1;  
10         }
```

**Nó 03**

```
11     }  
12     return (b+a);
```

**Nó 04**

```
    }  
}
```

**Encontrar os nós do GFC para o Algoritmo de Fibonacci:** Cada nó representa um bloco indivisível (sequencial) de comandos e cada arco representa um possível desvio de um bloco para outro

Nó	Bloco de Comandos
1	1 a 4
2	5
3	6 a 10
4	11 e 12



## Exemplo: Fibonacci (4)

```
public class Fibonacci {  
    public long fibonacci (int n) {
```

```
1      int i = 1;  
2      int a = 0;  
3      int b = 1;
```

**Nó 01**

```
4      if (n >= 1) {
```

```
5          while (i < n) {
```

**Nó 02**

```
6              long temp = b;  
7              b = b + a;  
8              a = temp;  
9              i = i + 1;  
10         }
```

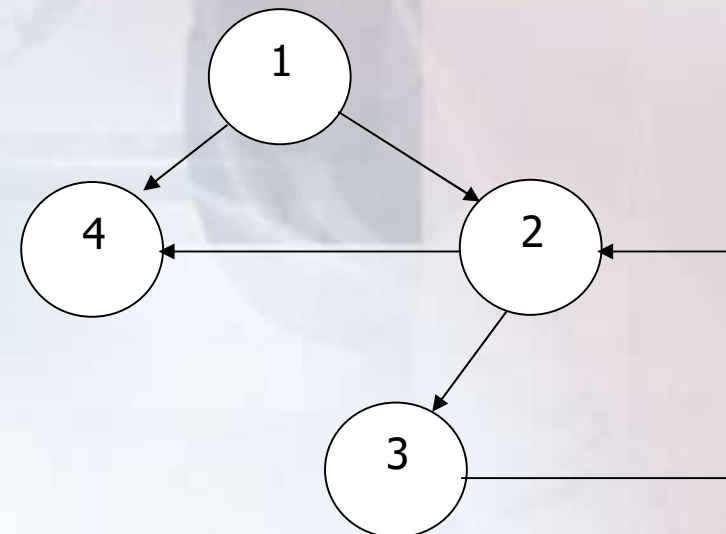
**Nó 03**

```
11     }  
12     return (b+a);
```

**Nó 04**

```
    }  
}
```

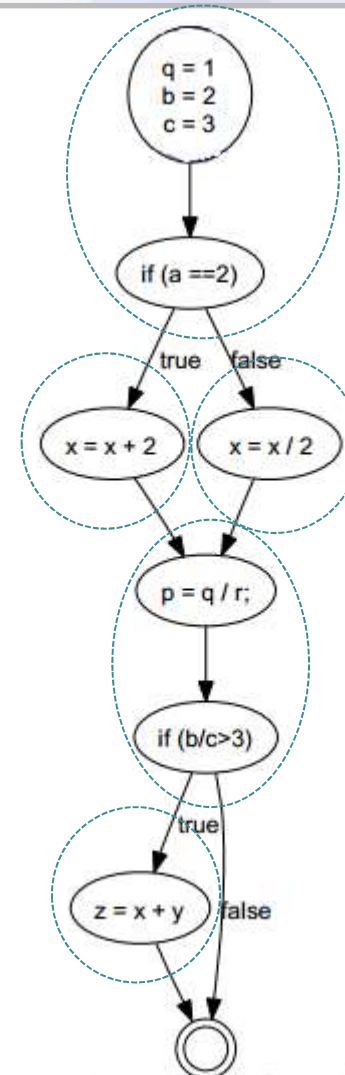
### Grafo de Fluxo de Controle (GFC)





# GFC – Programa de Exemplo do Coopeland (2004)

```
1  q = 1;  
2  b = 2;  
3  c = 3;  
4  if (a == 2) {  
5      x = x + 2;  
6  } else {  
7      x = x / 2;  
8  }  
9  p = q / r;  
10 if (b/c > 3) {  
11     z = x + y;  
12 }
```





# GFC: Para Praticar em Casa

```
void insercao(int a[], int size) {  
    int i, j, aux;  
    for (i = 1; i < size; i++) {  
        aux = a[i];  
        j = i - 1;  
        while (j >= 0 && a[j] >= aux) {  
            a[j + 1] = a[j];  
            j--;  
        }  
        a[j + 1] = aux;  
    }  
}
```



# Teste Estrutural: Critério Baseado na Complexidade – $V(g)$



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Campus São João da Boa Vista



# Critério Baseado na Complexidade: Teste do Caminho Básico (1)

- Utilizam informações sobre a Complexidade do Software para derivar casos de teste
- O Critério de McCabe ou **Teste de Caminho Básico** é o critério baseado em complexidade mais conhecido
- Utiliza a medida de Complexidade Ciclomática para derivar requisitos de teste
- **Complexidade ciclomática -  $V(g)$ :**
  - **Número de caminhos linearmente independentes** do conjunto básico de um software, indicando um **limite máximo para o número de casos de teste** que devem ser executados para garantir que **todas as instruções do programa sejam exercitadas pelo menos uma vez** (Delamaro et al., 2007)



## **Critério Baseado na Complexidade: Teste do Caminho Básico (2)**

- **Caminho linearmente independente:** qualquer caminho que introduza pelo menos um novo **conjunto de instruções** ou **uma nova condição**, ou seja, que introduza pelo menos um arco que não tenha sido atravessado
- Caminhos linearmente independentes estabelecem um conjunto básico para o GFC
- Se os Casos de Teste puderem ser projetados de modo a forçar a execução desses caminhos (conjunto básico):
  - cada instrução terá sido executada pelo menos uma vez
  - cada condição terá sido executada com V e F



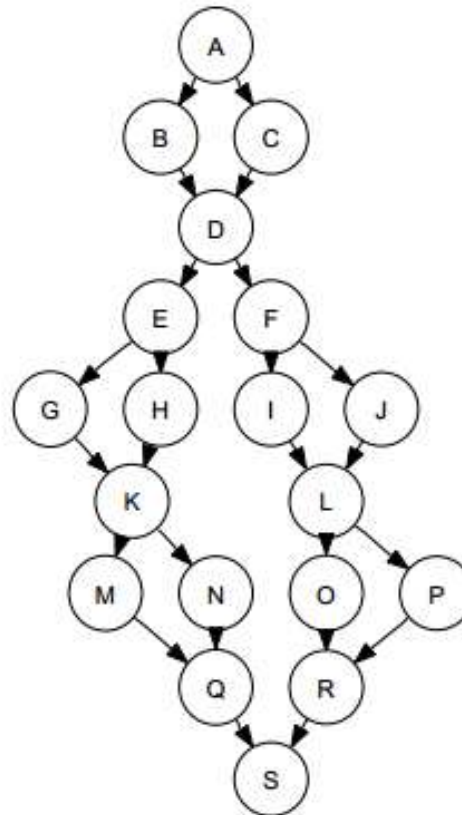
# Critério Baseado na Complexidade: Teste do Caminho Básico (3)

- **O conjunto básico não é único**
  - De fato, diferentes conjuntos básicos podem ser derivados para um GFC → Veremos em exemplos
- O tamanho do **Conjunto de Casos de Teste** (Número de Caminhos) é dado pela Complexidade Ciclomática, que pode ser calculada de várias formas, dentre elas (Delamaro et al., 2007):
  - $V(g) = n^{\circ} \text{ Arcos} - n^{\circ} \text{ Nos} + 2$



# Critério Baseado na Complexidade: Teste do Caminho Básico (4)

- Quantos Casos de Teste seriam necessários para o seguinte GFC gerado a partir de um programa específico?



McCabe (1976) define a Complexidade Ciclomática ( $C$ ) como:

$$C = \text{arcos} - \text{nós} + 2$$

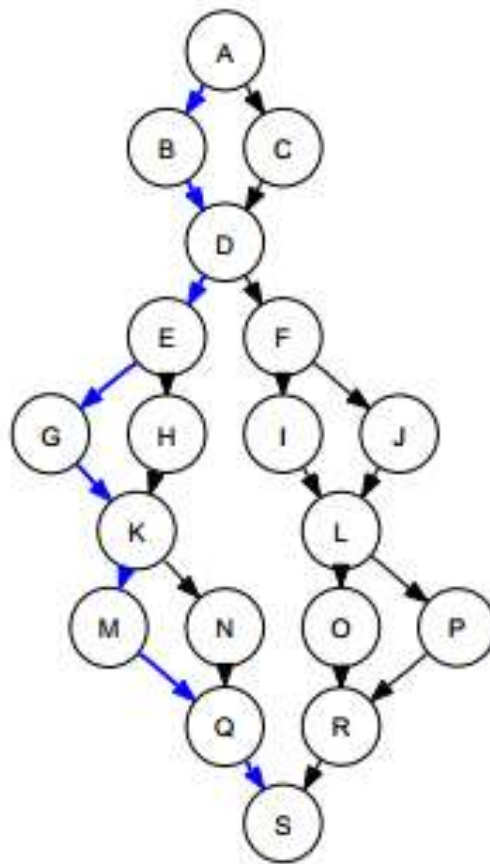
$$C = 24 - 19 + 2$$

$$C = 7$$



# Critério Baseado na Complexidade: Teste do Caminho Básico (4)

- Quais são os Caminhos Básicos para este GFC –  $V(g) = 7$ ?
  - Criação do Conjunto de Caminhos Básicos – Passo 1



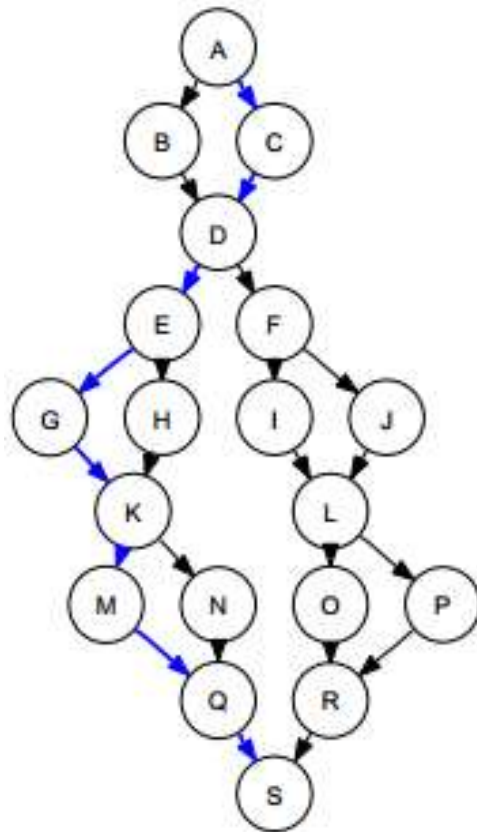
- ▶ Escolha um caminho básico. Esse caminho pode ser:
  - ▶ Caminho mais comum.
  - ▶ Caminho mais crítico.
  - ▶ Caminho mais importante do ponto de vista de teste.
- ▶ Caminho 1: ABDEGKMQS





# Critério Baseado na Complexidade: Teste do Caminho Básico (5)

- Quais são os Caminhos Básicos para este GFC –  $V(g) = 7$ ?
  - Criação do Conjunto de Caminhos Básicos – Passo 2



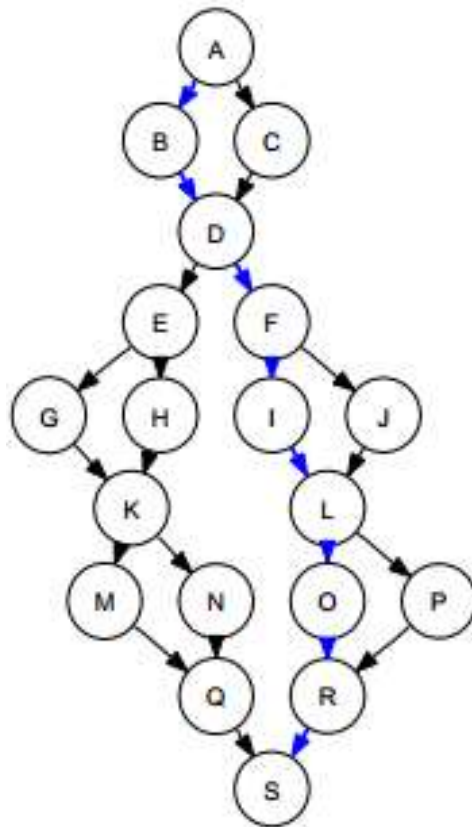
- ▶ Altere a saída do primeiro comando de decisão e mantenha o máximo possível do caminho inalterado.
- ▶ Caminho 2: ACDEGKM QS





# Critério Baseado na Complexidade: Teste do Caminho Básico (6)

- Quais são os Caminhos Básicos para este GFC –  $V(g) = 7$ ?
  - Criação do Conjunto de Caminhos Básicos – Passo 3

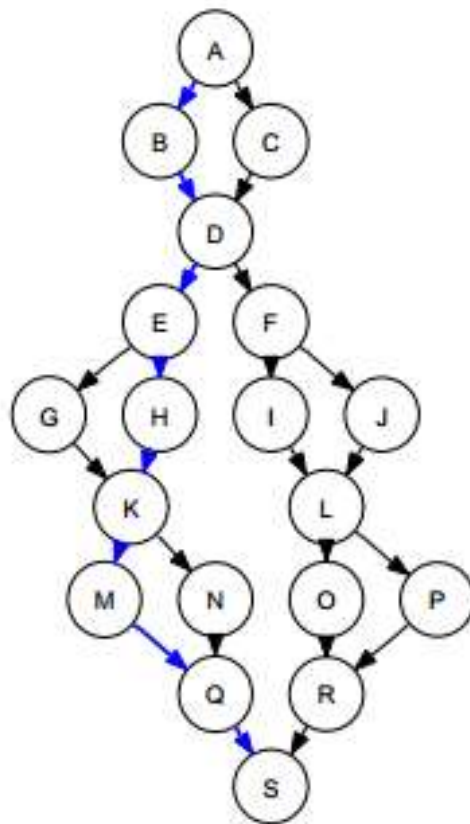


- ▶ A partir do caminho base alterar a saída do segundo comando de decisão.
- ▶ Caminho 3: ABDFILORS



# Critério Baseado na Complexidade: Teste do Caminho Básico (7)

- Quais são os Caminhos Básicos para este GFC –  $V(g) = 7$ ?
  - Criação do Conjunto de Caminhos Básicos – Passo 4

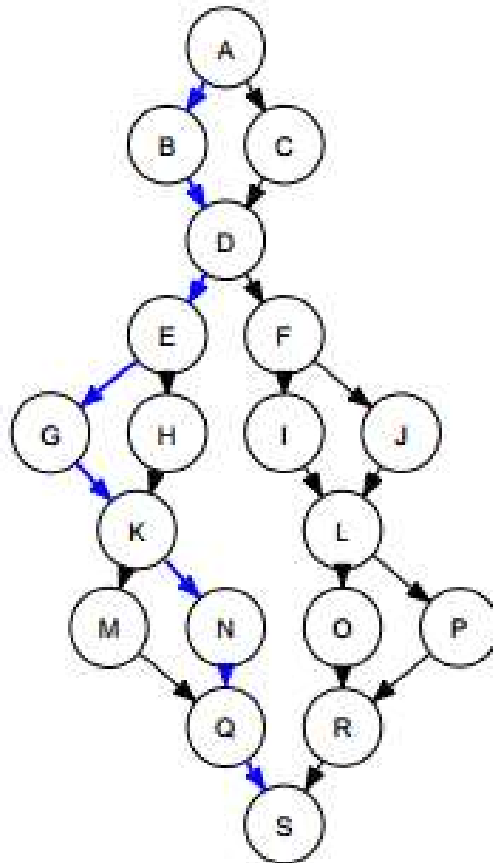


- ▶ A partir do caminho base alterar a saída do terceiro comando de decisão. Repetir esse processo até atingir o final do GFC.
- ▶ Caminho 4: ABDEHKMQS



# Critério Baseado na Complexidade: Teste do Caminho Básico (8)

- Quais são os Caminhos Básicos para este GFC –  $V(g) = 7$ ?
  - Criação do Conjunto de Caminhos Básicos – Passo 5

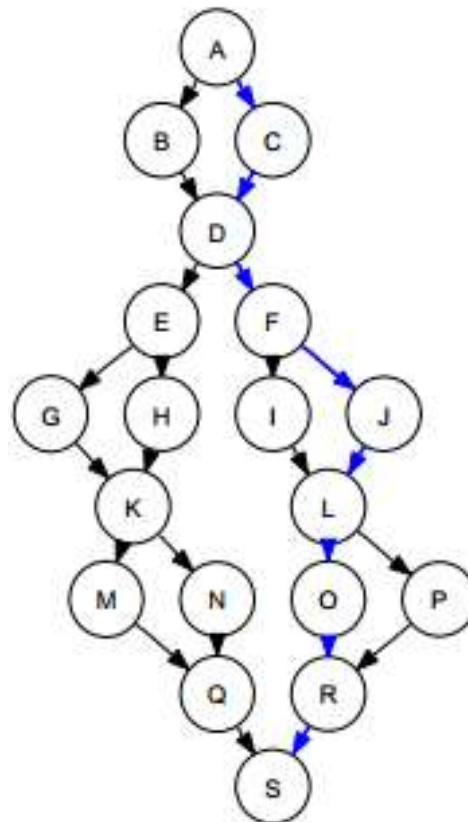


- ▶ Continuação do passo anterior.
- ▶ Caminho 5: ABDEGKNQS



# Critério Baseado na Complexidade: Teste do Caminho Básico (9)

- Quais são os Caminhos Básicos para este GFC –  $V(g) = 7$ ?
  - Criação do Conjunto de Caminhos Básicos – Passo 6

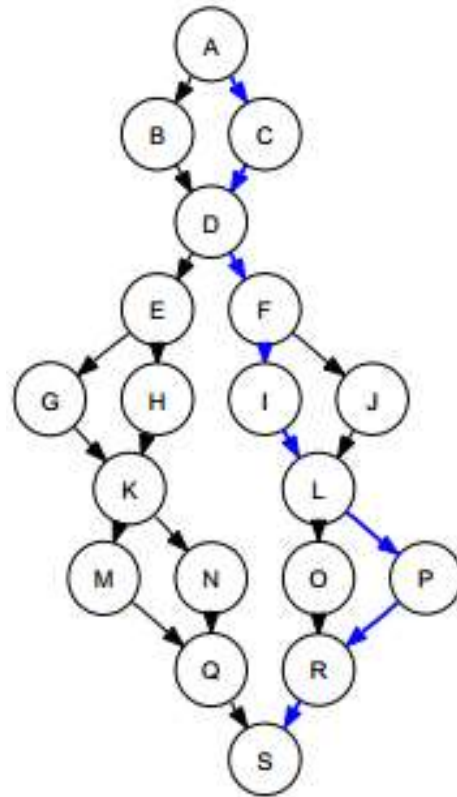


- ▶ Todas as decisões do caminho básico foram contempladas.
- ▶ A partir do segundo caminho, fazer as inversões dos comandos de decisão até o final do GFC.
- ▶ Esse padrão é seguido até que o conjunto completo de caminhos seja atingido.
- ▶ Caminho 6: ACDFJLORS



# Critério Baseado na Complexidade: Teste do Caminho Básico (10)

- Quais são os Caminhos Básicos para este GFC –  $V(g) = 7$ ?
  - Criação do Conjunto de Caminhos Básicos – Passo 7

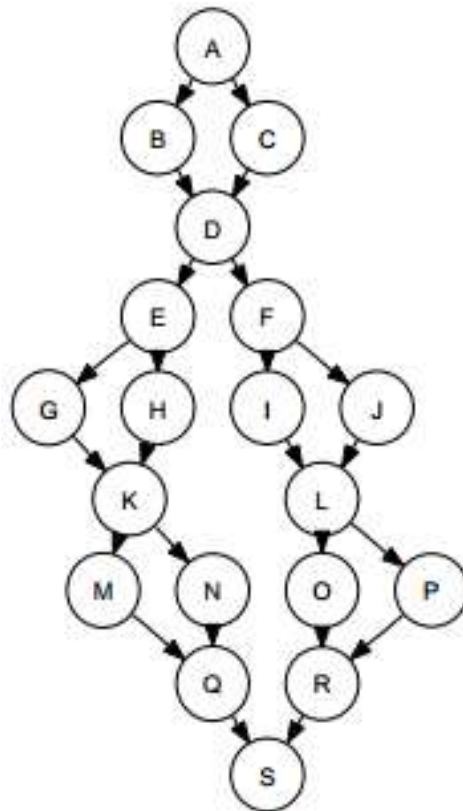


- ▶ Continuação do passo anterior.
- ▶ Caminho 7: ACDFILPRS



# Critério Baseado na Complexidade: Teste do Caminho Básico (11)

- Quais são os Caminhos Básicos para este GFC –  $V(g) = 7$ ?
  - Conjunto Completo de Caminhos Básicos



► Requisitos de testes derivado pelo critério.

- ABDEGKM QS
- ACDEGKM QS
- ABDFILORS
- ABDEHKMQS
- ABDEGKNQS
- ACDFJLORS
- ACDFILPRS

► Conjunto criado não é único.

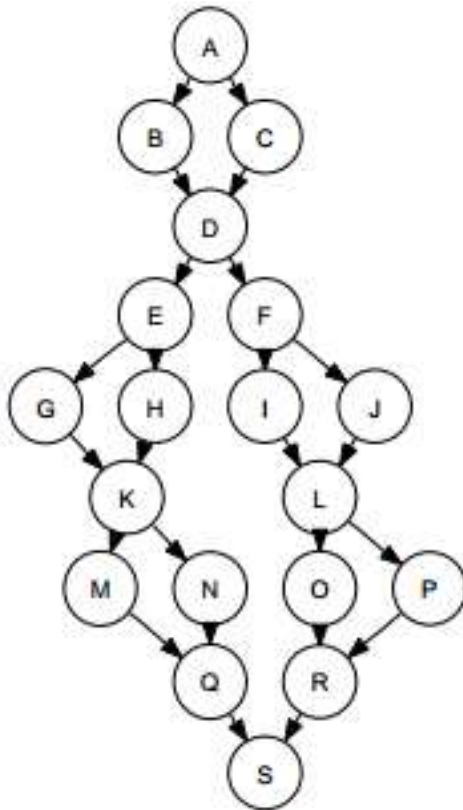
► Propriedade: o conjunto de teste que exercita os caminhos básicos também exercita todos-nós e todos-arcos do programa.





# Critério Baseado na Complexidade: Teste do Caminho Básico (12)

- Quais são os Caminhos Básicos para este GFC –  $V(g) = 7$ ?
  - Quais são os Casos de Teste com base nos Caminhos Básicos?
  - Vamos supor que para a esquerda é Verdadeiro e para direita é Falso



Teste de Caminho Básico

$V(g) = 24 - 19 + 2 = 7$

Vamos supor que  
p/ esquerda é V  
e p/ direita é F

	A	D	E	K	F	L	
#1	V	V	V	V	-	-	ABDEGKM QS
#2	F	V	V	V	-	-	ACDEGKM QS
#3	V	F	-	-	V	V	ABDFILORS
#4	V	V	F	V	-	-	ABDEHKM QS
#5	V	V	V	F	-	-	ABDEGKN QS
#6	V	F	-	-	F	V	ACDFJLORS
#7	V	F	-	-	V	F	ACDFILPRS

- ▶ ABDEGKM QS
- ▶ ACDEGKM QS
- ▶ ABDFILORS
- ▶ ABDEHKM QS
- ▶ ABDEGKN QS
- ▶ ACDFJLORS
- ▶ ACDFILPRS





# Exemplo: Fibonacci (1)

## V(g), Caminhos Básicos e Casos de Teste?

```
public class Fibonacci {  
    public long fibonacci (int n) {
```

```
1      int i = 1;  
2      int a = 0;  
3      int b = 1;
```

**Nó 01**

```
4      if (n >= 1) {
```

```
5          while (i < n) {
```

**Nó 02**

```
6              long temp = b;  
7              b = b + a;  
8              a = temp;  
9              i = i + 1;  
10         }
```

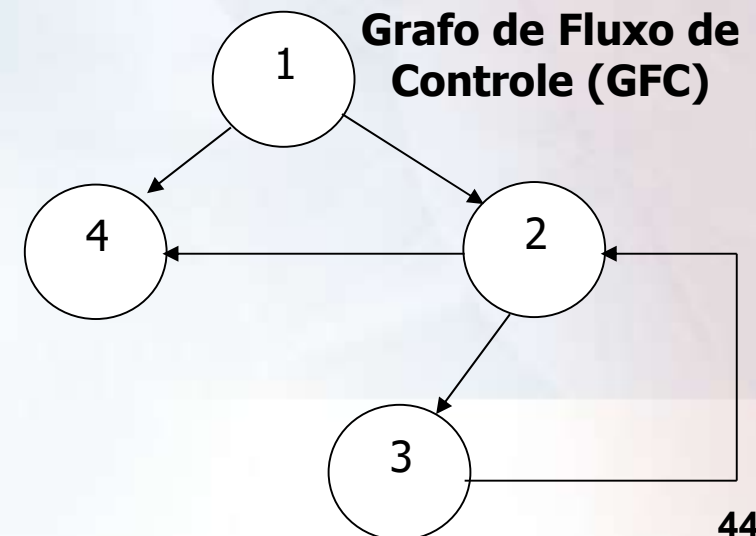
**Nó 03**

```
11     }  
12     return (b+a);
```

**Nó 04**

```
    }  
}
```

Nó	Bloco de Comandos
1	1 a 4
2	5
3	6 a 10
4	11 e 12





## Exemplo: Fibonacci (2)

### $V(g)$ , Caminhos Básicos e Casos de Teste?

- **$V(g) = ?$  / Casos de Teste: ? / Algoritmo está correto?**
  - $V(g) = 5 - 4 + 2 = 3$
  - Caminhos:
    - 1-4
    - 1-2-4
    - 1-2-3-2-4
  - Casos de Teste:
    - $n = 0$
    - $n = 1$
    - $n = 2$
  - Algoritmo está correto?
    - Sim



# Programa de Exemplo do Coopeland (1)

## V(g), Caminhos Básicos e Casos de Teste?

```
1  q = 1;  
2  b = 2;  
3  c = 3;  
4  if (a == 2) {  
5      x = x + 2;  
6  } else {  
7      x = x / 2;  
8  }  
9  p = q / r;  
10 if (b/c > 3) {  
11     z = x + y;  
12 }
```

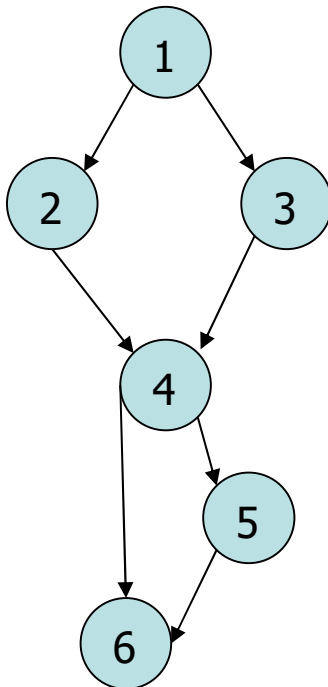
- GFC?
- $V(g) = ?$
- Caminhos Linearmente Independentes?
- Casos de Teste: ?
- Algoritmo está correto?



# Programa de Exemplo do Coopeland (2)

## V(g), Caminhos Básicos e Casos de Teste?

```
1 q = 1;  
2 b = 2;  
3 c = 3;  
4 if (a == 2) {  
5   x = x + 2;  
6 } else {  
7   x = x / 2;  
8 }  
9 p = q / r;  
10 if (b/c > 3) {  
11   z = x + y;  
12 }
```



Nó 01: 1-4

Nó 02: 5

Nó 03: 6-7

Nó 04: 8-10

Nó 05: 11

Nó 06: 12

$$V(g) = A - N + 2 = 7 - 6 = 3$$

Caminhos Linearmente Independentes:

C#01: 1,2,4,6

C#02: 1,3,4,6

C#03: 1,2,4,5,6

Casos de Teste:

CT#01: a = 2

CT#02: a != 2

CT#03: ??? Não tem como criar, pois b/c é sempre < 3, uma vez que b=2 e c=3. Nunca a condição b/c > 3 é verdadeira.

Não está Correto..

Nunca o algoritmo entra no Nó 05.



# GFC: Para Praticar em Casa

## Fibonacci Recursivo

```
public class Fibonacci {  
  
    public long fibonacci (int n)  
    {  
1        if (n == 1)  
2            return 1;  
3        else  
4            return fibonacci(n-1) + fibonacci(n-2);  
    }  
  
}
```

- GFC: ?
- $V(g)$ : ?
- Caminhos Linearmente Independentes: ?
- Casos de Teste: ?

# Teste Estrutural: Critério Baseado no Fluxo de Controle



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Campus São João da Boa Vista



# Critério Baseado no Fluxo de Controle

- Utilizam apenas características de controle da execução do software (comandos, desvios, laços) para derivar Casos de Teste
- Principais **Critérios**:
  - **Todos-Nós**: requer que cada nó do grafo (e por conseguinte cada comando do algoritmo) seja executado pelo menos uma vez (mínimo esperado de uma boa atividade de teste)
  - **Todos-Arcos**: requer que cada arco do grafo (cada desvio de fluxo de controle) seja exercitado pelo menos uma vez
  - **Todos-Caminhos**: requer que todos os caminhos possíveis no programa sejam exercitados (pode ser impraticável). Igual ao Critério Baseado na Complexidade





# Exemplo: Fibonacci (1)

## Critério Baseado no Fluxo de Controle

```
public class Fibonacci {  
    public long fibonacci (int n) {
```

```
1      int i = 1;  
2      int a = 0;  
3      int b = 1;
```

**Nó 01**

```
4      if (n >= 1) {
```

```
5          while (i < n) {
```

**Nó 02**

```
6              long temp = b;  
7              b = b + a;  
8              a = temp;  
9              i = i + 1;  
10         }
```

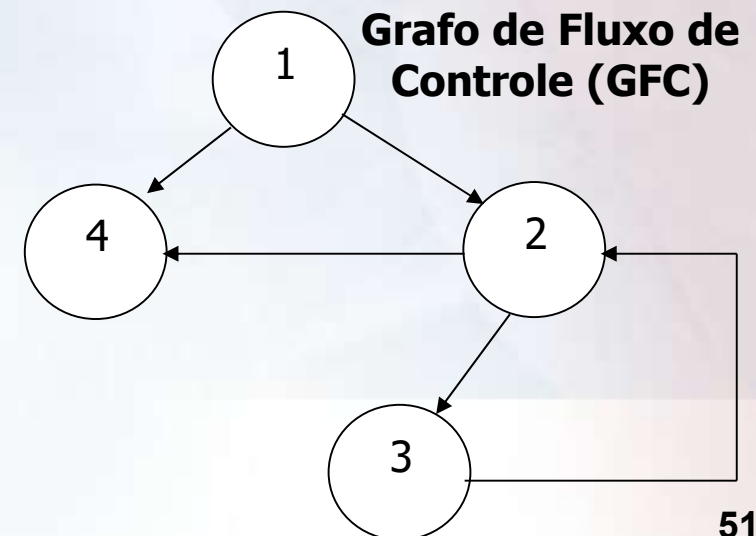
**Nó 03**

```
11     }  
12     return (b+a);
```

**Nó 04**

```
    }  
}
```

Nó	Bloco de Comandos
1	1 a 4
2	5
3	6 a 10
4	11 e 12

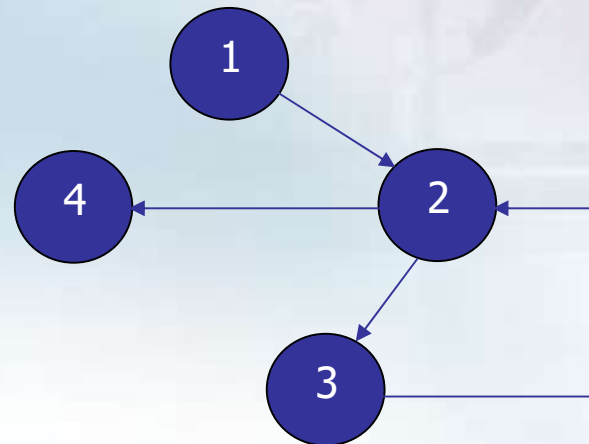




# Exemplo: Fibonacci (2)

## Critério Baseado no Fluxo de Controle

- **Todos-Nós:** requer que cada nó do grafo (e por conseguinte cada comando do algoritmo) seja executado pelo menos uma vez (mínimo esperado de uma boa atividade de teste)



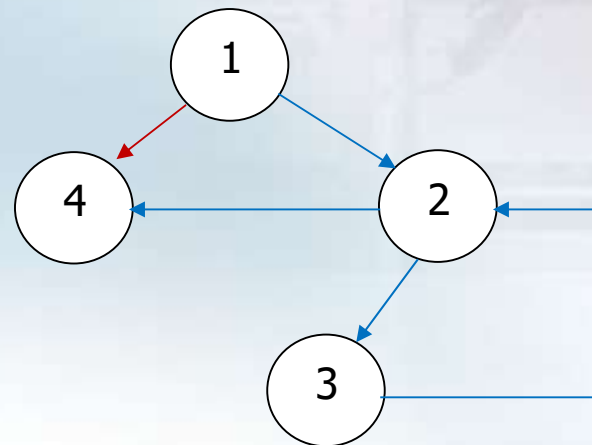
- Casos de Teste: ?
  - 1 CT:  $n = 2$



# Exemplo: Fibonacci (3)

## Critério Baseado no Fluxo de Controle

- **Todos-Arcos:** requer que cada arco do grafo (cada desvio de fluxo de controle) seja exercitado pelo menos uma vez



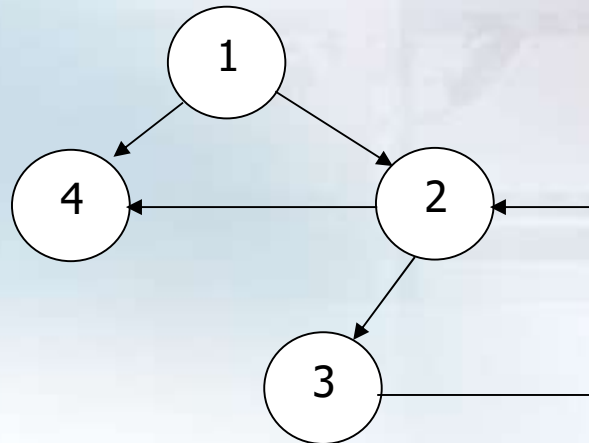
- Casos de Teste: ?
  - 2CT:  $n = 0$ ;  $n = 2$



# Exemplo: Fibonacci (4)

## Critério Baseado no Fluxo de Controle

- **Todos-Caminhos:** requer que todos os caminhos possíveis no programa sejam exercitados (pode ser impraticável). Igual ao Critério Baseado na Complexidade



- Casos de Teste: ?
  - Caminhos: 1-4, 1-2-4 e 1-2-3-2-4
    - Casos de Teste:
      - $n = 0$
      - $n = 1$
      - $n = 2$

# Teste Estrutural: Critério Baseado no Fluxo de Dados



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Campus São João da Boa Vista



# Critério Baseado no Fluxo de Dados (1)

- Baseiam-se nas associações entre a definição de uma variável e seus possíveis usos subsequentes para derivar Casos de Teste
- Visam testar as interações que envolvem definições de variáveis e subsequentes referências a essas definições
- **Premissa:** Comandos que têm uma relação de fluxo de dados são provavelmente parte de uma mesma função e, portanto, devem ser exercitados juntos



## Critério Baseado no Fluxo de Dados (2)

- **Principais Critérios:**
  - **Todas-Definições:** requer que cada definição de variável seja exercitada pelo menos uma vez
  - **Todos-Usos:** requer que todas as associações entre uma definição de variável e seus subsequentes usos sejam exercitadas pelo menos uma vez, em um caminho em que não há redefinição da variável (caminho livre de definição)





# Exemplo: Fibonacci (1)

## Critério Baseado no Fluxo de Dados

```
public class Fibonacci {  
    public long fibonacci (int n) {
```

```
1      int i = 1;  
2      int a = 0;  
3      int b = 1;
```

**Nó 01**

```
4      if (n >= 1) {
```

```
5          while (i < n) {
```

**Nó 02**

```
6              long temp = b;  
7              b = b + a;  
8              a = temp;  
9              i = i + 1;  
10         }
```

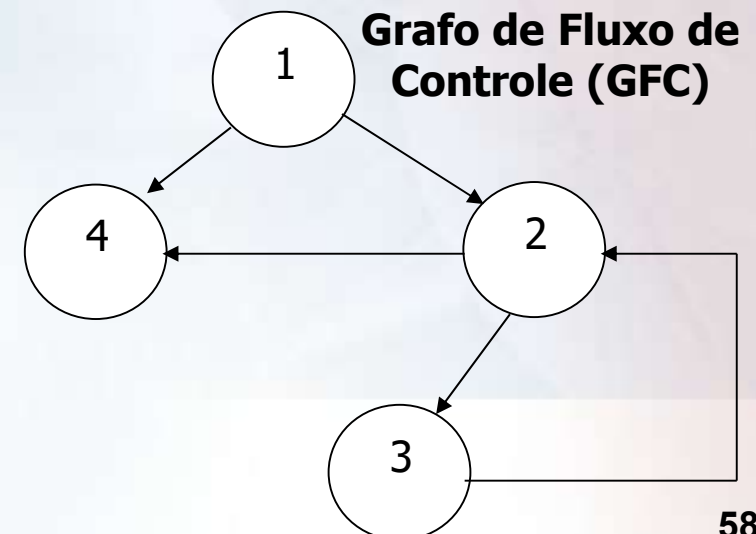
**Nó 03**

```
11     }  
12     return (b+a);
```

**Nó 04**

```
    }  
}
```

Nó	Bloco de Comandos
1	1 a 4
2	5
3	6 a 10
4	11 e 12





# Exemplo: Fibonacci (2)

## Critério Baseado no Fluxo de Dados

- **Todas-Definições**

- Blocos a Exercitar ?
- Casos de Teste ?

Blocos 1 e 3  
 $n=2$

- **Todos-Usos**

- Blocos a Exercitar ?
- Casos de Teste ?

Blocos 2, 3 e 4  
 $n=2$



# Desvantagens do Teste de Caixa Branca

- O número de caminhos a serem executados pode ser infinito (semelhante ao teste exaustivo)
- O Caso de Teste selecionado pode não revelar o defeito sensível a dado. Por exemplo:  $y = x * 2$ 
  - Deveria ser  $y = x ^ 2$
  - Funciona corretamente somente para  $x = 0, y=0$  e  $x = 2, y = 4$ .
- Determinação de caminhos não executáveis pode ser um problema: Dificuldade de automatização
- Habilidades de programação avançadas exigidas para compreender o código e decidir pela executabilidade ou não de um caminho



# Vantagens do Teste de Caixa Branca

- É possível garantir que partes essenciais ou críticas do programa tenham sido executadas
- **Requisito mínimo de teste:** garantir que o algoritmo foi liberado tendo seus comandos executados ao menos uma vez por pelo menos um caso de teste



# Teste Estrutural: Considerações Finais

- Foco Principal: teste de unidade
- Alguns elementos requeridos podem não ser executáveis
- Se um algoritmo não implementa uma função, não existirá um caminho que corresponda a ela e nenhum dado de teste será requerido para exercitá-la
- Essa técnica é vista como complementar às demais, uma vez que cobre classes distintas de defeitos



# Aplicando Técnicas de Teste de Software

- Uma **boa abordagem** para a introdução de práticas mais sistemáticas de teste em uma organização consiste em aplicar inicialmente critérios mais fracos, e talvez menos eficazes, porém menos custosos
- Em função da disponibilidade de orçamento e de tempo, da criticidade de um software e da maturidade da organização, aplicar critérios mais fortes, e eventualmente mais eficazes, porém mais caros



# Abordagem Incremental de Aplicação de Técnicas de Teste de Software

1. Elaborar um conjunto de testes funcionais ( $T_f$ ), utilizando um critério funcional ou uma combinação deles.
2. Avaliar a cobertura de  $T_f$  em relação aos critérios de teste estruturais.
3. Evoluir  $T_f$  até obter um conjunto de teste  $T_n$  adequado ao critério Todos-Nós.
4. Avaliar a cobertura de  $T_n$  em relação aos demais critérios de teste estruturais.
5. Evoluir  $T_n$  até obter um conjunto de teste  $T_a$  adequado ao critério Todos-Arcos.
6. Avaliar a cobertura de  $T_a$  em relação aos demais critérios de teste estruturais.
7. Evoluir  $T_a$  até obter um conjunto de teste  $T_u$  adequado ao critério Todos-Usos.



# Engenharia de Software II / Qualidade e Teste de Software

## Aula 08: Testes Estruturais (Caixa-Branca)

Breno Lisi Romano

**Dúvidas?**

<http://sites.google.com/site/blromano>

Instituto Federal de São Paulo – IFSP São João da Boa Vista

Bacharelado em Ciência da Computação – BCC (ENSC6)

Tecnologia em Sistemas para Internet – TSI (QTSL6)



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Campus São João da Boa Vista