



Universidade Estadual de Santa Cruz – UESC

TESTE DE SOFTWARE



Sérgio Fred

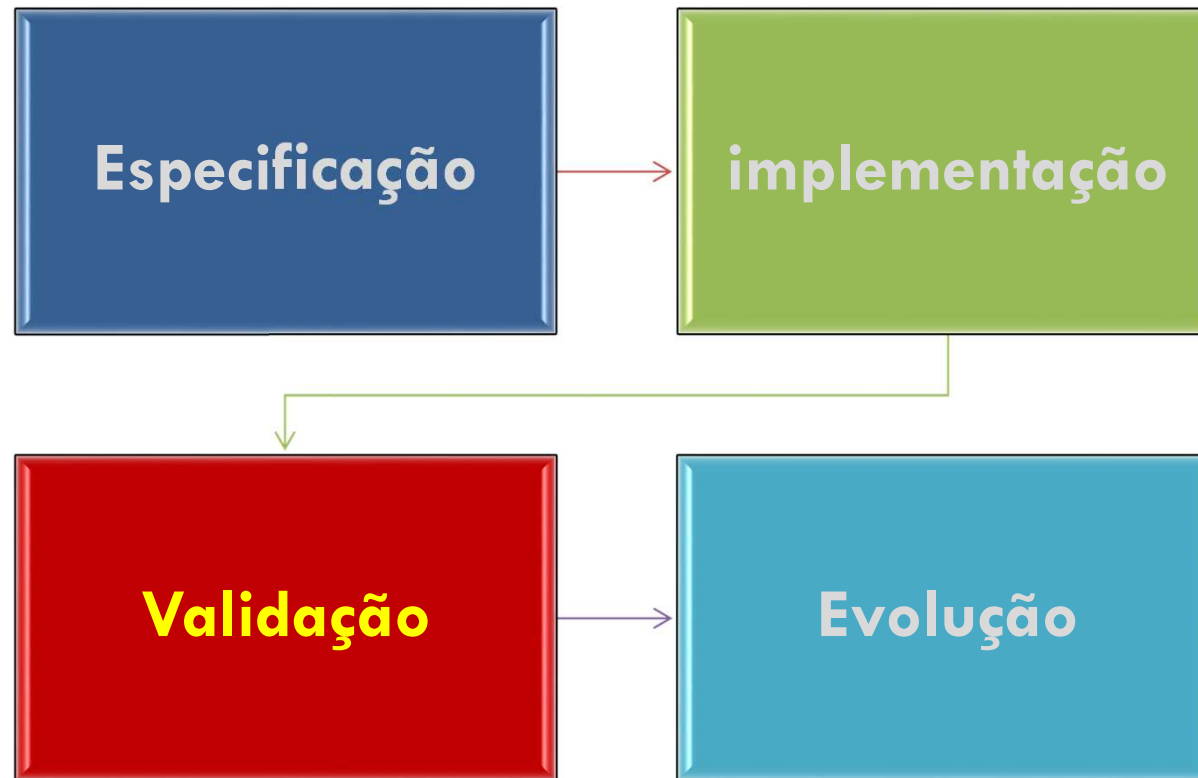
Roteiro

2

- Introdução
- Teste de componentes
- Teste de sistema
- Projeto de casos de teste
- Automação de testes

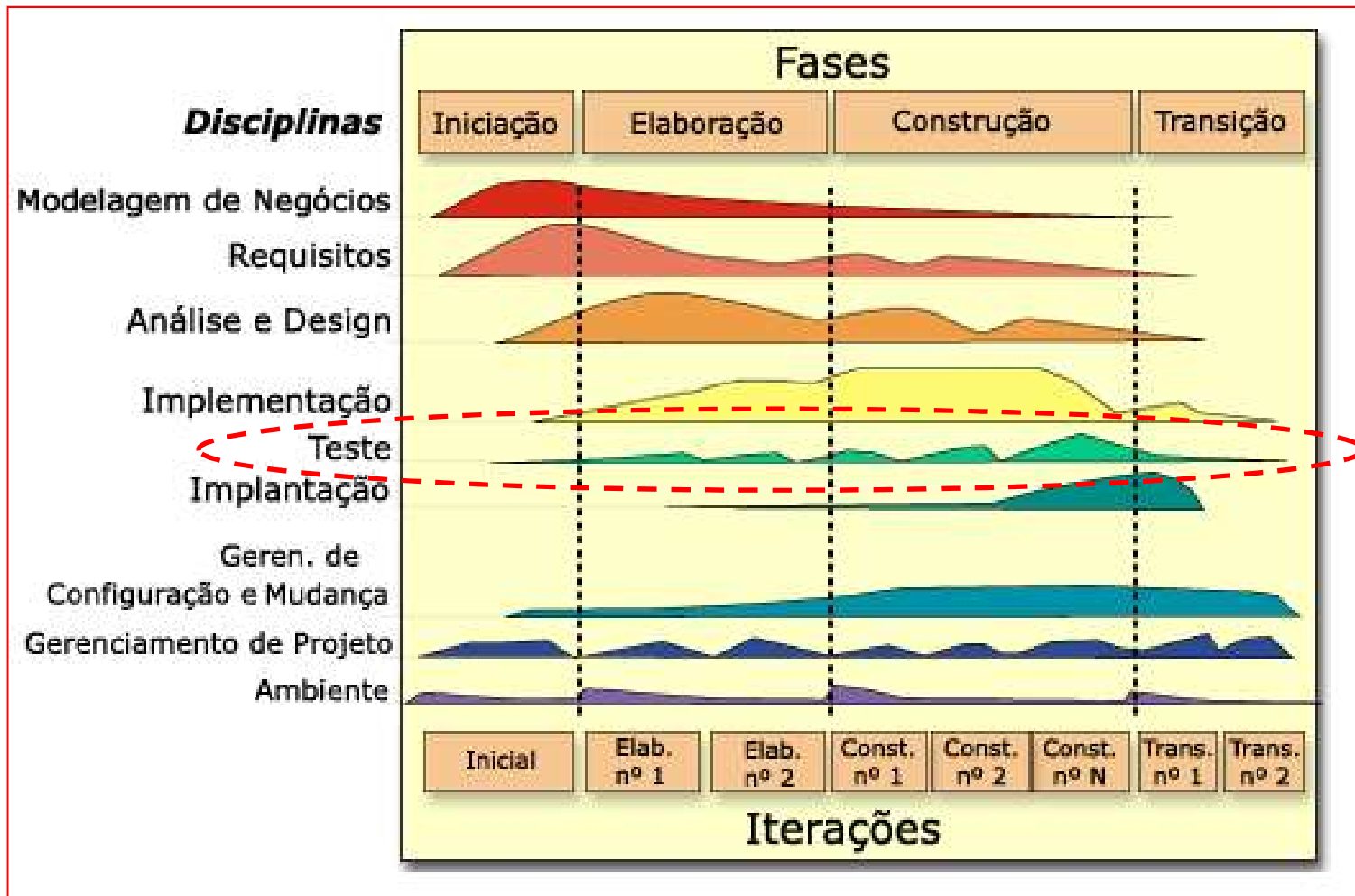
Atividades do Processo de Software

3



Processo Unificado

4



Objetivos do Processo de Teste

5

Demonstrar ao desenvolvedor e cliente que o software atende aos requisitos

Descobrir falhas ou defeitos no software que apresenta comportamento incorreto

Processo Básico de Teste

6

Teste de componentes (componentes individuais)

- Responsabilidade do desenvolvedor;

Teste de sistema (componentes integrados para sistema)

- Os testes são baseados em uma especificação de sistema.
- Os testes são derivados da experiência do desenvolvedor.

Teste de Aceitação

- Teste final do processo com dados dos clientes.

Política de Teste Geral

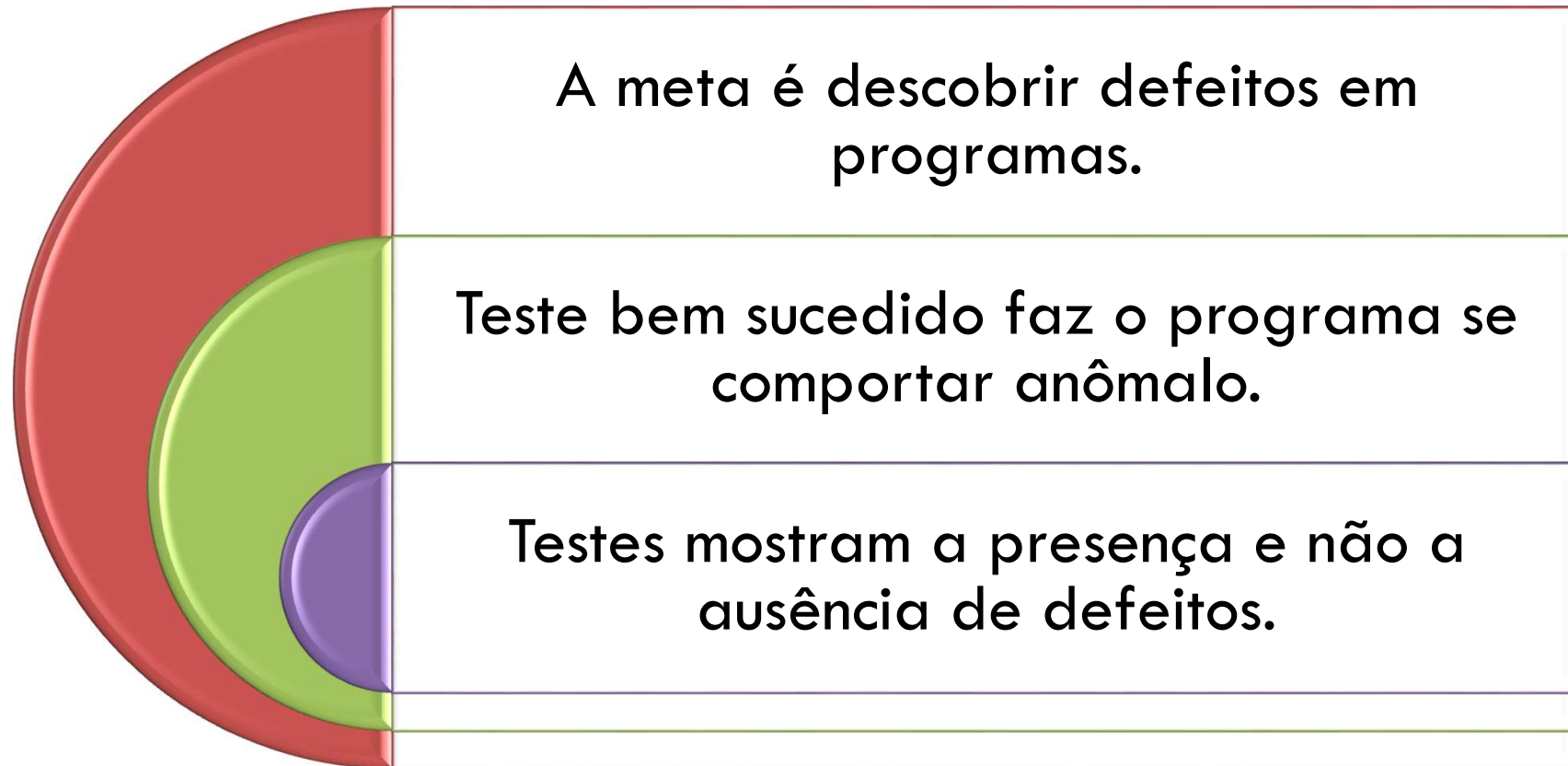
7

- Teste não demonstram que o software é **livre de defeitos**.
- Testes **exaustivos** são impossíveis.
- As **políticas de teste** abordam a seleção de testes de sistema:

- ✓ **Todas as funções acessadas por meio de menus devem ser testadas;**
- ✓ **Todos os procedimentos devem ser executadas;**
- ✓ **Devem ser testadas as entradas de usuário, com dados corretos e incorretos.**

Teste de Defeitos

8

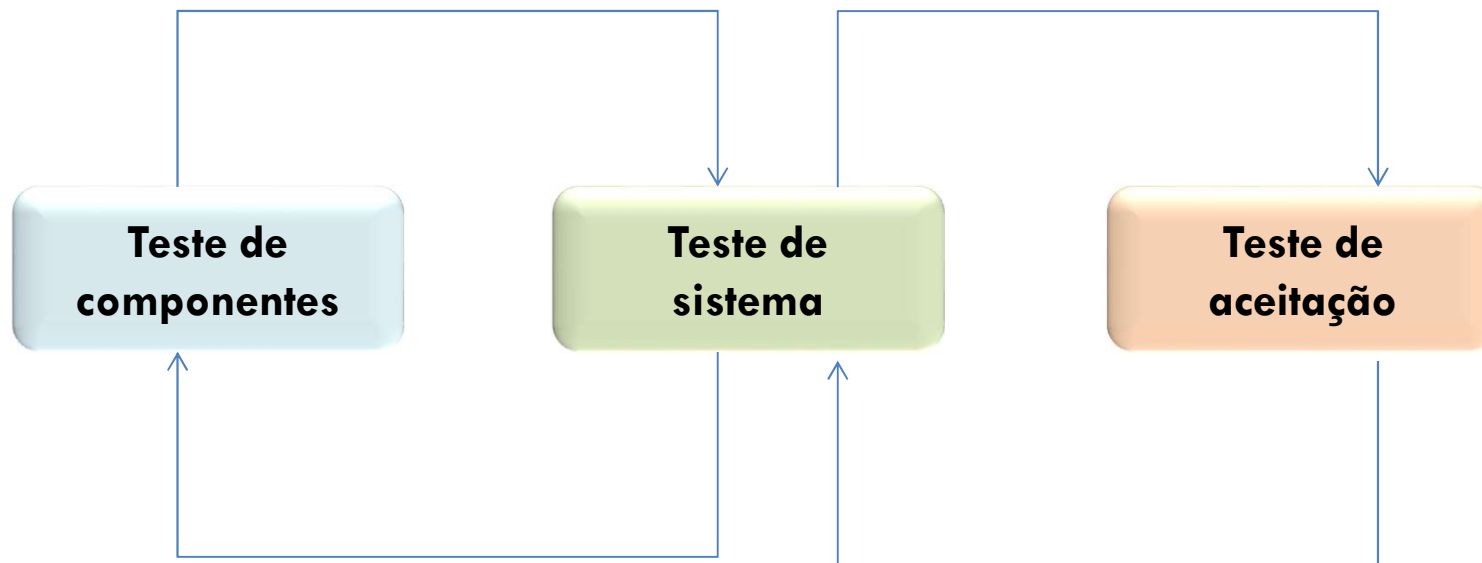


9



Processo Geral de Teste

10



Adaptado de Sommerville, 2006

TESTE DE COMPONENTES

Teste de Componentes — aplicações

12

Funções e métodos individuais de um objeto

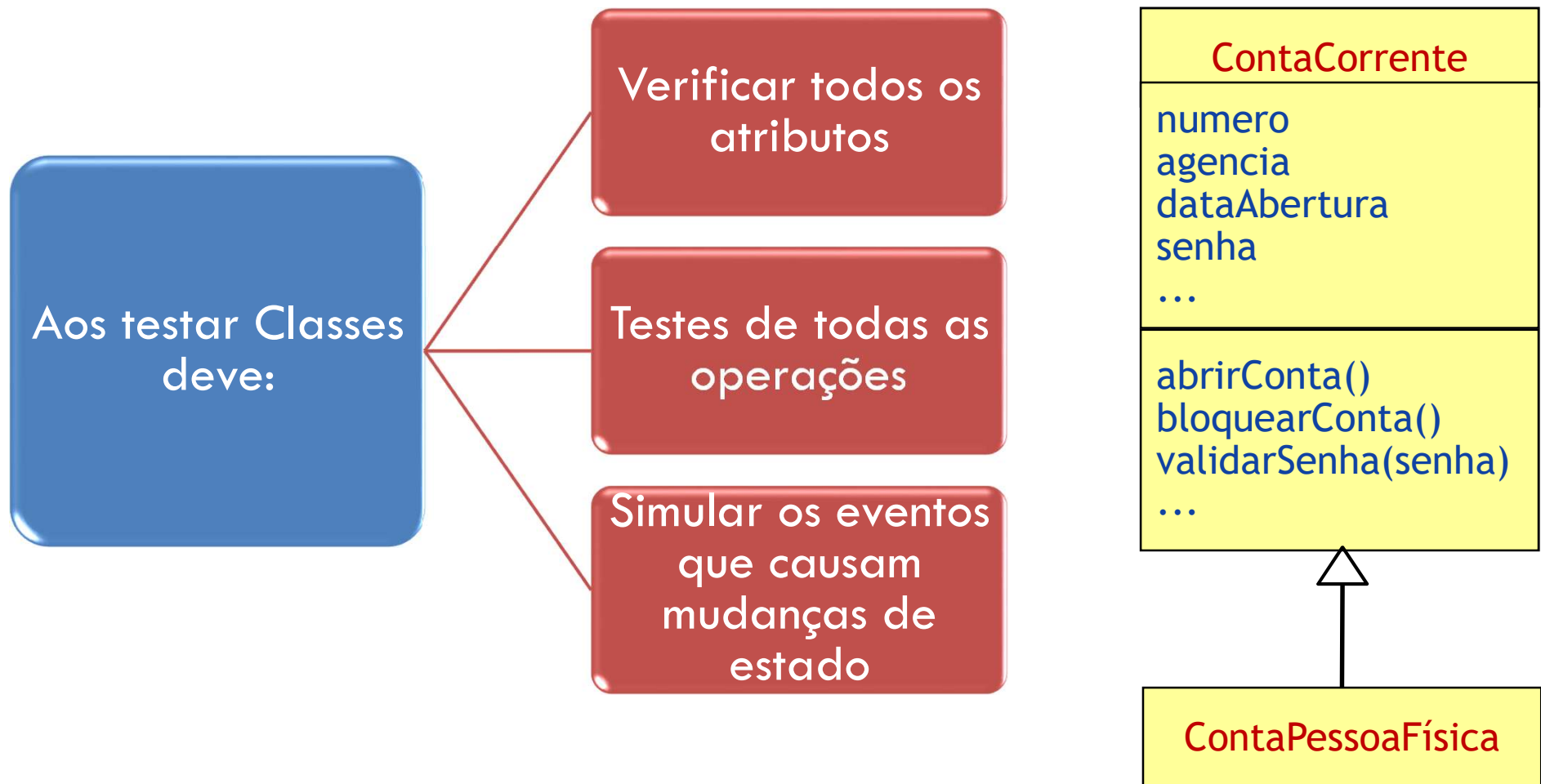
Composição de classes associadas com atributos e métodos

Componentes que constituem módulos e interface

Teste de Componentes

Testar Classes

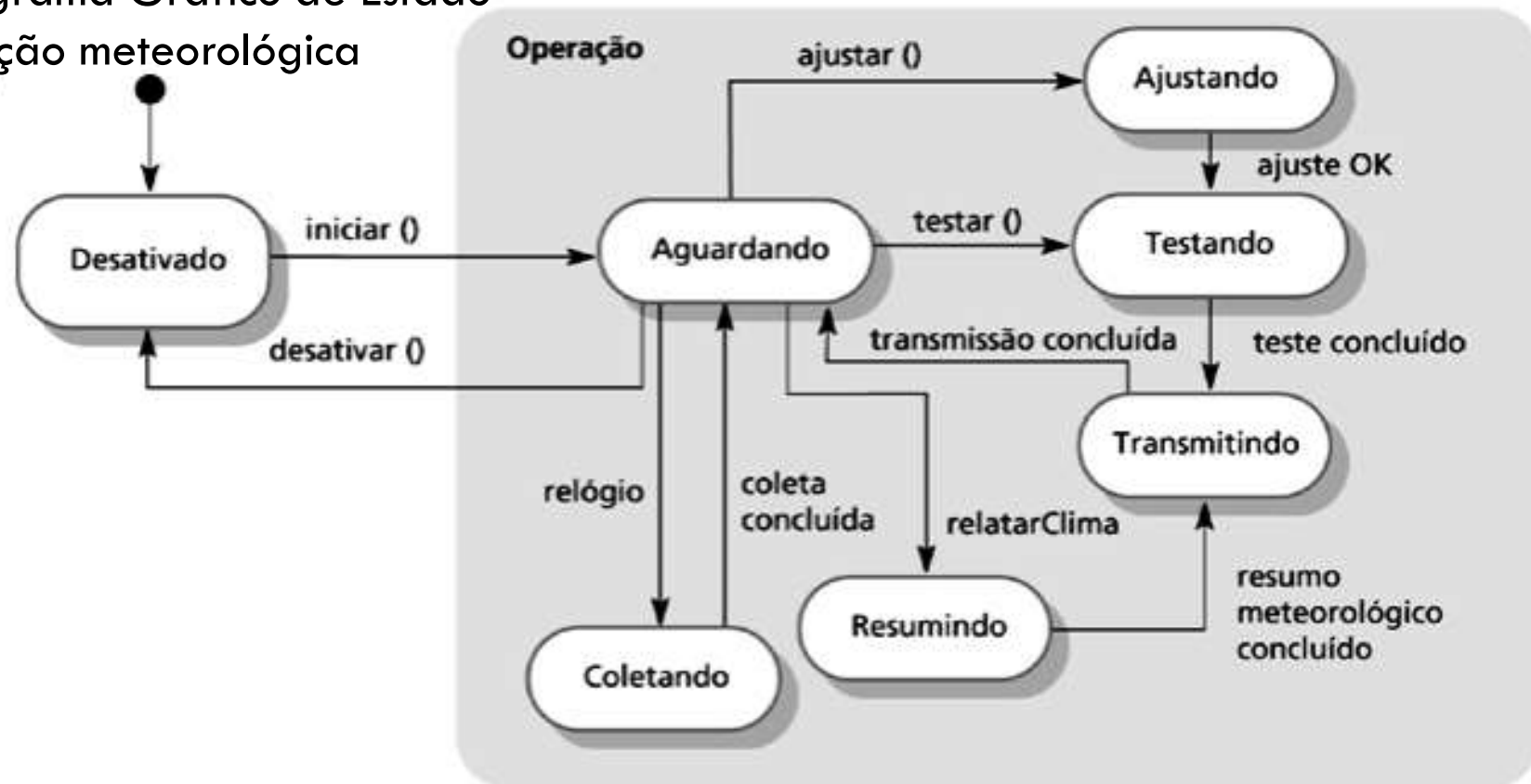
13



Teste de Estado de Objetos

14

Diagrama Gráfico de Estado
Estação meteorológica



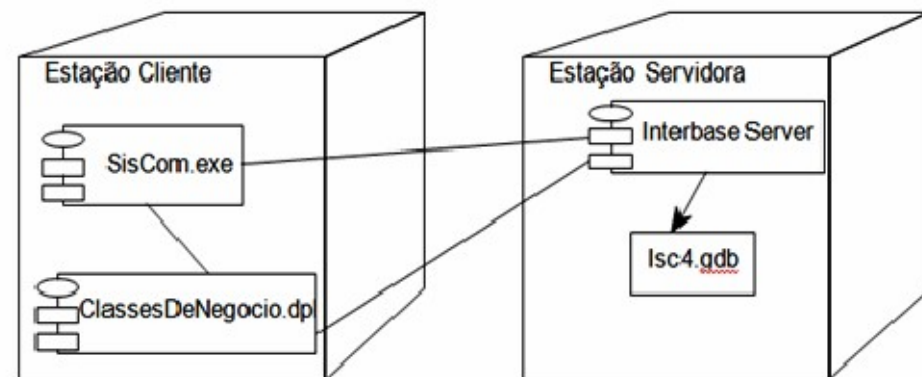
Identificar sequências de transições e definir sequências de eventos para testes.

Teste de Interfaces - Objetivos

15

Verificar defeitos em interface ou suposições inválidas sobre interfaces.

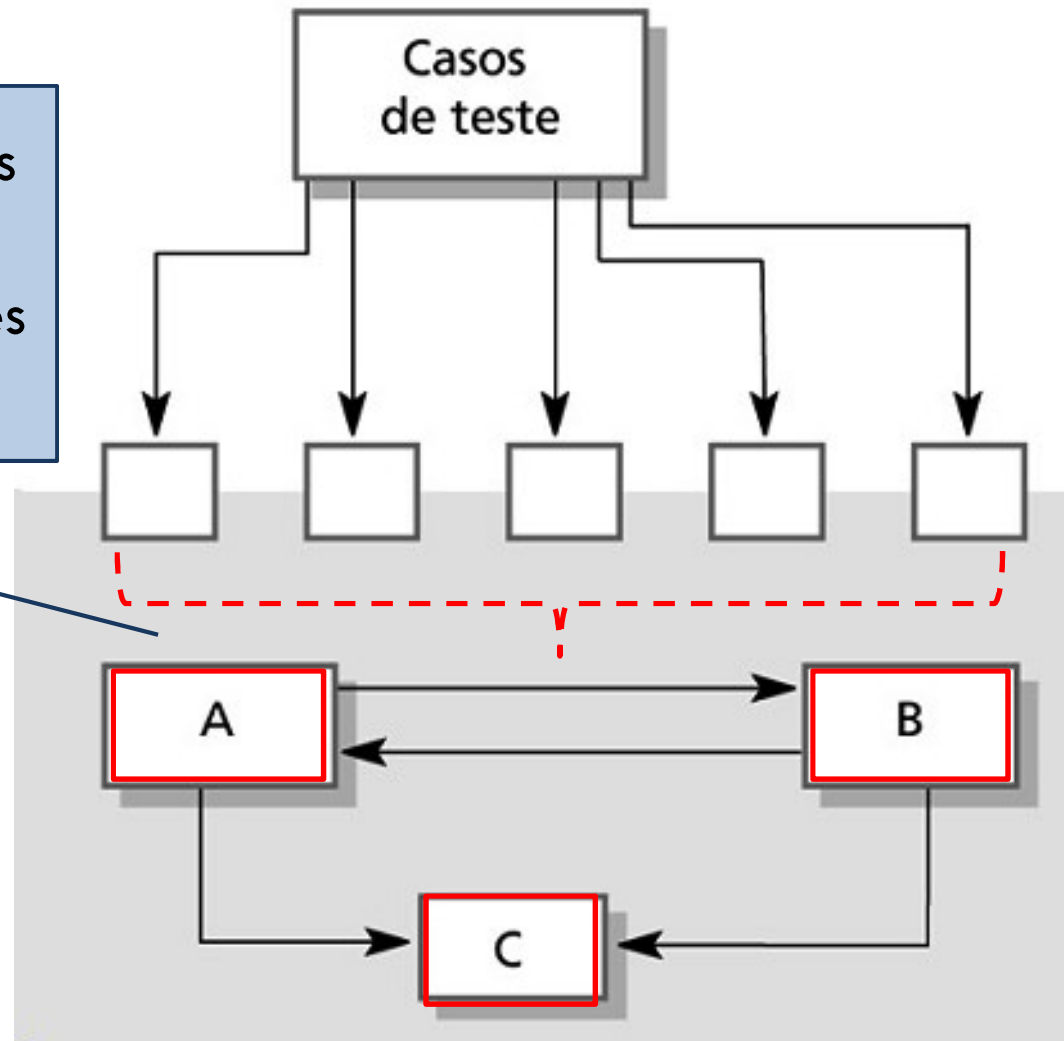
Concentra-se nos componentes, suas especificações e associações.



Teste de Interface

16

Testes aplicados
nas interfaces
dos componentes
compostos.



Tipos de Interface entre componentes

17

Tipos de interface podem apresentar erros:

Interface para passagem de parâmetros

Interface na memória compartilhada

Interface entre procedimentos

interface entre sistemas

E os principais erros podem ser ...

Erros de Interface

18

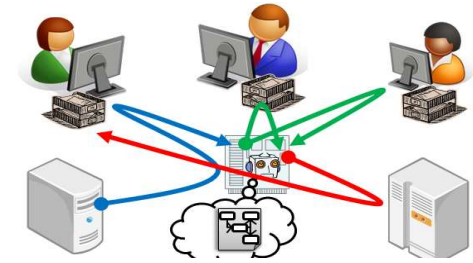
Mau uso de interface

Mau entendimento da interface
pelo componente chamador

Erros de *timing*, em sistemas de
tempo real

TESTE DE SISTEMA

Teste de Sistema

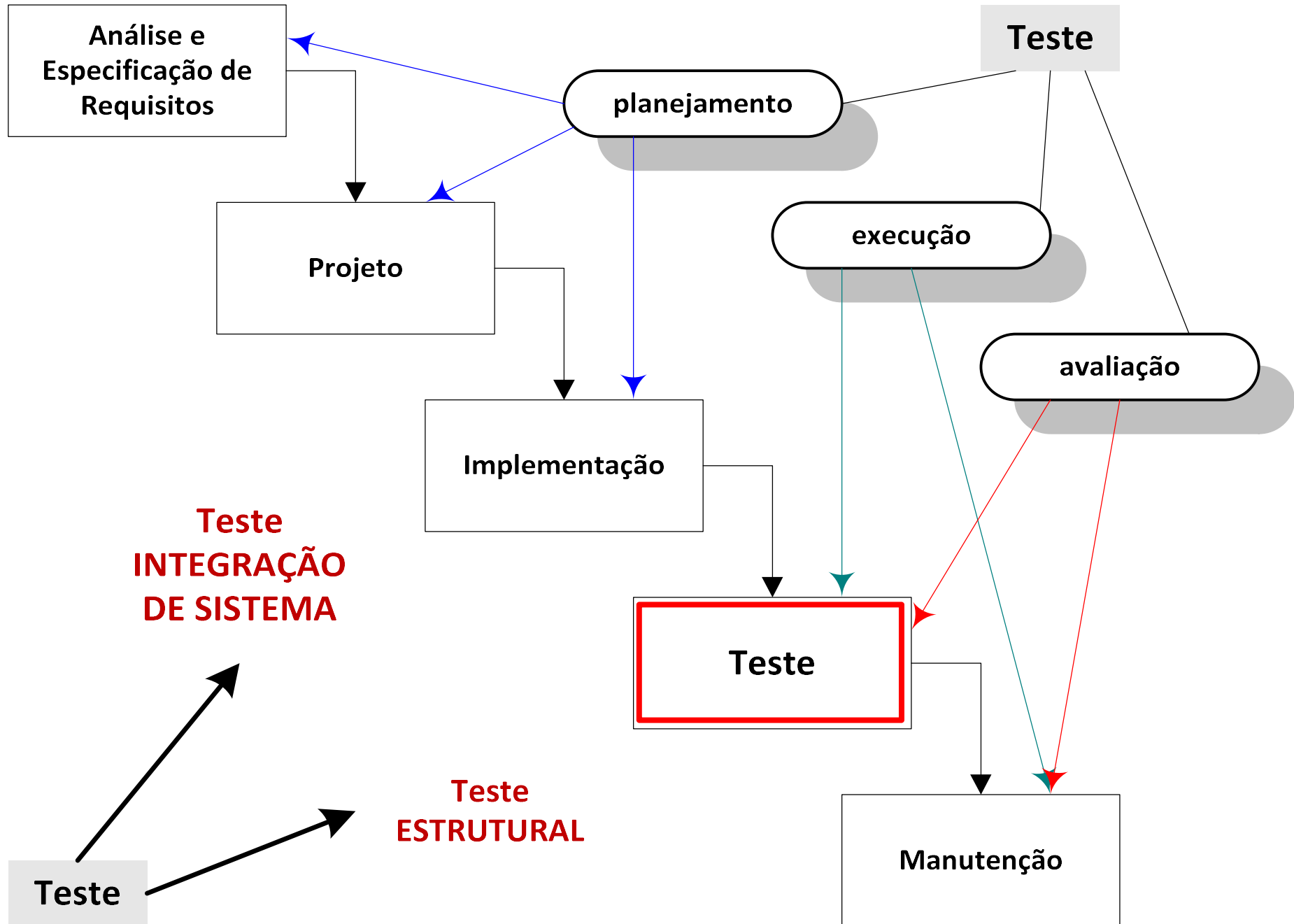


20

- Envolve a integração dos componentes que implementam funções do sistema.

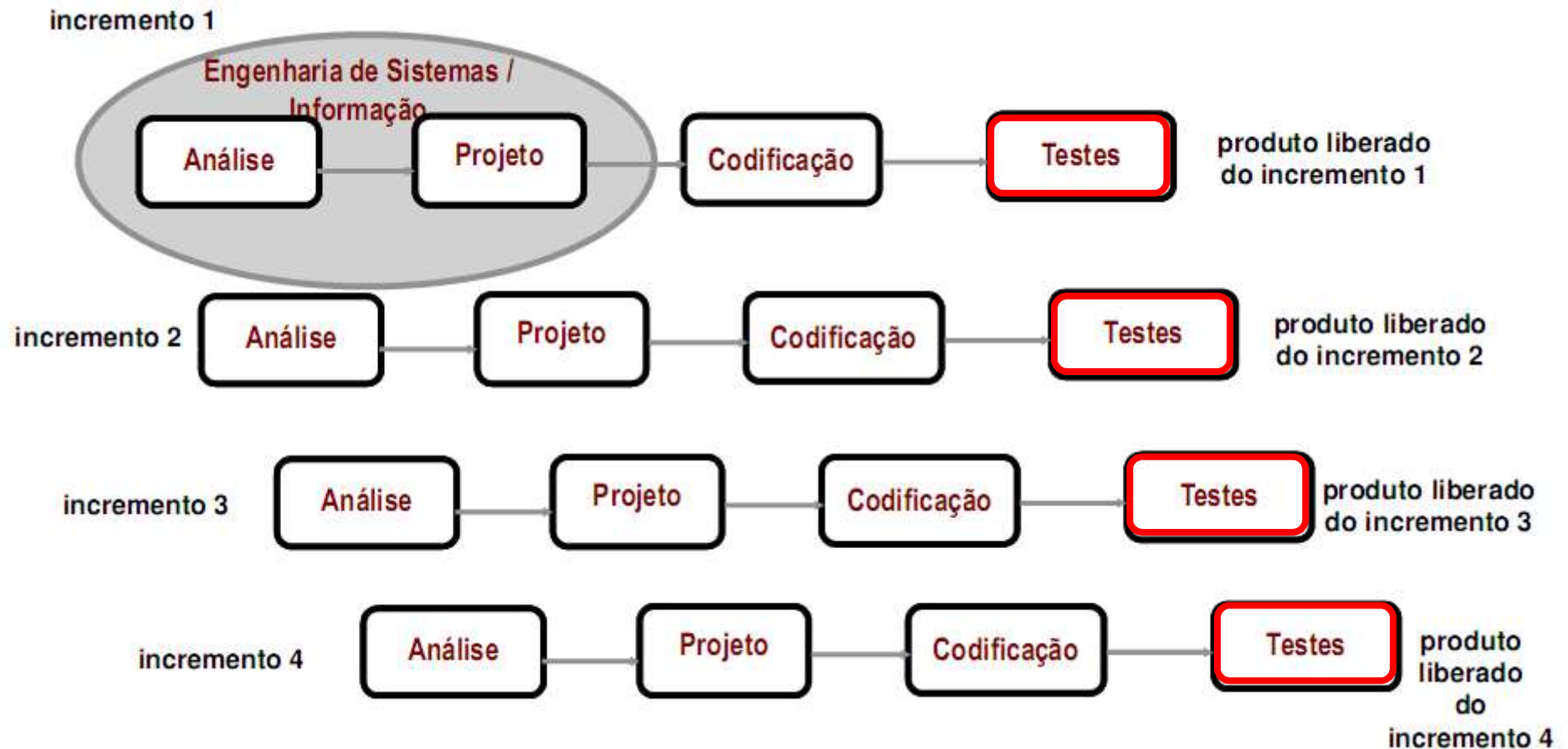
1. **Processo Cascata**: concentra-se no teste de **todo o sistema**.
2. **Processo Iterativo**: teste **a cada incremento** que será entregue.

Processo de teste - Cascata



Processo Iterativo e Incremental

22



Teste de Sistemas: fases distintas

23

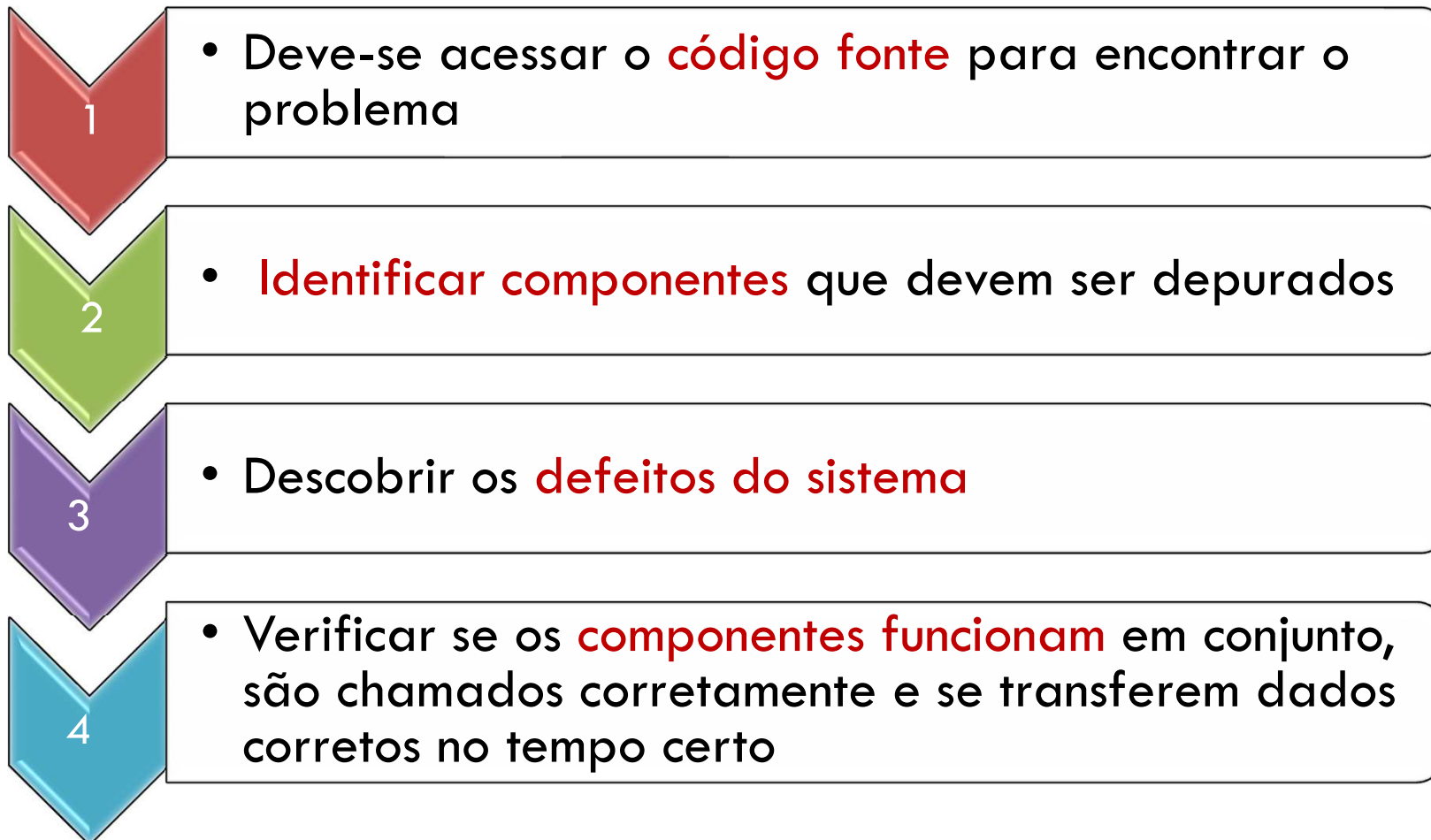
Teste de integração

Teste de *releases*

Teste de Integração

24

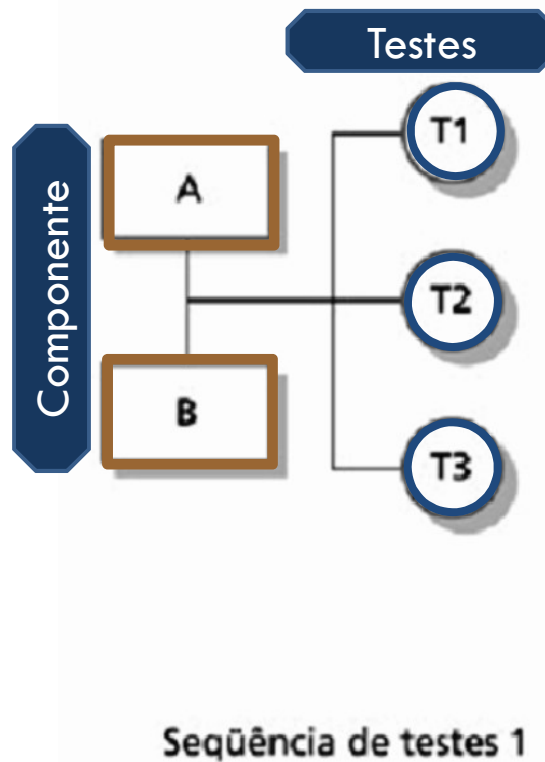
São relacionados à descobertas de **defeitos do sistema**.



Teste de Integração Incremental

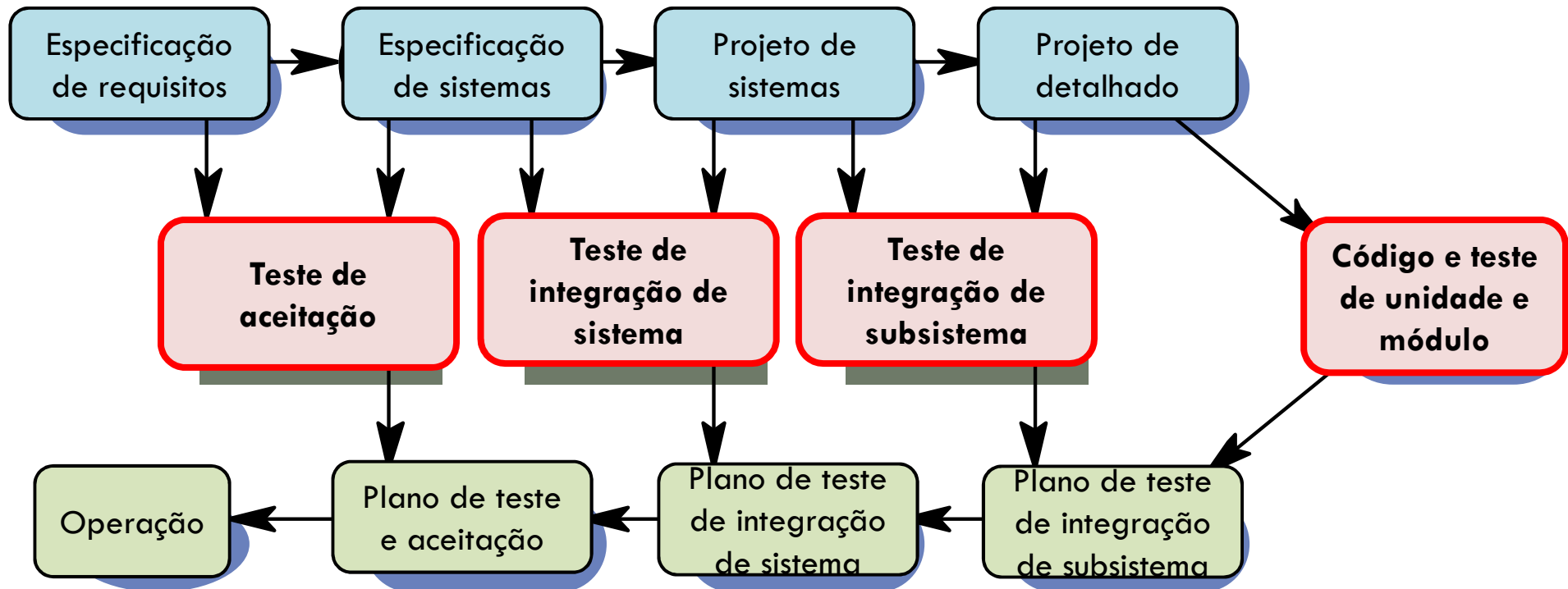
Teste de Regressão

25



Fases do Processo de Teste

26



Teste de Release

27

- Deve-se verificar e validar se o sistema atende aos requisitos do cliente.
- Aplicar um teste de “**caixa-preta**” que demonstra se o sistema funciona adequadamente ou não.

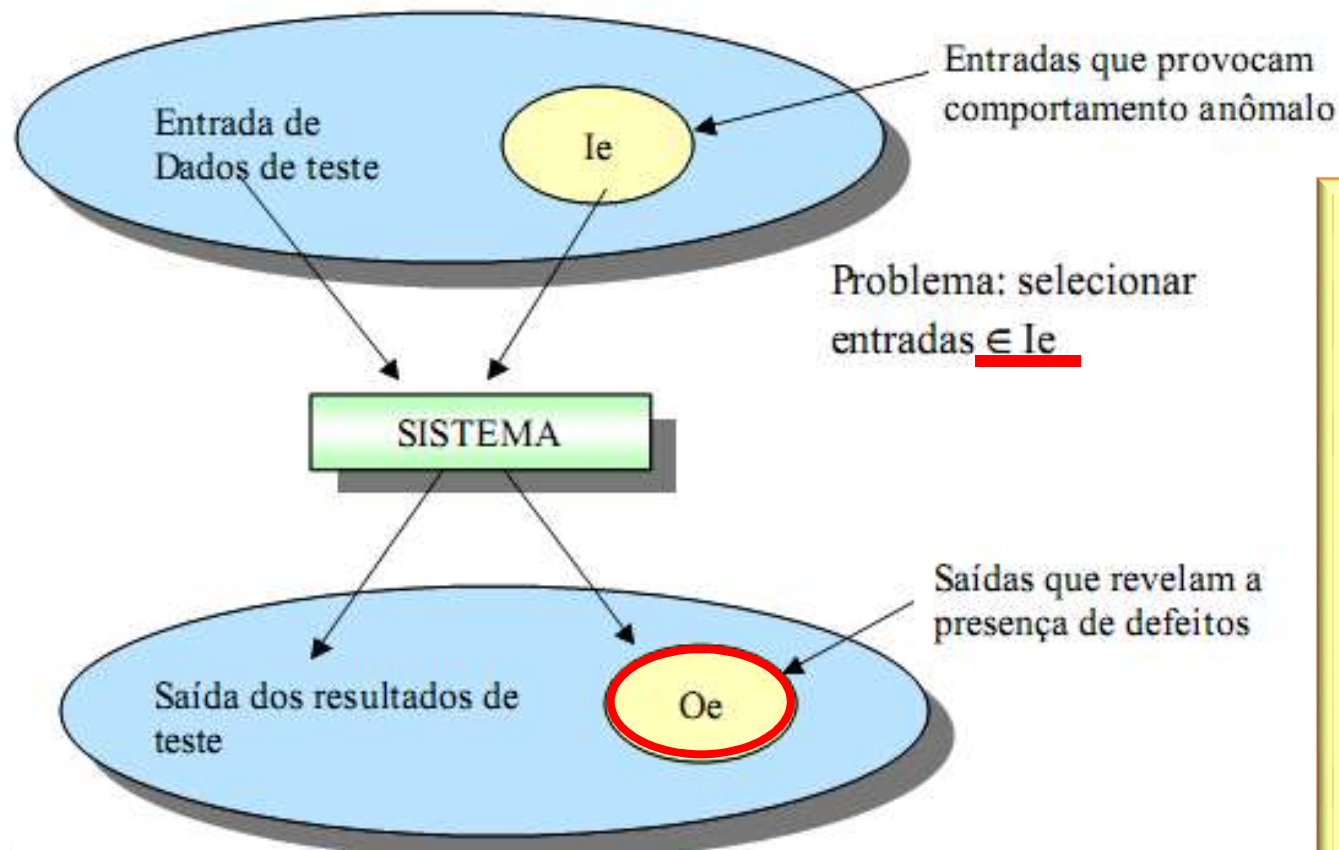
Teste Release

28

- Teste de release é, geralmente, um teste caixa-preta ou funcional;
- A meta é aumentar a confiança do fornecedor de que o sistema atende aos requisitos.
- ✓ É baseado somente na especificação de sistema;
- ✓ Os testadores não têm conhecimento da implementação do sistema.

Teste de Caixa-Preta

29



- 1) O testador fornece as entradas para o sistema;
- 2) Examina as saídas correspondentes;
- 3) Se as saídas **não** forem previstas (estiverem em **Oe**, detectou-se problema com o software;

Diretrizes para Teste Release

30

Escolher entradas que forcem gerar todas as **mensagens de erro**

Projetar entradas que causem **overflow** dos *buffers*

Repetir a mesma entrada várias vezes

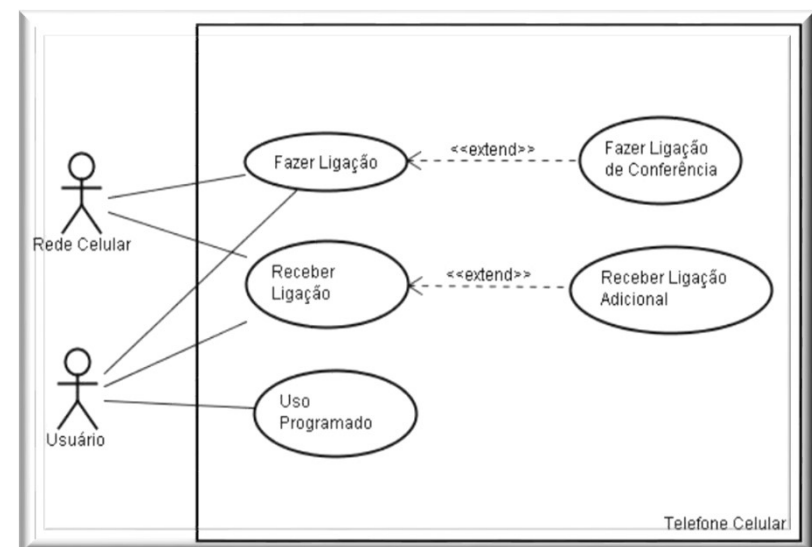
Forçar a geração de **saídas inválidas**

Forçar resultados de **cálculo** a serem muito grandes ou muito pequenos

Diagramas Casos de Uso/Sequência

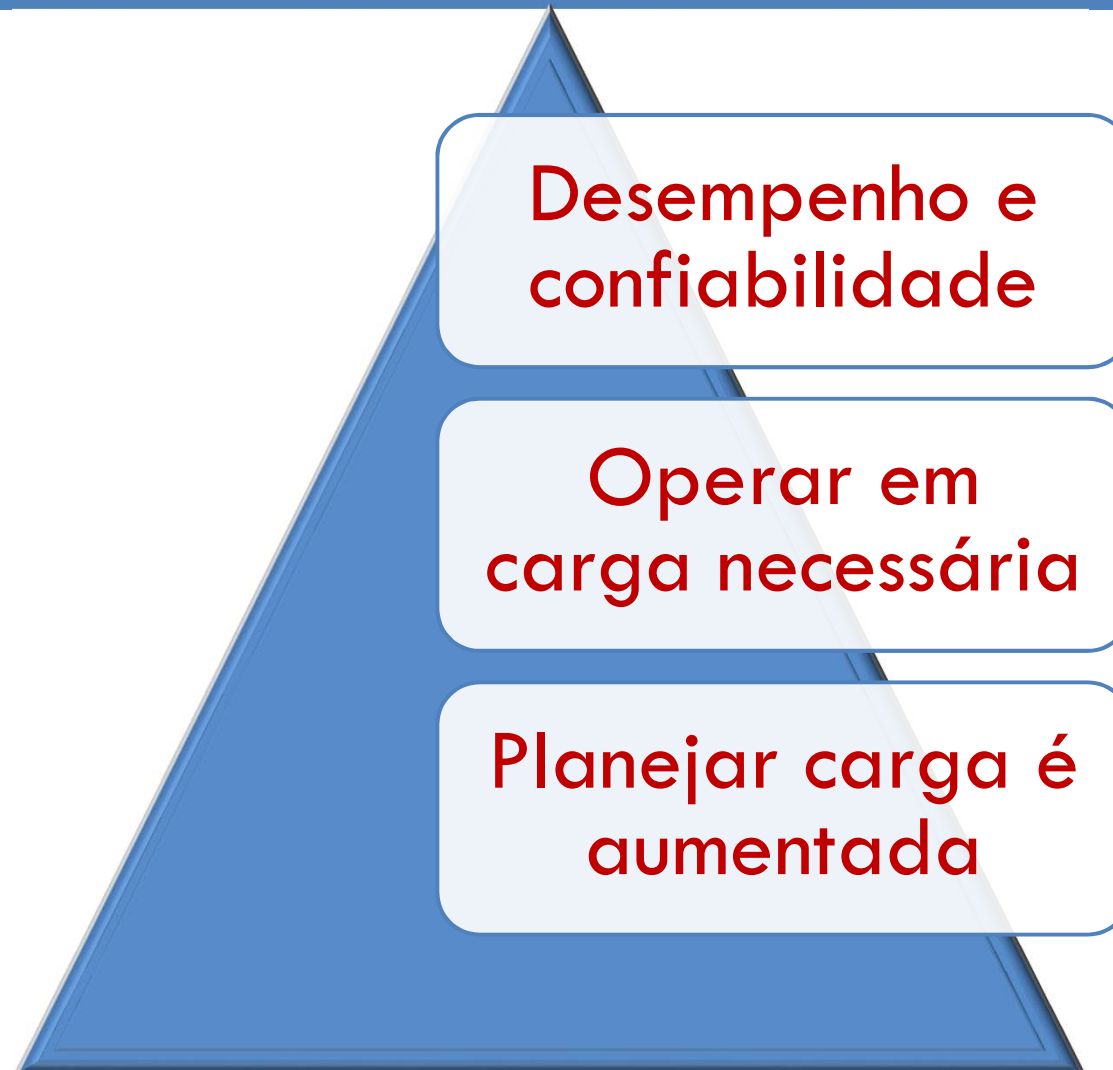
31

- ❑ **Casos de uso** podem ser uma base para derivar os testes e validação de sistema.
- ❑ **Diagrama de sequência** associado, onde as entradas e saídas para os testes podem ser identificadas.



Teste de Desempenho do Sistema

32



Teste de Desempenho - técnicas

33

Testar os requisitos de desempenho



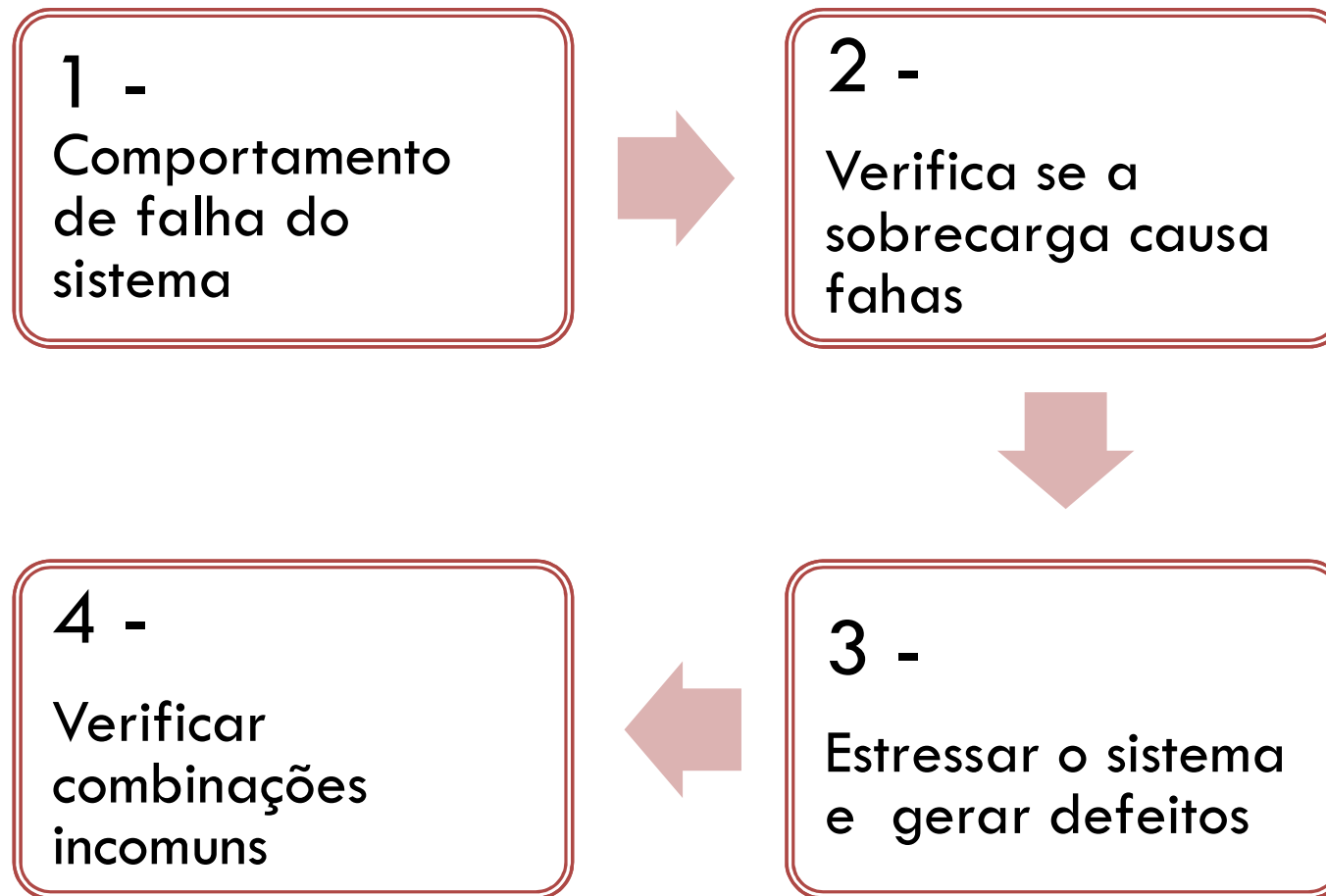
Testar todos os procedimentos mais usados pelo cliente



Projetor entradas no limite do sistema, ou seja, **teste de estresse**

Teste de Estresse

34



TESTES ESTRUTURAIS

complexidade ciclomática

Definição e Objetivo

36

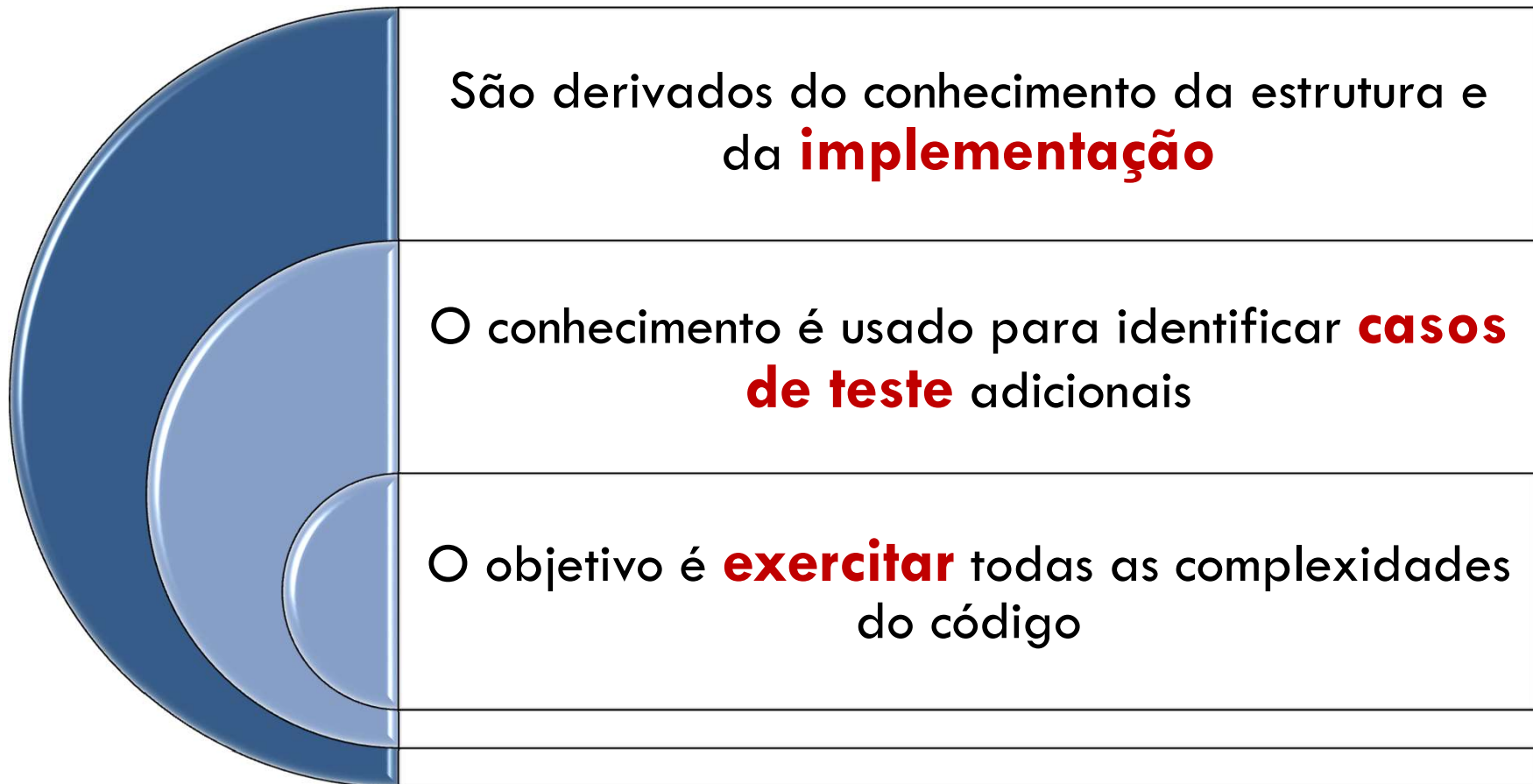
É parte dos testes de sistemas e componentes

Projeta-se casos de entradas e saídas esperadas nos controle de fluxos

O objetivo é encontrar defeitos e verificar se o sistema atende os requisitos

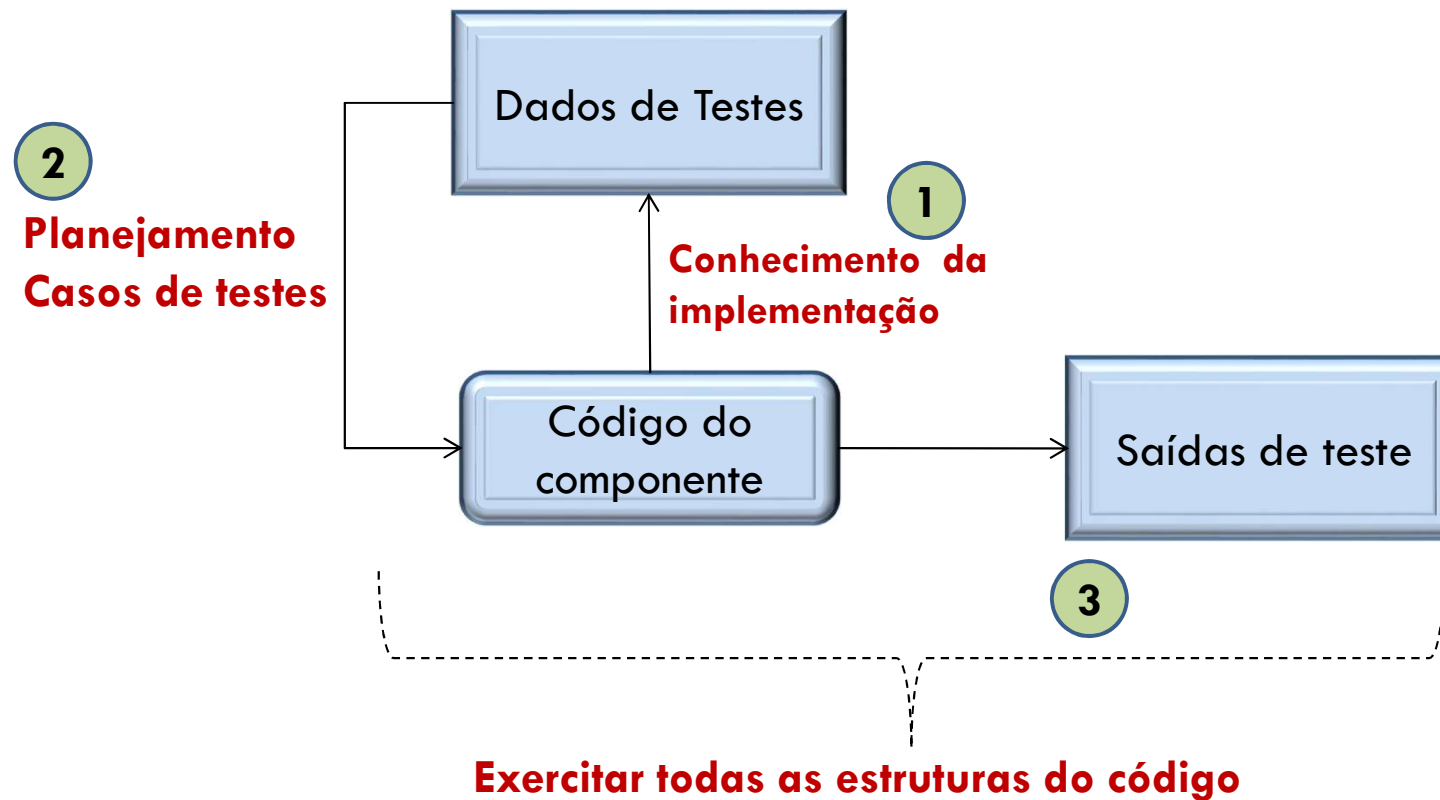
Teste Estrutural ou de Caixa-Branca

37



Esquema do Teste Estrutural

38



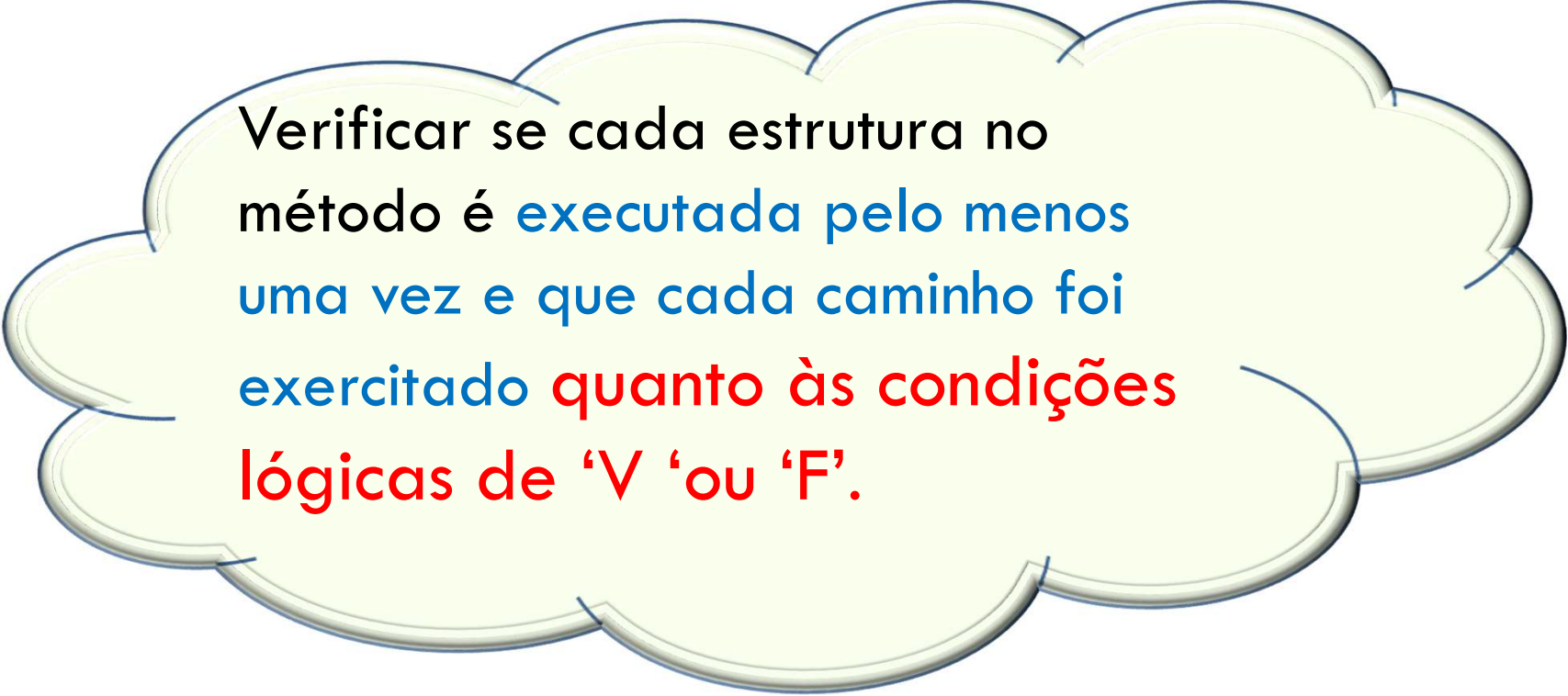
Complexidade Ciclomática

Teste de Caminho

Complexidade Ciclomática

Teste de Caminho

40

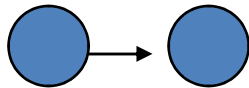


Verificar se cada estrutura no método é executada pelo menos uma vez e que cada caminho foi exercitado quanto às condições lógicas de 'V' ou 'F'.

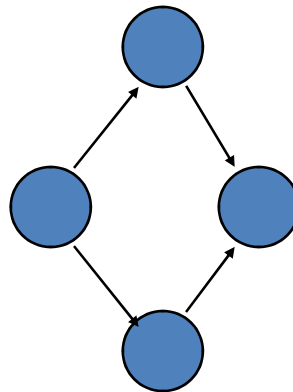
GRAFO DE FLUXO

41

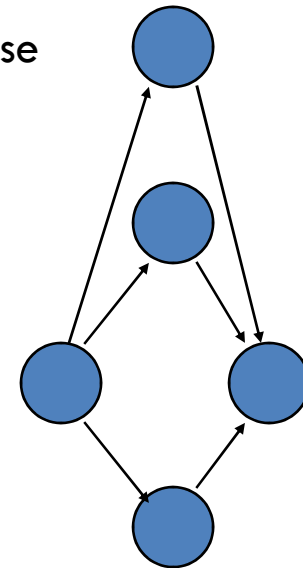
Sequence



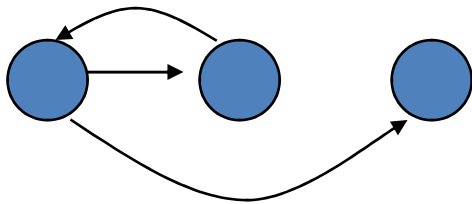
If



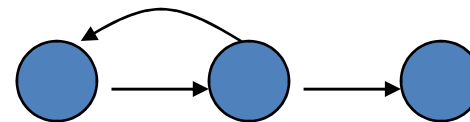
Case



While



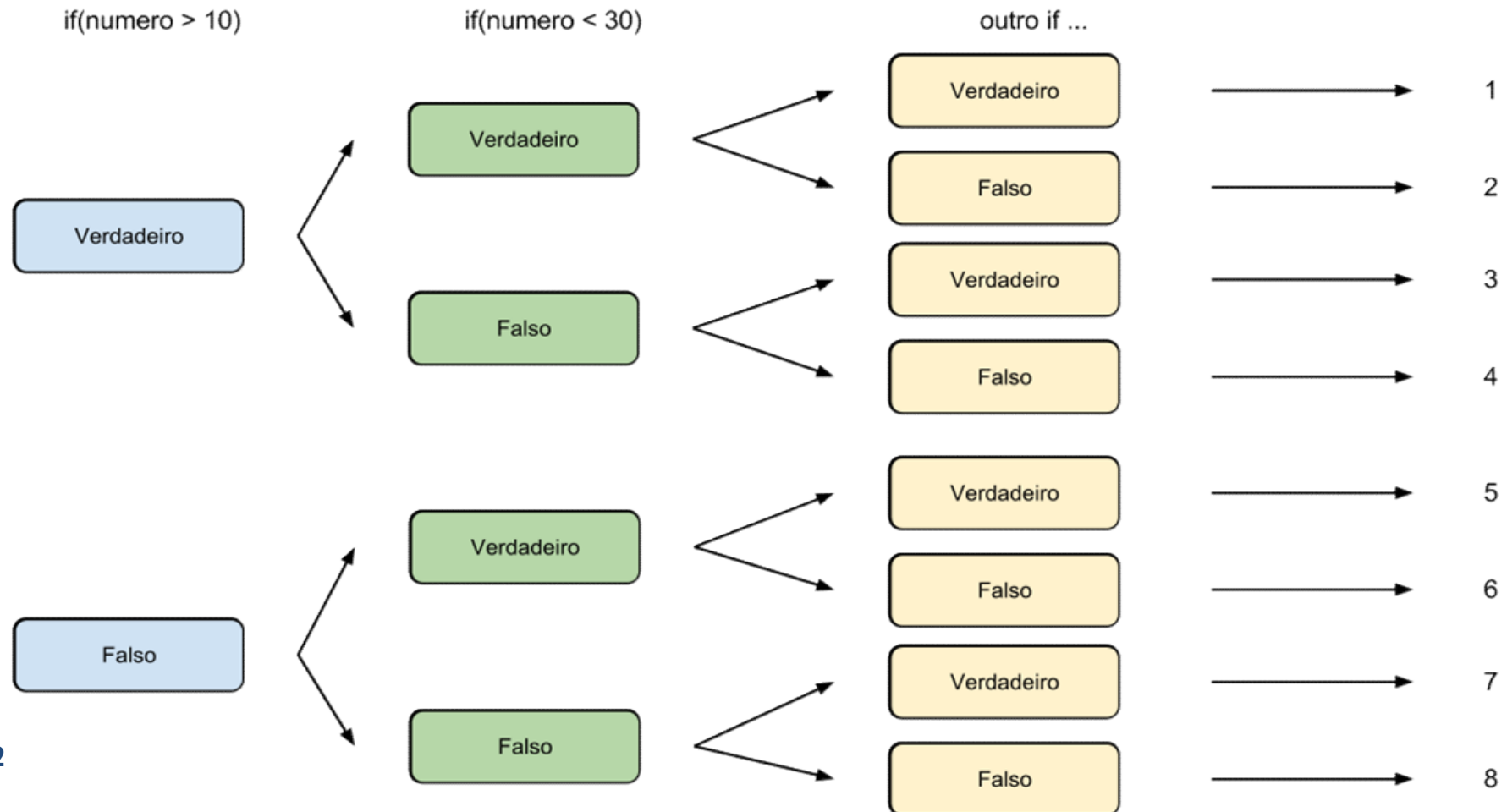
Until



Número da Complexidade, (Caleum, 2016)

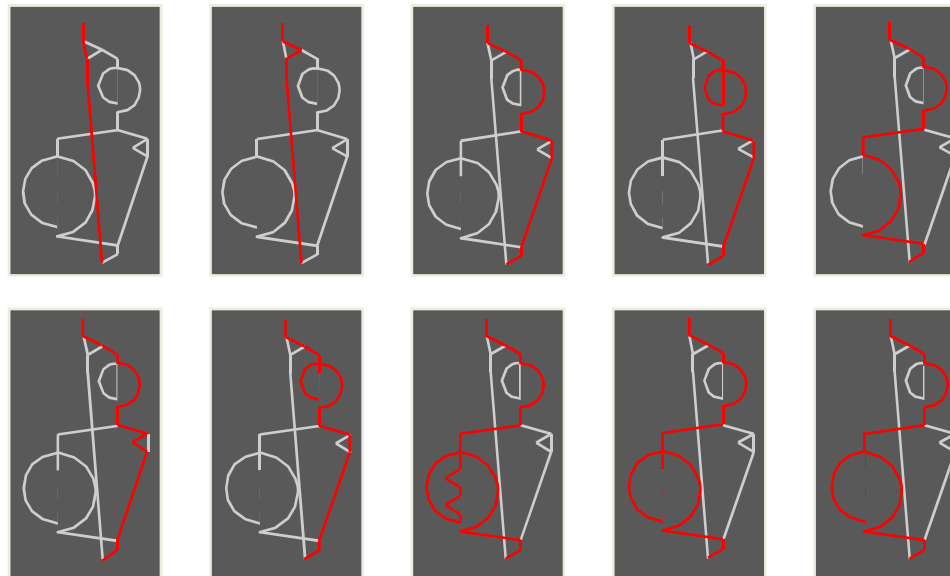
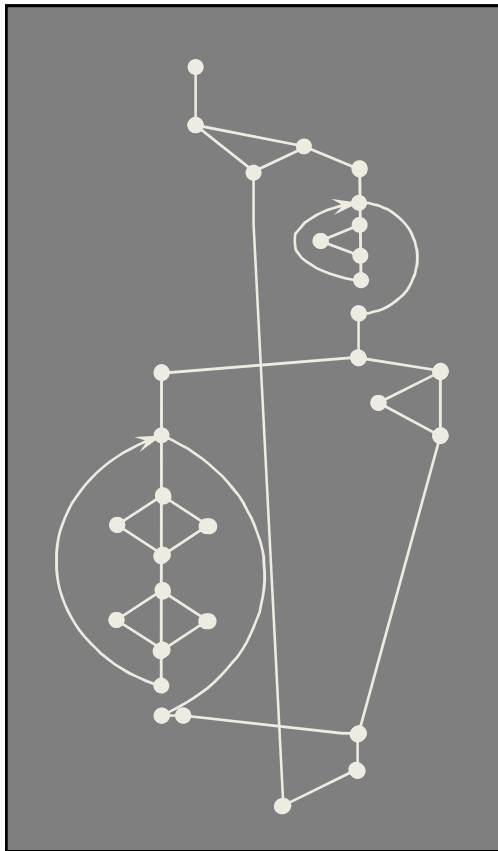
```
public int contaMaluca(int numero) {  
    int resultado = 10;  
    if (numero > 10) resultado += numero;  
    return resultado;  
}
```

```
public int contaMaluca(int numero) {  
    int resultado = 10;  
    if (numero > 10) resultado += numero;  
    if (numero < 30) resultado *= 2;  
    return resultado;  
}
```



POSSIBILIDADE DE EXECUÇÃO DE FLUXO

Análise do controle de fluxo nas várias possibilidades de execução



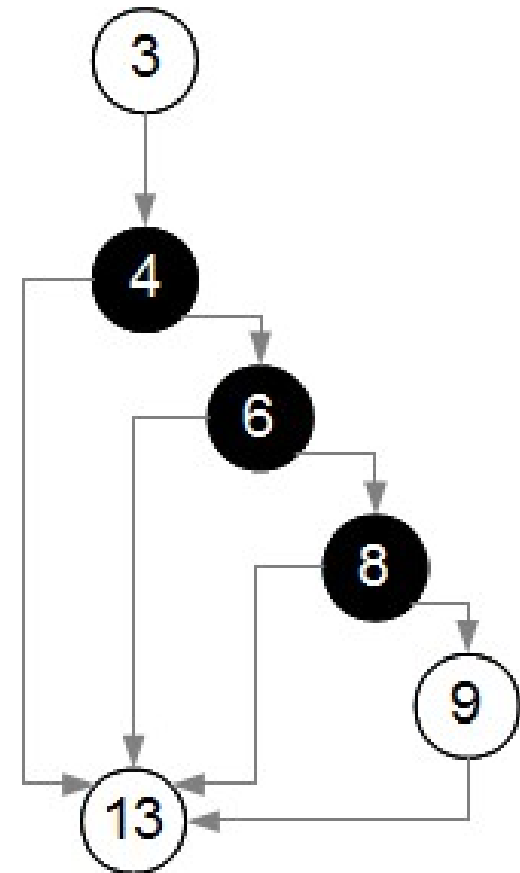
- **Complexidade = 10**
Significa que mínimo 10 testes deverão ocorrer:
- Todo o fluxo é revisto
 - Testa as Decisões Lógicas

Exemplo – busca binária

```
public static void pesquisa(int key, int[] vetor, Result r) {  
    int bottom = 0;  
    int top = vetor.length - 1;  
    int mid;  
    r.found = false;  
    r.index = -1;  
  
    while (bottom <= top) {  
        mid = (top + bottom) / 2;  
        if (vetor[mid] == key) {  
            r.index = mid;  
            r.found = true;  
            return;  
        } else {  
            if (vetor[mid] < key) {  
                bottom = mid + 1;  
            } else {  
                top = mid - 1;  
            }  
        }  
    }  
}
```

Complexidade Ciclomática

```
01) public class TesteCiclo {  
02) public double calcular (int parametro1, int parametro2) {  
03) double retorno = 0.0d;  
04) if (parametro1 <= 7) {  
05)     retorno = parametro1 * 1.5;  
06)     if (parametro2 > parametro1) {  
07)         retorno = retorno * 0.75;  
08)         if (parametro2 == 1) {  
09)             retorno = -1.0d;  
10)         }  
11)     }  
12) }  
13) return retorno;  
14) }  
15)}
```



CAMINHO INDEPENDENTE

(considerações complexidade ciclomática)

46

- **Um caminho independente** é um fluxo no grafo que inclui pelo menos **uma aresta nova** (que não tinha sido atravessada);
- **Conjunto básico** é o conjunto formado pelos caminhos independentes que cubram todas as arestas do grafo de fluxo;
- A **complexidade ciclomática** é uma métrica de software que proporciona uma medida da complexidade lógica de um programa;
- O valor da complexidade ciclomática estabelece **um limite superior** para o número de caminhos independentes.

Complexidade Ciclomática

Cálculo

1) Contar o número de sentenças lógicas (**P**) nos desvios e laços de loop;

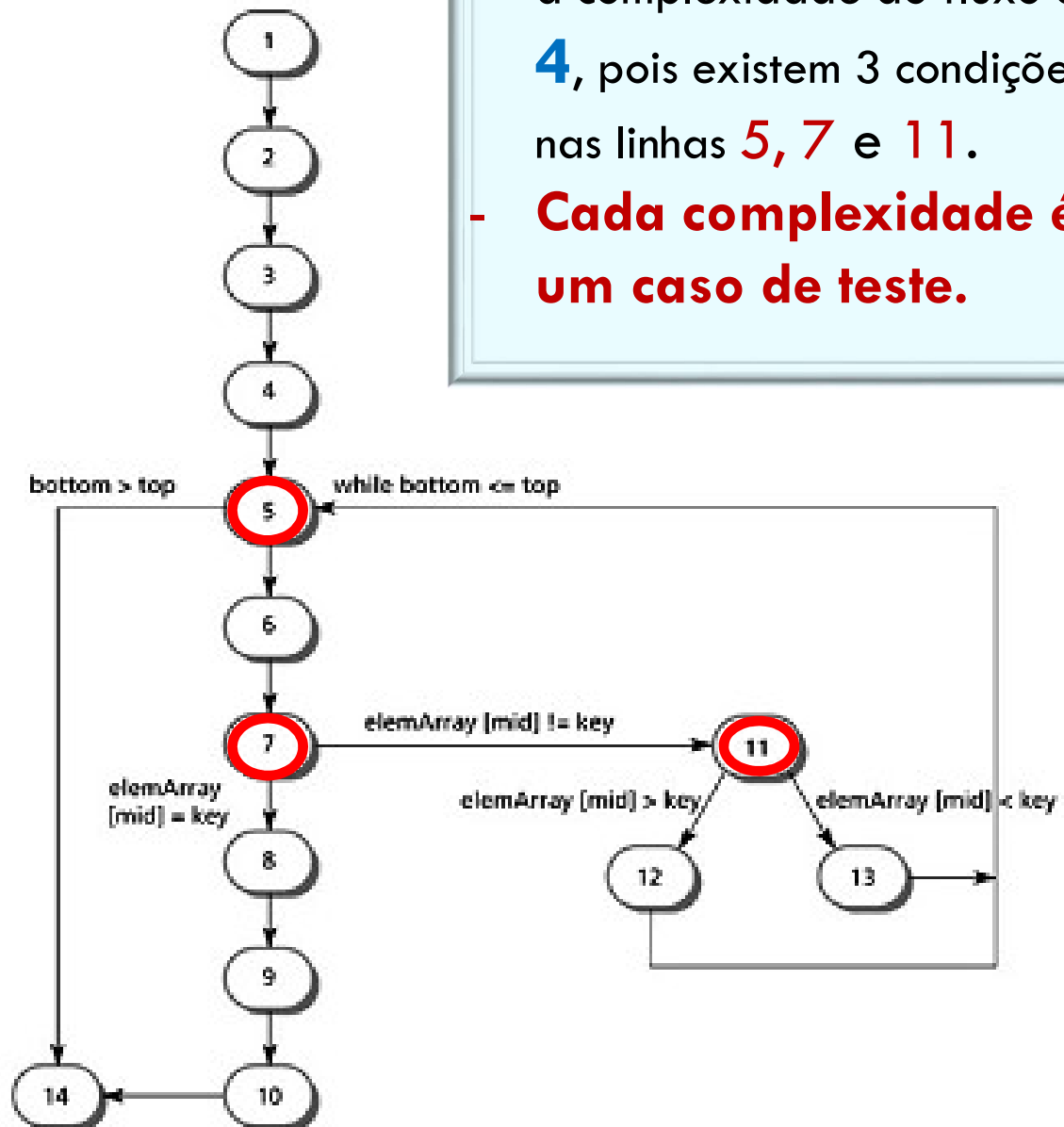
2) O Valor da complexidade cliclomática é um a mais do que o número de condições no programa, dada por:

$$V(G) = P + 1;$$

47

Exemplo:

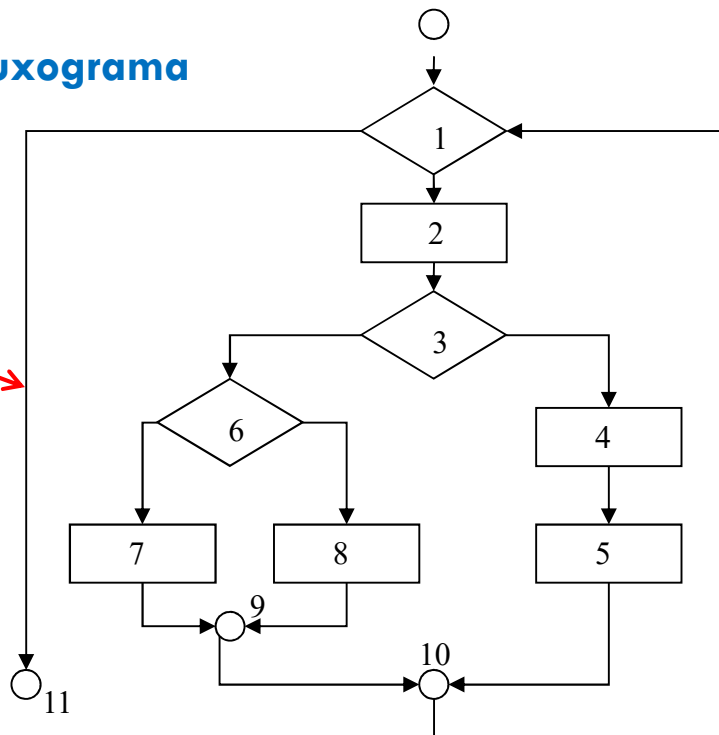
- a complexidade do fluxo é **4**, pois existem 3 condições nas linhas **5, 7 e 11**.
- **Cada complexidade é um caso de teste.**



GRAFO DE FLUXO

```
1: enquanto existir registro faça
2:   leia registro
3:   se registro.campo1 = 0 então
4:     processar registro e armazenar em buffer
5:     incrementar contador
6:   senão se registro.campo2 = 0 então
7:     resetar contador
8:   senão
9:     processar registro e armazenar em arquivo
10:  fimse
11: fimse
12: fimenquanto
```

Fluxograma

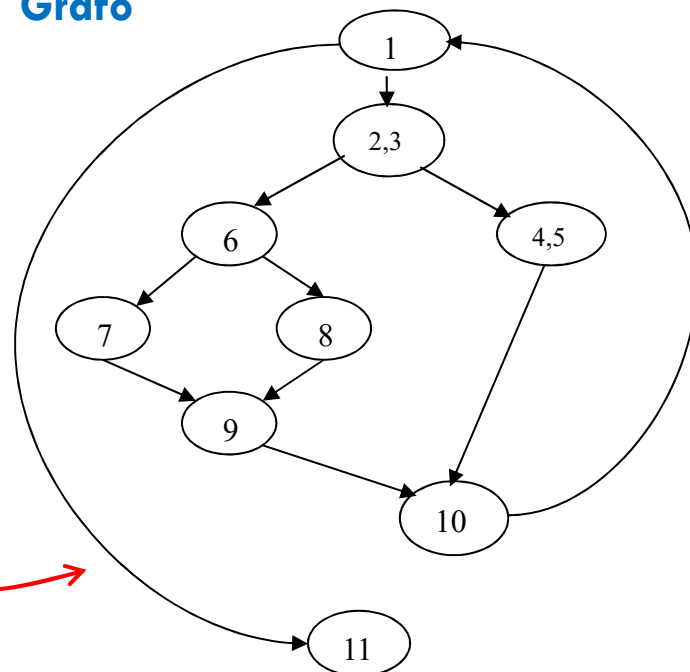


Cada círculo é o nó do grafo, representa um ou mais comandos.

Uma sequência de retângulos de processamento (atribuição, cálculo, declaração, inicialização) e um losango de decisão podem ser mapeados em um único nó.

As setas do grafo (arestas) são análogas ao do fluxograma.

Grafo



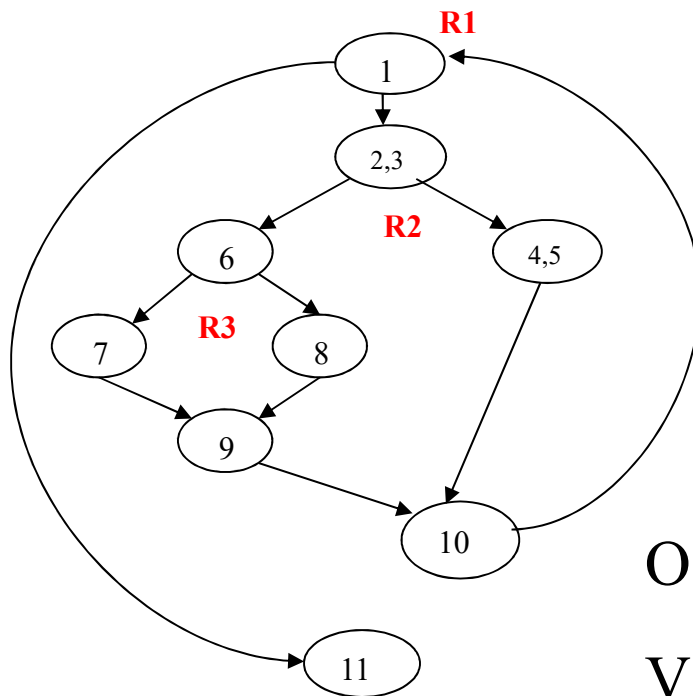
COMPLEXIDADE CICLOMÁTICA

49

- A complexidade ciclomática
 - ▣ Baseado na teoria dos grafos
 - ▣ Dado um grafo G , $V(G) =$
 - $E - N + 2$, onde E corresponde ao número de arestas e N o número de nós do grafo de fluxo.
 - $P + 1$, onde P é o número de nós predicativos (desviantes) contidos no grafo.

COMPLEXIDADE CICLOMÁTICA

50



O grafo de fluxo tem 4 regiões.

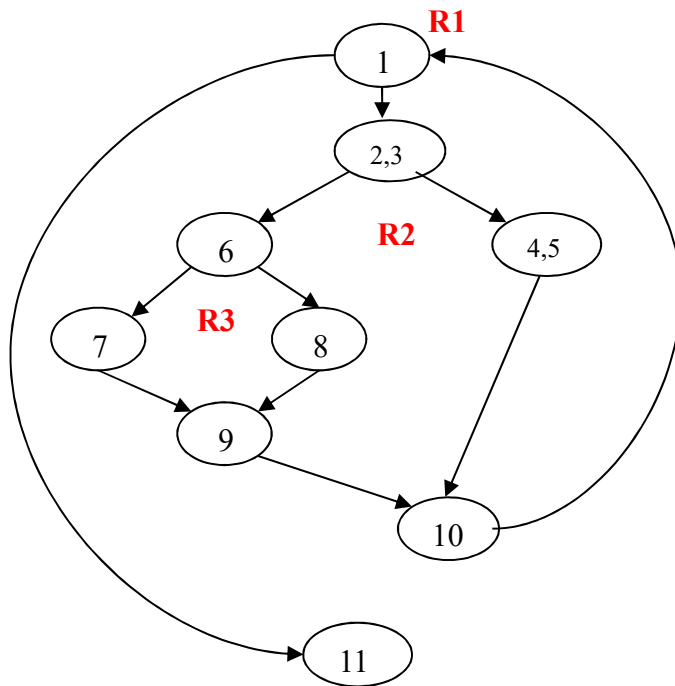
$$V(G) = 11 \text{ arestas} - 9 \text{ nós} + 2 = 4$$

$$V(G) = 3 \text{ nós predicativos} + 1 = 4$$

Portanto a complexidade ciclomática é 4

Caminhos Independentes (livres de laços)

51



1: 1-11

2: 1-2-3-4-5-10-1-11

3: 1-2-3-6-8-9-10-1-11

4: 1-2-3-6-7-9-10-1-11

Não são independentes

1-2-3-4-5-10-1-2-3-6-8-9-10-1-11

1-2-3-4-5-10-1-2-3-6-7-9-10-1-11

Metrica de Complexidade Cyclomática

(Thomas McCabe, 1976)

52

Table 1. Standard Values of Cyclomatic Complexity

Cyclomatic Complexity	X	Risk Complexity
1-10		a simple program, without much risk
11-20		more complex, moderate risk
21-50		complex, high risk
51+		untestable, very high risk

Cyclomatic Complexity	X	Error Probability
1 – 10		5%
20 –30		20%
> 50		40%
Approaching 100		60%

Metrica de Complexidade Cyclomática

(Steve McConNoell, 2004)

53

- **De 0 a 5:**

- Seu código está, provavelmente, ok;

- **Entre 6 e 10:**

- Pense em maneiras de simplificar o código;

- **Maior que 10:**

- Quebre o código em dois e insira uma chamada da primeira parte para a segunda;

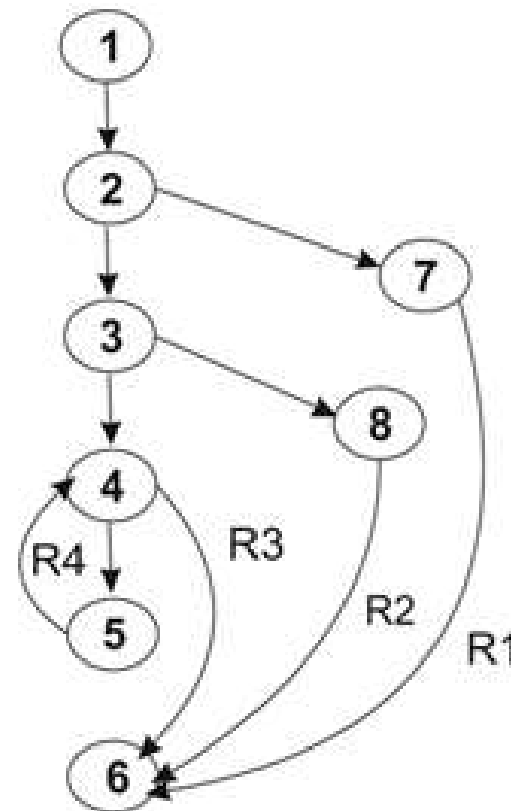
Teste Grafo do Fluxo

Matriz do grafo

Teste Grafo do Fluxo

Exemplo derivação para grafo

```
int fib (int n) {  
1  < int a = 1;  
    int b = 1;  
    int c = 2;  
  
2    if (0 == n)  
    return 0; ←7  
3    if (n < 3)  
    return 1; ←8  
  
4    while (n-- > 2) {  
5      < c = a + b;  
        a = b;  
        b = c;  
    }  
  
6    return c;  
}
```



Teste Grafo do Fluxo

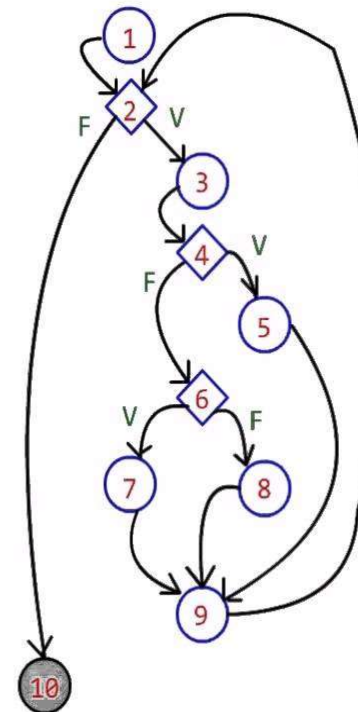
Exemplo derivação para grafo

```

public static int buscaBinaria(int[] array, int valor){
    int inicio = 0;
1   int fim = array.length-1;
    int retorno = -1;
2   while(inicio <= fim){
3       int meio = (inicio+fim)/2;

4       if(array[meio] == valor){
5           retorno = meio;
           fim = -1;
6       }else if(valor > array[meio]){
7           inicio = meio+1;
           } else {
8           fim = meio-1;
           }
9   }
10  return retorno;
}

```



$$C = A - N + 2$$

$$12 - 10 + 2$$

$$2 + 2$$

$$4$$

Caminhos
independentes

1.	{1,2,10}	
2.	{1,2,3,4,5,9,2,10}	
3.	{1,2,3,4,6,7,9,2,10}	
4.	{1,2,3,4,6,8,9,2,10}	

Matriz do Grafo (matriz de adjacência)

- 1) Derivar fluxos num grafo até determinar um conjunto de caminhos base. **Compor uma matriz de grafos** para o teste do caminho base.
- 2) Uma **matriz quadrada** é usada e o tamanho é igual à quantidade de arestas no grafo de fluxo, cada linha e coluna da matriz são correspondentes às quantidades de arestas.

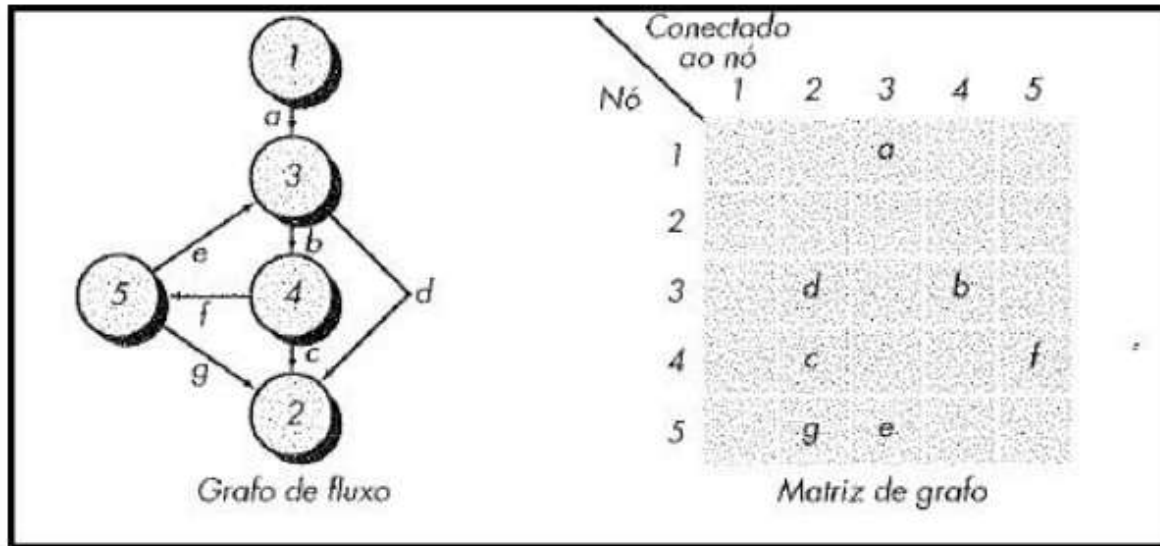
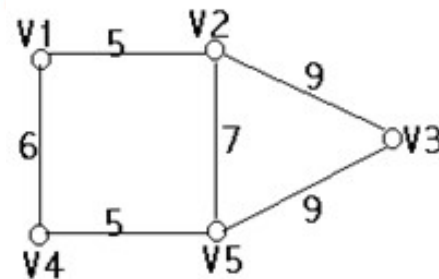


Figura 5 - Representação da Matriz de Grafos (Pressman. 2006)



	V1	V2	V3	V4	V5
V1	0	5	0	6	0
V2	5	0	9	0	7
V3	0	9	0	0	9
V4	6	0	0	0	5
V5	0	7	9	5	0

Automação de Testes

Considerações

59

São ferramentas ou *workbenches* que criam ambiente de testes automatizados

Reduzem tempo e custo de casos de testes

Podem gerar dados de testes

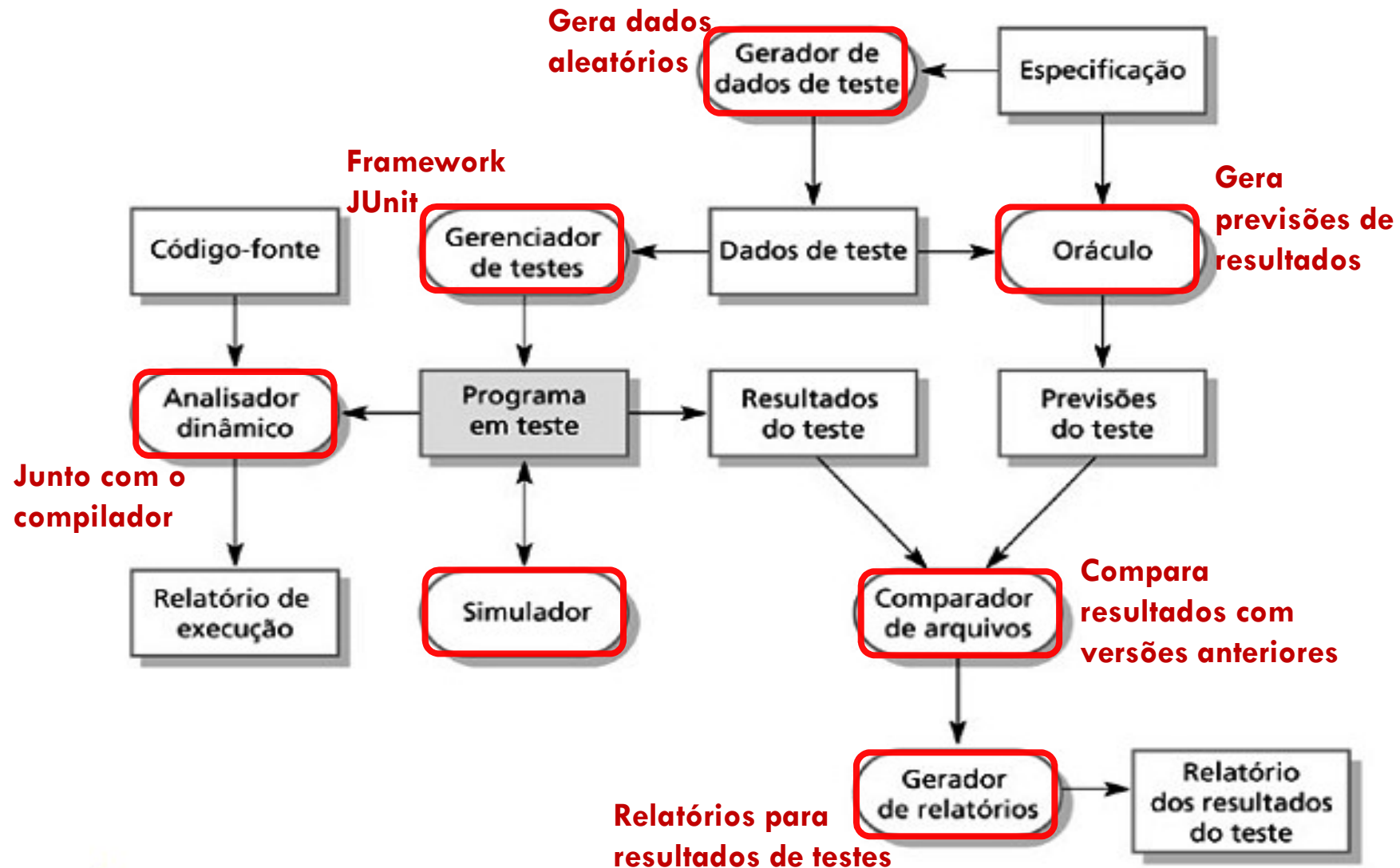
TestCase
<code>run()</code> <code>runTest()</code> <code>setUp()</code> <code>tearDown()</code>

Template Method



Workbench de Teste

60



Recomendações

61

Muito esforço para e tempo para criação de ferramentas

Recomendado para sistemas de grande porte

Recomendado ferramentas CASE para sistemas pequenos

Referências:

62

- ❑ BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivan. UML: guia do usuário. Rio de Janeiro: Campus, 2000..
- ❑ PAULA, Wilson de Pádua Filho. Engenharia de Software, 3ª edição, LTC, 2000.
- ❑ PRESMAN, Roger S. Engenharia de Software. Makron Books, São Paulo, 1995.
- ❑ Andrade, Rosana, Apresentação.*Ph.D, SITE, University of Ottawa, Canadá*, Profa. Departamento de Computação, Centro de Ciências, Universidade Federal do Ceará
- ❑ SOMMERVILLE, Ian. Engenharia de Software. Ed. 8ª. Pearson, São Paulo, 2006