

Testes de caixa branca e depuração

Prof Joel Santos



01

TÉCNICAS PARA CRIAÇÃO DE CASOS DE TESTE

O objetivo do teste é encontrar erros



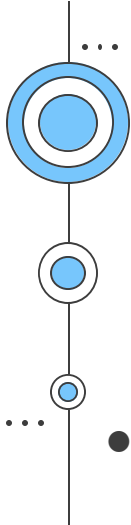
BOM TESTE

Tem alta probabilidade de encontrar um erro

Não é redundante

Em um grupo de testes similares, e com limitações de tempo e recursos, pode-se selecionar os melhores casos de teste

Não deve ser muito simples nem muito complexo

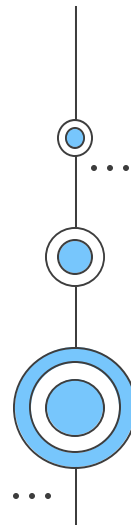


Teste de caixa-branca

Teste estrutural



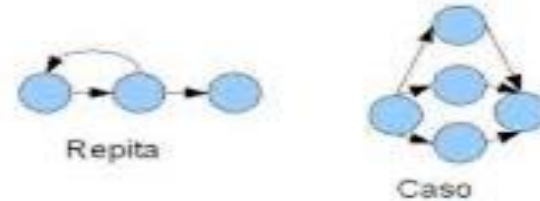
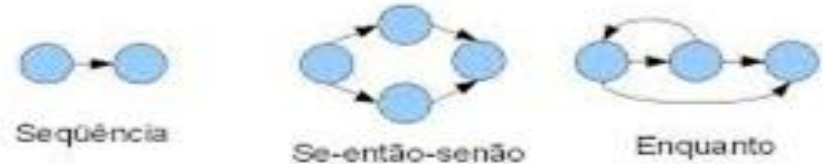
- Garante que todos os caminhos independentes de um módulo sejam executados pelo menos uma vez
- Exercita todas as decisões lógicas nos seus estados verdadeiro e falso
- Executa todos os ciclos em seus limites e dentro de suas fronteiras operacionais
- Exercita estruturas de dados internas para garantir sua validade



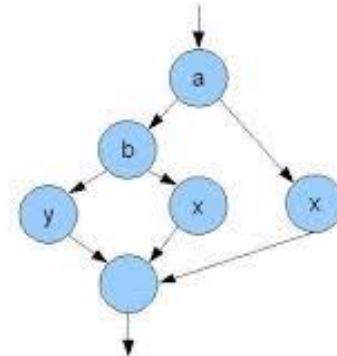
Teste de caixa-branca

Teste do caminho básico

- Deriva-se uma medida da complexidade lógica do projeto
- Usa essa medida para definir caminhos base de execução
 - Se executar todos os caminhos base, com certeza todas as instruções são executadas pelo menos uma vez



```
...  
se a ou b então  
    procedimento  
x  
senão  
    procedimento  
y  
fimse  
...
```



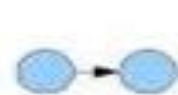


Teste de caixa-branca

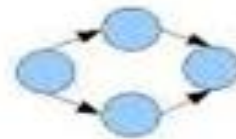
Teste do caminho básico



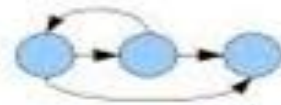
- **Aresta:** setas
- **Nó:** círculos
 - Pode juntar mais de uma instrução simples em um único nó
- **Região**
 - Área delimitada por arestas e nós
 - Contamos também a região externa
- **Nó predicado**
 - Nó contendo uma condição



Sequência



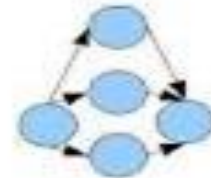
Se-então-senão



Enquanto

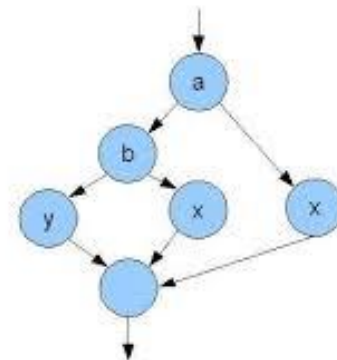


Repita



Caso

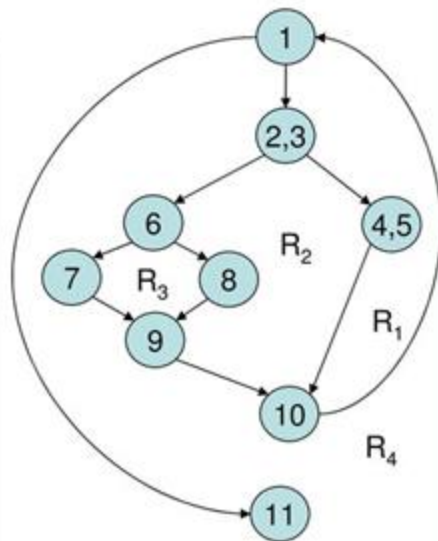
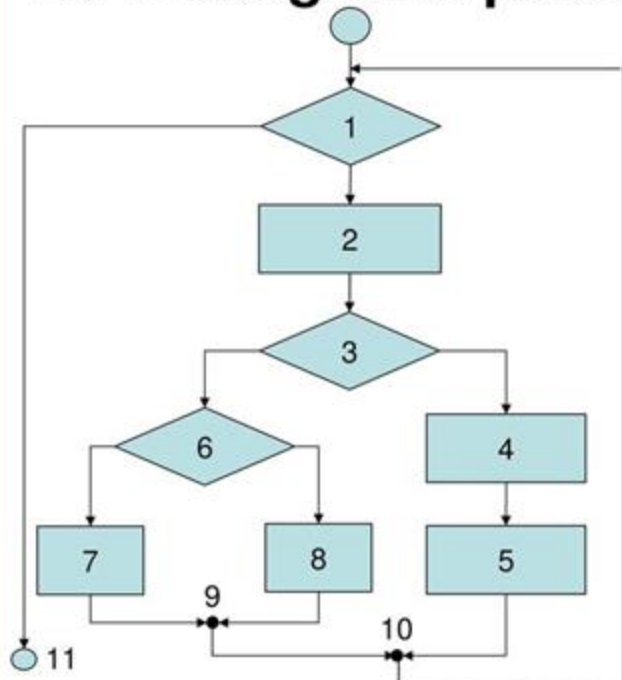
```
...  
se a ou b então  
    procedimento  
x  
senão  
    procedimento  
y  
fimse  
...
```



Teste de caixa-branca

Teste do caminho básico

De Fluxograma para Grafo de Fluxo



Áreas limitadas por arestas e nós são chamadas de **regiões**.



Teste do caminho básico

Caminhos de programa independentes

- Caminhos que introduzem um novo conjunto de comandos ou uma nova condição
- Deve incluir pelo menos uma aresta que não tenha sido visitada



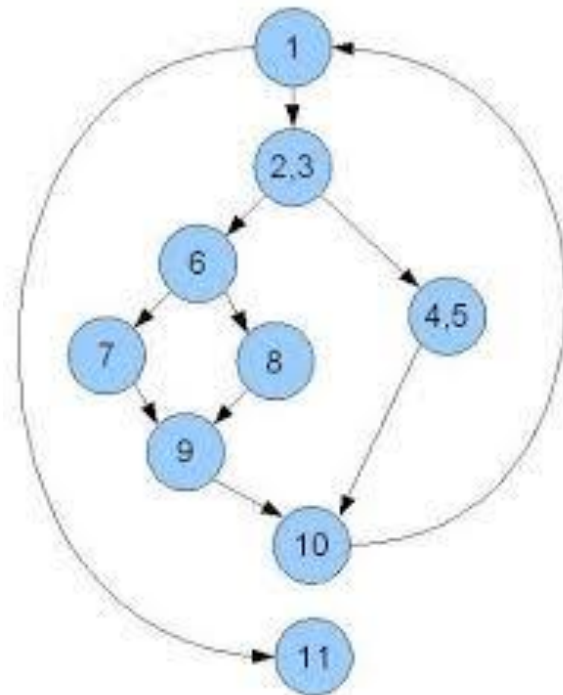
1-11

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

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

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

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



Teste do caminho básico

Caminhos de programa independentes

Se os testes forcarem a execução destes caminhos (**conjunto base**), cada comando do programa terá sido executado pelo menos uma vez, e cada condição terá sido executada em seus lados V e F

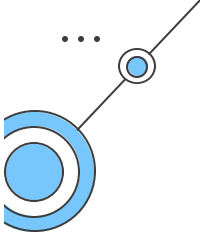
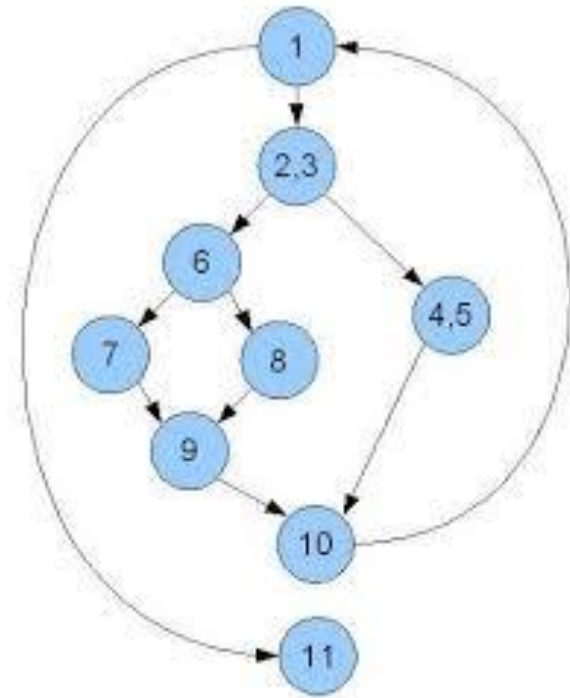
1-11

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

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

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

Como saber se tem mais algum?



Complexidade ciclomática

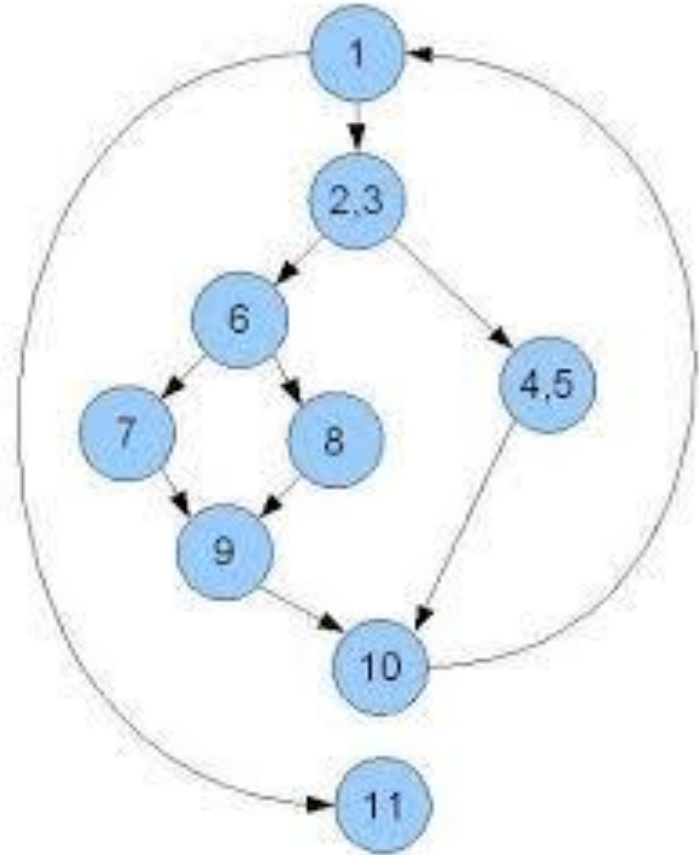
1. $V(G) = \text{número de regiões do grafo}$
2. $V(G) = E - N + 2$
E: número de arestas
N: número de nós
3. $V(G) = P + 1$
P: número de nós predicados

$$V(G)=4$$

$$V(G)=11-9+2=4$$

$$V(G)=3+1=4$$

Indica um limite superior para o número de casos de teste que devem ser projetados



Exemplo 1

Procedimento média(valor[])

i=1;

soma=0;

total.entrada=0;

total.válidas=0

Faça-Enquanto (valor[i]≠-999 **E** total.entrada<100)

 incremente total.entrada de 1;

Se (valor[i]≥min **E** valor[i]≤max)

Então

 incremente total.válidas de 1;
 soma = soma + valor[i]

Fim-Se

Se incremente i de 1;

Fim-Enquanto

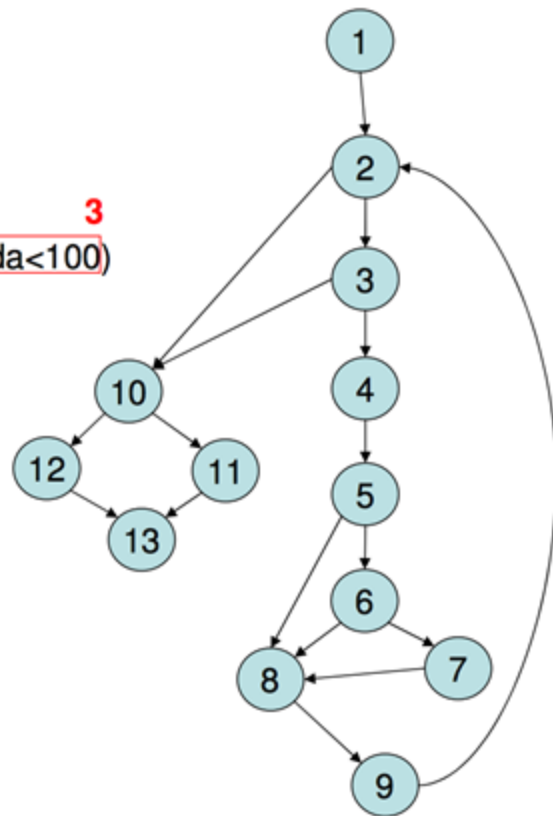
Se total.válidas>0

Então média = soma/total.válidas;

Senão média = -999;

Fim-Se

Fim média



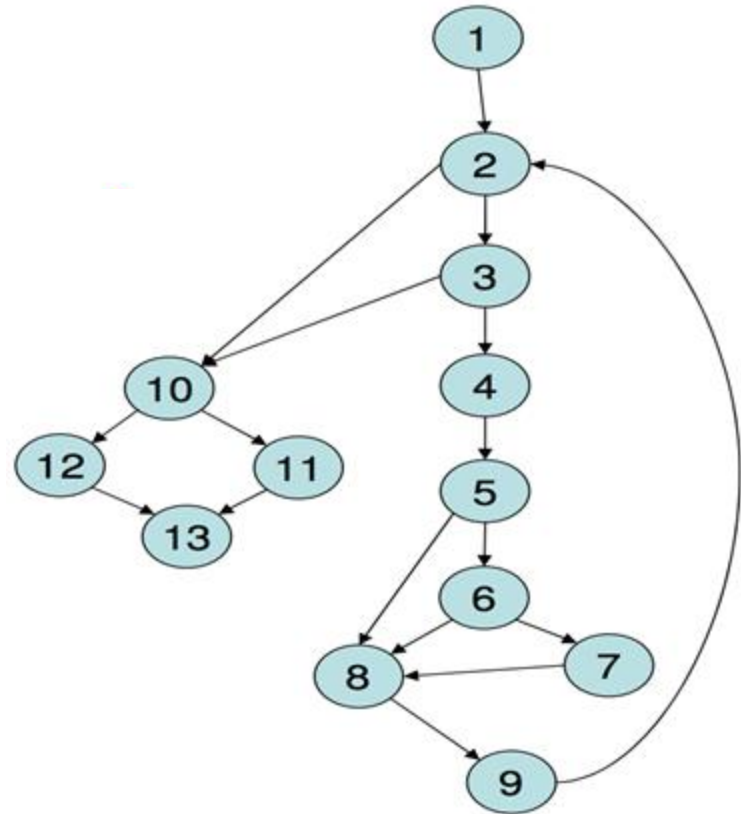
Exemplo 1

1. $V(G)$ = número de regiões do grafo
2. $V(G) = E - N + 2$
 - E : número de arestas
 - N : número de nós
3. $V(G) = P + 1$
 - P : número de nós predicados

$$V(G)=6$$

$$V(G)=17-13+2=6$$

$$V(G)=5+1=6$$



Exemplo 1

$V(G)=6$

Caminhos

1-2-10-11-13

1-2-10-12-13

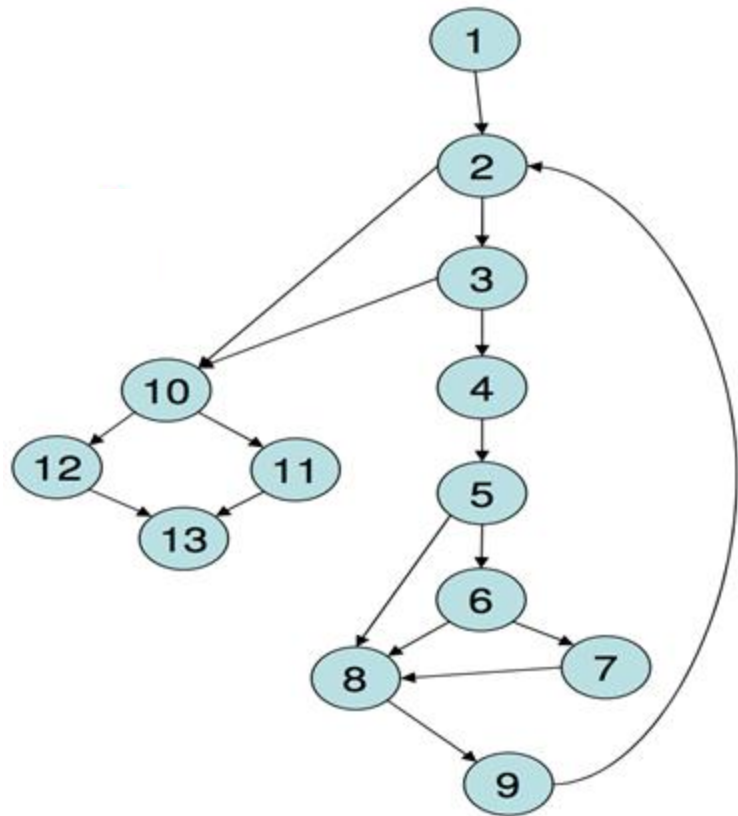
1-2-3-10-11-13

1-2-3-4-5-8-9-2-...

1-2-3-4-5-6-8-9-2-...

1-2-3-4-5-6-7-8-9-2-...

Prepare casos de teste que executem estes caminhos



Exemplo 1

Procedimento média(valor[])

i=1;

soma=0;

total.entrada=0;

total.válidas=0

Faça-Enquanto (valor[i]≠-999 **E** total.entrada<100)

 incremente total.entrada de 1;

Se (valor[i]≥min **E** valor[i]≤max)

Então

 incremente total.válidas de 1;
 soma = soma + valor[i]

Fim-Se

 incremente i de 1;

Fim-Enquanto

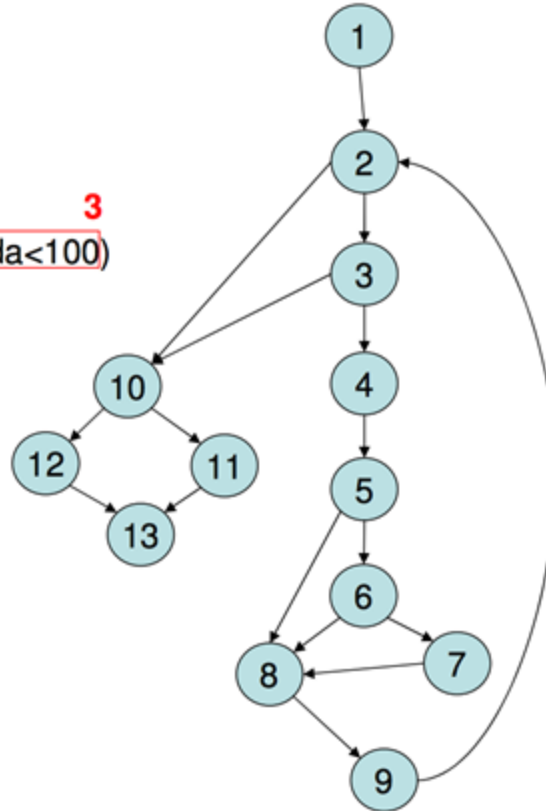
Se total.válidas>0

Então média = soma/total.válidas;

Senão média = -999;

Fim-Se

Fim média



Caminhos

1-2-10-11-13

1-2-10-12-13

1-2-3-10-11-13

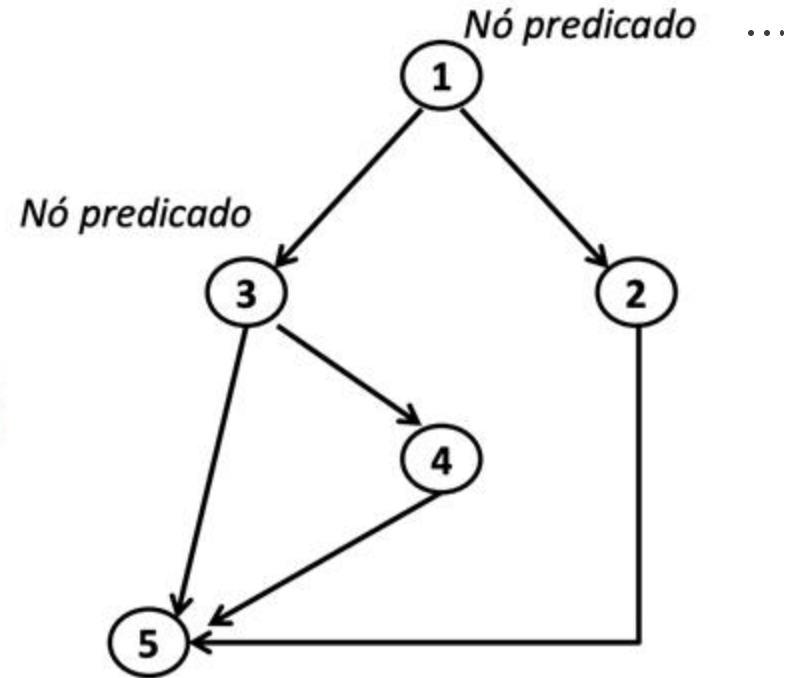
1-2-3-4-5-8-9-2-...

1-2-3-4-5-6-8-9-2-...

1-2-3-4-5-6-7-8-9-2-...

Exemplo 2

```
17 public boolean sacar(double valorSaque) {  
18     boolean status=false; ①  
19     if (valorSaque <= saldo) {  
20         saldo -= valorSaque; ②  
21         status= true;  
22     } else { ③  
23         if (valorSaque <= limiteDeCredito) {  
24             limiteDeCredito -= valorSaque; ④  
25             status= true;  
26         }  
27     }  
28  
29     return status; ⑤  
30 }
```



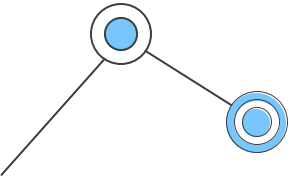
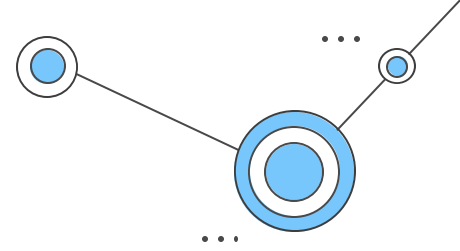
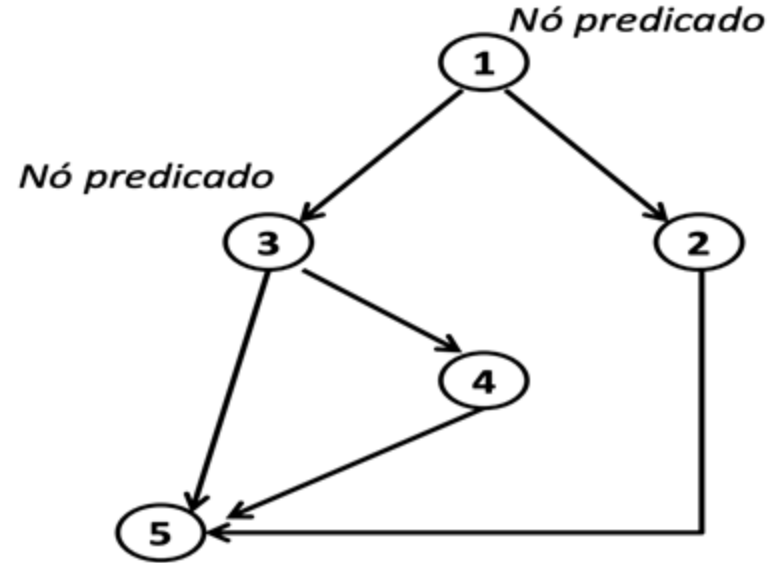
Exemplo 2

1. $V(G)$ = número de regiões do grafo
2. $V(G) = E - N + 2$
 - E: número de arestas
 - N: número de nós
3. $V(G) = P + 1$
 - P: número de nós predcados

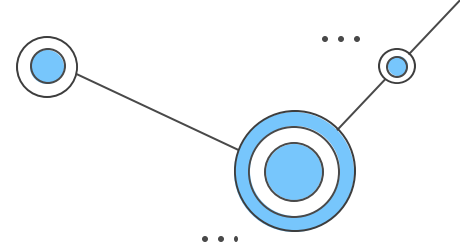
$$V(G)=3$$

$$V(G)=6-5+2=3$$

$$V(G)=2+1=3$$



Exemplo 2



$V(G)=3$

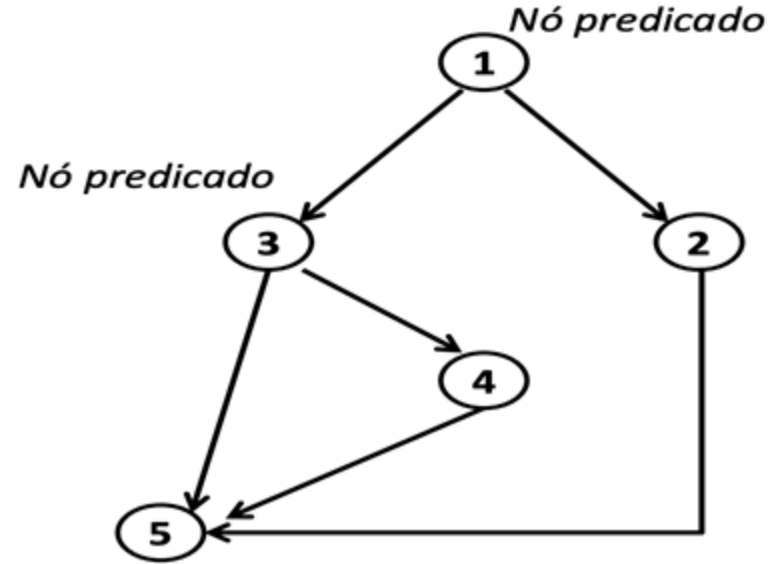
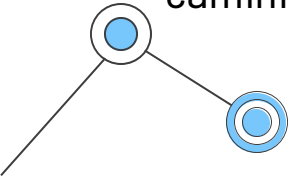
Caminhos

1-2-5

1-3-5

1-3-4-5

Prepare casos de teste que executem estes caminhos



Exemplo 2

```
17 public boolean sacar(double valorSaque) {  
18     boolean status=false;  
19     if (valorSaque <= saldo) ①  
20         saldo -= valorSaque;  
21         status= true; ②  
22     } else {  
23         if (valorSaque <= limiteDeCredito) ③ {  
24             limiteDeCredito -= valorSaque; ④  
25             status= true;  
26         }  
27     }  
28  
29     return status; ⑤  
30 }
```

Caminho 1: 1-2-5

T1: valorSaque <= saldo

Saldo = 100

Valor Saque = 99, 100

Caminho 2: 1-3-5

T2: valorSaque > saldo

valorSaque > limiteDeCredito

Saldo = 100

Crédito = 100

Valor Saque = 101

Caminho 3: 1-3-4-5

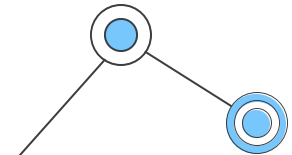
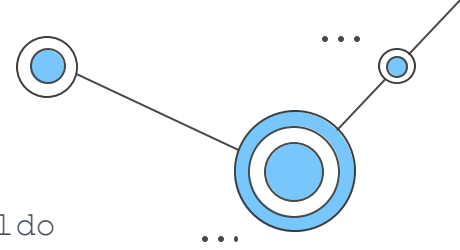
T3: valorSaque > saldo

valorSaque <= limiteDeCredito

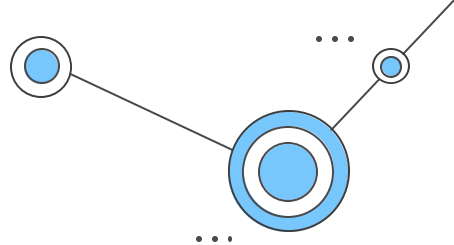
Saldo = 100

Crédito = 110

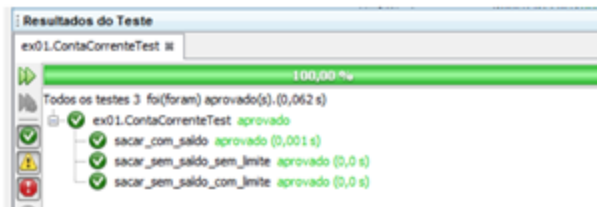
Valor Saque = 110



Exemplo 2



```
17 public class ContaCorrenteTest {
18     private ContaCorrente cc;
19
20     @Before
21     public void inicializa() {
22         cc = new ContaCorrente();
23     }
24
25     @Test
26     public void sacar_com_saldo() {
27         cc.setSaldo(100);
28         assertTrue(cc.sacar(100));
29     }
30
31     @Test
32     public void sacar_sem_saldo_com_limite() {
33         cc.setSaldo(100);
34         cc.setLimiteDeCredito(110);
35         assertTrue(cc.sacar(110));
36     }
37
38     @Test
39     public void sacar_sem_saldo_sem_limite() {
40         cc.setSaldo(100);
41         cc.setLimiteDeCredito(100);
42         assertFalse(cc.sacar(101));
43     }
44 }
45 }
```



Teste de caixa-preta

Teste comportamental ou teste funcional



Focaliza nos requisitos funcionais do software

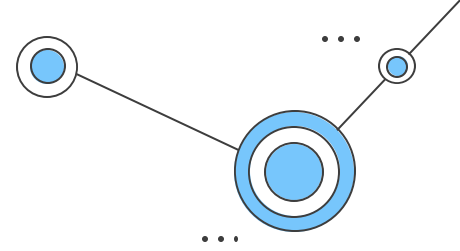
Busca utilizar completamente todos os requisitos funcionais para um programa

- a. Identifica funções incorretas ou ausentes
- b. Identifica erros de interface
- c. Identifica erros em estruturas de dados ou acesso a bases de dados externas
- d. Identifica erros de comportamento ou de desempenho
- e. Identifica erros de inicialização e término

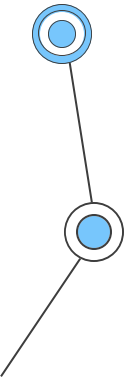


Teste de caixa-preta

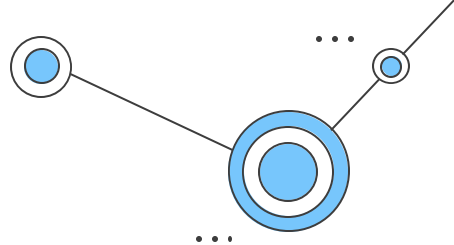
Questões que devem ser respondidas



1. Como a validade funcional é testada?
2. Como o comportamento e o desempenho do sistema são testados?
3. Que classes de entrada farão bons casos de teste?
4. O sistema é particularmente sensível a certos valores de entrada?
5. Que taxas e volumes de dados o sistema pode tolerar?



Comparação



TESTE CAIXA-BRANCA

Focaliza nas estruturas de controle do sistema

Tende a ser aplicado antecipadamente no processo de testes

Testes de unidade

TESTE CAIXA-PRETA

Focaliza nos requisitos funcionais

Tende a ser aplicado em estágios posteriores de teste

Testes de integração, sistema, alfa e beta



02

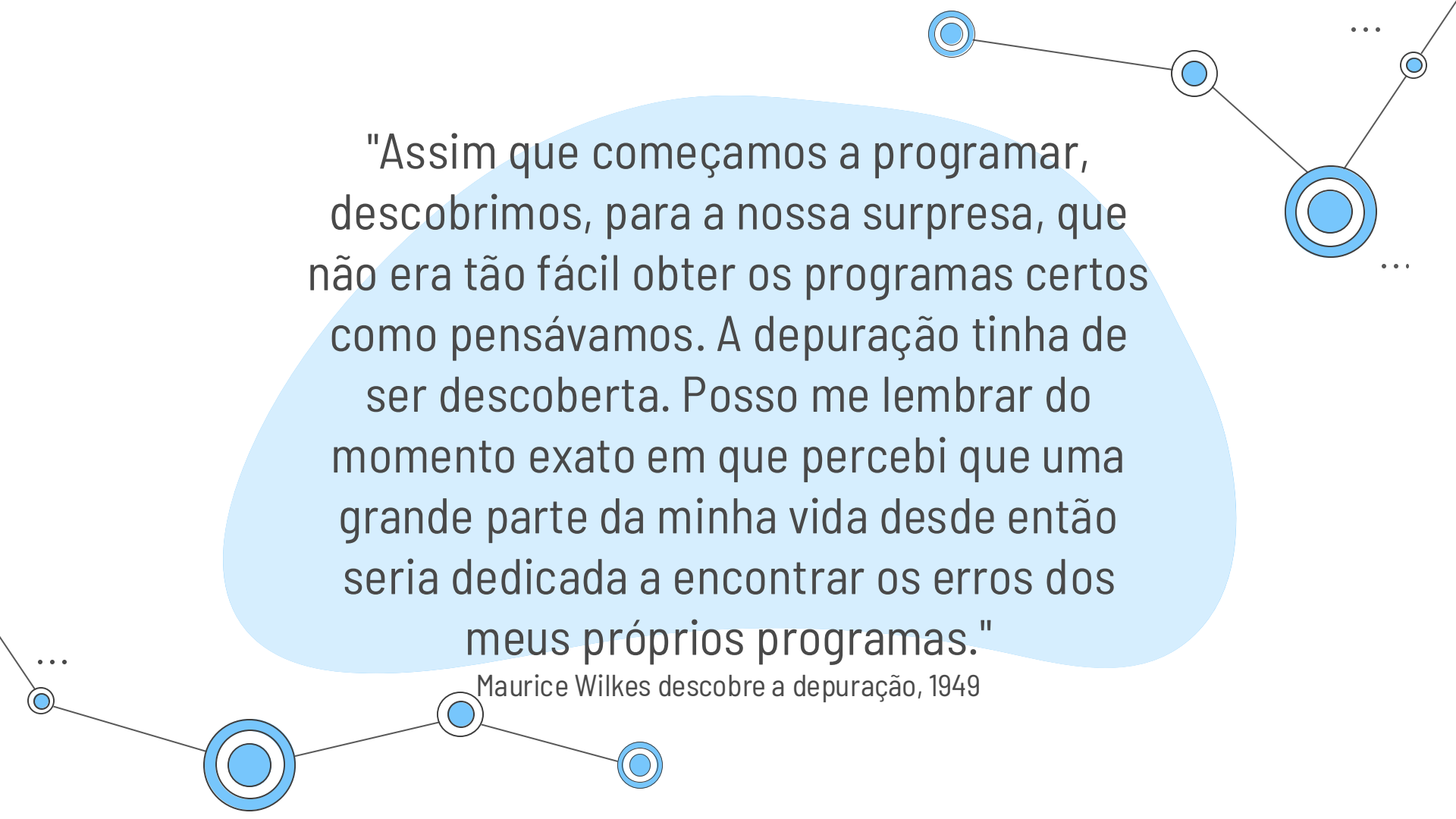
A arte da
Depuração



O que fazer quando um processo de testes encontra um erro?

DEPURAÇÃO



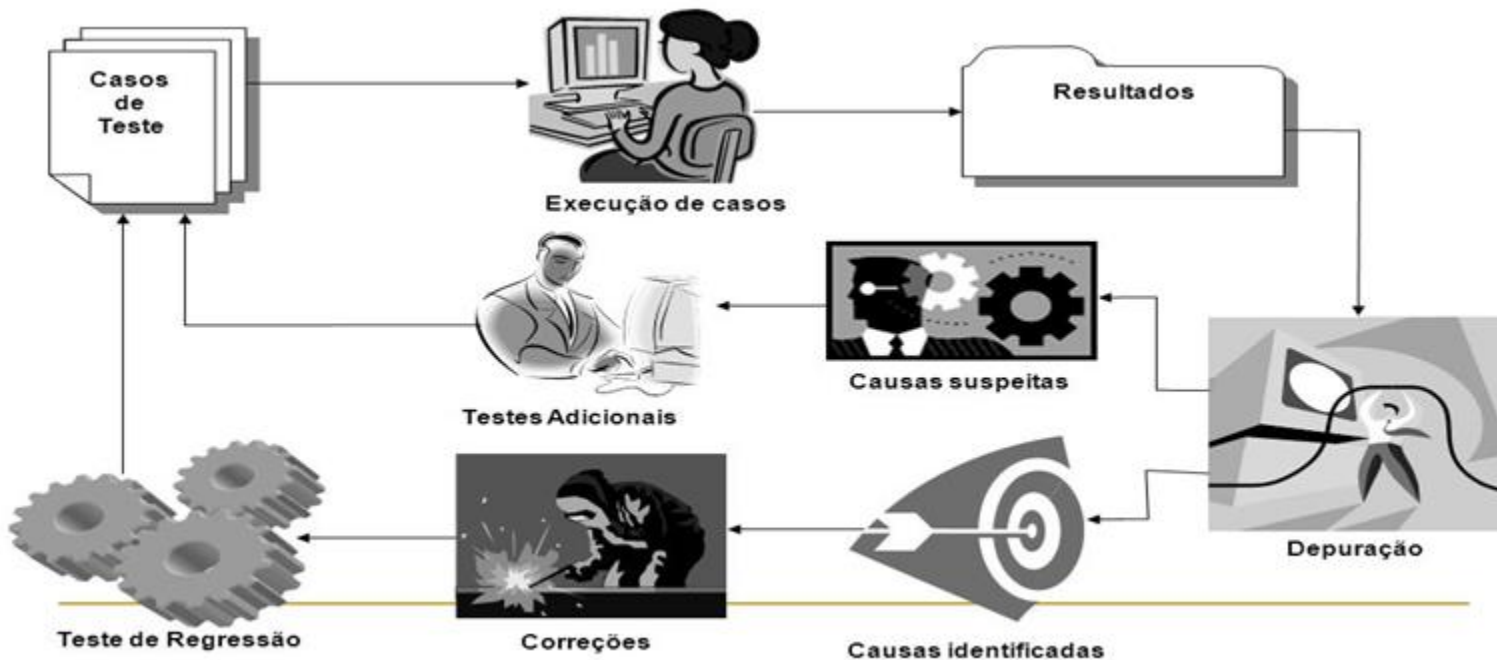


"Assim que começamos a programar, descobrimos, para a nossa surpresa, que não era tão fácil obter os programas certos como pensávamos. A depuração tinha de ser descoberta. Posso me lembrar do momento exato em que percebi que uma grande parte da minha vida desde então seria dedicada a encontrar os erros dos meus próprios programas."

Maurice Wilkes descobre a depuração, 1949

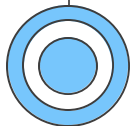
O processo de depuração

Não é teste, mas ocorre em consequência do teste
Começa com a execução de um caso de teste



Processo de depuração tenta combinar o sintoma com a causa

O que acontece quando se tem **causas suspeitas?**



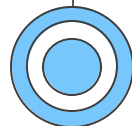
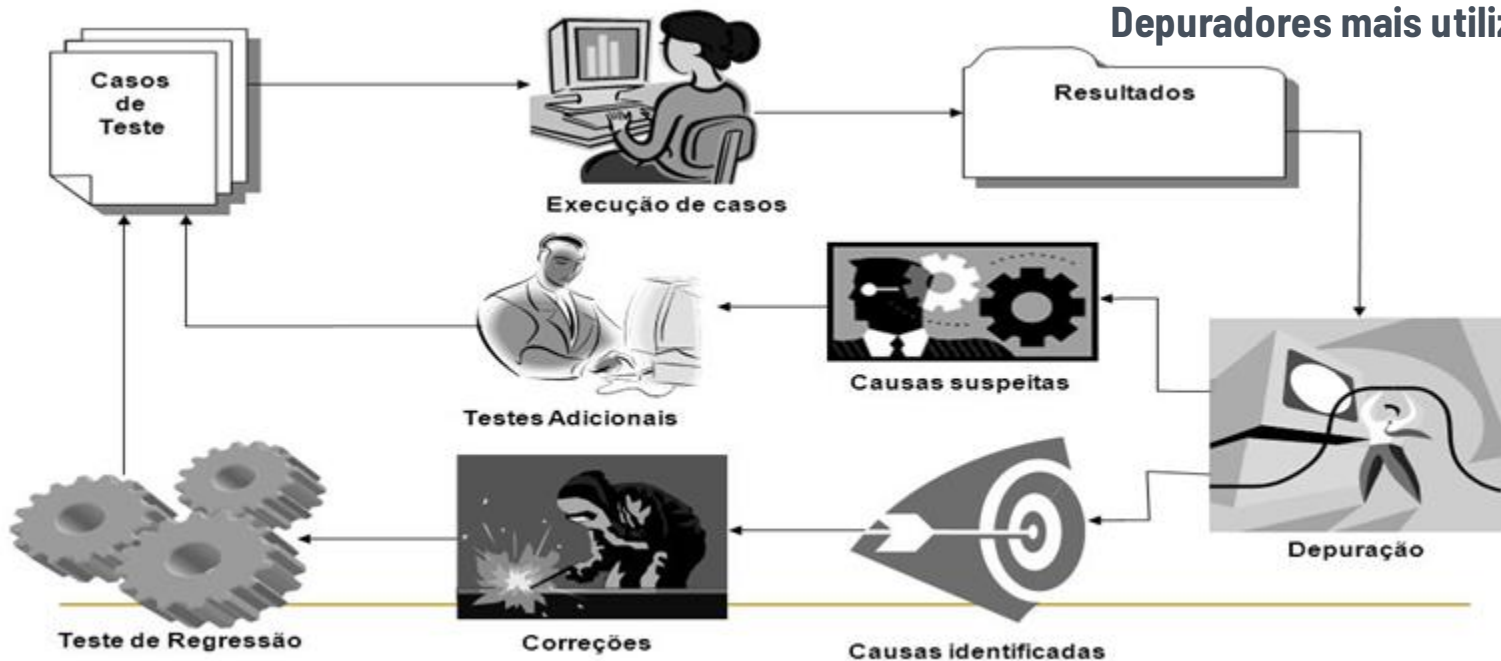
O processo de depuração

Causas suspeitas:

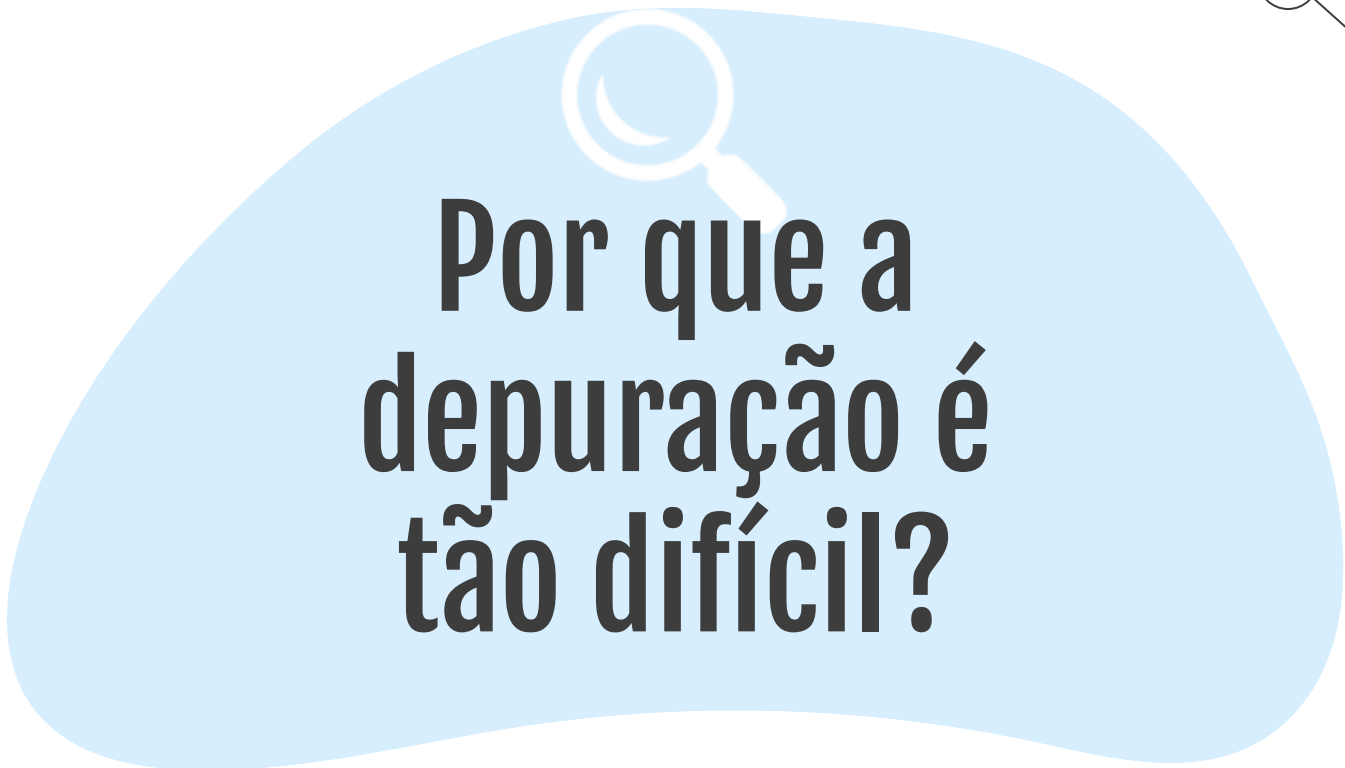
Criação de novos casos de testes para confirmar a suspeita

Correção do erro acontecerá de forma iterativa, quando a causa for confirmada

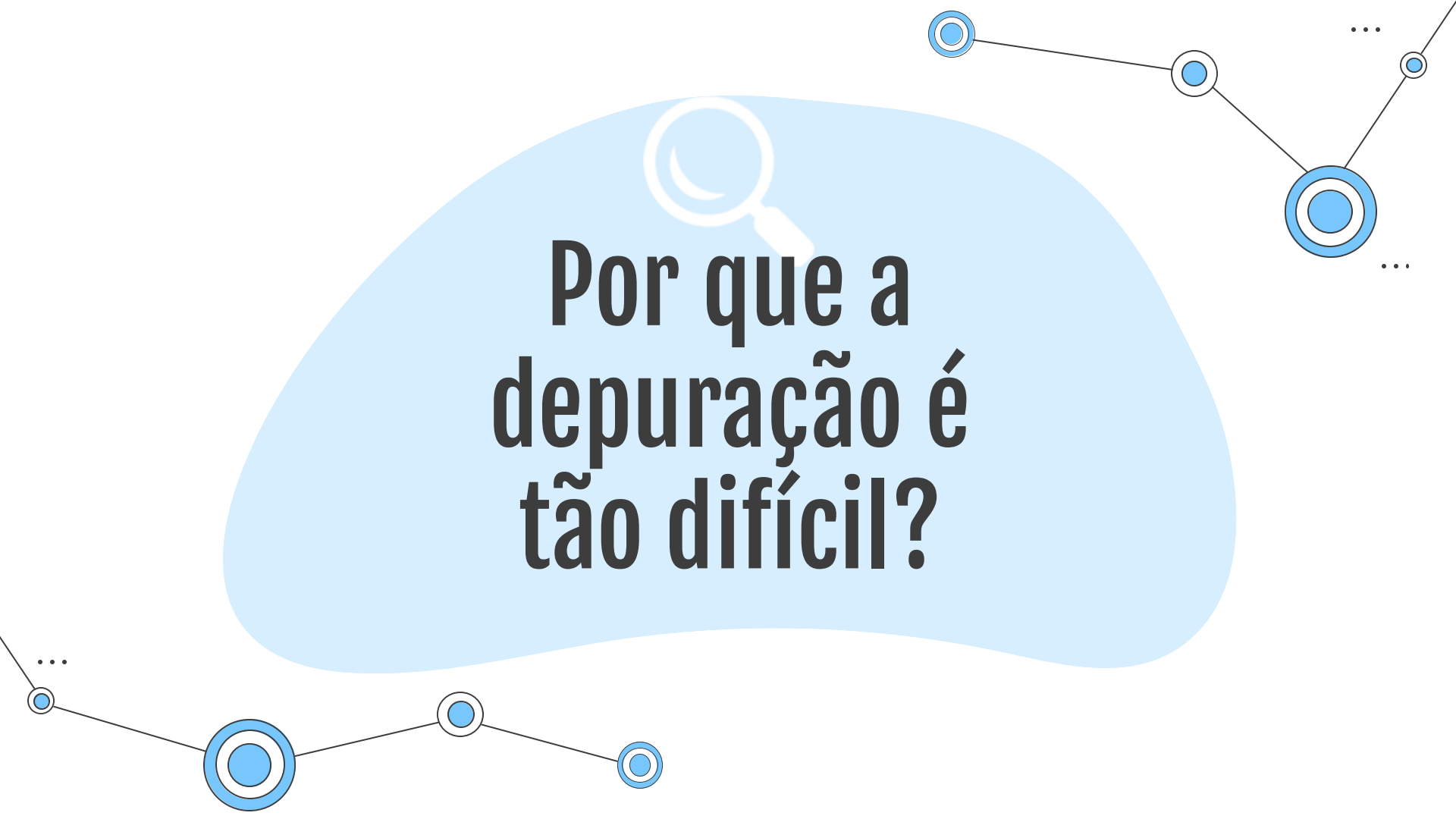
Depuradores mais utilizados: IDEs



...



**Por que a
depuração é
tão difícil?**



O sintoma e a causa podem estar localizados em locais muito diferentes do sistema

O sintoma pode ser causado por erro humano que não é facilmente rastreado

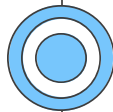
O sintoma pode ser causado por coisa que não são erros (por exemplo, imprecisões de arredondamento)

O sintoma pode desaparecer (temporariamente) quando outro erro for corrigido

Pode ser difícil reproduzir as condições de entrada com precisão (por exemplo, uma aplicação em tempo real)

O sintoma pode ser intermitente (mais comum em sistemas integrados com hardware)

Dificuldade de Depurar



Problema na depuração



- Erros podem ser muito simples (formato de saída incorreto)
- Erros podem ser catastróficos (o sistema falha, causando perdas econômicas)
- À medida que as consequências de um erro aumentam, a pressão para encontrá-lo também aumenta
- Às vezes, a pressão obriga o desenvolvedor a corrigir um erro e introduzir outros dois
- **Defina um limite (2 horas, por exemplo) para depurar um problema sozinho. Depois desse tempo, procure ajuda!**



Estratégias de depuração



Força Bruta

Método mais comum e menos eficiente

Deve ser usado quando todo o resto falha

O erro não é analisado para descobrir o programa

Espera-se que o próprio computador a descubra

Exemplo: inserir vários comandos de escrita no programa

Rastreamento (*backtracking*)

Usada com sucesso em programas pequenos

O código-fonte é investigado retroativamente, a partir do ponto onde o erro foi encontrado, de forma manual.

Pode ser um trabalho muito grande, conforme o número de caminhos retroativos cresce

Eliminação da causa

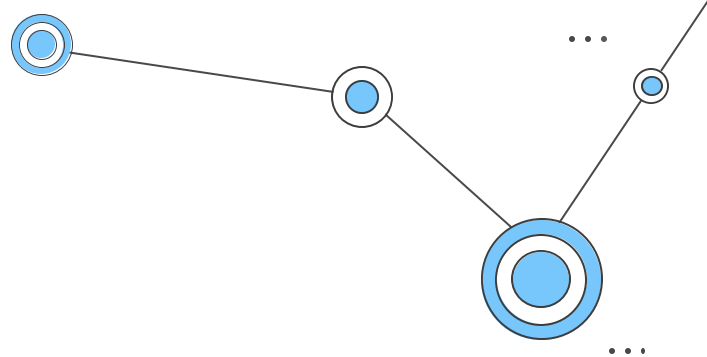
Os dados relacionados à ocorrência do erro são organizados para isolar as possíveis causas

É criada uma "hipótese da causa"

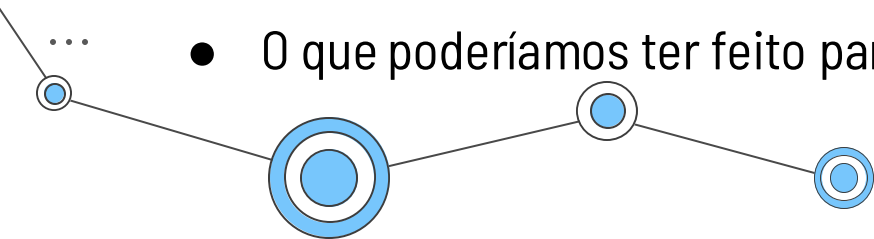
É feita uma lista de possíveis causas, e são criados testes para comprová-las ou eliminá-las

Correção do erro

O que analisar antes de começar?



- A causa do defeito é reproduzida em alguma outra parte do programa?
 - Pode iniciar outros erros
- Qual próximo erro poderia ser introduzido pela correção que estou prestes a fazer?
 - Partes altamente acopladas podem introduzir novos erros quando forem alteradas
- O que poderíamos ter feito para evitar esse defeito já no início?



PRÁTICA

