



Executar Teste e Implantação de Aplicativos Computacionais

SENAC PE

18 de Outubro de 2024

Teste de Mutação

- Contexto Geral



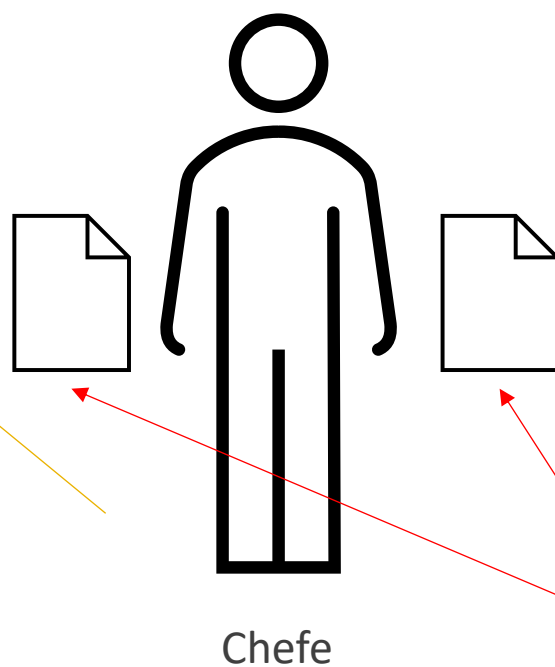
Teste de Mutação

Conceito

- O teste de mutação é um **critério baseado defeitos** ou seja, utiliza conhecimento sobre defeitos típicos que podem ocorrer ao escrevermos programa para nos ajudar a criar bons conjuntos de testes.

Teste de Mutação

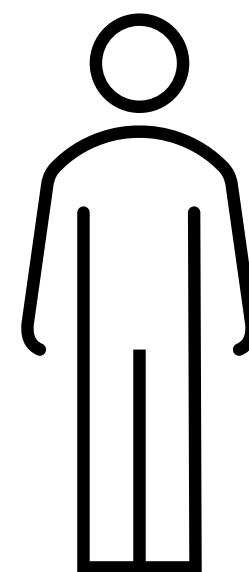
Imagine o Seguinte Cenário



Chefe

Além do desenvolvimento
ele quer ver também como
você testou o programa
para se certificar de que
funciona

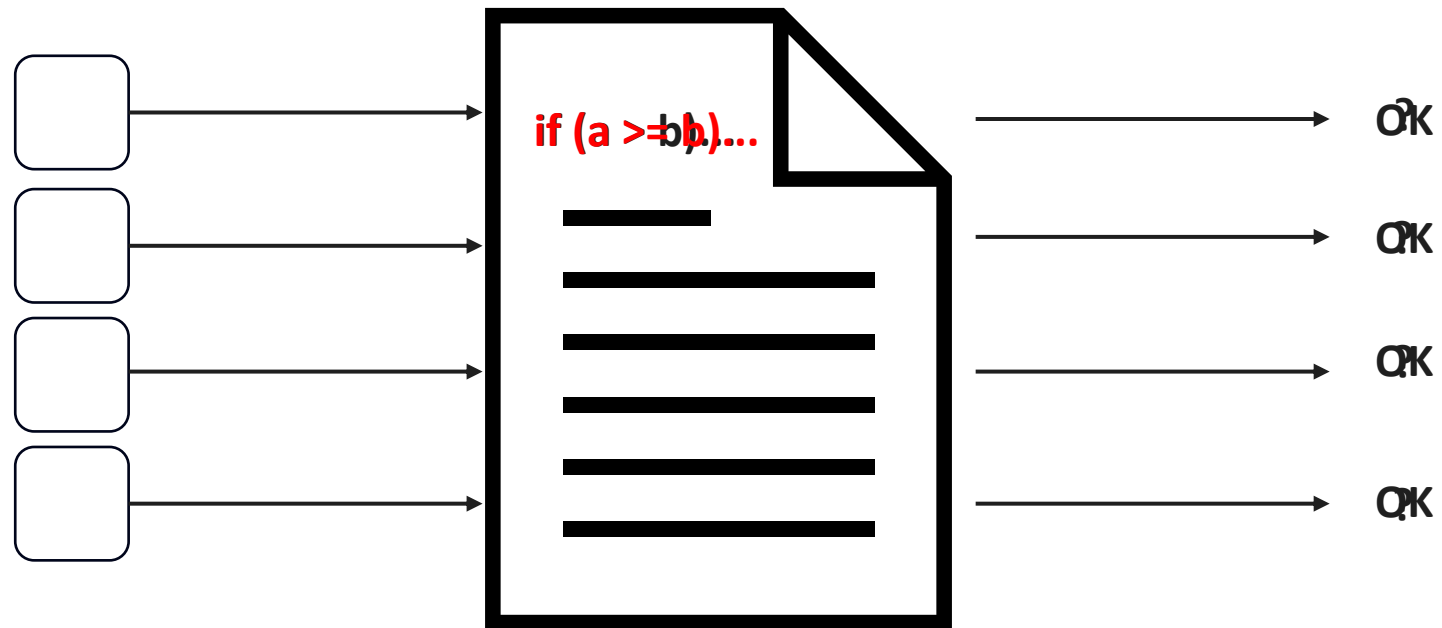
Conjunto de
atividades de
desenvolvimento



Dev. Novato

Teste de Mutação

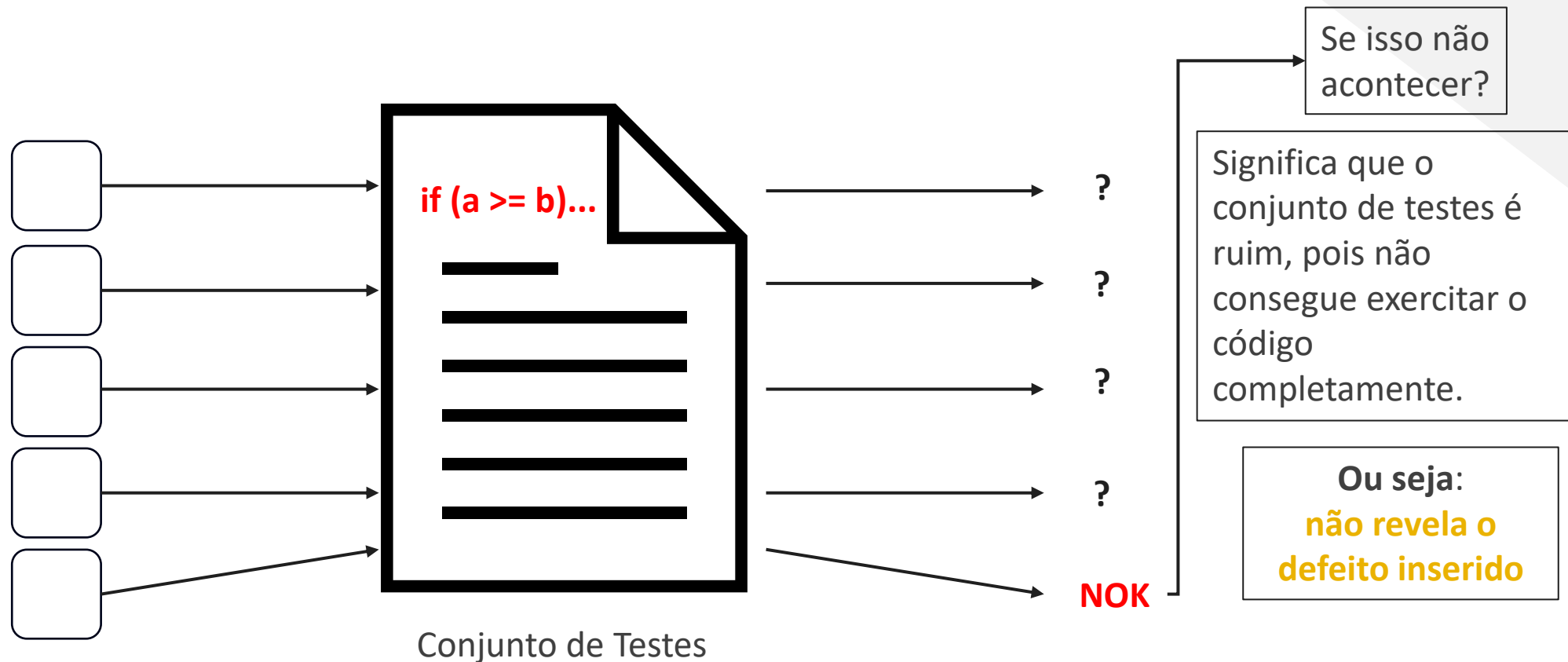
Imagine o Seguinte Cenário



Conjunto de Testes

Teste de Mutação

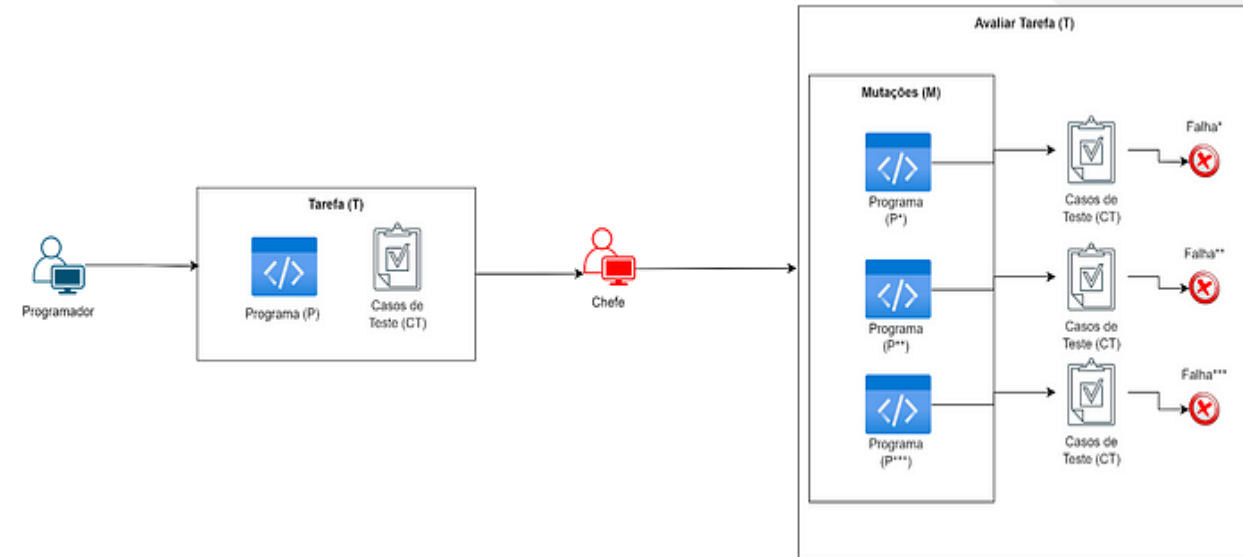
Imagine o Seguinte Cenário



Teste de Mutação

E isso é o teste de mutação

- Mas vez de fazermos isso uma única vez, fazemos várias vezes.
- Criamos vários programas que chamamos de **mutantes** ou seja, que possuem pequenas modificações relação ao programa sendo testado.
- Os casos de teste têm que mostrar que o seu programa apresenta sempre resultado correto, e que cada dos mutantes alguma vez apresenta resultado incorreto.



Teste de Mutação

Ou seja

- Quando o caso de teste consegue distinguir o comportamento de mutante do programa original dizemos que esse mutante está morto.
- Portanto, o **objetivo** que a gente tem com o teste de mutação é **achar conjuntos de teste que matem todos os mutantes** que foram criados.

Teste de Mutação

Como avaliar os casos de teste?

- Vamos “contar” quantos mutantes conseguimos matar
- Ex:
 - Total de mutantes: 110
 - Mutantes mortos com conjunto de testes: 60

- $$\text{Escore de mutação} = \frac{\text{Mutantes mortos}}{\text{Total de mutantes}} = \frac{60}{110} = 0,55$$

Teste de Mutação

Como são gerados os mutantes?

- Baseados em **Operadores de mutação** são regras que definem quais alterações devem ser feitas para que os mutantes sejam criados.
- Esses operadores estão fortemente ligados com a linguagem na qual o programa que está sendo testado foi escrito. Eles buscam explorar todas as características e estruturas da linguagem.

Operadores aritméticos	+	Adição
	-	Subtração
	*	Multiplicação
	/	Divisão
	//	Divisão Inteira
	%	Módulo
	**	Exponenciação
Operadores de atribuição	=	
	+=	$x = x + a$
	-=	$x = x - a$
	*=	$x = x * a$
	/=	$x = x / a$
	%=	$x = x \% a$
Operadores de comparação	>	Maior
	<	menor
	>=	Maior igual
	<=	Menor Igual
	!=	Diferente
	==	Igual
Operadores lógicos	and	e
	or	ou
	not	negação
Operadores de identidade	is	é
	is not	não é
Operadores de associação	in	interno
	not in	não interno

Operadores lógicos no python

Teste de Mutação

Ou seja

- Quando o caso de teste consegue distinguir o comportamento de mutante do programa original dizemos que esse mutante está morto.
- Portanto, o **objetivo** que a gente tem com o teste de mutação é **achar conjuntos de teste que matem todos os mutantes** que foram criados.


Exemplo

- ORRN

Troca de Operador Relacional - ORRN

Vamos ao Exemplo do código a seguir

- Trocar o operador em destaque por outros cinco operadores relacionais para criar os mutantes:
 - `>=`
 - `<`
 - `<=`
 - `==`
 - `!=`



```
def bubble_sort(x):
    n = len(x)
    for last in range(n-1, 0, -1):
        for i in range(last):
            if x[i] > x[i+1]:
                x[i], x[i+1] = x[i+1], x[i]
```

Troca de Operador Relacional - ORRN

Entrada: [1,2,3,4,5]

Original

```
def bubble_sort(x):
    n = len(x)
    for last in range(n-1, 0, -1):
        for i in range(last):
            if x[i] > x[i+1]:
                x[i], x[i+1] = x[i+1], x[i]
```

Saída: [1,2,3,4,5]

Mutante

```
def bubble_sort(x):
    n = len(x)
    for last in range(n-1, 0, -1):
        for i in range(last):
            if x[i] != x[i+1]:
                x[i], x[i+1] = x[i+1], x[i]
```

Saída: [5,4,3,2,1]

**Saída diferentes: mutante morto.
Só repetir o caso e verificar a saída
com os outros operadores.**

Mutante Equivalente

Nem sempre é fácil verificar a diferença...

Entrada: [1,2,3,4,5]

Original

```
def bubble_sort(x):
    n = len(x)
    for last in range(n-1, 0, -1):
        for i in range(last):
            if x[i] > x[i+1]:
                x[i], x[i+1] = x[i+1], x[i]
```

Saida: [1,2,3,4,5]

Mutante

```
def bubble_sort(x):
    n = len(x)
    for last in range(n-1, 0, -1):
        for i in range(last):
            if x[i] >= x[i+1]:
                x[i], x[i+1] = x[i+1], x[i]
```

Saida: [1,2,3,4,5]

Mutante Equivalente

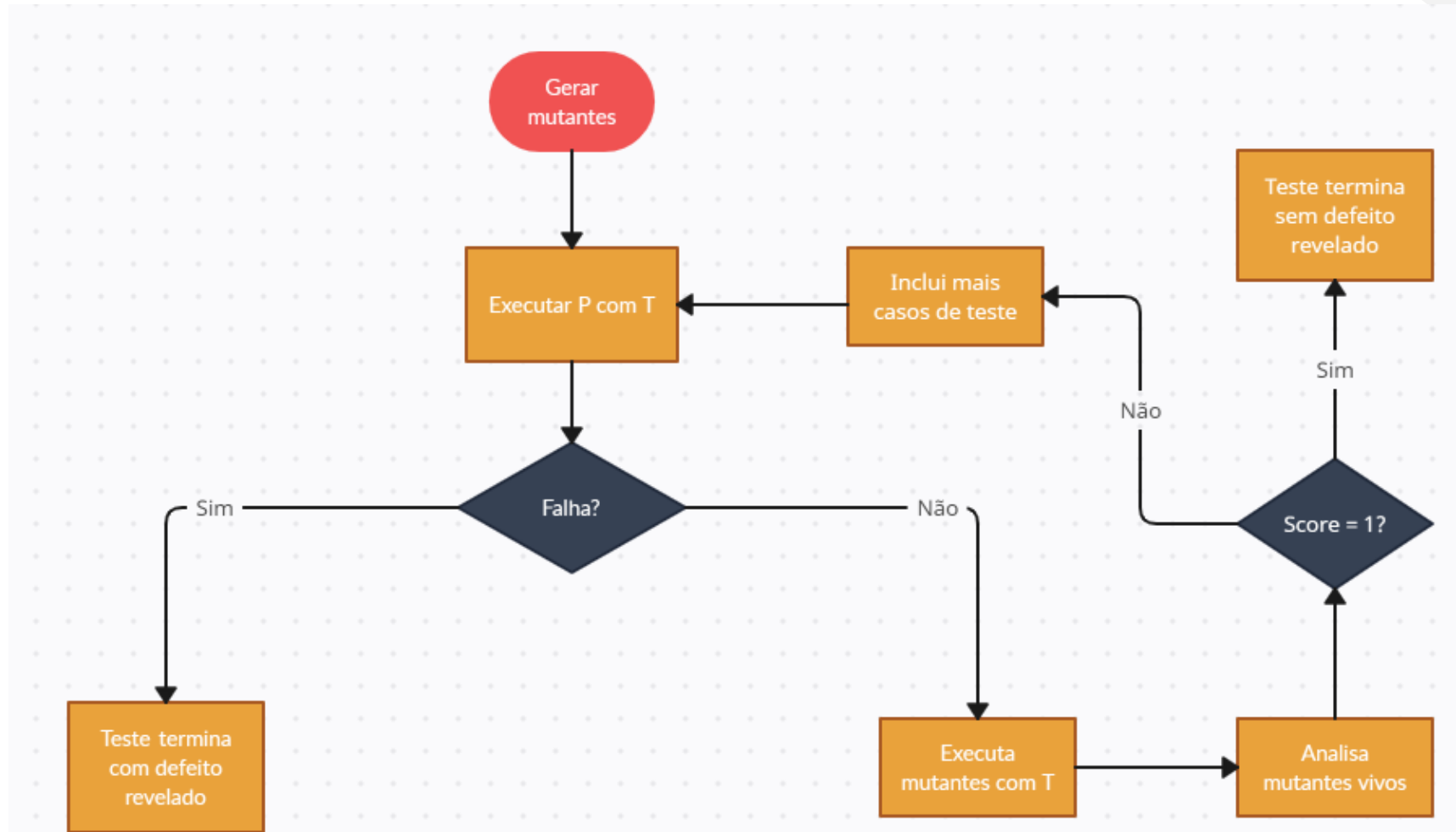
Mutante que nunca é “morto”
com nenhuma entrada dada

Mutante Equivalente

Como avaliar os casos de teste?

- Vamos “contar” quantos mutantes conseguimos matar
- Ex:
 - Total de mutantes: 110
 - Mutantes mortos com conjunto de testes: 60
 - Mutantes equivalentes: 10
- $$\text{Escore de mutação} = \frac{\text{Mutantes mortos}}{\text{Total} - \text{Equivalentes}} = \frac{60}{100} = 0,60$$

Resumo



Exercícios

- Teste de Mutação

Exercício

- Verifique o programa calc.py:
 - a) Crie um conjunto de testes (baseado em outros critérios estudados)
 - b) Crie 5 programas mutantes
 - c) Teste o programa de acordo com o resumo da aula
 - d) Documente os resultados (casos de teste)



Senac Pernambuco Educação Profissional Recife

Thiago Dias Nogueira

Instrutor Técnico

(81) 9 9627-0419

thiago.nogueira@pe.senac.br