Introdução ao Teste de Software

Teste Guiado por Modelos
(*Model-driven Test Design*)

Vinicius H. S. Durelli
✉ durelli@ufsj.edu.br

Organização

Recapitulando...

> “ Teste de software é capaz de mostrar a presença de falhas
>
> não a ausência. ”

A complexidade envolvida em testar software...

Nenhum outro campo da engenharia cria produtos tão complexos quanto software (Ammann & Offutt 2016).

O termo "corretude" é difícil de estabelecer/definir precisamente.

- É possível dizer que um *prédio* é "correto"?
- E um *carro*? Algum carro é "correto"?
- E um *sistema de metrô*?

Assim como em outras engenharias, teste de software envolve **usar abstrações para mitigar** a complexidade.

- Teste guiado por modelos (*model-driven test design*, MDTD)

O domínio de entrada dos programas...

O **domínio de entrada** de um programa P, denotado $\mathcal{D}(P)$ pode ser definido como a seguir:

Definição → Domínio de Entrada

Conjunto de todos os possíveis valores que podem ser usados para executar P (Delamaro et al. 2016).

De forma semelhante, o **domínio de saída** do programa é:

Definição → Domínio de Saída

Conjunto de todos os possíveis resultados produzidos por P (Delamaro et al. 2016). (Incluindo mensagens de erro, etc.)

De onde vem toda essa complexidade?

Até um programa pequeno é capaz de processar muitas entradas diferentes, impossibilitando assim o **teste exaustivo**.

```
public double computeAverage(int a, int b, int c)
```

- Em uma máquina de 32 bits, cada variável pode assumir aproximadamente 4 bilhões de valores diferentes.
- Mais de 80 octilhões de casos de teste.[1]
- Quase como se o domínio de entrada fosse infinito!

[1] Mais precisamente, 79228162514264337593543950336 entradas.

Quiz

Considere um programa que recebe como parâmetros de entrada dois números inteiros x e y (com $y \geqslant 0$) e computa o valor de x^y (indicando um erro caso os valores estejam fora do intervalo especificado).

Pergunta ①: Qual é o domínio de entrada?

Pergunta ②: Qual é o domínio de saída?

Quiz

Considere um programa que recebe como parâmetros de entrada dois números inteiros x e y (com $y \geqslant 0$) e computa o valor de x^y (indicando um erro caso os valores estejam fora do intervalo especificado).

Pergunta ①: Qual é o domínio de entrada?

> Todos os possíveis pares de números inteiros, i.e., (x, y).

Pergunta ②: Qual é o domínio de saída?

Quiz

Considere um programa que recebe como parâmetros de entrada dois números inteiros x e y (com $y \geqslant 0$) e computa o valor de x^y (indicando um erro caso os valores estejam fora do intervalo especificado).

Pergunta ①: Qual é o domínio de entrada?

> Todos os possíveis pares de números inteiros, i.e., (x, y).

Pergunta ②: Qual é o domínio de saída?

> O conjunto de todos os inteiros (i.e.,$\mathbb{Z}$) e mensagens de erro.

De onde vem toda essa complexidade? (2)

Idealmente, o programa sendo testado deveria ser executado com todos os elementos em $\mathcal{D}(P)$. Porém, **por causa da cardinalidade** de $\mathcal{D}(P)$, isso é **infactível**. Por exemplo, considerando o programa que computa x^y:

- $\mathcal{D}(P)$ é formado por todos os pares de inteiros.

De onde vem toda essa complexidade? (2)

Idealmente, o programa sendo testado deveria ser executado com todos os elementos em $\mathcal{D}(P)$. Porém, **por causa da cardinalidade** de $\mathcal{D}(P)$, isso é **infactível**. Por exemplo, considerando o programa que computa x^y:

- $\mathcal{D}(P)$ é formado por todos os pares de inteiros.
- Produz um conjunto de cardinalidade $2^n \times 2^n$.

De onde vem toda essa complexidade? (2)

Idealmente, o programa sendo testado deveria ser executado com todos os elementos em $\mathcal{D}(P)$. Porém, **por causa da cardinalidade** de $\mathcal{D}(P)$, isso é **infactível**. Por exemplo, considerando o programa que computa x^y:

- $\mathcal{D}(P)$ é formado por todos os pares de inteiros.
- Produz um conjunto de cardinalidade $2^n \times 2^n$.
- Máquina de 32 bits, $2^{64} = 18446744073709551616$.

De onde vem toda essa complexidade? (2)

Idealmente, o programa sendo testado deveria ser executado com todos os elementos em $\mathcal{D}(P)$. Porém, **por causa da cardinalidade** de $\mathcal{D}(P)$, isso é **infactível**. Por exemplo, considerando o programa que computa x^y:

- $\mathcal{D}(P)$ é formado por todos os pares de inteiros.
- Produz um conjunto de cardinalidade $2^n \times 2^n$.
- Máquina de 32 bits, $2^{64} = 18446744073709551616$.
- Se cada caso de teste for executado em 1 milissegundo, seriam necessários 5.849.424 séculos para executar todos.

Como abordar essa complexidade?

É preciso procurar formas de **utilizar apenas um subconjunto** reduzido de $\mathcal{D}(P)$, mas **que tenha alta probabilidade de revelar a presença de defeitos**.

➠ A ideia central é usar "subdomínios de teste".

Definição → Subdomínio de Teste

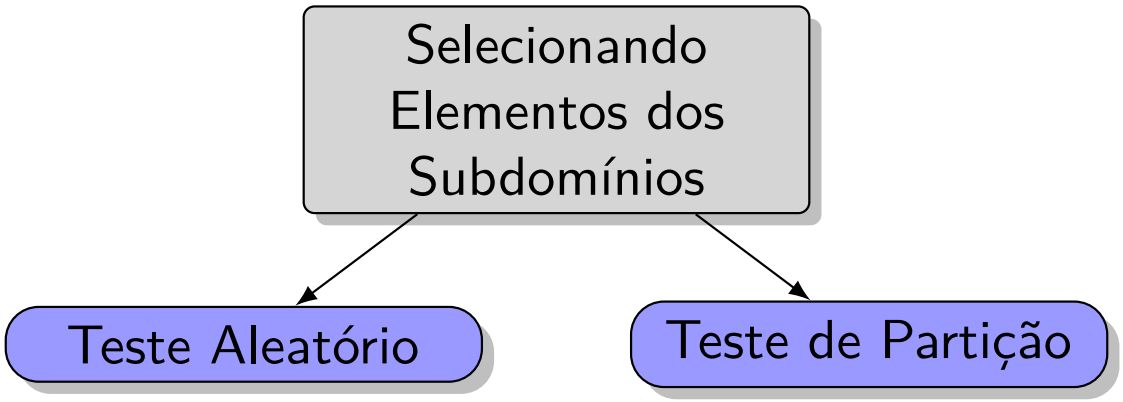
Um subconjunto de $\mathcal{D}(P)$ que contém dados de teste similares (Delamaro et al. 2016).

Por exemplo: $\langle(2, -1), \text{"Erro"}\rangle$ e $\langle(2, -2), \text{"Erro"}\rangle$ são semelhantes, i.e., espera-se que P se comporte de maneira semelhante.

✱ Generalizando, qualquer caso de teste da forma $\langle(x, y), \text{"Erro"}\rangle$ em que $x > 0$ e $y < 0$ se comporta do mesmo modo, i.e., pertence ao mesmo subdomínio.

Selecionando elementos do domínio...

Essencialmente, existem **duas formas** de se selecionar elementos de cada um dos subdomínios de teste:



Teste Aleatório

Testes são selecionados aleatoriamente, de forma que, probabilisticamente, exista uma boa chance que todos os subdomínios estejam representados.

Teste de Partição

Procura-se estabelecer quais subdomínios devem ser utilizados e, então, seleciona-se casos de teste para cada subdomínio.

Como determinar os subdomínios?

Em outras palavras, como determinar o menor número de entradas que é capaz de encontrar o maior número de problemas?

➠ **Critérios de cobertura** proporcionam uma forma estruturada e viável de se explorar o domínio de entrada.

- Proporcionam *traceability* dos modelos/artefatos de software para os casos de teste.
- Servem como "regra de parada".

Requisitos e critérios de teste

Definição → Critério de Teste

Estabelecem um conjunto de "regras" e um processo para definição de **requisitos de teste** (Delamaro et al. 2016; Ammann & Offutt 2016).

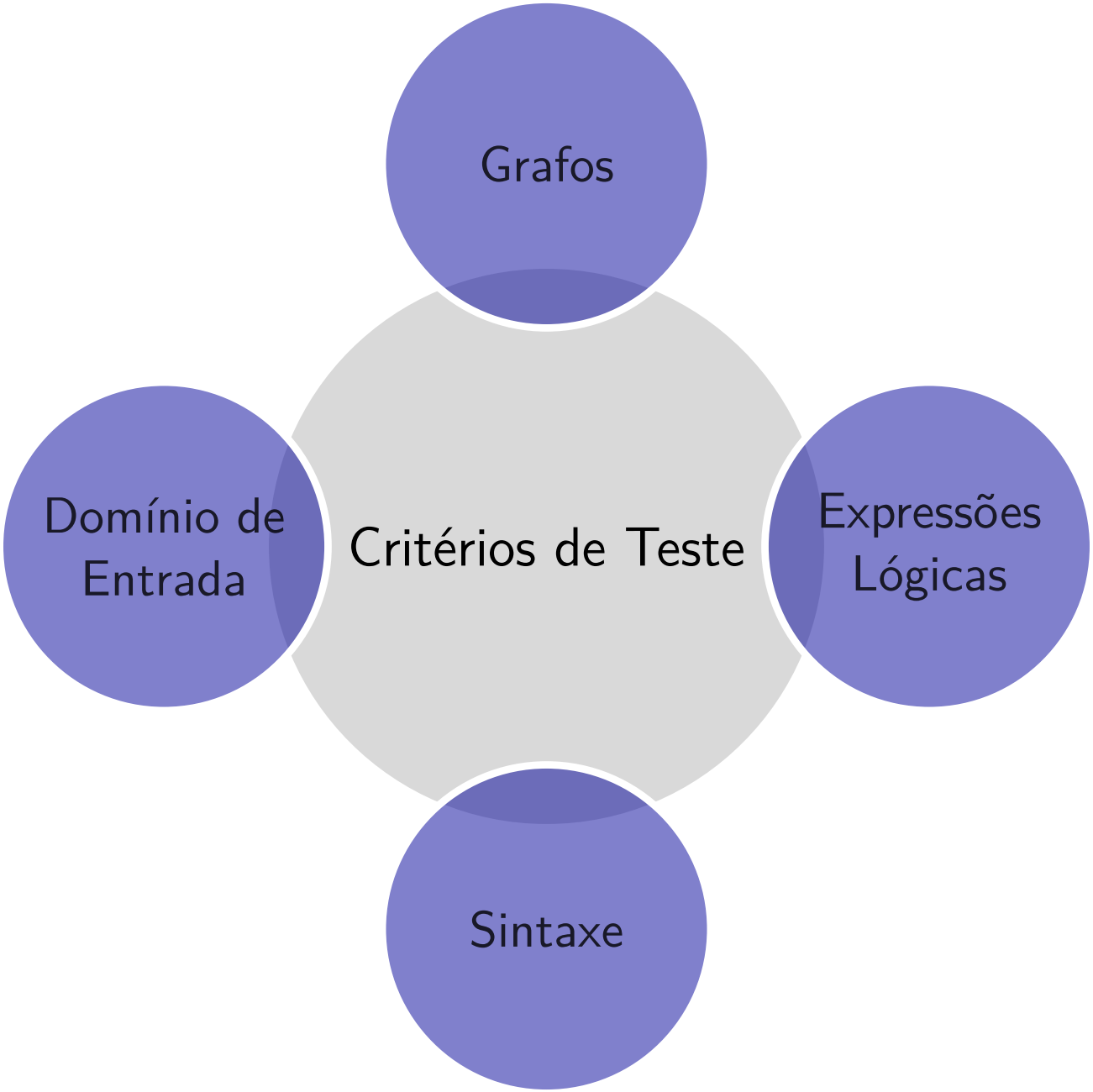
Requisitos de teste são os elementos que precisam ser cobertos ou satisfeitos durante o teste (e.g., executar uma determinada estrutura do programa).

> Os dados de teste que satisfazem um dado requisito de teste pertencem ao mesmo subdomínio.

Alguns exemplos de requisitos de teste:

- Toda sentença do programa;
- Cada requisito funcional.

Abstrações usadas para definição de critérios de teste



Antigamente, teste era definido em termos de "caixas coloridas"

Definição → Teste de Caixa Preta

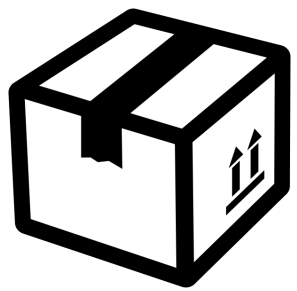
Testes são criados com base em descrições *externas* do software como, por exemplo, especificações e requisitos.

Definição → Teste de Caixa Branca

Testes são criados com base no código fonte: com base nos laços, condições e sentenças.

Definição → Teste Baseado em Modelos

Testes são derivados de modelos do software como, por exemplo, diagramas UML.



Considerações finais...

Na aula de hoje nós vimos:

- Teste é complicado pra xuxu!
  - Mitigando a complexidade: modelos.
- Critérios de teste;
  - Dividir para conquistar!
- Visão antiga: "caixas coloridas".

Na **próxima aula**:

- **Automação de teste de software**;

Referências

Ammann, Paul & Jeff Offutt (2016). *Introduction to Software Testing*. 2nd ed. Cambridge University Press, p. 364.

Delamaro, Marcio, Mario Jino, & Jose Carlos Maldonado (2016). *Introdução ao Teste de Software*. 2nd ed. Elsevier, p. 394.

☺**Próxima aula: exercício(s) sobre o conteúdo da aula de hoje!** ☺