

# Definições e Terminologia



INFORMAÇÃO  
TECNOLOGIA  
& INOVAÇÃO

Este material pode ser utilizado livremente respeitando-se a licença  
Creative Commons: Atribuição – Uso Não Comercial –  
Compartilhamento pela mesma Licença (by-nc-sa).



[Ver o Resumo da Licença](#) | [Ver o Texto Legal](#)

## Introdução

### Verificação, Validação & Teste

- Desafios do Teste

- Limitações do Teste

- Impossibilidade

### Casos de Teste

- Projeto de Casos de Teste

- Entrada/Saída

- Oráculo

- Ordem de Execução

## Introdução

### Verificação, Validação & Teste

- Desafios do Teste

- Limitações do Teste

- Impossibilidade

### Casos de Teste

- Projeto de Casos de Teste

- Entrada/Saída

- Oráculo

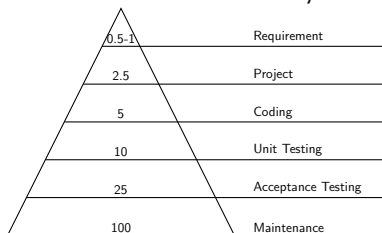
- Ordem de Execução

# Por que testar? I

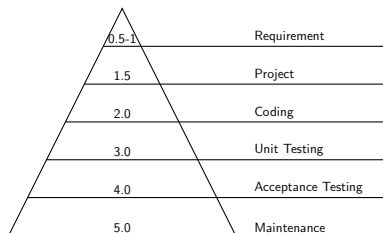
- ▶ Crescente interesse e importância do teste de software, principalmente devido à demanda por produtos de software de alta qualidade.
- ▶ Shull et al. (2002) alertam que quase não existem módulos livres de defeitos durante o desenvolvimento, e após a sua liberação, em torno de 40% podem estar livres de defeitos.
- ▶ Boehm e Basili (2001) também apontam que é quase improvável liberar um produto de software livre de defeitos.
- ▶ Além disso, quanto mais tarde um defeito é revelado, maior será o custo para sua correção.

# Por que testar? II

Escala de custo na correção de defeitos.



Boehm (1987)



Boehm e Basili (2001)

## Introdução

### Verificação, Validação & Teste

- Desafios do Teste

- Limitações do Teste

- Impossibilidade

### Casos de Teste

- Projeto de Casos de Teste

- Entrada/Saída

- Oráculo

- Ordem de Execução

# O que é teste? I

## Dijkstra (1970)

“Teste pode ser usado apenas para mostrar a presença de defeitos mas nunca sua ausência.”

## Myers (1979)

“Teste é o processo de executar um programa ou sistema com a intenção de encontrar erros”.

## Hetzel (1988)

“Teste é uma atividade que tem o objetivo de avaliar um atributo de um programa ou sistema. Teste é uma medida de qualidade de software”.





# O que é teste? II

Roper (1994)

“Teste é apenas uma amostragem”.

# O que é teste? III

## Craig e Jaskiel (2002)

“Teste é um processo de engenharia concorrente ao processo de ciclo de vida do software, que faz uso e mantém artefatos de teste usados para medir e melhorar a qualidade do produto de software sendo testado.”

## (ISO/IEC/IEEE, 2010)

“Teste é o processo de executar um sistema ou componente sob condições específicas, observando e registrando os resultados, avaliando alguns aspectos do sistema ou componente.”

# O que é teste? IV

## Validação

“Estamos construindo o sistema correto?” (Boehm, 1981)

“Processo de avaliar o software ao final de seu processo de desenvolvimento para garantir que está de acordo com o uso pretendido.” (Ammann e Offutt, 2008; ISO/IEC/IEEE, 2010)

## Verificação

“Estamos construindo corretamente o sistema?” (Boehm, 1981)

“Processo para determinar se os produtos de uma determinada fase de desenvolvimento satisfazem os requisitos estabelecidos na fase anterior.” (Ammann e Offutt, 2008; ISO/IEC/IEEE, 2010)



# O que é teste? V

- ▶ Diferentes organizações e indivíduos têm visões diferentes do propósito dos testes.
- ▶ Processo de Software *versus* Processo de Teste

# O que é teste? VI

- ▶ Níveis de maturidade de teste (Beizer, 1990):
  - ▶ Nível 0 - Não há diferença entre teste e depuração (*debugging*);
  - ▶ Nível 1 - O propósito do teste é mostrar que o software funciona.
  - ▶ Nível 2 - O propósito do teste é mostrar que o software não funciona.
  - ▶ Nível 3 - O propósito do teste não é provar nada, mas reduzir o risco de não funcionamento a um valor aceitável.
  - ▶ Nível 4 - Teste não é uma ação, mas sim uma disciplina mental (institucionalizada na empresa) que resulta em software de baixo risco sem que seja empregado muito esforço de teste.

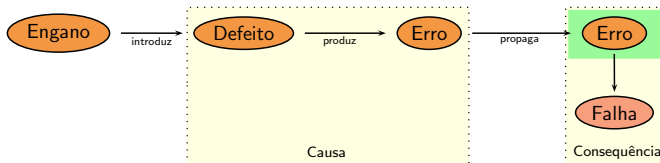
# Terminologia I

A ISO/IEC/IEEE 24765:2010(E) *Systems and Software Engineering – Vocabulary* (ISO/IEC/IEEE, 2010) diferencia os seguintes termos:

1. **Engano** (*Mistake*) – ação humana que produz um resultado incorreto.
2. **Defeito** (*Fault*) – um passo, processo, ou definição de dados incorreta em um produto de software. No uso comum, os termos “erro”, “bug”, e “defeito” são usados para expressar esse significado.
3. **Erro** (*Error*) – diferença entre o valor computado, observado ou medido e o valor teoricamente correto de acordo com a especificação.
4. **Falha** (*Failure*) – incapacidade do sistema ou componente realizar a função requerida, considerando as questões de desempenho exigidas.

# Terminologia II

Relação da terminologia padrão.



# Terminologia III

Considere o seguinte comando de atribuição: ( $z = y + x$ )

1. **Engano** – o comando é modificado para ( $z = y - x$ ), caracterizando um defeito.
2. **Defeito** – se ativado (executado) com  $x = 0$ , nenhum resultado incorreto é produzido. Para valores de  $x \neq 0$ , o defeito é ativado causando um erro na variável  $z$ .
3. **Erro** – o estado errôneo do sistema, quando propagado até a saída, causará uma **Falha**.



# Terminologia IV

- **Ammann e Offutt (2008)** definiram um modelo, denominado RIP, que estabelece as condições que devem ser estabelecidas para que a falha ocorra. São elas:
1. **Reachability** (Alcançabilidade): o local (ou locais) no programa contendo o defeito deve ser alcançado pelo caso de teste;
  2. **Infection** (Contaminação): o programa deve passar para um estado incorreto, ou seja, tem que produzir um erro;
  3. **Propagation** (Propagação): o estado contaminado deve se propagar de modo que o programa produza alguma saída incorreta que possa ser observada;

# Terminologia V

- ▶ Assim sendo, dois conceitos chaves para o sucesso da atividade de teste são:

## Observabilidade de Software

“Quão fácil é observar o comportamento de um programa em termos de sua saída, efeitos no ambiente e outros componentes de hardware e software.” (Ammann e Offutt, 2008)

## Controlabilidade de Software

“Quão fácil é fornecer as entradas necessárias para o programa, em termos de valores, operações e comportamentos.” (Ammann e Offutt, 2008)

# Desafios

- ▶ Alguém já testou algum produto de software?
- ▶ Quais foram os maiores desafios?
- ▶ Alguns problemas comuns são:
  - ▶ Não há tempo para o teste exaustivo.
  - ▶ Muitas combinações de entrada para serem exercitadas.
  - ▶ Dificuldade em determinar os resultados esperados para cada caso de teste.
  - ▶ Requisitos do software inexistentes ou que mudam rapidamente.
  - ▶ Não há tempo suficiente para o teste.
  - ▶ Não há treinamento no processo de teste.
  - ▶ Não há ferramenta de apoio.
  - ▶ Gerentes que desconhecem teste ou que não se preocupam com qualidade.



# Problemas Indecidíveis I

**Correção:** Não existe um algoritmo de propósito geral para provar a correção de um produto de software;

**Equivalência:** Dados dois programas, decidir se eles são equivalentes; ou dados dois caminhos (sequência de comandos) decidir se eles computam a mesma função;

**Executabilidade:** Dado um caminho (sequência de comandos) decidir se existe um dado de entrada que leve à execução de tal caminho.

**Correção Coincidente:** Um produto pode apresentar, coincidentemente, um resultado correto para um dado valor de entrada  $d \in D$  porque um defeito mascara o erro de outro defeito.

# Problemas Indecidíveis II

## Exemplos de Equivalência:

```

1 public class Factorial {
2     public static long compute(int x)
3         throws NegativeNumberException {
4         if (x >= 0) {
5             long r = 1;
6             for (int k = 2; k <= x; k++) {
7                 r *= k;
8             }
9             return r;
10        } else {
11            throw new NegativeNumberException();
12        }
13    }
14 }

```

```

1 public class Factorial {
2     public static long compute(int x)
3         throws NegativeNumberException {
4         if (x >= 0) {
5             long r = 1;
6             for (int k = 1; k <= x; k++) { //int k = 2;
7                 r *= k;
8             }
9             return r;
10        } else {
11            throw new NegativeNumberException();
12        }
13    }
14 }

```

INFORMAÇÃO  
TECNOLOGIA  
& INOVAÇÃO

# Problemas Indecidíveis III

## Exemplo de Não-executabilidade:

```
4 public boolean validateIdentifier(String s) {
5     char achar;
6     boolean valid_id = false;
7     achar = s.charAt(0);
8     valid_id = valid_s(achar);
9     if (s.length() > 1) {
10         achar = s.charAt(1);
11         int i = 1;
12         while (i < s.length() - 1) {
13             achar = s.charAt(i);
14             if (!valid_f(achar))
15                 valid_id = false;
16             i++;
17         }
18     }
19
20     if (valid_id && (s.length() >= 1) && (s.length() < 6))
21         return true;
22     else
23         return false;
24 }
```

- ▶ Não existe um caminho executável que inclua, simultaneamente, os comandos das linhas 15 e 21.



INFORMAÇÃO  
TECNOLOGIA  
& INOVAÇÃO

# Impossibilidade do Teste Exaustivo I

Por que, simplesmente, não se testa todo o produto de forma exaustiva?

# Impossibilidade do Teste Exaustivo II

- ▶ Pezzè e Young (2007) ressaltam que o teste exaustivo pode ser considerado uma “prova por casos” legítima, mas quanto tempo levaria para ser executado?
- ▶ Considere que programas são executados em máquinas reais com representação finita dos valores na memória.
- ▶ Quanto tempo o programa Java abaixo levaria para ser testado de forma exaustiva considerando que cada caso de teste leva um nanosegundo ( $10^{-9}$  segundos) para ser executado?

```
1 public class Trivial {  
2     static int sum(int a, int b) { return a + b; }  
3 }
```





# Impossibilidade do Teste Exaustivo III

Adaptado de Pezzè e Young (2007)

```
1 public class Trivial {  
2     static int sum(int a, int b) { return a + b;}  
3 }
```

- ▶ Em Java um `int` é representado por 32 bits
- ▶ Tamanho do Domínio de Entrada:  $2^{32} * 2^{32} = 2^{64} \approx 10^{20}$
- ▶ Um nanosegundo:  $10^{-9}$  segundos
- ▶ Tempo total de execução:  $10^{11}$  segundos  $\approx 3.100$  anos

# Impossibilidade do Teste Exaustivo IV

Observe o exemplo abaixo, adaptado de [Binder \(1999\)](#):

```
1 int blech(int j) {  
2     j = j - 1; // defeito - deveria ser = j + 1  
3     j = j / 30000;  
4     return j;  
5 }
```

- ▶ Considerando um tipo inteiro de 16 bits (2 bytes) – o menor valor possível é -32,768 e o maior é 32,767, resultando em 65.536 valores diferentes possíveis.
- ▶ Haverá tempo suficiente para se criarem 65.536 casos de teste? E se os programas forem maiores? Quantos casos de teste seriam necessários?

# Impossibilidade do Teste Exaustivo V

```
1 int blech(int j) {  
2     j = j - 1; // defeito - deveria ser = j + 1  
3     j = j / 30000;  
4     return j;  
5 }
```

Quais valores escolher?

Entrada (j)	Saída Esperada	Saída Obtida
1	0	0
42	0	0
40000	1	1
-64000	-2	-2

- ▶ Observe que nenhum dos casos de teste acima detecta o defeito.
- ▶ Quais valores de entrada levam o programa acima a falhar?



# Impossibilidade do Teste Exaustivo VI

- ▶ Os casos de testes anteriores não fazem o programa falhar.
- ▶ Somente quatro valores do intervalo de entrada válido resultam em falha:
- ▶ Os valores abaixo causam falha no programa:

Entrada (j)	Saída Esperada	Saída Obtida
-30000	0	-1
-29999	0	-1
30000	1	0
29999	1	0

- ▶ Qual a chance desses valores serem selecionados se:
  - ▶ for utilizada geração aleatória?
  - ▶ for utilizada geração guiada por critério de teste?

## Introdução

## Verificação, Validação & Teste

Desafios do Teste

Limitações do Teste

Impossibilidade

## Casos de Teste

Projeto de Casos de Teste

Entrada/Saída

Oráculo

Ordem de Execução

# Projeto de Casos de Teste

- ▶ O segredo do sucesso do teste está no projeto dos casos de teste.
- ▶ Relembrando...

Roper (1994)

“Teste é apenas uma amostragem”.



Diagrama simplificado de um processo de teste (adaptado de Roper (1994)).

# Partes de um Caso de Teste

- ▶ Casos de teste bem projetados são compostos por duas partes obrigatórias e outras opcionais:
  - ▶ Pré-condição;
  - ▶ Entrada;
  - ▶ Saída Esperada;
  - ▶ Ordem de execução.
- ▶  $(d, S(d))$  é um caso de teste, onde  $d \in D$  é a entrada e  $S(d)$  representa a saída esperada de  $d$  de acordo com a especificação  $S$ .

# Partes de um Caso de Teste – Entradas

- ▶ Geralmente identificadas como dados fornecidos via teclado para o programa executar.
- ▶ Entretanto, os dados de entrada podem ser fornecidos por outros meios, tais como:
  - ▶ Dados oriundos de outro sistema que servem de entrada para o programa em teste;
  - ▶ Dados fornecidos por outro dispositivo;
  - ▶ Dados lidos de arquivos ou banco de dados;
  - ▶ O estado do sistema quando os dados são recebidos;
  - ▶ O ambiente no qual o programa está executando.



# Partes de um Caso de Teste – Saídas Esperadas

- ▶ As saídas também podem ser produzidas de diferentes formas.
- ▶ A mais comum é aquela apresentada na tela do computador.
- ▶ Além dessa, as saídas podem ser enviadas para:
  - ▶ Outro sistema interagindo com o programa em teste;
  - ▶ Dados escritos em arquivos ou banco de dados;
  - ▶ O estado do sistema ou o ambiente de execução podem ser alterados durante a execução do programa.

# Partes de um Caso de Teste – Oráculo (1)

- ▶ Todas as formas de entrada e saída são relevantes.

# Partes de um Caso de Teste – Oráculo (1)

- ▶ Todas as formas de entrada e saída são relevantes.
- ▶ Durante o projeto de um caso de teste, determinar a correção da saída esperada é função do **oráculo** (*oracle*).

# Partes de um Caso de Teste – Oráculo (1)

- ▶ Todas as formas de entrada e saída são relevantes.
- ▶ Durante o projeto de um caso de teste, determinar a correção da saída esperada é função do **oráculo** (*oracle*).
- ▶ O oráculo corresponde a um mecanismo (programa, processo ou dados) que indica ao projetista de casos de testes se a saída obtida para ele é aceitável ou não.

## Partes de um Caso de Teste – Oráculo (2)

- ▶ **Beizer (1990)** lista cinco tipos de oráculos:
  - ▶ Oráculo “Kiddie” - simplesmente execute o programa e observe sua saída. Se ela parecer correta deve estar correta.
  - ▶ Conjunto de Teste de Regressão - execute o programa e compare a saída obtida com a saída produzida por uma versão mais antiga do programa.
  - ▶ Validação de Dados - execute o programa e compare a saída obtida com uma saída padrão determinada por uma tabela, fórmula ou outra definição aceitável de saída válida.
  - ▶ Conjunto de Teste Padrão - execute o programa com um conjunto de teste padrão que tenha sido previamente criado e validado. Utilizado na validação de compiladores, navegadores Web e processadores de SQL.
  - ▶ Programa existente - execute o programa em teste e o programa existente com o mesmo caso de teste e compare as saídas. Semelhante ao teste de regressão.

# Partes de um Caso de Teste – Ordem de Execução (1)

- ▶ Existem dois estilos de projeto de casos de teste relacionados com a ordem de execução:
  - ▶ Casos de teste em cascata.
  - ▶ Casos de teste independentes.

## Partes de um Caso de Teste – Ordem de Execução (2)

- ▶ Casos de teste em cascata - quando os casos de teste devem ser executados um após o outro, em uma ordem específica. O estado do sistema deixado pelo primeiro caso de teste é reaproveitado pelo segundo e assim sucessivamente.

Por exemplo, considere o teste de uma base de dados:

- ▶ criar um registro
- ▶ ler um registro
- ▶ atualizar um registro
- ▶ ler um registro
- ▶ apagar um registro
- ▶ ler o registro apagado
- ▶ Vantagem - casos de testes tendem a ser pequenos e simples. Fáceis de serem projetados, criados e mantidos.
- ▶ Desvantagem - se um caso de teste apresentar problemas na sua execução, pode comprometer a execução dos casos de teste subsequentes.



## Partes de um Caso de Teste – Ordem de Execução (3)

- ▶ Casos de teste independentes - Cada caso de teste é inteiramente auto contido.
  - ▶ Vantagem - casos de teste podem ser executados em qualquer ordem.
  - ▶ Desvantagem - casos de teste tendem a ser grandes e complexos, mais difíceis de serem projetados, criados e mantidos.



# Conjunto de Teste

- ▶ O termo **conjunto de teste** é usado para definir um conjunto de casos de teste.
- ▶ Quando um conjunto de teste  $T$  satisfaz todos os requisitos de um determinado critério de teste  $C$ ,  $T$  é dito adequado a  $C$ , ou simplesmente,  $T$  é  $C$ -adequado.
- ▶ A partir de um  $T$   $C$ -adequado é possível obter, em teoria, infinitos conjuntos de teste  $C$ -adequados, simplesmente incluindo mais casos de teste em  $T$ .
- ▶ Minimização de conjunto de teste.

## Taxonomia de Defeitos

### Tipos de Teste

#### Técnicas de Teste

### Fases de Teste

#### Teste de Unidade

#### Teste de Integração

#### Teste de Sistema

#### Teste de Aceitação

#### Teste Alfa e Beta

#### Pirâmide de Teste

#### Desenvolvimento e VV&T

### Resumo



# Taxonomia de Defeitos (1)

## Taxonomia (Copeland, 2004)

Classificação de coisas em grupos ordenados ou categorias que indicam relacionamentos hierárquicos ou naturais.

- ▶ Taxonomias não facilitam apenas a classificação ordenada de informação, elas também facilitam sua recuperação e descoberta de novas ideias.

## Classificação de Defeitos (2)

- ▶ Taxonomias auxiliam a (Copeland, 2004):
  - ▶ Guiar o testador na geração de ideias para o projeto de teste;
  - ▶ Auditar planos de teste para determinar a cobertura que os casos de teste estão oferecendo;
  - ▶ Compreender os defeitos, seus tipos e severidades;
  - ▶ Compreender o processo em uso que permite a ocorrência de tais defeitos;
  - ▶ Melhorar o processo de desenvolvimento;
  - ▶ Melhorar o processo de teste;
  - ▶ Treinar novos testadores sobre áreas importantes que merecem mais atenção dos testes;
  - ▶ Explicar aos gerentes a complexidade do teste de software.



## Classificação de Defeitos (3)

- Uma classificação geral divide os defeitos em duas classes distintas: defeitos de omissão e defeitos de comissão.

### Defeitos de Omissão (IBM, 2013)

Omissão significa esquecimento. Por exemplo, um comando de atribuição que foi esquecido.

### Defeitos de Comissão (IBM, 2013)

Comissão significa incorreto. Por exemplo, um comando de verificação que utiliza um valor incorreto.



## Taxonomia de Defeitos

## Tipos de Teste

### Técnicas de Teste

## Fases de Teste

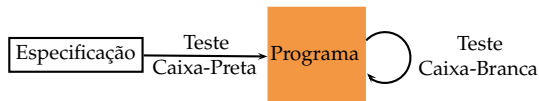
- Teste de Unidade
- Teste de Integração
- Teste de Sistema
- Teste de Aceitação
- Teste Alfa e Beta
- Pirâmide de Teste
- Desenvolvimento e VV&T

## Resumo

# Tipos de Teste I

- ▶ Diferentes tipos de testes podem ser utilizados para verificar se um programa se comporta como o especificado.
- ▶ Basicamente, os testes podem ser classificados como teste caixa-preta (*black-box testing*), teste caixa-branca (*white-box testing*) ou teste baseado em defeito (*fault-based testing*).
- ▶ Esses **tipos de teste** correspondem às chamadas **técnicas de teste**.

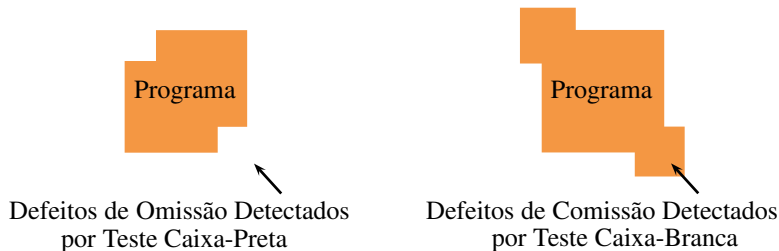
# Tipos de Teste II



Relacionamento entre as técnicas caixa-preta e caixa-branca (adaptado de [Roper \(1994\)](#))



# Tipos de Teste III



Relacionamento entre as técnicas caixa-preta e caixa-branca (adaptado de Roper (1994))

# Tipos de Teste IV

- ▶ A técnica de teste é definida pelo tipo de informação utilizada para realizar o teste.
  - ▶ **Técnica caixa-preta** - os testes são baseados exclusivamente na especificação de requisitos do programa. Nenhum conhecimento de como o programa está implementado é requerido.
  - ▶ **Técnica caixa-branca** - os testes são baseados na estrutura interna do programa, ou seja, na sua implementação.
  - ▶ **Técnica baseada em defeito** - os testes são baseados em informações históricas sobre defeitos cometidos frequentemente durante o processo de desenvolvimento de software.

# Tipos de Teste V

- ▶ Cada técnica de teste possui um conjunto de **critérios de teste**.
- ▶ Os critérios sistematizam a forma como os **requisitos de teste** devem ser produzidos a partir da fonte de informação disponível (especificação de requisitos, código fonte, histórico de defeitos, entre outras).
- ▶ Os requisitos de teste são utilizados para:
  - ▶ Gerar casos de teste.
  - ▶ Avaliar a qualidade de um conjunto de teste existente.
- ▶ Critérios de teste ajudam a decidir quando parar os testes.

# Tipos de Teste VI

- ▶ De modo mais formal, dados um programa  $P$  e um conjunto de teste  $T$ , definem-se:
  - ▶ **Critério de Adequação de Casos de Teste:** predicado para avaliar  $T$  no teste de  $P$ ;
  - ▶ **Método de Seleção de Casos de Teste:** procedimento para escolher casos de teste para o teste de  $P$ .

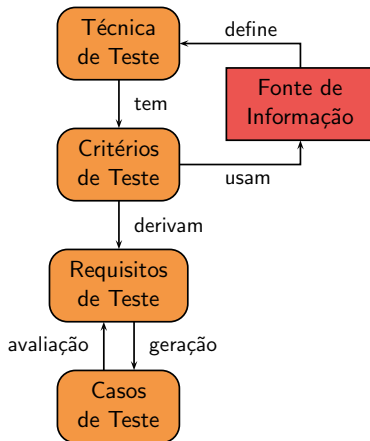
# Tipos de Teste VII

- ▶ Dados  $P$ ,  $T$  e um critério  $C$ , diz-se que o conjunto de teste  $T$  é  $C$ -adequado para o teste de  $P$  se  $T$  satisfizer os requisitos de teste estabelecidos pelo critério  $C$ .
- ▶ Outra questão relevante é:
  - ▶ dado um conjunto  $T$   $C_1$ -adequado, qual seria um critério de teste  $C_2$  que contribuiria para aprimorar  $T$ ? Essa questão tem sido investigada tanto em estudos teóricos quanto em estudos experimentais.

# Tipos de Teste VIII

- ▶ Em geral, pode-se dizer que as propriedades mínimas que devem ser preenchidas por um critério de teste  $C$  são:
  1. garantir, do ponto de vista de fluxo de controle, a cobertura de todos os desvios condicionais;
  2. requerer, do ponto de vista de fluxo de dados, ao menos um uso de todo resultado computacional; e
  3. requerer um conjunto de casos de teste finito.

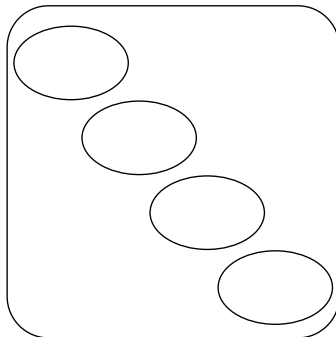
# Técnicas, Critérios e Requisitos de Teste



**\*\*Requisitos não-executáveis.**

# Papel dos Critérios de Teste

- ▶ Subdividir o domínio de entrada forçando o testador a escolher diferentes elementos para o teste.



Partições do domínio de entrada de um problema (adaptado de Pezzè e Young (2007))

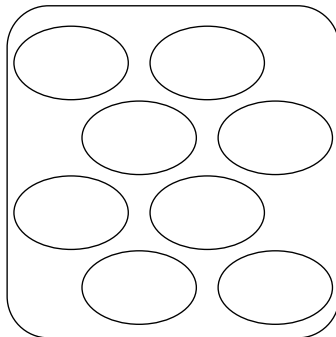


INFORMAÇÃO  
TECNOLOGIA  
& INOVAÇÃO



# Papel dos Critérios de Teste

- ▶ Subdividir o domínio de entrada forçando o testador a escolher diferentes elementos para o teste.



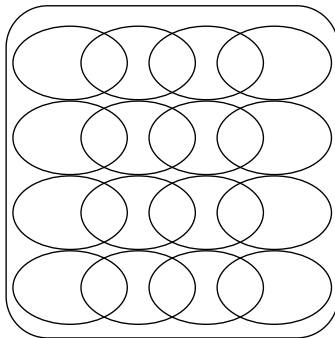
Partições do domínio de entrada de um problema (adaptado de Pezzè e Young (2007))



INFORMAÇÃO  
TECNOLOGIA  
& INOVAÇÃO

# Papel dos Critérios de Teste

- ▶ Subdividir o domínio de entrada forçando o testador a escolher diferentes elementos para o teste.



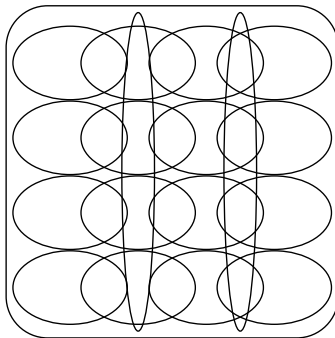
Partições do domínio de entrada de um problema (adaptado de Pezzè e Young (2007))



INFORMAÇÃO  
TECNOLOGIA  
& INOVAÇÃO

# Papel dos Critérios de Teste

- ▶ Subdividir o domínio de entrada forçando o testador a escolher diferentes elementos para o teste.



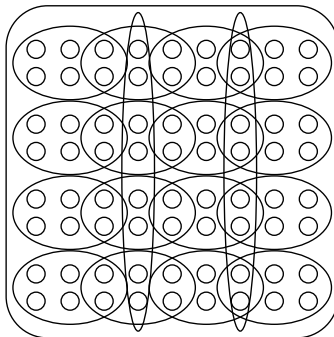
Partições do domínio de entrada de um problema (adaptado de Pezzè e Young (2007))



INFORMAÇÃO  
TECNOLOGIA  
& INOVAÇÃO

# Papel dos Critérios de Teste

- ▶ Subdividir o domínio de entrada forçando o testador a escolher diferentes elementos para o teste.



Partições do domínio de entrada de um problema (adaptado de Pezzè e Young (2007))



INFORMAÇÃO  
TECNOLOGIA  
& INOVAÇÃO

## Taxonomia de Defeitos

## Tipos de Teste

### Técnicas de Teste

## Fases de Teste

Teste de Unidade

Teste de Integração

Teste de Sistema

Teste de Aceitação

Teste Alfa e Beta

Pirâmide de Teste

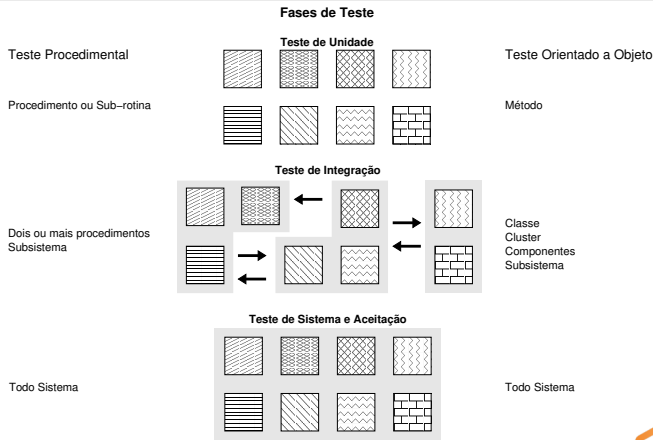
Desenvolvimento e VV&T

## Resumo

# Fases de Teste (1)

- ▶ A Atividade de Teste também é dividida em fases, conforme outras atividades de Engenharia de Software.
- ▶ O objetivo é reduzir a complexidade dos testes.
- ▶ Conceito de “dividir e conquistar”.
- ▶ Começar a testar a menor unidade executável até atingir todo o programa.

# Fases de Teste (2)



Principais Fases de Teste (adaptada de Binder (1999))



INFORMAÇÃO  
TECNOLOGIA  
& INOVAÇÃO

# Teste de Unidade

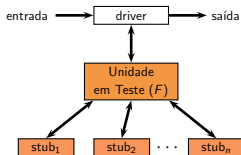
- ▶ O objetivo é identificar erros de lógica e de programação na menor unidade de programação.
- ▶ Diferentes linguagens possuem unidades diferentes.
  - ▶ Pascal e C possuem procedimentos ou funções.
  - ▶ Java e C++ possuem métodos (ou classes?).
  - ▶ Basic e COBOL (o que seriam unidades?).
- ▶ Como testar uma unidade que depende de outra para ser executada?
- ▶ Como testar uma unidade que precisa receber dados de outra unidade para ser executada?





# Driver e Stub

- ▶ Para auxiliar no teste de unidade, em geral, são necessários *drivers* e *stubs*.
- ▶ *driver* fornece os dados necessários para executar a unidade e, posteriormente, apresenta os resultados ao testador.
- ▶ *stub* simula o comportamento de uma unidade que ainda não foi desenvolvida, mas da qual a unidade em teste depende. Arcabouços de geração de mocks são bastante úteis para simular essas unidades.



Drivers e Stubs

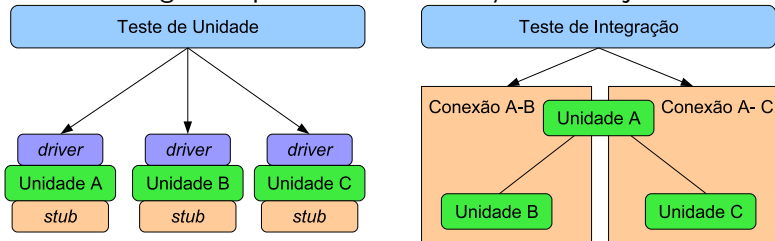


# Teste de Integração (1)

- ▶ O Objetivo é verificar se as unidades testadas individualmente se comunicam como desejado.
  1. Por que testar a integração entre unidades se elas, isoladamente, funcionam corretamente?

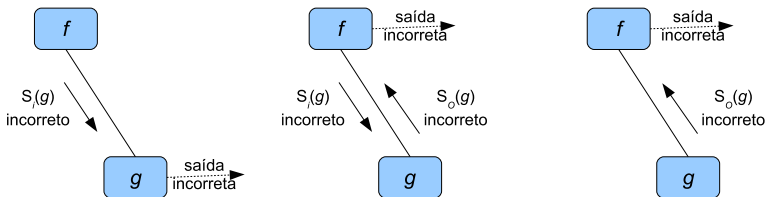
## Teste de Integração (2)

- ▶ Dados podem se perder na interface das unidades.
- ▶ Variáveis globais podem sofrer alterações indesejadas.



Teste de Unidade X Teste de Integração.

# Teste de Integração (3)



Tipos de Erros de Integração.

# Teste de Sistema

- ▶ O objetivo é verificar se o programa em si interage corretamente com o sistema para o qual foi projetado. Isso inclui, por exemplo, o SO, banco de dados, hardware, manual do usuário, treinamento, etc.
- ▶ Corresponde a um teste de integração de mais alto nível.
- ▶ Inclui teste de funcionalidade, usabilidade, segurança, confiabilidade, disponibilidade, performance, backup/restauração, portabilidade, entre outros (Norma ISO-IEC-9126 para mais informações ([ISO/IEC, 1991](#)))

# Teste de Aceitação

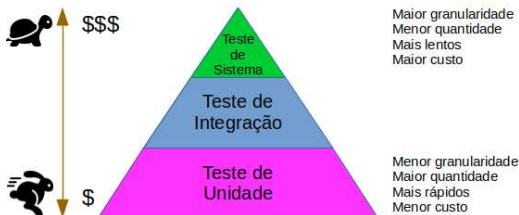
- ▶ O objetivo é verificar se o programa desenvolvido atende às exigências do usuário.

# Teste Alfa e Beta

- ▶ Teste Alfa
  - ▶ realizado pelo usuário no ambiente do desenvolvedor;
  - ▶ o desenvolvedor tem controle sobre o ambiente de teste;
  - ▶ monitora as ações do usuário registrando falhas e problemas de uso.
- ▶ Teste Beta
  - ▶ realizado pelo usuário no seu ambiente;
  - ▶ o desenvolvedor não tem controle sobre o ambiente utilizado nos testes;
  - ▶ O diferencial é a grande diversidade de ambientes e configurações de software e hardware.

# Pirâmide de Teste (1)

- ▶ Cohn (2009) discute a importância da Pirâmide de Teste no processo de garantia da qualidade em equipes que trabalham com métodos ágeis
- ▶ A pirâmide reflete onde a equipe deveria concentrar esforços de automatização
- ▶ Observa-se que mais ênfase é dada aos testes unitários e do que aos testes de sistema



Fonte: Adaptada de Cohn (2009)



INFORMAÇÃO  
TECNOLOGIA  
& INOVAÇÃO



# Pirâmide de Teste (2)

- ▶ A ênfase em teste unitário tem alguns motivos:
  - ▶ Teste unitário fornece valores específicos para o programador
  - ▶ Se uma falha é detectada por um teste unitário, o escopo para a busca pelo defeito é menor
  - ▶ São mais fáceis de automatizar e de manter

## Pirâmide de Teste (3)

- ▶ A menor ênfase nos testes de sistema via interface do usuário são devido a:
  - ▶ Facilidade de “quebra” dos testes mesmo quando ocorrem apenas pequenas mudanças na interface
  - ▶ Custo para escrever esses testes é maior pois existem diversos fatores que dificultam a ação das ferramentas de captura e reprodução, em geral, usadas nessa atividade
  - ▶ Tempo de execução elevado pois a execução dos testes via interface é bem mais lenta do que a execução sem a presença da interface

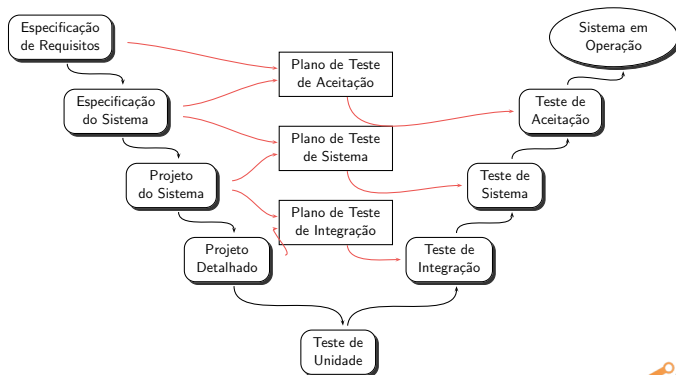
# Fases de Desenvolvimento versus Atividades de VV&T (1)

## Questões básicas segundo Pezzè e Young (2007)

1. Quando a verificação e validação começam? Quando estão completas?
2. Que técnicas específicas devem ser aplicadas durante o desenvolvimento do produto para atingir a qualidade exigida a um custo aceitável?
3. Como avaliar se um produto está pronto para a liberação?
4. Como o processo de desenvolvimento em si pode ser aprimorado no decurso de projetos atuais e futuros para melhorar os produtos e tornar a verificação mais rentável?

# Fases de Desenvolvimento versus Atividades de VV&T (2)

## Modelo V (V-Model)



# Fases de Desenvolvimento versus Atividades de VV&T (3)

	Elicitação de Requisitos	Especificação de Requisitos	Projeto da Arquitetura	Projeto Detalhado	Codificação	Integração e Liberação	Manutenção
Planejar e Monitorar	Identificar qualidades						
		Plano de teste de aceitação					
			Plano de teste de sistema				
				Plano de teste unitário e de integração			
			Monitorar processo de teste e análise				
Verificar Especificações		Validar especificações					
			Análise do projeto da arquitetura				
			Inspeção do projeto da arquitetura				
				Inspeção do projeto detalhado			
Gerar Casos de Teste					Inspeção do código		
		Gerar testes de sistema					
			Gerar testes de integração				
				Gerar teste unitário			
						Gerar teste de regressão	
						Atualizar teste de regressão	

Principais atividades de teste e análise ao longo do ciclo de vida do software (adaptada de Pezzè e Young (2007))

# Fases de Desenvolvimento versus Atividades de VV&T (4)

	Elicitação de Requisitos	Especificação de Requisitos	Projeto da Arquitetura	Projeto Detalhado	Codificação	Integração e Liberação	Manutenção
Executar Casos de Teste e Validar o Software				Projeto do código de teste			
				Projeto de oráculos			
					Execução do teste unitário		
					Análise de cobertura		
					Geração do teste estrutural		
					Execução do teste estrutural		
					Execução do teste de integração		
					Execução do teste de sistema		
Melhoria do Processo						Execução do teste de regressão	
					Coleta de dados de falhas		
					Análise de falhas		
					Melhoria do processo		

Principais atividades de teste e análise ao longo do ciclo de vida do software (adaptada de Pezzè e Young (2002)).



INFORMAÇÃO  
TECNOLOGIA  
& INOVAÇÃO

## Taxonomia de Defeitos

## Tipos de Teste

### Técnicas de Teste

## Fases de Teste

### Teste de Unidade

### Teste de Integração

### Teste de Sistema

### Teste de Aceitação

### Teste Alfa e Beta

### Pirâmide de Teste

### Desenvolvimento e VV&T

## Resumo

# Resumo I

- ▶ A atividade de teste é um processo executado em paralelo com as demais atividades do ciclo de vida de desenvolvimento do software.
- ▶ A principal tarefa da atividade de teste é o projeto de casos de teste.
  - ▶ O projeto de casos de teste deve levar em consideração o objetivo do teste, que é demonstrar que o programa em teste possui defeitos.
  - ▶ O segredo é selecionar aqueles casos de teste com maior probabilidade de revelar os defeitos existentes.
- ▶ Eles é que são responsáveis por:
  - ▶ expor as falhas do produto em teste; ou
  - ▶ assegurar um nível de confiança ao produto.





## Resumo II

- ▶ Diferentes técnicas de teste existem para auxiliar na atividade de teste.
- ▶ Cada técnica possui critérios de teste que contribuem para a geração ou avaliação da qualidade de conjuntos de teste.
- ▶ Os critérios de teste ajudam a decidir quando parar os testes.
- ▶ Além disso, o teste deve ser conduzido em fases para reduzir a complexidade.

# Referências I

- Ammann, P.; Offutt, J. *Introduction to Software Testing*. Cambridge University Press, 2008.
- Beizer, B. *Software Testing Techniques*. 2nd ed. New York: Van Nostrand Reinhold Company, 1990.
- Binder, R. V. *Testing Object-Oriented Systems: Models, Patterns, and Tools*, v. 1. Addison Wesley Longman, Inc., 1999.
- Boehm, B.; Basili, V. R. Software Defect Reduction Top 10 List. *Computer*, v. 34, n. 1, p. 135–137, bibtex\*[location=Los Alamitos, CA, USA;publisher=IEEE Computer Society Press], 2001.
- Boehm, B. W. *Software Engineering Economics*. 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.
- Boehm, B. W. Industrial Software Metrics Top 10 List. *IEEE Software*, v. 4, n. 5, p. 84–85, bibtex\*[owner=Auri;timestamp=2007.05.16], 1987.
- Cohn, M. *Succeeding with Agile: Software Development Using Scrum*. 1st ed. Addison-Wesley Professional, 2009.
- Copeland, L. *A Practitioner's Guide to Software Test Design*. Artech House Publishers, 2004.
- Craig, R. D.; Jaskiel, S. P. *Systematic Software Testing*. Artech House Publishers, 2002.
- Dijkstra, E. W. *Notes on Structured Programming*. Relatório Técnico 70-WSK-03, Technological University of Eindhoven, The Netherlands, available at: <http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF>. Accessed on: 10-27-2007., 1970.
- Hetzel, B. *The Complete Guide to Software Testing*. 2nd ed. Wellesley, MA, USA: QED Information Sciences, Inc., 1988.
- IBM. Orthogonal Defect Classification v 5.2 for Software Design and Code. Disponível em: [http://researcher.watson.ibm.com/researcher/view\\_project.php?id=480](http://researcher.watson.ibm.com/researcher/view_project.php?id=480). Acesso em: 02/02/2014. bibtex\*[howpublished=Página Web], 2013.
- ISO/IEC. *Quality Characteristics and Guidelines for their Use*. Padrão ISO/IEC 9126, ISO/IEC, 1991.
- ISO/IEC/IEEE. Systems and software engineering – vocabulary. *ISO/IEC/IEEE 24765:2010(E)*, vol. 1, 24765:2010(E), p. 1–418, bibtex\*[organization=ISO/IEC/IEEE;institution=ISO/IEC/IEEE], 2010.



## Referências II

Maldonado, J. C.; Barbosa, E. F.; Vincenzi, A. M. R.; Delamaro, M. E.; Souza, S. R. S.; Jino, M. *Introdução ao Teste de Software*. Relatório Técnico 65 – Versão 2004-01, Instituto de Ciências Matemáticas e de Computação – ICMC-USP, disponível on-line:  
[http://www.icmc.usp.br/CMS/Arquivos/arquivos\\_enviados/BIBLIOTECA\\_113\\_ND\\_65.pdf](http://www.icmc.usp.br/CMS/Arquivos/arquivos_enviados/BIBLIOTECA_113_ND_65.pdf).  
bibtex\*[howpublished=Notas Didáticas do ICMC], 2004.

Myers, G. J. *The Art of Software Testing*. Wiley, New York, 1979.

Pezzè, M.; Young, M. *Software Testing and Analysis: Process, Principles and Techniques*. John Wiley & Sons, 2007.

Roper, M. *Software Testing*. McGraw Hill, 1994.

Shull, F.; Basili, V.; Boehm, B.; Brown, A. W.; Costa, P.; Lindvall, M.; Port, D.; Rus, I.; Tesoriero, R.; Zelkowitz, M. What We Have Learned About Fighting Defects. In: *VIII International Symposium on Software Metrics - METRICS'02*, IEEE Computer Society, event-place: Washington, DC, USA, 2002, p. 249–258.