Teste de Software

Prof. Dr. Alan Souza

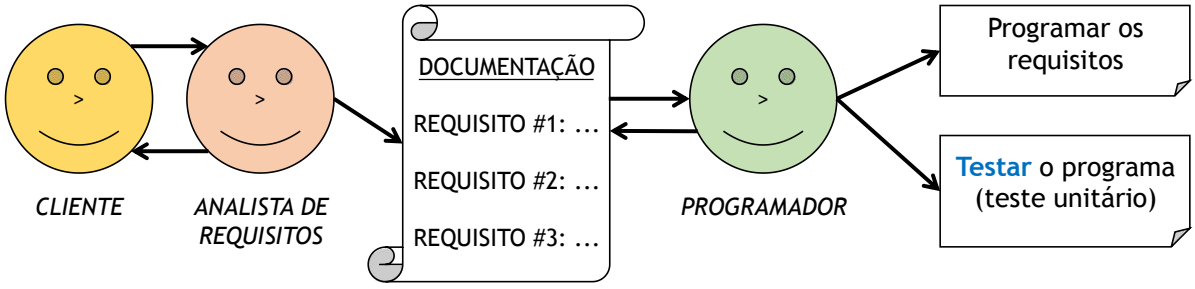
alan.souza@unama.br

2020

1. Fundamentos de Teste de Software

Objetivos do teste de software

a. Demonstrar ao desenvolvedor e ao cliente que o software atende seus requisitos; deve haver pelo menos um teste para cada requisito do documento de requisitos (Sommerville, 2011).



1. Fundamentos de Teste de Software

Objetivos do teste de software

b. Descobrir situações em que o software se comporta de forma incorreta, indesejável ou de forma diferente das especificações(Sommerville, 2011).

Preocupa-se em eliminar:

. panes (tela azul, travamento, erros enigmáticos...),

. interações indesejáveis com outros sistemas,

. processamento incorreto de dados,

. corrupção de dados.

Esses fatores geram insatisfação do cliente, perda de tempo, perda de receita, perda de credibilidade do mercado. Isso tanto para a empresa dona do sistema quanto para quem o adquiriu.

1. Fundamentos de Teste de Software

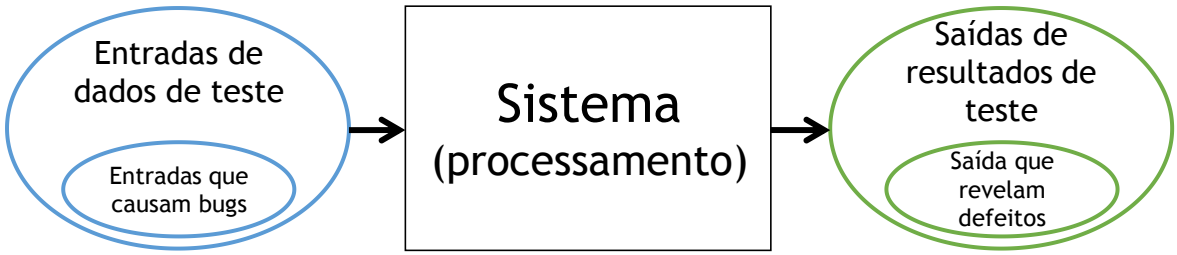
1. Fundamentos de Teste de Software

Lembre-se:

Os testes podem mostrar apenas a presença de erros e não sua ausência (Dijkstra et al., 1972)

1. Fundamentos de Teste de Software

Esquema geral de teste de software:



- Algumas entradas vão gerar saídas corretas e outras entradas vão gerar saídas erradas. Dessa forma, o defeito é achado e deve ser corrigido.

1. Fundamentos de Teste de Software

Validação e Verificação (V&V):

a. Validação:

. estamos construindo o produto **certo**? (Boehm, 1979).

b. Verificação:

. estamos construindo o produto de maneira **certa**? (Boehm, 1979).

• Servem para:

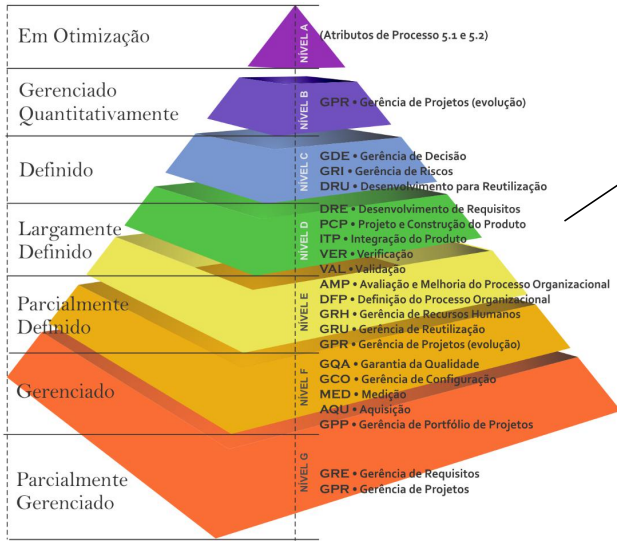
. garantir que o software atenda as expectativas do cliente;

. garantir que o software atenda os requisitos funcionais e não funcionais.

• Inspeções e revisões servem para V&V também.

1. Fundamentos de Teste de Software

MPS.BR - Melhoria de Processo do Software Brasileiro

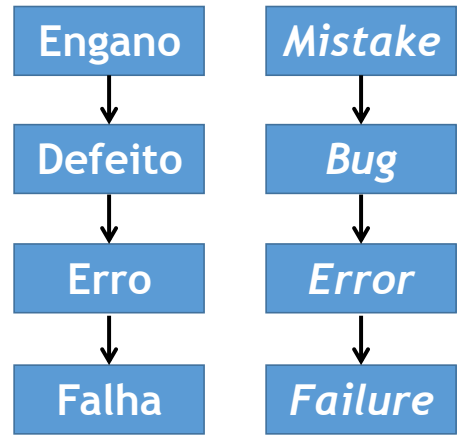


No nível D, temos verificação e validação, ou seja, testes.

1. Fundamentos de Teste de Software

Nomenclaturas:

1. Um **engano** do programador
2. Introduziu um **defeito** no código
3. Quando o software foi executado, gerou-se um **erro**
4. E o software **falha**
5. Mas o erro e a falha só serão detectados se as entradas que as geram forem usadas.



Teste de Software

Prof. Dr. Alan Souza

alan.souza@unama.br

2020

2. Tipos de Testes

2.1 Teste de desenvolvimento
- 2.1.1 Testes unitários
- 2.1.2 Escolha de casos de teste unitário
- 2.1.3 Testes de componentes
- 2.1.4 Testes de sistema

2.2 Desenvolvimento Dirigido a Testes (DDT) ou Test-Driven Development (TDD)

Desenvolvedores

2.3 Testes de *release* (aceitação)
- 2.3.1 Testes baseado em requisitos
- 2.3.2 Testes de cenário
- 2.3.3 Testes de desempenho/estresse

Analistas + Usuários

2.4 Testes de usuário — Usuários

2. Tipos de Testes

2.5 Mock de objetos
2.6 Testes de integração
2.7 Testes de serviços web
2.8 Testes de usabilidade/acessibilidade

E outros...

2.1 Teste de desenvolvimento

- Incluem todas as atividades de testes que são realizadas pela equipe de desenvolvimento do sistema;
- O programador é o testador (pode variar);
- Objetivo: descobrir bugs e garantir a regressão;
- Depuração: programar uma funcionalidade; testar; se der problema, corrigir; senão, tentar melhorar o código (refatoração) ou finalizar a programação da funcionalidade.

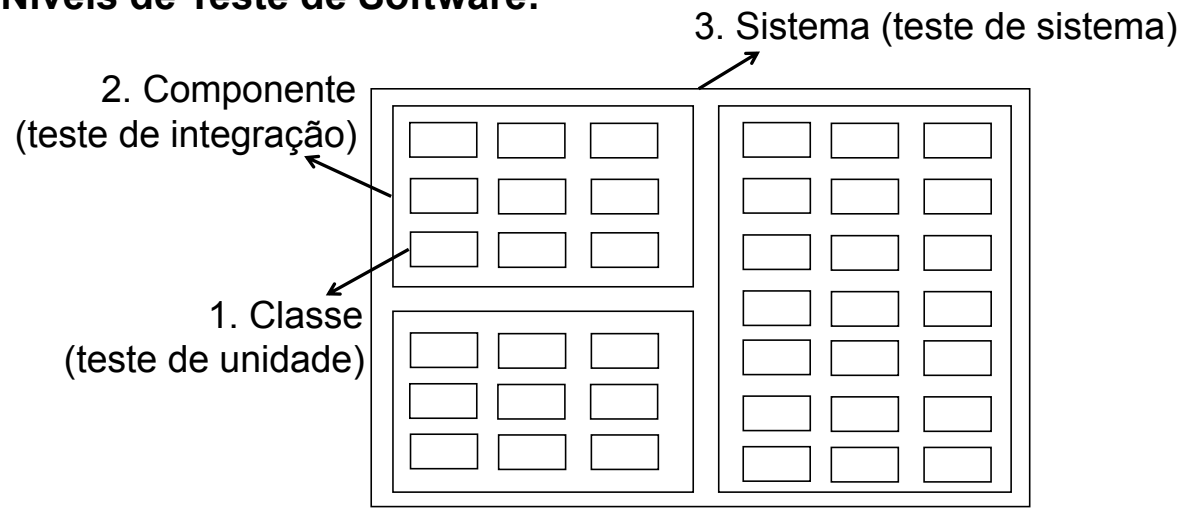
2.1 Teste de desenvolvimento

Teste de Regressão:

- Trata-se de uma bateria de testes automatizados que deve ser executada toda vez que ocorrer remoção, adição ou alteração de uma funcionalidade no sistema!
- Pode ser manual também, porém demora mais.

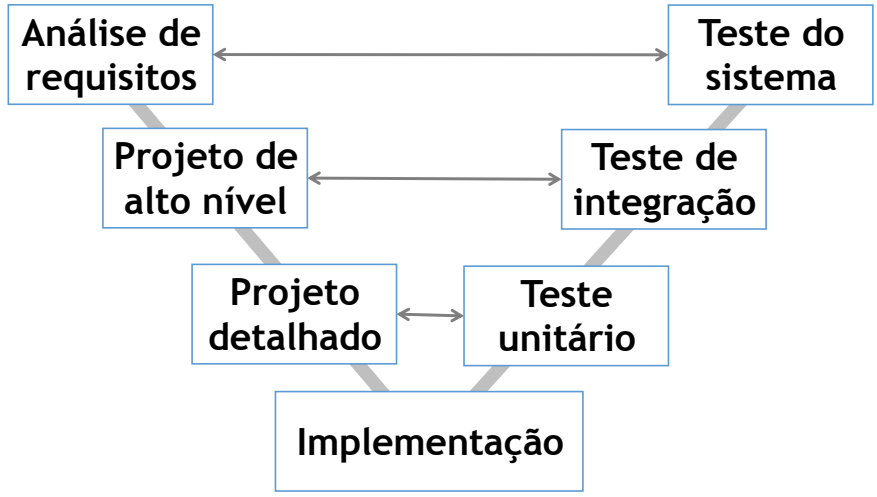
2.1 Teste de desenvolvimento

Níveis de Teste de Software:



2.1 Teste de desenvolvimento

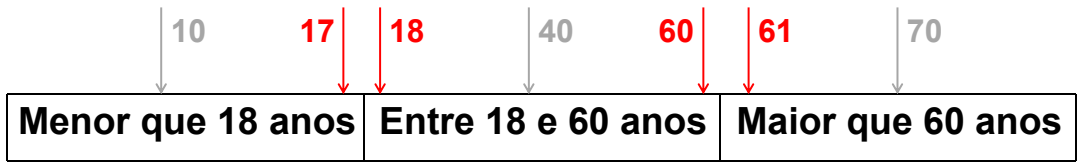
Níveis de Teste e Fases de Desenvolvimento (Modelo em V):



2.1 Teste de desenvolvimento

Partições de Equivalência:

- Que dados devem ser usados para testar?
- Ex: Se cliente for menor de idade, não permitir cadastro no sistema; se for maior de idade e menor que 60 anos, permitir cadastro sem restrições; se for idoso(a), permitir cadastro com restrições.



2.1.1 Teste unitário

- É o processo de testar os componentes do programa, como métodos ou classes de objetos. Nem tudo deve ou poderá ser testado...
- O programador/testador deve:

. testar todas as operações associadas ao objeto;

. definir e verificar o valor de todos os atributos associados ao objeto;

. colocar o objeto em todos os estados possíveis, simulando todos os eventos que causam mudança de estado.

2.1.1 Teste unitário

Esse tipo de teste pode/deve ser automatizado.

Em Java, utiliza-se o framework JUnit;

Em C#, o NUnit;

Em Javascript, existe o QUnit ou o Jest;

Em PHP, o PHPUnit.

Portanto, para cada linguagem, haverá uma ferramenta respectiva para automatizar testes unitários...

2.1.1 Teste unitário

Exemplo - Classe Numero

Requisito:

O cliente está pagando por um programa que solicita um número inteiro pelo teclado e que imprima se ele é positivo, negativo ou nulo.

2.1.1 Teste unitário

Exemplo - Classe Numero

```
1   package teste2020;
2
3   public class Numero {
4       public String posNegNulo(int n) {
5           return "nulo";
6       }
7   }
```

incompleta!

2.1.1 Teste unitário

O JUnit possui diversas facilidades para executar nossa bateria de testes. Para usá-lo no Netbeans, é preciso:

. Criar uma classe de teste: clicar com o **botão direito em cima do projeto**, opção **Novo**, opção **Teste JUnit**. Dar um nome e Finalizar.

. Analisando a classe de teste:

.. Importações (a do Assert é estática)

.. Anotação **@Test**: trata-se de um caso de teste, deve ser public e retorno void;

.. Método **assertEquals(VALOR ESPERADO, VALOR RETORNADO);**

Se o Esperado == Retornado, o teste passa **(verde)**

Se o Esperado != Retornado, o teste falha **(vermelho)**

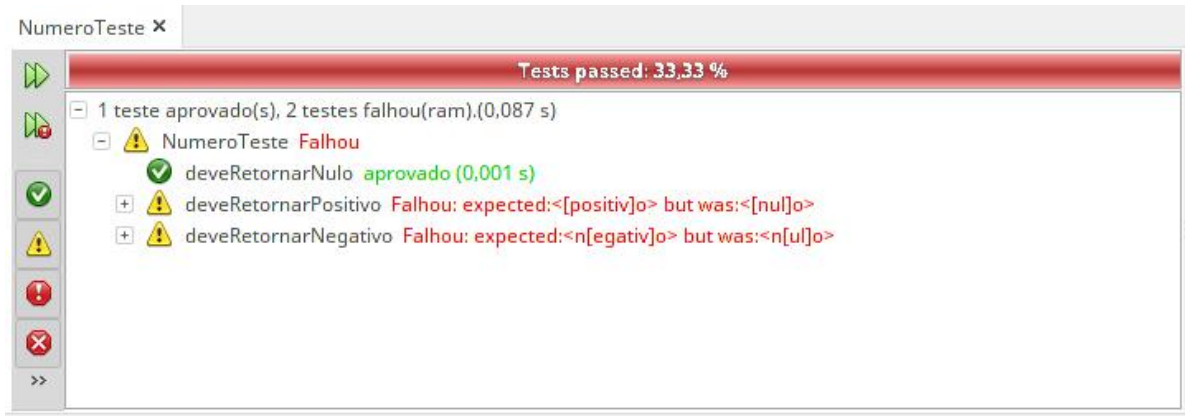
Exemplo de Classe de Teste para a classe Numero

É a bateria de testes...

```java
import org.junit.Test;
import static org.junit.Assert.*;
import teste2020.Numero;

public class NumeroTeste {
    Numero n;
    public NumeroTeste() {
        n = new Numero();
    }

    @Test
    public void deveRetornarNulo() {
        assertEquals("nulo", n.posNegNulo(0));
    }

    @Test
    public void deveRetornarPositivo() {
        assertEquals("positivo", n.posNegNulo(5));
    }

    @Test
    public void deveRetornarNegativo() {
        assertEquals("negativo", n.posNegNulo(-5));
    }
}
```

2.1.1 Teste unitário

Para executar o teste, clique com o botão direito em cima da classe de teste e escolha a opção “Executar Arquivo” ou selecione a classe de Teste e pressione as teclas **Shift+F6**.



2.1.1 Teste unitário

a.1 Exemplo prático #1:

Desenvolver um programa em Java que receba um número inteiro e que mostre se ele é negativo, positivo ou nulo. Exemplos de funcionamento (sem usar JUnit por enquanto):

Exemplo 1:
Entrada: 3
Saída: positivo

Exemplo 2:
Entrada: -10
Saída: negativo

Exemplo 3:
Entrada: 0
Saída: nulo

Exemplo 4:
Entrada:
Saída: indefinido

2.1.1 Teste unitário

a.2 Exemplo prático #1 (continuação...):

Clonar o projeto e implementar testes semi-automatizados (sem JUnit).

2.1.1 Teste unitário

a.3 Exemplo prático #1 (continuação...):

Clonar o projeto novamente e implementar testes automatizados (com JUnit).

2.1.1 Teste unitário

b. Exemplo prático #2:

Criar um programa em Java que receba números inteiros separados por vírgula, que efetue e mostre o somatório dos números informados para o usuário. Exemplos de funcionamento:

Exemplo 1:
Entrada: 3,7,1,2,-4
Saída: 9

Exemplo 2:
Entrada: 3
Saída: 3

Exemplo 3:
Entrada:
Saída: 0

Exemplo 4:
Entrada: 1,3,9,
Saída: 13

Exemplo 5:
Entrada: 5, 1, 9
Saída: 15

Usar JUnit