

INTRODUÇÃO AO TESTE DE SOFTWARE EM JAVA

**PROF. ME.
HÉLIO
ESPERIDIÃO**



Teste é uma forma operacional de checar um sistema através de experimentos



Execucutar de um sistema com base em determinados critérios



Comparar os resultados das execuções com resultados esperados



Veredito: ok ou não

INTRODUÇÃO



Com testes é possível detectar a existência, mas não é possível garantir a ausência de erros.



Erros sempre vão existir



Grandes empresas de desenvolvimento não estão livres de erros.

INTRODUÇÃO

INTRODUÇÃO

- Teste deve ser planejado antecipadamente e realizado sistematicamente.
- Defina um “template”, ou seja um conjunto de passos ao qual é possível alocar técnicas de projeto de **casos de teste** e estratégias de teste específicos.



TRÊS CONCEITOS BÁSICOS SOBRE TESTES



Defeito: implementação incorreta feita em uma aplicação.



Erro: manifestação desse defeito ou falha.



Falha: comportamento externo que resulta em problemas de execução de um sistema (por exemplo, problemas de rede ou hardware).

O Processo de Teste deve ser revisto continuamente, de forma a ampliar sua atuação e possibilitar aos profissionais uma maior visibilidade e organização dos seus trabalhos

Rever resulta numa maior agilidade e controle operacional dos projetos de testes.

PROCESSO



A depuração é a parte mais imprevisível do processo de teste.



Erro que indique uma discrepância de 0,01% entre resultados esperados e reais pode demorar uma hora, um dia ou um mês para ser diagnosticado e corrigido.

DEPURAÇÃO

TÉCNICAS DE TESTE DE SOFTWARE

Testes são realizados para demonstrar que cada função é totalmente operacional
(teste de caixa preta - “black box”)



Conhecendo-se o funcionamento interno de um produto, testes podem ser realizados para garantir que a operação interna de um produto tem um desempenho de acordo com as especificações e que os componentes internos foram adequadamente postos à prova (teste de caixa branca - “white box”)

TESTE DE CAIXA PRETA

Teste de caixa preta refere-se aos testes realizados nas interfaces do SW

A entrada é adequadamente aceita e a saída é corretamente produzida com a integridade das informações externas mantida.



Baseia-se em um minucioso exame dos detalhes procedimentais, através da definição de todos os caminhos lógicos possíveis.



Infelizmente estes testes apresentam problemas, levaria a um tempo muito grande.



Entretanto este tipo de teste não pode ser desprezado como pouco prático, podendo-se optar por um número limitado de opções

TESTE DE CAIXA BRANCA

VISÃO DA QUALIDADE

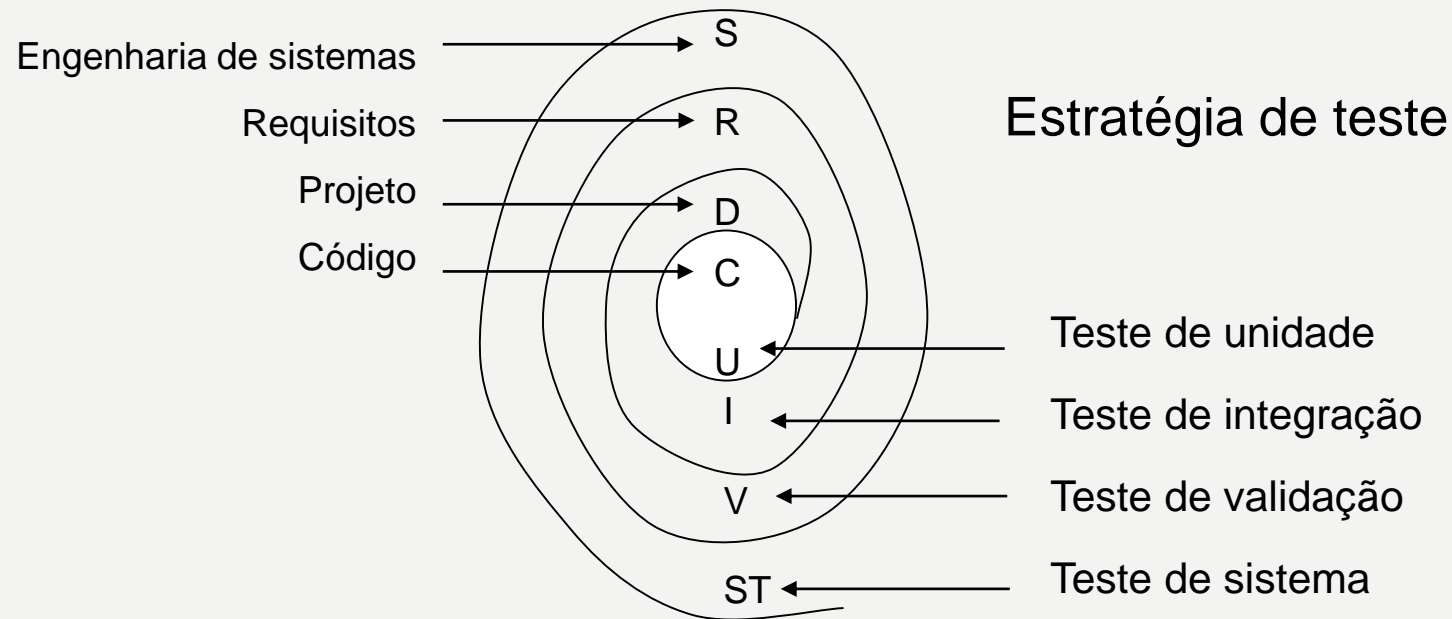
Teste x Verificação x Validação

- Verificação: “Estamos construindo certo o produto?”
- Validação: “Estamos construindo o produto certo?”

Teste x Qualidade

- Qualidade é um conceito mais amplo
- Teste gera informação sobre qualidade do produto

ESTRATÉGIAS DE TESTE DE SOFTWARE





Verificação da menor unidade de projeto de SW - o módulo.



Baseia-se quase sempre na técnica de caixa branca e pode ser realizado em paralelo para múltiplos módulos.

TESTES DE UNIDADE



O objetivo é, a partir dos módulos testados no nível de unidade, construir a estrutura de programa que foi determinada pelo projeto realizando-se ao mesmo tempo, testes para descobrir erros associados a interfaces



Entradas e saídas entre módulos devem se compatibilizar.

TESTES DE INTEGRAÇÃO



São definidas expectativas razoáveis na Especificação de Requisitos de SW, que descreve todos os atributos do SW visíveis ao usuário.



A validação é bem-sucedida quando o SW funciona de uma maneira razoavelmente esperada pelo cliente.

TESTES DE VALIDAÇÃO



É uma série de diferentes testes, cujo propósito primordial é pôr completamente à prova o sistema baseado em computador.



Módulos relacionados



Imagine em um ERP

TESTES DE SISTEMA



Teste de recuperação: força a falhar de diversas maneiras e verifica se a recuperação é adequadamente executada.



Teste de segurança: verifica se todos os mecanismos de proteção embutidos, verifica acessos indevidos.



Teste de estresse: Exige recursos em quantidade. Essencialmente o analista tenta destruir o programa DDOS.



Teste de desempenho: é idealizado para testar o desempenho de “runtime”.

TESTE DE SISTEMA

TEST-DRIVEN DEVELOPMENT (TDD)

- Desenvolvimento guiado pelos testes
 - Só escreva código novo se um teste falhar
 - Refatore até que o teste funcione

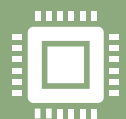




Necessário uso de ferramentas de suporte



Existem vários tipos de ferramentas



Record & Play

Registram ações de usuários na interface (através de mouse e teclado) e permitem repetir as operações

Para testes de aceitação, por exemplo



Geração de grandes quantidades de dados

Para testes de stress

AUTOMAÇÃO



Definição de uma proposta de testes baseada nas expectativas do Cliente



Possibilidade de dimensionar a equipe e estabelecer um esforço de acordo com as necessidades.

PLANEJAMENTO DE TESTES

Existem diversas categorias de testes, tais como de módulos, de integração, de sistema e de validação (SOMMERVILLE, 2016).

Um dos testes iniciais que podem ser feitos é o unitário, cujo objetivo é verificar se os métodos estão funcionando.

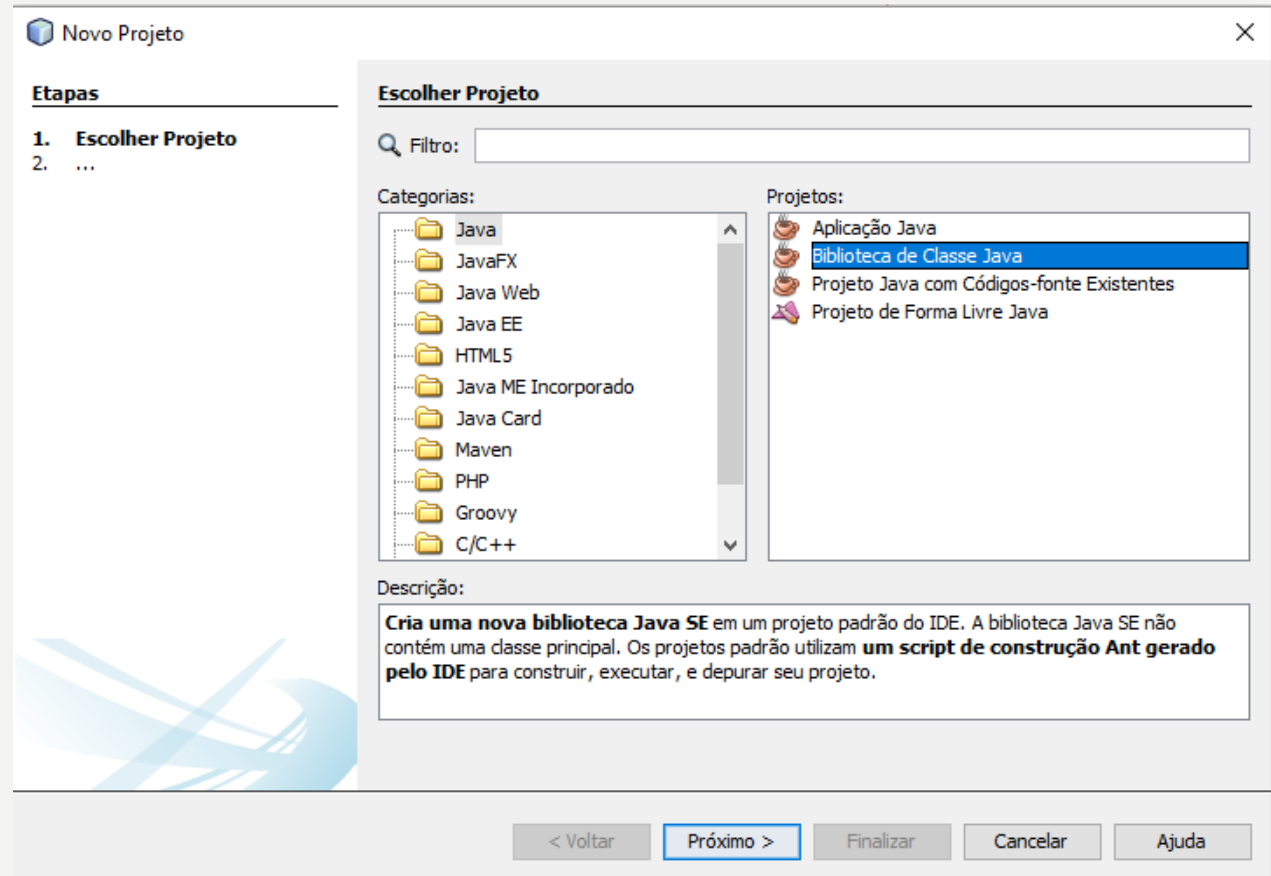
Uma das ferramentas utilizadas para realizar esse tipo de teste é o JUnit, com o qual é possível fazer diversas avaliações para confirmar se o código está funcionando corretamente (TAHCHIEV, 2010).

O teste unitário pode ser comparado a testar e validar todas as peças de um carro antes de fazer a montagem. Se as peças estão funcionando corretamente, as chances de a montagem funcionar aumentam

TESTES UNITÁRIOS EM JAVA

CRIE UM PROJETO

- O conteúdo desta página se aplica ao NetBeans IDE 7.2, 7.3, 7.4 e 8.0
- Criando o Projeto
- Criando o Projeto de Biblioteca de Classe Java



DEFINA O NOME DO PROJETO

Novo Biblioteca de Classe Java

5

Escolher Projeto

Nome e Localização

Nome do Projeto:

Localização do Projeto:

Pasta do Projeto:

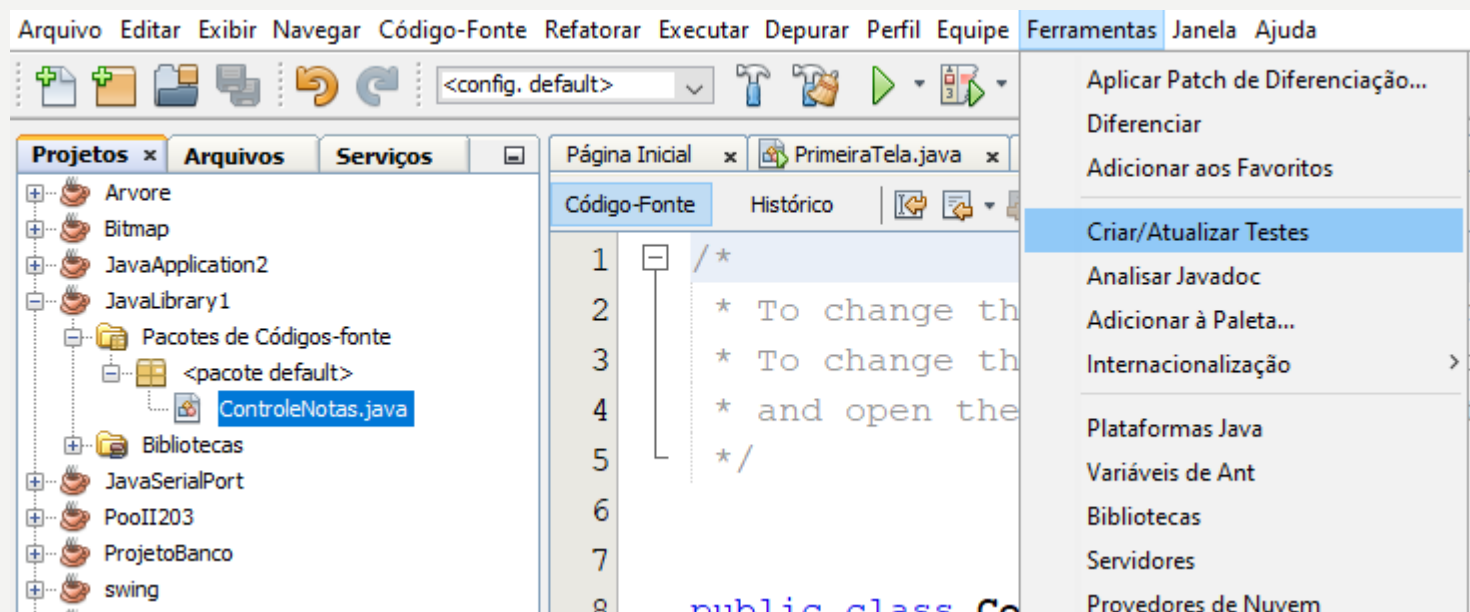
Usar Pasta Dedicada para Armazenar Bibliotecas

Pasta Bibliotecas:

Usuários e projetos diferentes podem compartilhar as bibliotecas de compilação (consulte a Ajuda para obter detalhes).

< Voltar Próximo > Finalizar Cancelar

COPIE AS CLASSES PARA O PROJETO



IMPLEMENTE OS TESTES

Criar Testes [X]

Classe a Testar: ControleNotas

Nome da Classe:

Localização:

Framework:

Testes de Integração

Geração de Código

Níveis de Acesso do Método	Código Gerado
<input checked="" type="checkbox"/> Público	<input checked="" type="checkbox"/> Inicializador de Testes
<input checked="" type="checkbox"/> Protegido	<input checked="" type="checkbox"/> Finalizador de Testes
<input checked="" type="checkbox"/> Pacote Privado	<input checked="" type="checkbox"/> Inicializador da Classe de Teste
	<input checked="" type="checkbox"/> Finalizador da Classe de Teste
	<input checked="" type="checkbox"/> Corpos de Métodos Default
	Comentários Gerados
	<input checked="" type="checkbox"/> Comentários Javadoc
	<input checked="" type="checkbox"/> Dicas de Código-fonte

RODE OS TESTES

- Faça a observação dos casos de teste

```
@test
public void testCalculaNotaFinal() {
    System.out.println("calculaNotaFinal");
    ControleNotas instance = null;
    double expectedResult = 0.0;
    double result = instance.calculaNotaFinal();
    assertEquals(result, expectedResult, 0.0);
    // TODO verifica o código de teste gerado e remove a
    fail("O caso de teste \u00e9 um prot\u00f3tipo.");
}
}
```

ControleNotasNGTest

JavaLibrary1 (test) x Resultados do Teste x

suite x

0,01

nenhum teste aprovado(s), 1 teste falhou(ram).(0,004 s)

Command line test Falhou

ControleNotasNGTest.testCalculaNotaFinal Falhou: java.lang.NullPointerException

[TestNG] Running:
Command line suite
calculaNotaFinal
Command line suite

```
public void testCalculaNotaFinal() {
    System.out.println("calculaNotaFinal");
    ControleNotas instance = new ControleNotas(1,2,3);
    double expectedResult = 2.0;
    double result = instance.calculaNotaFinal();
    assertEquals(result, expectedResult, 0.0);
    // TODO verifica o código de teste gerado e remove a chamada default para falha.
    fail("O caso de teste \u00e9 um prot\u00f3tipo.");
}
}
```

ControleNotasNGTest > testCalculaNotaFinal > expectedResult >

JavaLibrary1 (test) x Resultados do Teste x

suite x

0,00 %

nenhum teste aprovado(s), 1 teste falhou(ram).(0,005 s)

Command line test Falhou

ControleNotasNGTest.testCalculaNotaFinal Falhou: java.lang.AssertionError: expected [2.0] but fo

[TestNG] Running:
Command line suite
calculaNotaFinal
Command line suite
Total tests run: 1, Failures: 1, Skips: 0