

# *Testes de Software*



Prof<sup>a</sup> Flavia Garcia  
Senac Rio

# Competências

## **Competência:**

- Planejar e implementar testes de software.

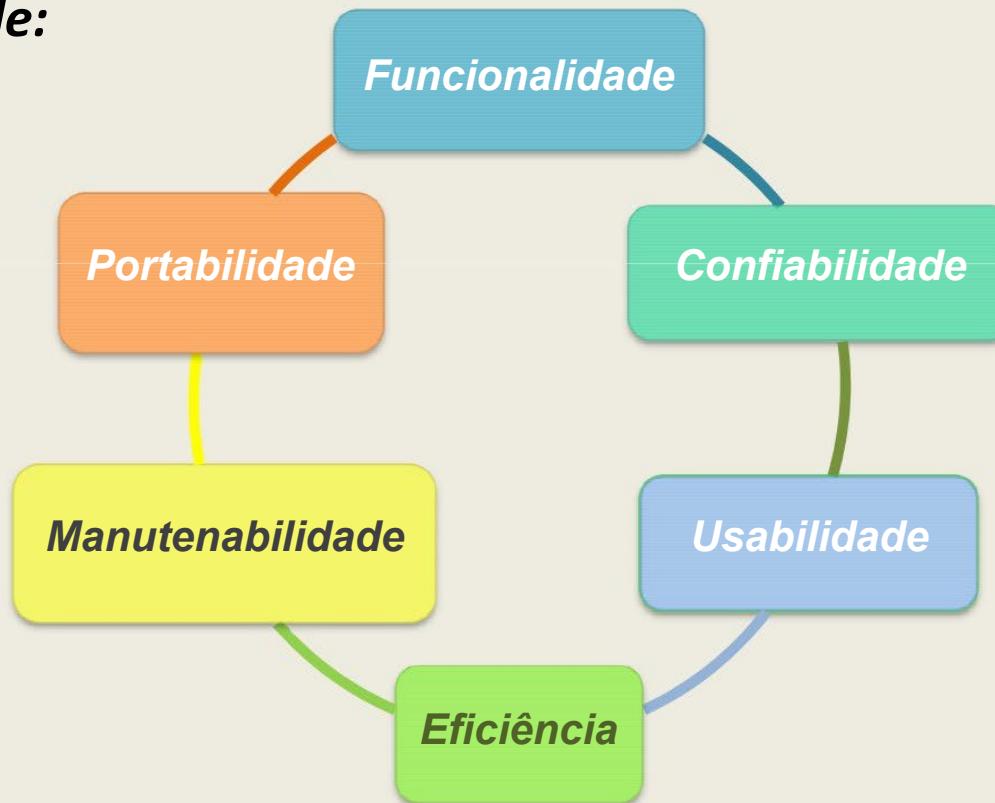
## **Bases tecnológicas (conteúdos):**

- Fundamentos de Teste de Software
- Planejamento de Testes de Software
- Processos de Teste de Software
- Técnicas de Validação de Requisitos para Teste de Software
- Técnicas de Testes de Software Estruturais e Funcionais
- Cenários de Teste
- Introdução a Automação de Testes de Software
- Controle de Falhas
- Relatórios de Teste de Software
- Principais Indicadores de Testes de Software

# *Como adquirir qualidade em um software?*



**A Norma ISO 9126 define as seguintes características para qualidade:**



# Introdução

## *O que é teste de software?*

Os testes são realizados com a intenção de descobrir erros e defeitos em um sistema. [Myres, 2004]

Os testes de software podem ser usados para mostrar a presença de defeitos, mas nunca para mostrar a ausência deles. [Dijkstra, 1972]

Os testes de software servem para medir a confiabilidade de um sistema: à medida que poucos defeitos são encontrados em um determinado tempo, o software é considerado mais confiável.

# ***Por que testar é necessário?***

Para assegurar que as necessidades dos usuários estejam sendo atendidas.

Porque é provável que o software possua defeitos.

Desenvolvedor já alocado para outro projeto teria que resolver muitos bugs de projetos anteriores em produção.

Porque falhas podem custar muito caro.

Para avaliar a qualidade do software.

# **Conceitos Básicos de Teste**

## **Artefatos de Teste**

- *todo o conjunto de documentação gerado pelo processo de teste de software.*

## **Caso de Teste**

- *é composto por um conjunto de entradas, por passos de execução e um resultado esperado.*

## **Roteiro de Teste**

- *É composto por um conjunto de casos de teste definidos para uma determinada especificação.*

***Requisitos***

- *regras de negócio do sistema.*

***Testar***

- *descobrir falhas através da execução do sistema.*

***Bug***

- *é um defeito encontrado no sistema em execução.*

## *Confiabilidade do Software*

*Confiabilidade do Software é a probabilidade que o software não causará uma falha no sistema por um tempo especificado, sob condições determinadas.*

## *O custo de um defeito*

*O custo da correção de um defeito tende a ser cada vez maior quanto mais tarde ele for descoberto. [Myres, 2004]*

# Fundamentos



Você acha  
importante realizar  
testes em um sistema  
que está sendo  
desenvolvido??

Não acredito que seja  
necessário

Nem considero entregar um  
software sem testar

# Fundamentos

Se não testarmos o sistema, como iremos garantir que cada item do projeto seja atendido conforme especificado?



Exemplo de itens que devem ser verificados no projeto

Requisitos Funcionais

Requisitos Não Funcionais

Regras de Negócio

Casos de uso

# Por que realizar testes no software?

A cartoon illustration featuring a boy with glasses and a purple cap, and a man sitting cross-legged on a globe while working on a laptop. The boy is speaking into a speech bubble, and three numbered callouts provide reasons for testing.

Por que testar ?!  
veja alguns  
motivos:

- 1 **Economia:**  
Reduz o tempo  
gasto com  
retrabalho  
relacionado às  
manutenções  
corretivas.
- 2 **Qualidade:**  
Os testes ajudam a  
garantir que o  
produto atendeu  
todas as  
especificações.
- 3 **Negócio:**  
Falhas percebidas  
pelo cliente são um  
risco ao negócio.  
Testes podem  
identificá-las a  
tempo.

Sem dúvida o impacto do  
desenvolvimento de  
software sem a correta  
aplicação de testes pode  
deixar o cliente bem  
insatisfeito...

# Conceitos incorretos

É importante definir corretamente Teste de Software!

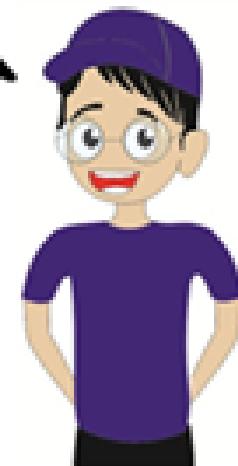
Uma visão distorcida deste conceito pode levar a um processo de testes ineficiente.

Veja:



O teste de software é o processo de demonstrar que os defeitos não estão presentes.

Errado! O que queremos com o teste é justamente encontrar erros.





Se o testador tiver por objetivo provar que o programa funciona, corre o risco de ser tendencioso na escolha dos casos de teste, ignorando situações que poderiam apontar defeitos.

Teste de software é a atividade de executar um programa ou sistema com a intenção de encontrar defeitos

Myers, 1979

Veja alguns conceitos mais adequados:

Conferir se o software está fazendo o que deveria fazer, de acordo com os seus requisitos, e não está fazendo o que não deveria fazer



Fonte:

NETO, Michele. Vídeo Aula1 – Conceitos iniciais em  
teste de Software. Disponível em:  
<http://www.youtube.com/watch?v=7kec8BT8gU4>.

# Terminologia



# *Erro, Defeito e Falha*

**Erro:** é uma ação humana que produz um resultado incorreto.

**Defeito:** A manifestação de um erro no software.

Também conhecido como Bug

**Falha:** quando o sistema se comporta de forma inesperada devido ao defeito.

# *Erro, Defeito e Falha*

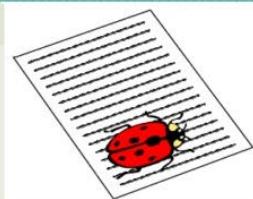


*Uma pessoa ...  
comete um  
**erro**...*

*que cria um  
**defeito** no  
software...*



*...que pode  
causar uma  
**falha** na  
operação.*

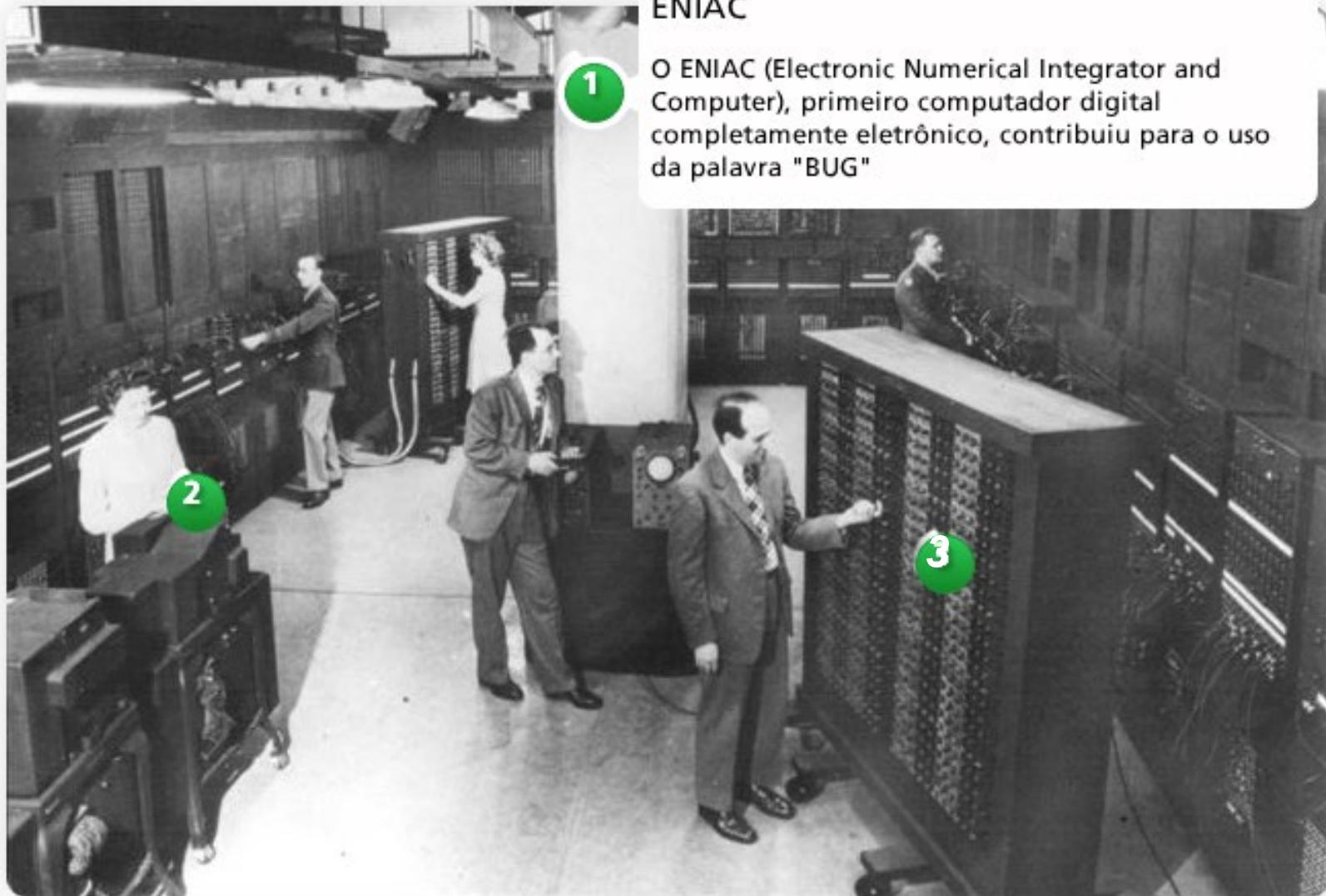


# Terminologia – *Bug* (*Curiosidade*)



O uso da palavra “Bug” na informática surgiu por causa de um problema em um supercomputador da Universidade de Harvard (EUA), nos anos de 1960. A máquina parou de funcionar e os técnicos não conseguiram descobrir a causa. Quando desmontaram o computador, encontraram um inseto (bug, em inglês) morto no meio dos circuitos, interrompendo a corrente elétrica.

## Curiosidade BUG...



# Terminologia



Mas afinal, qual é a diferença entre engano, defeito, erro e falha?

O foguete Ariane V explodiu em 1996. O software de controle do vôo indicou uma direção errada ao foguete (**esta foi a falha**) devido a uma conversão incorreta de uma variável tipo real de 64 bits em um inteiro de 16 bits (**este foi o defeito**). Esta conversão produziu um erro de overflow, isto é, um valor errôneo de uma variável (**este foi o erro**), que ocasionou a resposta incorreta do software. O **engano** foi, provavelmente, cometido por um programador, que inseriu no software um comando de atribuição indevido.



## Engano



Ação humana que produz um resultado incorreto, como uma ação incorreta tomada pelo programador.

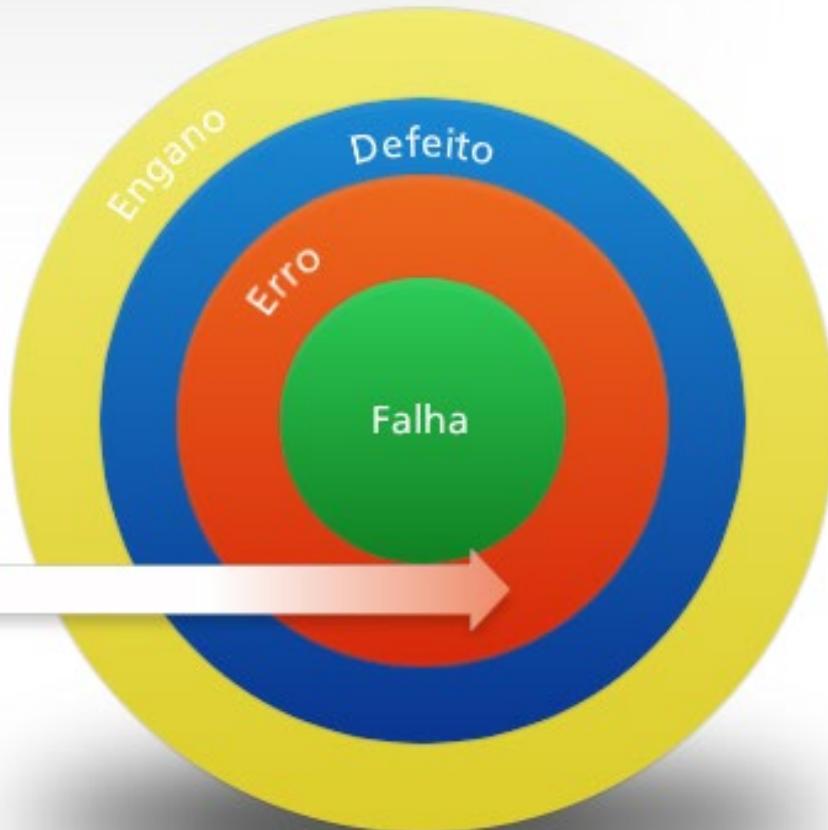
## Defeito

Passo, processo ou definição de dados incorretos, como por exemplo, uma instrução ou comando incorreto.



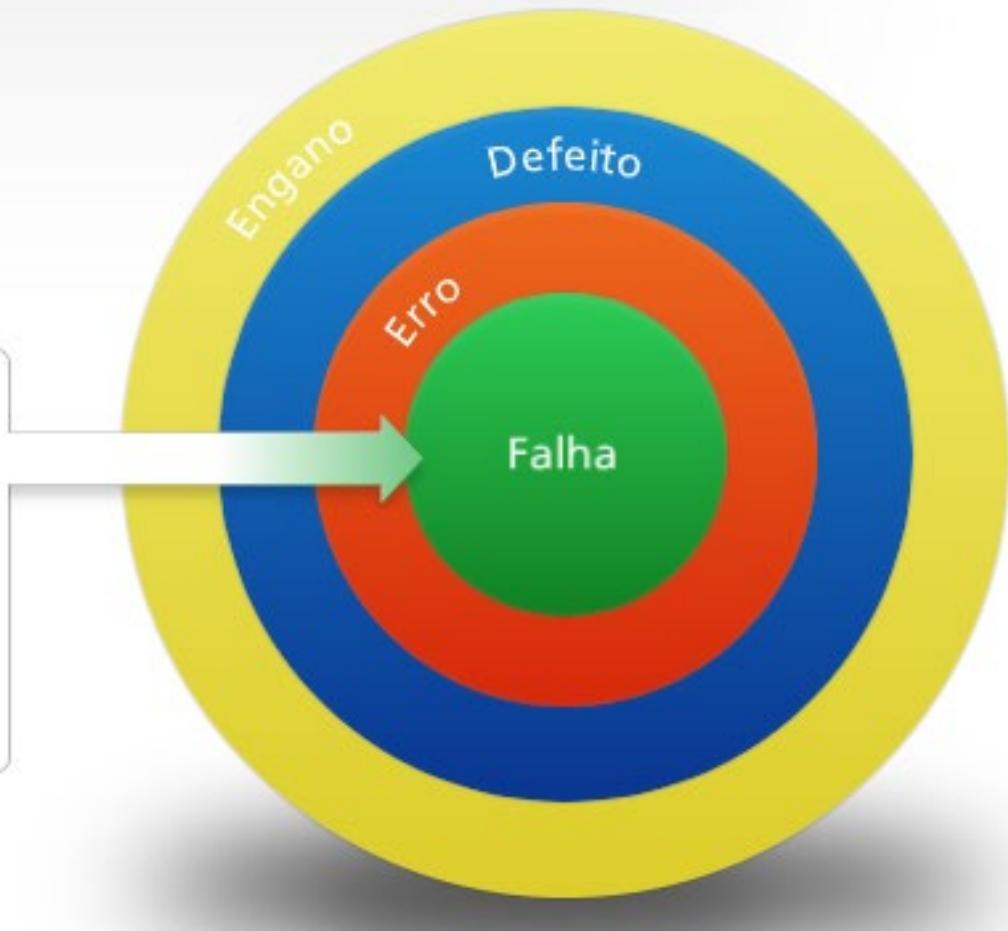
## Erro

Diferença entre o valor obtido e o valor esperado, ou seja, qualquer estado intermediário incorreto ou resultado inesperado na execução do programa constitui um erro.



## Falha

Produção de uma saída incorreta com relação à especificação. A falha é a percepção do erro.



# Resumindo ...

Teste de software é uma das atividades do processo de desenvolvimento de sistema de software que visa executar um programa de modo sistemático com o objetivo de encontrar falhas. Perceba que isto requer verificação e validação de software. Nesse sentido, definir quando as atividades de verificação e validação iniciam e terminam, como os atributos de qualidade serão avaliados e como os *releases* do software serão controlados, são questões que devem ser acompanhadas ao longo do processo de software.

Vale ressaltar que teste não deve ser a última atividade do processo de desenvolvimento de software. Ela ocorre durante todo o processo, como exemplificado na visão geral do processo RUP (*Rational Unified Process*).

# Resumindo ...

## **Engano:**

Ação humana que produz um resultado incorreto, como uma ação incorreta tomada pelo programador.

## **Defeito:**

Passo, processo ou definição de dados incorretos, como por exemplo, uma instrução ou comando incorreto.

## **Erro:**

Diferença entre o valor obtido e o valor esperado, ou seja, qualquer estado intermediário incorreto ou resultado inesperado na execução do programa constitui um erro.

## **Falha:**

Produção de uma saída incorreta com relação à especificação.

A falha é a percepção do erro.

Algumas terminologias que não podemos esquecer...

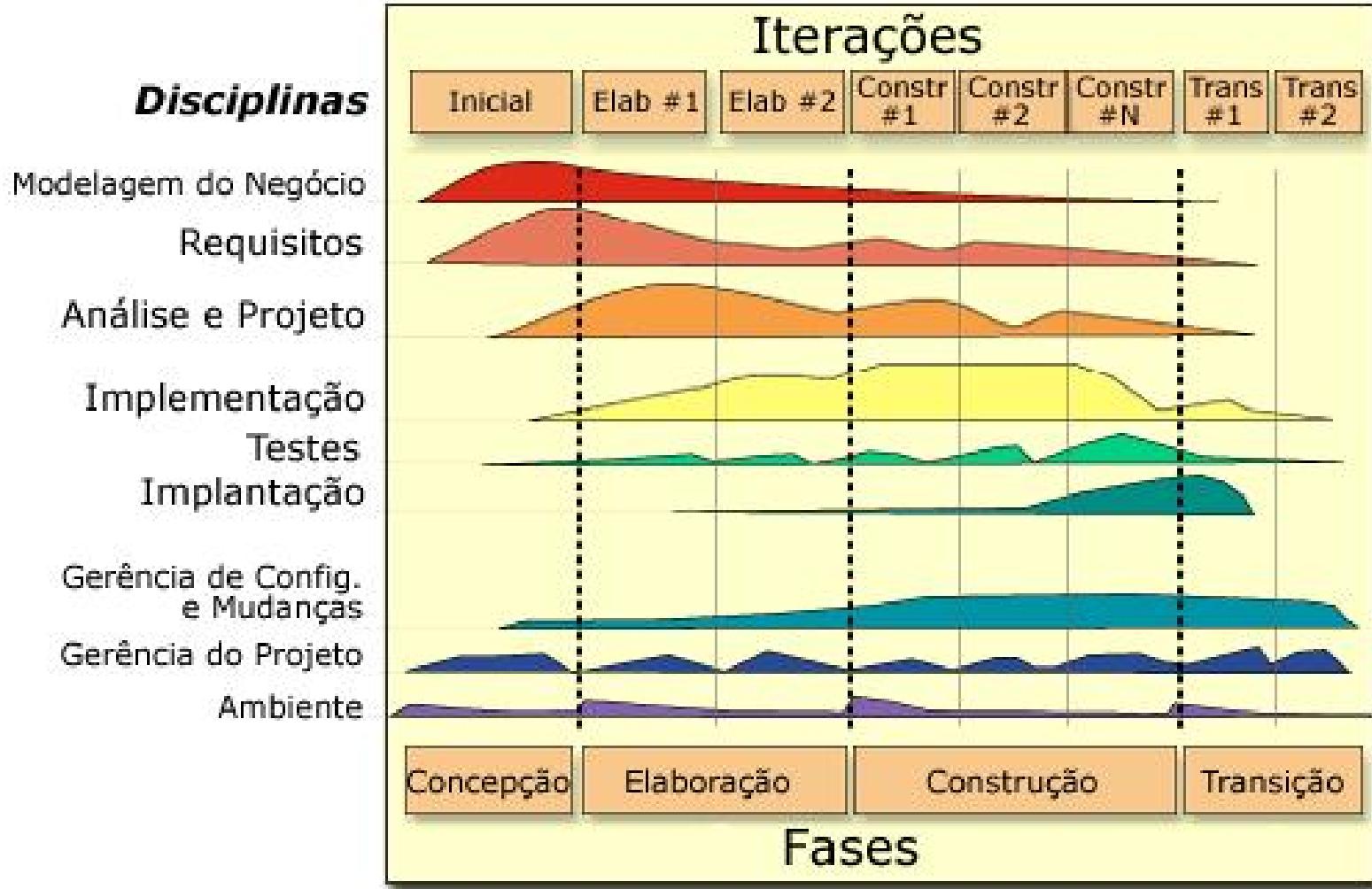


O estudo de testes de software geralmente é organizado em três categorias:

Técnicas de Teste ➔

Níveis de Teste ➔

Tipos de Teste ➔



# Planejamento de Testes

O plano de teste é um dos documentos produzidos na condução de um projeto. Ele funciona como:

- § Um ‘integrador’ entre diversas atividades de testes no projeto;
- § Mecanismo de comunicação para os stakeholders (i.e. a equipe de testes e outros interessados);
- § Guia para execução e controle das atividades de testes.

O plano de teste, que pode ser elaborado pelo gerente de projeto ou gerente de testes, visa planejar as atividades a serem realizadas, definir os métodos a serem empregados, planejar a capacidade necessária, estabelecer métricas e formas de acompanhamento do processo. Nesse sentido, deve conter:

- § Introdução com identificação do projeto (definições, abreviações, referências), definição de escopo e objetivos;
- § Conjunto de requisitos a serem testados;
- § Tipos de testes a serem realizados e ferramentas utilizadas;
- § Recursos utilizados nos testes;
- § Cronograma de atividades (e definição de marcos de projeto).

# Planejamento de Testes

- O Plano de Testes é o documento responsável por apresentar o planejamento para execução dos testes do software em desenvolvimento, incluindo a abrangência, abordagem, recursos e cronograma das atividades de teste.
- Tendo como referência a norma IEEE 829, padrão internacional que trata da documentação de teste de software, verifica-se a necessidade das seguintes informações em um plano de testes:

# Planejamento de Testes

**Em outras palavras, um plano de teste deve definir:**

1. Os itens a serem testados: o escopo e objetivos do plano devem ser estabelecidos no início do projeto.
2. Atividades e recursos a serem empregados: as estratégias de testes e recursos utilizados devem ser definidos, bem como toda e qualquer restrição imposta sobre as atividades e/ou recursos.
3. Os tipos de testes a serem realizados e ferramentas empregadas: os tipos de testes e a ordem cronológica de sua ocorrência são estabelecidos no plano.
4. Critérios para avaliar os resultados obtidos: métricas devem ser definidas para acompanhar os resultados alcançados.

Perceba que o planejamento é necessário a fim de antecipar o que pode ocorrer e, portanto, provisionar os recursos necessários nos momentos adequados. Isto significa coordenar o processo de teste de modo a perseguir a meta de qualidade do produto (sistema de software).

# Planejamento de Testes

## Exemplificando o Plano de Teste

O plano de teste contém um conjunto de informações que permite ao gerente de projeto não apenas coordenar as atividades de testes de um projeto, mas também monitorar seu progresso e verificar se o executado está em conformidade com o planejado.

A **Tabela** apresenta uma relação dos itens consideradas imprescindíveis em um plano de teste. A relação de itens não pressupõe a intenção de ser completo, mas de apontar os itens considerados como obrigatórios num plano de teste de uma instituição.

# Planejamento de Testes

<b><i>Itens de um Plano de Teste</i></b>	<b><i>Conteúdo</i></b>
<b><i>1. Introdução</i></b>	Contém uma identificação do projeto, descrição dos objetivos do documento, o público ao qual ele se destina e escopo do projeto a ser desenvolvido. Pode adicionalmente conter termos e abreviações usadas, além de informar como o plano deve evoluir.
<b><i>2. Requisitos a serem testados</i></b>	Esta seção descreve em linhas gerais o conjunto de requisitos a serem testados no projeto a ser desenvolvido, comunicando o que deve ser verificado. Exemplos de requisitos a serem testados são: desempenho, segurança, interface de usuário, controle de acesso, funcionalidades.

# Planejamento de Testes

<p><i>3. Estratégias e ferramentas de teste</i></p>	<p>Apresenta um conjunto de tipos de testes a serem realizados, respectivas técnicas empregadas e critério de finalização de teste. Além disso, é listado o conjunto de ferramentas utilizadas.</p>
<p><i>4. Equipe e infra-estrutura</i></p>	<p>Contém descrição da equipe e da infra-estrutura utilizada para o desenvolvimento das atividades de testes, incluindo: pessoal, equipamentos, software de apoio, materiais, dentre outros. Isto visa garantir uma estrutura adequada para a execução das atividades de testes previstas no plano.</p>

# Planejamento de Testes

## *5. Cronograma de atividades*

Contém uma descrição de marcos importantes (*milestones*) das atividades (incluindo as datas de início e fim da atividade). Apenas marcos relevantes devem ser listados, ou seja, aqueles que contribuirão nas atividades de testes. Por exemplo: projeto de testes, execução de testes ou avaliação de testes.

## *6. Documentação complementar*

Apresenta-se uma relação dos documentos pertinentes ao projeto.

# Processo de Teste - PDCA



# *Entendendo o Processo de Teste*



# *Entendendo o Processo de Teste*



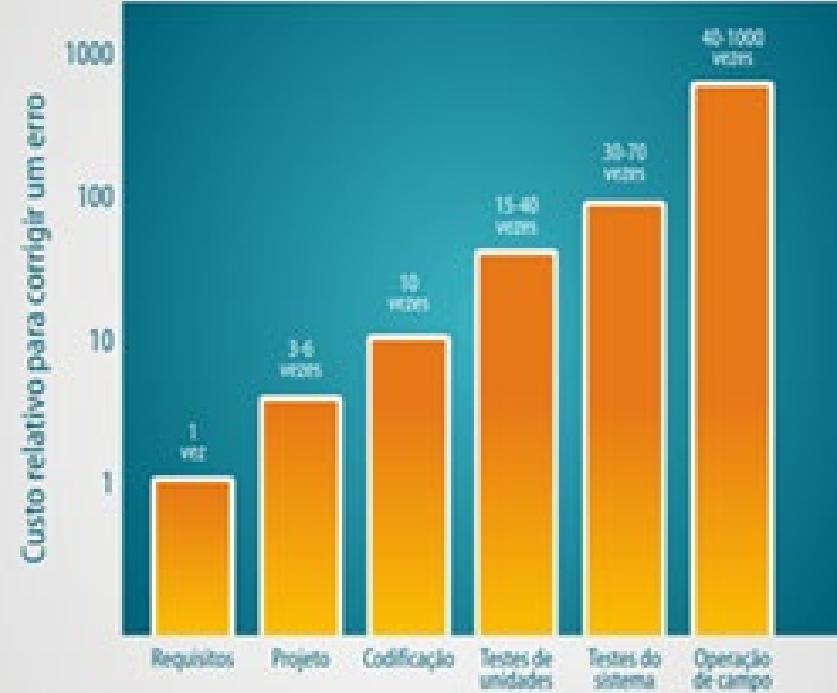
Será que o teste, visto apenas como uma das etapas finais do processo de desenvolvimento, é suficiente?



Claro que não!  
Os testes devem ocorrer em paralelo ao processo de desenvolvimento, executados por profissionais capacitados, usando metodologia apropriada.

# *Entendendo o Processo de Teste*

O projeto de teste de software deve começar paralelamente ao projeto de desenvolvimento, pois quanto antes o defeito for encontrado, menor será o custo de sua correção. Essa realidade é comprovada pela Regra 10 de Myers, que estabelece que o custo de correção de defeitos tende a aumentar quanto mais tarde o defeito é detectado.



# Processo de Teste de Software

*Um processo de teste básico deve contemplar:*

- Planejamento e controle;

- Análise e modelagem;

- Implementação e execução;

- Avaliação de critérios de saída e relatórios;

- Atividade de encerramento dos testes.

- Melhoria/Corretiva/Preventiva.

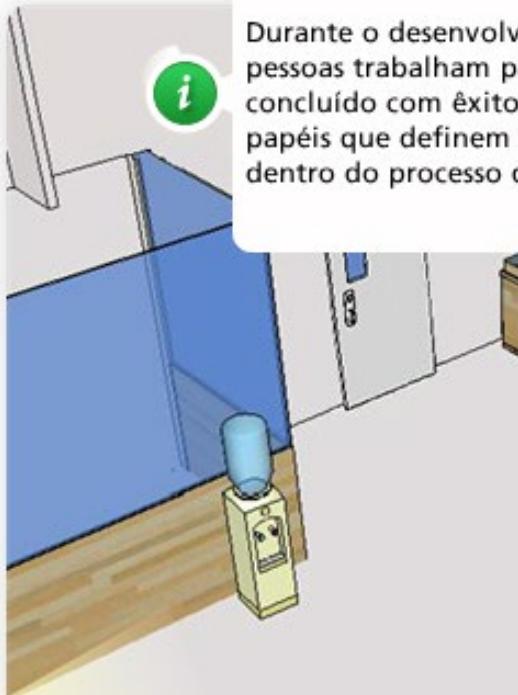
# Processo de Teste de Software (IEEE 12207)

- Prover uma metodologia de trabalho, ferramentas, e profissionais de forma a colaborar com o processo de desenvolvimento de software da organização, onde:

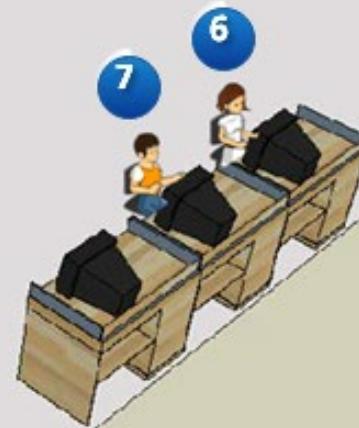
# Principais Funções :

## Quem participa do processo de teste?

### Introdução



Durante o desenvolvimento de um software várias pessoas trabalham para garantir que o projeto seja concluído com êxito, a essas pessoas são atribuídos papéis que definem o que cada uma deverá fazer dentro do processo de desenvolvimento.



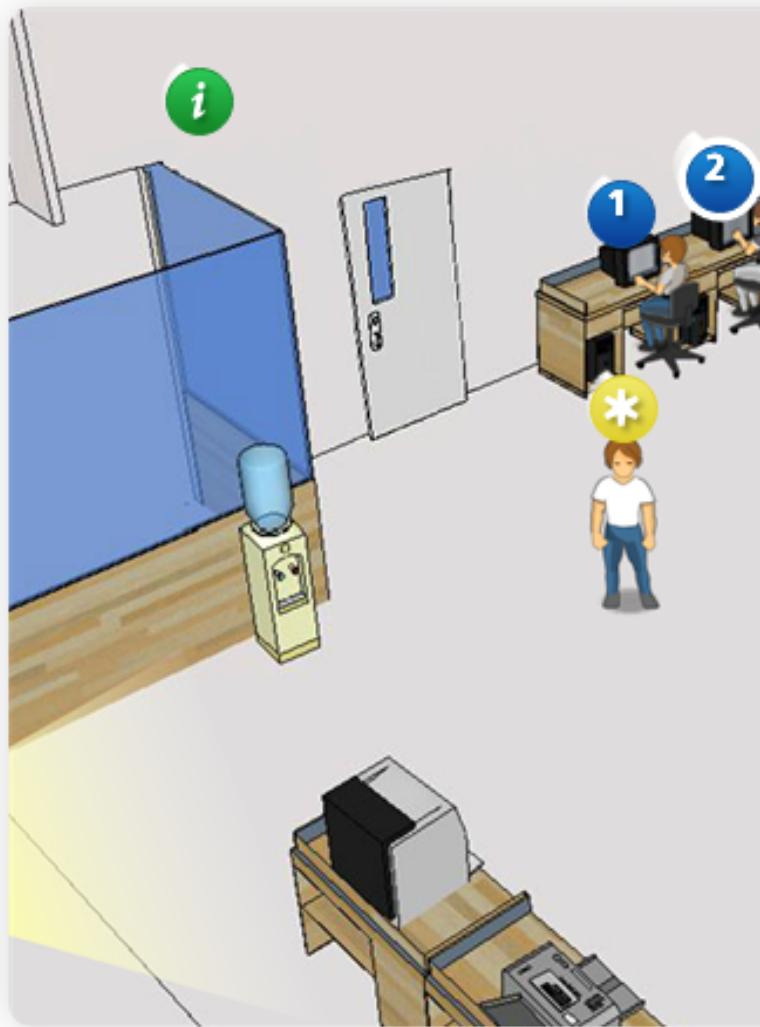
## Quem participa do processo de teste?

### Gestor de qualidade

Durante o ciclo de vida do desenvolvimento do software, eu respondo pelas ações de controle de qualidade dos produtos, ou seja, se estes foram desenvolvidos de acordo com as especificações acordadas e se atendem às expectativas dos usuários (ou clientes).



## Quem participa do processo de teste?

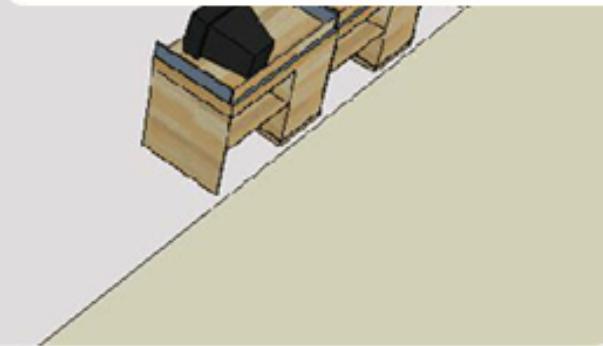


### Analista de sistemas

Como Analista de Sistemas acumulo maior número de responsabilidades no ciclo de desenvolvimento de software.

Minhas atividades são:

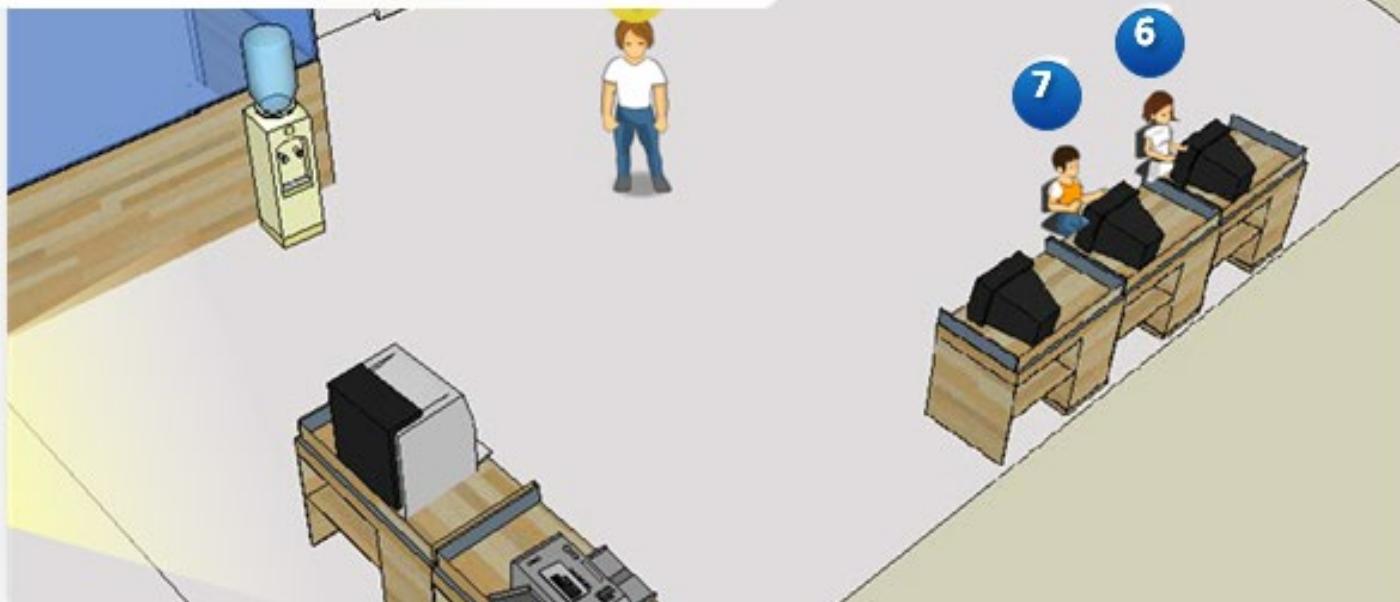
- Levantar e documentar os requisitos;
- Apoiar atividades de planejamento do projeto;
- Modelagem do software;
- Preparar e manter a documentação do projeto;
- Acompanhar as atividades de construção dos testes unitários;
- Planejar e executar os teste de integração.



Qu

## Programador

Como programador sou responsável pela codificação dos programas conforme a modelagem definida pelo analista. Porém, a construção dos testes de unidade também são minha responsabilidade, assim como a realização dos testes de integração.



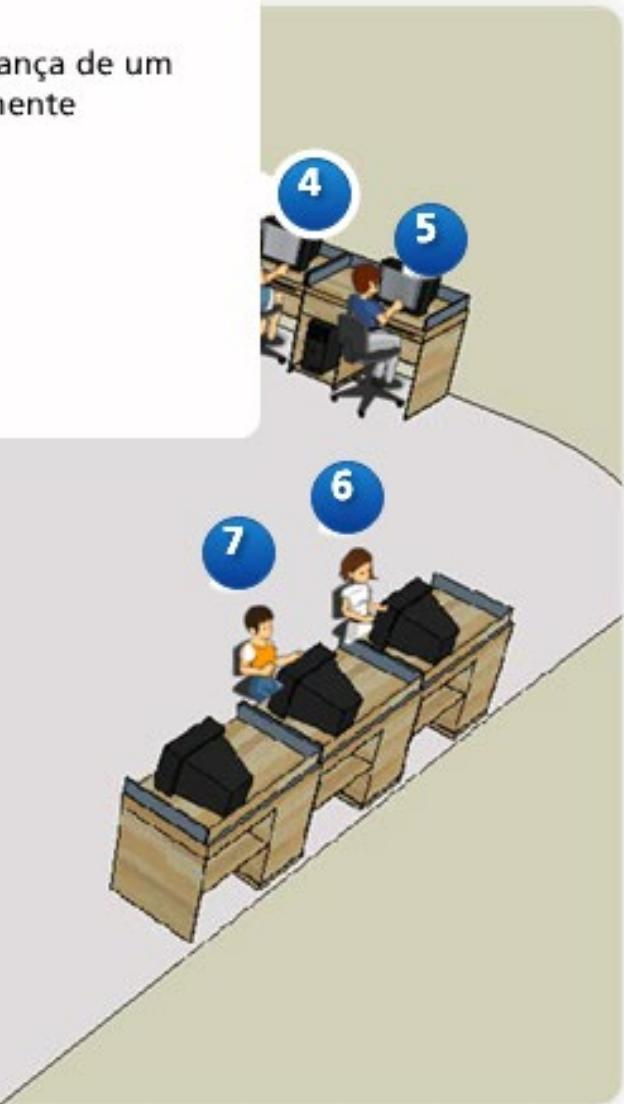
## Quem participa do processo de teste?

### Líder do projeto de teste



**i**

Sou o técnico responsável pela liderança de um projeto de teste específico, normalmente relacionado a um sistema.



**4**

**5**

**6**

**7**

## Quem participa do processo de teste?

### Arquiteto de teste

Meu trabalho como arquiteto diz respeito à organização da infra-estrutura de teste:

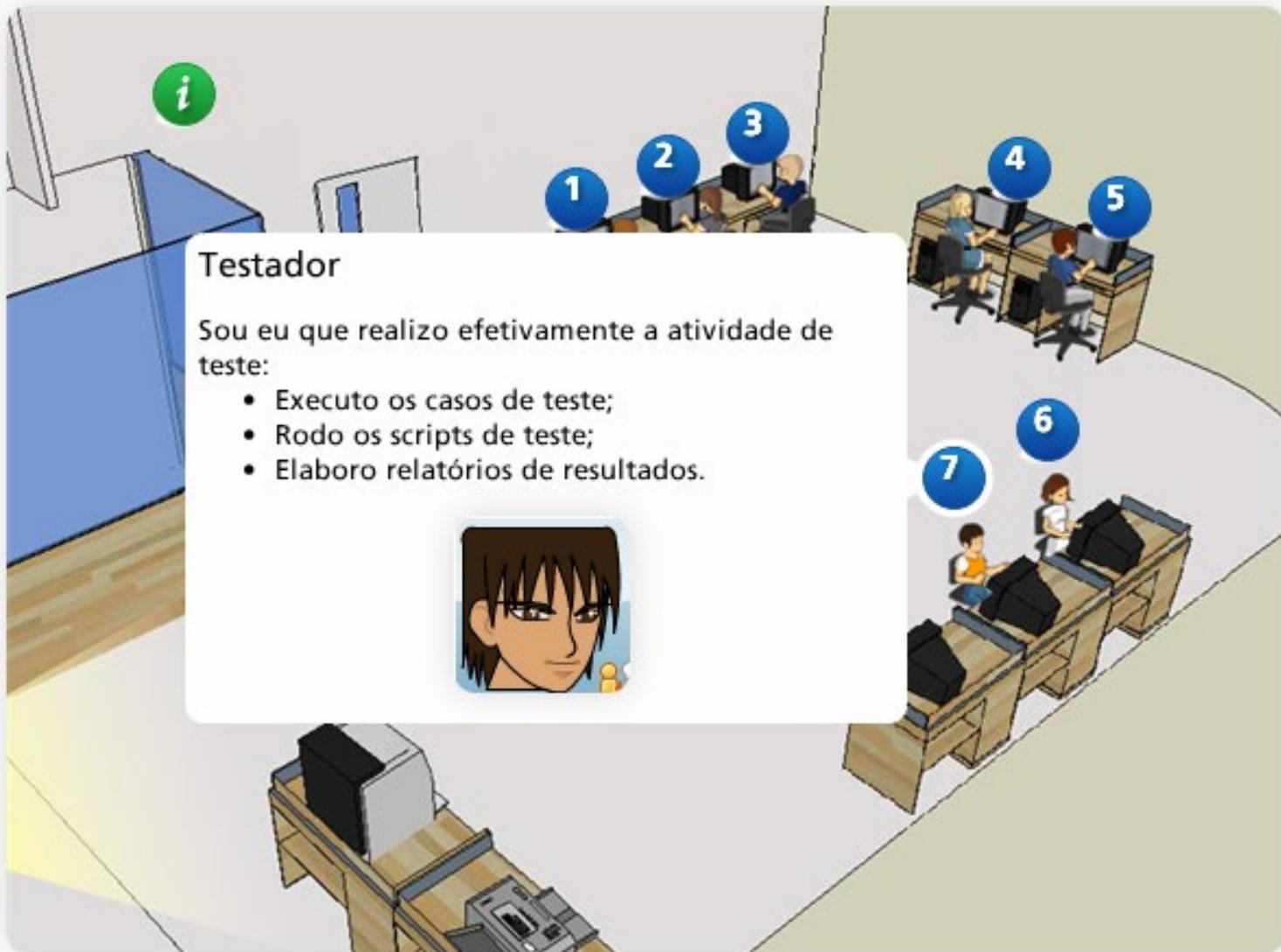
- Ambiente de teste;
- Ferramentas e Capacitação da equipe.



## Quem participa do processo de teste?



## Quem participa do processo de teste?



## Quem participa do processo de teste?





Vamos exercitar?



Qual definição para teste de software está correta conforme o conceito definido por Myers?

- O objetivo do teste é mostrar que um programa executa corretamente as funções para as quais foi destinado.
- Testar é o processo de criar a confiança de que um programa faz aquilo que se supõe que ele faça.
- Testar é o processo de execução de um programa com a intenção de encontrar erros.
- O teste é o processo de demonstrar que os erros não estão presentes

**Quais afirmações abaixo não são definidas pela regra 10 de Myers:**

- O custo para correção de um defeito aumenta de forma diretamente proporcional a complexidade do software.
- O custo para correção de um defeito aumenta de forma diretamente proporcional ao tempo de demora em sua localização.
- O custo para correção de um defeito aumenta de forma diretamente proporcional ao tamanho do software.
- O custo para correção de um defeito aumenta de forma inversamente proporcional ao tamanho da equipe.

No processo de teste os casos de teste são:

- Elaborados pelo líder do projeto de teste e executados pelo testador
- Elaborados e executados pelo testador
- Elaborados pelo arquiteto de testes e executados pelo testador
- Elaborados pelo analista de testes e executados pelo testador

Entre os papéis relacionados no processo de desenvolvimento, clique sob quem se preocupa em assegurar que custos e prazos estipulados serão cumpridos.



**Analista de  
Sistemas**



**Gestor de  
Qualidade**



**Gerente de  
Projetos**



**Arquiteto  
de Testes**

O processo de desenvolvimento e de testes devem coexistir. Fato pelo qual, alguns dos profissionais participam das duas atividades, como por exemplo: "Além de elaborar requisitos e modelar o sistema, também participa da construção de testes unitários e execução de testes de integração". De quem estamos falando?

- Analista de Sistemas
- Analista de Testes
- Gerente de Projetos
- Programador

Relacione as colunas segundo o conceito de Engano, Defeito, Erro e Falha.

Engano

Um comportamento inesperado do programa percebido pelo usuário

Defeito

Algo que pode acarretar um comportamento inesperado.

Erro

A diferença entre o resultado que se esperava de um processo e o resultado que ele apresenta

Falha

Um equívoco ou um descuido no momento da codificação do software.

Segundo a terminologia apresentada pela IEEE 610.12-1990, podemos concluir que:

- Um engano é ocasionado por uma ou mais falhas
- Um defeito, nesse contexto, é sinônimo de falha.
- Uma falha é ocasionada por um ou mais defeitos.
- Uma falha, nesse contexto, é sinônimo de defeito.

Uma vez que defeitos podem ser inseridos no software durante todo seu processo de desenvolvimento, é importante que as atividades de teste ocorram desde o início do projeto. Mas, é possível fazer algum tipo de teste antes de termos, ao menos, uma versão estável para executar?

- Não, os testes dependem de uma versão estável do programa, ou ao menos parte dele, para execução das atividades de teste.
  - Sim, durante a definição de requisitos e modelagem do futuro sistema, já é
- possível realizar atividades inspeção e análise da qualidade dos artefatos. Esta atividade recebe o nome de "Teste Estático"
- Os testes de unidade, são testes estáticos, que não precisam uma versão estável do sistema em execução.
- Os testes servem para conferir se o sistema atende às necessidades do cliente. Portanto, não existem testes sem o programa implementado para testar.

Sobre as atividades de Verificação e Validação (V&V), podemos afirmar que:

- Verificação constitui um teste estático (exige a execução do produto em teste).
- Verificação constitui um teste dinâmico (exige a execução do produto em teste)
- Validação constitui um teste dinâmico (exige a execução do produto em teste).
- Validação constitui um teste estático (não exige a execução do produto em teste).

# Praticando

Em grupo de 4 alunos (2 desenvolvedores e 2 usuários), simular uma reunião para especificação de requisitos de um Sistema de Controle de Biblioteca

Ao término da “reunião”, crie um documento com:

- Nome do sistema
- Áreas envolvidas
- Objetivos do sistema
- Restrições
- Descrição funcional

# Praticando

Em grupo de 4 alunos (2 desenvolvedores e 2 usuários), simular uma reunião para especificação de requisitos de um Sistema de Controle de Biblioteca

Ao término da “reunião”, crie um plano de testes com:

- Nome do sistema
- Áreas envolvidas
- Objetivos do sistema
- Restrições
- Descrição funcional

Entre outros...

# Praticando

Em grupo de 4 alunos (2 desenvolvedores e 2 usuários), desenvolver um plano de testes completo para :

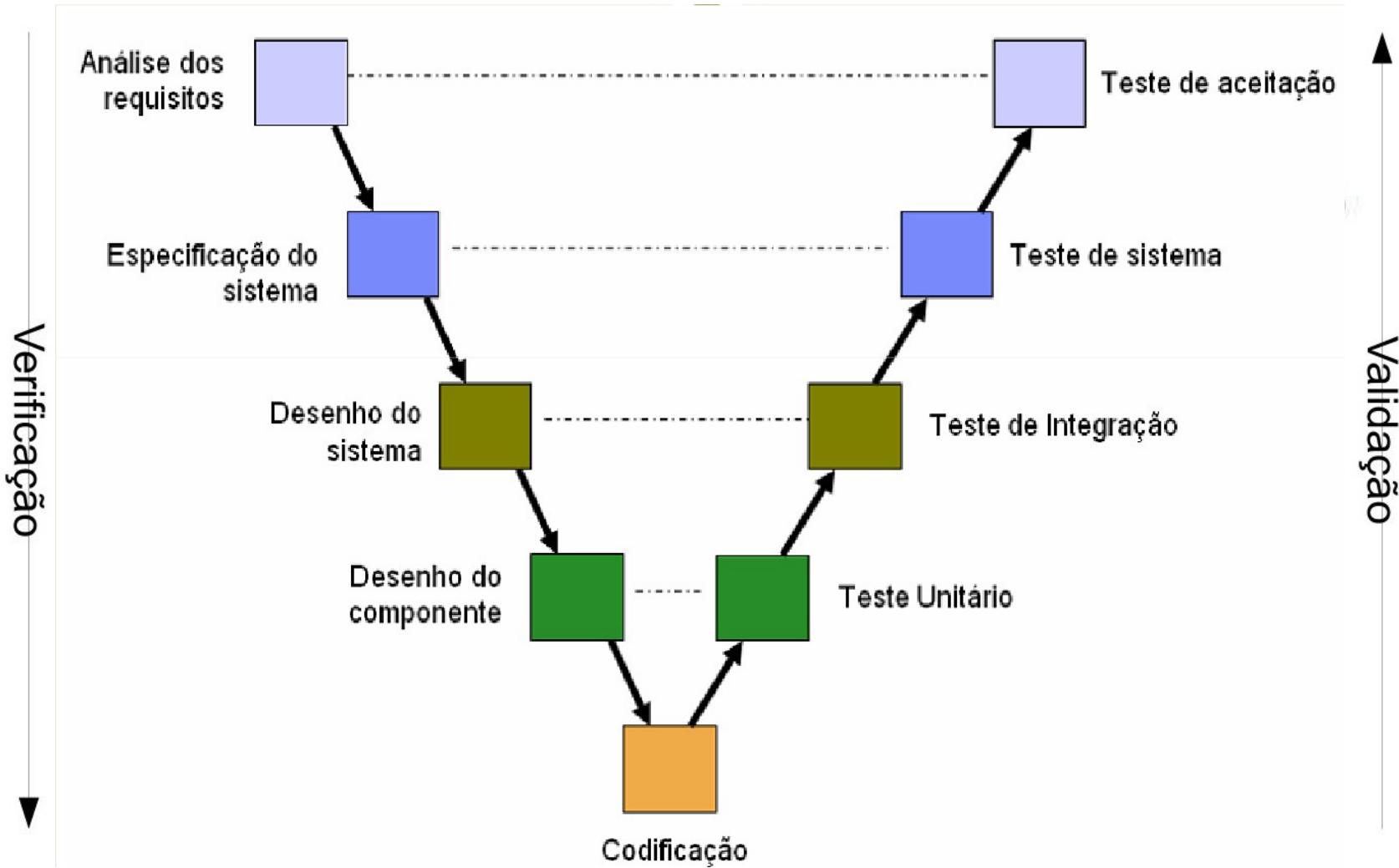
A empresa JAVAS MODAS vende um software de gerenciamento de E-commerce Varejistas.

O cliente necessita de um sistema web/mobile que proporcione maior lucratividade e gestão em suas vendas de forma online, onde seja possível efetuar compras, gestão comercial e estoque.

O Sistema permitirá os gerenciamentos dos Usuário e Produtos, como cadastrar, alterar, pesquisar e excluir. Possibilitará uma listagem dos produtos disponíveis no estoque, realizar vendas e gerar seus relatórios.

# Verificação e Validação

## Processo em “V”

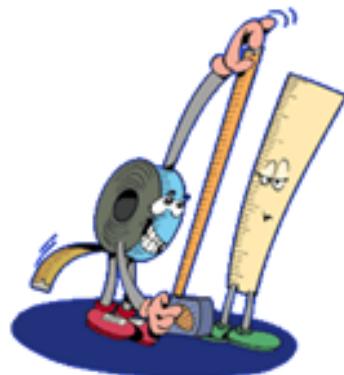


# *Níveis de Teste de Software*

Atributos	Nível dos Testes			
	Testes Unitários	Testes de Integração	Testes de Sistema	Testes de Aceitação
Escopo	<i>Unidades</i>	<i>Conjunto de unidades agrupadas</i>	<i>Sistema todo</i>	<i>Sistema todo</i>
Equipe	<i>Desenvolvedores</i>	<i>Desenvolvedores e Analistas de Sistema</i>	<i>Analista de Testes e Testadores</i>	<i>Analista de Testes, Testadores e Usuários</i>
Origem dos dados	<i>Criação manual</i>	<i>Criação manual</i>	<i>Criação automática / dados reais</i>	<i>Dados reais</i>
Volume dos dados	<i>Pequeno</i>	<i>Pequeno</i>	<i>Grande</i>	<i>Grande</i>
Interfaces	<i>Não existem</i>	<i>Não existem</i>	<i>Simuladas / Reais</i>	<i>Reais</i>
Ambientes	<i>Desenvolvimento</i>	<i>Desenvolvimento</i>	<i>Testes</i>	<i>Testes / Produção</i>



Verificação e Validação - V&V  
são termos comumente usados  
ao estudarmos teste e  
qualidade de software. Mas o  
que são exatamente?



As atividades de V&V tem por  
objetivo assegurar que o  
software seja adequado e  
atenda às necessidades, ou  
seja, a confirmação de que  
cumpra suas especificações.





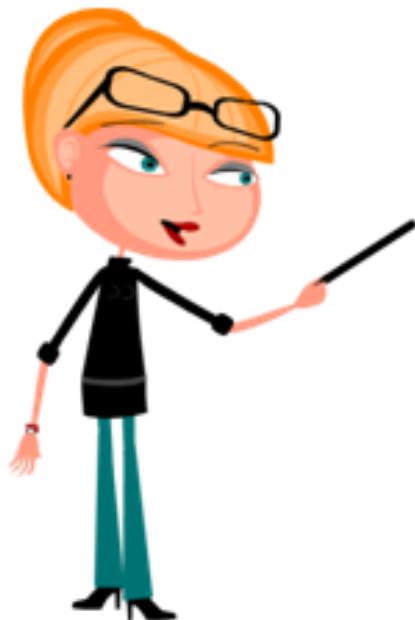
Verificação é a confirmação de que o software cumpre com suas especificações.  
Busca responder a pergunta:  
Estamos construindo corretamente o produto?



Enquanto validação é a confirmação de que o software está de acordo com o que o usuário deseja.  
Estamos construindo o produto certo?



As atividades de V&V podem ser **estáticas** (quando não exigem execução do software testado) e **dinâmicas** (obrigam a existência e execução do software testado)



#### Exemplos de atividades estáticas:

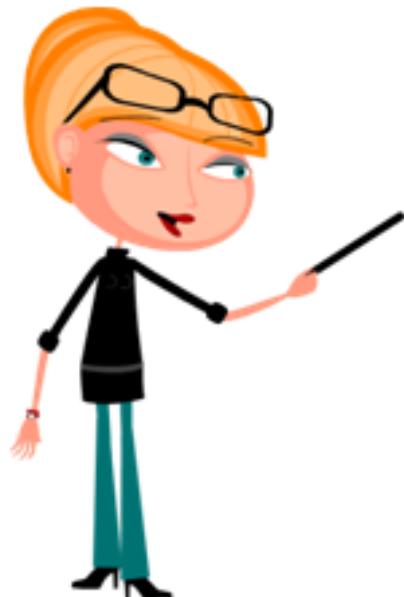
- Revisões de requisitos e modelos
- Inspeção de código
- walkthroughs

#### Exemplos de atividades dinâmicas:

- Teste unitário
- Teste de integração
- Teste de sistemas
- Teste de aceitação



As atividades de V&V podem ser **estáticas** (quando não exigem execução do software testado) e **dinâmicas** (obrigam a existência e execução do software testado)



#### Exemplos de atividades estáticas:

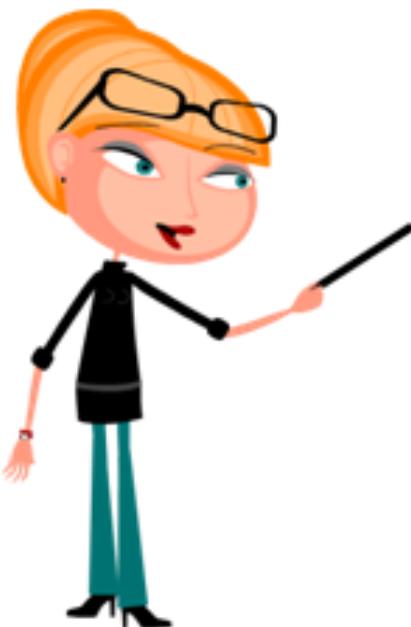
- Revisões de requisitos e modelos
- Inspeção de código
- walkthroughs

#### Exemplos de atividades dinâmicas:

- Teste unitário
- Teste de integração
- Teste de sistemas
- Teste de aceitação



As atividades de V&V podem ser **estáticas** (quando não exigem execução do software testado) e **dinâmicas** (obrigam a existência e execução do software testado)



#### Exemplos de atividades estáticas:

- Revisões de requisitos e modelos
- Inspeção de código
- walkthroughs

#### Exemplos de atividades dinâmicas:

- Teste unitário
- Teste de integração
- Teste de sistemas
- Teste de aceitação

# Requisitos

A primeira etapa do desenvolvimento de software envolve o levantamento, análise e classificação de requisitos.

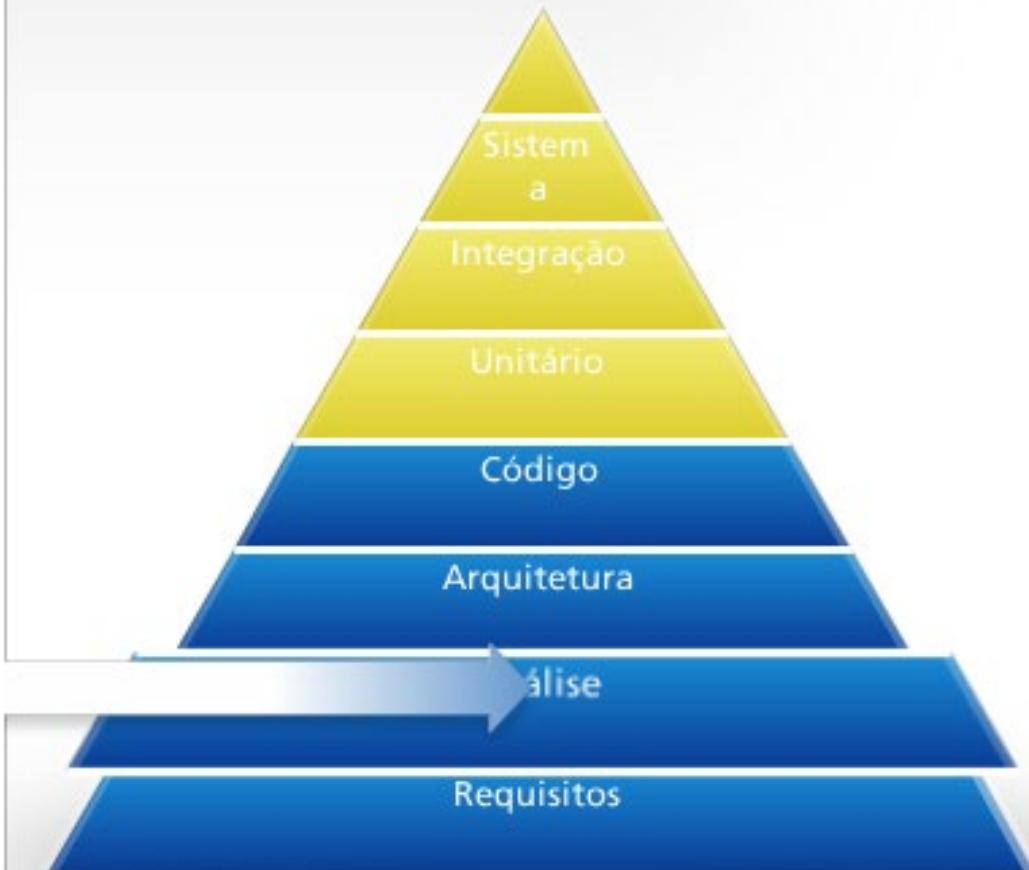
São os requisitos que irão determinar as especificações do sistema a ser desenvolvido. Desta forma, são também importantes para o processo de teste.

Atividades de revisão podem melhorar a qualidade dos requisitos, que serão base



# Análise

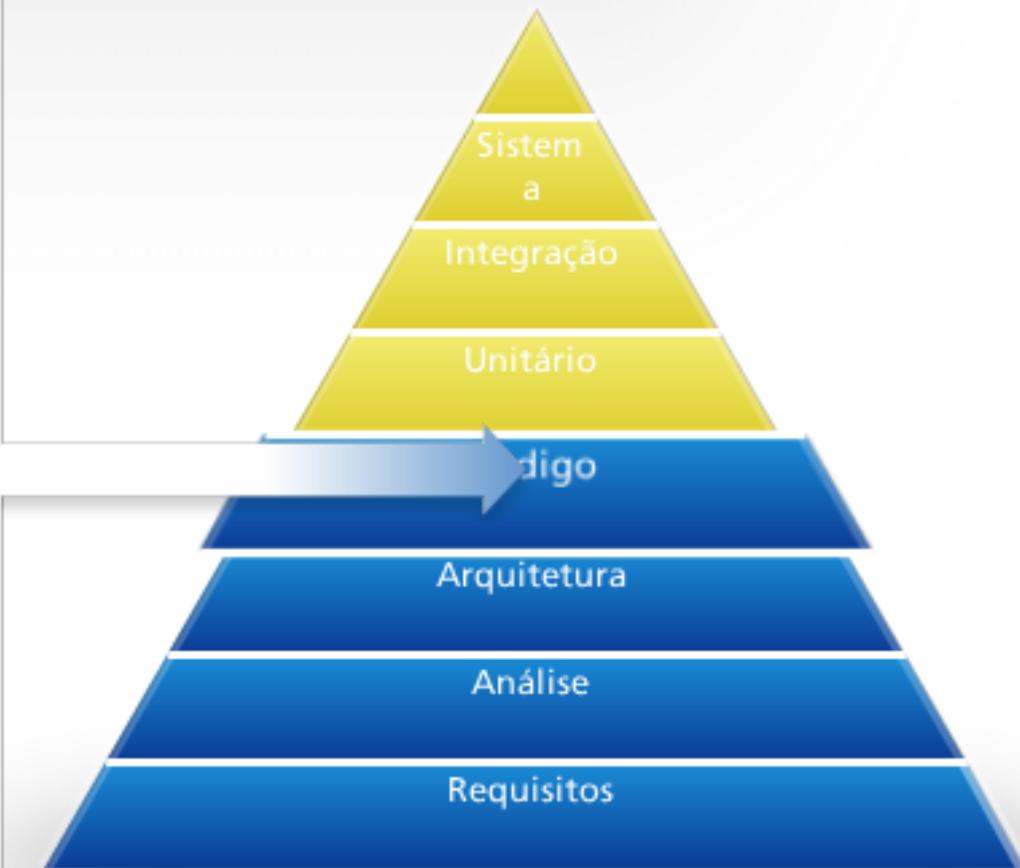
A análise dos artefatos gerados no processo de desenvolvimento, durante a etapa modelagem, como diagramas UML, documentos com detalhamento e classificação de requisitos, etc... pode apontar inconsistências ou erros de interpretação.



# Código

A técnica mais comum é a revisão de código, que pode ser feita de duas formas:

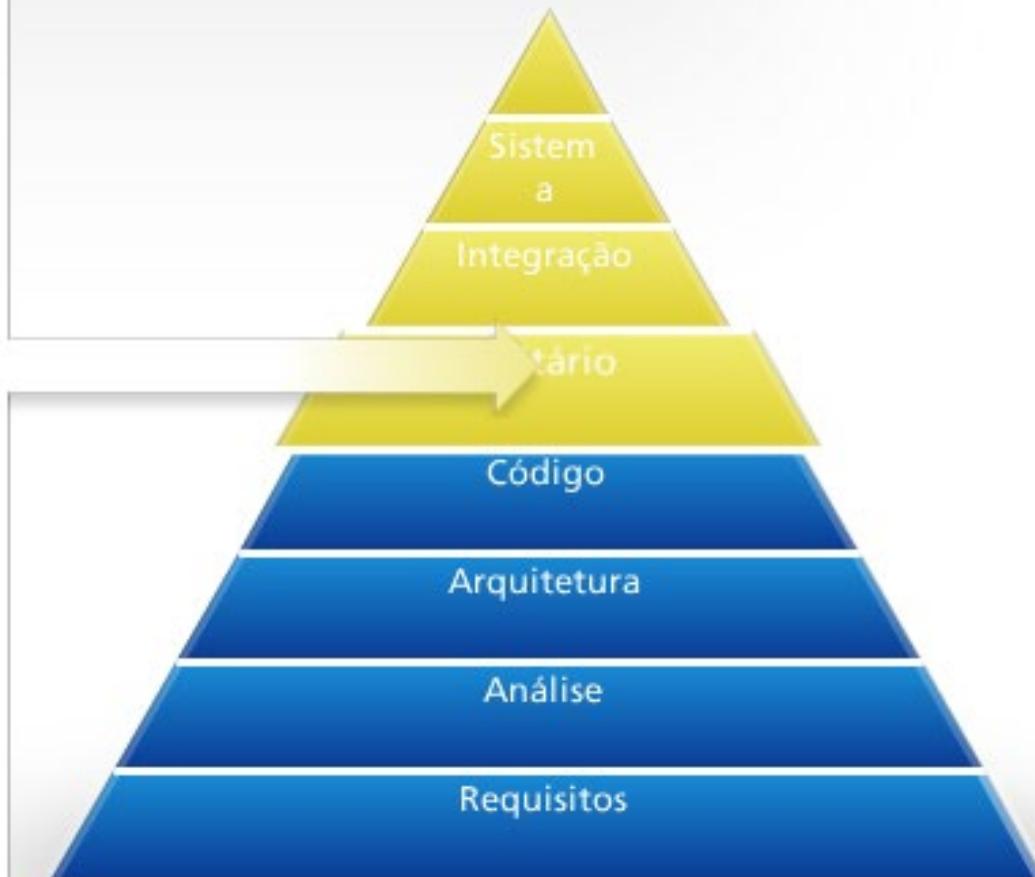
- 1. Walkthroughs:** o código fonte e a documentação correspondente são comparados pela equipe de revisão na busca por divergências.
- 2. Inspeção de código:** É mais formal que o walkthroughs. Agora a comparação entre o código fonte e a documentação é norteada por uma lista de aspectos predeterminados.



# Unitário

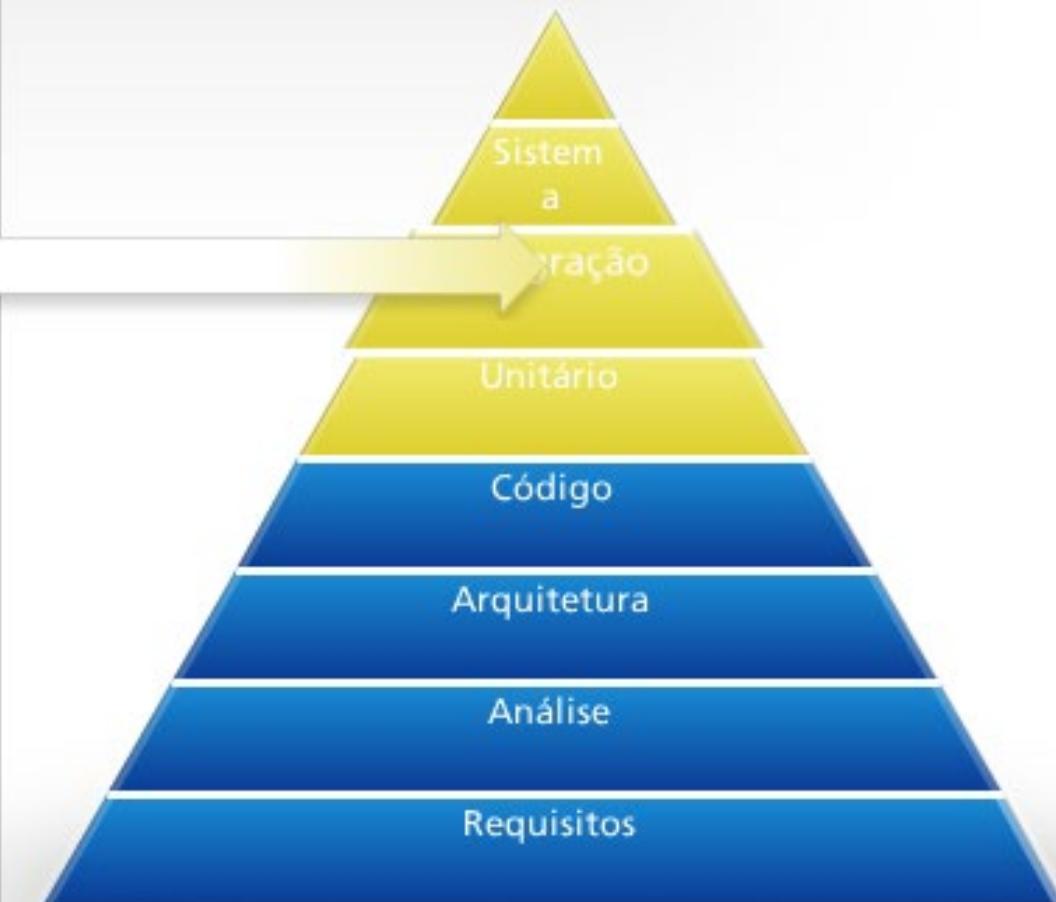
Testes unitários são implementações de teste feitas por desenvolvedores de software com intuito de testar o menor elemento testável, que geralmente em sistemas orientado a objetos são métodos.

Uma ferramenta conhecida entre os desenvolvedores Java, por exemplo, é o JUnit



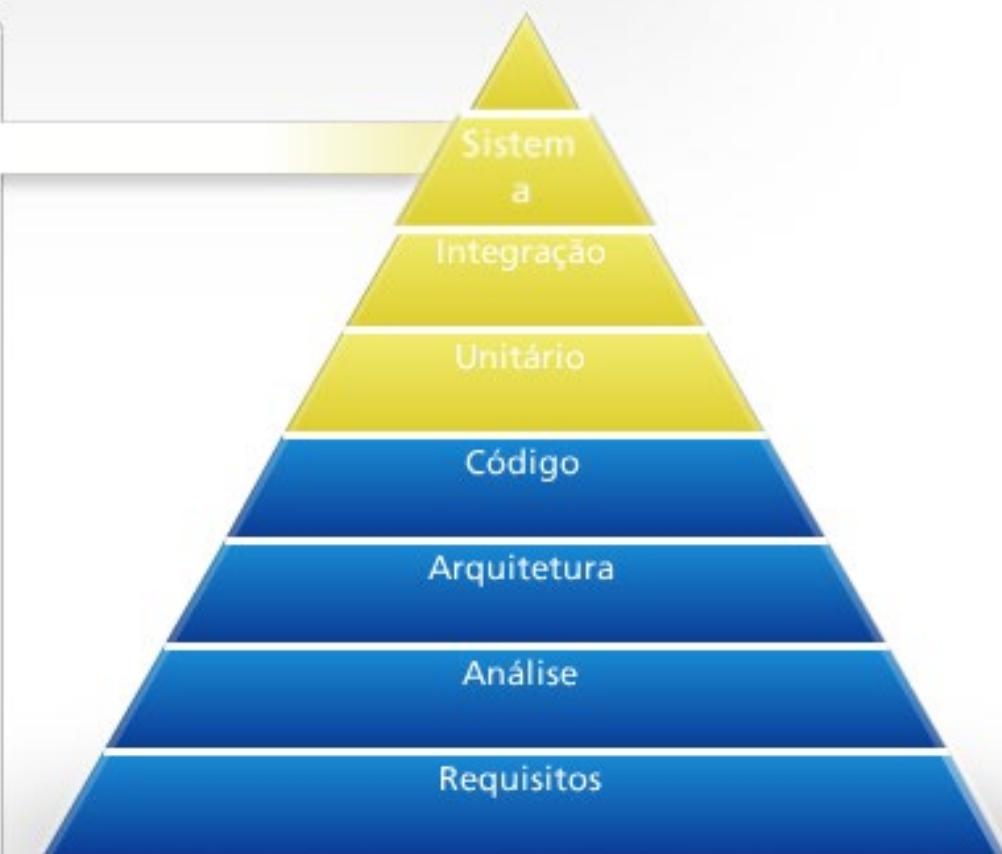
# Integração

Quando todos os componentes de um software foram testados, surge uma pergunta: quando estiverem integrados, continuarão funcionando? O teste de componentes individuais antecede o teste de integração. Contribui para assegurar a correção de componentes individuais, mas não é capaz de garantir que as dependências funcionais entre componentes estejam perfeitamente implementadas.



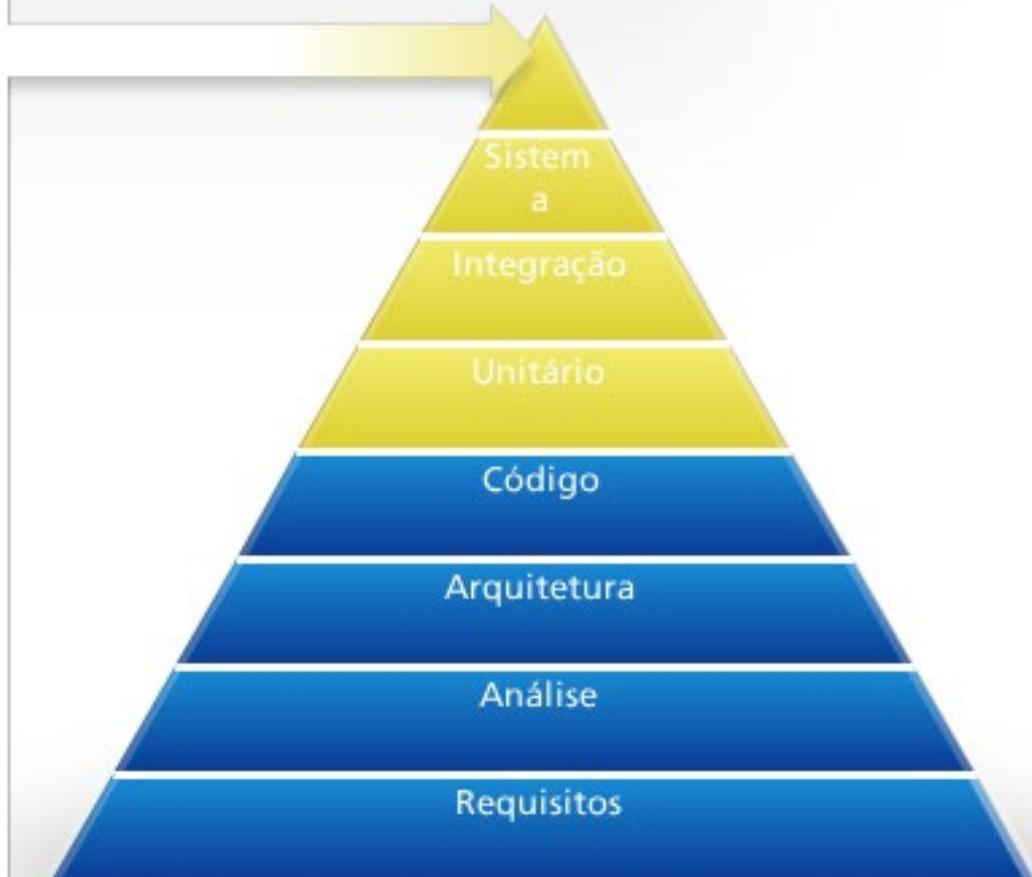
## Sistema

O teste de sistema é uma fase em que o sistema já está completamente integrado e é testado como um todo. Suas especificações (requisitos funcionais, não funcionais, regras de negócio e casos de uso) são exercitadas em um ambiente controlado.



# Aceitação

O teste de aceitação é a última ação de teste antes da implantação do software. A meta do teste de aceitação é verificar se o software está pronto e pode ser usado pelos usuários finais para executar as funções e as tarefas para as quais foi criado. A principal característica que distingue o teste de aceitação do teste de sistema é a participação (envolvimento) do usuário no teste.



# Técnicas de Teste de Software

## *Tipos de Teste de Software*

### ***Testes de Caixa-Branca (Estrutural)***

*Testes de Unidade*

*Teste de Integração*

### ***Testes de Caixa-Preta (Funcional)***

*Testes Funcionais*

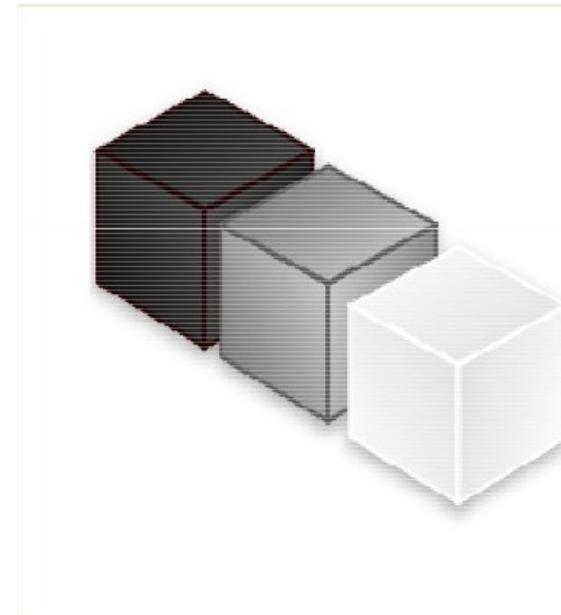
*Testes de Aceitação*

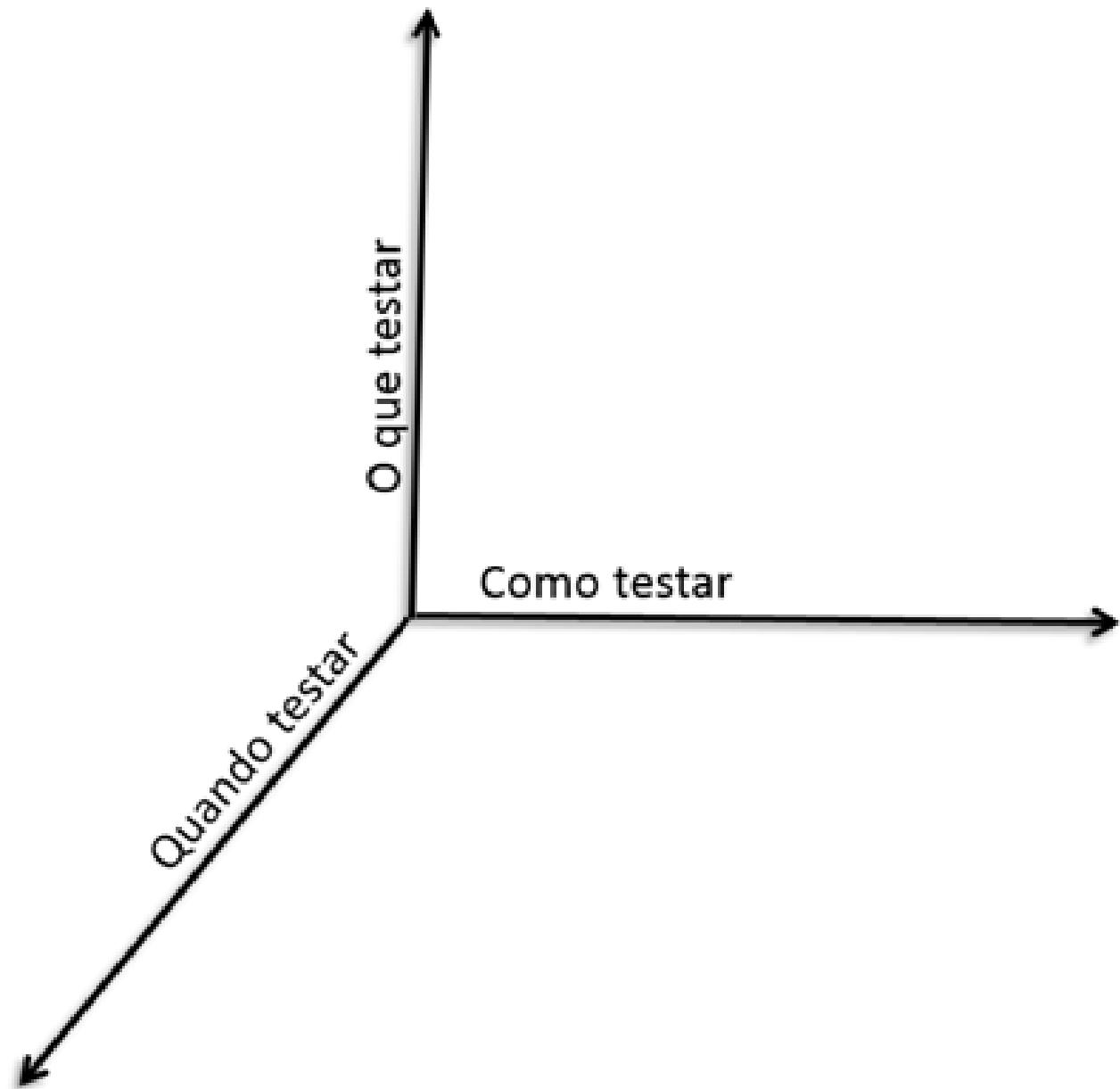
*Testes Exploratórios*

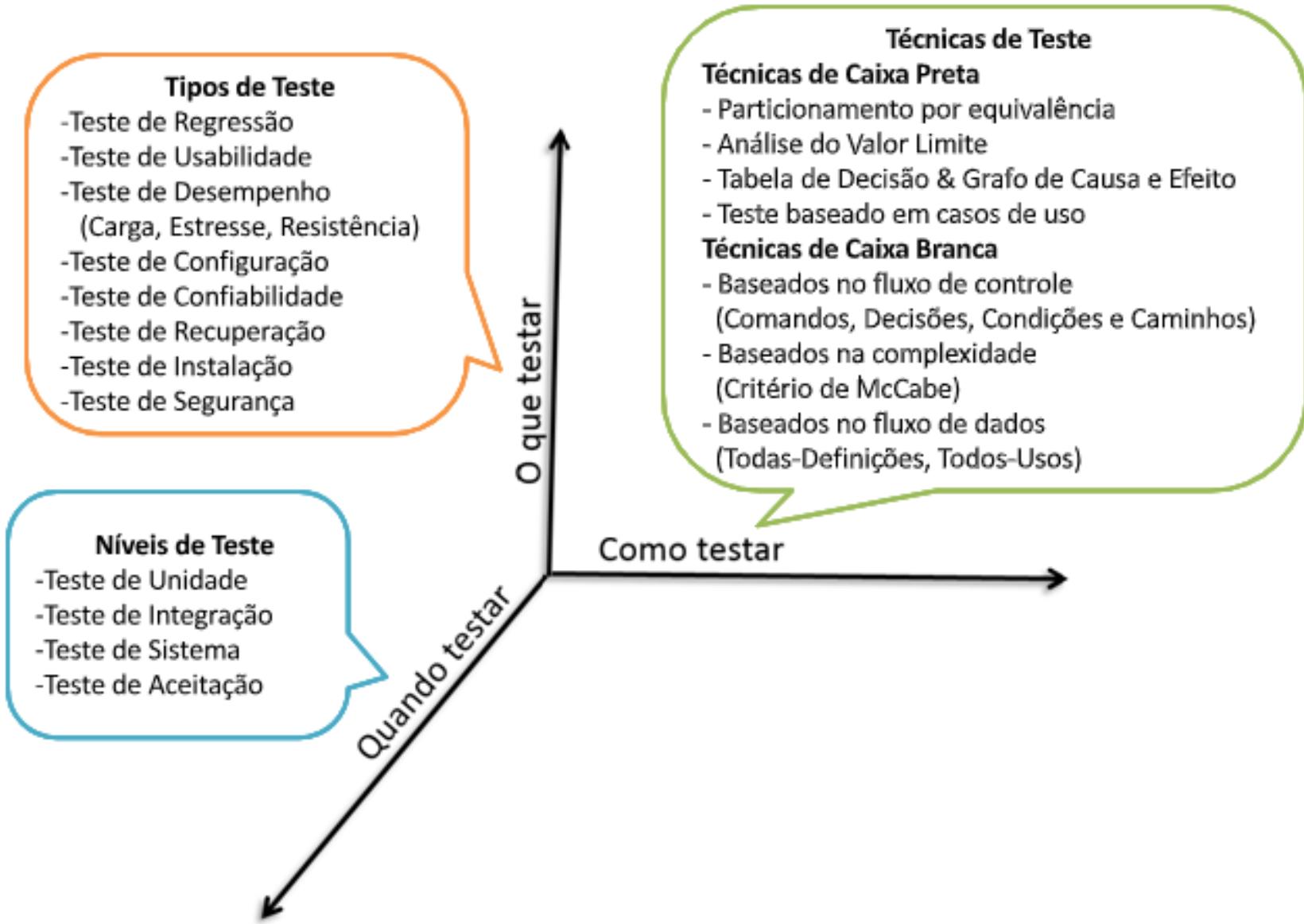
### ***Testes de Caixa-Cinza***

*Testes de Regressão*

*Testes de Cobertura*







- Tipos de Teste**
- Teste de Regressão
  - Teste de Usabilidade
  - Teste de Desempenho  
(Carga, Estresse, Resistência)
  - Teste de Configuração
  - Teste de Confiabilidade
  - Teste de Recuperação
  - Teste de Instalação
  - Teste de Segurança

- Níveis de Teste**
- Teste de Unidade
  - Teste de Integração
  - Teste de Sistema
  - Teste de Aceitação

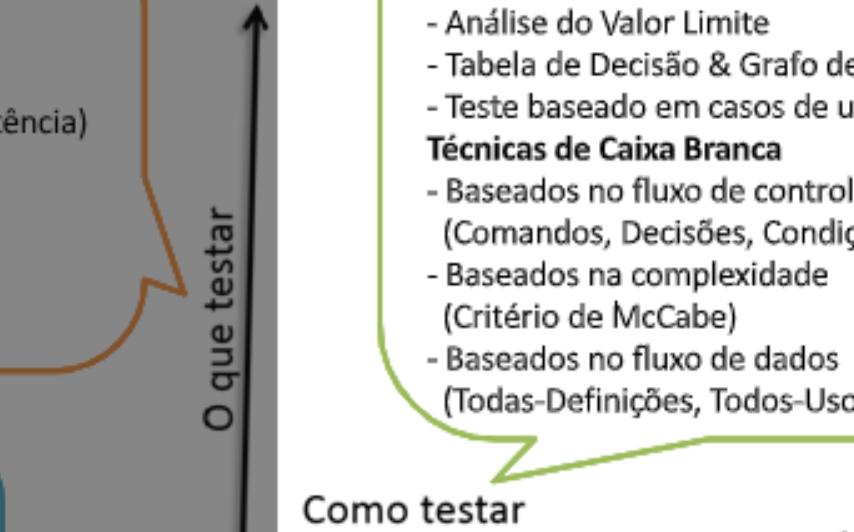
### Técnicas de Teste

#### Técnicas de Caixa Preta

- Particionamento por equivalência
- Análise do Valor Limite
- Tabela de Decisão & Grafo de Causa e Efeito
- Teste baseado em casos de uso

#### Técnicas de Caixa Branca

- Baseados no fluxo de controle  
(Comandos, Decisões, Condições e Caminhos)
- Baseados na complexidade  
(Critério de McCabe)
- Baseados no fluxo de dados  
(Todas-Definições, Todos-Usos)



É por aqui que  
iremos começar!



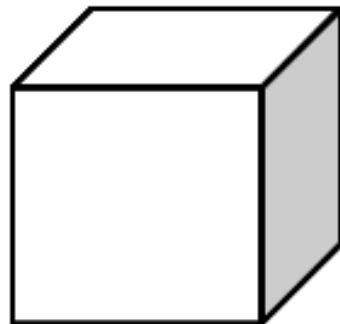


As técnicas de teste irão apresentar formas diferentes de elaboração de casos de teste explorando as diferentes características de um sistema!

Podemos dizer que as técnicas de teste definem a maneira como os testes são planejados e executados, ajudando na escolha do conjunto ideal de casos de teste!

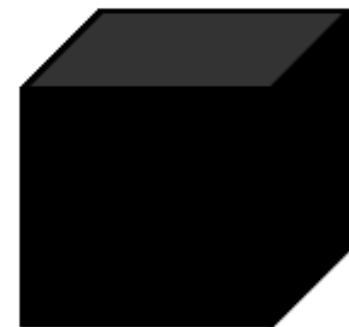


# Quais as técnicas de Teste Existentes?



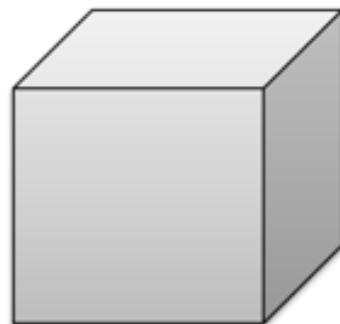
Caixa Branca

Também chamada caixa de vidro **ou teste estrutural**



Caixa Preta

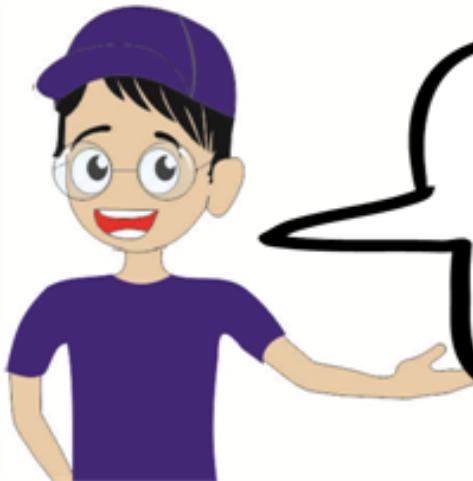
Também chamada teste **funcional**



Caixa Cinza

Termo usado por alguns autores quando o  
projetista utiliza critérios das duas técnicas anteriores

# Critérios de Teste

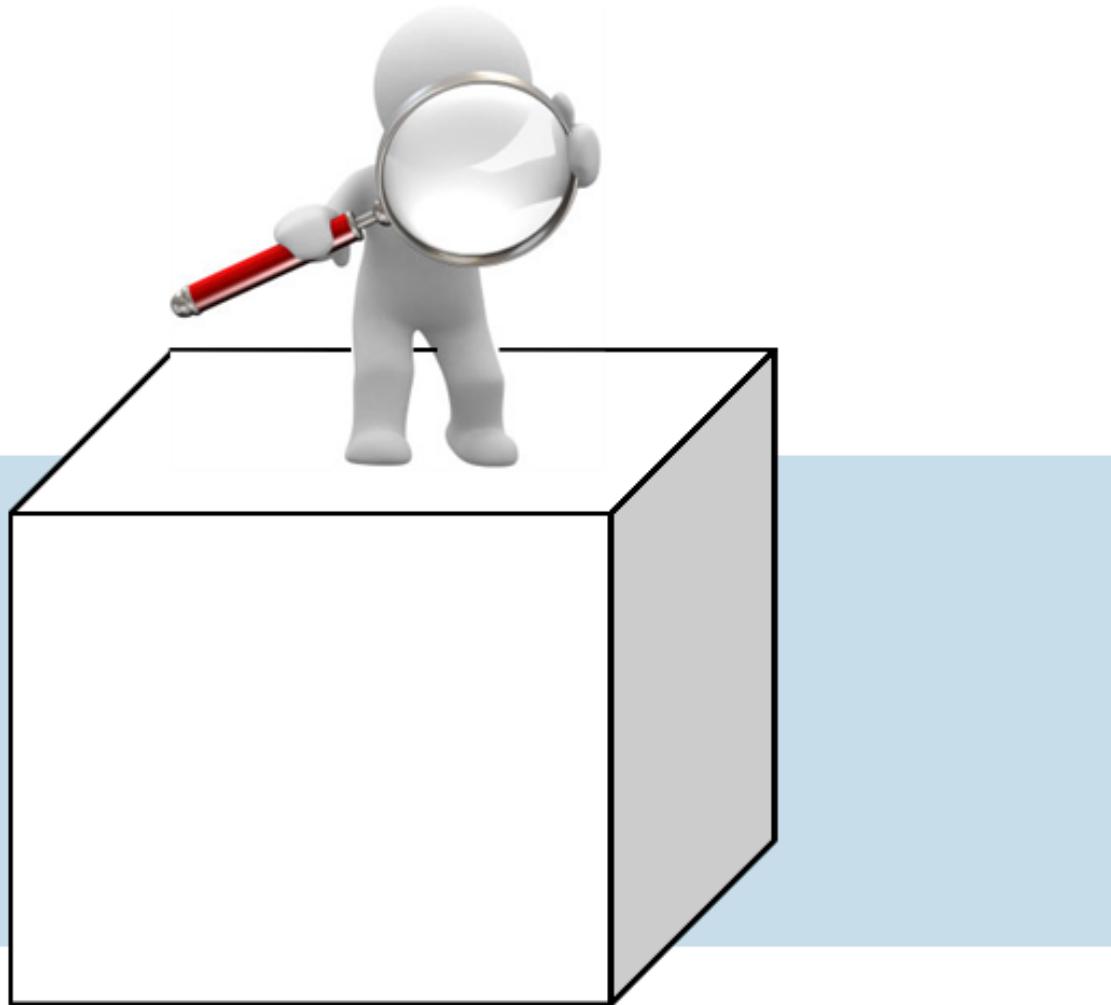


Cada técnica apresenta um conjunto de **critérios** que determinam uma forma de extração de casos de teste.

Ou seja, cada critério define uma metodologia para derivar casos de teste.



# Técnica Caixa Branca





A técnica CAIXA BRANCA ou teste estrutural é responsável em avaliar o comportamento interno do componente de software.  
A função faz o que se espera dela?

Para essa técnica é necessário conhecer o código-fonte do programa.

```
public Usuario getUsuario(Integer codUsuario){  
    return getUsuarioPorCodigo(codUsuario);  
}
```

# Técnica Caixa Branca: Critérios Baseados no Fluxo de Controle



Utilizando o conhecimento detalhado do código-fonte é possível criar uma série de testes de modo a exercitar todos os seus elementos internos.

- Está retornando o usuário certo?

```
Public Usuario getUsuario(Integer codUsuario){  
    return getUsuarioPorCodigo(codUsuario);  
}
```

# Técnica Caixa Branca: Critérios Baseados no Fluxo de Dados



Dentre os critérios mais conhecidos da técnica caixa branca podemos citar:

Teste de Caminhos

Teste de Comandos

Teste de Decisões

# Teste de caminhos



# Exemplo

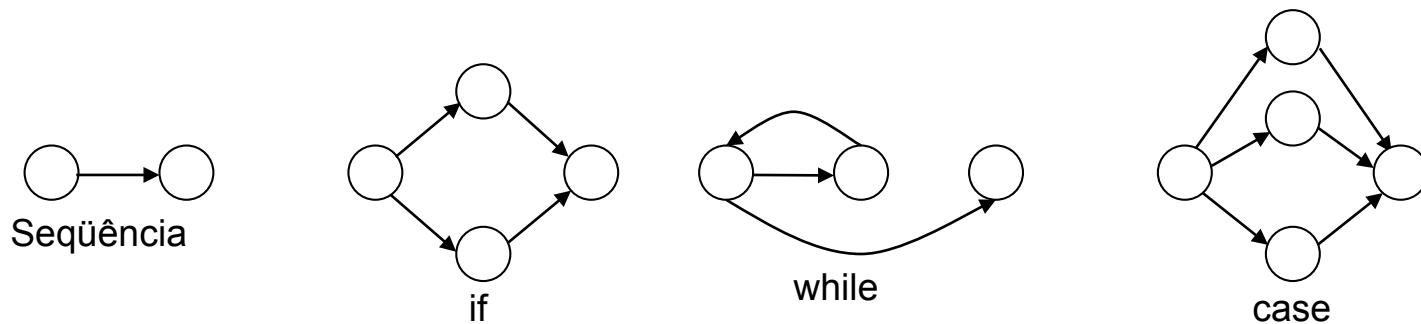


# Teste de caminho básico- exercício

- É uma **técnica de teste de caixa branca** que possibilita que o projetista do caso de teste derive uma medida de complexidade lógica de um projeto procedural e use essa medida como guia para definir **um conjunto básico de caminhos de execução**.

Notação de grafo de fluxo:

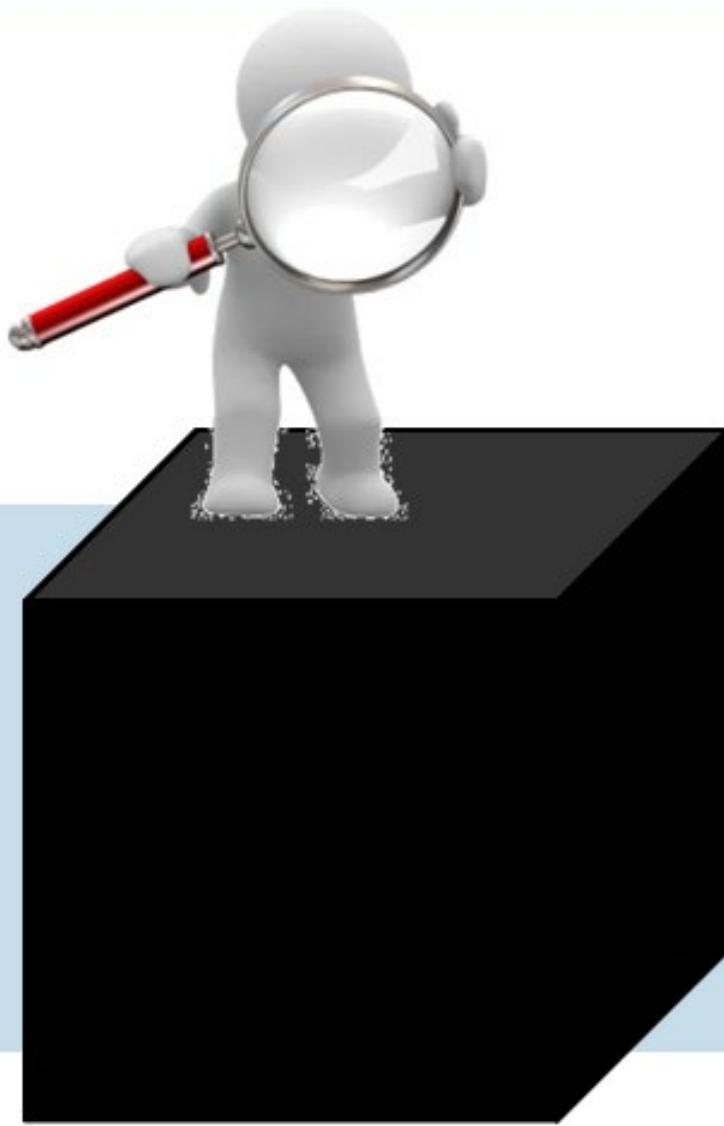
- *notação simples para representação do fluxo de controle, que descreve o fluxo lógico:*



# Complexidade Ciclomática

- É uma métrica de SW que proporciona uma medida quantitativa da complexidade lógica de um programa
- O valor computado da complexidade ciclomática **define o número de caminhos independentes do conjunto básico de um programa** e oferece-nos um limite máximo para o **número de testes que deve ser realizado** para garantir que todas as instruções sejam executadas pelo menos uma vez.

# Técnica Caixa Preta



<https://www.youtube.com/embed/FeKCRxhLopg?autohide=0&showinfo=0&theme=light&rel=0>

# Teste de caixa preta

O analista não tem acesso ao código fonte e desconhece a estrutura interna do sistema. É também conhecido como **teste funcional**, pois é baseado nos **requisitos funcionais** do software. O foco, nesse caso, é nos requisitos da aplicação, ou seja, nas ações que ela deve desempenhar.

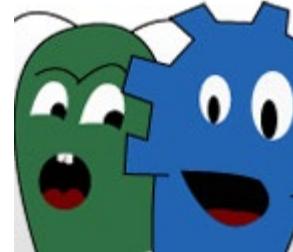
Para mostrar quais problemas que esse tipo de teste rastreia, podemos citar alguns exemplos:

- Data de nascimento preenchida com data futura;
- Campos de preenchimento obrigatório que não são validados;
- Utilizar números negativos em campos tipo valor a pagar;
- Botões que não executam as ações devidas;

Enfim, todo tipo de falha funcional, ou seja, falhas que contrariam os requisitos da aplicação.

Há que se destacar, contudo, que existe um elemento comum aos dois tipos de teste. Tanto no teste de caixa branca quanto no teste de caixa preta, o analista não sabe qual será o comportamento da aplicação ou do **alvo de teste** em uma determinada situação. A imprevisibilidade de resultados é comum aos dois casos.

# *Os tipos de bugs de softwares*



Podemos separar os **bugs de software** em algumas categorias para classificar e conseguir dar prioridade na resolução dos mais “graves”.

A classificação leva em consideração a natureza do bug e o que ele provoca no sistema.

Temos 5 classificações que são amplamente aceitas e difundidas no meio da comunidade de testadores como um todo. São elas:

- **Impeditiva** – Ocorrência que impede o uso do sistema. Não existe forma de contorná-la.

*Exemplo:* Ao clicar em um botão, o sistema trava e para seu funcionamento.

- **Funcional** – Ocorrência não impeditiva, ou seja, pode ser contornada pelo usuário e é relativa ao funcionamento do sistema.

- É o não cumprimento de algum requisito.

*Exemplo:* Ao clicar no botão “Limpar”, o campo não é limpo.

- **Interface** – Ocorrências relativas à interface, como problemas de renderização, cores, etc.

*Exemplo:* O *label* de um botão aparece deslocado, fora da área do botão.

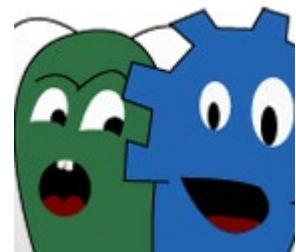
**Texto** – Qualquer ocorrência relacionada ao uso incorreto de algum idioma.

**Melhoria** – Sugestões de melhorias para o sistema.

Muitos testadores ficam em dúvida em relação ao **bug de segurança**. Pela definição citada, o bug de segurança é funcional, pois a falha na segurança é apenas o não cumprimento de um requisito. Porém, ele é comparado ao **bug**

**impeditivo** em termos de impacto causado no sistema. Esse tipo de bug costuma ser tão crítico quanto um impeditivo. Por isso, há uma certa confusão na hora de classificar esse tipo de ocorrência.

<https://app.crowdtest.me/entenda-teste-caixa-preta-garante-qualidade-software/>



## Ferramenta Selenium

Quando utilizamos o Selenium para fazer testes automatizados, basicamente temos duas ferramentas para utilizar:

- **Selenium IDE:** é uma ferramenta que permite a rápida prototipagem de scripts de testes. É um *plugin* do Firefox que possibilita gravar o comportamento do usuário: página que ele acessou, textos escritos em formulários, cliques em links e botões etc. Depois de gravadas, estas ações podem ser exportadas para um script de teste em diversas linguagens de programação: Java, Python, Perl, JavaScript etc.
- **Selenium WebDriver:** é uma ferramenta que oferece uma API que permite a escrita de forma mais produtiva e organizada de scripts de testes. Essa é a escolha natural quando desejamos escrever testes automatizados para aplicações *web* utilizando o Selenium.

## Tutorial Selenium

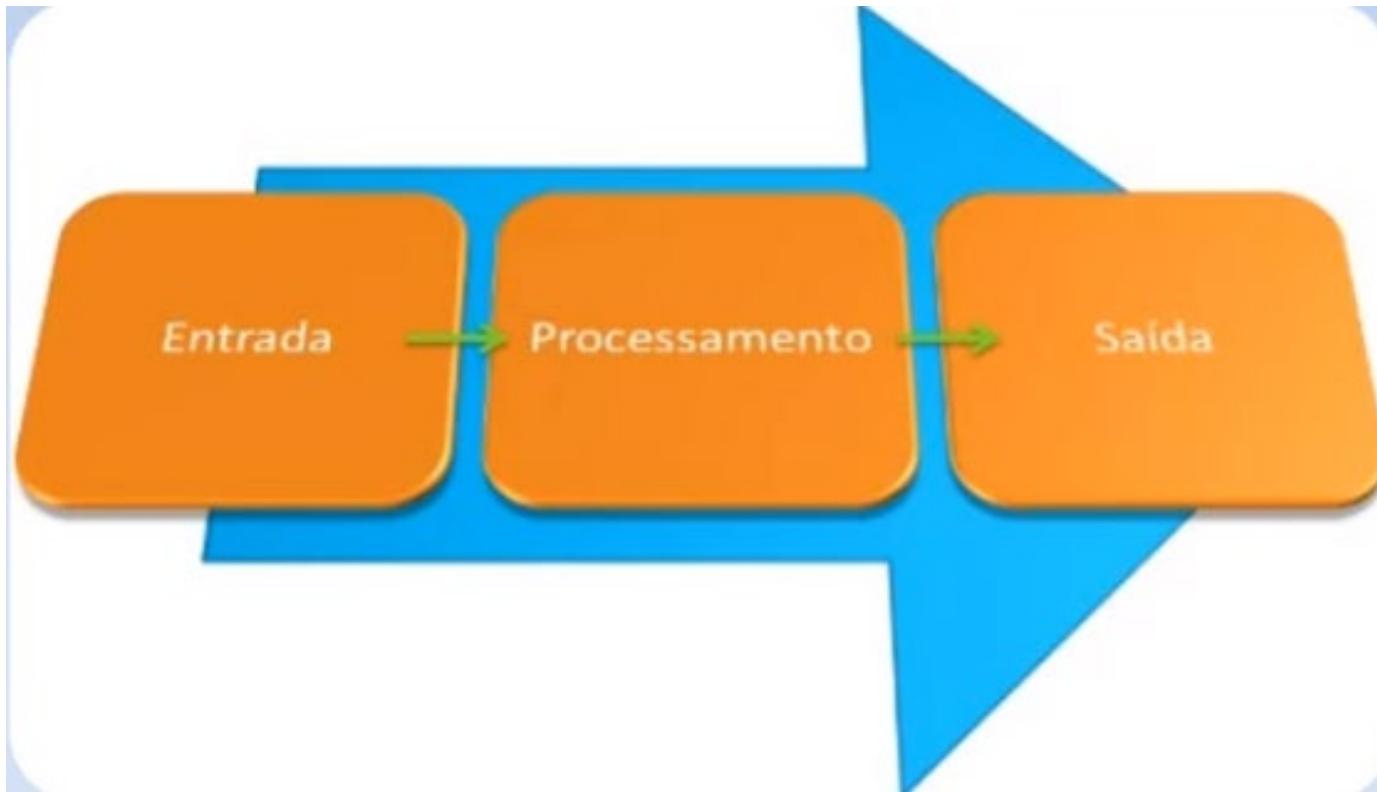
***<https://www.nuget.org/packages/Selenium.WebDriver>***

***<https://www.tiposdetestesdesoftware.com/videos>***

# Conceitos – Técnica Caixa Preta



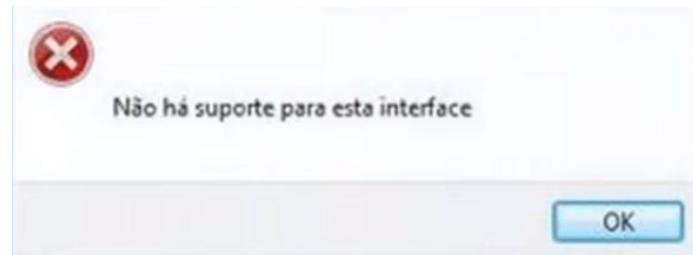
- Define se os requisitos desejados foram totalmente ou parcialmente satisfeitos pelo produto.
- Feito com base na funcionalidade do programa e avaliando se ele é capaz de gerar as saídas corretas para entradas de dados pré-estabelecidas.
- A estrutura do programa é ignorada para que o foco seja na funcionalidade. Não é necessário saber quais componentes do sistema são executados, e sim, o que ele deve fazer.



*Não necessita do código do Sistema . O testador irá se concentrar nas funções que o Sistema deve desempenhar nas Saídas de certos conjuntos de dados. Buscando erros que o usuário pode cometer e que fogem da especificação do Sistema.*

*O teste irá revelar :*

## *Funções Incorretas*



*Erros de Interface*



*Erros de desempenho e comportamento*



*Erros de Inicialização e Finalização*

# Critérios – Técnica Caixa Preta

Partições de Equivalência

Análise do Valor Limite

Teste baseado em casos de uso

Grafo Causa-Efeito

Alguns critérios  
da técnica de  
caixa preta



# Exemplo prático – Observe as Funcionalidades

Como vimos, a técnica caixa preta consiste na aplicação de testes sem a necessidade do conhecimento do código fonte. Por exemplo, podemos testar o funcionamento de um aparelho de TV sem precisar saber como ele funciona por dentro.



Basta, para nosso exemplo, conhecermos os seguintes requisitos:

- O botão deve mudar apenas para próximo canal
- O botão deve mudar apenas para o canal anterior
- O botão deve apenas aumentar o volume
- O botão deve apenas diminuir o volume

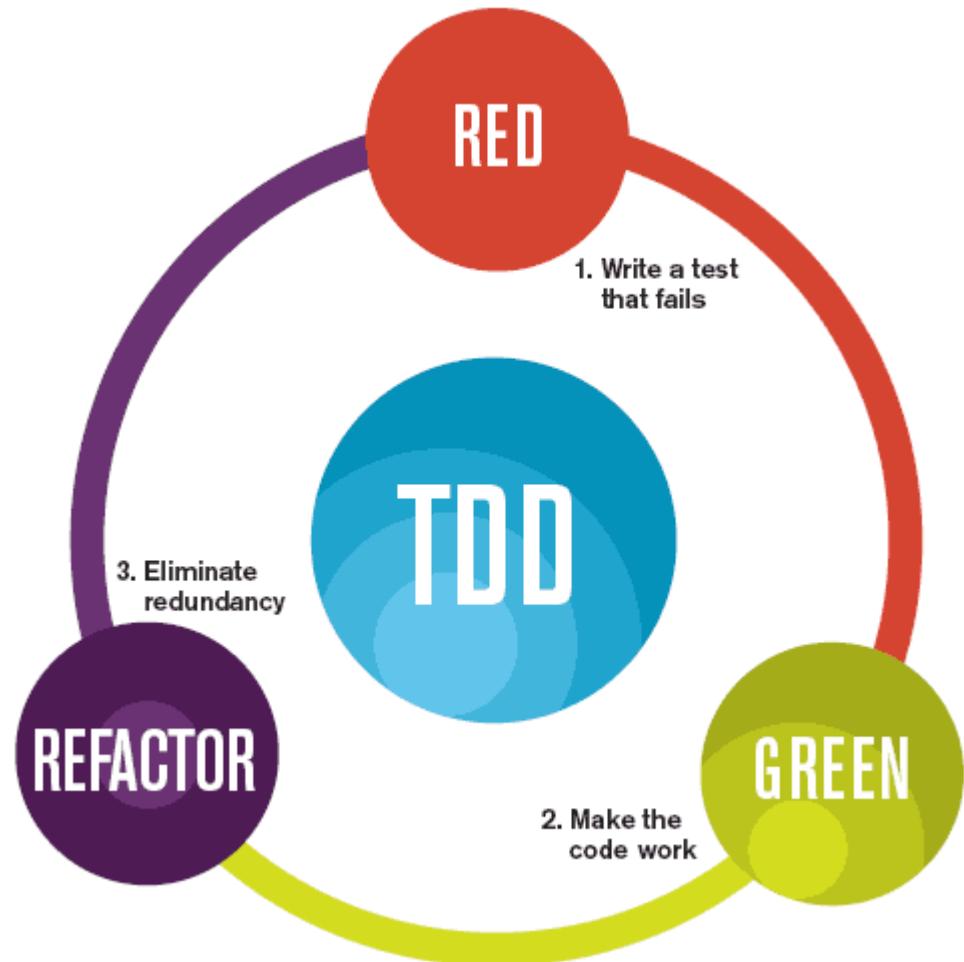
TDD

## Test Driven Development (TDD)

TDD é o desenvolvimento de software orientado a testes, Test Driven Development em inglês. Porém mais do que simplesmente testar seu código, TDD é uma filosofia, uma cultura. E foi fortemente adotado e influenciado pelo movimento ágil.

De acordo com Kent Beck, um método ágil é comparável ao ato de dirigir um carro: você deve observar a estrada e fazer correções contínuas para se manter no caminho. Neste contexto onde a agilidade é fundamental, o testador seria aquele que ajuda o motorista a chegar com segurança ao seu destino, impedindo que sejam feitas conversões incorretas durante o percurso, evitando que o motorista se perca e fazendo com que ele pare e peça instruções quando necessário.

Neste ambiente, destaca-se o TDD, como sendo uma abordagem evolutiva na qual o desenvolvedor escreve o teste antes de escrever o código funcional necessário para satisfazer aquele teste



Possuindo 3 grandes passos:

**Red:** etapa inicial do TDD, onde você escreve um teste que falha, para alguma funcionalidade que você ainda vai escrever.

**Green:** já com o teste criado, é o momento que você precisa fazer o teste passar, lembrando sempre de ir para solução mais simples primeiro.

**Refactor:** etapa para eliminar códigos redundantes, remover acoplamentos e deixar o design de código mais legível.

## **Por que poucos fazem TDD?**

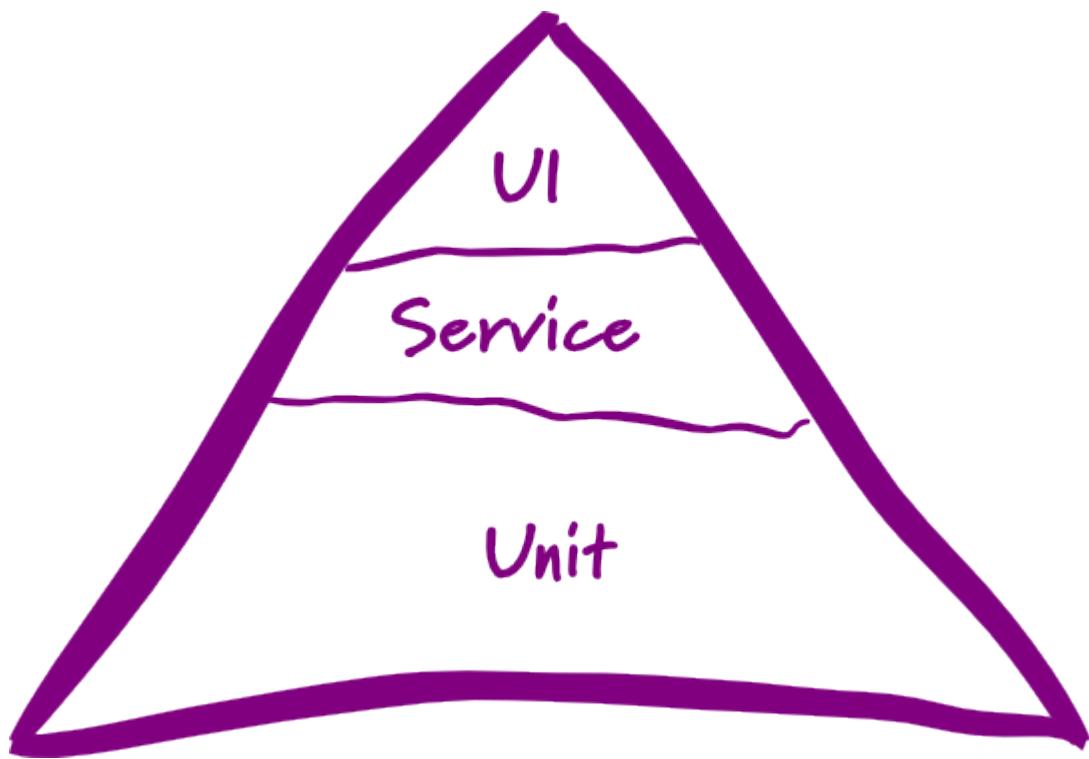
Existem alguns pontos principais que os desenvolvedores falam, quando tentam justificar porque não fazem TDD:

Perda de tempo: os desenvolvedores acham que por ter de escrever além do código, os testes, vai fazer com que demorem mais para desenvolver. Para quê perder tempo aprendendo uma coisa que eu não vejo como útil/importante não é mesmo?

Curva de aprendizado: o estilo de programar muda, é uma nova cultura e para isso, você precisa se adaptar. E é nesse ponto que a maioria desiste, pois não conseguem ver o valor do esforço inicial.

## **O que ganho com TDD?**

Reduz o tempo gasto em depuração e correção de bugs



# Test-Driven Development (TDD)

- Desenvolvimento guiado pelos testes
  - *Só escreva código novo se um teste falhar*
  - *Refatore até que o teste funcione*
  - *Alternância: "red/green/refactor" - nunca passe mais de 10 minutos sem que a barra do JUnit fique verde.*
- Técnicas
  - *"Fake It Til You Make It": faça um teste rodar simplesmente fazendo método retornar constante*
  - *Implementação óbvia: se operações são simples, implemente-as e faça que os testes rodem*

# Testes Automatizados

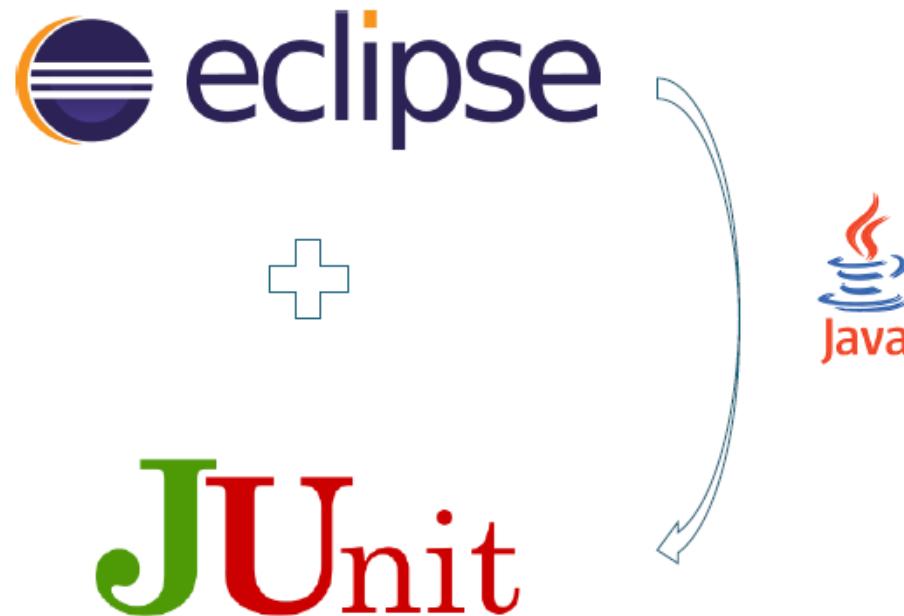
O que são Testes Automáticos?

Testes Manuais x Testes Automáticos

Testes Manuais	Testes Automáticos
Velocidade de execução baixa	Execução muito rápida
Repetitivo e Cansativo	Não cansam e não sentem preguiça.
Não exige tecnologias	Exige domínio de tecnologias específicas
Alto custo a cada execução	Alto custo apenas na criação
Possuem limitações quando o teste envolve situações de grande paralelismo	Permite testar situações impossíveis de testar manualmente
Podem explorar além do cenário de teste, quando necessário	Faz apenas o programado para fazer
Podem avaliar questões visuais como cores e formas	Não avaliam questões visuais
Podem avaliar questões de usabilidade	Não avaliam questões de usabilidade

# Testes Automatizados

## Metodologia e Tecnologias



# Montagem do Ambiente

## Download Free Java Software

<https://java.com/download> • Traduzir esta página

This page is your source to download or update your existing Java Runtime Environment (JRE, Java Runtime), also known as the Java plug-in (plugin), Java ...

## Eclipse Downloads

<https://www.eclipse.org/downloads/> • Traduzir esta página

Download Eclipse Technology that is right for you. IBM. Early registration prices end May 5!

## JUnit - About

[junit.org/](http://junit.org/) • Traduzir esta página

JUnit is a simple framework to write repeatable tests. It is an instance ... JUnit Milestones are available: ... Download and Install - Getting started; Release Notes;

[JUnit](#) • [JUnit API](#) • [Frequently Asked Questions](#) • [Dependency Information](#)

Não precisa fazer download. Acesse nosso Github

# Montagem do Ambiente

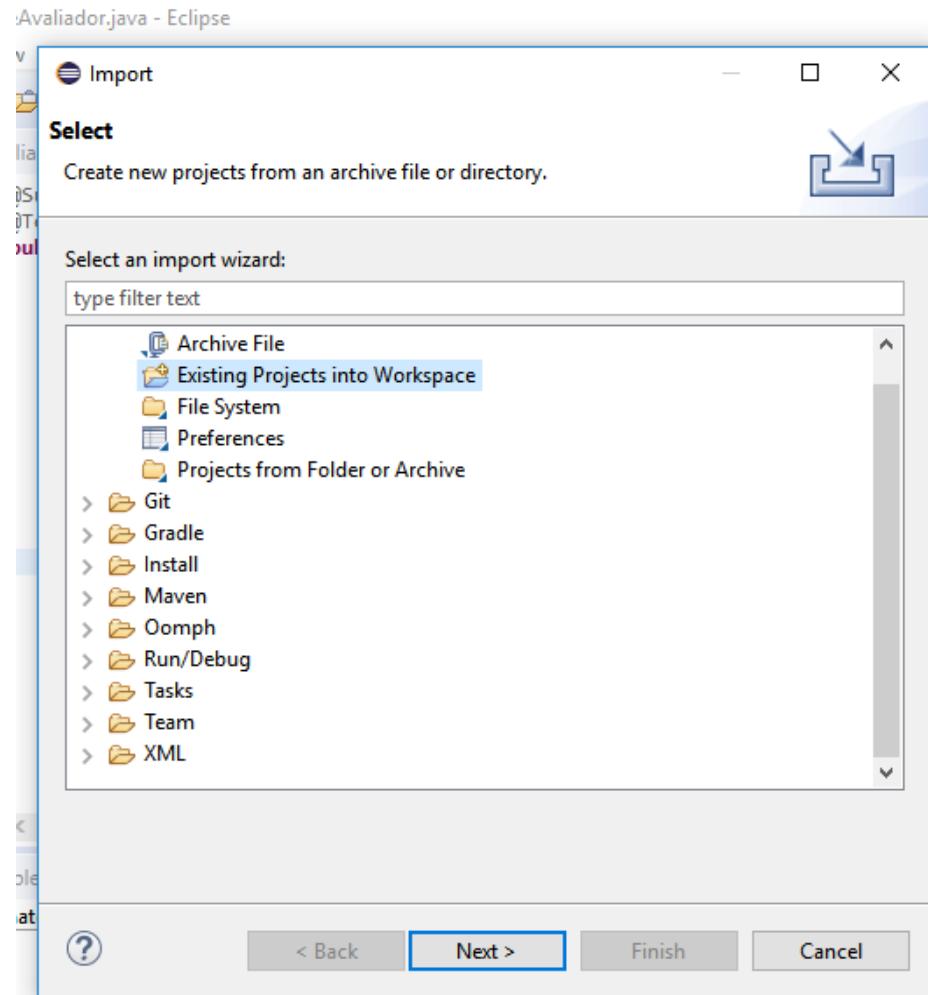
sultados da Pesquisa em Flavitach > eclipse >

Nome	Data de modific...	Tipo	Tar
configuration	15/09/2018 13:54	Pasta de arquivos	
dropins	15/09/2018 13:53	Pasta de arquivos	
plugins	15/09/2018 13:53	Pasta de arquivos	
readme	15/09/2018 13:53	Pasta de arquivos	
.eclipseproduct	22/12/2017 02:01	Arquivo ECLIPSE...	
eclipse	30/03/2018 08:01	Aplicativo	
eclipse	15/09/2018 13:53	Parâmetros de co...	
eclipsec	30/03/2018 08:01	Aplicativo	
hamcrest-core-1.3	13/09/2018 21:32	Executable Jar File	

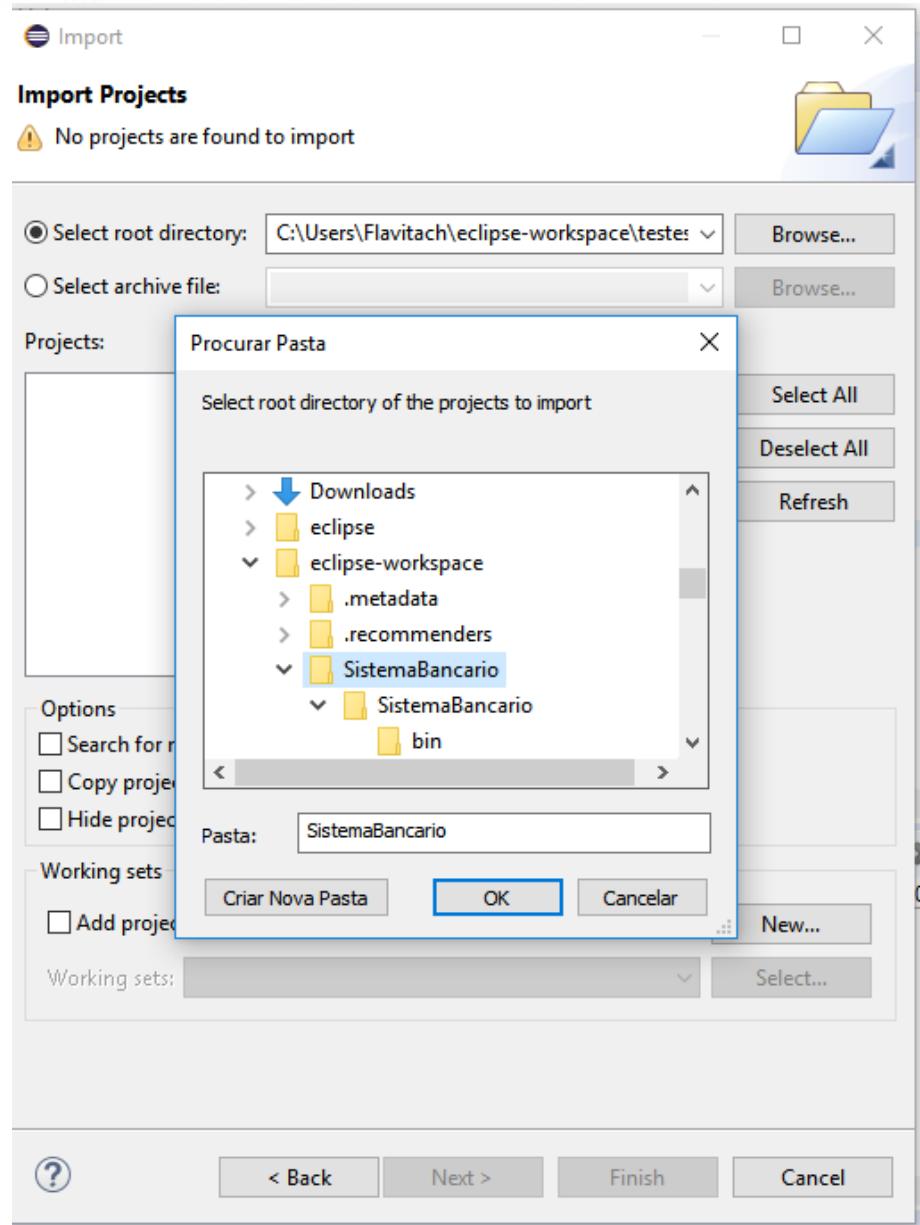
Este Computador > Disco Local (C:) > Usuários > Flavitach > eclipse-workspace

	Nome	Data de modific...	Tipo	Tamanho
alhc	.metadata	15/09/2018 14:17	Pasta de arquivos	
rado	.recommenders	15/09/2018 14:35	Pasta de arquivos	
	SistemaBancario	13/10/2018 00:48	Pasta de arquivos	
	testes-cap1	12/10/2018 19:35	Pasta de arquivos	
	leilao	07/10/2018 02:34	WinRAR ZIP archive	8 KB
	SistemaBancario	13/09/2018 21:32	WinRAR ZIP archive	44 KB
m1	testes-cap1	15/09/2018 13:24	WinRAR ZIP archive	7 KB

# Montagem do Ambiente

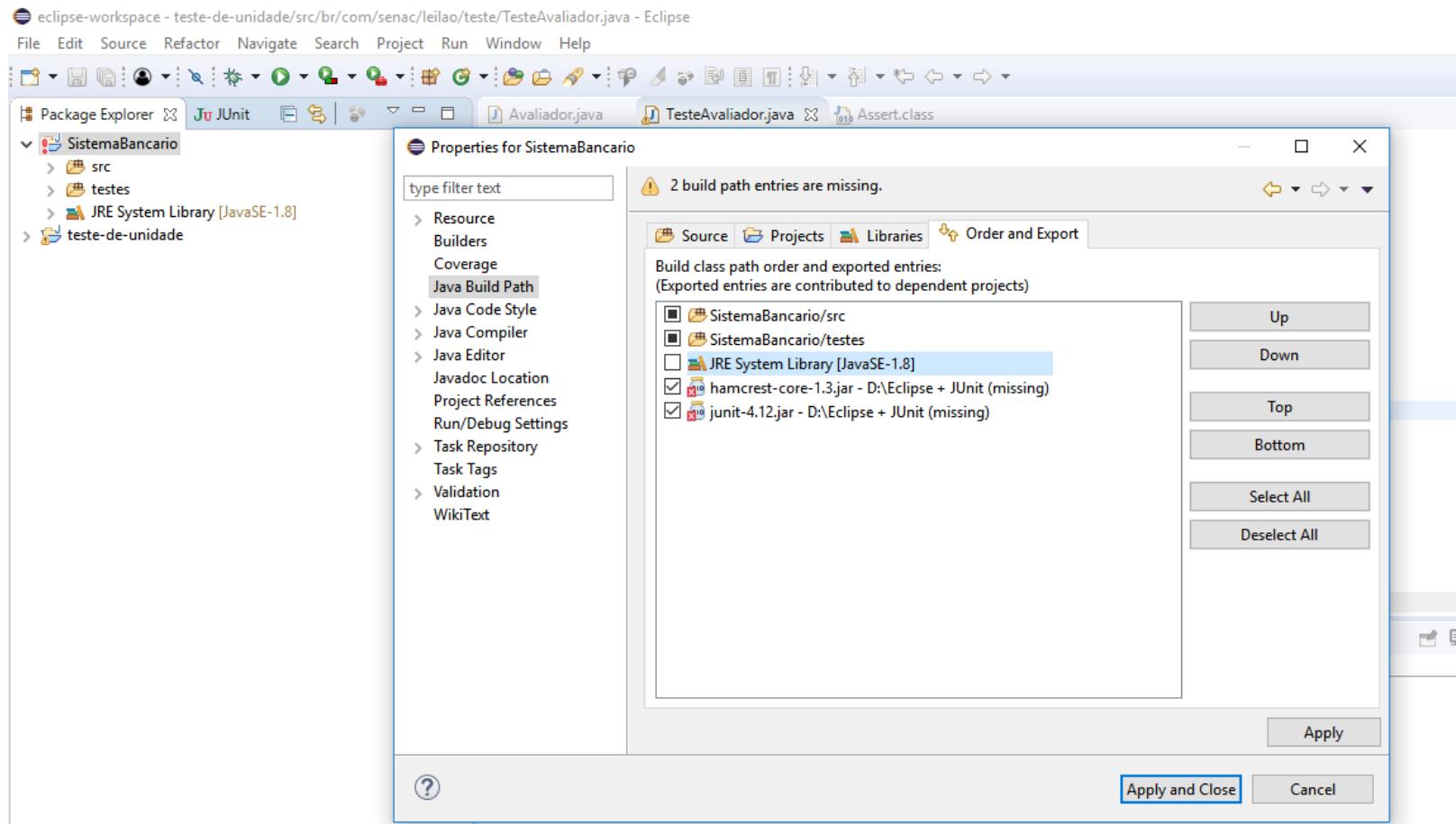


# Montagem do Ambiente



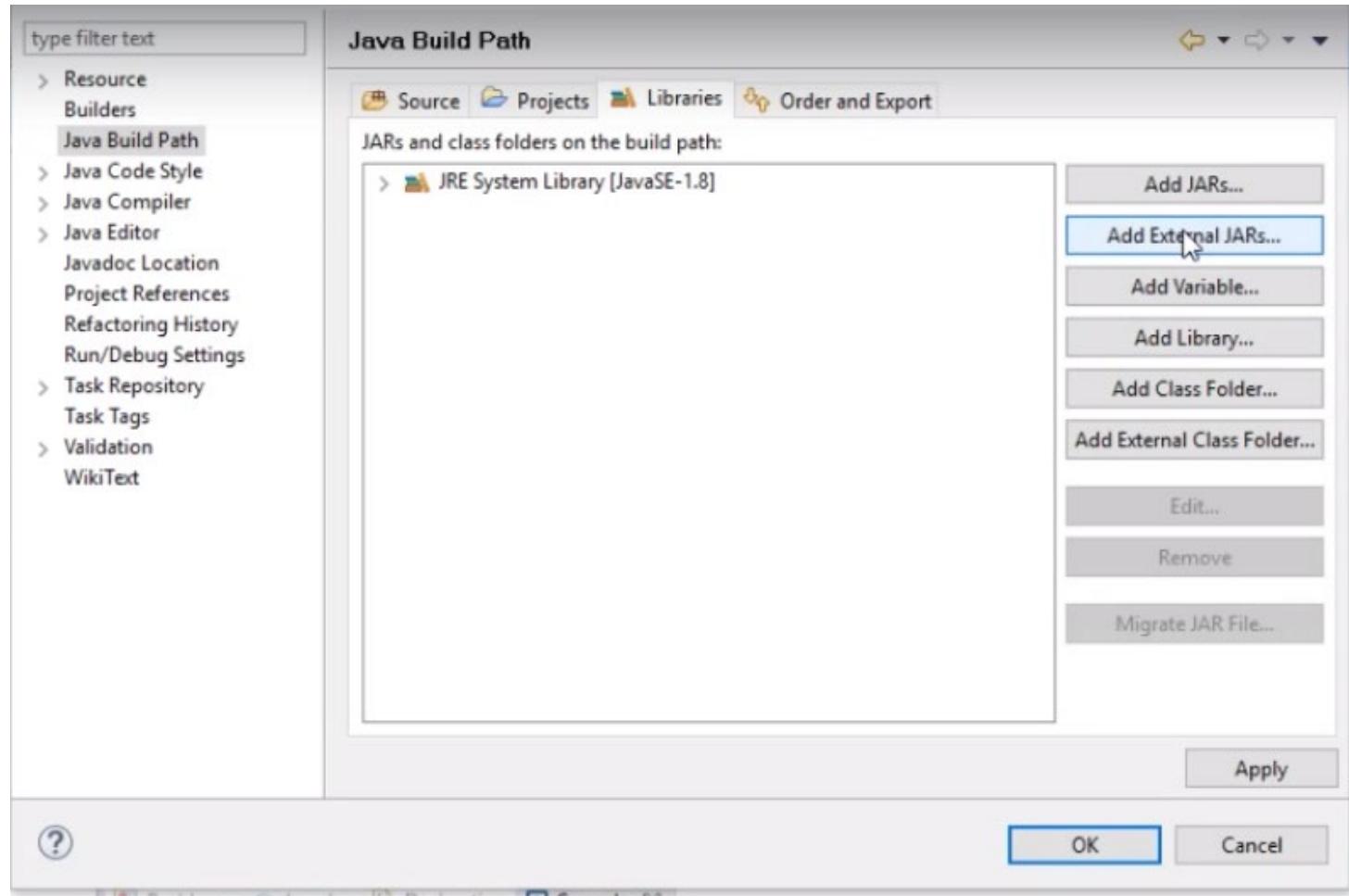
# Montagem do Ambiente

## Botão direito / Properties / Java Build Path

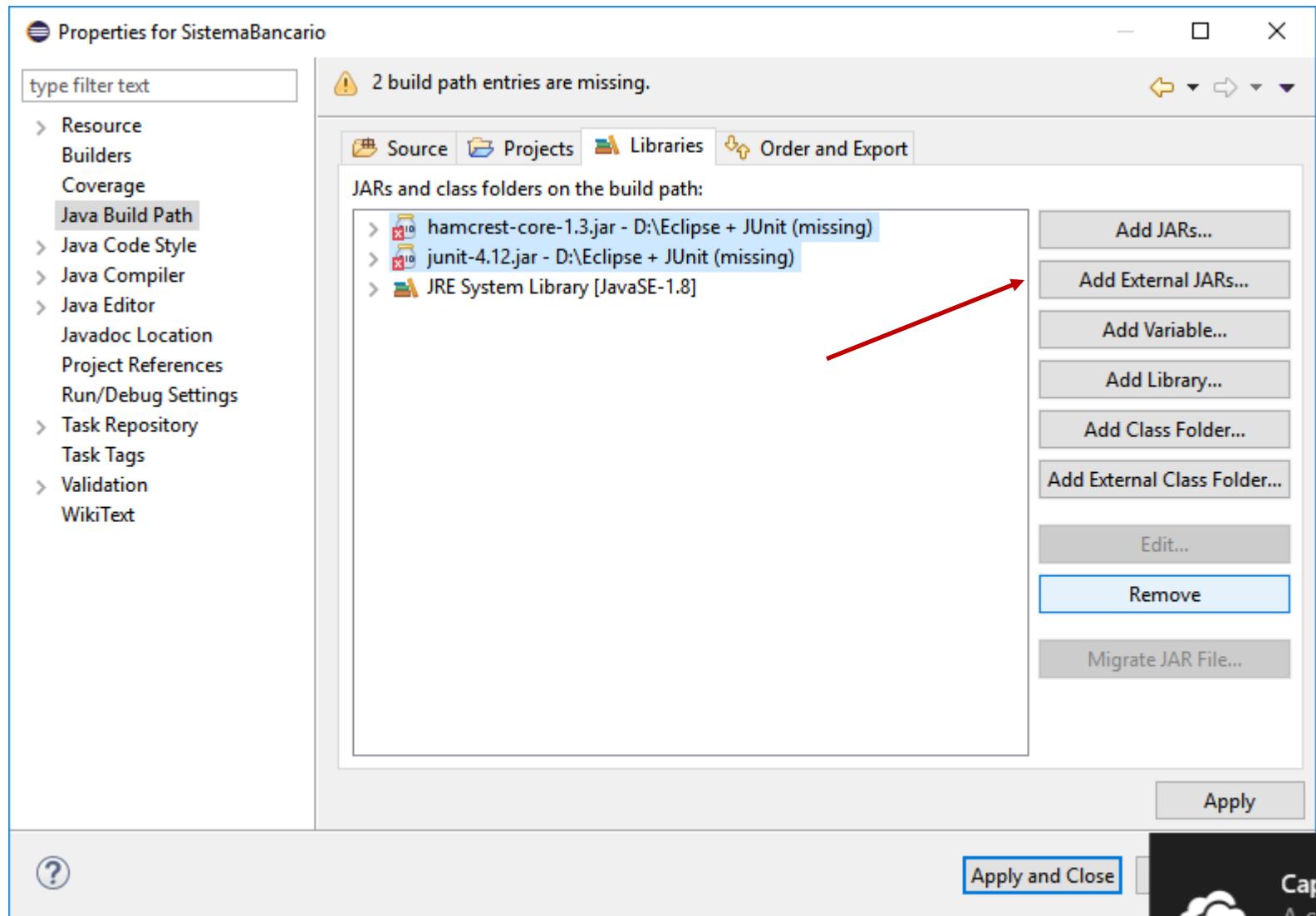


# Testes Automatizados

## Montagem do Ambiente



# Montagem do Ambiente



# Projeto a Ser Testado

# Testes Automatizados

Sistema bancário, com apenas 1 pacote , com a classe Cliente e Conta corrente

The screenshot shows the Eclipse IDE interface with the following details:

- Package Explorer:** Shows the project structure:
  - SistemaBancario** (selected)
  - src**:
    - negocio**: Cliente.java, ContaCorrente.java, GerenciadoraClientes.java, GerenciadoraContas.java, IdadeNaoPermitidaException.java, Main.java
    - testes**:
      - negocio**: GerenciadoraClientesTest\_Ex1.java, GerenciadoraClientesTest\_Ex10.java, GerenciadoraClientesTest\_Ex2.java, GerenciadoraClientesTest\_Ex3.java, GerenciadoraClientesTest\_Ex4.java, GerenciadoraClientesTest\_Ex5.java, GerenciadoraClientesTest\_Ex7.java, GerenciadoraClientesTest\_Ex8.java, GerenciadoraClientesTest\_Ex9.java, GerenciadoraContasTest\_Ex3.java, GerenciadoraContasTest\_Ex4.java, GerenciadoraContasTest\_Ex6.java, GerenciadoraContasTest\_Ex7.java, TodosOsTestes.java
  - JUnit**: Shows several test classes listed in the top bar.
  - Code Editor:** Displays the content of **GerenciadoraClientesTest\_Ex1.java**. The code is a JUnit test for the **GerenciadoraClientes** class, specifically the **pesquisaCliente** method. It creates two clients, adds them to a list, and then performs assertions on the retrieved client's ID and email.
  - Bottom Bar:** Shows tabs for Problems, Javadoc, Declaration, and Console. The message "No consoles to display at this time." is visible in the Console tab.

# Testes Automatizados

The screenshot shows a Java code editor with the following details:

- Toolbar:** Standard Java development toolbar with icons for file operations, search, and navigation.
- Quick Access:** A dropdown menu labeled "Quick Access" containing items like "Recent Projects", "Recent Files", and "Recent Servers".
- Project Explorer:** Shows files in the current project:
  - Cliente.java
  - ContaCorrente.java
  - GerenciadoraClientes.java
  - GerenciadoraContas.java
  - Main.java
  - GerenciadoraClientesTest (selected)
- Code Editor:** The code for `GerenciadoraClientesTest_Ex1`. The code is annotated with line numbers (11 to 34) and highlights:
  - Annotations: `@Test` at line 13.
  - String literals: "Gustavo Farias", "Felipe Augusto", "gugafarias@gmail.com", "felipeaugusto@gmail.com".
  - Class names: `Cliente`, `GerenciadoraClientes`.
  - Method names: `testPesquisaCliente`, `pesquisaCliente`.
  - Variables: `cliente01`, `cliente02`, `clientesDoBanco`.
  - Imports: `List<Cliente>`.
  - Assertions: `assertThat` statements at lines 29 and 30.

```
11 public class GerenciadoraClientesTest_Ex1 {
12
13     @Test
14     public void testPesquisaCliente() {
15
16         // criando alguns clientes
17         Cliente cliente01 = new Cliente(1, "Gustavo Farias", 31, "gugafarias@gmail.com", 1, true);
18         Cliente cliente02 = new Cliente(2, "Felipe Augusto", 34, "felipeaugusto@gmail.com", 1, true);
19
20         // inserindo os clientes criados na lista de clientes do banco
21         List<Cliente> clientesDoBanco = new ArrayList<>();
22         clientesDoBanco.add(cliente01);
23         clientesDoBanco.add(cliente02);
24             |
25         GerenciadoraClientes gerClientes = new GerenciadoraClientes(clientesDoBanco);
26
27         Cliente cliente = gerClientes.pesquisaCliente(1);
28
29         assertThat(cliente.getId(), is(1));
30         assertThat(cliente.getEmail(), is("gugafarias@gmail.com"));
31
32     }
33
34 }
```

# Botão direito /Run AS / Javaapplication

- Eclipse

Run Window Help

The screenshot shows the Eclipse IDE interface. The main window displays the code for Main.java:

```
77
78         if(cliente3 != null){
79             cliente3.setAtivo(false);
80             System.out.println("Cliente desativado com sucesso!");
81         }
82     }
```

Below the code editor, the Eclipse tool bar includes icons for Problems, Javadoc, Declaration, Search, and Console. The Console tab is selected, showing the output of a Java application run:

Main [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (13 de out de 2018 00:58:37)

O que você deseja fazer?

- 1) Consultar por um cliente
- 2) Consultar por uma conta corrente
- 3) Ativar um cliente
- 4) Desativar um cliente
- 5) Sair

1  
Digite o ID do cliente: 1  
=====Id: 1  
Nome: Gustavo Farias  
Email: gugafarias@gmail.com  
Idade: 31  
Status: Ativo  
=====

O que você deseja fazer?

- 1) Consultar por um cliente
- 2) Consultar por uma conta corrente
- 3) Ativar um cliente
- 4) Desativar um cliente
- 5) Sair

# Primeiro Teste

Veja que temos o pacote Negocio, e Pacote Testes

The screenshot shows the Eclipse IDE interface with the following details:

- Left Panel (Package Explorer):** Shows the project structure:
  - SistemaBancario
  - src
    - negocio (containing Cliente.java, ContaCorrente.java, GerenciadoraClientes.java, GerenciadoraContas.java, IdadeNaoPermitidaException.java, Main.java)
    - testes (containing negocio
      - GerenciadoraClientesTest\_Ex1.java, GerenciadoraClientesTest\_Ex10.java, GerenciadoraClientesTest\_Ex2.java, GerenciadoraClientesTest\_Ex3.java, GerenciadoraClientesTest\_Ex4.java, GerenciadoraClientesTest\_Ex5.java, GerenciadoraClientesTest\_Ex7.java, GerenciadoraClientesTest\_Ex8.java, GerenciadoraClientesTest\_Ex9.java, GerenciadoraContasTest\_Ex3.java, GerenciadoraContasTest\_Ex4.java, GerenciadoraContasTest\_Ex6.java, GerenciadoraContasTest\_Ex7.java, TodosOsTestes.java
  - JRE System Library [JavaSE-1.8]
  - Referenced Libraries

**Middle Panel (Code Editor):** Displays the code for `GerenciadoraClientesTest_Ex1.java`:11 public class GerenciadoraClientesTest\_Ex1 {  
12 @Test  
13 public void testPesquisaCliente() {  
14 // criando alguns clientes  
15 Cliente cliente01 = new Cliente("João", "joao@email.com", 18);  
16 Cliente cliente02 = new Cliente("Maria", "maria@email.com", 22);  
17 // inserindo os clientes criados  
18 List<Cliente> clientesDoBanco = new ArrayList<Cliente>();  
19 clientesDoBanco.add(cliente01);  
20 clientesDoBanco.add(cliente02);  
21 GerenciadoraClientes gerClientes = new GerenciadoraClientes();  
22 Cliente cliente = gerClientes.pesquisarPorNome("João");  
23 assertEquals(cliente.getId(), 1);  
24 assertEquals(cliente.getEmail(), "joao@email.com");  
25 }  
26 }  
27  
28  
29  
30  
31  
32 }  
33  
34 }

**Right Panel (Context Menu):** A context menu is open over the code editor, showing options like Open Declaration, Open Type Hierarchy, and Run As.

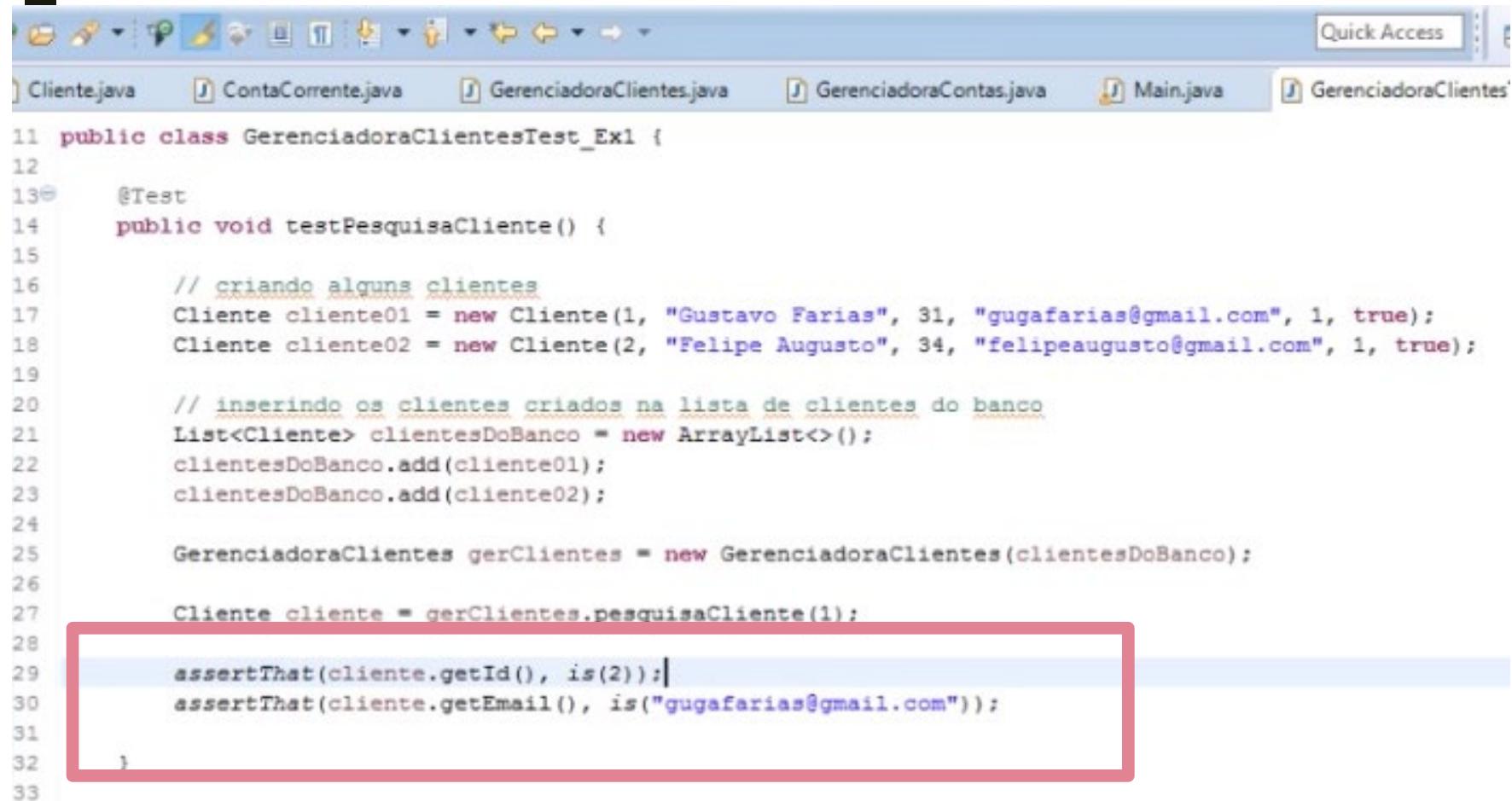
**Bottom Bar:** Shows the JUnit icon, the text "1 JUnit Test", and the keyboard shortcut "Alt+Shift+X, T". It also includes "Run Configurations..." and "No consoles to display".

# Testes Automatizados

The screenshot shows a Java development environment with an integrated test runner. On the left, a results window for 'GerenciadoraClientesTest\_Ex1' shows 1 run, 0 errors, and 0 failures. An arrow points from this window to the test code on the right. The test code, named 'GerenciadoraClientesTest\_Ex1', contains a single test method 'testPesquisaCliente'. The code sets up two clients, inserts them into a list, creates a client manager, and then performs assertions on the retrieved client's ID and email.

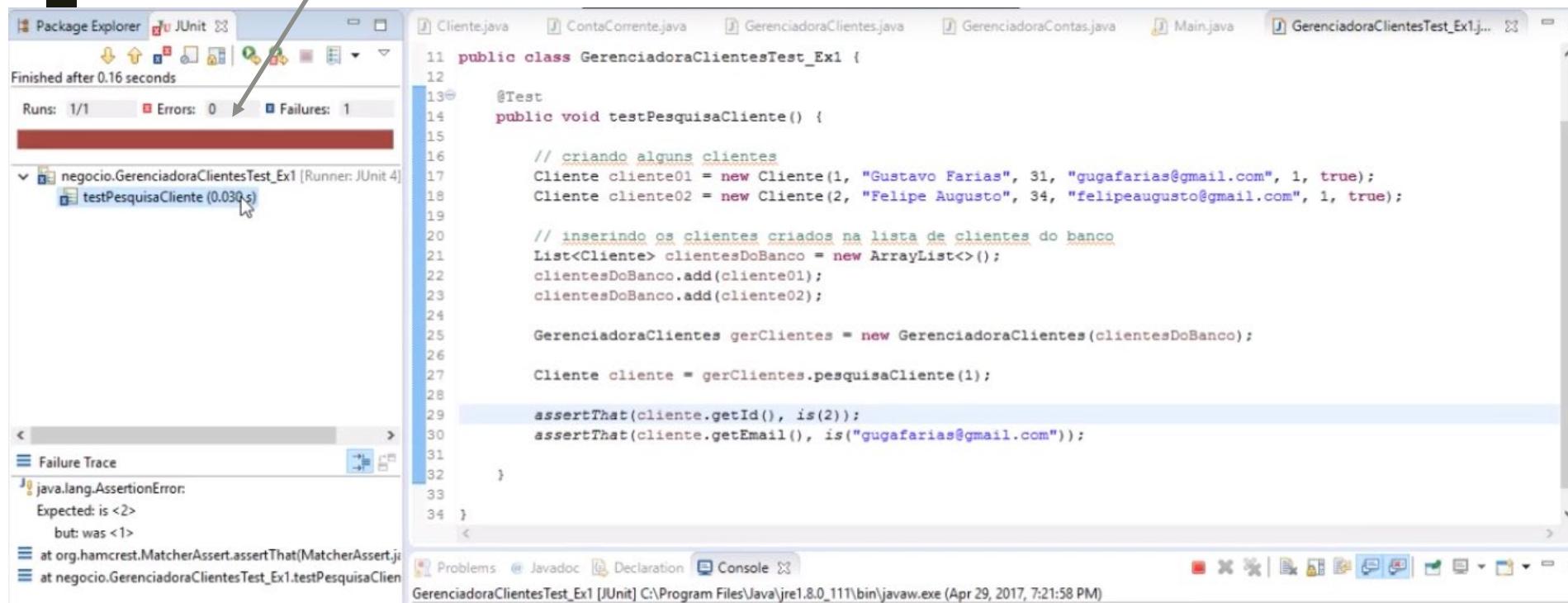
```
11 public class GerenciadoraClientesTest_Ex1 {
12
13     @Test
14     public void testPesquisaCliente() {
15
16         // criando alguns clientes
17         Cliente cliente01 = new Cliente(1, "Gustavo Farias", 31, "gugafarias@gmail.com", 1, true);
18         Cliente cliente02 = new Cliente(2, "Felipe Augusto", 34, "felipeaugusto@gmail.com", 1, true);
19
20         // inserindo os clientes criados na lista de clientes do banco
21         List<Cliente> clientesDoBanco = new ArrayList<>();
22         clientesDoBanco.add(cliente01);
23         clientesDoBanco.add(cliente02);
24
25         GerenciadoraClientes gerClientes = new GerenciadoraClientes(clientesDoBanco);
26
27         Cliente cliente = gerClientes.pesquisaCliente(1);
28
29         assertEquals(cliente.getId(), 1);
30         assertEquals(cliente.getEmail(), "gugafarias@gmail.com");
31     }
32
33
34 }
```

# Testes Automatizados



```
11 public class GerenciadoraClientesTest_Ex1 {
12
13     @Test
14     public void testPesquisaCliente() {
15
16         // criando alguns clientes
17         Cliente cliente01 = new Cliente(1, "Gustavo Farias", 31, "gugafarias@gmail.com", 1, true);
18         Cliente cliente02 = new Cliente(2, "Felipe Augusto", 34, "felipeaugusto@gmail.com", 1, true);
19
20         // inserindo os clientes criados na lista de clientes do banco
21         List<Cliente> clientesDoBanco = new ArrayList<>();
22         clientesDoBanco.add(cliente01);
23         clientesDoBanco.add(cliente02);
24
25         GerenciadoraClientes gerClientes = new GerenciadoraClientes(clientesDoBanco);
26
27         Cliente cliente = gerClientes.pesquisaCliente(1);
28
29         assertThat(cliente.getId(), is(2));
30         assertThat(cliente.getEmail(), is("gugafarias@gmail.com"));
31
32     }
33 }
```

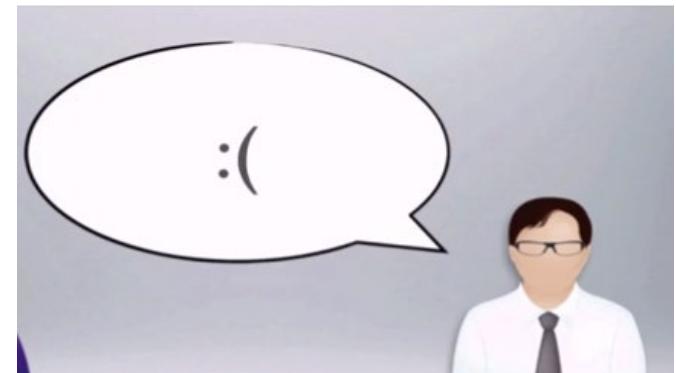
# Testes Automatizados



The screenshot shows a Java development environment with the following details:

- Top Bar:** Package Explorer, JUnit, Main.java, GerenciadoraClientesTest\_Ex1.java.
- Left Sidebar:** Shows "Finished after 0.16 seconds", "Runs: 1/1", "Errors: 0", and "Failures: 1". A mouse cursor is hovering over the "Failure" link.
- Middle Area:** Displays the source code for `GerenciadoraClientesTest_Ex1`. The code is a JUnit test for searching clients. It creates two clients, adds them to a bank list, and then searches for the client with ID 2. The failure trace indicates an `AssertionError` where the expected ID was 2 but the actual was 1.
- Bottom Bar:** Problems, Javadoc, Declaration, Console, and a status bar showing "GerenciadoraClientesTest\_Ex1 [JUnit] C:\Program Files\Java\jre1.8.0\_111\bin\javaw.exe (Apr 29, 2017, 7:21:58 PM)".

Bugs (Infelizmente) são comuns



# Sistema de Leilão –

*Onde vamos dar lances e ao final teremos uma vencedor*

- Vamos acessar nosso Github e baixar o arquivo **Teste\_Leilão.zip**
- Agora vamos descompactar em nossa Worksapce (Eclipse)

***Exemplo: C:\Users\senacaluno\workspace1***



Import



## Select

Create new projects from an archive file or directory.



Select an import wizard:

type filter text

- Archive File
  - Existing Projects into Workspace
  - File System
  - Preferences
  - Projects from Folder or Archive
- > Git
- > Gradle
- > Install
- > Maven
- > Oomph
- > Run/Debug
- > Tasks
- > Team
- > XML



< Back

Next >

Finish

Cancel

## Import

### Import Projects

Select a directory to search for existing Eclipse projects.



Select root directory: C:\Users\Flavitach\workspace1

Select archive file:

#### Projects:

- exercicios (C:\Users\Flavitach\workspace1\exercicios)
- jdbc (C:\Users\Flavitach\workspace1\jdbc)
- Livraria (C:\Users\Flavitach\workspace1\Livraria)
- LivroOO (C:\Users\Flavitach\workspace1\LivroOO)
- ObjetoCalculadora (C:\Users\Flavitach\workspace1\ObjetoCal)
- Poli (C:\Users\Flavitach\workspace1\Poli)
- TESTE (C:\Users\Flavitach\workspace1\TESTE)
- teste-de-unidade (C:\Users\Flavitach\workspace1\leilao\leilao)

#### Options

- Search for nested projects
- Copy projects into workspace
- Hide projects that already exist in the workspace

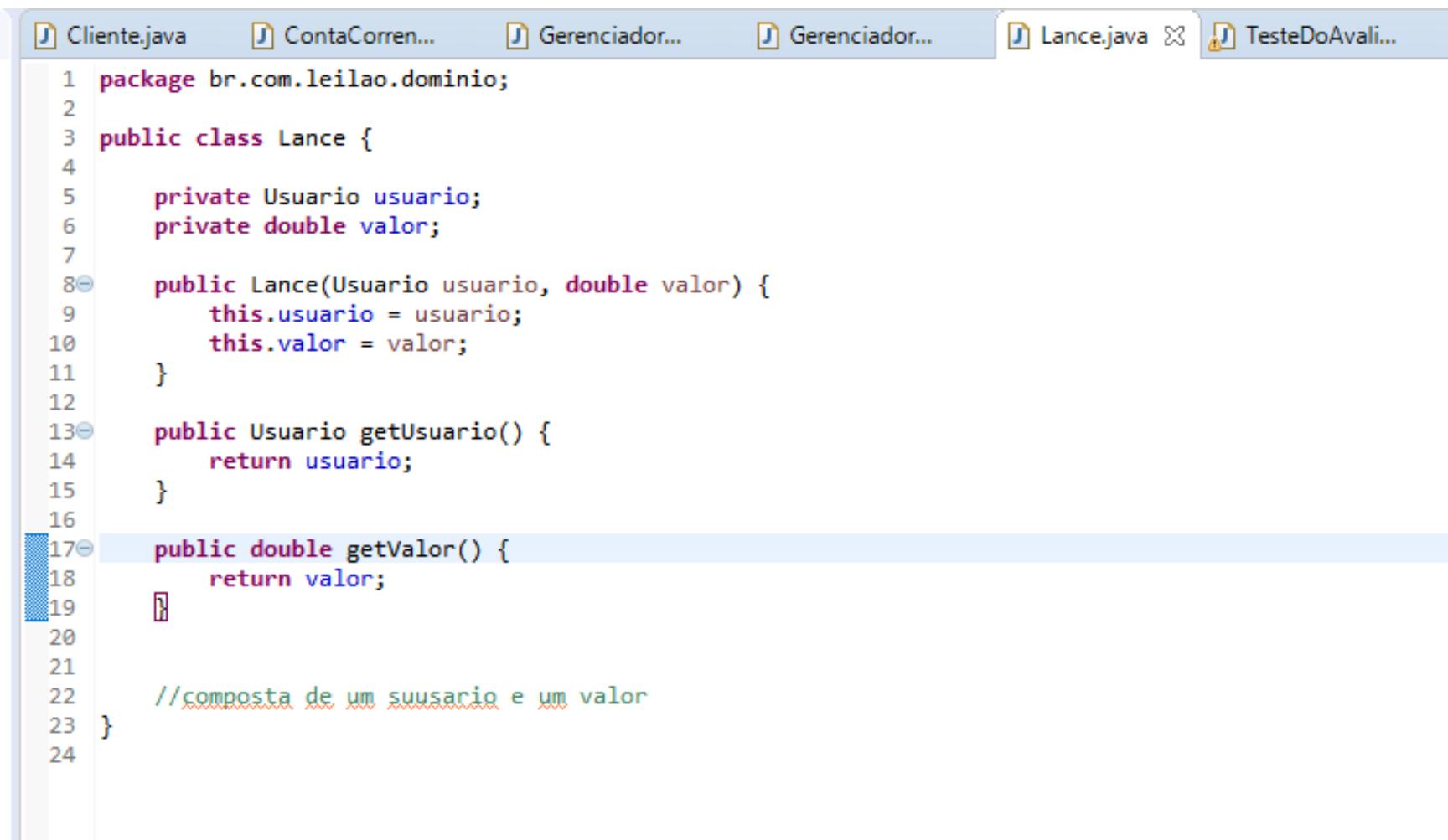
#### Working sets

Add project to working sets

Working sets:



## Vamos navegar pelas nossas classes: Classe Lance, composta de usuário e um valor



The screenshot shows a Java code editor with the file `Lance.java` open. The code defines a class `Lance` with fields `Usuario` and `double`, and methods to set and get them. A comment at the bottom indicates it's composed of a user and a value.

```
1 package br.com.leilao.dominio;
2
3 public class Lance {
4
5     private Usuario usuario;
6     private double valor;
7
8     public Lance(Usuario usuario, double valor) {
9         this.usuario = usuario;
10        this.valor = valor;
11    }
12
13     public Usuario getUsuario() {
14         return usuario;
15     }
16
17     public double getValor() {
18         return valor;
19     }
20
21
22     //composta de um suusario e um valor
23 }
24
```

## Classe Leilão, composta de vários lances e descrição

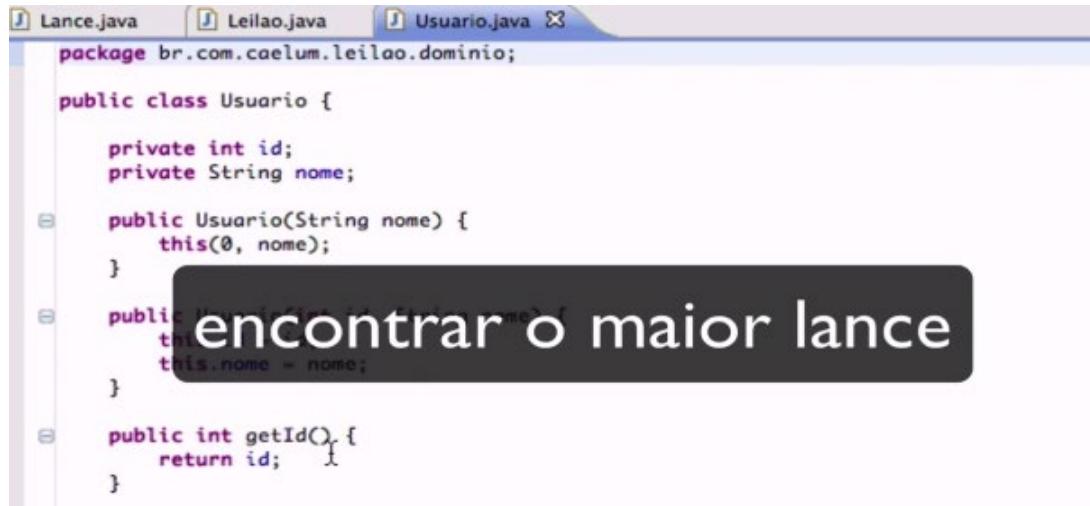
The screenshot shows a Java code editor with the tab bar at the top containing several tabs: Cliente.java, ContaCorren..., Gerenciador..., Lance.java, Leilao.java, and another tab partially visible. The Leilao.java tab is currently selected. The code editor displays the following Java code:

```
1 package br.com.leilao.dominio;
2
3 import java.util.ArrayList;
4
5 public class Leilao {
6
7     private String descricao;
8     private List<Lance> lances;
9
10    public Leilao(String descricao) {
11        this.descricao = descricao;
12        this.lances = new ArrayList<Lance>();
13    }
14
15    public void propoe(Lance lance) {
16        lances.add(lance);
17    }
18
19    public String getDescricao() {
20        return descricao;
21    }
22
23    public List<Lance> getLances() {
24        return Collections.unmodifiableList(lances);
25    }
26
27
28
29    //composta por varios lances e descricao
30
31}
```

The code defines a class named Leilao with private fields for descricao and lances. It includes a constructor, a method to add a lance, a method to get the description, and a method to get an unmodifiable list of lances. A comment at the bottom indicates the class is composed of various lances and a description.

```
1 package br.com.leilao.dominio;
2
3 public class Usuario {
4
5     private int id;
6     private String nome;
7
8     public Usuario(String nome) {
9         this(0, nome);
10    }
11
12    public Usuario(int id, String nome) {
13        this.id = id;
14        this.nome = nome;
15    }
16
17    public int getId() {
18        return id;
19    }
20
21    public String getNome() {
22        return nome;
23    }
24
25
26    //composta por um id e nome
27    //meta: dado um leilao qual o maior lance dado
28    //vamos criar uma classe avaliado no pacote e
29    //vmos escrever o metodo avali q e publico e nao retorna nada
30}
31}
```

*Vamos implementar uma funcionalidade para verificar qual o maior lance*



```
package br.com.caelum.leilao.dominio;

public class Usuario {

    private int id;
    private String nome;

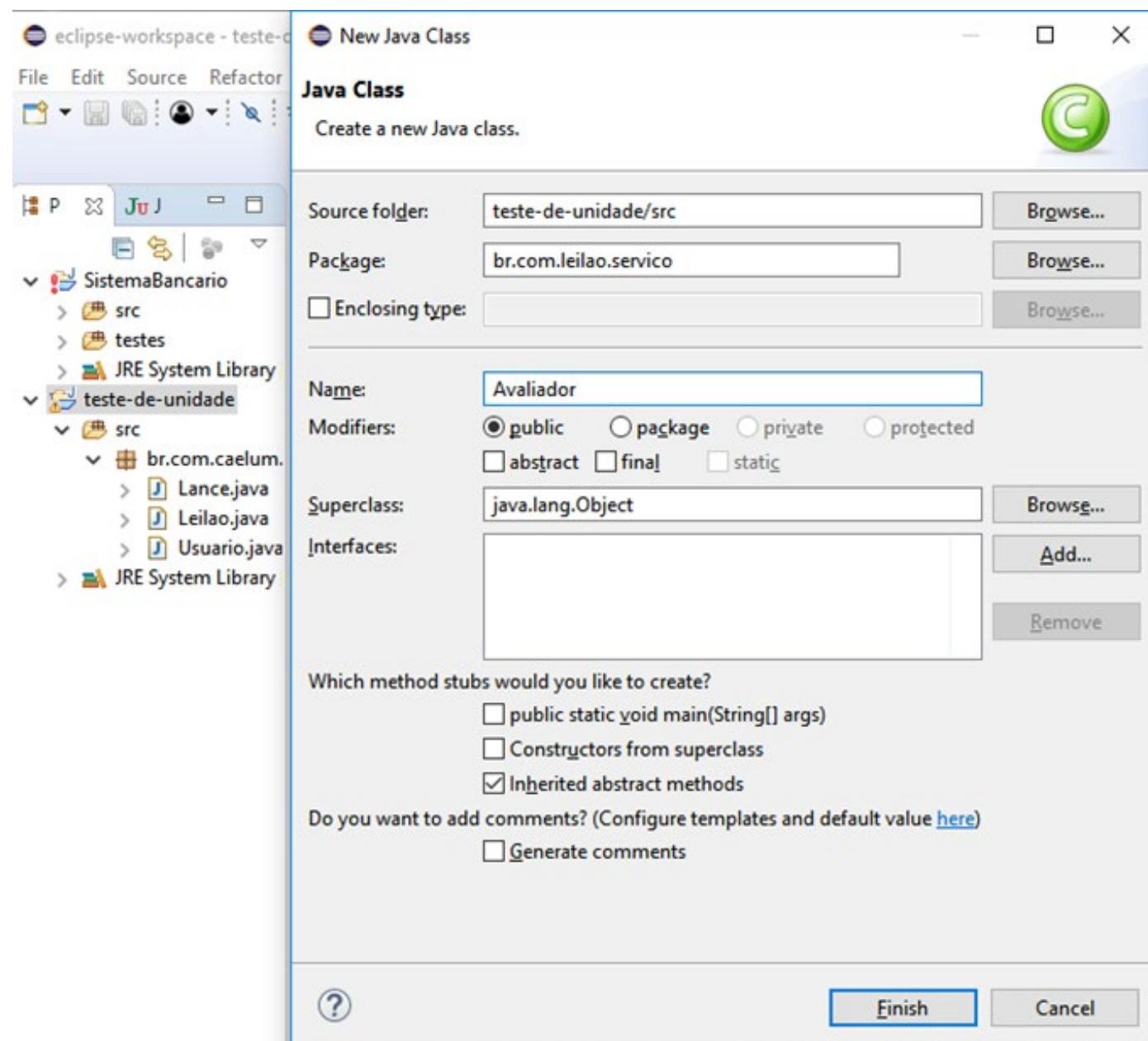
    public Usuario(String nome) {
        this(0, nome);
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

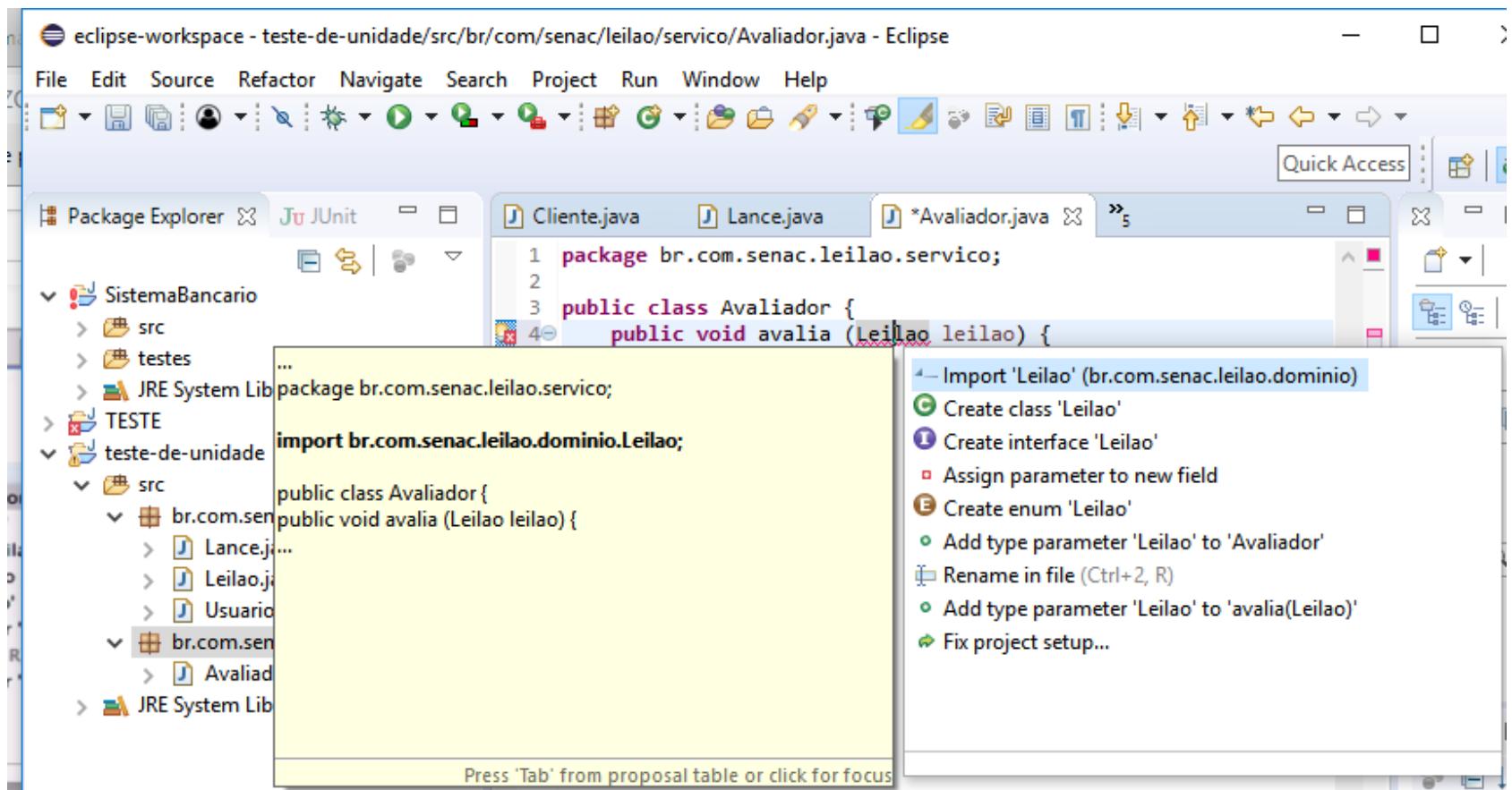
    public int getId() {
        return id;
    }
}
```

encontrar o maior lance

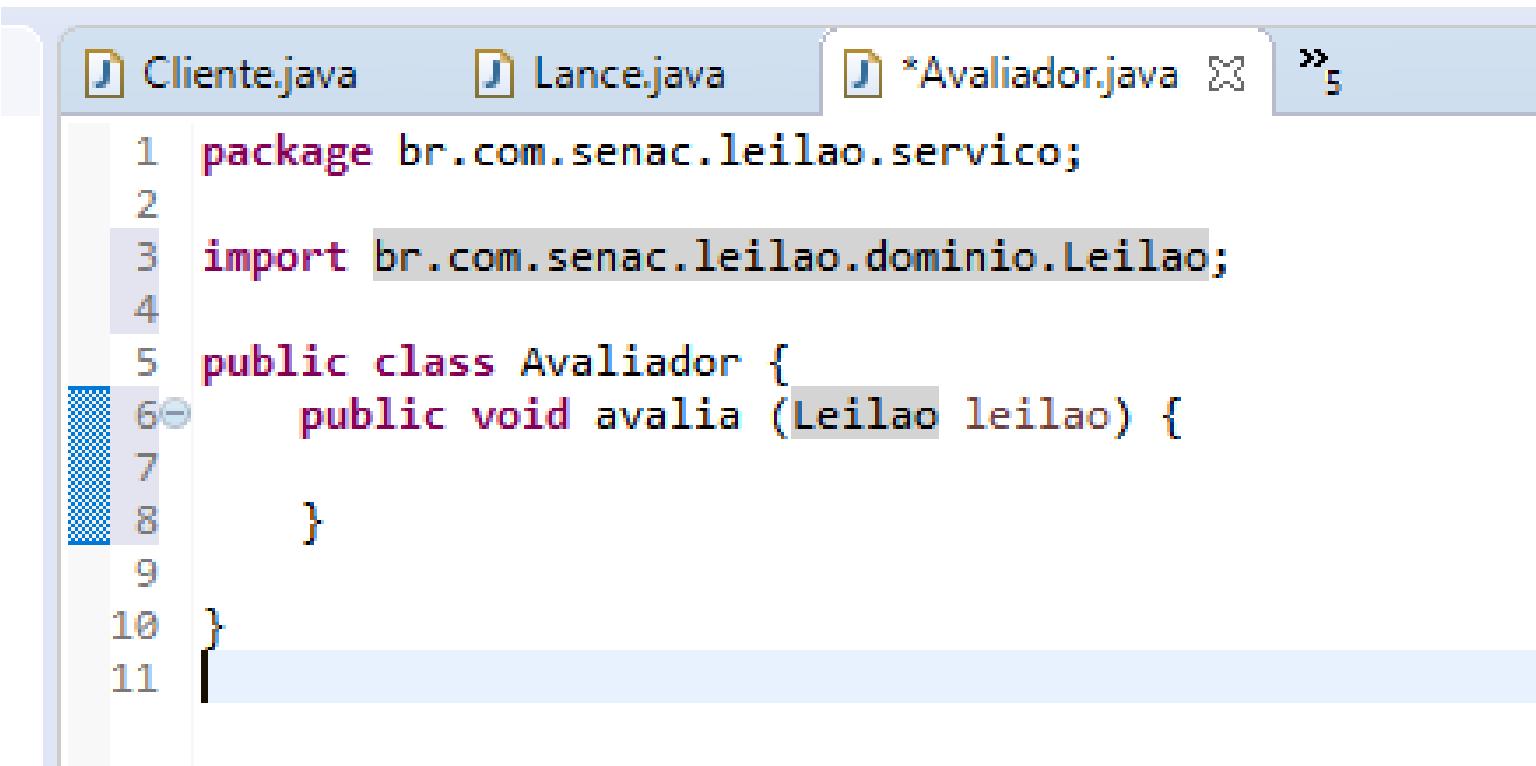
# Vamos criar a classe Avaliador no pacote Serviço



Vamos agora criar um método *Avalia*, que é público e não retorna nada! Este método irá receber um *leilão* e para isso precisamos importar a classe *Leilão* já criada. Para isso , clique *CTRL+1* em cima da palavra *leilão*.



*Vamos percorrer todos os lances deste leilão e encontrar o maior Valor*

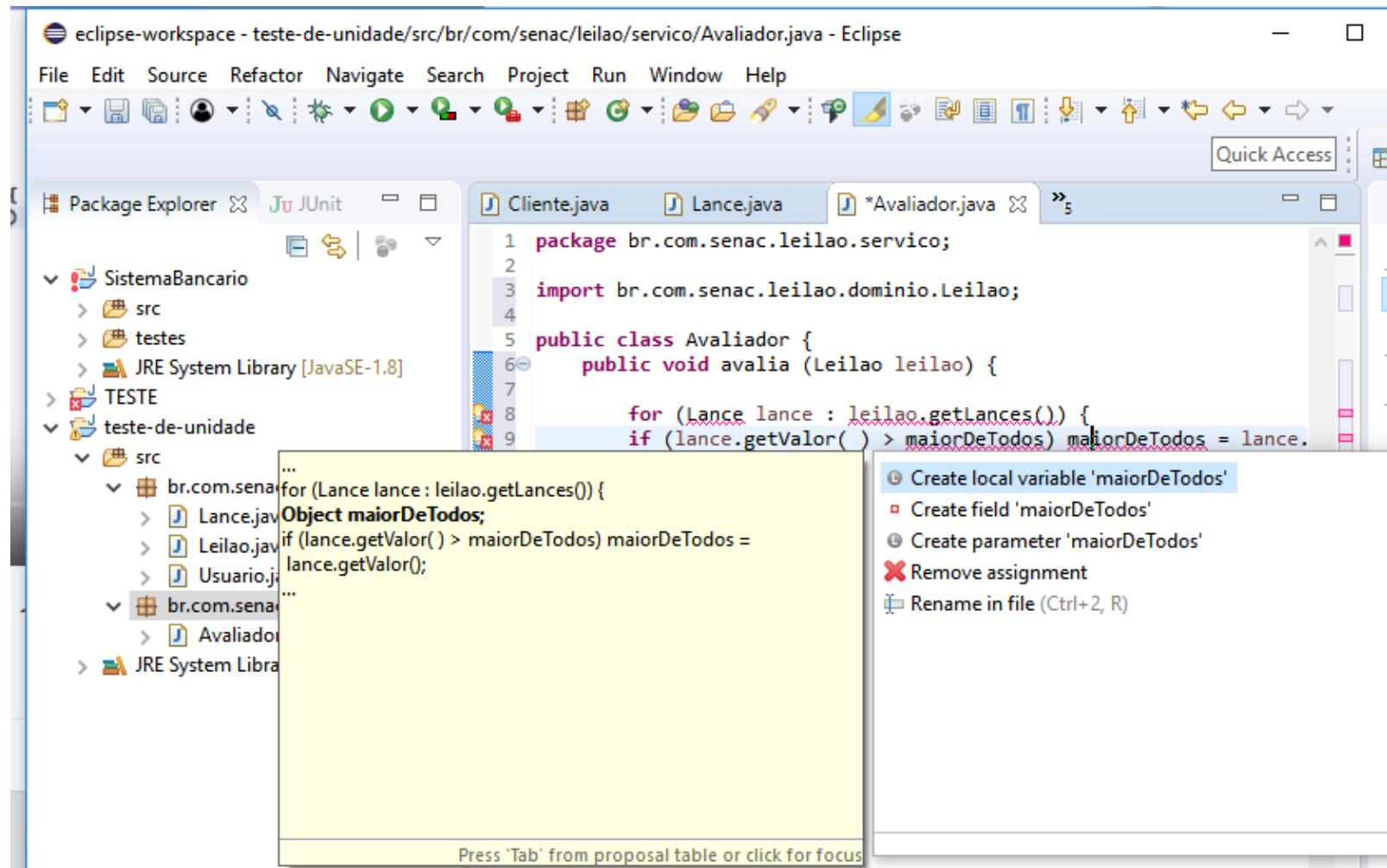


The screenshot shows a Java code editor with three tabs at the top: Cliente.java, Lance.java, and \*Avaliador.java. The \*Avaliador.java tab is selected. The code in the editor is as follows:

```
1 package br.com.senac.leilao.servico;
2
3 import br.com.senac.leilao.dominio.Leilao;
4
5 public class Avaliador {
6     public void avalia (Leilao leilao) {
7
8     }
9
10 }
11
```

The code defines a class named Avaliador with a single method, avalia, which takes a Leilao object as a parameter. The code is numbered from 1 to 11 on the left side.

*Será necessário criar um atributo da classe, em cima deste método, para isso vamos novamente CTRL +1 - Create Field*



*Vamos percorrer todos os lances deste leilão e encontrar o maior Valor*

The screenshot shows the Eclipse IDE interface with the title bar "eclipse-workspace - teste-de-unidade/src(br.com.senac.leilao.servico/Avaliador.java) - Eclipse". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, Find, and Run. The left sidebar displays the project structure under "SistemaBancario" with packages like "teste-de-unidade" and its "src" folder containing classes such as "Lance.java", "Leilao.java", and "Usuario.java". The main editor window shows the code for the "Avaliador.java" class:

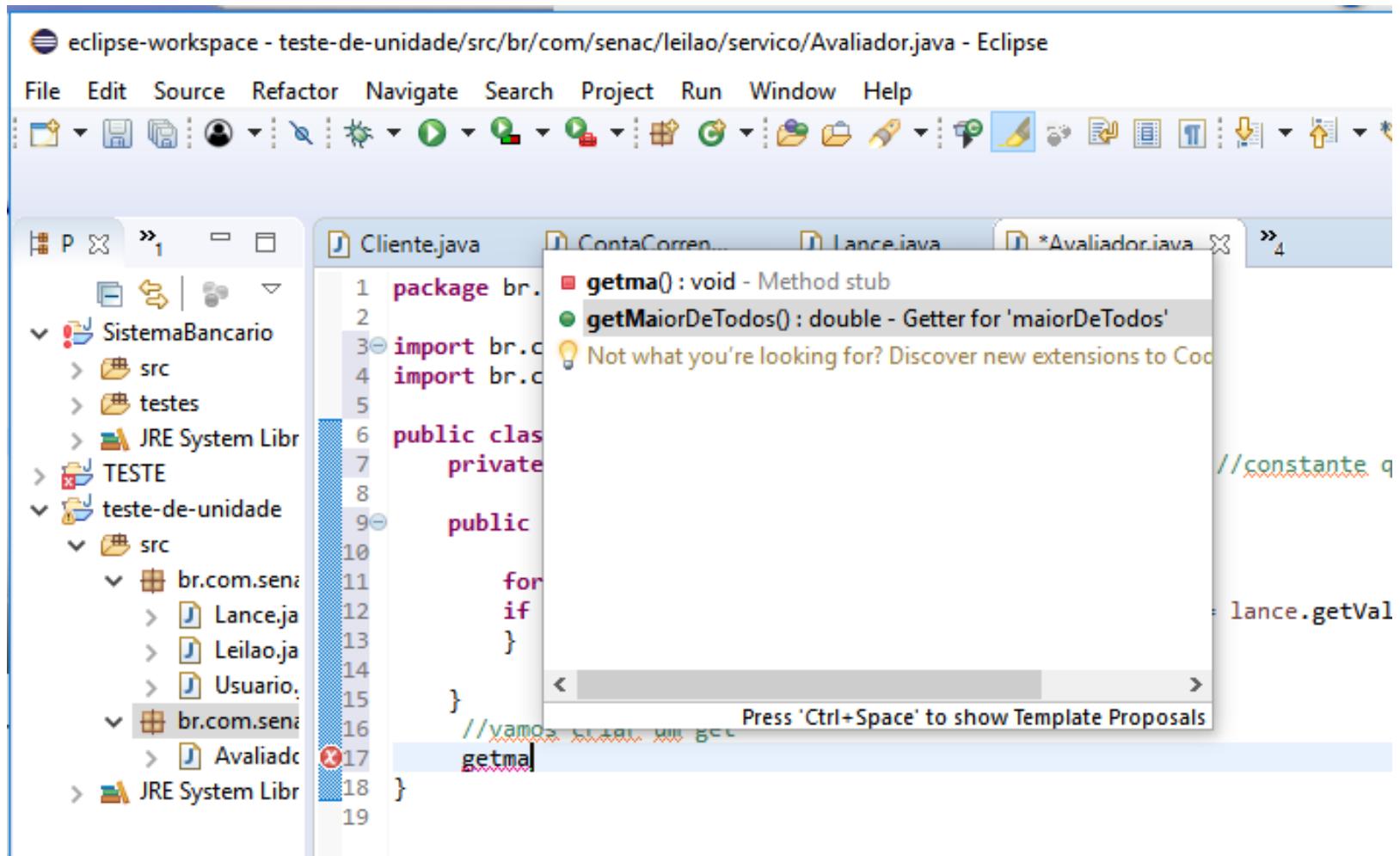
```
1 package br.com.senac.leilao.servico;
2
3 import br.com.senac.leilao.dominio.Lance;
4 import br.com.senac.leilao.dominio.Leilao;
5
6 public class Avaliador {
7     private double maiorDeTodos;
8
9     public void avalia (Leilao leilao) {
10
11         for (Lance lance : leilao.getLances()) {
12             if (lance.getValor( ) > maiorDeTodos) maiorDeTodos = lance.getValor();
13         }
14     }
15 }
16
17 }
18 }
```

*Alterar para Double e setar a constante negative.Infinity onde esta constante q guarda o menor numero q cabe dentro de um double.*

The screenshot shows the Eclipse IDE interface with the title bar "eclipse-workspace - teste-de-unidade/src/br/com/senac/leilao/servico/Avaliador.java - Eclipse". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, Copy, Paste, and Cut. The quick access bar on the right has icons for search, replace, and other tools. The left side shows the project structure in the Package Explorer with packages like Sistematico, TESTE, and teste-de-unidade, and source files like Lance.java, Leilao.java, Usuario.java, and Avaliador.java. The right side is the code editor for "Avaliador.java" with the following content:

```
1 package br.com.senac.leilao.servico;
2
3 import br.com.senac.leilao.dominio.Lance;
4 import br.com.senac.leilao.dominio.Leilao;
5
6 public class Avaliador {
7     private double maiorDeTodos= Double.NEGATIVE_INFINITY;
8
9     public void avalia (Leilao leilao) {
10
11         for (Lance lance : leilao.getLances()) {
12             if (lance.getValor( ) > maiorDeTodos) maiorDeTodos = lance.getValor();
13
14         }
15     }
16 }
17
18 }
```

Vamos criar um método get, para isso digite getma  
CTRL+space e alterar o nome para para getMaiorLance



Cliente.java

ContaComen...

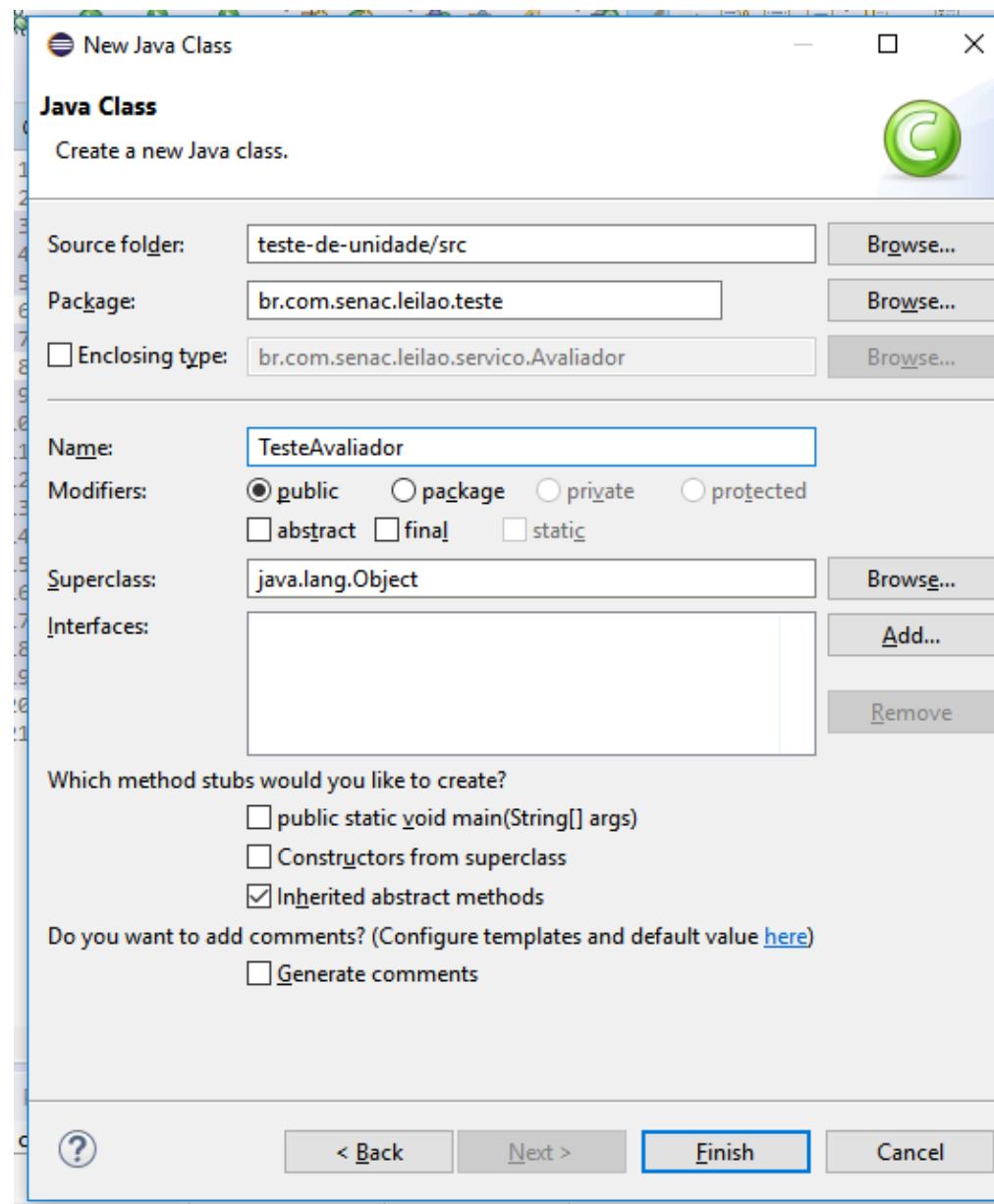
Lance.java

\*Avaliador.java

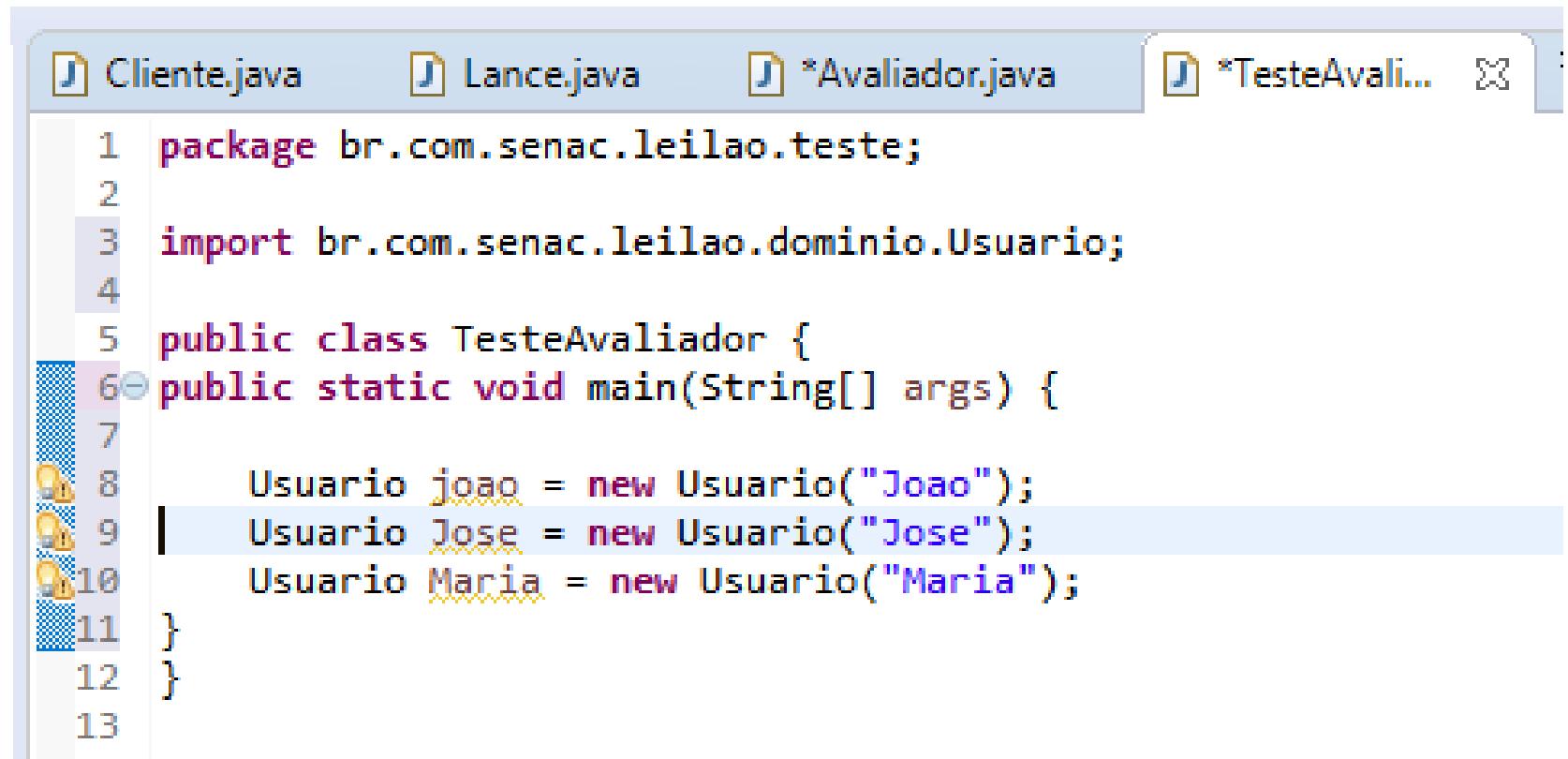
»  
4

```
1 package br.com.senac.leilao.servico;
2
3 import br.com.senac.leilao.dominio.Lance;
4 import br.com.senac.leilao.dominio.Leilao;
5
6 public class Avaliador {
7     private double maiorDeTodos= Double.NEGATIVE_INFINITY; //constante q guarda
8
9     public void avalia (Leilao leilao) {
10
11         for (Lance lance : leilao.getLances()) {
12             if (lance.getValor( ) > maiorDeTodos) maiorDeTodos = lance.getValor();
13         }
14
15     }
16     //vamos criar um get
17     public double getMaiorDeTodos() {
18         return maiorDeTodos;
19     }
20 }
21 }
```

# Vamos criar uma classe main ctrl+n , no pacote Teste.



Na classe Main TesteAvaliador vamos instanciar nossos objetos.



The screenshot shows a Java code editor with several tabs at the top: Cliente.java, Lance.java, \*Avaliador.java, and \*TesteAvali... (the latter two are faded). The code in the editor is as follows:

```
1 package br.com.senac.leilao.teste;
2
3 import br.com.senac.leilao.dominio.Usuario;
4
5 public class TesteAvaliador {
6     public static void main(String[] args) {
7
8         Usuario joao = new Usuario("Joao");
9         Usuario Jose = new Usuario("Jose");
10        Usuario Maria = new Usuario("Maria");
11    }
12 }
13
```

The code creates three instances of the Usuario class: joao, Jose, and Maria. The line 'Usuario Jose = new Usuario("Jose");' is highlighted with a light blue background.

Cliente.java

Lance.java

\*Avaliador.java

\*TesteAvali...

>> 5

```
1 package br.com.senac.leilao.teste;
2
3 import br.com.senac.leilao.dominio.Lance;
4 import br.com.senac.leilao.dominio.Leilao;
5 import br.com.senac.leilao.dominio.Usuario;
6 import br.com.senac.leilao.servico.Avaliador;
7
8 public class TesteAvaliador {
9     public static void main(String[] args) {
10
11         Usuario joao = new Usuario("Joao");
12         Usuario Jose = new Usuario("Jose");
13         Usuario Maria = new Usuario("Maria");
14
15         //vamos criar um leilao e propor lances nesse leilao
16         Leilao leilao = new Leilao("Playstation 3 novo");
17
18         leilao.propoe(new Lance(joao, 300.0));
19         leilao.propoe(new Lance(Jose, 400.0));
20         leilao.propoe(new Lance(Maria, 250.0));
21
22         Avaliador leiloeiro = new Avaliador();
23         leiloeiro.avalia(leilao);
24
25         System.out.println(leiloeiro.getMaiorLance());
26     }
27 }
```

```

1 package br.com.senac.leilao.teste;
2
3 import br.com.senac.leilao.dominio.Lance;
4 import br.com.senac.leilao.dominio.Leilao;
5 import br.com.senac.leilao.dominio.Usuario;
6 import br.com.senac.leilao.servico.Avaliador;
7
8 public class TesteAvaliador {
9 public static void main(String[] args) {
10
11     Usuario joao = new Usuario("Joao");
12     Usuario Jose = new Usuario("Jose");
13     Usuario Maria = new Usuario("Maria");
14
15     //vamos criar um leilao e propor lances nesse leilao
16     Leilao leilao = new Leilao("Playstation 3 novo");
17
18     leilao.propoe(new Lance(joao, 300.0));
19     leilao.propoe(new Lance(Jose, 400.0));
20     leilao.propoe(new Lance(Maria, 250.0));
21
22     //leiloeiro ira avaliar o nosso valor no leilao
23     Avaliador leiloeiro = new Avaliador();
24     leiloeiro.avalia(leilao);
25
26     System.out.println(leiloeiro.getMaiorLance()); //com isso o resultado deve ser 400.0
27 }

```

Não esquecer importar  
as classes criadas  
anteriormente - CTRL + 1



# Uma certeza que temos!

```
public static void main(String[] args) {  
    Usuario joao = new Usuario("João");
```

# Software muda o tempo todo!

Agora, além de encontrar o maior lance ,nossa classe também irá:

```
ic static void main(String[] args) {  
    Usuario joao = new Usuario("João");  
    Usuario maria = new Usuario("Maria");  
    Usuario lucas = new Usuario("Lucas");  
  
    encontrar o m  
    lucas.nome = "Lucas";
```

# Inserido condição

```
1 package br.com.senac.leilao.servico;
2
3 import br.com.senac.leilao.dominio.Lance;
4 import br.com.senac.leilao.dominio.Leilao;
5
6 public class Avaliador {
7     private double maiorDeTodos= Double.NEGATIVE_INFINITY; //constante q guarda o menor valor
8
9     public void avalia (Leilao leilao) {
10
11         for (Lance lance : leilao.getLances()) {
12             if (lance.getValor() > maiorDeTodos) maiorDeTodos = lance.getValor();
13             //inserindo a condição para encontrar o menor valor
14             else if (lance.getValor() < menorDeTodos) menorDeTodos = lance.getValor();
15
16         }
17
18     }
19
20     //vamos criar um get
21     public double getMaiorLance() { //vamos alterar o nome para getMaiorLance
22         return maiorDeTodos;
23     }
24 }
25 }
```

# Vamos criar outro field para o atributo menorDeTodos

```
br.com.senac.leilao.servico;  
  
br.com.senac.leilao.dominio.Lance;  
br.com.senac.leilao.dominio.Leilao;  
  
class Avaliador {  
    private double maiorDeTodos = Double.NEGATIVE_INFINITY; //constante q guarda o menor numero q cabe dentro de um double  
  
    public void avalia (Leilao leilao) {  
  
        for (Lance lance : leilao.getLances()) {  
            if (lance.getValor() > maiorDeTodos) maiorDeTodos = lance.getValor();  
            //inserindo a condição para encontrar o menor valor  
            else if (lance.getValor() < menorDeTodos) menorDeTodos = lance.getValor();  
        }  
  
        //vamos criar um get  
        public double getMaiorLance() { //vamos alterar o nome  
            return maiorDeTodos;  
        }  
    }  
}
```

- ① Create local variable 'menorDeTodos'
- ② Create field 'menorDeTodos' (selected)
- ③ Change to 'maiorDeTodos'
- ④ Create parameter 'menorDeTodos'
- ⓧ Remove assignment
- ⑤ Rename in file (Ctrl+2, R)

```
...  
for (Lance lance : leilao.getLances()) {  
    double menorDeTodos;  
    if (lance.getValor() > maiorDeTodos) maiorDeTodos = lance.getValor();  
    //inserindo a condição para encontrar o menor valor  
    ...
```

Press 'Tab' from proposal table or

criado o atributo como positive para que assim todo mundo será menor q  
ele e assim será substituído na primeira vez

```
Cliente.java  ContaCorren...  Gerenciador...  Gerenciador...  IdadeNaoPerm...
1 package br.com.senac.leilao.servico;
2
3 import br.com.senac.leilao.dominio.Lance;
4 import br.com.senac.leilao.dominio.Leilao; mundo
5
6 public class Avaliador {
7     private double maiorDeTodos= Double.NEGATIVE_INFINITY; //constante q guarda o menor nu
8     private double menorDeTodos = Double.POSITIVE_INFINITY; //criado o atributo como positiv
9
10    public void avalia (Leilao leilao) {
11
12        for (Lance lance : leilao.getLances()) {
13            if (lance.getValor( ) > maiorDeTodos) maiorDeTodos = lance.getValor();
14            //inserindo a condição para encontrar o menor valor
15            else if (lance.getValor() < menorDeTodos) menorDeTodos = lance.getValor();
16
17        }
18
19    }
20
21    //vamos criar um get
22    public double getMaiorLance() { //vamos alterar o nome para getMaiorLance
23        return maiorDeTodos;
24    }
25}
26
```

# Vamos criar outro método get para o Menor Lance

The screenshot shows an IDE interface with several tabs at the top: Cliente.java, ContaCorren..., Lance.java, \*Avaliador.java (selected), and TesteAvalia... . The code editor displays Java code for the Avaliador class. A code completion tooltip is open over the line:

```
1 package br.com.senac.leilao.servico;
2
3 import br.com.senac.leilao.dominio.Lance;
4 import br.com.senac.leilao.dominio.Leilao; mundo
5
6 public class Avaliador {
7     private double maiorDeTodos= Double.NEGATIVE_INFINITY; //constante q guarda o menor .
8     private double menorDeTodos = Double.POSITIVE_INFINITY;//criado o atributo como posi
9
10    public void getme() : void - Method stub
11    public double getMenorDeTodos() : double - Getter for 'menorDeTodos'
12    for
13    if
14    //i
15    else
16
17    }
18
19}
20
21//vamo
22public
23    ret
24}
25
26getmel
27}
```

The tooltip lists two suggestions: "getme() : void - Method stub" (red square icon) and "getMenorDeTodos() : double - Getter for 'menorDeTodos'" (green circle icon). Below the suggestions, there is a message: "Not what you're looking for? Discover new extensions to Code Completion". At the bottom of the tooltip, it says "Press 'Ctrl+Space' to show Template Proposals".

```
Cliente.java ContaCorren... Lance.java *Avaliador.java »4  
TesteAvalia...  
4 import br.com.senac.leilao.dominio.Leilao;mundo  
5  
6 public class Avaliador {  
7     private double maiorDeTodos= Double.NEGATIVE_INFINITY; //constante q guarda o menor valor  
8     private double menorDeTodos = Double.POSITIVE_INFINITY;//criado o atributo como publico  
9  
10    public void avalia (Leilao leilao) {  
11  
12        for (Lance lance : leilao.getLances()) {  
13            if (lance.getValor( ) > maiorDeTodos) maiorDeTodos = lance.getValor();  
14            //inserindo a condição para encontrar o menor valor  
15            else if (lance.getValor() < menorDeTodos) menorDeTodos = lance.getValor();  
16  
17        }  
18  
19    }  
20  
21    //vamos criar um get  
22    public double getMaiorLance() { //vamos alterar o nome para getMaiorLance  
23        return maiorDeTodos;  
24    }  
25  
26    public double getMenorLance() {  
27        return menorDeTodos;  
28    }  
29}  
30
```

```
1 package br.com.leilao.servico;
2
3④ import br.com.leilao.dominio.Lance;
4
5
6 public class Avaliador {
7
8     //esse metodo recebe um leilao e acalisse leilao
9     //precisa ser importada para ctrl +1 import
10    private double maiorDeTodos = Double.NEGATIVE_INFINITY; //constante que guarda menor numero que
11    private double menorDeTodos = Double.POSITIVE_INFINITY;
12
13④    public void avalia (Leilao leilao) {
14        for (Lance lance : leilao.getLances()) {
15            //ctrl+1 crate field
16            if (lance.getValor() > maiorDeTodos)
17                maiorDeTodos = lance.getValor();
18            //else
19            if (lance.getValor() < menorDeTodos )
20                menorDeTodos = lance.getValor();
21
22        }
23    }
24    //getma +ctrl espaco
25④    public double getMaiorLance() {
26        return maiorDeTodos;
27    }
28④    public double getMenorLance() {
29        return menorDeTodos;
30    }
31}
32
```

```

1 package br.com.senac.leilao.teste;
2
3 import br.com.senac.leilao.dominio.Lance;
4 import br.com.senac.leilao.dominio.Leilao;
5 import br.com.senac.leilao.dominio.Usuario;
6 import br.com.senac.leilao.servico.Avaliador;
7
8 public class TesteAvaliador {
9 public static void main(String[] args) {
10
11     Usuario joao = new Usuario("Joao");
12     Usuario Jose = new Usuario("Jose");
13     Usuario Maria = new Usuario("Maria");
14
15     //vamos criar um leilao e propor lances nesse leilao
16     Leilao leilao = new Leilao("Playstation 3 novo");
17
18     leilao.propoe(new Lance(joao, 300.0));
19     leilao.propoe(new Lance(Jose, 400.0));
20     leilao.propoe(new Lance(Maria, 250.0));
21
22     //leiloeiro ira avaliar o nosso valor no leilao
23     Avaliador leiloeiro = new Avaliador();
24     leiloeiro.avalia(leilao);
25
26     System.out.println(leiloeiro.getMaiorLance()); //com isso o resultado deve ser 400
27     System.out.println(leiloeiro.getMenorLance());
28 }
29 }
```

*Em nossa classe principal , vamos imprimir o maior e o menor valor.  
Nosso resultado deverá ser ; 400 e 250 .  
Pronto tudo funcionando corretamente!*

Problems @ Javadoc Declaration Search Console

terminated> TesteAvaliador [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (12 de out de 2018 22:10:16)

400.0  
250.0

# Vamos colocar nosso software em produção?

**Será que funciona?**

# Será que funciona mesmo?

*Por via das dúvidas vamos realizar mais 1 teste !*



# Vamos ver o que acontece se alteramos os valores

```
4 import br.com.senac.leilao.dominio.Leilao;
5 import br.com.senac.leilao.dominio.Usuario;
6 import br.com.senac.leilao.servico.Avaliador;
7
8 public class TesteAvaliador {
9     public static void main(String[] args) {
10
11     Usuario joao = new Usuario("Joao");
12     Usuario Jose = new Usuario("Jose");
13     Usuario Maria = new Usuario("Maria");
14
15     //vamos criar um leilao e propor lances nesse leilao
16     Leilao leilao = new Leilao("Playstation 3 novo");
17
18     leilao.propoe(new Lance(joao, 250.0));
19     leilao.propoe(new Lance(Jose, 300.0));
20     leilao.propoe(new Lance(Maria, 400.0)); ←
21
22     //leiloeiro ira avaliar o nosso valor no leilao
23     Avaliador leiloeiro = new Avaliador();
24     leiloeiro.avalia(leilao);
25
26     System.out.println(leiloeiro.getMaiorLance()); //com isso o resultado deve ser 400
27     System.out.println(leiloeiro.getMenorLance());
28 }
29 }
```

Problems @ Javadoc Declaration Search Console X

<terminated> TesteAvaliador [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (12 de out de 2018 22:35:50)

400.0

Infinity ←

*Veja! Ele trouxe 400 e Infinity...  
Ou seja , um Bug!!! Quase colocamos um código com  
bug em produção ...  
E isso aconteceu por que ?*



*Testes manuais são caros, chatos e lentos de serem executados*



*A solução é não usar humanos para testar e sim ensinar uma máquina para testar . Ou seja escrever um programa que teste nossos códigos!*



# Testes manuais:

- Pensar em um cenário
- Executar uma ação
- Validar a saída

Onde :

Cenário = o que acontece com um formulário de cadastro se eu não passar o cpf.

Ação = Clicar no botão Salvar

Saída = Como o programa se comportou

# Testes Automatizados tem os mesmos três passos:

```
public class TesteDoAvaliador {  
  
    public static void main(String[] args) {  
        // parte 1: cenário  
        Usuario joao = new Usuario("João");  
        Usuario jose = new Usuario("José");  
        Usuario maria = new Usuario("Maria");  
  
        Leilao leilao = new Leilao("Playstation 3 Novo");  
  
        leilao.propoe(new Lance(joao, 250.0));  
        leilao.propoe(new Lance(jose, 300.0));  
        leilao.propoe(new Lance(maria, 400.0));  
  
        // parte 2: ação  
        Avaliador leiloeiro = new Avaliador();  
        leiloeiro.avalia(leilao);  
  
        // parte 3: validação  
        System.out.println(leiloeiro.getMaiorLance());  
        System.out.println(leiloeiro.getMenorLance());  
    }  
}
```

```
public static void main(String[] args) {
    // parte 1: cenario
    Usuario joao = new Usuario("João");
    Usuario jose = new Usuario("José");
    Usuario maria = new Usuario("Maria");

    Leilao leilao = new Leilao("Playstation 3 Novo");

    leilao.propoe(new Lance(joao, 250.0));
    leilao.propoe(new Lance(jose, 300.0));
    leilao.propoe(new Lance(maria, 400.0));
    // parte 2: AGAO
    Avaliador leiloeiro = new Avaliador();
    leiloeiro.avalia(leilao);

    // parte 3: validacao
    System.out.println(leiloeiro.getMaiorLance());
```

Ainda é um pouco manual!

*Veja : A saída ainda é manual*

```
// parte 3: validacao
System.out.println(leiloeiro.getMaiorLance());
System.out.println(leiloeiro.getMenorLance());
}
}
```

*Vamos criar dois atributos de teste:*

***double maiorEsperado = 400;***

***double menorEsperado = 250;***

```
public static void main(String[] args) {
    // parte 1: cenário
    Usuario joao = new Usuario("João");
    Usuario jose = new Usuario("José");
    Usuario maria = new Usuario("Maria");

    Leilao leilao = new Leilao("Playstation 3 Novo");

    leilao.propoe(new Lance(joao, 250.0));
    leilao.propoe(new Lance(jose, 300.0));
    leilao.propoe(new Lance(maria, 400.0));

    // parte 2: ação
    Avaliador leiloeiro = new Avaliador();
    leiloeiro.avalia(leilao);

    // parte 3: validação
    double maiorEsperado = 400;
    double menorEsperado = 250;

    System.out.println(maiorEsperado == leiloeiro.getMaiorLance());
    System.out.println(menorEsperado == leiloeiro.getMenorLance());
```

```
ublic class TesteDoAvaliador {  
  
    public static void main(String[] args) {  
        // parte 1: cenário  
        Usuario joao = new Usuario("João");  
        Usuario jose = new Usuario("José");  
        Usuario maria = new Usuario("Maria");  
  
        Leilao leilao = new Leilao("Playstation 3 Novo");  
  
        leilao.propoe(new Lance(joao, 250.0));  
        leilao.propoe(new Lance(jose, 300.0));  
        leilao.propoe(new Lance(maria, 400.0));  
  
        // parte 2: Ação  
        Avaliador leiloeiro = new Avaliador();  
        leiloeiro.avalia(leilao);  
  
        // parte 3: validação
```

O computador valida a saída!

```
15     //vamos criar um leilao e propor lances nesse leilao
16     Leilao leilao = new Leilao("Playstation 3 novo");
17
18     leilao.propoe(new Lance(joao, 250.0));
19     leilao.propoe(new Lance(Jose, 300.0));
20     leilao.propoe(new Lance(Maria, 400.0));
21
22     //leiloeiro ira avaliar o nosso valor no leilao
23     //parte2: acao
24     Avaliador leiloeiro = new Avaliador();
25     leiloeiro.avalia(leilao);
26
27     //parte3:validacao|
28     double maiorEsperado = 400;
29     double menorEsperado = 250;
30     System.out.println(maiorEsperado == leiloeiro.getMaiorLance()); //com isso c
31     System.out.println(menorEsperado == leiloeiro.getMenorLance());
32 }
```

Problems @ Javadoc Declaration Search Console

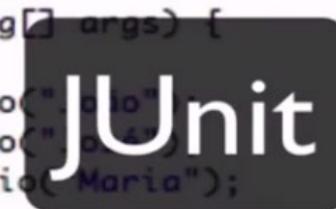
<terminated> TesteAvaliador [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (12 de out de 2018)

true  
false

Writable Smart Insert 27 · 22

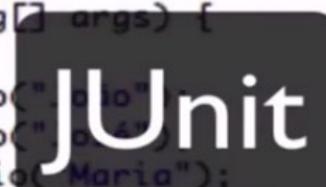
*Para automatizar nosso teste vamos , utilizar o framewok - Junit*

```
public static void main(String[] args) {  
    // parte 1: cenario  
    Usuario joao = new Usuario("joao");  
    Usuario jose = new Usuario("jose");  
    Usuario maria = new Usuario("Maria");  
  
    Leilao leilao = new Leilao("Playstation 3 Novo");  
  
    leilao.propoe(new Lance(joao, 250.0));  
    leilao.propoe(new Lance(jose, 300.0));  
    ...  
}
```

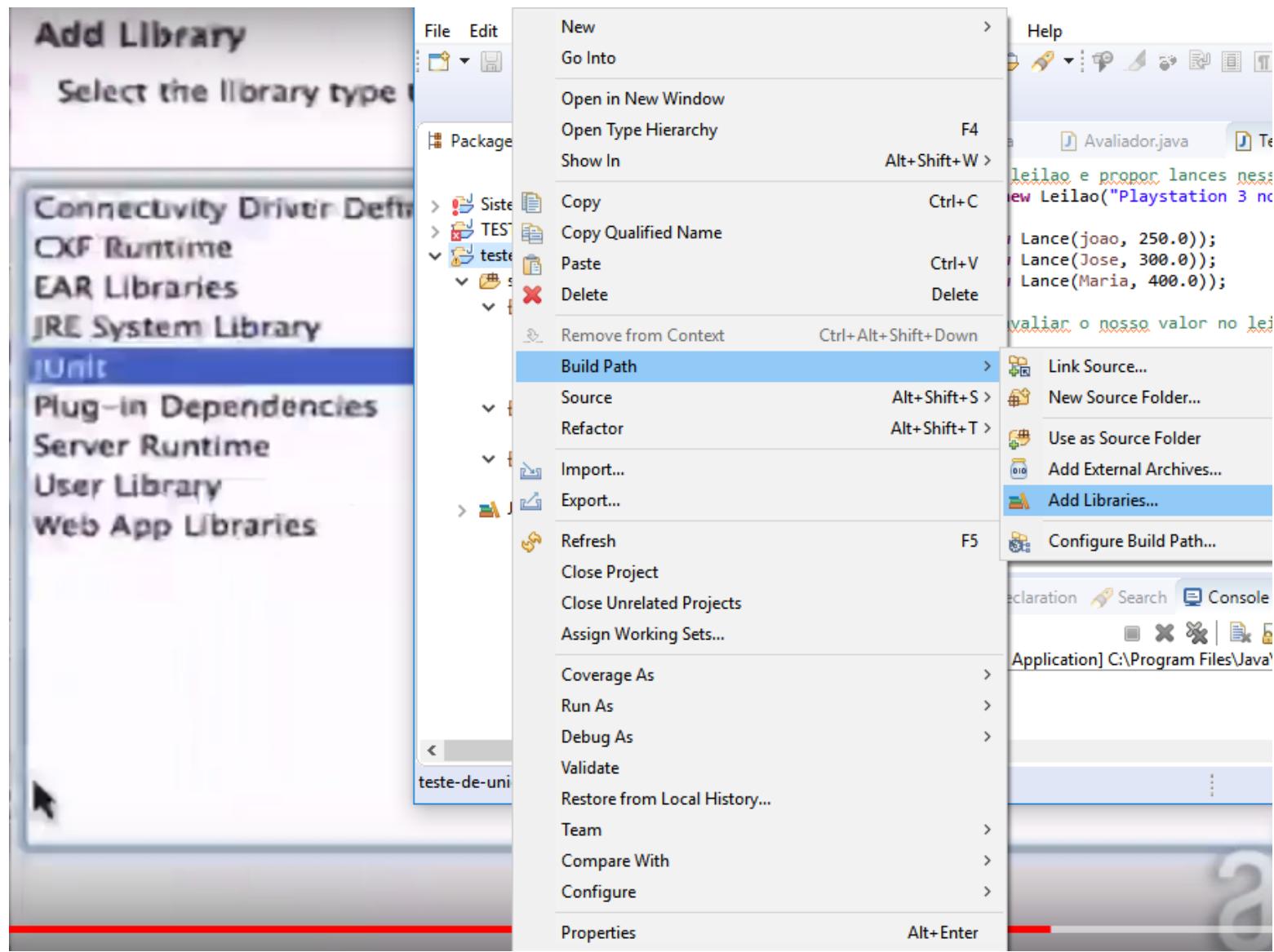


*Para automatizar nosso teste vamos , utilizar o framewok - Junit*

```
public static void main(String[] args) {  
    // parte 1: cenario  
    Usuario joao = new Usuario("joao");  
    Usuario jose = new Usuario("jose");  
    Usuario maria = new Usuario("Maria");  
  
    Leilao leilao = new Leilao("Playstation 3 Novo");  
  
    leilao.propoe(new Lance(joao, 250.0));  
    leilao.propoe(new Lance(jose, 300.0));  
}
```



# Botão direito no nosso projeto, Build Path





## Add Library

Select the library type to add.



JRE System Library

JUnit

Maven Managed Dependencies

User Library

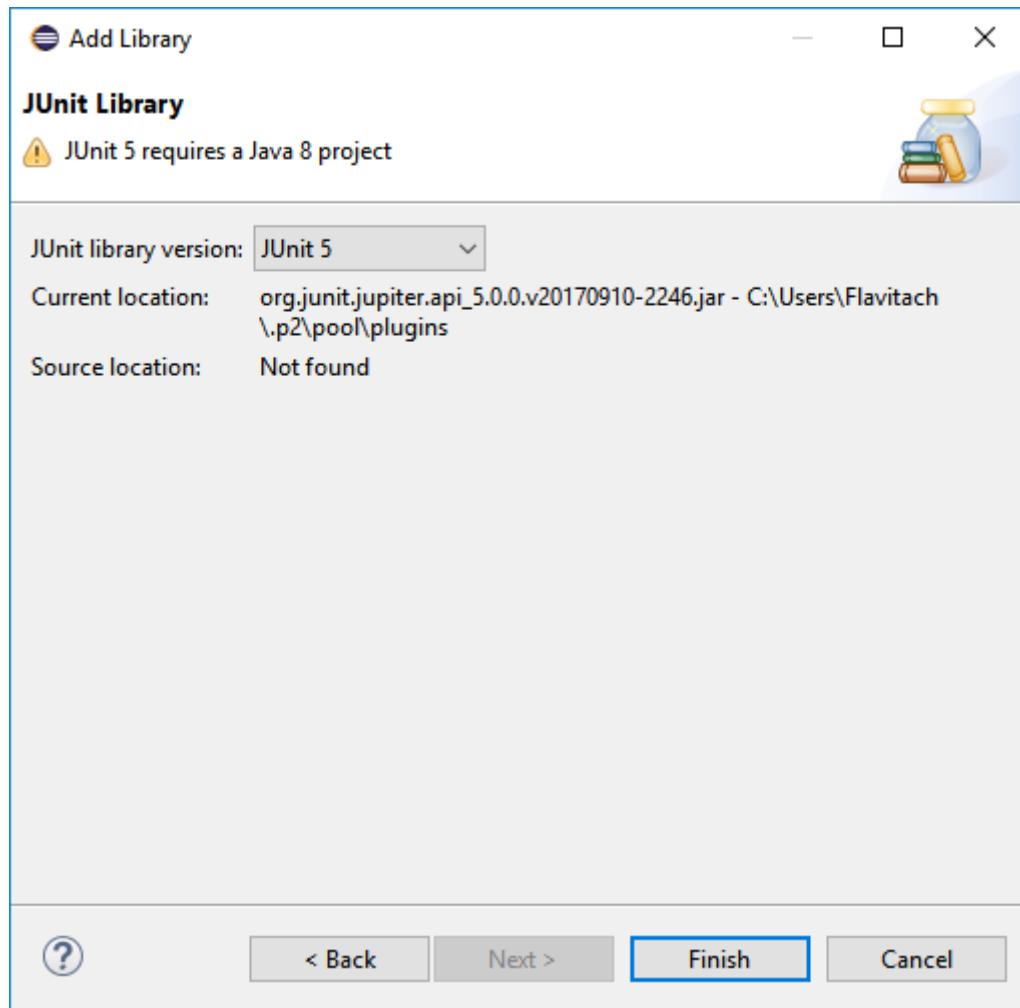


< Back

Next >

Finish

Cancel



The screenshot shows the Eclipse IDE interface with the following details:

- Toolbar:** Standard Eclipse toolbar with icons for file operations, search, and navigation.
- Project Explorer:** Shows files: Cliente.java, ContaCorren..., Gerenciador..., Gerenciador..., and Lance.java.
- Code Editor:** Displays the Java code for `TesteAvaliador.java`. The code is annotated with handwritten notes in red ink:
  - Line 9: ~~public static void main(String[] args)~~
  - Line 10: ~~//parte1: cenário~~
  - Line 13: ~~Usuario Maria = new Usuario("Maria");~~
  - Line 16: ~~Leilao leilao = new Leilao("Playstation 3 novo");~~
  - Line 22: ~~//leiloeiro irá avaliar o nosso valor no leilao~~
  - Line 23: ~~//parte2: ação~~
  - Line 25: ~~Avaliador leiloeiro = new Avaliador();~~
  - Line 26: ~~leiloeiro.avalia(leilao);~~
  - Line 27: ~~//parte3: validação~~
  - Line 28: ~~double maiorEsperado = 400;~~
  - Line 29: ~~double menorEsperado = 250;~~
- Bottom Bar:** Problems, Javadoc, Declaration, Search, Console.

Para o JUnit, entender o nosso método de teste, precisamos mudar a assinatura:

Ex: O método não pode ser estático e nem receber argumentos .

Passando ser : Public void main()

Obrigatoriamente não recebe nada .

```
1 package br.com.senac.leilao.teste;
2
3 import org.junit.Test;
4
5 import br.com.senac.leilao.dominio.Lance;
6 import br.com.senac.leilao.dominio.Leilao;
7 import br.com.senac.leilao.dominio.Usuario;
8 import br.com.senac.leilao.servico.Avaliador;
9
10 public class TesteAvaliador {
11
12 @Test
13 public void main() {
14     //parte1: cenário
15     Usuario joao = new Usuario("Joao");
16     Usuario Jose = new Usuario("Jose");
17     Usuario Maria = new Usuario("Maria");
18
19     //vamos criar um leilão e propor lances nesse leilão
20     Leilao leilao = new Leilao("Playstation 3 novo");
21
22     leilao.propoe(new Lance(joao, 250.0));
23     leilao.propoe(new Lance(Jose, 300.0));
24     leilao.propoe(new Lance(Maria, 400.0));
25
26     //leiloeiro irá avaliar o nosso valor no leilão
27     //parte2: ação
28     Avaliador leiloeiro = new Avaliador();
29     leiloeiro.avalia(leilao);
30 }
```

*Passa a receber a anotação @Test de  
import.org.Junit*

*Para finalizar na classe de Validação, usaremos a classe Assert, que irá nos ajudar a conferir o resultado calculado.*

*Vamos utilizar o `AssertEquals` com 2 parâmetros onde (1 irá calcular o valor Esperado e outro o valor Calculado) . Usaremos também um `DELTA` , com máscara `0.00001`, visto que o `double` sempre resulta em problemas de arredondamento.*

# Assertivas

- São métodos auxiliares utilizados para verificar se uma dada condição é verdadeira. Em caso negativo, geram um erro indicando que o teste falhou.
- Devem ser importadas para que possam ser utilizadas.

Assertiva/método	Descrição
assertEquals(int esperado, int atual) assertEquals(long esperado, long atual)	Verifica se <b>esperado</b> e <b>atual</b> são valores iguais.
assertEquals(double esperado, double atual, double delta)	Verifica se a diferença entre <b>esperado</b> e <b>atual</b> está dentro de <b>delta</b> .
assertEquals(Object esperado, Object atual)	Verifica se <b>esperado</b> e <b>atual</b> são objetos iguais segundo o método <b>equals</b> .
assertFalse(boolean condição)	Verifica se <b>condição</b> é falsa.
assertTrue(boolean condição)	Verifica se <b>condição</b> é verdadeira.

Para executar: Botão direito / Run as /Junit

The screenshot shows an IDE interface with two tabs: 'Avaliador.java' and '\*TesteAvaliador.java'. The code in 'Avaliador.java' is as follows:

```
16 Usuario joao = new Usuario("Joao");
17 Usuario Jose = new Usuario("Jose");
18 Usuario Maria = new Usuario("Maria");
19
20 //vamos criar um leilao e propor lances nesse leilao
21 Leilao leilao = new Leilao("Playstation 3 novo");
22
23 leilao.propoe(new Lance(joao, 250.0));
24 leilao.propoe(new Lance(Jose, 300.0));
25 leilao.propoe(new Lance(Maria, 400.0));
26
27 //leiloeiro ira avaliar o nosso valor no leilao
28 //parte2: acao
29 Avaliador leiloeiro = new Avaliador();
30 leiloeiro.avalia(leilao);
31
32 //parte3:validacao
33 double maiorEsperado = 400;
34 double menorEsperado
35 Assert.assertEquals(expected, actual, delta);
36
37 //System.out.println(maiorEsperado);
38 //System.out.println(menorEsperado);
39 }
```

A tooltip is displayed over the line 'Assert.assertEquals(expected, actual, delta);' with the following content:

expected
<input type="radio"/> menorEsperado
<input checked="" type="radio"/> maiorEsperado
0

The status bar at the bottom shows: 'terminated> testes [JUnit] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (12 de out de 2018 23:18:21)'.



Avaliador.java TesteAvaliador.java Assert.class

```

10
11 public class TesteAvaliador {
12
13 @SuppressWarnings("deprecation")
14 @Test
15 public void main() {
16     //parte1: cenario
17     Usuario joao = new Usuario("Joao");
18     Usuario Jose = new Usuario("Jose");
19     Usuario Maria = new Usuario("Maria");
20
21     //vamos criar um leilao e propor lances nesse leilao
22     Leilao leilao = new Leilao("Playstation 3 novo");
23
24     leilao.propoe(new Lance(joao, 250.0));
25     leilao.propoe(new Lance(Jose, 300.0));
26     leilao.propoe(new Lance(Maria, 400.0));
27
28     //leiloeiro ira avaliar o nosso valor no leilao
29     //parte2:acao
30     Avaliador leiloeiro = new Avaliador();
31     leiloeiro.avalia(leilao);
32
33     //parte3:validacao

```

Problems @ Javadoc Declaration Search Console

terminated> TesteAvaliador (1) [JUnit] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (12 de out)

JU 1 JUnit Test

Alt+Shift+E, T

Coverage Configurations...

- undo Add @SuppressWarnings to 'main()' Ctrl+Z
- Revert File
- Save Ctrl+S
- Open Declaration F3
- Open Type Hierarchy F4
- Open Call Hierarchy Ctrl+Alt+H
- Show in Breadcrumb Alt+Shift+B
- Quick Outline Ctrl+O
- Quick Type Hierarchy Ctrl+T
- Open With >
- Show In Alt+Shift+W >
- Cut Ctrl+X
- Copy Ctrl+C
- Copy Qualified Name
- Paste Ctrl+V
- Quick Fix Ctrl+1
- Source Alt+Shift+S >
- Refactor Alt+Shift+T >
- Surround With Alt+Shift+Z >
- Local History >
- References >
- Declarations >
- Add to Snippets...
- Coverage As >
- Run As >
- Debug As >
- Validate
- Create Snippet... >
- Team >
- Compare With >

Note que , exibiu uma faixa vermelha ! Ou seja temos erros em nosso código

The screenshot shows the Eclipse IDE interface with the JUnit perspective active. The top bar includes tabs for 'Package Explorer', 'JUnit' (which is selected), and 'Assert.class'. The JUnit view displays the following information:

- Finished after 0,077 seconds
- Runs: 1/1
- Errors: 0
- Failures: 1

The failure trace shows a single assertion error:

```
junit.framework.AssertionFailedError: expected:<250.0> but was:<Infinity>
at br.com.senac.leilao.teste.TesteAvaliador.main(TesteAvaliador.java:37)
```

A red arrow points from the bottom left towards the failure trace area.

The code editor displays two files: 'Avaliador.java' and 'TesteAvaliador.java'. The 'TesteAvaliador.java' file contains the failing test code:10
11 public class TesteAvaliador {
12
13 @SuppressWarnings("deprecation")
14 @Test
15 public void main() {
16 //parte1: cenário
17 Usuario joao = new Usuario("Joao");
18 Usuario Jose = new Usuario("Jose");
19 Usuario Maria = new Usuario("Maria");
20
21 //vamos criar um leilao e propor lances nesse leilao
22 Leilao leilao = new Leilao("Playstation 3 novo");
23
24 leilao.propoe(new Lance(joao, 250.0));
25 leilao.propoe(new Lance(Jose, 300.0));
26 leilao.propoe(new Lance(Maria, 400.0));
27
28 //leiloeiro irá avaliar o nosso valor no leilao
29 //parte2: ação
30 Avaliador leiloeiro = new Avaliador();
31 leiloeiro.avalia(leilao);
32
33 //parte3: validação

The 'Avaliador.java' file is partially visible at the top of the code editor.

The bottom status bar shows the terminal output:

```
<terminated> TesteAvaliador (1) [JUnit] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (12 de
```

O erro está no else , que está soprando em nosso código

The screenshot shows the Eclipse IDE interface with the following details:

- Top Bar:** Package Explorer, JUnit (with a red error icon), and other standard icons.
- Left Sidebar:** Shows the status "finished after 0,077 seconds", "Runs: 1/1", "Errors: 0", "Failures: 1", and the test class "br.com.senac.leilao.teste.TesteAvaliador [Runner]" with a duration of "main (0,025 s)".
- Failure Trace:** Displays the stack trace:

```
junit.framework.AssertionFailedError: expected:<250>
at br.com.senac.leilao.teste.TesteAvaliador.main(Te
```
- Code Editor (Avaliador.java):**

```
1 package br.com.senac.leilao.servico;
2
3 import br.com.senac.leilao.dominio.Lance;
4
5 public class Avaliador {
6     private double maiorDeTodos= Double.NEGATIVE_INFINITY; //constante q guarda o menor numero
7     private double menorDeTodos = Double.POSITIVE_INFINITY;//criado o atributo como positive
8
9
10    public void avalia (Leilao leilao) {
11
12        for (Lance lance : leilao.getLances()) {
13            if (lance.getValor() > maiorDeTodos) maiorDeTodos = lance.getValor();
14            //inserindo a condição para encontrar o menor valor
15            else if (lance.getValor() < menorDeTodos) menorDeTodos = lance.getValor();
16
17        }
18
19    }
20
21    //vamos criar um get
22    public double getMaiorLance() { //vamos alterar o nome para getMaiorLance
23        return maiorDeTodos;
24    }
25}
```
- Bottom Bar:** Problems, Javadoc, Declaration, Search, Console (with a red error icon), and other standard icons.
- Console:** Shows the message "<terminated> TesteAvaliador (1) [JUnit] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (12 de out de 2018 23:55:03)".

## Corrigindo:

```
Avaliador.java X TesteAvaliador.java Assert.class
1 package br.com.senac.leilao.servico;
2
3+import br.com.senac.leilao.dominio.Lance;+
4
5
6 public class Avaliador {
7     private double maiorDeTodos= Double.NEGATIVE_INFINITY; //constante q guarda o menor nu
8     private double menorDeTodos = Double.POSITIVE_INFINITY;//criado o atributo como positi
9
10+    public void avalia (Leilao leilao) {
11
12        for (Lance lance : leilao.getLances()) {
13            if (lance.getValor( ) > maiorDeTodos) maiorDeTodos = lance.getValor();
14            if (lance.getValor( ) < menorDeTodos) menorDeTodos = lance.getValor();
15            //inserindo a condição para encontrar o menor valor
16            //else if (lance.getValor() < menorDeTodos) menorDeTodos = lance.getValor();
17
18        }
19
20    }
21
22    //vamos criar um get
23+    public double getMaiorLance() { //vamos alterar o nome para getMaiorLance
24        return maiorDeTodos;
25    }
```

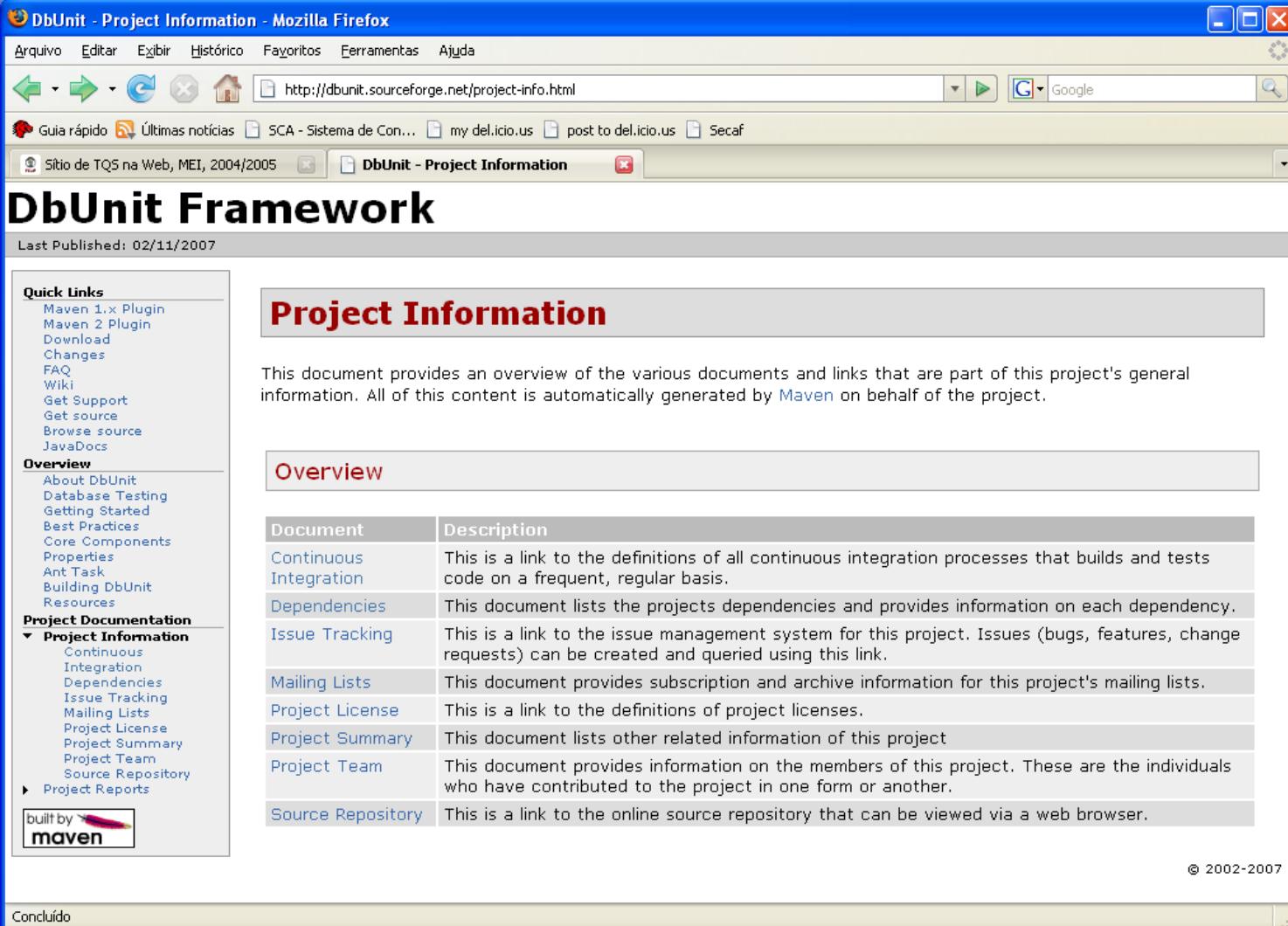
A screenshot of the Eclipse IDE interface. At the top, there's a toolbar with various icons. A red arrow points from the top center towards the toolbar. Below the toolbar is the 'Package Explorer' view, which shows a green progress bar indicating the test was finished in 0.027 seconds. The 'JUnit' tab is selected, showing 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. In the center, the code editor displays Java code for a test class named 'TesteAvaliador.java'. The code includes imports for 'Avaliador.java', 'Assert.class', and 'br.com.senac.leilao.dominio.Usuario'. It contains several test methods and assertions. The bottom part of the interface shows the 'Failure Trace' view and the 'Console' view, which displays the termination message for the JUnit test.

```
13 @SuppressWarnings("deprecation")
14 @Test
15 public void EntendeLance() {
16     //parte1: cenario
17     Usuario joao = new Usuario("Joao");
18     Usuario Jose = new Usuario("Jose");
19     Usuario Maria = new Usuario("Maria");
20
21     //vamos criar um leilao
22     Leilao leilao = new Leilao("Leilao de Teste");
23
24     leilao.propoe(new Lance(joao, 250.0));
25     leilao.propoe(new Lance(Jose, 300.0));
26     leilao.propoe(new Lance(Maria, 400.0));
27
28     //leiloeiro ira avaliar o nosso valor no leilao
29     //parte2:acao
30     Avaliador leiloeiro = new Avaliador();
31     leiloeiro.avalia(leilao);
32
33     //parte3:validacao
34     double maiorEsperado = 400;
35     double menorEsperado = 250;
36     Assert.assertEquals(maiorEsperado, leiloeiro.getMaiorLance(), 0.00001);
```

# Frameworks para Testes de Unidade

- Similares ao JUnit (linguagem Java):
  - *Python*
    - PyUnit
  - *C++*
    - CppUnit
  - *Perl*
    - PerlUnit
  - *.NET*
    - NUnit, NUnitForms, dotUnit, EasyMock.NET, csUnit

# Testes envolvendo acesso a Base de Dados

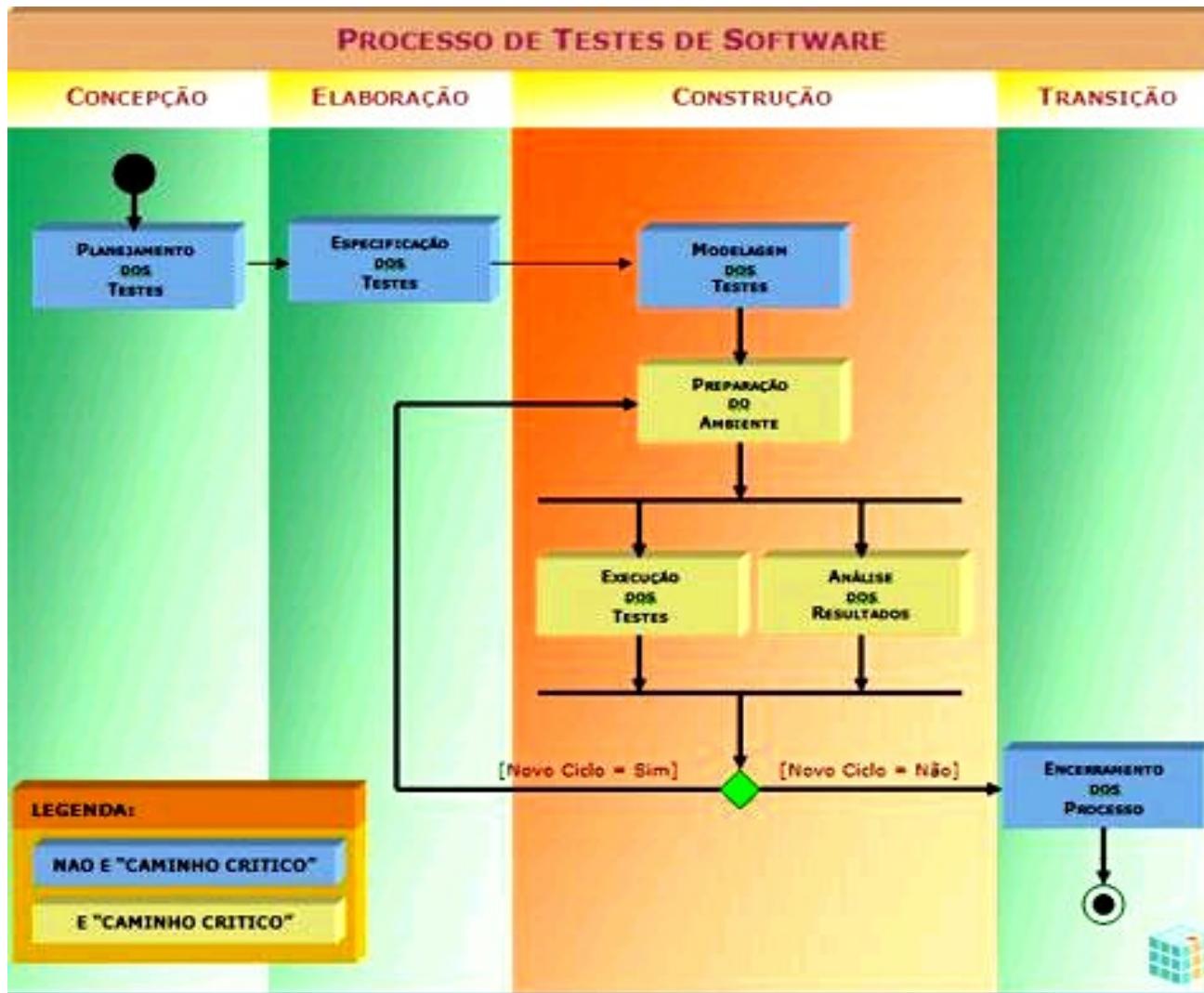


The screenshot shows a Mozilla Firefox browser window with the title "DbUnit - Project Information - Mozilla Firefox". The address bar displays the URL <http://dbunit.sourceforge.net/project-info.html>. The page content is the "Project Information" section of the DbUnit project's website. The page header includes "DbUnit Framework" and "Last Published: 02/11/2007". On the left, there is a sidebar with "Quick Links" (Maven 1.x Plugin, Maven 2 Plugin, Download, Changes, FAQ, Wiki, Get Support, Get source, Browse source, JavaDocs) and "Overview" (About DbUnit, Database Testing, Getting Started, Best Practices, Core Components, Properties, Ant Task, Building DbUnit, Resources). The main content area has a red header "Project Information" which states: "This document provides an overview of the various documents and links that are part of this project's general information. All of this content is automatically generated by Maven on behalf of the project." Below this is a "Overview" section with a table:

Document	Description
Continuous Integration	This is a link to the definitions of all continuous integration processes that builds and tests code on a frequent, regular basis.
Dependencies	This document lists the projects dependencies and provides information on each dependency.
Issue Tracking	This is a link to the issue management system for this project. Issues (bugs, features, change requests) can be created and queried using this link.
Mailing Lists	This document provides subscription and archive information for this project's mailing lists.
Project License	This is a link to the definitions of project licenses.
Project Summary	This document lists other related information of this project
Project Team	This document provides information on the members of this project. These are the individuals who have contributed to the project in one form or another.
Source Repository	This is a link to the online source repository that can be viewed via a web browser.

At the bottom of the page, there is a "built by maven" logo and a copyright notice: "© 2002-2007".

# Processo de Teste de Software na visão do RUP



# Planejamento de Testes

- Definição de uma proposta de testes baseada nas expectativas do Cliente em relação à :
  - *prazos*,
  - *custos*
  - *qualidade esperada*
- Possibilidade de dimensionar a equipe e estabelecer um esforço de acordo com as necessidades apontadas pelo Cliente.

# Especificação dos Testes

- Identificação dos casos de testes que deverão ser construídos e/ou modificados em função das mudanças solicitadas pelo Cliente.

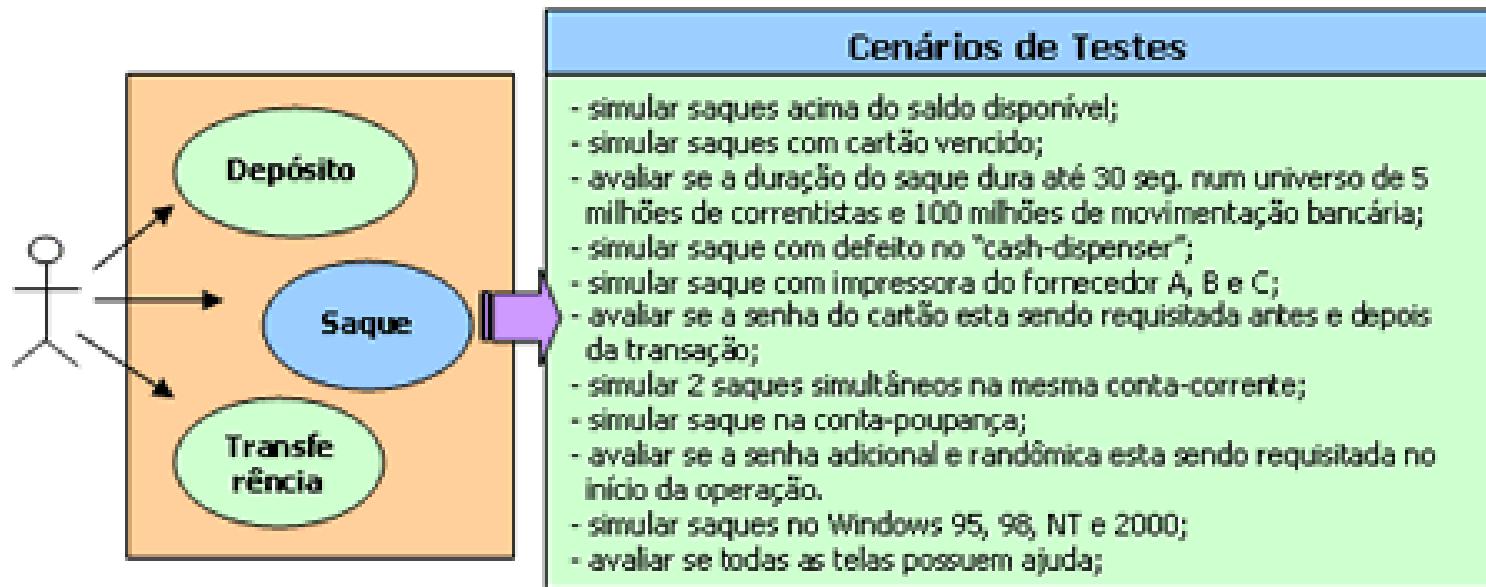


Figura - Levantamento dos Cenários sem aplicar os conceitos de Categorização

# Especificação dos Testes (Categorias)

Funcional	Segurança	Usabilidade	Performance
<ul style="list-style-type: none"><li>- simular saques acima do saldo disponível;</li><li>- simular saque na conta-poupança;</li><li>- <b>simular saque acima do valor do limite da conta;</b></li><li>- <b>simular saque com valores não múltiplos das notas;</b></li><li>- <b>simular saque com valores não múltiplos das notas;</b></li></ul>	<ul style="list-style-type: none"><li>- simular saques com cartão vencido;</li><li>- avaliar se a senha do cartão está sendo requisitada antes e depois da transação;</li><li>- avaliar se a senha adicional é randômica está sendo requisitada no início da operação;</li><li>- <b>simular saque noturno acima do valor permitido;</b></li></ul>	<ul style="list-style-type: none"><li>- avaliar se todas as telas possuem ajuda;</li><li>- <b>avaliar se mensagens são claras e objetivas;</b></li><li>- avaliar se o padrão visual é mantido em todos os momentos;</li><li>- avaliar se todas as operações possuem caminhos de fuga;</li></ul>	<ul style="list-style-type: none"><li>- avaliar se a duração do saque dura até 30 seg. num universo de 5 milhões de correntistas e 100 milhões de movimentação bancária;</li><li>- garantir que manipulação com dispositivos físicos no saque não ultrapassem 10 seg. da operação;</li></ul>
Carga e Concorrência	Configuração	Recuperação	Contingência
<ul style="list-style-type: none"><li>- simular 2 saques simultâneos na mesma conta-corrente;</li><li>- <b>simular 10.000 saques simultâneos;</b></li></ul>	<ul style="list-style-type: none"><li>- simular saque com impressora do fornecedor A, B e C;</li><li>- simular saques no Windows 95, 98, NT e 2000;</li><li>- <b>simular saque com impressora do fornecedor X, Y e Z;</b></li></ul>	<ul style="list-style-type: none"><li>- simular saque com defeito no "cash-dispenser";</li><li>- <b>simular saque com defeito na impressora;</b></li><li>- <b>simular saque com falha de conexão com a central;</b></li><li>- <b>simular saque com queda de energia;</b></li></ul>	<ul style="list-style-type: none"><li>- disparar processo de instalação emergencial;</li></ul>

# Modelagem dos Testes

- Identificação de todos os elementos necessários para a implementação de cada caso de teste especificado:
  - *modelagem das massas de testes*
  - *definição dos critérios de tratamento de arquivos (descaracterização e comparação de resultados).*

# Preparação do Ambiente

- Conjunto de atividades que visa a disponibilização física de um ambiente de testes para sofrer a bateria de testes planejadas nas etapas anteriores de forma contínua e automatizada (sem intervenção humana).

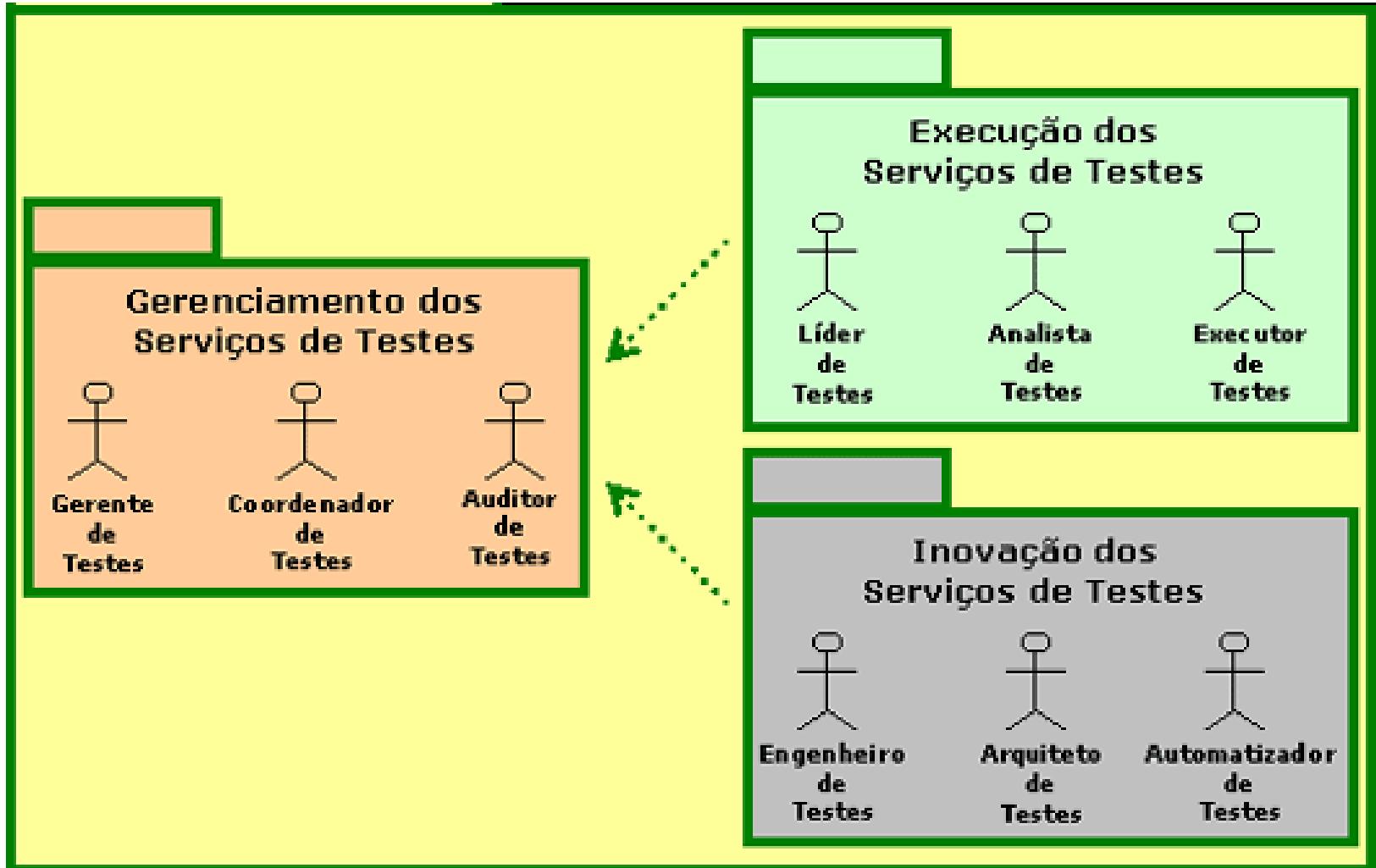
# Execução dos Testes

- Execução e conferência dos testes planejados, de forma a garantir que o comportamento do aplicativo permanece em "conformidade" com os requisitos contratados pelo Cliente.

# Análise dos Resultados

- Análise e confirmação dos resultados relatados durante a fase de execução dos testes.
- Os resultados em "não-conformidade" deverão ser "confirmados" e "detalhados" para que a Fábrica de Software realize as correções necessárias.
- Já os em "conformidade" deverão ter seu resultado "POSITIVO" reconfirmedo.

# Equipes de Teste



## Norma IEEE 829-1998

- A norma IEEE 829-1998 descreve um conjunto de documentos para as atividades de teste de um produto de software. Os documentos cobrem as tarefas de planejamento, especificação e relato de testes.

Perguntas?