

# **Teste de Software**

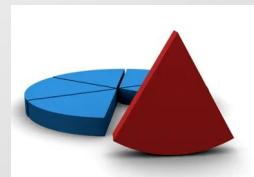
Prof. Dr. Bruno Queiroz Pinto

# Introdução

## Por que testar?

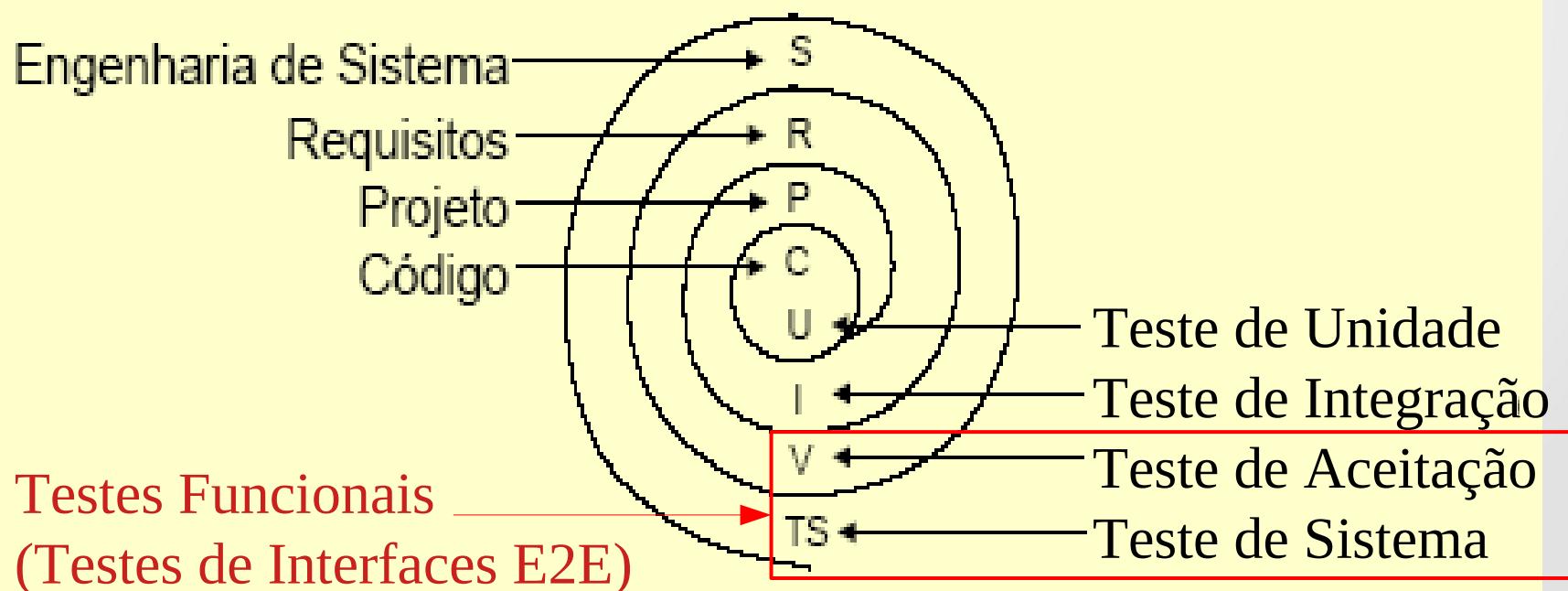
- ✓ Para garantir a qualidade do software.
  - Para **verificar** o correto funcionamento do código.
  - Para **validar** se as funcionalidades/requisitos foram implementados corretamente.

Testes lógicos - "Verificação" (construir o produto corretamente - código)



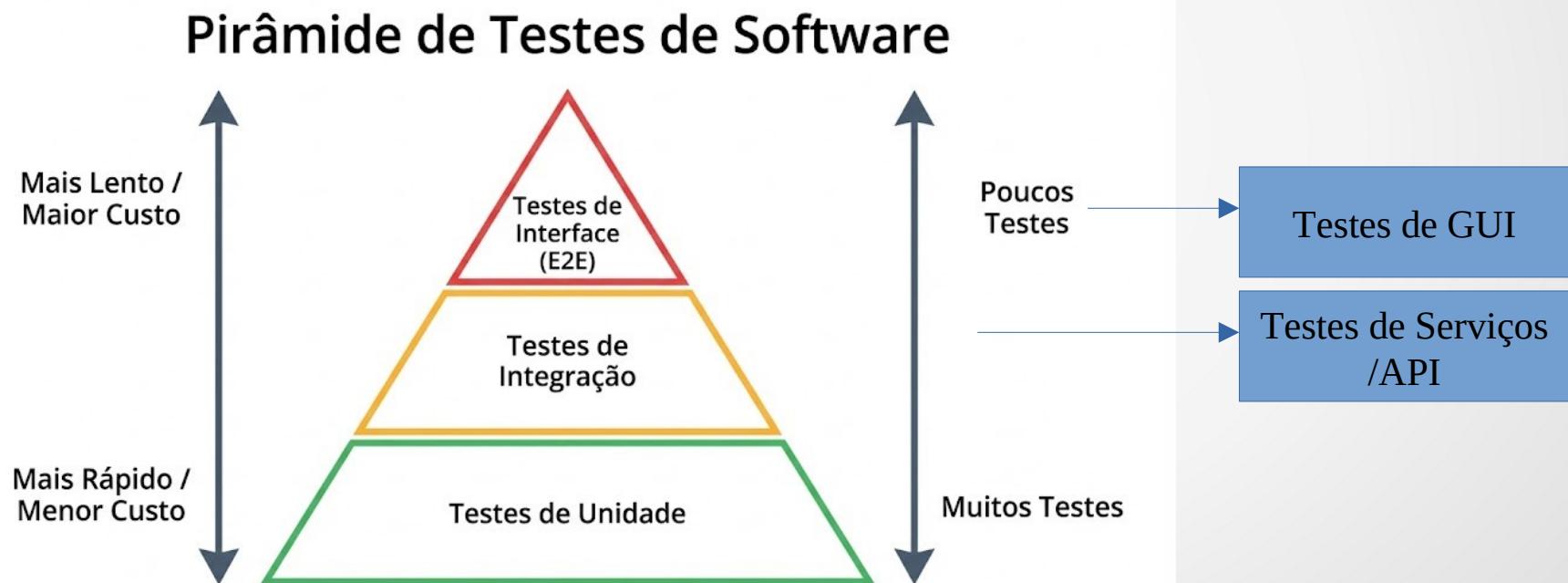
# Estratégias de Teste

Relação entre atividades de desenvolvimento e estratégias de teste.



# Estratégias de Teste

## Pirâmide de Teste de Software



# Teste de Unidade (Unitários)

Analogia com a Engenharia Elétrica.

Testar  
individualmente  
o plug macho



Testar  
individualmente  
o plug fêmea

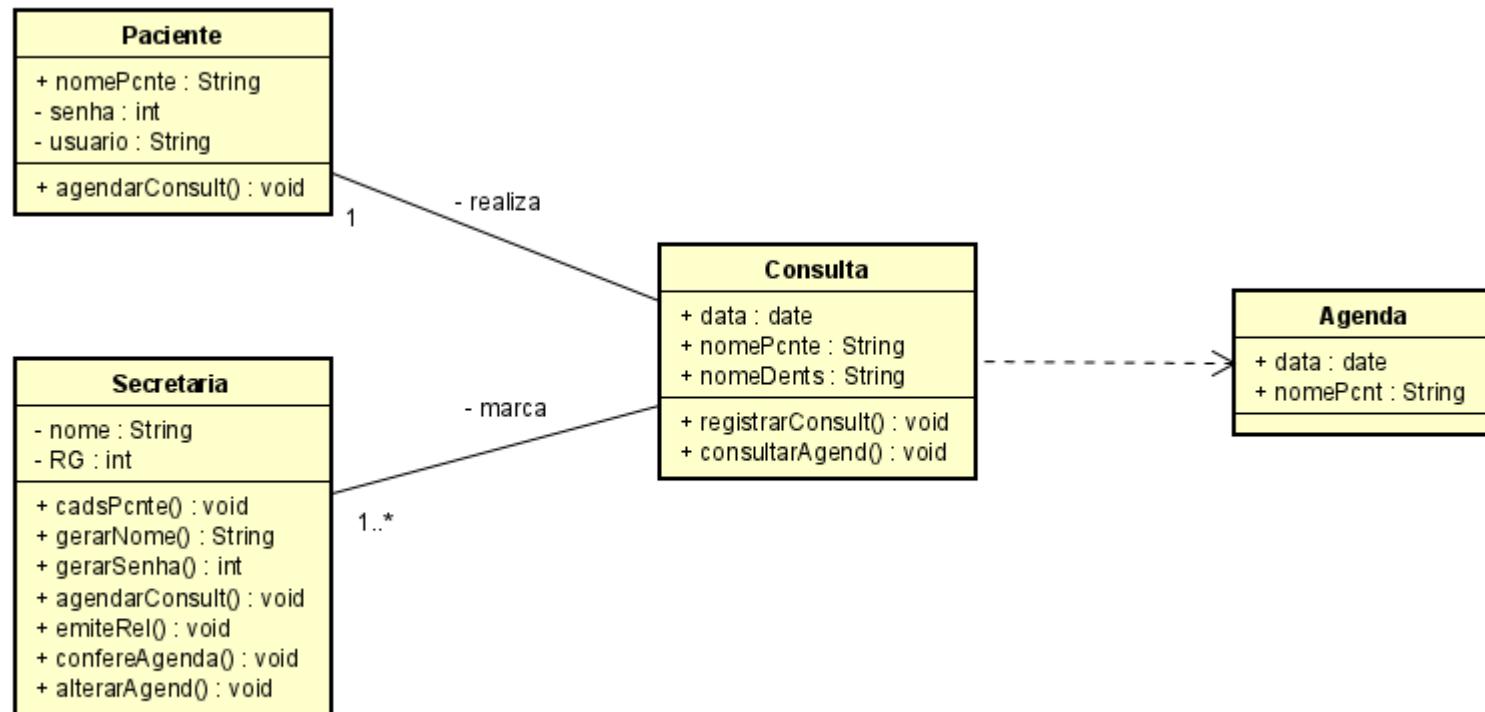


# Teste de Unidade (Unitários)



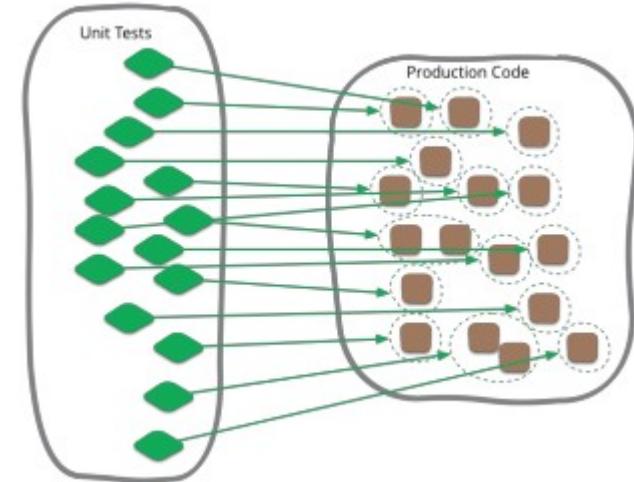
Testar cada parte da suspensão

# Teste de Unidade (Unitários)



Testar cada parte, as classes do sistema

# Teste de Unidade (Unitários)



- ✓ Concentra-se na verificação de unidades de um projeto: módulo/classes
  - ✓ pode ser realizado em paralelo para vários módulos/classes;
  - ✓ simplificando : irá validar os métodos da classe;
  - ✓ aspectos considerados:



# Exemplo de Aspectos a serem testados

```
public void sacar(double valor) {  
    // 1. Validação de Entrada  
    if (valor <= 0) {  
        throw new IllegalArgumentException("Valor inválido");  
    }  
  
    // 2. Lógica de Negócio (Fluxo)  
    if (valor > this.saldo) {  
        throw new SaldoInsuficienteException("Saldo insuficiente");  
    }  
  
    // 3. Mudança de Estado  
    this.saldo = this.saldo - valor;  
}
```

## Interfaces/Entradas:

O método recebe o parâmetro correto?  
Teste: Tentar passar `null` ou tipos errados (se a linguagem permitir).

## Condições de Limite (Boundary):

O que acontece nos extremos?  
Teste: Sacar 0.01 (mínimo), sacar exatamente o valor do saldo (saldo zero), sacar valor negativo.

## Caminhos Independentes (Fluxo):

Garantir que o teste passe por todos os `ifs` e `elses`.  
Teste A: Caminho do Sucesso (Saldo suficiente).  
Teste B: Caminho da Falha (Saldo insuficiente).

## Tratamento de Erros:

A exceção correta é lançada quando algo dá errado?  
Teste: Verificar se lança `SaldoInsuficienteException` e não um erro genérico.

## Estruturas de Dados (Estado):

A variável saldo foi atualizada corretamente no final?

# Teste de Unidade

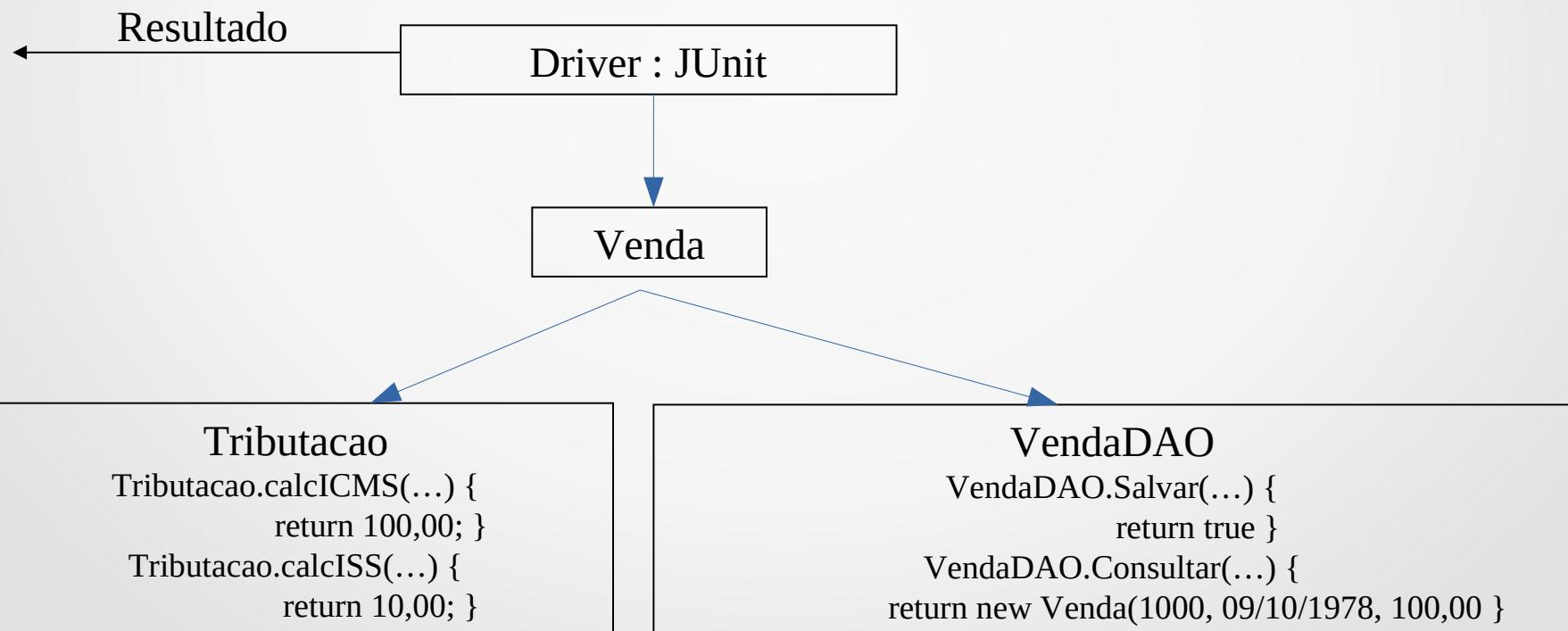
JUnit

- ✓ **driver**: é um “Framework de teste” que aceita dados de casos de teste, passa esses dados para o módulo/classe a ser testada e imprime os dados relevantes que ele recebe de retorno.
- ✓ **stub/MOCK**: são módulos que servem para substituir outros módulos que estejam subordinados, isto é, que são chamados pelo módulo testado;
  - ele usa a interface do módulo subordinado, faz o mínimo de manipulação de dados, imprime uma verificação da entrada e retorna

# Teste de Unidade

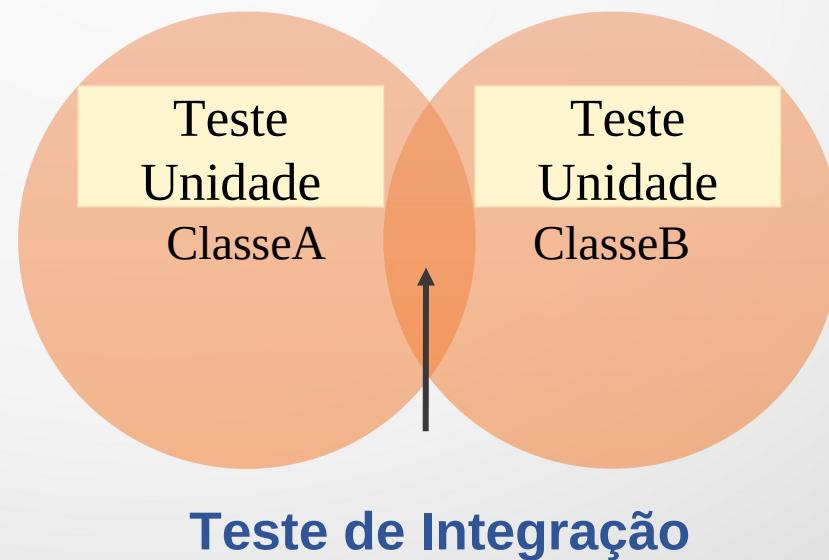
Exemplo em Java:

(Mock) Classes Virtuais:  
Não há lógica nos métodos.



# Teste de Integração

- ✓ Teste focado em verificar se a **comunicação entre componentes** ou módulos da aplicação, e também recursos externos, estão interagindo entre si corretamente.



# Teste de Integração

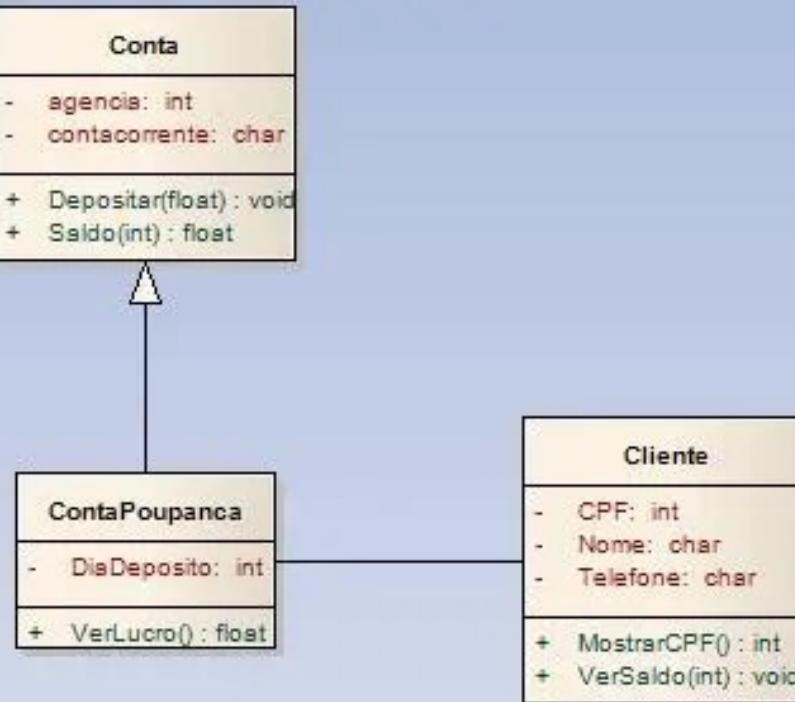


Quando conectamos os plugs,  
eles conseguem transmitir energia?

Identifica falhas de projeto

插座知识点 Socket of knowledge points ▶				
category A 两脚扁型	category B 两脚	category C 八字扁型脚	category D 八字扁型脚	category E 双脚圆型 (4.0mm)
category F 双脚圆型 (4.8mm)	category G 双脚圆型+接地孔	category I 三脚扁型-英国插头	category J 两脚扁型-中国插头	category K 三脚圆型 (6A)
category K 三脚圆型 (16A)	category M 瑞士插头	category N 意大利插头	category O 丹麦插头	category P 以色列插头

# Teste de Integração



Quando conectadas elas continuam produzindo o resultado esperado

# Teste Aceitação/Funcional

- ✓ É um teste do ponto de vista do usuário, se uma determinada funcionalidade está executando corretamente, produzindo o resultado ou comportamento desejado pelo usuário.

Automação no Java :  
Selenium

Caso de Teste

CT-AUT-140:Validar cadastro de cliente

Versão 1

Objetivo do Teste:

Verificar se realiza o cadastro do cliente informando nome, CPF e telefone.

Pré-condições

1. Usuário cadastrado e autenticado no Portal ABC;
2. Usuário com perfil Administrador;
3. Possuir CPF válido.

Ações do Passo	Resultados Esperados:
1 - Acessar a tela de cadastro de cliente no Portal ABC: Menu principal > Cadastros > Cliente; 2 - Preencher os campos com dados válidos: <ul style="list-style-type: none"><li>• Nome</li><li>• CPF</li><li>• Telefone</li></ul> 3 - Clicar em Salvar;  4 - Verifique se o cadastro do cliente foi salvo no Banco de dados.	1 - Sistema exibe a tela de cadastro de cliente com os campos vazios;  2 - Após salvar o cadastro exibe a mensagem de sucesso: "Cliente cadastrado.";  3 - O registro do cliente é salvo no Banco de dados.

# Teste Aceitação/Funcional

Caso de Teste

CT-AUT-140:Validar cadastro de cliente

Versão 1

Objetivo do Teste:

Verificar se realiza o cadastro do cliente informando nome, CPF e telefone.

Pré-condições

1. Usuário cadastrado e autenticado no Portal ABC;
2. Usuário com perfil Administrador;
3. Possuir CPF válido.

Ações do Passo	Resultados Esperados:
1 - Acessar a tela de cadastro de cliente no Portal ABC: Menu principal > Cadastros > Cliente;  2 - Preencher os campos com dados válidos: <ul style="list-style-type: none"><li>• Nome</li><li>• CPF</li><li>• Telefone</li></ul> 3 - Clicar em Salvar;  4 - Verifique se o cadastro do cliente foi salvo no Banco de dados.	1 - Sistema exibe a tela de cadastro de cliente com os campos vazios; <span style="color:red;">X</span> <span style="color:green;">+</span>  2 - Após salvar o cadastro exibe a mensagem de sucesso: "Cliente cadastrado.";  3 - O registro do cliente é salvo no Banco de dados.

# Teste de Sistemas

- ✓ Testa todo o sistema
- ✓ Utiliza bases de dados grandes para teste.

Testes de desempenho

JMeter

Testes de carga (estresse)

Testes de recuperação

Testes de segurança

# Teste de Sistemas

## Testes de carga (estresse)

Usado para verificar o limite de dados processados pelo software até que ele não consiga mais processá-lo

## Testes de desempenho

Intuito de testar o software a fim de encontrar o seu limite de processamento de dados no seu melhor desempenho. No teste normalmente é avaliada a capacidade resposta em determinados cenários e configurações.

# Teste de Sistemas

## Testes de recuperação

Teste utilizado para verificar a robustez e também a capacidade de um determinado software para retornar a um estado operacional após estar em um estado de falha.

## Testes de Segurança

Permite avaliar as vulnerabilidades em aplicações serviços frente a diferentes tipos de ataques de segurança – como Ataques de negação de serviço ou Ataque man-in-the-middle – e descobrir novas vulnerabilidades antes que sejam exploradas por atacantes.

# Testes manuais x automatizados

- Teste manual é um teste onde o desenvolvedor usa o sistema avaliando se um grupo de entradas resultou nas saídas pretendidas;
  - Exemplo:
    - Testes de interface :
      - Ações: Clica nos botões da interface, fornece entradas fictícias, checa os resultados, etc.
    - Testes de código :
      - Crie um função executora (exemplo : método main), chame determinados métodos e verifique o resultado no terminal

Evidentemente, o problema de testes manuais é o fato de ser demorado, caro e atrasar a entrada em produção de um sistema.

## Testes manuais x automatizados

- Um teste automatizado é parecido com o manual, ou seja, monta-se o cenário, executa a ação que quer testar, e verifica se o sistema se comportou da maneira que se esperava.
- Entretanto, em um teste automatizado, os passos citados são descritos em código-fonte.
- Um programa que testa outro programa, permitindo que testes sejam re-executados sempre que preciso e produzam seus resultados automaticamente;

## Benefícios

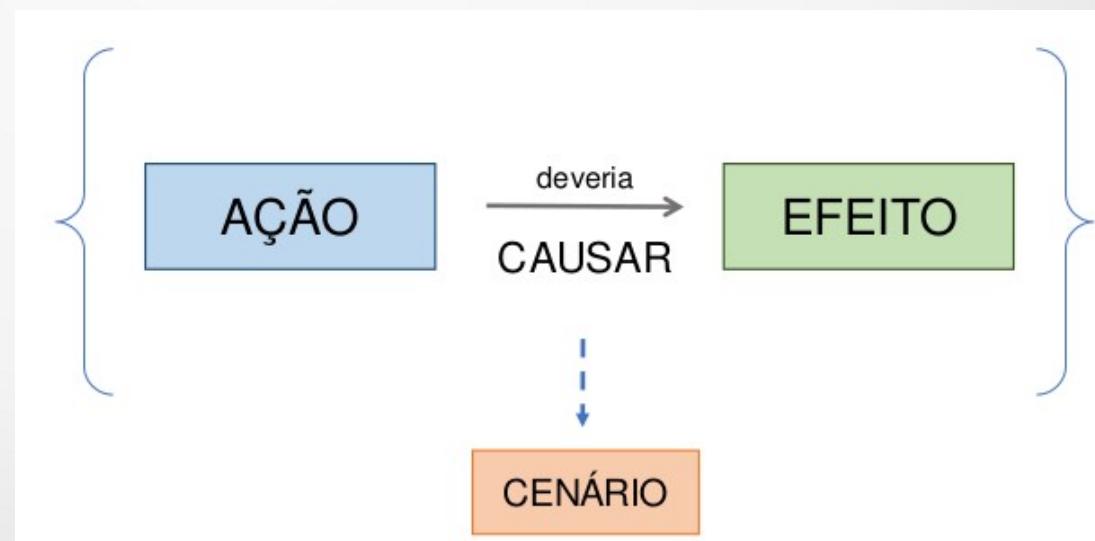
- ✓ Detectar se mudanças violaram as regras do sistema.
- ✓ É uma forma de documentação (comportamento e entradas/saídas esperadas).
- ✓ Redução de custos e manutenções, especialmente em fases avançadas.
- ✓ Incentiva a confecção de um melhor design de classes (mais modulares)

Padrões de Projeto

# Projeto de Teste (manual ou automático)

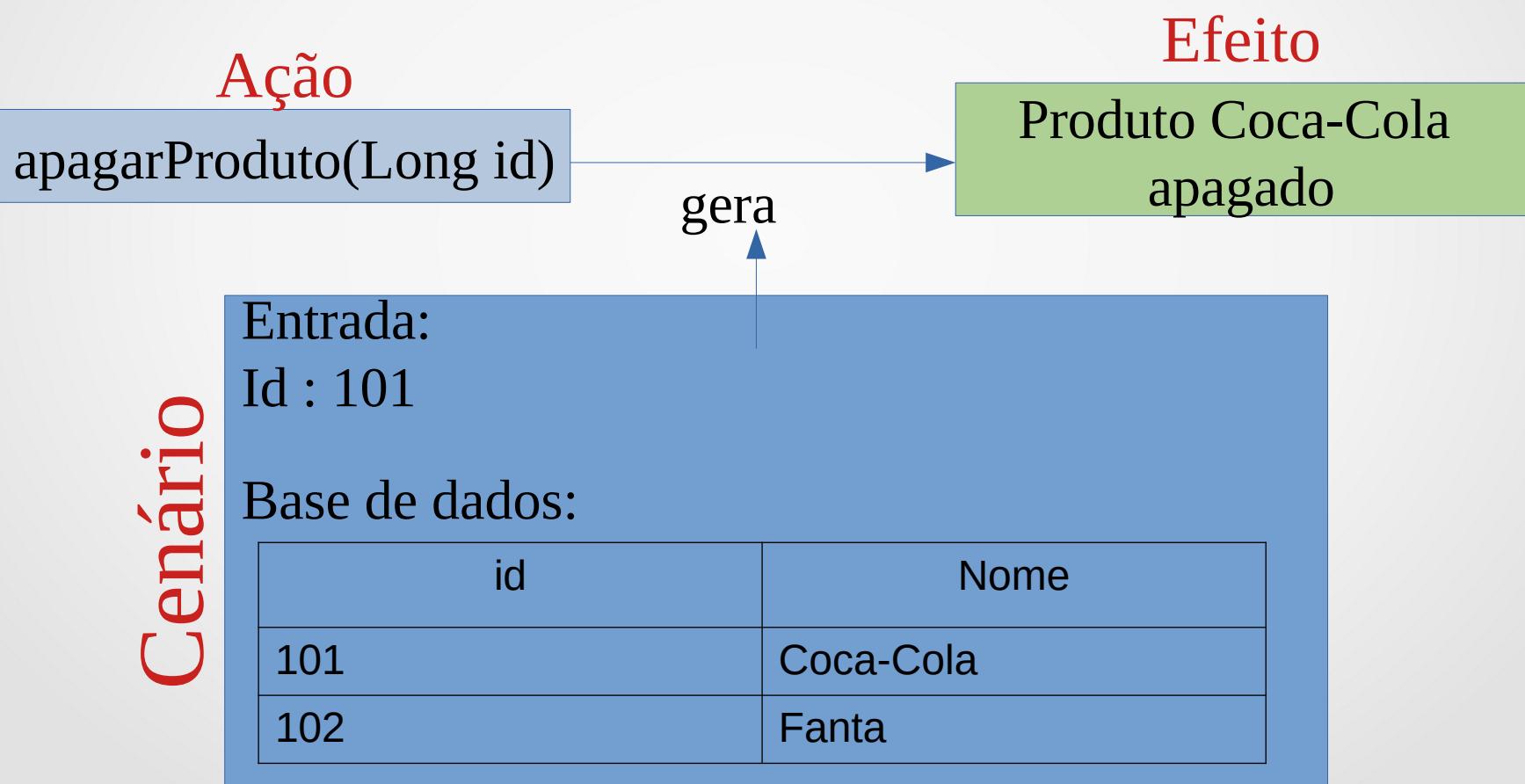
## Partes de um cenário de teste

- Ação: Executar método apagarProduto(Long id)
- Efeito: Deletar produto do banco de dados
- Cenário: Produto existente no banco de dados



# Cenário de Teste

## Testes de Unidade/Integração



# Cenário de Teste

## Testes de Aceitações

### Cenário

#### Ação

Cadastro de Cliente

causa

#### Efeito

Cliente cadastrado no Banco de dados

- 1 - Acessar a tela de cadastro de cliente no Portal ABC: Menu principal > Cadastros > Cliente;
- 2 - Preencher os campos com dados válidos:
  - Nome
  - CPF
  - Telefone
- 3 - Clicar em Salvar;
- 4 - Verifique se o cadastro do cliente foi salvo no Banco de dados.

- 1 - Sistema exibe a tela de cadastro de cliente com os campos vazios;
- 2 - Após salvar o cadastro exibe a mensagem de sucesso: "Cliente cadastrado.";
- 3 - O registro do cliente é salvo no Banco de dados.

# Boas práticas na criação de Testes automatizados

## Padrão AAA

- Arrange: instancie os objetos necessários
- Act: execute as ações necessárias
- Assert: declare o que deveria acontecer (resultado esperado)

Define como escrever o cenário de teste.

# Teste Manual

```
package calculadora;  
  
public class TesteManual {  
    public static void main (String args[]){  
        Calculadora calc = new Calculadora();  
        //cenário de teste  
        double primeiroNumeroDaSoma = 10;  
        double segundoNumeroDaSoma = 20;  
        double resultadoEsperadoDaSoma = 30;  
        //executa (Ação)  
        double resultadoObtidoNaSoma = calc.somar(  
            primeiroNumeroDaSoma, segundoNumeroDaSoma);  
        //valida resultado (Efeito esperado)  
        if (resultadoObtidoNaSoma!=resultadoEsperadoDaSoma){  
            System.out.println("Oops! Deu um resultado não esperado: "+resultado);  
        }  
        else {  
            System.out.println("OK! Passou do teste.");  
        }  
    }  
}
```

Arrange

O.O

Instanciar um  
objeto

Act

O.O

Chamar  
métodos

Assert

# Boas práticas

## **Independência / isolamento:**

- Um teste não pode depender de outros testes, nem da ordem de execução

## **Cenário único:**

- O teste deve ter uma lógica simples, linear
- O teste deve testar apenas um cenário
- Não use condicionais e loops

# Boas práticas

## Previsibilidade

- O resultado de um teste deve ser sempre o mesmo para os mesmos dados
- Não faça o resultado depender de coisas que variam, tais como tempo atual e valores aleatórios.