

TESTE E QUALIDADE DE SOFTWARE

João Choma Neto

joao.choma@gmail.com

Unicesumar – Maringá – 2023/2

ANTERIORMENTE

TESTE DE
UNIDADE



TESTE UNITÁRIO

- Teste unitário é uma prática de teste de software na qual cada componente individual ou "unidade"

TESTE UNITÁRIO

- Uma unidade pode ser a menor parte testável de um aplicativo
 - Função
 - Método
 - Classe
 - Módulo

FRAMEWORK

- JUnit (Java)
- NUnit (.NET)
- pytest (Python)
- Jasmine (JavaScript)

CASOS DE TESTE

- **Crie Casos de Teste**
- Para cada unidade de código, escreva casos de teste que cubram diferentes cenários
- Incluir casos em que a unidade deve produzir resultados corretos
- Incluir casos em que deve lidar com entradas inválidas ou inesperadas

TESTES

- **Escreva os Testes**
- Escreva o código dos testes usando o framework escolhido

TESTES

- **Dados de Teste**
- Inserir dados de teste relevantes para os casos de teste
- Criação de objetos fictícios
- Valores de entrada simulados
- Cenários de teste específicos

EXECUTAR

- **Execute os Testes**
- **Verifique os Resultados**
- O framework de testes geralmente fornecerá relatórios indicando quais testes passaram e quais falharam

REPETIR

- Repetir o processo de teste para todas as unidades de código
- Cada nova modificação recursos ao código repetir os testes
- Cada adição novos recursos, criar novos testes de unidade e repetir os testes

TESTE FUNCIONAL



VISÃO PARA TESTAR

- Segundo Pressman (2011), qualquer produto de engenharia pode ser **testado** por uma de duas maneiras:

VISÃO PARA TESTAR (1)

- Segundo Pressman (2011), qualquer produto de engenharia pode ser **testado** por uma de duas maneiras:
 - Conhecendo a **função** para o qual um produto foi projetado para realizar
 - Construir testes que demonstrem que cada uma das funções é totalmente operacional

VISÃO PARA TESTAR (2)

- Segundo Pressman (2011), qualquer produto de engenharia pode ser **testado** por uma de duas maneiras:
 - Conhecendo o **funcionamento interno** de um produto, podem ser realizados testes para garantir que “tudo se encaixa”
 - Construir testes que demonstrem que as operações internas foram realizadas de acordo com as especificações

CAIXA PRETA

- A primeira abordagem de teste usa uma visão externa e é chamada de teste **caixa-preta**
- Faz referência a testes realizados na interface do software
- Examina alguns aspectos fundamentais de um sistema, com pouca preocupação em relação à estrutura lógica interna do software

CAIXA BRANCA

- A segunda abordagem requer uma visão interna e é chamada de teste **caixa-branca**.
- Fundamenta-se em um exame rigoroso do detalhe procedimental.
- Os caminhos lógicos do software e as colaborações entre componentes são testados.

TESTE FUNCIONAL

- Teste FUNCIONAL ou CAIXA-PRETA focaliza os **requisitos funcionais** do software
- As técnicas de teste caixa-preta permitem derivar séries de condições de entrada que utilizarão completamente **todos os requisitos funcionais** para um programa

TESTE FUNCIONAL

- O teste caixa-preta **não** é uma alternativa às técnicas caixa-branca.
- É uma **abordagem complementar**, com possibilidade de descobrir uma classe de erros diferente

TESTE FUNCIONAL

- Tenta encontrar **erros** nas seguintes **categorias**:
 - Funções incorretas ou faltando.
 - Erros de interface, erros em estruturas de dados ou acesso a bases de dados externas.
 - Erros de comportamento ou de desempenho.
 - Erros de inicialização e término.

TESTE FUNCIONAL

- Os testes funcionais são projetados para simular situações reais de uso do software, a fim de validar se:
 - Os critérios definidos são atendidos
 - As funcionalidades estão implementadas de acordo com as expectativas dos usuários e dos stakeholders

TESTE FUNCIONAL

- O teste funcional tem foco no comportamento do sistema
- Os testes buscam examinar como o software responde às entradas, como o software interage com o usuário e como o software produz saídas esperadas

TESTE FUNCIONAL

- Os testes funcionais são projetados para avaliar o software a partir da perspectiva do usuário
- NÃO estão preocupados com a implementação interna, apenas com o comportamento externo do sistema

TESTE FUNCIONAL

- Os testes funcionais normalmente envolvem a criação de cenários de uso realista
- Simular as ações dos usuários, para verificar se o software executa corretamente nessas situações

COMO FAZER?

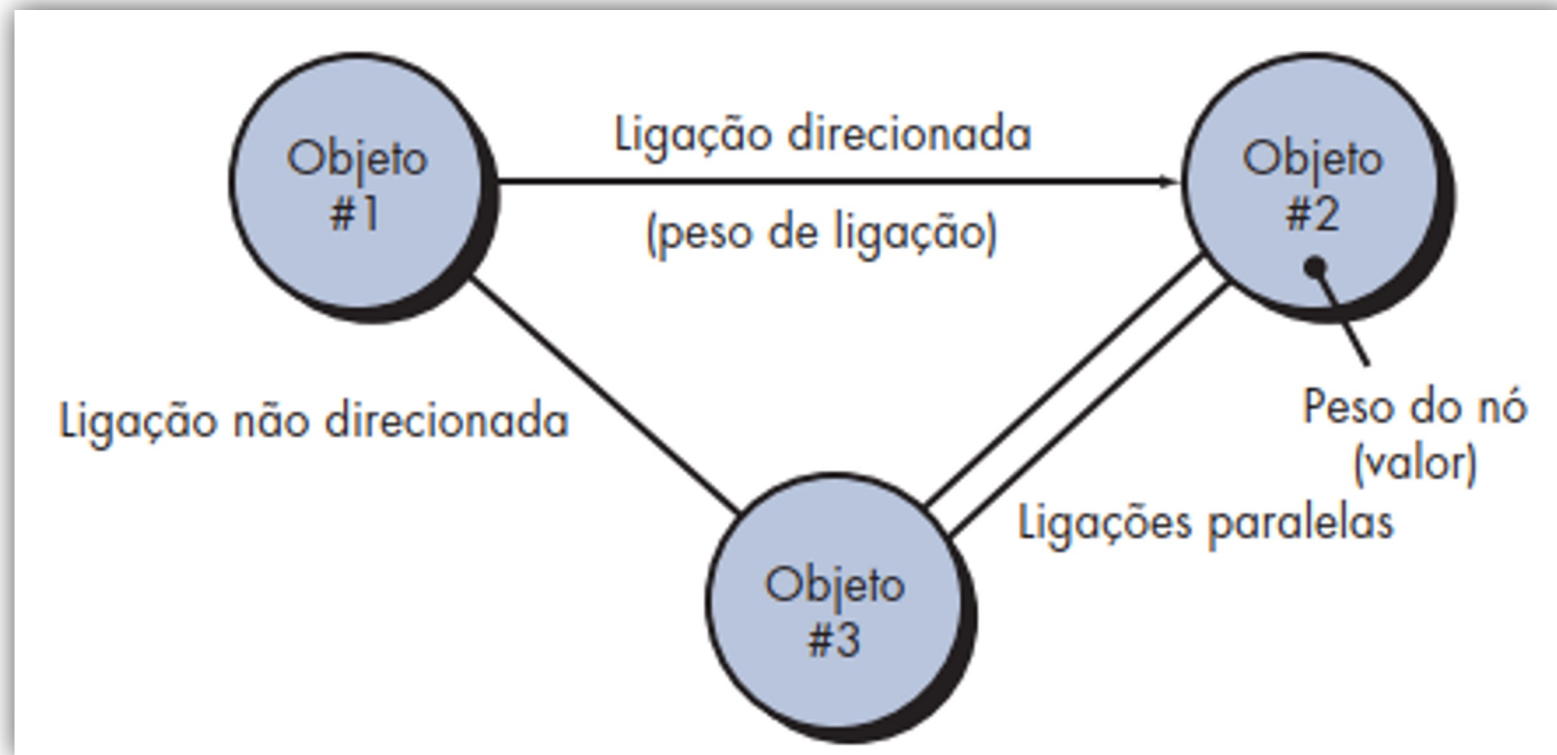
Entender os objetos que são modelados no software e as **relações** que unem esses objetos.

- Uma vez conseguido isso, o próximo passo é definir uma série de testes que verificam que “**todos os objetos têm a relação esperada uns com os outros**”.
- O teste começa **criando um grafo de objetos importantes** e suas relações e então imaginando uma série de testes que abrangerá o grafo.

COMO FAZER?

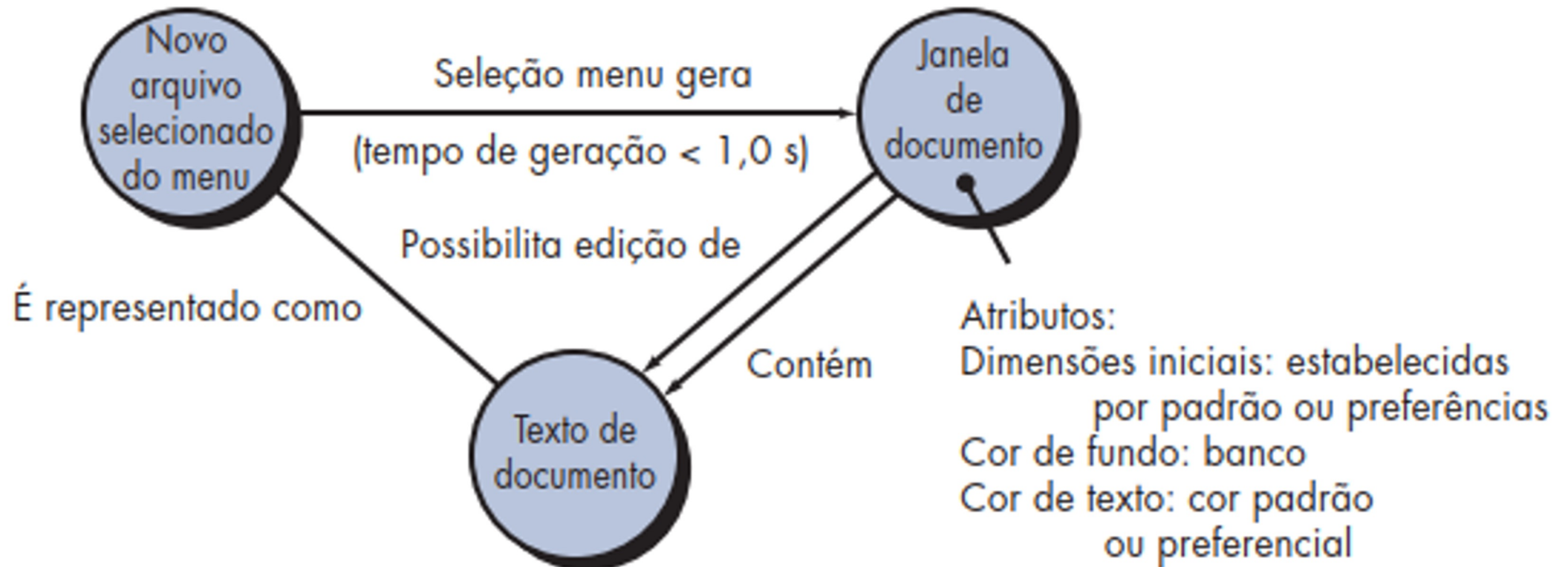
- Você começa criando um grafo:
 - Uma coleção de nós que representam objetos,
 - Ligações que representam as relações entre objetos,
 - Pesos de nó que descrevem as propriedades de um nó (por exemplo, o valor específico de um dado ou comportamento de estado),
 - Pesos de ligação que descrevem alguma característica de uma ligação.

COMO FAZER?



COMO FAZER?

- PROCESSADOR DE TEXTO



ESTUDO DE CASO

- Vamos supor que temos uma função que calcula o preço total de uma compra com desconto, e queremos testar se essa função está funcionando corretamente
- <https://github.com/JoaoChoma/testeequalidadedesoftware/blob/main/SEMANA05/TesteFuncional/src/main/java/CalculadoraDePreco.java>
- <https://github.com/JoaoChoma/testeequalidadedesoftware/blob/main/SEMANA05/TesteFuncional/src/test/java/ExemploTesteFuncional.java>

ESTUDO DE CASO

- O métodos **testCalcularPrecoComDesconto** é um caso de teste funcional
- Ele define um cenário onde uma compra de 100 reais recebe um desconto de 20%
- O método **calcularPrecoTotal** calcula o preço total com desconto.
- A asserção `assertEquals` verifica se o resultado calculado é igual ao valor esperado (80 reais) com uma margem de erro permitida de 0.0001.

FRAMEWORKS PARA TESTE FUNCIONAL

JUnit (Java): Ele suporta testes de unidade, testes de integração e testes funcionais.

TestNG (Java): TestNG é uma alternativa ao JUnit para testes de unidade e funcionais em Java.

pytest (Python): Ele é fácil de usar e oferece recursos avançados de descoberta automática de testes e geração de relatórios.

NUnit (C#): Ele oferece suporte a parametrização de testes e outras funcionalidades avançadas.

FRAMEWORKS PARA TESTE FUNCIONAL

Cucumber (Várias Linguagens): O Cucumber é uma ferramenta de teste de aceitação que utiliza a linguagem Gherkin para escrever cenários de teste em linguagem natural. Ele é frequentemente usado para testes funcionais.

Selenium (Web Applications): O Selenium é uma ferramenta popular para testar aplicativos da web. Ele permite a automação de testes de interface do usuário em navegadores.

FRAMEWORKS PARA TESTE FUNCIONAL

Robot Framework (Várias Linguagens): O Robot Framework é uma estrutura genérica de automação de teste que pode ser usada para testes funcionais e de aceitação, suportando várias linguagens de programação.

Jest (JavaScript/Node.js): O Jest é um framework de teste para JavaScript e Node.js. É especialmente útil para testar aplicativos React e possui recursos como "snapshot testing".

PHP Unit (PHP): O PHPUnit é um framework de teste para PHP, projetado para testes de unidade e funcionais. Ele segue uma abordagem semelhante ao JUnit.

SELENIUM

- <https://www.selenium.dev/>

CLASSES DE EQUIVALÊNCIA

- Classes de equivalência ajudam a organizar e simplificar a criação de casos de teste, permitindo que você escolha representantes de um grupo de dados que compartilham características semelhantes
- Isso é particularmente útil quando se trata de testar diferentes valores de entrada que devem ser tratados de maneira semelhante pelo sistema

CLASSES DE EQUIVALÊNCIA

- Classes de equivalência dividem o conjunto de dados de entrada em grupos ou classes que devem ser tratados da mesma maneira pelo software
- Ao testar um valor em uma classe, você pode fazer suposições sobre o comportamento do sistema em relação a outros valores na mesma classe.

CLASSES DE EQUIVALÊNCIA

- **Cenário: Valores Positivos e Negativos**
- Classes de Equivalência: Valores positivos, valores negativos e zero.
- Exemplo de Casos de Teste:
 - Teste com -10 (valor negativo)
 - Teste com 5 (valor positivo)
 - Teste com 0.

CLASSES DE EQUIVALÊNCIA

- **Cenário: Notas em uma Avaliação**
- **Classes de Equivalência:**
 - Notas abaixo da faixa válida (por exemplo, < 0)
 - Notas válidas (por exemplo, 0 a 10)
 - Notas acima da faixa válida (por exemplo, > 10)
- **Exemplo de Casos de Teste:**
 - Teste com -2 (nota abaixo da faixa válida)
 - Teste com 7.5 (nota válida)
 - Teste com 12 (nota acima da faixa válida)

REFERÊNCIAS

Ian Sommerville – Engenharia de Software. 10^a Edição. São Paulo: Pearson Education do Brasil, 2019.

Roger S. Pressman – Engenharia de software: uma abordagem profissional. 7^a Edição. Porto Alegre: AMGH Editora Ltda, 2011.

Shari Lawrence Pfleeger – Engenharia de Software: teoria e prática. 2^a Edição. São Paulo: Pearson Education do Brasil, 2004.

TESTE E QUALIDADE DE SOFTWARE

João Choma Neto

joao.choma@gmail.com

Unicesumar – Maringá – 2023/2