TESTE E QUALIDADE DE SOFTWARE

João Choma Neto

joao.choma@gmail.com

Unicesumar – Maringá – 2023/2

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 demonstram que cada uma das funções é totalmente operacional

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

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 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

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.

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)

CLASSES EQUIVALÊNCIA

- **Método de teste testAdd() (Adição):**
  - **Classe de Equivalência 1:** Números positivos, por exemplo, (2, 3).
  - **Classe de Equivalência 2:** Números negativos, por exemplo, (-2, -3).
  - **Classe de Equivalência 3:** Zero e um número positivo, por exemplo, (0, 5).
  - **Classe de Equivalência 4:** Um número positivo e zero, por exemplo, (7, 0).

CLASSES EQUIVALÊNCIA

- **Método de teste testSubtract() (Subtração):**
  - **Classe de Equivalência 1:** Números positivos, onde o primeiro número é maior, por exemplo, (6, 3).
  - **Classe de Equivalência 2:** Números negativos, por exemplo, (-6, -3).
  - **Classe de Equivalência 3:** Um número positivo e zero, por exemplo, (5, 0).
  - **Classe de Equivalência 4:** Zero e um número positivo, por exemplo, (0, 7).

CLASSES EQUIVALÊNCIA

- **Método de teste testMultiply() (Multiplicação):**
  - **Classe de Equivalência 1:** Números positivos, por exemplo, (4, 3).
  - **Classe de Equivalência 2:** Números negativos, por exemplo, (-4, -3).
  - **Classe de Equivalência 3:** Zero e qualquer número, por exemplo, (0, 5) ou (0, -7).

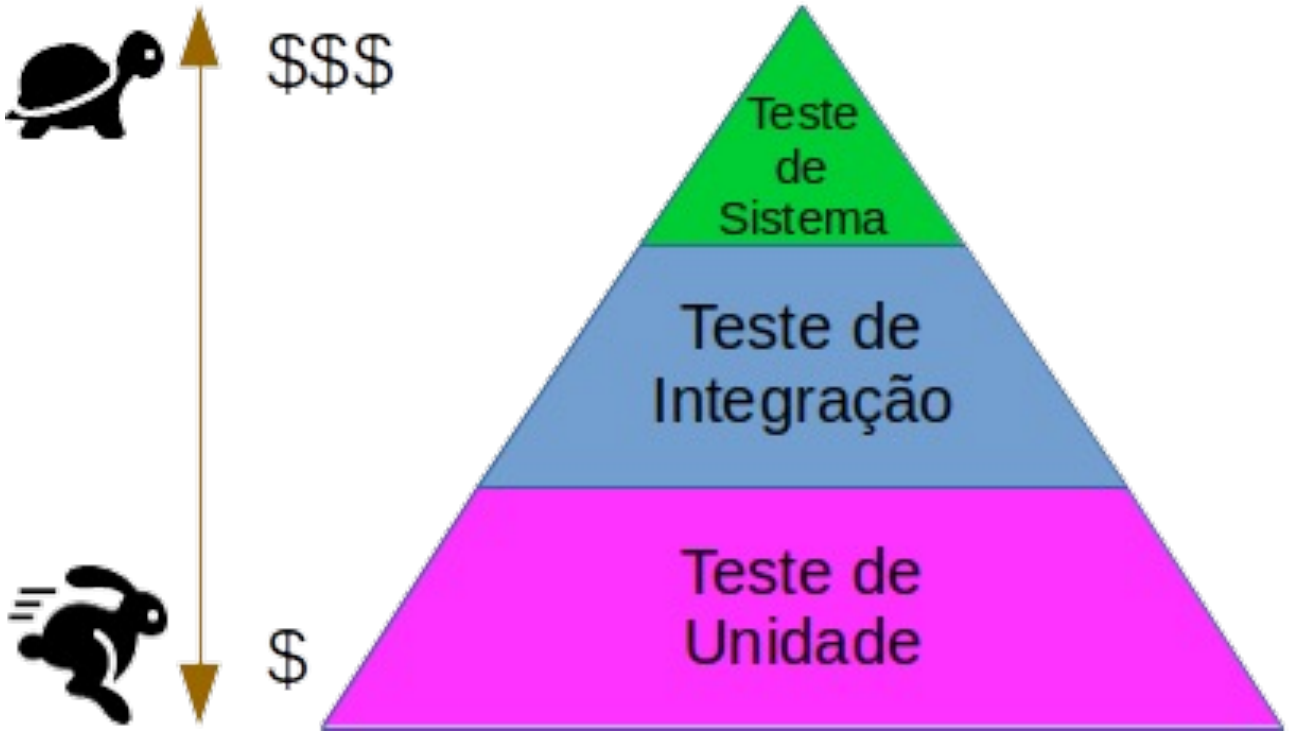
CLASSES EQUIVALÊNCIA

- **Método de teste testDivide() (Divisão):**
  - **Classe de Equivalência 1:** Números positivos, onde o divisor é maior que o dividendo, por exemplo, (6, 12).
  - **Classe de Equivalência 2:** Números negativos, onde o divisor é menor que o dividendo, por exemplo, (-6, -2).
  - **Classe de Equivalência 3:** Divisão por 1, por exemplo, (8, 1).
  - **Classe de Equivalência 4:** Divisão por -1, por exemplo, (10, -1).
  - **Classe de Equivalência 5:** Divisão por zero, por exemplo, (7, 0).

CLASSES EQUIVALÊNCIA

- **Método de teste testDivideByZero() (Exceção de Divisão por Zero):**
  - **Classe de Equivalência 1:** Tentativa de divisão por zero, por exemplo, (10, 0).

TESTE DE UNIDADE



Maior granularidade
Menor quantidade
Mais lentos
Maior custo

Menor granularidade
Maior quantidade
Mais rápidos
Menor custo

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

TESTE UNITÁRIO

- O objetivo principal dos testes unitários é identificar defeitos nas unidades de código antes que elas sejam integradas ao restante do sistema

TESTE UNITÁRIO

- Isolar as unidades de código e testá-las independentemente
- Os desenvolvedores podem verificar se cada unidade executa as tarefas designadas de acordo com suas especificações

TESTE UNITÁRIO

- Identificar problemas cedo no processo de desenvolvimento
- Tornar mais fácil e mais econômico corrigir erros

TESTE UNITÁRIO

- Cada teste unitário deve ser automatizado para que possa ser executado repetidamente e integrado a fluxos de trabalho de desenvolvimento contínuo

TESTE UNITÁRIO

- Processo estruturado para garantir que cada componente individual do código seja testado de maneira isolada

UNIDADE

- **Identifique as Unidades de Código**
- Determine as unidades de código que você deseja testar

FRAMEWORK

- **Escolha um Framework de Testes**
- Utilize um framework de testes apropriado para a linguagem de programação que você está usando

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

REFERÊNCIAS

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

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

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