

TESTE E QUALIDADE DE SOFTWARE

João Choma Neto

joao.choma@gmail.com

Unicesumar – Maringá – 2023/2

ANTERIORMENTE TESTE FUNCIONAL



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

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)

TESTE FUNCIONAL CONTINUANDO



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).

VAMOS PARA O
CÓDIGO



CÓDIGO

- <https://github.com/JoaoChoma/testeequalidadedesoftware/tree/main/SEMANA06/TesteFuncionalAulaPratica>

Selenium IDE

Open source record and playback test automation for the web



CHROME DOWNLOAD



FIREFOX DOWNLOAD



LATEST ZIP



Star 2,496

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

Selenium IDE - seed project*

Project: seed project*

Test suites +

Search tests...

all tests

check

click

click at

comment

confirmation dialog

control flow do

control flow else

control flow else if

control flow if

control flow times

control flow while

execute script*

execute script array

execute script object

execute script primi...

frames

select

▶▶⌂⌚

http://the-internet.herokuapp.com

	Command	Target	Value	
1	execute script	return "a"	myVar	
2	if	\${myVar} === "a"		
3	execute script	return "a"	output	⋮
4	else if	\${myVar} === "b"		
5	execute script	return "b"	output	
6	else			
7	execute script	return "c"	output	
8	end			
9	assert	output	a	

Command

if

//

Target

\${myVar} === "a"

⌕

Value

Description

Opens Window

☐

Log

Reference

⊗

Preparing plugins for test run...

Running 'execute script'

1. executeScript on return 6 with value blah... OK

2. assert on blah with value 6... OK

3. executeScript on true... OK

echo: 6

'execute script' completed successfully

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 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.

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.

TESTE E QUALIDADE DE SOFTWARE

João Choma Neto

joao.choma@gmail.com

Unicesumar – Maringá – 2023/2