

PROGRAMAÇÃO MODULAR

TESTE UNITÁRIO E TDD (DESENVOLVIMENTO DIRIGIDO POR TESTES)

PROF. JOÃO CARAM

2

TESTES DE SOFTWARE

CONTEXTUALIZAÇÃO

SOFTWARE DE QUALIDADE

3

- Número e severidade dos defeitos aceitáveis
- Entregue dentro do prazo e custo;
- Atende aos requisitos/expectativas
- Pode ser mantido eficientemente após a implantação

(Rios; Moreira Filho, 2013)

SOFTWARE DE QUALIDADE

4

- Número e severidade de defeitos em níveis aceitáveis
- Entregue dentro do prazo e custo
- Atende aos requisitos/expectativas
- Pode ser mantido eficientemente após a implantação

(Rios; Moreira Filho, 2013)

TESTE DE SOFTWARE!

5

- Uma maneira de se garantir a qualidade!
- Mostrar que um programa faz o que é proposto e descobrir defeitos antes do uso. *(Sommerville, 2019)*
- Um conjunto de atividades que podem ser planejadas antecipadamente e conduzidas sistematicamente. *(Pressman;Maxim, 2016)*

PRINCÍPIOS DO TESTE DE SOFTWARE

6

■ Verificação

- O sistema foi construído corretamente?

■ Validação

- Foi construído o sistema correto?

7

TESTE UNITÁRIO

TESTE UNITÁRIO

8

- Testa cada componente (classe, método, rotina, página...) individualmente.
- Objetivo principal: garantir que cada unidade funciona de acordo com sua especificação

TEORIA DO TESTE UNITÁRIO

9

- ▣ *“Para cada trecho de código operacional – uma classe ou um método –, o código operacional deve estar pareado com um código de teste unitário”*

“If you can’t write a test for what you are about to code, then you shouldn’t even be thinking about coding.” (George; Williams, 2003)

TESTE UNITÁRIO

10

- Chama o código operacional a partir da sua interface pública, de diversas formas, e verifica os resultados.
- Não precisa ser exaustivo.
 - código de teste já agrega valor, mesmo nos casos centrais.

CLASSE DATA E TESTE UNITÁRIO

MÃOS NA MASSA...

TESTE UNITÁRIO COM XUNIT E JUNIT

12

- xUnit: nome genérico para uma coleção de plataformas de testes unitários.
 - SUnit (Kent Beck, 1998)
- A primeira letra costuma indicar a plataforma.
 - JUnit, RUnit, CUnit, CppUnit, ShUnit, JSUnit...
 - xUnit.net, minitest, Pytest...

TESTANDO UMA CLASSE DATA

13

- A data é válida?
- A data é válida para bissexto?
- A impressão está no formato correto?

TESTANDO UMA CLASSE DATA

14

- A data é válida?
 - O que acontece com data inválida?
- A data é válida para bissexto?
 - Testar ano bissexto e ano comum?
- A impressão está no formato correto?
 - Testar datas inválidas?

15

ENGROSSANDO OS EXERCÍCIOS

CLASSE *MICROONDAS*

COMPLETANDO O EXERCÍCIO

16

- No seu exercício da classe *Microondas*, proponha testes unitários para:
 - Temporizador não pode ter tempo inválido;
 - Ligar o micro-ondas obedece à especificação;
 - Pausa, passagem do tempo e retomada sem alteração do temporizador.

17

TEST-DRIVEN DEVELOPMENT (TDD)

DESENVOLVIMENTO GUIADO POR TESTES

CÓDIGO DE QUALIDADE

18

- ▣ Objetivo final do seu código:

“Código limpo que funciona”

(Ron Jeffries)

CÓDIGO DE QUALIDADE QUE FUNCIONA

19

- Mas talvez não de primeira...

“First make it work. Then make it right. Then make it fast and small.”

(Kent Beck)

TEST-DRIVEN DEVELOPMENT (TDD)

20

- ▣ Metodologia de desenvolvimento de software que enfatiza o teste.

“Escreva o código de teste primeiro. Então, escreva o código operacional e depure o programa até que ele passe no teste.”

TDD

21

- Premissa: os testes de unidade são desenvolvidos antes do código
- Software é desenvolvido em pequenas iterações

“LEIS” DO TDD

22

- Primeira lei: não escreva qualquer implementação antes que você tenha escrito um teste que falhe
- Desenvolvimento orgânico
- Código em execução gera o retorno necessário para tomar as decisões que orientam o desenvolvimento;

“LEIS” DO TDD

23

- Segunda lei: não escreva mais que um teste unitário para demonstrar uma falha
- Teste unitário:
 - Cada desenvolvedor deve escrever seus testes;
 - O ambiente de desenvolvimento fornece respostas rápidas para pequenas mudanças;

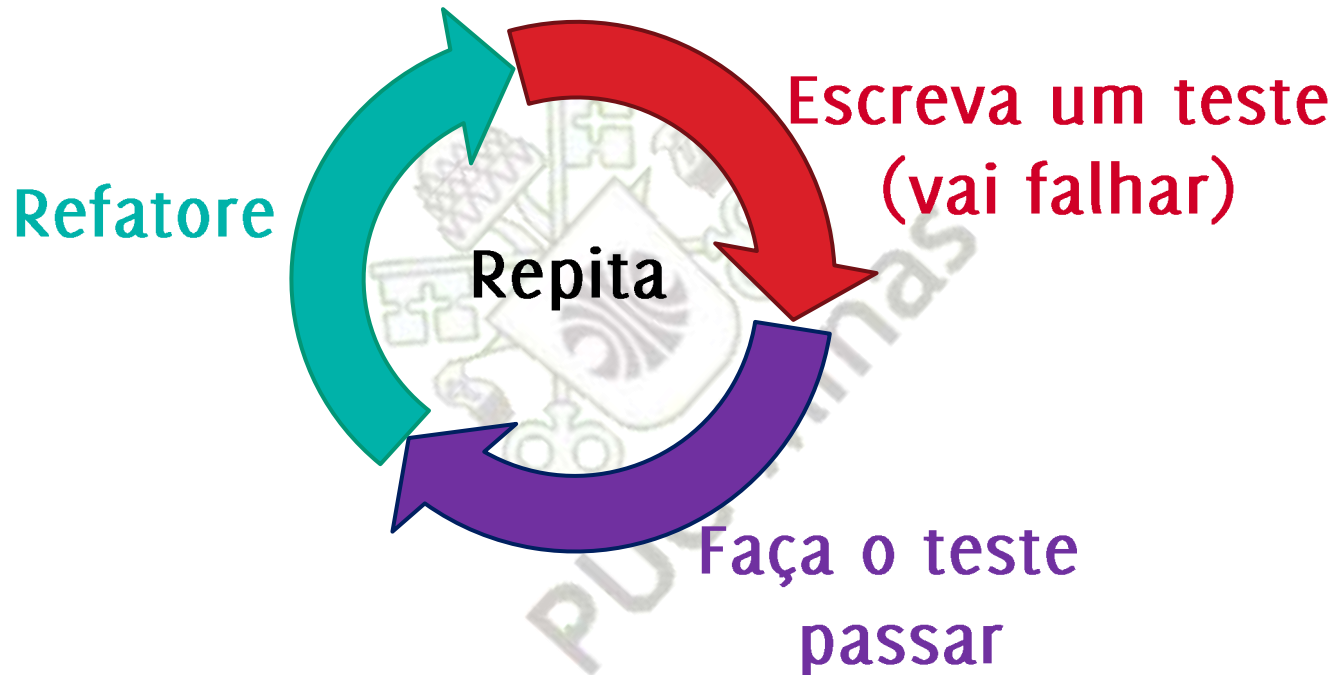
“LEIS” DO TDD

24

- Terceira lei: não escreva mais do que o necessário (*just enough*) para passar por um teste que está falhando;
- O projeto deve ter alta coesão e componentes fracamente acoplados
 - Testes facilitados
 - Evolução facilitada

TDD EM UMA FIGURA

25



TDD EM ESTÁGIOS

26

1. Escreva um teste simples
2. Compile. Ele não deve compilar porque o código de implementação do que está sendo testado ainda não foi escrito.
3. Implemente apenas o código necessário (*just enough*) para fazer o teste compilar.
4. Execute o teste e o veja *falhar*.

TDD EM ESTÁGIOS

27

5. Implemente apenas o código necessário (*just enough*) para fazer o teste passar.
6. Execute e veja o teste *passar*.
7. Refatore o código para torná-lo mais claro.
8. Repita.

28

JUNIT - ASSERTÇÕES

JUNIT E ASSERTS

29

Assert	Objetivo
<code>assertTrue(teste)</code>	falha se teste booleano é false .
<code>assertFalse(teste)</code>	falha se teste booleano é true .
<code>assertEquals(esperado, teste real)</code>	falha se valores não são iguais.
<code>assertNull(teste)</code>	falha se valor não for null .
<code>assertNotNull(teste)</code>	falha se valor for null .

JUNIT E ASSERTS

30

Assert	Objetivo
<code>assertThrows(classe esperada, execução)</code>	falha se a exceção não é lançada.
<code>assertSame(esperado, teste real)</code>	falha se valores não são os mesmos (por meio de <code>==</code>).
<code>assertNotSame(esperado, teste real)</code>	falha se valores são os mesmos (por meio de <code>==</code>).
<code>fail ()</code>	faz com que o teste interrompa sua execução e falhe.

STRING DE INFORMAÇÃO

31

- ▣ Parâmetro adicional, exibido caso o teste falhe

```
assertTrue(metodo(a,b), “metodo com” +a+ “ e ” +b );
```

JUNIT, ANOTAÇÕES E INFORMAÇÕES

32

- Anotações para guiar tanto o ambiente de testes como os desenvolvedores
- *Strings* adicionais como informação de erros

@DISPLAYNAME

33

- Define um nome/descrição para a classe ou o método de teste



INICIALIZAÇÃO E FINALIZAÇÃO

34

- Métodos executados antes ou depois de cada caso de teste

```
@BeforeEach  
void setUp() throws Exception {  
}
```

```
@AfterEach  
void tearDown() throws Exception {  
}
```

INICIALIZAÇÃO E FINALIZAÇÃO

35

- Métodos executados uma única vez, antes ou depois da execução de toda a classe de testes.

```
@BeforeAll
```

```
static void setUpBefore() throws Exception {  
}
```

```
@AfterAll
```

```
void tearDownAfter() throws Exception {  
}
```

@TestMethodOrder

36

- ▣ Define a modo de ordem de execução dos testes:
 - ▣ MethodOrderer.MethodName.class
 - ▣ OrderAnnotation.class
 - @Order(*n*)
 - ▣ MethodOrderer.Random.class

37

EXEMPLO: UMA PIZZARIA

MÃOS NA MASSA

UMA PIZZARIA

38

- Uma pizzaria precisa de um sistema de software para controlar suas vendas.
- As vendas poderão ser feitas presencialmente ou *online*, sem diferenciação inicial entre elas.
- Toda venda tem seus dados registrados assim que for encerrada. Neste momento, deve gerar uma nota de compra simplificada com descrição e valor a ser pago pela pizza.

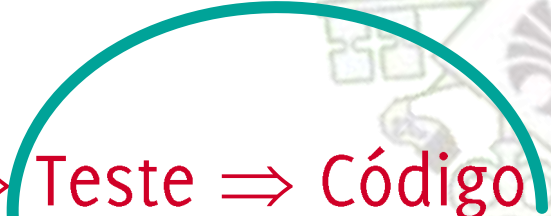
UMA PIZZARIA: MODELO DE VENDAS

39

- A pizzaria está iniciando os negócios com um modelo simplificado de vendas:
 - A pizza básica contém queijo e calabresa. Seu preço inicial de R\$25.
 - A pizza pode ser personalizada com até 8 ingredientes adicionais. Cada ingrediente adicional custa R\$4.
 - Já que todos têm o mesmo valor, não é necessário diferenciar os tipos ou nomes de adicionais.

UMA PIZZARIA: UM SISTEMA

40

- Por onde começamos??
 - Análise \Rightarrow Projeto \Rightarrow Código
 - TDD \Rightarrow Teste \Rightarrow Código \Rightarrow Documentação
- 
- ```
graph LR; TDD --> Teste; Teste -->Codigo; Codigo --> Documentacao;
```



# REFERÊNCIAS

41

- SOMMERVILLE, Ian. Engenharia de software. 10. ed. São Paulo: Pearson Education do Brasil, c2019
- PRESSMAN, Roger S.; MAXIM, Bruce R. Engenharia de software : uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016
- BECK, Kent. TDD: desenvolvimento guiado por teste. Porto Alegre, RS: Bookman, 2010.

# REFERÊNCIAS

42

- GEORGE, Bobby; WILLIAMS, Laurie. A Structured experiment of test-driven development. 2003
- TORCZUK, Arek. Test Driven Development is the best thing that has happened to software design. 2019. Disponível em <https://www.thoughtworks.com/pt/insights/blog/test-driven-development-best-thing-has-happened-software-design>

# OBRIGADO.

DÚVIDAS?