TÉCNICAS DE TESTES ÁGEIS

SÃO JUDAS BUTANTÃ

Jessica Almeida Mesquita - 824156980
Sarah Luanne Bezerra de Souza - 824157332
Kaik Jose Rodrigues - 824159059

Técnica de testes ágeis TDD

O processo seguirá os passos clássicos de TDD: escrever um teste, implementar o código e refatorar.

1. Escrever um Teste

Começamos definindo um teste para uma função que conta palavras em uma string. Vamos usar Python e o framework `unittest`.

```
import unittest

def contar_palavras(texto):
    pass # A função ainda não foi implementada

class TestContadorPalavras(unittest.TestCase):
    def test_contar_palavras(self):
        self.assertEqual(contar_palavras("Olá mundo"), 2)
        self.assertEqual(contar_palavras("Contagem de palavras"), 4)
        self.assertEqual(contar_palavras(""), 0)
        self.assertEqual(contar_palavras(" Espaços "), 0) # Considera espaços vazios como 0 palavras
        self.assertEqual(contar_palavras("Uma frase com muitos espaços"), 7)

if __name__ == '__main__':
    unittest.main()
```

2. Implementar o Código

Agora, vamos implementar a função `contar_palavras` para que os testes passem:

```
def contar_palavras(texto):
    return len(texto.split())
```

3. Executar os Testes

Execute o código dos testes para garantir que todos eles passam:

```
python -m unittest test_contador_palavras.py
```

4. Refatorar o Código

Neste caso, a função `contar_palavras` é simples e já está otimizada. No entanto, se houver mais lógica no futuro ou se quisermos melhorar a legibilidade, podemos refatorar. Por enquanto, ela atende aos requisitos.

A técnica TDD ajuda a garantir que o código atenda às especificações desde o início e

promove um design melhor.

Ciclo Repetitivo

Repita o ciclo: escreva novos testes para novas funcionalidades, implemente o código e refatore. Por exemplo, se você quiser adicionar uma função de subtração:

1. Escreva o teste para a função `subtrai`.
2. Implemente a função.
3. Execute os testes.
4. Refatore se necessário.

Técnica de BDD

O foco do BDD é descrever o comportamento do sistema de forma clara e compreensível, usando uma linguagem que todos os envolvidos possam entender.

Passo 1: Definir o Comportamento

Primeiro, escrevemos as especificações em Gherkin. Vamos criar um arquivo `.feature` que descreve como o contador de palavras deve se comportar.

Feature: Contar palavras

Scenario: Contar palavras em uma frase simples

Given que tenho a frase "Olá mundo"

When eu conto as palavras

Then o resultado deve ser 2

Scenario: Contar palavras em uma frase com espaços

Given que tenho a frase " Espaços "

When eu conto as palavras

Then o resultado deve ser 0

Scenario: Contar palavras em uma frase com pontuação

Given que tenho a frase "Olá, mundo!"

When eu conto as palavras

Then o resultado deve ser 2

Scenario: Contar palavras em uma frase vazia

Given que tenho a frase ""

When eu conto as palavras

Then o resultado deve ser 0

Passo 2: Implementar os Testes

Usando uma ferramenta como Behave em Python, criamos o arquivo de passos correspondente.

```
from behave import given, when, then

import string


class ContadorPalavras:

    def __init__(self):

        self.resultado = 0


    def contar(self, texto):

        texto = texto.translate(str.maketrans('', '', string.punctuation))
        self.resultado = len(texto.split())


@given('que tenho a frase "{frase}"')
```

```
def step_given_frase(context, frase):
    context.contador = ContadorPalavras()
    context.frase = frase

@when('eu conto as palavras')
def step_when_contar(context):
    context.contador.contar(context.frase)

@then('o resultado deve ser {resultado:d}')
def step_then_resultado(context, resultado):
    assert context.contador.resultado == resultado
```

Passo 3: Executar os Testes

Agora você pode executar os testes usando o Behave. No terminal, você pode

rodar: behave

Passo 4: Refatorar o Código

Após a execução bem-sucedida dos testes, você pode revisar o código para melhorar sua legibilidade ou eficiência. No nosso caso, o código está bem estruturado, mas sempre é bom verificar se existem melhorias.

Ciclo Repetitivo

Assim como no TDD, você pode adicionar novos cenários e funcionalidades, sempre escrevendo primeiro as especificações, depois implementando o código e, por fim, refatorando.
O BDD ajuda a alinhar o desenvolvimento com os requisitos do negócio, garantindo que todos na equipe tenham uma compreensão compartilhada do que está sendo construído