TESTES DE UNIDADE PARA QA

APRENDA E INFLUENCIE

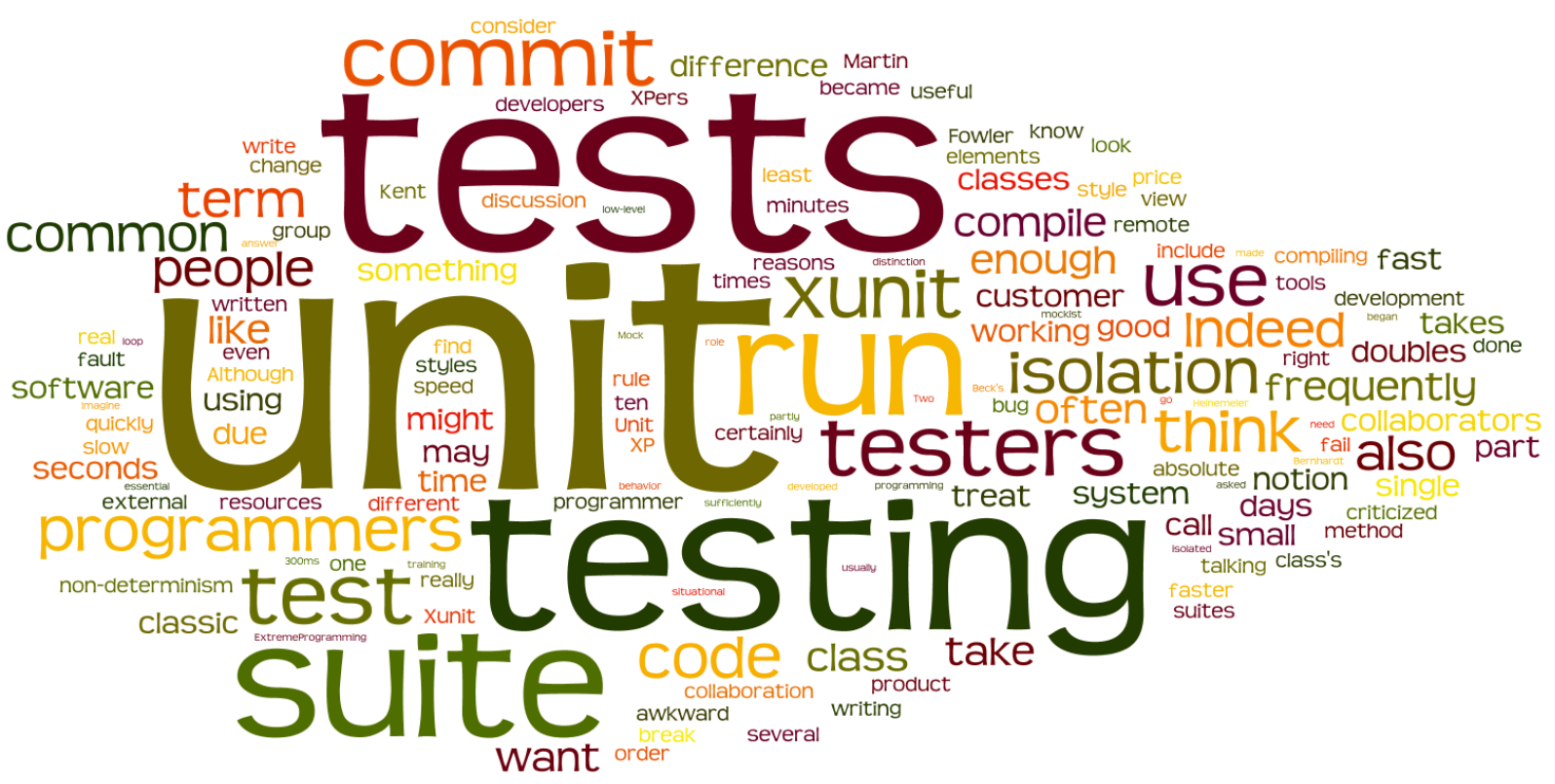
Explore as vantagens dos testes de unidade e inspire os desenvolvedores da sua equipe a adotá-los

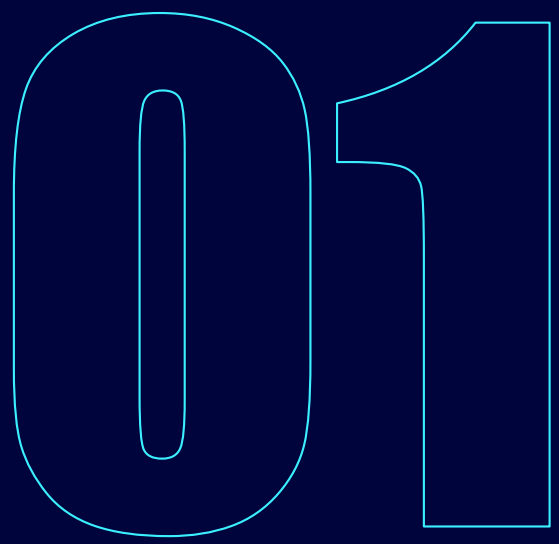
JAQUELINE CONSTANTINO

GUIA PRÁTICO

Testes de Unidade em Desenvolvimento de Software

Os **testes de unidade** são uma parte essencial do desenvolvimento de software. Eles ajudam a garantir que partes individuais do código funcionem conforme o esperado. Neste guia, exploraremos as principais vantagens dos testes de unidade e apresentaremos exemplos práticos para ilustrar seu uso.

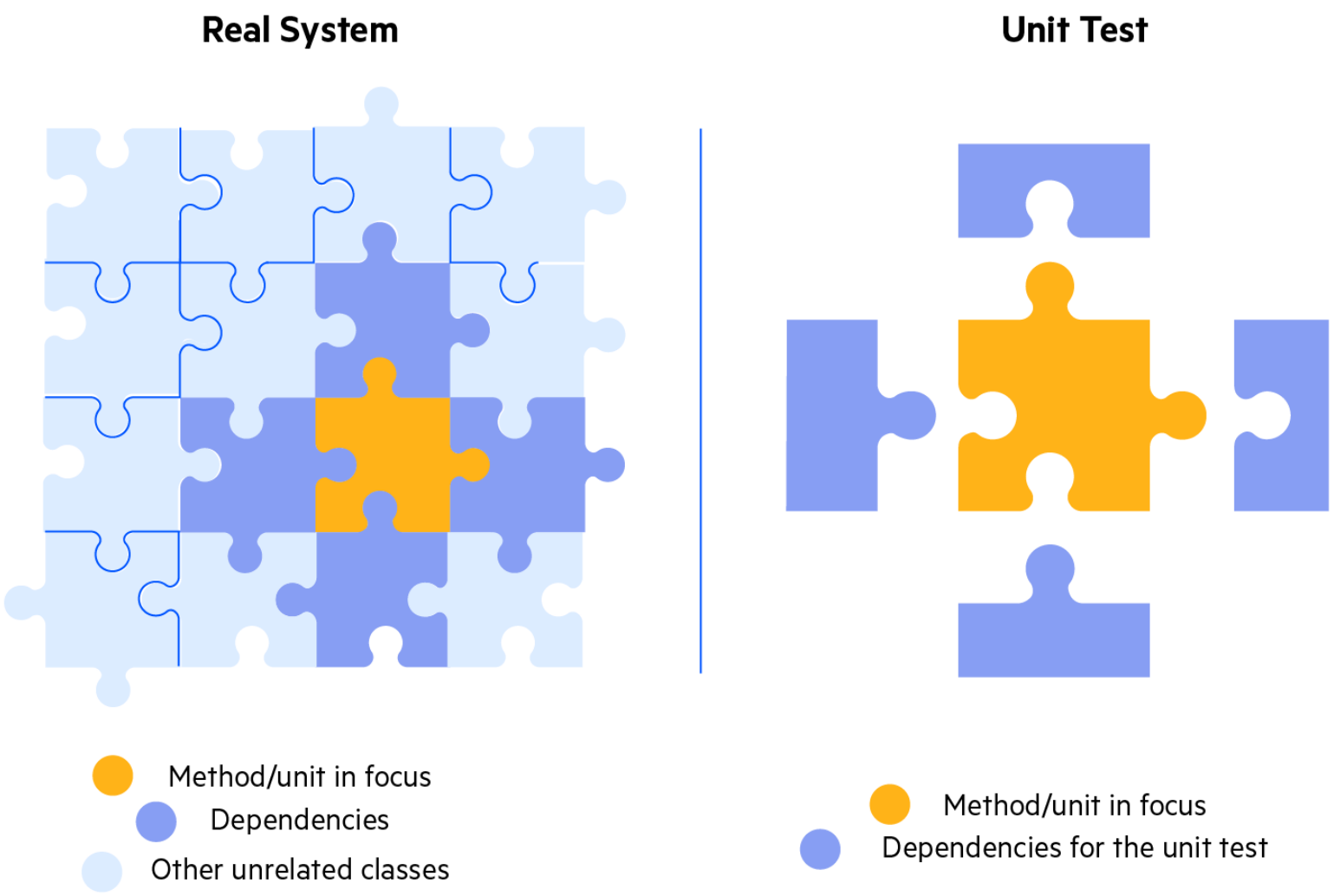




O QUE SÃO?

O QUE SÃO TESTES DE UNIDADE?

Testes de unidade são pequenos trechos de código que verificam se funções ou métodos específicos do seu programa estão funcionando corretamente. Eles focam em componentes isolados, testando seu comportamento de forma independente do restante do sistema.



02

VANTAGENS

1. DETECÇÃO PRECOCE DE ERROS

Identificar problemas o mais cedo possível é crucial para reduzir custos e esforço no desenvolvimento de software. Testes de unidade ajudam a capturar erros logo após a implementação de uma função ou método.

Exemplo em código:

```python
def somar(a, b):
    return a + b

def test_somar():
    assert somar(1, 2) == 3
    assert somar(-1, 1) == 0
    assert somar(0, 0) == 0
```

2. FACILITA REFATORAÇÕES

Com testes de unidade abrangentes, você pode refatorar o código com confiança, sabendo que qualquer mudança inesperada será capturada pelos testes.

Exemplo em código:

```python
def calcular_area_retangulo(largura, altura):
    return largura * altura

def test_calcular_area_retangulo():
    assert calcular_area_retangulo(2, 3) == 6
    assert calcular_area_retangulo(0, 5) == 0
    assert calcular_area_retangulo(7, 8) == 56

# Refatoração
def calcular_area_retangulo(retangulo):
    return retangulo.largura * retangulo.altura

class Retangulo:
    def __init__(self, largura, altura):
        self.largura = largura
        self.altura = altura

def test_calcular_area_retangulo():
    retangulo = Retangulo(2, 3)
    assert calcular_area_retangulo(retangulo) == 6
    retangulo = Retangulo(0, 5)
    assert calcular_area_retangulo(retangulo) == 0
    retangulo = Retangulo(7, 8)
    assert calcular_area_retangulo(retangulo) == 56
```

3. DOCUMENTAÇÃO EXECUTÁVEL

Testes de unidade atuam como uma forma de documentação viva para o código. Eles mostram como cada função deve ser utilizada e quais são suas expectativas de saída.

Exemplo em código:

```python
def inverter_string(s):
    return s[::-1]

def test_inverter_string():
    assert inverter_string("hello") == "olleh"
    assert inverter_string("world") == "dlrow"
    assert inverter_string("") == ""
```

4. FACILIDADE NA INTEGRAÇÃO CONTÍNUA

Integração contínua (CI) é uma prática de desenvolvimento onde o código é frequentemente integrado e testado automaticamente. Testes de unidade são fundamentais para CI, pois permitem verificar rapidamente se novas mudanças introduzem falhas.

Exemplo em código:

```yaml
# Exemplo de configuração de CI em um arquivo .yaml para GitHub Actions
name: Python application

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - name: Check out code
      uses: actions/checkout@v2

    - name: Set up Python
      uses: actions/setup-python@v2
      with:
        python-version: 3.x

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install pytest

    - name: Run tests
      run: |
        pytest
```

5. MAIOR MANUTENIBILIDADE

Código coberto por testes de unidade é geralmente mais modular e menos acoplado, o que facilita a manutenção e evolução do software.

Exemplo em código:

```python
def calcular_imc(peso, altura):
    return peso / (altura ** 2)

def test_calcular_imc():
    assert calcular_imc(70, 1.75) == pytest.approx(22.86, 0.01)
    assert calcular_imc(80, 1.80) == pytest.approx(24.69, 0.01)
    assert calcular_imc(50, 1.60) == pytest.approx(19.53, 0.01)
```

03

PRÁTICAS

ESCREVA TESTES SIMPLES E CLAROS

Testes devem ser fáceis de entender e manter. Evite lógica complexa dentro dos testes. Eles devem ser diretos e verificar apenas uma coisa de cada vez.

Exemplo em código:

```python
def somar(a, b):
    return a + b

def test_somar():
    assert somar(1, 1) == 2  # Simples e claro
    assert somar(-1, 1) == 0
    assert somar(0, 0) == 0
```

UTILIZE FRAMEWORKS DE TESTE

Frameworks de teste como pytest (Python), JUnit (Java), e NUnit (C#) facilitam a escrita, organização e execução dos testes.

Exemplo em código:

```python
# Exemplo com pytest
import pytest

def dividir(a, b):
    if b == 0:
        raise ValueError("Divisão por zero")
    return a / b

def test_dividir():
    assert dividir(10, 2) == 5
    assert dividir(9, 3) == 3
    with pytest.raises(ValueError):
        dividir(1, 0)
```

MOCKS E STUBS

Use mocks e stubs para simular componentes externos ou dependências, isolando a unidade de código que está sendo testada.

Exemplo em código:

```python
from unittest.mock import Mock

def obter_dados_do_banco():
    # Simula uma função que obtém dados de um banco de dados
    pass

def processar_dados(dados):
    # Processa os dados de alguma forma
    return len(dados)

def test_processar_dados():
    dados_mock = Mock()
    dados_mock.return_value = [1, 2, 3, 4]
    assert processar_dados(dados_mock()) == 4
```

TESTE LIMITES E CASOS ESPECIAIS

Garanta que sua função lide corretamente com entradas nos limites dos valores aceitáveis e casos especiais.

Exemplo em código:

```python
def calcular_fatorial(n):
    if n < 0:
        raise ValueError("Número negativo não permitido")
    if n == 0 or n == 1:
        return 1
    fatorial = 1
    for i in range(2, n + 1):
        fatorial *= i
    return fatorial

def test_calcular_fatorial():
    assert calcular_fatorial(0) == 1
    assert calcular_fatorial(1) == 1
    assert calcular_fatorial(5) == 120
    with pytest.raises(ValueError):
        calcular_fatorial(-1)
```

INTEGRAÇÃO COM FERRAMENTAS DE COBERTURA DE CÓDIGO

Utilize ferramentas de cobertura de código para garantir que todas as partes do seu código estão sendo testadas. Isso ajuda a identificar áreas não cobertas por testes.

Exemplo em código:

```
# Instalando cobertura de código com pytest
pip install pytest-cov

# Executando testes com cobertura de código
pytest --cov=seu_modulo tests/

# Gerando relatório de cobertura de código
pytest --cov=seu_modulo --cov-report=html tests/
```

AGRADECIMENTOS

OBRIGADA POR LER ATÉ AQUI

Este Ebook foi gerado por IA, e diagramado por humano.
O passo a passo se encontra no meu Github.

Este conteúdo foi gerado com fins didáticos como atividade da disciplina de “Introdução a Engenharia de Prompts com ChatGPT” do Bootcamp Santander 2024 – Fundamentos de IA para Devs.

https://github.com/cnjaqueline/ebook-created-using-chatGPT