Introdução aos Testes e PyTest

DEVinHouse

Parcerias para desenvolver a sua carreira

SENAI

<LAB365>

AGENDA DA SEMANA

- Introdução e configuração aos Testes e ao PyTest;
- Criação de novos testes unitários;
- Criando testes unitários utilizando o banco de dados;
- Criando testes unitários utilizando o banco de dados II;

AGENDA

- Alinhamentos
- Quais níveis de testes?
- O que é TDD?
- Configurando o projeto para utilizar testes unitários.

Alinhamentos

Antes de começarmos, iremos bater um papo sobre as aulas anteriores.

Quais níveis de testes?

O que caracteriza cada nível de teste:

- Objetivos de cada um;
- Base de teste para cada teste;
- Defeitos e falhas típicas de cada nível;
- Abordagens e responsabilidades específicas.

Teste de unitário ou componente

É concentrado em componentes testados separadamente. Exemplo: Métodos de uma classe.

- Verificar comportamentos funcionais e não funcionais dos componentes;
- Encontrar defeitos no componente;
- Base de teste para cada teste;
- Evitar que os defeitos espalhem para níveis mais altos do sistema;

Teste de unitário ou componente

Falhas típicas:

- Funcionalidade incorreta;
- Problemas no fluxo de dados;
- Código e lógica incorretos;
- Vazamento de memória;

Responsável por implementar e manter:

O programador. Eventualmente o testador auxilia, caso possua o conhecimento necessário.

Teste de integração

Pode ser dividido em dois níveis. Integração entre componentes ou integração entre sistemas ou microsserviços.

- Verificar comportamentos funcionais das integrações;
- Construir confiança na qualidade das interfaces;
- Encontrar defeitos, que pode estar nas interfaces ou nos componentes envolvidos;
- Evitar que os defeitos espalhem para níveis mais altos;

Teste de integração

Falhas típicas:

- Erro no sequenciamento de chamada de interfaces;
- Incompatibilidade de interfaces;
- Falhas de comunicação entre componentes;

Importante!

Diferente do teste de componente, o teste de integração deve focar na integração entre as partes e não no funcionamento unitário de cada funcionalidade, que é responsabilidade do teste de nível unitário.

Teste de End-to-End

Executa as tarefas de ponta a ponta e comportamentos não funcionais exibidos ao executar as tarefas.

- Verificar comportamentos funcionais e não funcionais de todo o sistemas;
- Verificar se o sistema foi implementado conforme especificado;
- Encontrar defeitos;
- Evitar que defeitos cheguem em produção;

Teste de Aceite

Valida o comportamento e a capacidade de todo um sistema ou produto.

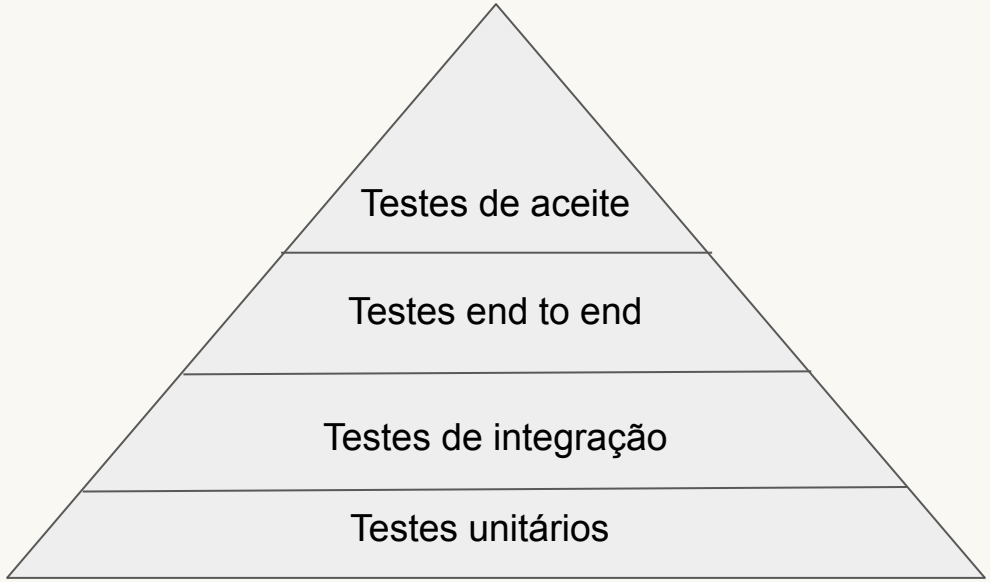
- Estabelecer confiança na qualidade do sistema ou produto;
- Validar que o sistema está completo funcionará como esperado;
- Verificar se os comportamentos funcionais e não funcionais do sistema são os especificados;

Importante!

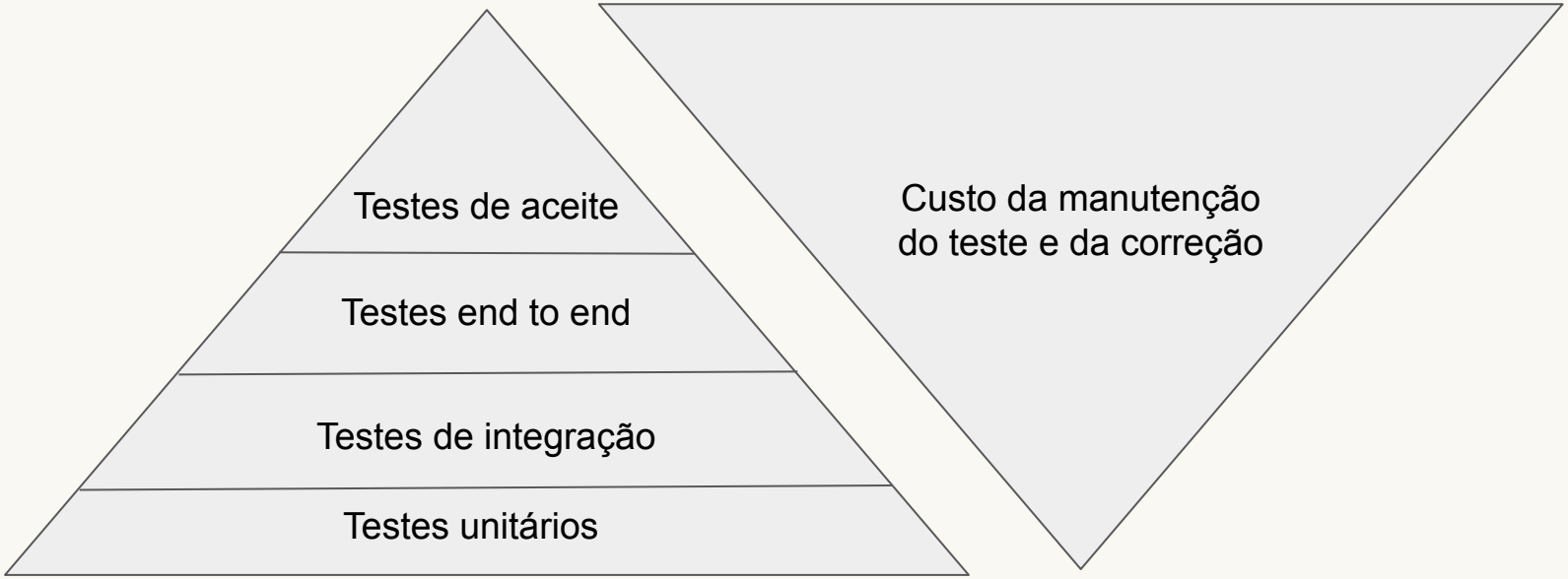
O teste de aceite deve focar em validar junto com o usuário que o sistema atende ao que foi solicitado por ele. O objetivo não é mais encontrar defeito, como no teste de sistema.

Este nível de teste é executado no ambiente de homologação, nas instalações do cliente.

Pirâmide de testes



Pirâmide de testes

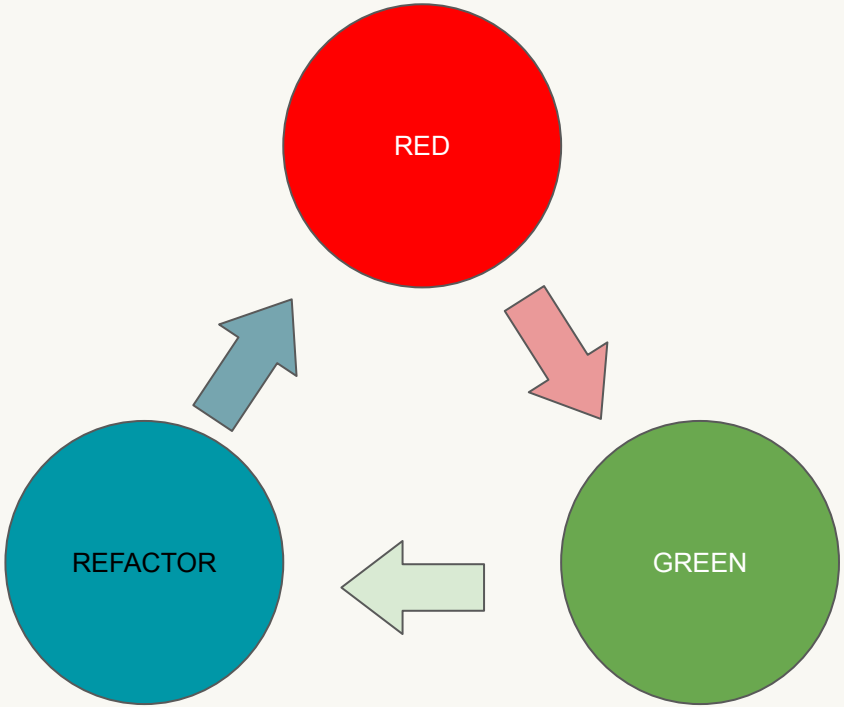


O que é Test Driven Development (TDD)?

No TDD basicamente são criados testes antes do código de cada funcionalidade. Isso funciona em ciclos, onde inicialmente escrevemos o teste e o executamos com o objetivo de que ele falhe. Após isso, criamos o código de nossa funcionalidade e rodamos novamente o teste, que por sua vez irá passar.

Com o teste passando inicia-se o processo de refatoração, onde o código é melhorado, ajustado e otimizado. A partir desse momento o teste deve sempre passar para garantir que nada do que foi planejado para essa funcionalidade foi quebrado. Esse processo é comumente conhecido como “Red, Green, Refactor”.

O que é Test Driven Development (TDD)?



O que é Test Driven Development (TDD)?

Benefícios:

- Segurança para realizar modificações no código e rápido feedback em problemas durante refatorações;
- Menos bugs no produto, ocasionando em um menor tempo de depuração e correção, liberando o foco do time de desenvolvimento;
- Correção mais completa dos bugs encontrados;
- Código mais simples e melhor escrito;
- Viabilização de uma documentação atualizada sobre cada parte do sistema;
- Reforço à cultura da qualidade

Configurando o projeto para utilizar testes unitários.

Passos:

- 1) Clonar a aplicação: git clone https://github.com/DEVin-ConectaNuvem/Modulo-3-Flask
- 2) Configurar a aplicação conforme o README
- 3) Instalar novas dependências de desenvolvimento
  - poetry add --dev pytest pytest-flask pytest-cov
- 4) Criar uma pasta chamada “**tests**” na raiz do projeto
- 5) Dentro da pasta “**tests**”, criar um arquivo chamado **conftest.py**
- 6) Dentro da pasta “**tests**”, criar um arquivo chamado **test_app.py**

Configurando o projeto para utilizar testes unitários.

```
# No arquivo conftest.py da pasta app, adicione os seguintes blocos de código:

import pytest

from flask import json


@pytest.fixture(scope="session")

def app():

  app_on = create_app('testing')

  routes(app_on)

  return app_on
```

Configurando o projeto para utilizar testes unitários.

```
def test_app_is_created(app):
  assert app.name == "src.app"


def test_config_is_loaded(config):
  assert config['DEBUG'] is False


def test_request_returns_404(client):
  assert client.get('/').status_code == 404
```

DEVinHouse

Parcerias para desenvolver a sua carreira

OBRIGADO!

SENAI

<LAB365>