Test Driven Development

Real Programmers test early

Sumário

- Exercícios de TDD

Quando Não Usar TDD?

- Os requisitos não estão claros.
- Testar não faz sentido ou não traz benefícios.
- Os testes são muito lentos.
- O projeto exige verificação visual (UX/UI)

3 Leis do TDD

- Você não pode escrever escrever nenhum código de produção, a não ser para fazer um teste falho passar.
- Você não pode escrever mais testes de unidade que o suficiente para falhar (erros de compilação são falhas).
- Você não pode escrever mais código de produção que o suficiente para passar no único teste de unidade com falha.

Dicas

- Cada teste deve ser independente
  - Não reutilizar dados de um teste em outro teste
- Não é necessário testar as funções da linguagem
- Compare os resultados processados com valores fixos
- Tente encontrar o equilíbrio ideal entre a velocidade de publicação de um software e a cobertura de testes. Pode ser mais eficiente lançar um produto com bugs para que se receba Feedback mais rapidamente.

Em Geral, Não se deve testar Detalhes de implementação, Teste Comportamentos!

Dicas

- Não crie testes específicos para métodos ou classes.
- A origem de um teste deve ser um **requisito do sistema**.
- Crie testes para os **casos de uso** ou histórias de usuários.
- Detalhes de implementação mudam, teste apenas o contrato estável de suas APIs.
- Ao escrever um teste, primeiro imagine o melhor caso, depois trabalhe com as exceções.
- Evitar sistemas de arquivos e bancos de dados, pois dessa forma, um teste pode impactar em outro. E, caso isso seja evitado, os testes podem se tornar lentos.

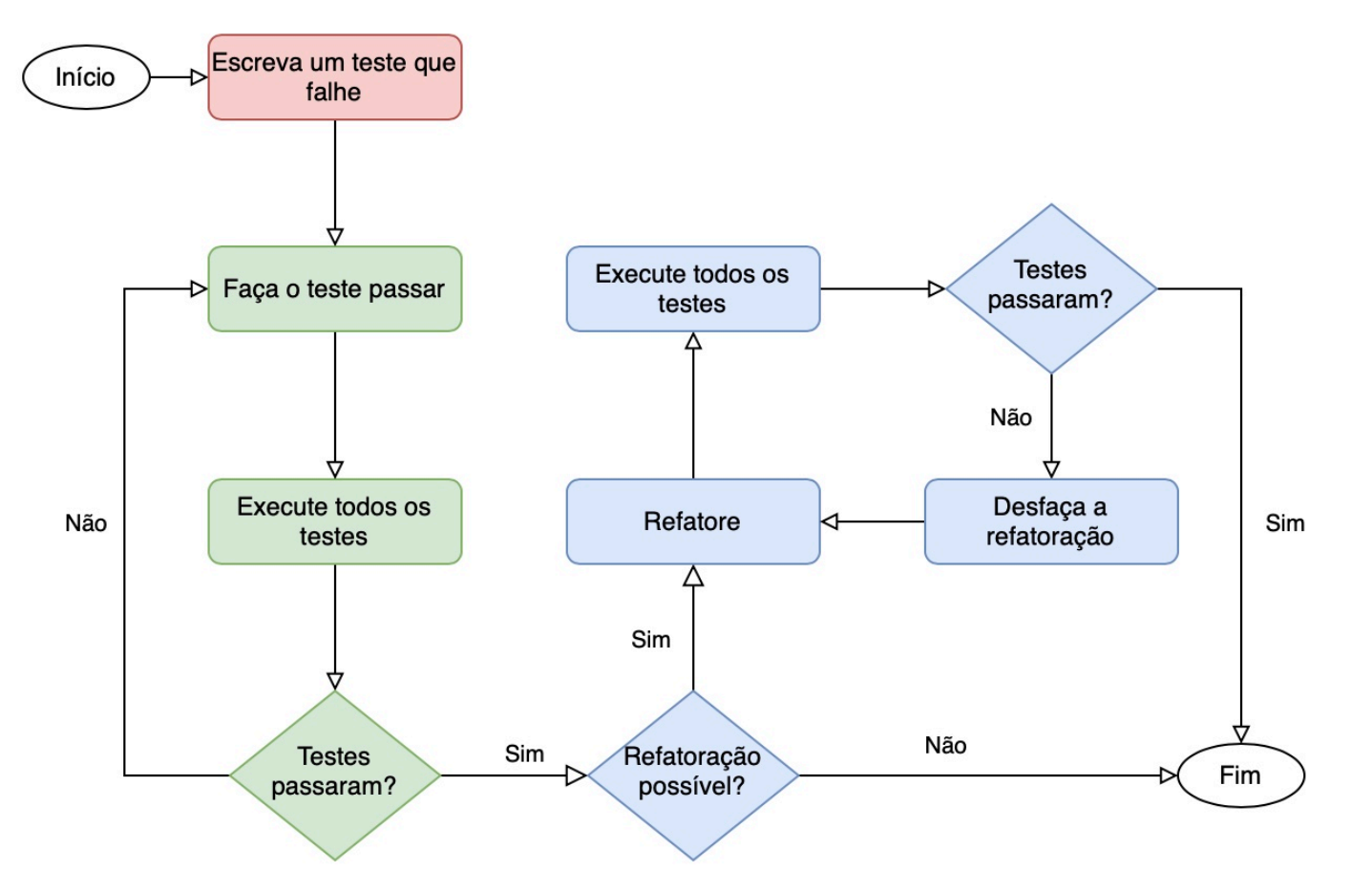
Refatoração

- Um dos pontos críticos do TDD é a refatoração.
  - É na refatoração que se escreve código limpo (Clean Code).
  - Melhora-se o código: removem-se código duplicado, code smells, aplicam-se padrões.
- Dependência é um problema chave em todos os níveis do desenvolvimento de software.
- Portanto, deve-se eliminar a dependência entre código e testes.

Refatoração

- É o processo de mudar o software sem mudar seu comportamento externo e melhorando sua estrutura interna.
- Ao refatorar, melhora-se o código depois que ele foi escrito.

PASSOS: RED GREEN REFACTOR



Como Escrever Bons Testes?

- Um bom teste deve:
  - Se parecer com uma história: toda a informação necessária ao entendimento do teste deve estar contido nele
  - Ter um nome descritivo (GiveWhenThen, ShouldWhen)
  - Ser claro (Arrange-Act-Assert)
  - Verificar apenas um comportamento
  - Usar dados significativos
  - Omitir dados irrelevantes para o teste

Exemplo Guiado

- Fazer uma função que receba um número e retorne uma string contendo a quantidade de centenas, dezenas e unidades.
  - Requisitos:
    - O número deve estar entre 0 e 999.
    - Se o número for menor que 0, retornar “Número inválido”
    - Se o número for maior que 999, retornar “Em construção”
    - Implementar primeiro números de 1 dígito e depois 2 dígitos

Projeto da Disciplina

- **Big Idea**: Desenvolvimento de Software
- **Engage**: Como desenvolver software de qualidade com rapidez?
- **Challenge**: Desenvolver um pequeno protótipo em Python utilizando TDD.
- Entrega: Arquivos .py de produção e arquivos .py dos testes
- Data: 23/08