Raul Germano 3ºDs T.U.

Renato Bueno 3ºDs T.U.

No mundo do desenvolvimento de software, todos nós já passamos por momentos de frustração, especialmente quando os clientes estão pressionando e os bugs parecem insuperáveis. Nesse cenário, a solução que muitos descobriram é o Desenvolvimento Orientado a Testes, ou Test Driven Development (TDD). Neste artigo, vamos explorar o que é TDD, seus benefícios e como implementá-lo.

O que é Desenvolvimento Orientado a Testes (TDD)?

O Desenvolvimento Orientado a Testes é um processo no qual os testes são escritos antes do código real. A ideia é que, à medida que todos os testes passam com sucesso, o código é aprimorado e refinado. Embora pareça que o TDD gira em torno de testes, a premissa principal, de acordo com Kent Beck, é que todos os códigos devem ser testados e refatorados de forma contínua. Portanto, não são os testes em si que são o foco, mas sim os meios para atingir outros objetivos.

Benefícios do Test Driven Development:

Melhoria da qualidade do código: Ao escrever testes antes de codificar, os desenvolvedores estão focados em criar um código mais limpo e eficiente desde o início.

Identificação precoce de problemas: TDD ajuda a detectar problemas no código no início do processo de desenvolvimento, economizando tempo e recursos no longo prazo.

Maior confiabilidade: Código testado exaustivamente tem menos probabilidade de conter bugs, proporcionando um software mais confiável para os clientes.

Aumento da produtividade: A longo prazo, o TDD acelera o desenvolvimento, uma vez que a base de código é mais sólida e menos propensa a falhas.

Eliminação do medo: Como Kent Beck destaca, o TDD visa eliminar o medo no desenvolvimento de aplicações. O medo pode prejudicar a produtividade e a criatividade dos desenvolvedores.

Desenvolvedores escrevendo testes para seu próprio código?

Embora existam argumentos contra desenvolvedores escrevendo testes para seu próprio código, essa recomendação geralmente se aplica a testes de alto nível de aplicação. Para testes de desenvolvimento, faz sentido que o próprio desenvolvedor escreva os testes, pois isso ajuda a entender os requisitos e o processo de implementação. Lembre-se de que o TDD não gira em torno dos testes, mas sim de criar um código robusto.

Integração do TDD com a metodologia Agile:

O Test Driven Development tem raízes na metodologia Agile. Em 1996, a equipe do projeto C3 da Chrysler praticou a programação test-first, uma das práticas que posteriormente se tornaram conhecidas como Extreme Programming (XP). Membros dessa equipe, como Kent Beck, Martin Fowler e Ron Jeffries, estiveram envolvidos na criação do Manifesto Agile. O TDD é uma prática essencial no contexto Agile, pois está alinhado com o valor "Software funcional acima de documentação abrangente" ao proteger o software funcional por meio de testes e criar documentação como um subproduto natural.

Ciclos do TDD:

Redação de Testes: Primeiro, escreva um teste unitário que inicialmente irá falhar, uma vez que o código ainda não foi implementado.

Implementação do Código: Crie o código que satisfaça esse teste, ou seja, implemente a funcionalidade desejada. Essa primeira implementação deve passar no teste escrito no ciclo anterior.

Refatoração do Código: Após a implementação bem-sucedida, refatore o código para aprimorar a legibilidade e outras qualidades. Em seguida, execute o teste novamente para garantir que a nova versão do código também passe no teste originalmente escrito.

É importante salientar que o TDD não se trata apenas de escrever testes; é uma metodologia abrangente para o desenvolvimento de software.

O ciclo de TDD é composto por três fases:

1. Fase "Red" (Vermelho): Nesta etapa, você escreve um teste automatizado que falhará quando executado.

2. Fase "Green" (Verde): Aqui, você implementa o código necessário para que o teste escrito na fase anterior seja aprovado.

3. Fase "Refactor" (Refatoração): Após a implementação bem-sucedida, você aprimora o código, tornando-o mais funcional e limpo.

Esse ciclo é comumente referido como "red-green-refactor" ou "red-green-blue".

O objetivo do TDD é aplicar esses ciclos de teste unitário, que avaliam o comportamento de classes e métodos específicos. Ao invés de implementar o código diretamente, você cria testes relacionados a essas classes e métodos a serem desenvolvidos. Uma vez que o código não foi implementado, o teste inicialmente falhará (fase "Red"). Em seguida, você implementa a funcionalidade necessária para que o teste passe (fase "Green"). Posteriormente, é possível refatorar o código, eliminando duplicidades e tornando-o mais eficiente (fase "Refactor"). Após a refatoração, você executa o teste novamente, e ele deve passar com sucesso. Dessa forma, você obtém um código funcional e adequadamente testado.

Ciclo "Red" (Vermelho):

Primeiro, escrevemos o teste que inicialmente falhará. Vamos criar um arquivo chamado test\_calculadora.py com o seguinte conteúdo:

import unittest

from calculadora import soma

class TestCalculadora(unittest.TestCase):

def test\_soma(self):

resultado = soma(2, 3)

self.assertEqual(resultado, 5)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

Neste ponto, o código de soma ainda não está implementado, portanto, o teste falhará.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ciclo "Green" (Verde):

Agora, implementaremos a função soma. Crie um arquivo chamado calculadora.py com o seguinte conteúdo:

def soma(a, b):

return a + b

Agora, execute o teste novamente. Se o código estiver correto, o teste passará (fase "Green").

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ciclo "Refactor" (Refatoração):

Nesta etapa, não há refatoração necessária, pois nosso código é simples. Portanto, não faremos alterações no código.

Agora, você pode executar o teste novamente para garantir que ele ainda passe com sucesso.

Para executar os testes, use o seguinte comando no terminal:

python -m unittest test\_calculadora.py

Se tudo estiver correto, você verá a saída indicando que o teste foi bem-sucedido. Esse é um exemplo básico de TDD em Python, com os três ciclos: red, green e blue, usado para criar uma função simples de som

Referências:

<https://blog.onedaytesting.com.br/test-driven-development/>

<https://www.nimblework.com/pt-br/agile/desenvolvimento-orientado-a-testes-tdd/>

https://www.treinaweb.com.br/blog/afinal-o-que-e-tdd#:~:text=Ciclo%20do%20TDD&text=Perceba%20a%20exist%C3%AAncia%20dos%20tr%C3%AAs,mais%20funcional%20e%20mais%20limpo

a.