Membros do grupo:
Stevão Severo Rodrigues, Lucas Florão, Matheus Mattos, Adrian Feijó Fagundes
Turma: TDS24-1N

TDD - TEST DRIVEN DEVELOPMENT

Método TDD ?

TDD, ou Desenvolvimento Orientado por Testes (Test Driven Development), é uma abordagem de desenvolvimento de software que coloca uma ênfase significativa na escrita de testes automatizados antes de escrever o próprio código do software.

Criada por Kent Beck e um dos pilares do XP (Extreme Programming), essa técnica prioriza os testes de códigos baseada em pequenos ciclos de repetições, em que antes é criado um teste para cada funcionalidade do sistema. Assim, desenvolvemos o nosso software baseado em testes que são escritos antes do nosso código de produção.

Ciclo de Desenvolvimento

O ciclo de desenvolvimento do Test Driven Development, é dividido em algumas etapas: RED, GREEN e REFACTOR. Cada etapa cumpre um papel diferente no desenvolvimento.

RED

A primeira fase do ciclo TDD é conhecida como “Red” ou “Vermelho”. Nesta etapa, o desenvolvedor escreve um novo teste para uma funcionalidade ou melhoria que ainda não existe no código.

Como o código correspondente a esse teste ainda não foi implementado, o teste falhará. Esta falha é esperada, e é um passo essencial no TDD. Ela permite confirmar que o teste foi bem elaborado e consegue identificar a ausência da funcionalidade desejada.

Ao ver o teste falhar (e entender a falha), o desenvolvedor tem a confirmação de que pode e precisa avançar para a próxima etapa, escrevendo o código necessário

GREEN

Após o teste falhar na fase “Red” , vem a fase “Green” ou “Verde”. Nesta etapa, o objetivo é escrever o código mínimo necessário para o teste passar. Mas, por quê?

O foco não está na qualidade ou na eficiência do código, mas sim em fazer com que o teste – que anteriormente falhou – não falhe mais. (Por isso, antes Red e agora Green).

Agora, o desenvolvedor pode confirmar que a funcionalidade ou correção implementada realmente pode resolver o problema identificado. Após o teste, a funcionalidade está “tecnicamente completa” , mas não necessariamente está no seu estado ideal em termos de estrutura de código e/ou performance

REFACTOR

A última fase do ciclo TDD é a “Refactor” ou “Refatorar”. Após a fase do teste, o desenvolvedor agora volta sua atenção para melhorar a qualidade do código – mas sem alterar seu comportamento.

Esta etapa é dedicada à limpeza, otimização e organização do código, garantindo que ele esteja bem estruturado e fácil de entender e manter.

A refatoração é essencial para manter a base de código saudável e sustentável a longo prazo, e garante que as mudanças não quebraram nenhuma funcionalidade – já que os testes existentes continuarão funcionando (e, consequentemente, obtendo resultados positivos)

Este ciclo Red-Green-Refactor pode ser repetido várias vezes, conforme necessário, em cada nova implementação ou alteração de funcionalidade. É importante executar todos os testes existentes após cada alteração no código, para garantir que as novas implementações não quebrem funcionalidades existentes.

Testes de software

O teste de software é essencial no desenvolvimento, verificando se o software atende às suas especificações. Testes podem ser feitos em partes individuais ou no sistema completo, incluindo dados e segurança.

Existem cerca de 17 tipos diferentes de testes, cada um adequado para diferentes softwares, como: unidade, integração, operacional, positivo-negativo, regressão, caixa-preta, caixa-branca, funcional, interface, performance, carga, aceitação do usuário, volume, stress, configuração, instalação e segurança.

Os testes podem ser manuais, realizados por um profissional, ou automatizados, realizados por outro software.

No TDD os testes utilizados são os testes de unidade.

Testes de unidade

Um teste de unidade é um bloco de código que verifica a precisão de um bloco menor e isolado de código de aplicação, normalmente uma função ou um método. Ele é projetado para verificar se o bloco de código é executado conforme o esperado, de acordo com a lógica teórica do desenvolvedor por detrás dele.

Para escrever testes de unidade são utilizadas algumas técnicas, e um padrão de design para os testes de Unidade no TDD é o Arrange-Act-Assert (AAA).

Esse padrão estrutura a organização dos testes em três fases distintas:

1. Arrange (Preparação) Nessa fase, prepara-se o cenário do teste, definindo o estado inicial necessário para sua execução. Isso inclui criar objetos, configurar parâmetros e definir variáveis, garantindo que o código seja testado em um ambiente controlado e previsível.

2. Act (Execução) Nessa fase, a ação ou comportamento a ser testado é executado. Isso envolve chamar o método ou função com os parâmetros definidos na fase de preparação, visando testar a funcionalidade e obter os resultados desejados.

3. Assert (Verificação): Nesta fase, verifica-se se os resultados da execução correspondem ao comportamento esperado. Compara-se o resultado real com o esperado conforme a especificação do teste. Se coincidir, o teste é bem-sucedido; caso contrário, indica um problema no código ou na especificação.

Frameworks para testes

Muitas vezes, o tempo necessário para escrever testes unitários e usá-los de forma eficiente pode ser um problema. Para resolver isso, existem diversas ferramentas específicas para testes unitários em cada linguagem de programação. Por exemplo:

- Python: `unittest`, `pytest`
- JavaScript: `Jest`, `Mocha`
- Java: `JUnit`, `TestNG`
- C#: `NUnit`, `xUnit.net`
- Ruby: `RSpec`, `Minitest`
- PHP: `PHPUnit`, `Codeception`
- Swift: `XCTest`, `Quick`

- Kotlin: `KotlinTest`, `Spek`

Essas ferramentas ajudam a simplificar e acelerar o processo de teste, aumentando a eficiência e a qualidade do código.

Benefícios

Devido a escrita de testes antes do desenvolvimento das funcionalidades, diversos benefícios são obtidos entre eles:

- Foco nos requisitos
- Facilita a manutenção
- Feedback Rápido
- Melhoria no Desigh
- Melhoria na qualidade do código
- Redução de custos a longo prazo

Quando usar

O TDD é excelente para projetos onde a qualidade do código é uma prioridade. Ele ajuda a garantir que o software funcione conforme o esperado desde o início.

Para projetos mais complexos, o TDD ajuda a decompor o problema em partes menores e mais gerenciáveis, facilitando o desenvolvimento e a detecção de erros.

Se o projeto em que você está trabalhando possui uma expectativa de muitos recursos e uma longa vida útil, o TDD é a alternativa certa para gerenciar o controle de qualidade do seu código. Dessa forma, os testes servirão para mostrar se o código contém bugs ou não, caso haja a necessidade de retornar a uma funcionalidade antiga e adicionar mais código.

Quando não usar

Para projetos que exigem a criação de uma DEMO ou um Protótipo, o TDD pode atrasar o processo. E, por último, se a equipe de desenvolvimento não está acostumada com o TDD, a eficiência do projeto pode ser afetada – e, neste caso, o ideal seria oferecer cursos, palestras ou workshops sobre o assunto antes do projeto se iniciar.

Não faz sentido escrever testes automatizados para métodos extremamente simples, como getters e setters. Eles são linhas de código muito simples que não agregam valor ao produto final, no entanto é bom tomar cuidado ao decidir se é necessário ou não um teste.

TDD x Kanban

Considerando que o TDD é uma metodologia voltada diretamente para o desenvolvimento de software e o Kanban uma metodologia para organização de maneira geral, é possível trabalhar bem com ambas em conjunto.
O ciclo red-green-refactor do TDD, que é claro na forma como as funcionalidades são desenvolvidas, se alinha perfeitamente com o uso de um quadro Kanban.

Desafios de implementação

Apesar de ser uma ótima metodologia, o TDD não é uma metodologia fácil de aplicar e algumas das dificuldades ao aplicar o método são:

Adoção inicial: Implementar o TDD em um projeto ou organização que não o utiliza pode ser difícil no início, especialmente se os membros da equipe não estiverem familiarizados com a abordagem.

Curva de aprendizado: Aprender a escrever testes eficazes e implementar o ciclo red-green-refactor pode levar tempo e esforço, especialmente para desenvolvedores sem experiência prévia em testes.

Dicas ao utilizar o TDD

- Sempre siga a ordem da metodologia
- Preocupe-se em refatorar e melhorar o código
- Não escreva vários testes simultaneamente
- Execute os testes e obtenha o seu feedback
- Não construa um ambiente de testes que demore muito tempo para ser executado
- Não combine testes de integração com os testes unitários. Cada um deve ter o seu próprio arquivo de teste

Referências

https://mercadoonlinedigital.com/blog/tdd/

https://www.objective.com.br/insights/test-driven-development/

https://www.treinaweb.com.br/blog/afinal-o-que-e-tdd

https://www.devmedia.com.br/test-driven-development-tdd-simples-e-pratico/18533

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

https://dev.to/matheusgomes062/tdd-e-seus-fundamentos-4i5e

https://coodesh.com/blog/dicionario/o-que-e-tdd/

https://blog.xpeducacao.com.br/tdd-test-driven-development/

https://fwctecnologia.com/blog/post/tdd-test-driven-development

https://blog.betrybe.com/tecnologia/tdd-test-driven-development/

https://www.devmedia.com.br/tdd-fundamentos-do-desenvolvimento-orientado-a-testes/28151

https://coodesh.com/blog/candidates/metodologias/tdd-e-seu-significado-por-que-ele-ajuda-a-aumentar-a-sua-produtividade/

https://www.dio.me/articles/o-que-e-metodo-tdd-e-como-ele-pode-me-ajudar-no-desenvolvimento-de-aplicacoes

https://medium.com/@lenonrodrigues/test-driven-development-tdd-e-o-contexto-do-pattern-aaa-arrange-act-assert-b1e88a79bda2#:~:text=Um%20padrão%20comumente%20utilizado%20em,legíveis%20e%20fáceis%20de%20entender.

https://www.objective.com.br/insights/testes-de-software/