O que é TDD?

Prof. Marcos Caldas

TDD é uma sigla em inglês para TEST-DRIVEN DEVELOPMENT que nada mais é que desenvolvimento orientado a teste ele foi descoberto em 2003 por Kent Beck engenheiro de software americano que também é um dos pais do **XP.**

A prática do TDD

- A prática do TDD é uma das mais populares da área de desenvolvimento de software e tem como premissa principal escrever, codificar o teste do seu código antes mesmo do seu código existir.
- Ou seja, você precisaria saber que tipo de resultado gostaria de ter no seu código e assim um teste seria criado para validar isso depois o próprio código seria desenvolvido e este teste já serviria para testá-lo.
- A grande vantagem deste tipo de abordagem é que independente das alterações feitas em seu código é preciso que ela passe por um teste para validá-lo.

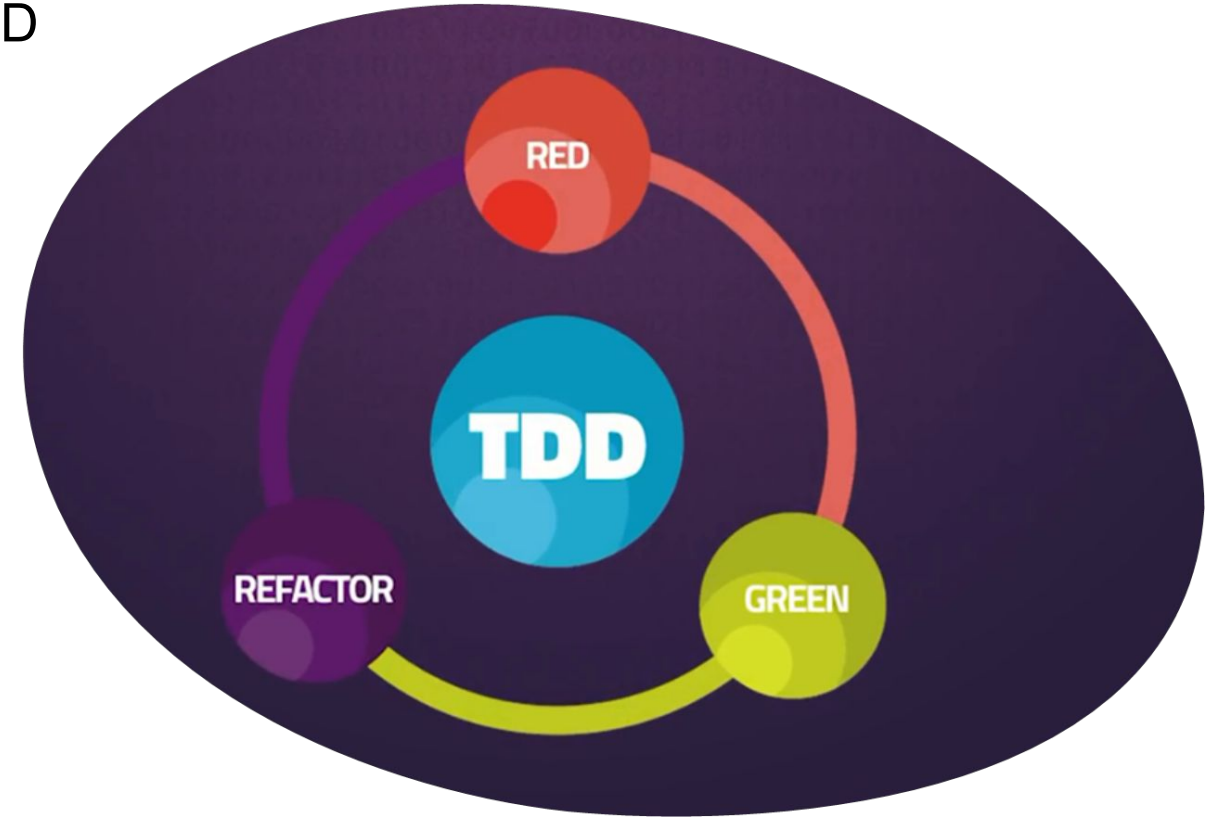
A base do TDD

O TDD se baseia então em testes unitários, que nada mais é que validar a menor parte testável do seu sistema, também conhecida como unidade, que geralmente é uma classe ou um método.

Mas TDD não são só testes unitários, podemos afirmar que os testes unitários são as ferramentas e o TDD é a técnica, isso significa que é possível escrever um teste unitário sem necessariamente ter um TDD.

O TDD é muito mais que eliminar os prints, echo, writes e outras formas bem inconvenientes de debugar as funções, ele é uma prática portanto nada mais importante que praticar seus conceitos e regras para aprender a utilizar o TDD.

Ciclo de um TDD



O Processo de Desenvolvimento com TDD é sempre :

- RED - Falha
- GREEN – Desenvolva solução
- REFACTOR – Refatore a funcionalidade do teste

RED - Falha

- O primeiro passo é escrever um teste que falha, sim é para falhar mesmo pois o código de sua funcionalidade ainda nem existe.

- Depois execute o teste e acompanhe a falha, pode ser um código falso mesmo que retorne o erro.

GREEN – Desenvolva solução

- Desenvolva primeiro a solução mais simples
- Depois teste novamente só que agora é para o teste passar
- Se não passar volte para entrada anterior

REFACTOR – Refatore a funcionalidade para passar

- Refatore sua funcionalidade e agora escreva ela por completo.
- Nesta etapa elimine redundância e acoplamentos e vá deixando o designer do código mais bonito e legível.
- Passe para próxima funcionalidade iniciando um novo teste.

Uma forma de lembrar:

- Primeiro faça, depois faça certo, depois faça melhor.

Usar o TDD diminui a produtividade

- Não, isso é mito, é bem verdade que a prática necessita de uma adaptação da cultura e estilo de programação, mas com o tempo todos vão indagar porque não usava isso antes.

- Em verdade é que o TDD o programador não vai mais perder tempo depurando e tentando encontrar as falhas e não teremos mais códigos desnecessários pois o código será pensado em passar no teste e no fim teremos um código com mais confiabilidade e mais qualidade por estas razões da para perceber que esta perda de tempo é na realidade mito.

Como usar o TDD

- Existem diversas ferramentas e até frameworks para se utilizar o TDD em sua linguagem favorita e com estas ferramentas é possível se ganhar ainda mais tempo e qualidade no software associando as práticas do TDD com o processo de Deployment contínuo .

- Toda vez que for realizar uma alteração no seu sistema e for fazer o deploy dele estes mesmos testes que foram criados durante o processo de desenvolvimento poderão ser automatizados e uma vez que o número de acertos não for o esperado o deploy pode ser cancelado desta forma se ele for realizado sem erros você terá a segurança que tudo está em ordem.

Motivos para o uso do TDD

- Feedback rápido sobre a nova funcionalidade e sobre as outras funcionalidades existentes no sistema;
- Código mais limpo, já que escrevemos códigos simples para o teste passar;
- Segurança no Refactoring pois podemos ver o que estamos ou não afetando;
- Segurança na correção de bugs;
- Maior produtividade já que o desenvolvedor encontra menos bugs e não desperdiça tempo com depuradores;
- Código da aplicação mais flexível, já que para escrever testes temos que separar em pequenos "pedaços" o nosso código, para que sejam testáveis, ou seja, nosso código estará menos acoplado.