02 de agosto de 2023

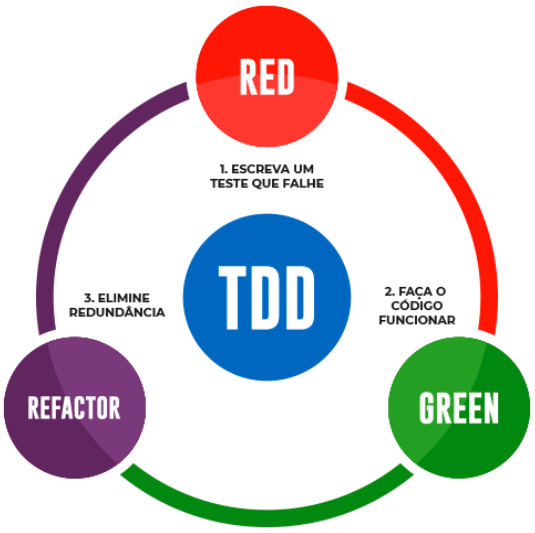
Por: Beatriz Linhares, Isadora Palú, Isadora Velho e Vishnu Priya.

TDD (Test Driven Development)

É uma prática de desenvolvimento de software baseada na testagem unitária, antes mesmo do código começar a ser desenvolvido e implementado no sistema. Ou seja, nessa técnica são criadas pequenas iterações e estas são testadas uma a uma de forma isolada.

O principal benefício no uso do Test Driven Development é que, graças às testagens regulares, quando o código for escrito ele já vai ser criado de uma maneira simplificada e sem perder a qualidade do sistema. Além disso, as chances de existirem grandes erros no código serão menores, pois muitos testes foram feitos antes dele ser criado.

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.



Red: escreva um pequeno teste automatizado que, ao ser executado, irá falhar;

Green: implemente um código que seja suficiente para ser aprovado no teste recém-escrito;

Refactor: refatore o código, a fim dele ser melhorado, deixando-o mais funcional e mais limpo.

O principal objetivo do Test-Driven Development é fazer com que os códigos sejam testados e refatorados de forma contínua.

EXEMPLO 1:

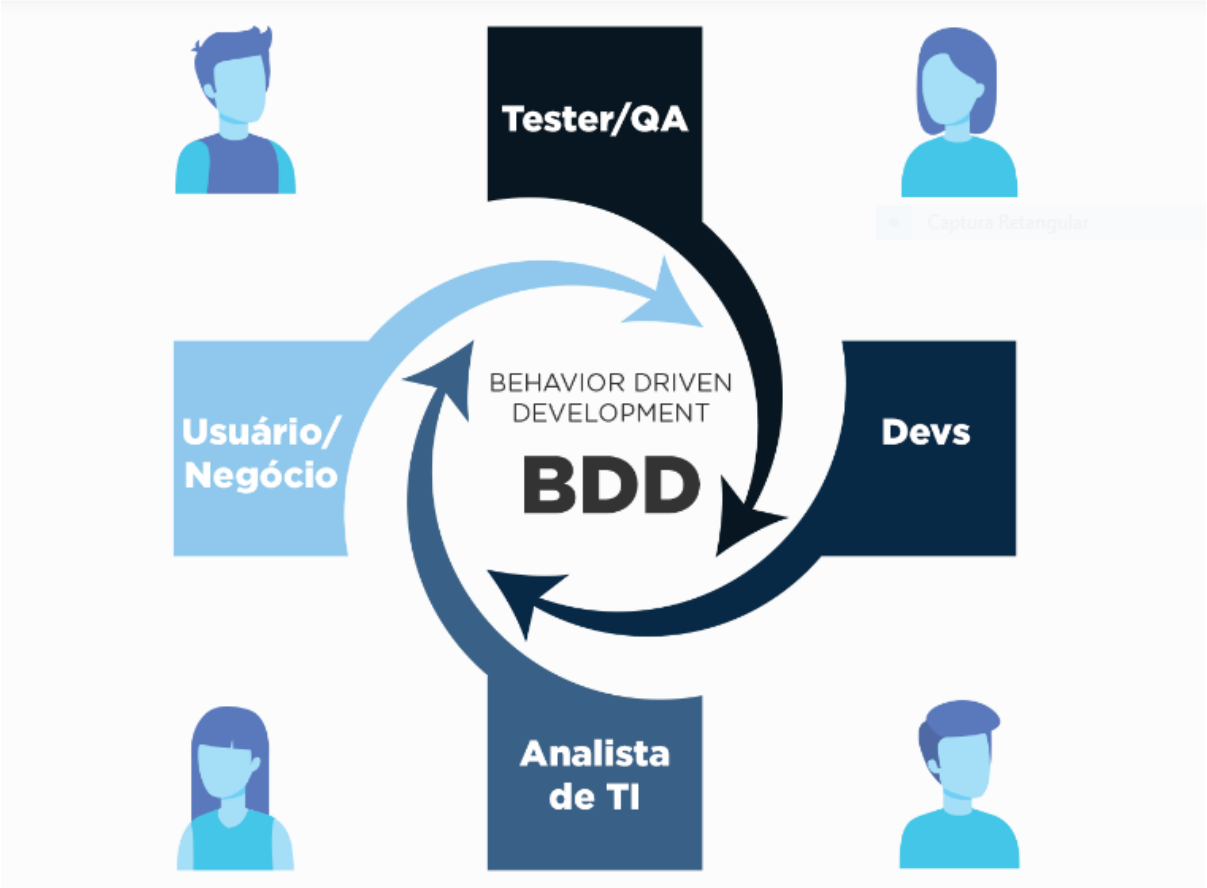
- Criar um sistema simples de uma calculadora usando Java e JUnit, onde iremos escrever os testes, depois implementar a funcionalidade e após isso refatorar.
- Inicialmente teremos a classe de testes onde criaremos nosso primeiro teste para a operação de soma de dois números inteiros; Criamos esse teste antes mesmo de ter criado a classe Calculadora e o método da operação de soma.
- Ao executar essa classe de teste visualizamos o erro, pois nem mesmo a classe existe. Consequentemente, o método da operação de soma também não.
- Agora implementamos a classe Calculadora e criamos o método com a operação de soma de dois números inteiros.
- Rodamos novamente nossa classe de teste e a partir desse ponto o teste da operação soma começa a passar com sucesso.
- A partir desse ponto o ciclo se inicia novamente, onde iremos criar um novo teste para uma nova operação, que seria a subtração de dois números inteiros.
- Rodamos nossos testes novamente e a classe de testes falha, pois o método da operação de subtração não está implementado ainda em nossa classe Calculadora.
- Vamos então à classe calculadora e refatorar o código, adicionando o método de subtração de dois números inteiros logo após nosso método de soma.
- Rodamos novamente a classe de testes e agora os dois testes estão passando com sucesso, garantindo que nosso novo código não gerou bugs e está atendendo corretamente as funcionalidades.
- Nesse exemplo pudemos ver um passo a passo bem simples, porém objetivo, demonstrando como funciona o Test Driven Development na prática e como os conceitos são aplicados no código de teste e das funcionalidades.

EXEMPLO 2:

Um sistema de cadastro feito para uma empresa, onde é necessário testar o cadastro, senha e permissão de cada usuário. Os testes serão feitos em cada parte individualmente, garantindo que ao juntá-los haverá uma redução de chances de erros. Primeiramente o sistema de cadastro do usuário, verificando que tal seja

salvo no banco de dados de forma correta. Também se a senha verifica, não deixando passar a diferença de letra maiúscula ou minúscula por exemplo. A permissão do usuário também deve ser verificada, garantindo que um usuário comum não terá permissões que somente um administrador deve ter.

BDD (Behavior Driven Development)



- É uma técnica utilizada para integrar regras de negócios e linguagem de programação, caracterizando-se por um vocabulário bem específico e pequeno, que minimiza as dificuldades de comunicação e possibilita todos os membros do time utilizarem uma mesma linguagem para realizar o trabalho.
  Com a utilização dos BDD's, é possível unir diversas práticas ágeis para um desenvolvimento qualitativo, sugerindo que analistas/testadores escrevam os cenários antes mesmo que a implementação aconteça, possibilitando assim, que os desenvolvedores tenham uma visão geral do projeto antes de codificá-lo.

- O BDD vem sendo cada vez mais utilizado por permitir:

- Que negócio e tecnologia se refiram a funcionalidades do software de uma só forma;
- A visualização do valor de cada funcionalidade do software para o negócio;
- Analisar, projetar e planejar tudo de cima a baixo sem retorno decrescente;
- Compartilhamento de conhecimento entre analistas, desenvolvedores e testers;
- Promover uma documentação dinâmica do software sem qualquer esforço a mais.

- **O BDD durante o Teste do Software:** Na metodologia do BDD o analista de testes consegue planejar e criar os seus testes antes mesmo do desenvolvimento ser iniciado, assim, quebramos o paradigma de que o BDD seria aplicado como uma camada durante o desenvolvimento e passamos a vê-lo como um processo completo durante todo o projeto.

- **Escrevendo seu teste funcional utilizando BDD:** O planejamento e a escrita dos seus testes funcionais com a semântica do BDD é uma atividade importante, pois é a partir dessa etapa que você irá criar uma documentação viva que será utilizada por todos os membros do seu time.

- **Semântica do BDD:** Para criar cenários de teste utilizando BDD usaremos as palavras chaves:

  - Dado (Given);
  - Quando (When);
  - Então (Then).

  Veja como estas palavras chaves devem ser utilizadas:

  - Dado: Define as pré-condições verdadeiras para executar o seu teste
  - Quando: Define a ação que será executada
  - Então: Seguindo a ação descrita no QUANDO define o resultado esperado para o seu teste
  - E: adiciona uma sentença positiva no Dado, no Quando ou no Então.

DDD (Domain-driven Design)

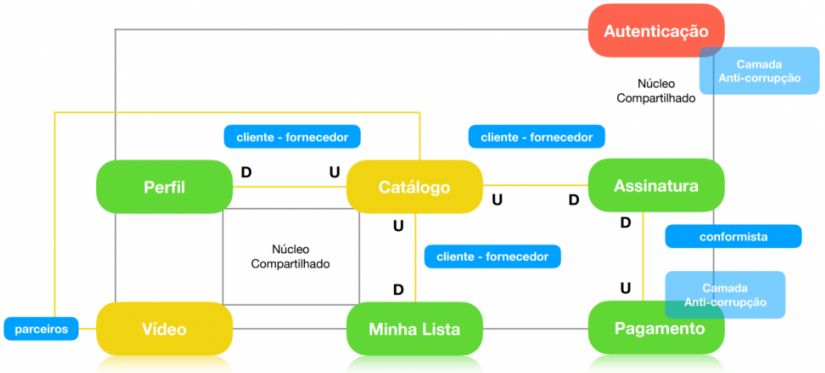
É um conjunto de princípios com foco em domínio, exploração de modelos de formas criativas e definir e falar a linguagem Ubíqua, baseado no contexto delimitado.”

Precisa de:
Comunicação entre os envolvidos no projeto.

Feito para:
Garantir as necessidades do domínio.
Evitar o risco de não atingir a expectativa.
Para tornar os projetos mais eficientes.
Entender de forma completa as necessidades.

Como fazer:
Se dedicar e estudar o domínio, o problema e motivo do site.
A ideia é centralizar o conhecimento para a criação desses modelos de domínio.
(o domínio é um conjunto de ideias, informações do sistema)

Pilares:
linguagem ubíqua - falada no cotidiano de uma empresa
bounded contexts (**contextos delimitados**) - delimita os contextos da aplicação. Levantamento do escopo do projeto.
Context Map (**Mapa de contexto**)- mapeamento do contexto,



(1) o foco está no domínio do sistema;
(2) desenvolvedores e especialistas no negócio devem explorar esse domínio de forma colaborativa;
(3) Como resultado, eles devem se comunicar usando uma linguagem ubíqua, mas dentro de um contexto delimitado.

Curiosidade:
O termo DDD ganhou popularidade com o livro “Domain-Driven Design: tackling complexity in the heart of software”, do consultor Eric Evans, publicado em 2003.