METODOLOGIAS ÁGEIS - BDD

METODOLOGIAS ÁGEIS:
Os métodos ágeis de desenvolvimento de software surgiram como uma alternativa às abordagens tradicionais, com o intuito de despender menos tempo com documentação e mais com a resolução de problemas de forma iterativa e interativa.

DDD: Domain-Driven Design (DDD) representa uma forma de desenvolver software em que o processo de design de uma aplicação é guiado pelo modelo de domínio. Em conjunto com a abordagem de desenvolvimento DDD, a existência de testes durante a implementação de uma aplicação é necessária para garantir a qualidade.

Linguagem oblíqua: Uma das principais características em DDD é a Linguagem Ubíqua. Essa linguagem comum entre todos os membros envolvidos (stakeholders) no projeto tem o objetivo de associar todas as atividades relacionadas ao desenvolvimento de software, de modo a facilitar a comunicação e o entendimento em relação ao domínio.

Práticas: Os procedimentos existentes no DDD também se aplicam para casos em que os analistas e desenvolvedores possuem maior conhecimento sobre o domínio e refatorar o código e suas entidades conforme a aplicação é implementada e, com isso, práticas de Test-Driven Development e refatoração são complementares às práticas propostas pelo DDD

Conclusão das metodologias: Percebe-se que no mercado de desenvolvimento de software atender aos requisitos do cliente e às constantes mudanças desses requisitos é essencial. Por isso se faz cada vez mais necessária a utilização de práticas ágeis de desenvolvimento, como o DDD e de testes como o TDD e o BDD.

METODOLOGIA TDD - TEST-DRIVEN DEVELOPMENT

O Test-Driven Development (TDD), ou Desenvolvimento Orientado a Testes, é uma abordagem no desenvolvimento de software que coloca o foco na criação de testes automatizados antes da implementação do código real. Essa metodologia é parte integrante das práticas ágeis e visa aprimorar a qualidade do código, promover a simplicidade e aumentar a eficiência no processo de desenvolvimento.
Ciclo TDD:

Red (Fase Vermelha): Nesta etapa, você escreve um teste automatizado que captura um aspecto específico do comportamento desejado do software. Esse teste inicialmente falhará, representando que o código de produção ainda não foi implementado.
Green (Fase Verde): Aqui, você escreve o código de produção mínimo necessário para passar no teste criado na etapa anterior. O objetivo é fazer o teste passar com a implementação mais simples possível.

Refactor (Fase de Refatoração): Uma vez que o teste passou, é o momento de refinar o código, melhorando sua qualidade sem alterar seu comportamento externo. Isso ajuda a manter o código limpo e sustentável ao longo do tempo.
Benefícios do TDD:

Código Confiável: O TDD resulta em uma suíte abrangente de testes automatizados, o que significa que os desenvolvedores têm mais confiança de que as mudanças no código não introduzem erros.
Design Incremental: O foco na criação de testes primeiro pode levar a um design mais modular e componentes mais bem definidos, já que os testes impulsionam a definição clara das funcionalidades.

Documentação Viva: Os testes servem como documentação viva do comportamento esperado do software, facilitando a compreensão para os membros da equipe.
Feedback Rápido: O ciclo curto do TDD permite receber feedback rapidamente, o que ajuda a identificar e corrigir problemas mais cedo no processo.
Mudanças Mais Fáceis: Como o código é testado continuamente, é mais fácil fazer alterações e refatorações sem medo de causar problemas inadvertidos.
Integração com DDD:

O TDD se alinha bem com a metodologia de Domain-Driven Design (DDD). Testes são essenciais para garantir que o modelo de domínio esteja sendo implementado corretamente e que as regras de negócio sejam cumpridas. A prática de escrever testes que refletem a linguagem ubíqua do DDD ajuda a manter a clareza e a consistência entre o código e o domínio.
Conclusão:

O Test-Driven Development é uma abordagem poderosa que promove a qualidade, confiabilidade e manutenibilidade do software. Ao combinar TDD com práticas como o Domain-Driven Design (DDD), os desenvolvedores podem criar aplicações que atendam às necessidades dos clientes de forma ágil e eficaz, garantindo a entrega de um código de qualidade e funcionalidades alinhadas ao domínio do problema.