Domain Driven Design

É uma abordagem de design de software disciplinada que reune um conjunto de conceitos, técnicas e princípios para construção de softwares baseados em um modelo de domínio. Domínio é todo e qualquer conhecimento utilizado em uma determinada area. Ele veio do titulo do livro escrito por Eric Evans, dono da DomainLanguage, uma empresa especializada em treinamento e consultoria para desenvolvimento de software. O livro de Evans é um grande catálogo de padrões, baseados em experiencias do autor ao longo de mais de 20 anos desenvolvendo software utilizando técnicas de Orientação a Objetos.

Principals conceitos do DDD

- Alinhamento do código corn o negócio: o contato dos desenvolvedores corn os especialistas do domínio é algo essencial quando se faz DDD (o pessoal de métodos ágeis já sabe disso faz tempo). Se faz necessário o uso de uma linguagem ubíqua (comum entre todos) para descrever o domínio e suas regras.

• Favorecer reutilização: os blocos de construção, que veremos adiante, facilitam aproveitar urn mesmo conceito de domínio ou um mesmo código em vários lugares.

- Mínimo de acoplamento: com um modelo bem feito, organizado, as várias partes de um sistema interagem sem que haja muita dependência entre módulos ou classes de objetos de conceitos distintos.

- Independência da Tecnologia: DDD não foca em tecnologia, mas sim em entender as regras de negócio e como elas devem estar refletidas no código e no modelo de domínio. Não que a tecnologia usada não seja importante, mas essa não é uma preocupação do DDD.

Criando um modelo de domínio (MDD)

A ideia por trás do MDD é a de que o seu modelo abstrato deve ser uma representação perfeita do seu domínio. Tudo que existe no seu negócio deve aparecer no modelo. Só aparece no modelo aquilo que está no negócio. O desenho do modelo é criado em conjunto entre especialistas de negócio e domínio, analistas, arquitetos e desenvolvedores, utilizando a linguagem ubíqua para que todos tenham o mesmo entendimento do domínio. O processo de maturação de um sistema desenvolvido usando MDD deve ser contínuo. O modelo serve de guia para a criação do código e, ao mesmo tempo, o código ajuda a aperfeiçoar o modelo.

Blocos de construção do MDD

Uma vez que decidimos criar urn modelo usando MDD, precisamos, inicialmente, isolar o modelo de domínio das demais partes que compõem o sistema. Essa separação pode ser feita utilizando-se uma arquitetura em camadas, que dividira nossa aplicação em quatro partes:

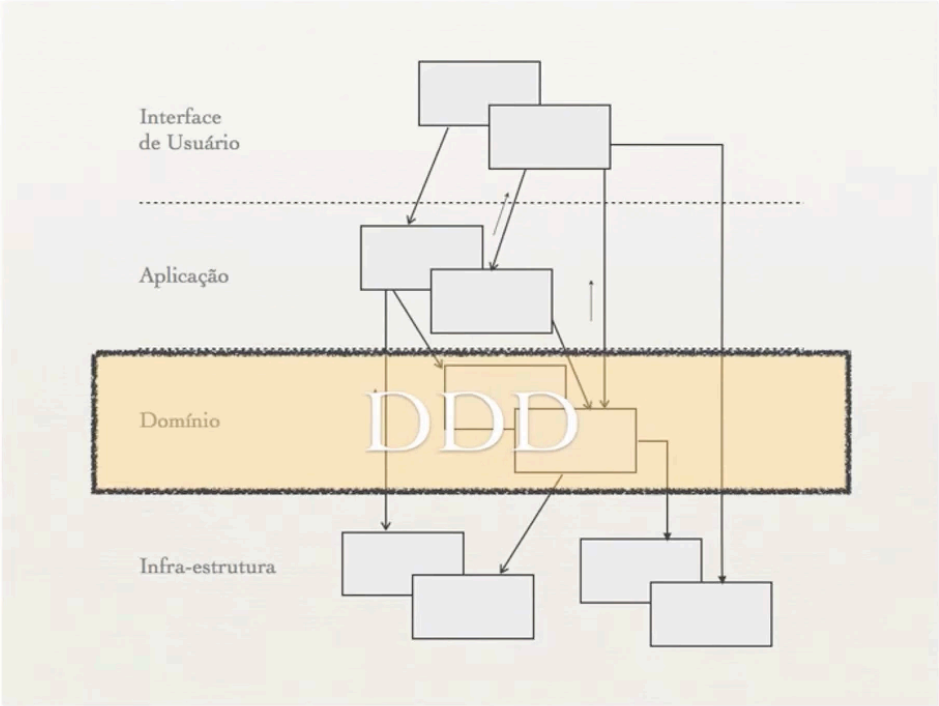
- Interface de Usuário — parte responsável pela exibição de informações do sistema ao usuário e também por interpreter comandos do usuário.

- Aplicação — essa camada não possui lógica de negócio. Ela é apenas uma camada fina, responsável por conectar a Interface de usuário às camadas inferiores.

- Domínio — representa os conceitos, regras e lógicas de negócio. Todo o foco do DDD está nessa camada. Nosso trabalho, daqui para frente, sera aperfeiçoar e compreender profundamente essa parte.

• Infra-estrutura — fornece recursos técnicos que darão suporte às camadas superiores. São normalmente as partes de urn sistema responsáveis por persistência de dados, conexões corn bancos de dados, envio de mensagens por redes, gravação e leitura de discos, etc.

Arquitetura padrão do MDD



Regras para modelagem do Domínio

- Entidades - classes de objetos que necessitam de uma identidade. Normalmente são elementos do domínio que possuem ciclo de vida dentro de nossa aplicação: um Cliente, por exemplo, se cadastra no sistema, faz compras, se torna inativo, é excluido, etc.

- Objetos de Valores - objetos que só carregam valores, mas que não possuem distinção de identidade. Bons exemplos de objetos de valores seriam: strings, números ou cores. Por exemplo: se o lápis de cor da criança acabar e voce der um novo lápis a ela, da mesma cor, só que de outra caixa, ela não vai se importar. Para a criança, o lápis vermelho de uma caixa é igual ao lápis vermelho de outra caixa. As instâncias de objetos de valores são imutáveis, isto é, uma vez criados, seus atributos internos não poderão mais ser modificados.

- Agregados - compostos de entidades ou objetos de valores que são encapsulados numa única classe. O agregado serve para manter a integridade do modelo. Elegemos uma classe para servir de raiz do agregado. Quando algum cliente quiser manipular dados de uma das classes que compõem o agregado, essa manipulação só poderá ser feita através da raiz.

- Fábricas — classes responsáveis pelo processo de criação dos Agregados ou dos Objetos de Valores. Algumas vezes, Agregados são relativamente complexos e não queremos manter a lógica de criação desses Agregados nas classes que os compõem. Extraímos então as regras de criação para uma classe externa: a fábrica.

- Serviços — classes que contêm lógica de negocio, mas que não pertence a nenhuma Entidade ou Objetos de Valores. É importante ressaltar que Serviços não guardam estado, ou seja, toda chamada a um mesmo serviço, dada uma mesma pré-condição, deve retornar sempre o mesmo resultado.

- Repositórios — classes responsáveis por administrar o ciclo de vida dos outros objetos, normalmente Entidades, Objetos de Valor e Agregados. Os repositórios são classes que centralizam operações de criação, alteração e remoção de objetos.

- Módulos — abstrações que têm por objetivo agrupar classes por um determinado conceito do domínio. A maioria das linguagens de programação oferecem suporte a módulos (pacotes em Java, namespaces em .NET ou módulos em Ruby).