Resenha do Documento "Domain-Driven Design Reference"

Flávio de Souza Ferreira Junior

Setembro de 2025

Resumo

Esta resenha, escrita sob a perspectiva de um desenvolvedor júnior, explora os conceitos fundamentais apresentados na "Domain-Driven Design Reference"de Eric Evans (2015). O foco está nos padrões que mais impactam a comunicação e a modelagem tática: a Linguagem Ubíqua, o Contexto Delimitado e a separação entre Entidades e Objetos de Valor.

1 Introdução: O Desafio da Complexidade

Que artigo interessante! Eu tinha ouvido falar de **DDD (Domain-Driven Design)**, mas ler o "Domain-Driven Design Reference"do Eric Evans é como se eu tivesse acabado de ganhar um mapa para sair da bagunça no código. Antes, eu jogava tudo numa classe só ou usava termos que só a gente do desenvolvimento entendia. Agora, a coisa é séria, é sobre o **domínio** do negócio. O DDD foca em três pontos: focar no domínio principal, explorar modelos em colaboração e falar uma linguagem ubíqua em um contexto delimitado.

2 Os Pilares Estratégicos: Comunicação e Limites

Pra mim, as duas ideias mais importantes que o DDD traz, e que são um desafio *muito* legal de implementar, são a Linguagem Ubíqua e o Contexto Delimitado.

2.1 1. A Magia da Linguagem Ubíqua

A ideia da **Linguagem Ubíqua** (*Ubiquitous Language*) é simplesmente genial. O Evans fala que o modelo do domínio tem que ser o *backbone* da linguagem que TODO MUNDO usa: os especialistas de negócio, o time de QA e nós, desenvolvedores. Se o termo no código (nome da classe, método) não é o mesmo que o especialista de domínio usa, o modelo está errado. Essa mudança de *mindset* garante que o código *expresse* o domínio de forma literal, sendo a melhor defesa contra a confusão na comunicação.

2.2 2. Contextos Delimitados (*Bounded Context*)

O conceito de **Contexto Delimitado** (*Bounded Context*) me deu um alívio. O artigo deixa claro que em projetos grandes, ter um *único* modelo para tudo é impossível e causa instabilidade e dificuldade de compreensão.

O Contexto Delimitado é a fronteira lógica. É o subsistema onde um modelo específico é válido e aplicável. Por exemplo, o conceito de "Cliente"pode ser diferente no Contexto de "Vendas"e no Contexto de "Suporte". Essa separação permite que os modelos sejam puros e se mantenham **coerentes** sem serem atrapalhados por questões externas.

3 Os Blocos Táticos: Estruturando o Código

Na parte de blocos de construção (*Building Blocks*), a distinção é crucial:

3.1 Entidades (*Entities*) e Objetos de Valor (*Value Objects*)

- **Entity (Entidade):** É o objeto que tem uma **identidade** e um ciclo de vida. Dois Clientes são diferentes, mesmo que tenham o mesmo nome ou dados, por causa da sua identidade.
- **Value Object (Objeto de Valor):** É o objeto que só é definido por seus **atributos**. Não tem identidade. Uma "Data de Nascimento"ou um "Endereço"são iguais se tiverem os mesmos valores. O Evans enfatiza que eles devem ser **imutáveis** (*immutable*), o que previne efeitos colaterais e simplifica o design.

3.2 Agregados (*Aggregates*)

Este é o chefe que garante a consistência. O Agregado tem uma **Raiz (Root)** que é a única forma de acessar os outros objetos dentro dele. Se você quer mudar algo, você tem que passar pelo chefe, o que impede que a gente quebre as regras de negócio sem querer.

4 Conclusão

O DDD não é só sobre arquitetura, é sobre linguagem e foco no que realmente importa, que é o **Core Domain** (Domínio Principal). Agora a meta é aplicar esses conceitos e refatorar o código para que ele reflita um *deep model* (modelo profundo) do negócio!