Resenha – Domain-Driven Design Reference: Definitions and Pattern Summaries

O *Domain-Driven Design Reference*, do Eric Evans, é quase um dicionário de bolso do DDD. Não é daqueles textos cheios de teoria pesada, mas sim um guia prático para quem precisa lidar com sistemas complexos sem se perder no meio do caminho.

A primeira ideia que Evans bate na tecla é a linguagem ubíqua: todo mundo do time — devs, analistas, gente de negócio — precisa falar a mesma língua. Isso significa que os nomes usados no código, nas reuniões e na documentação devem bater. Se o negócio fala “Pedido”, o código também fala “Pedido”, e não “OrderEntityV2”.

Depois ele apresenta os blocos de construção que sustentam o modelo:

- Entidades, aquelas que têm identidade própria (um cliente continua sendo o mesmo mesmo que mude de endereço).
- Value Objects, que são definidos só pelos atributos (como uma moeda ou um endereço).
- Aggregates, que agrupam entidades e valores para manter consistência.
- Repositórios, que funcionam como coleções inteligentes para acessar agregados.
- E ainda Serviços de Domínio e Eventos de Domínio, que dão conta do que não cabe naturalmente nos outros blocos.

Outro ponto forte do artigo é o conceito de Bounded Contexts — ou seja, delimitar onde um modelo vale e onde ele não vale. Isso evita confusões clássicas, tipo quando “Cliente” significa uma coisa para o time de vendas e outra para o time de suporte. Entre esses contextos, Evans propõe formas de integração, como criar camadas anticorrupção para não deixar um modelo poluir o outro.

Por fim, ele lembra que nem tudo tem o mesmo peso: existe o Core Domain (o que diferencia seu sistema no mercado e merece mais investimento), os Subdomínios de Suporte (importantes, mas não únicos) e os Subdomínios Genéricos (coisas comuns que você pode até usar pronto, tipo autenticação).

A lição geral é simples e poderosa: DDD é sobre manter o software vivo e alinhado com o negócio. Não basta desenhar bonito no início, é preciso revisar e refatorar sempre que o entendimento do domínio mudar. Assim você evita que o sistema vire um Frankenstein impossível de manter.