Artigo 7 - Strategic Design and Domain-Driven Design

Nome do estudante: João Victor Filardi Souza Pinto

O texto *Domain-Driven Design Reference*, de Eric Evans, sintetiza os principais conceitos do *Domain-Driven Design* (DDD), uma abordagem arquitetural voltada para alinhar o desenvolvimento de software com as necessidades reais do domínio de negócio. Evans parte da premissa de que muitos problemas enfrentados por sistemas complexos não estão apenas na tecnologia, mas na dificuldade de traduzir de forma precisa e sustentável o conhecimento do negócio em código. O DDD surge, portanto, como uma maneira de estruturar sistemas a partir de um modelo de domínio rico, consistente e compartilhado entre especialistas do negócio e desenvolvedores.

Um dos pontos centrais apresentados no texto é a importância da linguagem ubíqua (*ubiquitous language*). Essa linguagem comum deve ser construída em conjunto entre técnicos e especialistas do domínio, de modo que todos utilizem os mesmos termos e conceitos para se referirem a processos, regras e entidades do negócio. Isso reduz ambiguidades, melhora a comunicação e permite que o código reflita de forma fiel o domínio que ele busca representar.

Outro conceito essencial é a divisão do sistema em bounded contexts. Como os domínios podem ser muito amplos e complexos, Evans argumenta que é necessário delimitar contextos bem definidos, dentro dos quais a linguagem ubíqua se mantém coesa e aplicável. A interação entre diferentes contextos deve ser cuidadosamente mapeada por meio de contratos claros, evitando confusões e inconsistências. Essa ideia contribui diretamente para reduzir o acoplamento entre módulos e para possibilitar a evolução independente de diferentes partes do sistema.

O texto também descreve elementos táticos do DDD, como entidades, objetos de valor, agregados, repositórios e serviços de domínio. Cada um desses elementos tem um papel específico na modelagem. As entidades representam objetos com identidade única, enquanto objetos de valor expressam atributos imutáveis que caracterizam propriedades de negócio. Agregados agrupam entidades e valores em torno de uma raiz, definindo limites claros de consistência. Os repositórios cuidam da persistência, fornecendo acesso controlado às coleções de agregados, e os serviços de domínio representam operações que não pertencem a nenhuma entidade específica, mas fazem parte do núcleo das regras do negócio.

Do ponto de vista prático, os conceitos de Evans se aplicam de forma clara em projetos de mercado. Imagine, por exemplo, uma fintech que desenvolve uma plataforma de crédito digital. O domínio envolve clientes, propostas de empréstimo, análise de risco e contratos. Usando DDD, seria possível criar uma linguagem ubíqua que defina de forma clara o que significa “proposta ativa”, “risco aprovado” ou “contrato fechado”. Além disso, delimitando *bounded contexts*, a empresa poderia separar a parte de análise de crédito do módulo de pagamento, permitindo que equipes diferentes trabalhem em paralelo sem gerar confusão nos conceitos. O uso de agregados garantiria consistência em processos críticos, como a aprovação de um empréstimo, enquanto repositórios e serviços de domínio estruturariam de maneira limpa a persistência e as operações centrais do sistema.

Em resumo, Evans mostra que o DDD não é apenas um conjunto de padrões técnicos, mas uma filosofia de desenvolvimento orientada à colaboração e ao entendimento profundo do domínio. Ao colocar o negócio no centro da modelagem, o DDD promove sistemas mais compreensíveis, sustentáveis e alinhados às reais necessidades dos clientes. Essa abordagem se mantém extremamente relevante no mercado atual, especialmente em sistemas complexos e distribuídos, nos quais clareza conceitual e flexibilidade arquitetural são diferenciais estratégicos.