Domain Driven Design Reference - Definitions and Pattern Summaries

Capítulos 4, 5 e 6

A estratégia de design em sistemas complexos exige um foco meticuloso que se estende para além do código individual. O primeiro passo fundamental é o Mapeamento de Contexto, onde se define o limite de aplicação de cada modelo de domínio — o Bounded Context. É vital ir além do isolamento e analisar as relações entre esses contextos. A escolha do padrão de integração é uma decisão de negócio e de organização: pode-se optar por uma colaboração de alto risco (Partnership), onde as equipes triunfam ou falham juntas, ou uma relação *upstream-downstream* mais comum. Nesses casos, o time cliente (downstream) pode exercer influência (Customer/Supplier) ou simplesmente se render ao modelo do fornecedor (Conformist). Para evitar que um modelo externo e inferior corrompa o sistema local, o uso da Anticorruption Layer torna-se indispensável, atuando como um tradutor defensivo. Quando o custo da integração supera o benefício, a melhor decisão é Separate Ways.

Essa gestão de fronteiras e relações é complementada pela Destilação do Domínio, uma estratégia para concentrar o investimento. Em todo sistema, o valor real reside no Core Domain (Domínio Central), a parte única e especializada que diferencia o negócio. A principal ação estratégica é separar esse núcleo dos Generic Subdomains (Subdomínios Genéricos) e das funcionalidades de suporte. Os recursos mais valiosos da engenharia devem ser direcionados para desenvolver uma modelagem profunda e flexível no Core Domain. Estruturalmente, isso se traduz em refatorar o código para criar um Segregated Core, onde os conceitos centrais são isolados, coesos e protegidos de acoplamentos desnecessários com a periferia do sistema, garantindo clareza e manutenibilidade.

Finalmente, para que o sistema mantenha a coerência em grande escala, é necessário estabelecer uma Estrutura de Larga Escala. Este é um princípio arquitetural que fornece uma linguagem e um guia para a organização de todo o projeto. Adotar conceitos como Responsibility Layers (Camadas de Responsabilidade) permite que desenvolvedores entendam o papel de sua parte no todo, sem precisar de conhecimento detalhado dos demais componentes. Essa estrutura, que pode começar com uma System Metaphor, deve ser tratada não como uma regra rígida imposta de antemão, mas sim como um guia que evolui com o sistema, adaptando-se às descobertas do desenvolvimento e mantendo a clareza conceitual.