Resenha “Strategic Design and Domain-Driven Design” (Capítulos 4, 5 e 6) - Pedro Henrique Maia Alves

O artigo começa apresentando os fundamentos do *Strategic Design* dentro do *Domain-Driven Design (DDD)*, destacando como a organização de contextos e a clareza de limites conceituais são essenciais para lidar com a complexidade de sistemas grandes. Os capítulos 4, 5 e 6 exploram especificamente as noções de Context Maps, Relationships entre Contextos e Organização de Times e Sistemas. O ponto central é que, para manter a evolução saudável de um software, não basta modelar bem as entidades dentro de um domínio; é necessário também entender como diferentes domínios e equipes interagem.

Por que esses problemas surgem? Projetos de grande porte dificilmente são construídos por um único time ou em um único domínio de negócio. À medida que a organização cresce, surgem vários Bounded Contexts, cada um refletindo diferentes subdomínios. Se esses contextos não são bem definidos e se as relações entre eles não são claras, o software se torna inconsistente, confuso e difícil de evoluir. Portanto, o *Strategic Design* surge como resposta à necessidade de organizar essa complexidade em estruturas mais previsíveis e coordenadas.

Algumas características perceptíveis em sistemas que aplicam o *Strategic Design*, segundo os capítulos 4, 5 e 6, são:

- **Bounded Contexts bem definidos** – Cada contexto possui limites claros de responsabilidade, vocabulário e modelo conceitual.

- **Context Maps documentados** – Diagramas e descrições que explicitam como os diferentes contextos se relacionam.

- **Linguagem Ubíqua preservada por contexto** – Evita ambiguidades, já que o mesmo termo pode ter significados diferentes em domínios distintos.

- **Relacionamentos explícitos entre contextos** – Parcerias, conformidades ou integrações são bem estabelecidas, reduzindo mal-entendidos.

- **Organização alinhada com times** – Estrutura técnica reflete a divisão organizacional, reforçando a autonomia de cada equipe.

- **Coordenação estratégica** – Decisões sobre dependências e integrações são tomadas conscientemente, em vez de surgirem de forma improvisada.

Nos capítulos, o artigo apresenta padrões de relacionamentos entre contextos. Por exemplo:

- **Shared Kernel**, onde dois contextos compartilham um pequeno núcleo comum, exigindo alta colaboração.

- **Customer-Supplier**, no qual um contexto depende das decisões de outro, demandando comunicação constante.

- **Conformist**, quando um contexto simplesmente adota o modelo de outro sem contestação.

- **Anticorruption Layer**, utilizado para proteger um contexto de modelos externos, criando uma barreira de tradução.
  Esses padrões ajudam a lidar com as pressões reais de integração entre diferentes partes de um sistema.

Algumas estratégias são utilizadas para lidar com a aplicação do *Strategic Design*. Mapear todos os contextos e documentar suas interações evita dependências invisíveis. Estabelecer contratos claros entre equipes reduz conflitos. Aplicar camadas de anticorrupção permite preservar a integridade de um domínio central, mesmo quando é necessário integrar sistemas legados. Além disso, alinhar a organização de times com os limites dos contextos reforça a autonomia e a clareza na tomada de decisões.

O artigo conclui que os princípios de *Strategic Design* são essenciais para sistemas de grande porte, pois garantem consistência conceitual e previsibilidade de evolução. Enquanto o *tactical design* do DDD foca nos detalhes de modelagem dentro de um contexto, o *strategic design* oferece a visão macro, necessária para coordenar equipes e sistemas inteiros. Nos capítulos 4, 5 e 6, fica evidente que ignorar a definição de contextos e suas relações leva inevitavelmente à confusão e à lentidão no desenvolvimento.

No dia a dia do mercado, acredito que aplicar *Bounded Contexts* bem delimitados, mapear explicitamente as interações e adotar padrões como *Anticorruption Layer* e *Customer-Supplier* são fundamentais para manter a saúde de um sistema complexo. Essa abordagem não só melhora a manutenção e evolução do software, mas também organiza o trabalho das equipes, permitindo que diferentes times colaborem de maneira mais eficiente e previsível.