Um Guia Consolidado para o Design Estratégico em DDD

Em projetos de software de grande escala, a complexidade não reside apenas no código, mas também na interação entre diferentes equipes, modelos de negócio e subsistemas. O Design Estratégico oferece um conjunto de ferramentas e padrões para visualizar, discutir e organizar essa complexidade em um nível elevado. Ele se baseia em três atividades principais: mapear o cenário dos diferentes modelos, destilar o que é mais importante e definir uma estrutura coerente para todo o sistema.

Antes de qualquer decisão de design profundo, é preciso entender o terreno. O Mapeamento de Contextos é a prática de identificar os diferentes modelos de domínio em jogo e as relações entre eles.

Conceitos Fundamentais

Bounded Context (Contexto Delimitado): Uma fronteira explícita (como um subsistema ou o trabalho de uma equipe) dentro da qual um modelo de domínio específico é consistente e aplicável. Fora dessa fronteira, o mesmo termo (ex: "Cliente") pode ter um significado completamente diferente.

Upstream-Downstream (Fluxo Acima-Abaixo): Uma relação de dependência onde as ações de uma equipe upstream (acima no fluxo) impactam a equipe downstream (abaixo no fluxo). A equipe upstream geralmente pode ter sucesso independentemente da downstream.

Padrões de Relacionamento entre Contextos

O Mapa de Contexto documenta as relações entre os Contextos Delimitados, que geralmente se enquadram nos seguintes padrões:

Partnership (Parceria): Quando duas equipes ou contextos dependem mutuamente um do outro para o sucesso. A solução é forjar uma parceria explícita com planejamento e gerenciamento conjuntos.

Shared Kernel (Núcleo Compartilhado): Para contextos com uma interdependência muito íntima, um subconjunto pequeno e explícito do modelo de domínio (e código) é compartilhado. As alterações neste núcleo exigem consulta mútua.

Customer/Supplier (Cliente/Fornecedor): Em uma relação upstream-downstream, formaliza-se a dinâmica. A equipe downstream atua como cliente, e suas prioridades são negociadas e incluídas no planejamento da equipe upstream (fornecedor).

Conformist (Conformista): Quando a equipe downstream não tem poder de negociação, ela opta por aderir estritamente ao modelo do upstream para simplificar a integração, mesmo que o modelo não seja ideal para suas necessidades.

Anticorruption Layer (Camada Anticorrupção - ACL): A equipe downstream constrói uma camada de software isolante que traduz o modelo do sistema upstream (especialmente se for legado ou mal projetado) para o seu próprio modelo, protegendo-o de influências externas.

Open-Host Service (Serviço de Host Aberto): O sistema upstream define e publica um protocolo de acesso bem documentado (API). Os sistemas downstream são responsáveis por se comunicar através deste serviço, geralmente usando uma ACL.

Published Language (Linguagem Publicada): A integração ocorre através de uma linguagem de intercâmbio de dados bem documentada e compartilhada (como JSON Schema, XML, Protobuf), desacoplando os modelos internos de cada contexto.

Separate Ways (Caminhos Separados): A melhor solução quando não há benefício real na integração. Os contextos são declarados como não tendo conexão, permitindo soluções simples e focadas em cada um.

Lidando com a Realidade: Big Ball of Mud (Grande Bola de Lama): Para sistemas existentes sem limites claros, a abordagem pragmática é traçar uma fronteira ao redor de toda a confusão, tratá-la como um único contexto e, o mais importante, impedir que sua desordem se espalhe para outros contextos.

Com o mapa do terreno claro, o próximo passo é decidir onde concentrar os esforços. A Destilação é o processo de separar a essência do modelo de domínio das partes de suporte para maximizar o valor do negócio.

Identificando as Peças

Core Domain (Domínio Principal): A parte do sistema que contém o diferencial competitivo e a lógica de negócio mais complexa e valiosa. É aqui que os melhores talentos devem ser aplicados, e o design deve ser profundo e flexível.

Generic Subdomains (Subdomínios Genéricos): Partes do sistema que são necessárias, mas não são um diferencial (ex: autenticação, envio de notificações). A recomendação é dar-lhes baixa prioridade, usar soluções prontas sempre que possível e não alocar os principais especialistas de domínio para eles.

Técnicas para Clarificar e Proteger o Núcleo

Domain Vision Statement (Declaração de Visão do Domínio): Um documento curto e de alto nível que descreve a proposta de valor do Core Domain, alinhando a equipe no início do projeto.

Highlighted Core (Núcleo Destacado): Tornar o Core Domain explícito, seja através de um documento de design mais detalhado ou, mais praticamente, sinalizando os artefatos de código que pertencem ao núcleo.

Segregated Core (Núcleo Segregado): Uma refatoração estrutural para mover o código do Core Domain para seus próprios módulos, separando-o fisicamente do código de suporte para fortalecer sua coesão e reduzir o acoplamento.

Cohesive Mechanisms (Mecanismos Coesos): Fatorar algoritmos complexos e genéricos (o "como") em frameworks leves e separados, deixando o modelo de domínio focado em expressar as regras de negócio (o "o quê").

Abstract Core (Núcleo Abstrato): Criar um pequeno modelo de interfaces e classes abstratas que expressa os conceitos fundamentais do Core Domain, permitindo que outros módulos dependam dessa abstração estável em vez das implementações detalhadas e voláteis.

Uma vez que o núcleo está focado, como garantimos que todo o sistema, em sua totalidade, tenha uma forma coerente? A Estrutura em Larga Escala é um princípio organizador que permite que todos entendam o "quadro geral".

O Princípio Orientador: Evolving Order (Ordem em Evolução)

A estrutura de larga escala não deve ser uma camisa de força imposta desde o início. A abordagem mais eficaz é permitir que ela evolua com a aplicação. Uma estrutura deve ser adotada quando ela emerge naturalmente do modelo e ajuda a esclarecer o design, não a restringi-lo. "Menos é mais" é a filosofia chave.

Padrões de Estrutura em Larga Escala

System Metaphor (Metáfora do Sistema): Organizar o design em torno de uma analogia concreta e compartilhada que ajuda a equipe a raciocinar sobre o sistema como um todo. A metáfora se torna parte da linguagem ubíqua, mas deve ser reavaliada constantemente.

Responsibility Layers (Camadas de Responsabilidade): Estruturar o modelo em camadas com base em responsabilidades conceituais e taxas de mudança. Cada camada (ex: Apresentação, Domínio, Infraestrutura) conta uma parte da história do sistema, e os objetos se encaixam claramente em uma delas.

Knowledge Level (Nível de Conhecimento): Separar o modelo em dois níveis: um nível concreto com os objetos operacionais e um "nível de conhecimento" (meta-nível) que contém objetos que descrevem e restringem o comportamento

do primeiro nível. Isso permite alta customização sem alterar o código do núcleo.

Pluggable Component Framework (Framework de Componentes Conectáveis): Um padrão avançado para domínios maduros. Consiste em destilar um núcleo abstrato de interfaces e criar um framework que permite que diferentes implementações concretas sejam "conectadas" e substituídas, promovendo um ecossistema de ferramentas interoperáveis.