O capítulo 4, "Context Mapping for Strategic Design", introduz uma verdade fundamental e muitas vezes ignorada em grandes projetos: não existe um único modelo unificado. Tentar forçar um único modelo sobre toda a organização é uma receita para o fracasso. Em vez disso, a realidade é um ecossistema de múltiplos modelos, cada um válido dentro de seu próprio **Contexto Delimitado (Bounded Context)**. O desafio, então, não é eliminar essa diversidade, mas gerenciá-la.

A ferramenta principal para isso é o **Mapa de Contextos (Context Map)**. Este não é um diagrama técnico formal, mas sim um artefato estratégico que visualiza a paisagem dos diferentes contextos e, mais importante, as relações entre eles. O capítulo detalha um vocabulário de padrões para descrever essas relações, que são, na essência, padrões de colaboração e integração entre equipes.

- **Colaboração Estreita:** Padrões como **Parceria (Partnership)** e **Núcleo Compartilhado (Shared Kernel)** descrevem cenários onde duas equipes devem colaborar intensamente, pois seu sucesso ou fracasso estão intrinsecamente ligados. Eles exigem um alto grau de comunicação e integração contínua.
- **Dinâmicas de Poder (Upstream/Downstream):** A relação **Cliente/Fornecedor (Customer/Supplier)** formaliza uma dinâmica saudável onde a equipe a jusante (downstream) tem influência sobre as prioridades da equipe a montante (upstream). Em contraste, o padrão **Conformista (Conformist)** descreve uma situação onde a equipe a jusante simplesmente adere ao modelo da equipe a montante, sem poder de negociação. Para se proteger de um modelo a montante problemático ou instável, a equipe a jusante pode construir uma **Camada Anticorrupção (Anticorruption Layer)**, que atua como um tradutor defensivo, isolando e protegendo seu próprio modelo.
- **Exposição e Desacoplamento:** Quando um sistema precisa servir múltiplos clientes, o **Serviço de Host Aberto (Open-host Service)**, combinado com uma **Linguagem Publicada (Published Language)**, oferece uma maneira de expor suas capacidades através de um protocolo bem definido, permitindo que as equipes clientes se integrem de forma autônoma.

Em suma, o Mapeamento de Contextos fornece as ferramentas para entender a realidade política e técnica de um projeto, permitindo que as equipes tomem decisões informadas sobre como e quando colaborar, em vez de tropeçar em integrações mal definidas.

Uma vez que o território é mapeado, o próximo passo estratégico é decidir onde concentrar os esforços mais valiosos. O capítulo 5, "Distillation for Strategic Design", trata exatamente disso: separar o que é essencial do que é meramente suporte. Em qualquer sistema grande, nem todas as partes são igualmente importantes para o sucesso do negócio. A destilação é o processo de identificar e priorizar o coração do software.

O conceito central aqui é o **Domínio Principal (Core Domain)**. Este não é apenas o subsistema mais complexo, mas sim a área que contém a vantagem competitiva única do negócio. É o "molho secreto", a razão de ser do software. O DDD defende que os melhores talentos e os maiores esforços de modelagem devem ser investidos aqui. A meta é desenvolver um modelo profundo e um design flexível (**Supple Design**) para o Domínio Principal, pois é ele que gera o maior valor.

Para tornar o Domínio Principal mais claro, é preciso separá-lo do ruído. Isso é feito identificando **Subdomínios Genéricos (Generic Subdomains)**. São partes do sistema que são necessárias, mas não são um diferencial competitivo (por exemplo, um módulo de envio de e-mails ou um sistema de contabilidade padrão). A estratégia para subdomínios genéricos é simples: não os construa do zero se puder evitar. Compre uma solução pronta ou use uma estrutura existente. Isso libera recursos preciosos para focar no que realmente importa.

A destilação é, portanto, um ato de priorização implacável. Ela garante que a complexidade seja gerenciada de forma estratégica, aplicando o máximo esforço onde o retorno é maior e evitando a armadilha de tratar todas as partes do sistema com a mesma importância.

Com os contextos mapeados e o núcleo destilado, o desafio final é garantir que o sistema como um todo permaneça coeso e compreensível à medida que cresce. O capítulo 6, "Large-scale Structure for Strategic Design", aborda a necessidade de um princípio organizador que transcenda os componentes individuais e forneça uma visão coerente do sistema. Uma estrutura em larga escala é uma linguagem que permite discutir e entender o sistema em traços largos.

O princípio orientador é a **Ordem Evolutiva (Evolving Order)**. Ao contrário de uma arquitetura rígida definida no início, essas estruturas devem emergir e evoluir com o sistema. A ideia é aplicar uma estrutura que esclareça o design sem restringi-lo indevidamente. "Menos é mais" é o lema.

O capítulo apresenta quatro padrões exemplares de estruturas de larga escala:

1. **Metáfora do Sistema (System Metaphor):** Popularizada pela Extreme Programming, esta estrutura organiza o design em torno de uma analogia concreta e tangível que a equipe compartilha. A metáfora ajuda a guiar o pensamento e a manter a consistência em todo o sistema.
2. **Camadas de Responsabilidade (Responsibility Layers):** Esta estrutura organiza o modelo em camadas conceituais com base nas taxas de mudança e dependências. Por exemplo, uma camada pode lidar com decisões operacionais de baixo nível, enquanto outra lida com políticas de alto nível que mudam com menos frequência. Isso ajuda a dar coerência à atribuição de responsabilidades em todo o modelo.
3. **Nível de Conhecimento (Knowledge Level):** Este padrão separa os objetos que representam a operação do dia-a-dia de um conjunto de objetos que descrevem e restringem seu comportamento (as "regras"). Essencialmente, cria um nível de metadados que pode ser configurado por um superusuário, tornando o sistema mais flexível.
4. **Estrutura de Componentes Conectáveis (Pluggable Component Framework):** A mais madura das estruturas, emerge quando um modelo de domínio está tão bem destilado que seu núcleo abstrato pode ser transformado em um framework. Isso permite que diferentes implementações e aplicações sejam "conectadas" a esse núcleo, promovendo a interoperabilidade e o reuso em um nível muito alto.