Os capítulos abordam os aspectos estratégicos do DDD que permitem lidar com a complexidade de grandes sistemas de forma estruturada, coerente e sustentável, indo além do nível técnico e alcançando a organização conceitual do negócio. O capítulo 4 trata do Context Mapping, um conceito que reconhece a realidade inevitável de múltiplos modelos coexistindo em sistemas extensos. À medida que equipes diferentes trabalham em partes distintas de um mesmo produto, cada uma desenvolve seu próprio entendimento e sua própria linguagem do domínio. Quando esses modelos se misturam sem limites bem definidos, o sistema tende a se tornar confuso, frágil e difícil de manter. Por isso, o autor propõe a definição de Bounded Contexts, que estabelecem fronteiras claras para cada modelo, e o uso de um Context Map, que ilustra visualmente como esses contextos se relacionam entre si. Padrões como Shared Kernel, Customer/Supplier, Conformist e Anticorruption Layer são apresentados como formas de gerenciar essas interações, evitando que decisões de um time contaminem o trabalho de outro. O capítulo também alerta para o risco do chamado Big Ball of Mud, quando não há qualquer organização e os módulos se tornam um emaranhado caótico. Assim, esse capítulo mostra que mapear contextos e relações reduz conflitos, aumenta a clareza organizacional e fortalece a comunicação entre equipes.

O capítulo 5 aprofunda a ideia de Distillation, que busca identificar e proteger aquilo que realmente é essencial no sistema: o Core Domain. Em projetos grandes, é comum que o núcleo do domínio, onde está o maior valor competitivo, acabe ofuscado por inúmeras funcionalidades secundárias. O texto destaca que, para entregar valor real ao negócio, as equipes devem concentrar seus esforços criativos e de design nesse núcleo, enquanto as partes genéricas podem ser tratadas como Generic Subdomains, usando soluções prontas ou até sendo terceirizadas. Para manter o foco, o autor propõe a criação de um Domain Vision Statement, uma declaração da visão do domínio que orienta todos os envolvidos, e estratégias como Highlighted Core, Segregated Core e Abstract Core, que ajudam a manter o núcleo protegido e isolado das mudanças periféricas. Esse capítulo ressalta que, ao destilar o que é essencial, a equipe consegue usar melhor seus recursos, manter o sistema competitivo e evitar a dispersão de esforços em funcionalidades que não agregam valor estratégico.

Já o capítulo 6 discute como estruturar o sistema em larga escala, garantindo que ele continue compreensível e organizado mesmo crescendo muito. O autor mostra que, à medida que o sistema se expande, é preciso definir uma estrutura de alto nível que sirva como um mapa conceitual, permitindo que os desenvolvedores compreendam o papel de cada parte dentro do todo. Para isso, são apresentados padrões como Evolving Order, que permite que a arquitetura evolua gradualmente sem rigidez excessiva, System Metaphor, que cria uma metáfora compartilhada para unificar a comunicação sobre o sistema, Responsibility Layers, que organizam as responsabilidades em camadas claras e previsíveis, Knowledge Level, que separa as regras de negócio das definições que as controlam, e Pluggable Component

Framework, que possibilita adicionar ou substituir componentes sem afetar a estrutura central. Ao adotar esse tipo de organização de grande escala, os times mantêm a coesão do sistema, evitam que ele se torne caótico e conseguem evoluí-lo de maneira coordenada, mesmo com múltiplas equipes e domínios envolvidos.

Um exemplo prático seria o desenvolvimento de uma plataforma de gestão agrícola com vários módulos (estoque, lavouras, maquinário, pecuária, vendas etc.) e diversas equipes trabalhando simultaneamente. Aplicando os ensinamentos do capítulo 4, a empresa poderia definir Bounded Contexts separados, como “Gestão de Lavouras” e “Gestão de Maquinário”, conectando-os por meio de um Context Map e usando um Anticorruption Layer para evitar que mudanças em um contexto afetem negativamente o outro. Pelo capítulo 5, identificaria “Gestão de Lavouras” como Core Domain, concentrando os melhores desenvolvedores e especialistas nesse módulo, enquanto trataria o módulo de estoque como Generic Subdomain, usando soluções prontas e de baixo custo. E seguindo o capítulo 6, a arquitetura geral seria organizada em Responsibility Layers, com uma metáfora central de “fazenda digital” (System Metaphor) para alinhar a comunicação entre os times, além de um framework de componentes plugáveis que permita adicionar novos módulos sem impactar o núcleo. Dessa forma, os três capítulos se complementam, mostrando como manter a integridade conceitual, priorizar o que realmente importa e garantir a evolução sustentável de sistemas complexos e de grande escala.