REVISTA TÓPICOS
ISSN 2965-6672 | Qualis A2

A IMPLEMENTAÇÃO DE DOMAIN-DRIVEN DESIGN (DDD): UMA INVESTIGAÇÃO EXPLORATÓRIA DAS PRÁTICAS, DESAFIOS E TENDÊNCIAS RECENTES

THE IMPLEMENTATION OF DOMAIN-DRIVEN DESIGN (DDD): AN EXPLORATORY INVESTIGATION OF PRACTICES, CHALLENGES, AND RECENT TRENDS

Ciências Exatas e da Terra, Engenharias • 19/08/2025

REGISTRO DOI: 10.5281/zenodo.16900064

Cláudio Filipe Lima Rapôso[1]

RESUMO

Este artigo apresenta uma pesquisa exploratória sobre a implementação prática do Domain-Driven Design (DDD) em contextos organizacionais diversos. Partindo de uma revisão teórica aprofundada e de uma análise sistemática da literatura científica e técnica, o estudo busca compreender como os princípios do DDD têm sido operacionalizados por equipes de desenvolvimento, quais práticas se consolidaram, que benefícios são frequentemente relatados e quais barreiras dificultam sua adoção plena. Os resultados revelam que o DDD, embora fundado em uma base conceitual robusta, é aplicado de forma variada e, por vezes, fragmentada, sofrendo adaptações metodológicas conforme as condições técnicas, culturais e estruturais das organizações. São destacadas práticas como a modelagem colaborativa do domínio, a definição de bounded contexts e o uso de eventos do domínio, bem como benefícios como clareza conceitual, modularidade arquitetural e alinhamento entre tecnologia e negócio. Por outro lado, são identificados desafios relacionados à complexidade conceitual, à escassez de capacitação e à resistência organizacional à colaboração interdisciplinar. A pesquisa contribui ao oferecer uma visão crítica e atualizada do uso do DDD e propõe direções para estudos futuros que possam ampliar a compreensão empírica sobre sua efetividade.

Palavras-chave: Domain-Driven Design, DDD, modelagem de domínio, desenvolvimento de software, pesquisa exploratória.

ABSTRACT

This article presents an exploratory study on the practical implementation of Domain-Driven Design (DDD) in various organizational contexts. Based on an in-depth theoretical review and a systematic analysis of scientific and technical literature, the research aims to understand how DDD principles have been applied

by software development teams, which practices have become established, what benefits are commonly reported, and what barriers hinder its full adoption. The findings show that although DDD is grounded in a robust conceptual foundation, it is often applied in diverse and sometimes fragmented ways, with methodological adaptations according to the technical, cultural, and structural conditions of each organization. Practices such as collaborative domain modeling, the definition of bounded contexts, and the use of domain events are highlighted, along with reported benefits including conceptual clarity, architectural modularity, and alignment between business and technology. On the other hand, challenges are identified related to conceptual complexity, lack of training, and organizational resistance to interdisciplinary collaboration. This study contributes by providing a critical and updated perspective on the use of DDD and proposes directions for future research to expand empirical understanding of its effectiveness.
Keywords: Domain-Driven Design, DDD, domain modeling, software development, exploratory research.

1 Introdução

Nas últimas duas décadas, a crescente complexidade dos sistemas de software e a necessidade de maior alinhamento entre áreas técnicas e de negócio têm impulsionado o surgimento e a consolidação de abordagens arquitetônicas que promovam uma modelagem centrada no domínio. Nesse contexto, o Domain-Driven Design (DDD), proposto por Eric Evans em 2003, tem se destacado como uma abordagem que busca integrar conhecimento do domínio ao processo de desenvolvimento, por meio da colaboração

entre especialistas de negócio e desenvolvedores, do uso de linguagem ubíqua e da delimitação clara de contextos (Evans, 2003).

A implementação de DDD vai além da aplicação de padrões técnicos; ela requer uma mudança cultural e metodológica profunda, exigindo uma reestruturação do processo de desenvolvimento de software com base na lógica do domínio de negócio. Tal complexidade é tanto uma força quanto um obstáculo: por um lado, DDD permite maior modularidade, clareza arquitetural e alinhamento estratégico; por outro, apresenta desafios significativos relacionados à sua adoção, como a ausência de ferramentas padronizadas, a dificuldade de aprendizado e a escassez de indicadores objetivos para avaliar seus impactos (Rapôso, 2025; Özkan, 2023).

Embora a literatura apresente diversas abordagens sobre os conceitos fundamentais do DDD e estudos de caso que exploram seus benefícios e limitações, ainda há uma lacuna significativa no que diz respeito à sistematização dos modos como essa abordagem tem sido efetivamente implementada em contextos reais de desenvolvimento. A maioria dos estudos disponíveis adota enfoques descritivos ou normativos, sem aprofundar de forma exploratória a diversidade de práticas, adaptações contextuais e obstáculos enfrentados durante a adoção da abordagem.

Este estudo adota uma abordagem metodológica de natureza qualitativa, orientada por princípios de pesquisa exploratória. O principal objetivo da pesquisa é compreender, a partir da análise de fontes científicas e técnicas, como o Domain-Driven Design (DDD) tem sido implementado em ambientes de desenvolvimento de software, considerando suas práticas, variações, desafios e

benefícios. A escolha por um delineamento exploratório justifica-se pela escassez de estudos empíricos consolidados sobre a aplicação prática do DDD, bem como pela natureza aberta da questão norteadora, que visa identificar padrões, levantar hipóteses e delinear caminhos para investigações futuras. O percurso metodológico estruturou-se em três etapas complementares: levantamento bibliográfico, análise de conteúdo e síntese interpretativa.

O corpus documental da pesquisa foi composto por artigos científicos, dissertações, teses, publicações técnicas e relatórios de casos, obtidos em bases de dados acadêmicas e técnicas reconhecidas incluindo IEEE Xplore, ACM Digital Library, Scopus, SpringerLink, ScienceDirect, Google Scholar e repositórios institucionais nacionais (como o Repositório Institucional da UFU e da UFSC). Os critérios de inclusão adotados foram: (i) publicações entre 2003 e 2025; (ii) foco explícito na implementação prática de DDD; (iii) presença de elementos analíticos ou descritivos relacionados a práticas, benefícios ou dificuldades da abordagem. Foram excluídos estudos exclusivamente teóricos ou que abordassem DDD de forma tangencial, sem foco em sua aplicação.

A seleção dos textos foi realizada por meio de busca por palavras-chave como “Domain-Driven Design implementation”, “DDD adoption”, “practical DDD”, “bounded contexts”, “linguagem ubíqua”, entre outras, em português e inglês. Após a filtragem inicial por títulos e resumos, os textos selecionados foram lidos integralmente para a composição do corpus final de análise.

A análise do material foi conduzida com base na técnica de análise de conteúdo, conforme proposta por Bardin (2011), especialmente

em sua vertente categorial temática. Inicialmente, foi realizada uma codificação aberta dos textos, identificando segmentos relevantes relacionados à implementação do DDD. Em seguida, os trechos foram organizados em categorias emergentes, agrupando-se práticas recorrentes, dificuldades relatadas, estratégias adaptativas, contextos de aplicação e variações metodológicas.

Essa análise permitiu a identificação de padrões e contradições presentes nos relatos estudados, bem como a elaboração de inferências sobre os fatores que influenciam a adoção bem-sucedida ou problemática do DDD em diferentes ambientes organizacionais e arquiteturais.

Com base na análise de conteúdo, foi elaborada uma síntese interpretativa que visa oferecer uma visão crítica e articulada das práticas identificadas. A interpretação foi orientada pela perspectiva de Evans (2003) sobre os fundamentos do DDD, e dialoga com as contribuições mais recentes da literatura internacional e nacional, permitindo compreender como os princípios originais têm sido reinterpretados, adaptados ou confrontados em contextos contemporâneos, como arquiteturas baseadas em microserviços ou paradigmas funcionais.

Essa etapa final culminou na construção das categorias analíticas que estruturam o desenvolvimento do artigo, permitindo não apenas descrever os achados, mas também refletir sobre suas implicações metodológicas, organizacionais e técnicas.

A relevância deste estudo reside na sua capacidade de fornecer uma visão abrangente e fundamentada sobre a implementação do DDD, partindo de uma perspectiva qualitativa e exploratória, que permite

captar nuances e complexidades não visíveis em estudos quantitativos ou experimentais. Espera-se, com isso, promover uma reflexão crítica sobre o estado atual do uso de DDD no desenvolvimento de software e estimular a formulação de abordagens metodológicas mais adequadas à sua aplicação nos mais diversos contextos organizacionais.

2 Fundamentação Teórica

Esta seção tem como objetivo estabelecer os fundamentos conceituais e estruturais do Domain-Driven Design (DDD), a partir de sua formulação original por Evans (2003) e dos desdobramentos que a abordagem tem sofrido ao longo do tempo. Serão abordados os princípios centrais da modelagem orientada ao domínio, os elementos do design estratégico, como bounded contexts e context maps, e os padrões táticos utilizados na construção de modelos coesos e expressivos. Por fim, serão discutidas as principais evoluções e adaptações do DDD em contextos contemporâneos, evidenciando sua plasticidade conceitual e aplicabilidade em diferentes paradigmas arquiteturais.

2.1 Fundamentos Conceituais do Domain-Driven Design

O Domain-Driven Design (DDD) constitui uma abordagem de desenvolvimento de software centrada no domínio do negócio e na colaboração entre especialistas técnicos e especialistas do domínio. Introduzido por Eric Evans em sua obra seminal Domain-Driven Design: Tackling Complexity in the Heart of Software (2003), o DDD propõe que o conhecimento do domínio seja o principal elemento orientador do design e da implementação de sistemas complexos. Segundo Evans (2003), os sistemas de software frequentemente

falham não por limitações técnicas, mas por insuficiência na compreensão do problema que se propõem a resolver. Nesse contexto, o DDD surge como um paradigma que privilegia a modelagem conceitual e a comunicação efetiva entre os atores do processo de desenvolvimento.

A premissa central do DDD é que o modelo do domínio, isto é, a representação conceitual das entidades, regras e comportamentos relevantes do problema, deve estar no centro da arquitetura do sistema. Esse modelo deve ser construído e refinado continuamente em colaboração com os especialistas do negócio, o que requer uma linguagem compartilhada, precisa e consistente. Essa linguagem, denominada por Evans (2003) como "linguagem ubíqua" (ubiquitous language), é empregada tanto na modelagem quanto no código, nos testes, na documentação e nas conversas diárias da equipe. A linguagem ubíqua elimina ambiguidades, aproxima o código da lógica de negócio e promove um alinhamento mais efetivo entre as partes envolvidas (Vernon, 2013; Brandolini, 2015).

A ênfase na modelagem do domínio distingue o DDD de outras abordagens arquitetônicas mais orientadas à tecnologia ou à estrutura de dados. No DDD, a complexidade essencial, entendida como aquela que está intrinsecamente ligada à natureza do problema, deve ser isolada e representada de forma explícita no modelo. Essa modelagem deve capturar não apenas os dados, mas os comportamentos, regras e interações que dão significado ao sistema. A clareza conceitual é, portanto, um valor central no DDD, pois permite a evolução controlada do sistema e a minimização de dívidas técnicas relacionadas à incompreensão do domínio (Evans, 2003; Khononov, 2021).

Além da linguagem ubíqua, outro conceito fundamental no DDD é a ideia de que os modelos devem evoluir de forma iterativa e colaborativa. O processo de modelagem não é estático nem linear, mas sim uma atividade contínua de aprendizado, refinamento e validação com os especialistas de negócio. Essa característica torna o DDD especialmente adequado para ambientes complexos, incertos ou em constante mudança, onde requisitos emergem de forma não previsível. Em tais contextos, o valor do DDD reside não apenas nos artefatos produzidos, mas no próprio processo de descoberta e alinhamento entre os envolvidos (Evans, 2003; Vernon, 2016).

A abordagem proposta por Evans (2003) encontra ressonância em princípios contemporâneos do desenvolvimento ágil e da engenharia de software orientada a domínio. Por exemplo, os princípios do Manifesto Ágil, como colaboração com o cliente, resposta a mudanças e comunicação face a face, estão em sintonia com os valores do DDD, especialmente no que diz respeito à modelagem conjunta e ao foco na entrega de valor de negócio. Por sua vez, os princípios da engenharia de software moderna, como modularidade, encapsulamento e coesão, são operacionalizados no DDD por meio de seus mecanismos táticos e estratégicos, conforme será discutido nas seções seguintes (Evans, 2003; Vernon, 2016; Zimmermann et al., 2017).

Contudo, é importante destacar que o DDD não se apresenta como uma metodologia prescritiva ou um conjunto rígido de práticas. Trata-se, antes, de um conjunto de princípios, padrões e heurísticas que orientam a construção de sistemas baseados em modelos conceituais ricos e expressivos. Essa característica torna o DDD altamente adaptável a diferentes contextos organizacionais, arquiteturas técnicas e estilos de desenvolvimento. Entretanto, essa

mesma flexibilidade pode gerar interpretações equivocadas, implementações parciais ou distorções dos conceitos originais, como apontado por Özkan (2023) em sua revisão sistemática da literatura.

Pesquisas recentes têm evidenciado o potencial do DDD para promover benefícios significativos no desenvolvimento de sistemas, tais como maior modularidade, melhor alinhamento entre áreas técnicas e de negócio, clareza arquitetural e facilidade de manutenção (Jordanov, 2023; Rapôso, 2025). No entanto, tais benefícios estão condicionados à capacidade das organizações de internalizar os fundamentos conceituais do DDD, o que exige investimento em capacitação, mudanças culturais e disposição para processos colaborativos e iterativos. A adoção superficial ou tecnicista do DDD, centrada apenas na aplicação de padrões estruturais, tende a falhar em capturar seu verdadeiro potencial transformador (Khononov, 2021).

Nesse sentido, a fundamentação conceitual do DDD deve ser compreendida como uma base filosófica para o design de software. Trata-se de um convite à reflexão sobre o papel do desenvolvedor como modelador do domínio, sobre a importância da linguagem como instrumento de design e sobre a necessidade de uma escuta ativa e contínua dos especialistas do negócio. A prática do DDD exige, portanto, mais do que domínio técnico. Requer sensibilidade para a complexidade do mundo real, disposição para o aprendizado contínuo e compromisso com a qualidade conceitual do sistema.

Com base nessa compreensão dos fundamentos do DDD, torna-se possível avançar para uma análise mais aprofundada de seus componentes estruturais. Na seção seguinte, serão discutidos os

elementos do design estratégico, com ênfase nos conceitos de bounded context, context maps e suas implicações para a arquitetura e a organização do sistema.

2.2 Design Estratégico: Bounded Contexts e Context Maps

O design estratégico é uma das dimensões centrais do Domain-Driven Design (DDD), sendo responsável por definir os limites conceituais, linguísticos e técnicos de cada subsistema dentro de uma solução de software. Esse aspecto do DDD lida com decisões de mais alto nível, que transcendem a implementação de componentes individuais e dizem respeito à organização geral da arquitetura, ao alinhamento com o negócio e à governança das integrações entre diferentes partes de um sistema (Evans, 2003; Vernon, 2016).

O principal conceito estruturante do design estratégico é o de bounded context. De acordo com Evans (2003), um bounded context é um espaço lógico no qual um determinado modelo do domínio é coerente, consistente e semanticamente estável. Dentro de um bounded context, os termos usados na linguagem ubíqua mantêm seu significado específico, e as regras de negócio são aplicadas de forma uniforme. Fora desse limite, no entanto, esses significados podem mudar, dando origem a outros modelos, outras regras e outras linguagens. Portanto, o papel do bounded context é preservar a integridade semântica e funcional do modelo em ambientes complexos e multifacetados.

A introdução de bounded contexts é uma resposta ao problema clássico da ambiguidade semântica em sistemas grandes e distribuídos. Quando múltiplas equipes trabalham sobre diferentes

partes de um sistema e utilizam termos iguais com significados distintos, surgem conflitos de interpretação, falhas de integração e inconsistências nos dados. O bounded context impõe uma separação explícita desses significados, promovendo a autonomia de desenvolvimento e a modularização semântica. Essa separação, por sua vez, permite que cada equipe se concentre na modelagem precisa do seu próprio domínio, sem a obrigação de harmonizar todos os detalhes com outros subsistemas (Brandolini, 2015; Vernon, 2016).

A definição correta dos bounded contexts é uma das atividades mais desafiadoras no DDD, exigindo não apenas conhecimento técnico, mas também profundo entendimento do negócio. Essa tarefa envolve a identificação de fronteiras naturais entre responsabilidades organizacionais, fluxos de valor, comportamentos de usuários e regras de negócio. Tais fronteiras nem sempre coincidem com estruturas técnicas existentes, como módulos de código, microserviços ou domínios de banco de dados, o que demanda esforço de análise, negociação e adaptação por parte das equipes envolvidas (Khononov, 2021).

Em complemento aos bounded contexts, o DDD introduz o conceito de context map, ou mapa de contextos. O context map é uma representação explícita das relações entre os diversos bounded contexts existentes em um sistema, incluindo os tipos de integração que os conectam. Segundo Evans (2003), o context map não é apenas um diagrama técnico, mas um artefato estratégico que expressa acordos de colaboração, fluxos de dependência e mecanismos de proteção entre contextos.

O context map define, por exemplo, se dois bounded contexts compartilham um núcleo comum (shared kernel), se existe uma relação de dependência controlada (customer/supplier), se há um mecanismo de tradução entre os modelos (anticorruption layer) ou se os contextos interagem de forma aberta e não coordenada (open host service). Cada uma dessas estratégias de integração traz implicações distintas para o acoplamento, a governança e a evolução dos sistemas. A escolha da estratégia adequada depende do grau de confiança, estabilidade e autonomia desejado entre os contextos (Vernon, 2016; Zimmermann et al., 2017).

Um exemplo clássico da aplicação desses conceitos ocorre em arquiteturas de microserviços, nas quais o uso de bounded contexts permite que cada serviço represente um modelo de domínio distinto e evolua de maneira independente. Em tais arquiteturas, o context map atua como uma ferramenta de governança para evitar que os serviços se tornem excessivamente acoplados ou compartilhem conhecimento indevido de implementações alheias. Dessa forma, o design estratégico do DDD se alinha a princípios de autonomia, escalabilidade e resiliência típicos de sistemas distribuídos contemporâneos (Fowler, 2015; Jordanov, 2023).

Além do impacto técnico, os bounded contexts e os context maps têm implicações organizacionais significativas. Em muitas organizações, a estrutura do sistema de software reflete, conscientemente ou não, a estrutura da própria organização, conforme postula a Lei de Conway (Conway, 1968). Ao utilizar bounded contexts como instrumento de alinhamento entre arquitetura de software e estrutura organizacional, é possível reduzir os atritos entre equipes, melhorar a comunicação e facilitar a responsabilidade clara sobre subsistemas. Essa abordagem fortalece

a ideia de que o design do sistema não pode ser dissociado da cultura e da dinâmica da organização em que ele está inserido (Skelton & Pais, 2019).

Entretanto, a prática demonstra que nem sempre os bounded contexts são implementados de maneira coerente ou sustentável. Uma das críticas recorrentes é que muitas equipes aplicam o conceito de forma excessivamente técnica, limitando-se à criação de microsserviços sem considerar a coerência semântica e os limites de negócio. Tais implementações tendem a reproduzir os mesmos problemas de ambiguidade e acoplamento que o DDD procura evitar, resultando em complexidade desnecessária e baixa manutenibilidade (Rapôso, 2025; Özkan, 2023).

Outro desafio identificado em estudos recentes é a ausência de ferramentas e práticas consolidadas para apoiar a definição e o mapeamento dos bounded contexts. Embora existam abordagens como EventStorming (Brandolini, 2015) e Domain Storytelling (Wicke et al., 2022), sua adoção ainda é incipiente em muitos ambientes corporativos, especialmente em contextos onde o conhecimento do domínio está fragmentado ou disperso entre múltiplos stakeholders. A falta de uma modelagem colaborativa estruturada dificulta a construção de um modelo compartilhado e compromete a qualidade dos bounded contexts definidos.

Portanto, o design estratégico no DDD exige uma combinação de habilidades técnicas, analíticas e relacionais. A definição de bounded contexts e a elaboração de context maps são atividades críticas para o sucesso da abordagem e requerem engajamento ativo das equipes de desenvolvimento, especialistas do negócio e lideranças organizacionais. Quando bem conduzidas, essas práticas

proporcionam uma base sólida para a construção de sistemas modulares, expressivos e alinhados ao domínio. Quando negligenciadas ou mal interpretadas, elas comprometem a integridade do modelo e limitam o potencial do DDD como ferramenta de design centrado no domínio.

Na próxima subseção, será apresentada uma análise detalhada dos elementos do design tático do DDD, abordando os padrões estruturais utilizados na implementação do modelo do domínio e suas implicações para a expressividade e coesão dos sistemas desenvolvidos.

2.3 Design Tático: Padrões e Modelagem de Domínio

O design tático constitui a dimensão operacional do Domain-Driven Design (DDD), focando na implementação concreta do modelo de domínio por meio de padrões estruturais e comportamentais. Enquanto o design estratégico define os limites contextuais e as relações entre subsistemas, o design tático busca organizar e expressar com clareza os elementos internos de um domínio, promovendo coesão, consistência e expressividade no código. Trata-se de um conjunto de técnicas e abstrações que auxiliam no mapeamento conceitual do domínio para sua realização em estruturas de software (Evans, 2003; Vernon, 2016).

No cerne do design tático estão os building blocks ou blocos de construção do modelo de domínio. Estes incluem entities, value objects, aggregates, repositories, domain services e domain events. Cada um desses elementos possui uma função específica no modelo e deve ser utilizado de acordo com as características do domínio a ser representado. A escolha e o uso apropriado desses

padrões são essenciais para manter a integridade conceitual do sistema, reduzir a complexidade acidental e favorecer a evolução contínua do modelo (Evans, 2003; Vernon, 2013).

As entities são objetos identificáveis por uma chave única e cuja identidade persiste ao longo do tempo, mesmo com alterações em seus atributos. Elas representam conceitos do domínio que têm continuidade e trajetória histórica, como um cliente, um pedido ou um contrato. De acordo com Evans (2003), o foco principal no design de uma entity deve ser sua identidade, e não seus dados. Assim, a modelagem deve garantir que as invariantes que definem a consistência dessa entidade sejam preservadas em todas as operações que a envolvem.

Em contraste, os value objects representam conceitos que não possuem identidade própria e são definidos unicamente por seus atributos. São imutáveis por natureza e podem ser comparados por igualdade estrutural. Exemplos comuns incluem medidas, coordenadas, intervalos de tempo ou endereços. O uso de value objects favorece a imutabilidade, reduz efeitos colaterais e torna o código mais expressivo e seguro (Vernon, 2013; Khononov, 2021). A modelagem adequada de value objects contribui significativamente para a clareza do domínio e a robustez do modelo.

Outro padrão essencial do design tático é o aggregate. Um aggregate é uma unidade de consistência transacional composta por uma entity root e os objetos que a ela estão logicamente vinculados. Segundo Evans (2003), o objetivo do aggregate é proteger as invariantes do modelo, garantindo que todas as alterações ocorram de forma controlada por meio da raiz do agregado (aggregate root). Essa organização reduz o acoplamento

entre partes do sistema e simplifica a concorrência e a persistência, pois os aggregates podem ser tratados como fronteiras de isolamento.

A correta definição dos limites de um aggregate é uma tarefa desafiadora, pois envolve decisões sobre coesão, consistência e granularidade. Como aponta Vernon (2016), aggregates excessivamente grandes tendem a dificultar o desempenho e a escalabilidade do sistema, enquanto aggregates muito fragmentados podem comprometer a expressividade do modelo. A recomendação geral é que um aggregate represente um conceito do domínio que possua invariantes internas fortes, com escopo transacional claramente delimitado.

Os repositories são responsáveis por abstrair o acesso a coleções de aggregates, permitindo que o domínio permaneça independente de detalhes de persistência. Um repository deve oferecer métodos que façam sentido no contexto do modelo, como “buscar cliente por CPF” ou “listar pedidos pendentes”, evitando expor operações genéricas que enfraqueçam o vínculo semântico com o domínio. Essa abstração favorece o isolamento do modelo de domínio em relação a tecnologias específicas, como bancos relacionais, NoSQL ou sistemas de mensageria (Evans, 2003; Vernon, 2013).

Os domain services, por sua vez, são usados para representar comportamentos do domínio que não pertencem naturalmente a uma única entidade ou value object. Tais serviços encapsulam lógica de negócio que atravessa múltiplos objetos e devem ser nomeados de forma clara e alinhada à linguagem ubíqua. É fundamental que esses serviços não se tornem depósitos genéricos de lógica e mantenham alta coesão com o domínio (Khononov, 2021).

Além dos padrões mencionados, o DDD moderno incorporou o uso de domain events, que são eventos significativos ocorridos no domínio e que podem desencadear reações em outros componentes do sistema. Os domain events promovem uma arquitetura orientada a eventos e reforçam o desacoplamento entre partes do sistema, permitindo reações assíncronas e flexíveis a mudanças de estado. Esse padrão é especialmente útil em arquiteturas distribuídas e sistemas baseados em mensageria, como Kafka ou RabbitMQ (Fowler, 2015; Vernon, 2016).

A aplicação dos padrões do design tático não deve ser mecânica, mas orientada pelo entendimento profundo do domínio. O uso indiscriminado desses blocos pode gerar modelos inflados, sobreengenharia e complexidade desnecessária. Como observa Brandolini (2015), a eficácia do DDD depende mais da qualidade da modelagem e da colaboração entre especialistas do que da adoção formal de seus padrões. Nesse sentido, práticas colaborativas como EventStorming ou Domain Storytelling têm ganhado relevância por facilitar a descoberta e validação coletiva dos modelos (Wicke et al., 2022).

Estudos empíricos indicam que a correta aplicação dos padrões do design tático está associada a ganhos em manutenibilidade, clareza arquitetural e facilidade de testes (Jordanov, 2023; Rapôso, 2025). Entretanto, tais benefícios só se concretizam quando as equipes compreendem o significado dos conceitos envolvidos e os utilizam em sintonia com os princípios do design estratégico. A dissociação entre os dois níveis de design frequentemente leva à criação de modelos incoerentes, desarticulados ou inconsistentes com os limites contextuais definidos anteriormente.

Outro ponto crítico é a capacitação técnica e conceitual das equipes. Muitos desenvolvedores entram em contato com os padrões do DDD por meio de tutoriais técnicos que abordam apenas sua implementação em frameworks como Spring ou NestJS, sem compreender os fundamentos conceituais que os sustentam. Essa lacuna favorece práticas equivocadas, como a confusão entre entity e aggregate, o uso de repositories como gateways genéricos ou a atribuição indevida de lógica a domain services (Khononov, 2021; Özkan, 2023).

Em síntese, o design tático oferece um conjunto robusto de ferramentas para a modelagem de domínios complexos de forma expressiva e sustentável. Quando integrado ao design estratégico e orientado pela linguagem ubíqua, esse conjunto permite que os sistemas reflitam com fidelidade as regras de negócio, promovendo alinhamento, clareza e evolução contínua. No entanto, sua eficácia depende da maturidade das equipes, da compreensão dos fundamentos do DDD e da capacidade de aplicar tais padrões com critério, adaptando-os às particularidades de cada contexto organizacional e técnico.

Na próxima subseção, será discutida a evolução contemporânea do DDD, com ênfase nos desafios práticos enfrentados durante sua implementação, bem como nas adaptações recentes observadas em diferentes contextos, como microserviços, computação em nuvem e programação funcional.

2.4 Evoluções Contemporâneas e Desafios Práticos da Implementação

Desde sua formulação inicial por Evans (2003), o Domain-Driven Design (DDD) tem sido continuamente reinterpretado, adaptado e expandido para atender às demandas emergentes da engenharia de software contemporânea. Essa evolução tem se intensificado nas últimas duas décadas, sobretudo diante da adoção crescente de arquiteturas distribuídas, como microserviços, da popularização de paradigmas como a programação funcional e do uso intensivo de plataformas baseadas em nuvem. Esses contextos, caracterizados por alta complexidade, descentralização e escalabilidade, criaram um ambiente propício para que os princípios do DDD fossem rediscutidos e reinterpretados, tanto na academia quanto na indústria (Fowler, 2015; Khononov, 2021; Jordanov, 2023).

Uma das principais linhas de evolução do DDD contemporâneo se dá na interface com arquiteturas baseadas em microserviços. Como destacado por Newman (2015), os microserviços pressupõem a decomposição de sistemas em serviços autônomos, pequenos e especializados, o que está em consonância com os princípios de bounded contexts do DDD. Nesse arranjo, cada serviço representa um modelo de domínio específico, com sua própria linguagem ubíqua, regras de negócio e persistência de dados. Essa compatibilidade estrutural favoreceu a incorporação do DDD como abordagem de modelagem preferencial em projetos de microserviços (Vernon, 2016; Zimmermann et al., 2017).

Contudo, essa associação entre DDD e microserviços, embora produtiva, nem sempre ocorre de forma coerente com os fundamentos conceituais do DDD. Muitos projetos iniciam a fragmentação técnica de sistemas em serviços sem antes realizar uma definição adequada dos bounded contexts, resultando em serviços mal definidos, com sobreposição de responsabilidades,

redundância semântica e integração frágil. Esse problema é frequentemente mencionado como uma das causas da "complexidade acidental" que afeta sistemas de microserviços mal arquitetados (Khononov, 2021; Özkan, 2023). Assim, a aplicação correta do DDD em arquiteturas distribuídas requer uma compreensão aprofundada dos limites do domínio, da governança dos contextos e das estratégias de integração entre subsistemas.

Outra tendência relevante diz respeito à incorporação de princípios da programação funcional na implementação de DDD. Autores como Khononov (2021) e Vernon (2021) argumentam que a imutabilidade, a ausência de efeitos colaterais e o uso explícito de funções puras na programação funcional favorecem a clareza e a previsibilidade do modelo de domínio. A separação entre dados e comportamento, típica desse paradigma, pode parecer contraditória aos padrões tradicionais do DDD, como entities e domain services, mas também pode ser reinterpretada de forma complementar. Em vez de implementar comportamentos diretamente em objetos, a lógica de negócio pode ser modelada como funções que operam sobre value objects imutáveis, promovendo maior testabilidade e transparência (Jordanov, 2023).

Apesar dessas inovações, a implementação prática do DDD ainda enfrenta diversos desafios, tanto de natureza técnica quanto organizacional. Um dos obstáculos mais recorrentes é a curva de aprendizado elevada, especialmente para equipes que não possuem familiaridade com modelagem conceitual, abstrações de alto nível ou colaboração direta com especialistas do negócio. Conforme apontado por Rapôso (2025), muitas equipes têm dificuldade em distinguir conceitos como entity, aggregate e domain service, o que

resulta em modelos incoerentes, duplicação de lógica e baixa expressividade semântica.

Outro problema frequente refere-se à ausência de ferramentas consolidadas para apoiar a modelagem colaborativa do domínio. Embora existam métodos como EventStorming (Brandolini, 2015), Domain Storytelling (Wicke et al., 2022) e Example Mapping, sua adoção ainda é limitada, principalmente em contextos corporativos tradicionais. A falta de práticas estruturadas para a descoberta e validação de modelos compromete a construção de uma linguagem ubíqua compartilhada e dificulta a consolidação dos bounded contexts. Além disso, a dependência de ferramentas visuais ou presenciais pode ser um entrave em ambientes remotos ou distribuídos, exigindo adaptações metodológicas e culturais.

Do ponto de vista técnico, a integração entre contextos é outro ponto crítico na implementação do DDD. Embora os context maps ofereçam um vocabulário rico para descrever relações como anticorruption layers, shared kernels e open host services, a aplicação prática desses padrões exige planejamento, testes e infraestrutura adequada. Muitas vezes, as equipes recorrem a integrações rápidas e frágeis, como chamadas diretas de API sem abstração ou contratos sem versionamento, o que gera acoplamento indevido entre contextos e compromete a autonomia dos modelos (Zimmermann et al., 2017; Özkan, 2023).

No plano organizacional, o sucesso da adoção do DDD está diretamente relacionado à maturidade da organização em lidar com domínio, conhecimento tácito e interdisciplinaridade. A colaboração entre desenvolvedores e especialistas de negócio, requisito central do DDD, demanda estruturas de comunicação abertas, confiança

mútua e tempo para aprendizado compartilhado. Em organizações hierárquicas ou fragmentadas, a ausência de canais diretos entre as equipes pode comprometer a construção da linguagem ubíqua e a eficácia do processo de modelagem. Além disso, a pressão por entregas rápidas e a resistência a mudanças estruturais dificultam a adoção de práticas reflexivas e iterativas, que são fundamentais para o DDD (Skelton & Pais, 2019).

Estudos de caso apontam também que a adoção do DDD costuma ser mais bem-sucedida quando integrada a outras abordagens complementares, como metodologias ágeis (Scrum, Kanban), DevOps e arquitetura orientada a eventos. Essa integração favorece ciclos curtos de feedback, experimentação contínua e automatização de processos, todos elementos que ampliam a capacidade da equipe de lidar com complexidade e incerteza (Jordanov, 2023; Rapôso, 2025). No entanto, essa sinergia requer coesão metodológica e uma cultura organizacional orientada ao aprendizado contínuo.

Finalmente, cabe destacar que, apesar dos avanços significativos na consolidação do DDD como abordagem de modelagem centrada no domínio, ainda existem lacunas importantes na literatura e na prática profissional. Falta padronização de métricas para avaliação da qualidade dos modelos, escassez de estudos quantitativos sobre os impactos do DDD em projetos reais e insuficiência de diretrizes para sua adoção gradual em contextos legados. Tais lacunas representam oportunidades relevantes para pesquisas futuras, sobretudo do tipo empírico, experimental e longitudinal.

3 Resultados e Discussão

Com base na fundamentação teórica estabelecida na seção anterior, esta parte do artigo apresenta os achados da investigação exploratória conduzida sobre a implementação do DDD em ambientes reais de desenvolvimento de software. A análise foi organizada em quatro eixos principais: as práticas mais frequentemente adotadas por equipes técnicas, os benefícios relatados em projetos que empregaram a abordagem, os desafios e barreiras enfrentados durante sua adoção e, por fim, as variações metodológicas observadas em diferentes contextos organizacionais e arquiteturais. Essa sistematização visa oferecer uma visão abrangente e crítica sobre o estado atual da aplicação do DDD.

3.1 Evoluções Contemporâneas e Desafios Práticos da Implementação

A investigação exploratória da literatura científica e técnica sobre a implementação de Domain-Driven Design (DDD) revela um conjunto de práticas recorrentes que têm sido adotadas por equipes de desenvolvimento em diferentes contextos organizacionais. Tais práticas refletem tanto os fundamentos conceituais da abordagem quanto adaptações necessárias ao ambiente técnico e à cultura da organização. Esta subseção apresenta as principais práticas identificadas, organizadas em três eixos: modelagem colaborativa do domínio, delimitação e gestão de bounded contexts e aplicação dos padrões táticos no desenvolvimento.

A primeira prática amplamente observada é o uso de técnicas de modelagem colaborativa para construção da linguagem ubíqua e do modelo do domínio. Entre as técnicas mais mencionadas está o EventStorming, proposta por Brandolini (2015), que consiste em sessões facilitadas com a presença de desenvolvedores, analistas e

especialistas do negócio, onde eventos do domínio são identificados e organizados cronologicamente. Essa prática permite não apenas levantar requisitos, mas também compreender os fluxos de negócio, identificar agregados e delimitar contextos. Estudos como o de Wicke et al. (2022) e Khononov (2021) destacam que essa prática favorece o alinhamento entre áreas técnicas e de negócio, promove engajamento dos stakeholders e acelera a construção de modelos mais expressivos e realistas.

Além do EventStorming, outras abordagens como Domain Storytelling e Example Mapping têm sido utilizadas para capturar narrativas do domínio a partir da perspectiva dos usuários. Essas práticas se destacam por sua ênfase em histórias, personas e fluxos de interação, contribuindo para a descoberta de regras de negócio implícitas e para a validação iterativa do modelo. A literatura aponta que equipes que adotam essas técnicas tendem a construir modelos mais coesos e com maior aderência à realidade operacional da organização (Wicke et al., 2022; Rapôso, 2025).

Outro conjunto de práticas recorrentes refere-se à identificação e gestão de bounded contexts. A definição explícita desses contextos, com base em critérios semânticos, organizacionais e técnicos, tem sido apontada como um fator-chave para a modularidade e a escalabilidade dos sistemas. A prática consiste em mapear domínios de negócio, processos e unidades organizacionais, identificando onde há significados distintos para os mesmos termos ou variações de regras. Com base nessa análise, são delimitados os bounded contexts, que passam a ter modelos, equipes e responsabilidades autônomas.

Uma prática complementar a essa delimitação é a construção de context maps, que representam graficamente os relacionamentos entre os contextos, os padrões de integração adotados e os fluxos de dependência. O uso de context maps como artefato estratégico tem sido destacado por Vernon (2016) e Zimmermann et al. (2017) como essencial para a governança arquitetural e para a redução de acoplamento entre subsistemas. A prática de revisar e refinar periodicamente os context maps, com base na evolução do domínio e nas interações reais entre contextos, também foi identificada como um elemento importante de sustentabilidade da arquitetura.

No que se refere à implementação dos modelos, observa-se o uso extensivo dos padrões do design tático, especialmente aggregates, entities, value objects e repositories. A literatura analisada aponta que equipes que compreendem e aplicam adequadamente esses padrões conseguem construir modelos coesos, alinhados à linguagem ubíqua e com menor acoplamento técnico. A modelagem de agregados com foco em invariantes de negócio, o uso de objetos de valor imutáveis para representar conceitos atômicos e a organização dos repositórios por comportamento do domínio são práticas recorrentes em equipes maduras na aplicação do DDD (Evans, 2003; Vernon, 2013; Khononov, 2021).

Uma prática emergente identificada em estudos recentes é o uso de domain events como mecanismo de orquestração e integração entre contextos. Equipes que adotam arquitetura orientada a eventos têm utilizado domain events para capturar mudanças significativas no domínio e permitir que outros subsistemas reajam a essas mudanças de forma assíncrona. Essa prática tem sido particularmente eficaz em ambientes distribuídos, com múltiplos bounded contexts e alto grau de desacoplamento. A literatura

técnica e acadêmica aponta que o uso disciplinado de eventos do domínio favorece a rastreabilidade, a resiliência e a coesão do modelo (Fowler, 2015; Jordanov, 2023).

Em termos de ferramentas, observa-se o uso de recursos visuais, como quadros Kanban físicos ou digitais, murais interativos e softwares de modelagem como Miro, Lucidchart ou ferramentas específicas de EventStorming. Essas ferramentas facilitam a documentação viva do modelo, promovem a colaboração síncrona e assíncrona e permitem a atualização contínua da linguagem ubíqua. Além disso, práticas como o versionamento de modelos, a criação de testes de aceitação baseados em cenários do domínio e o uso de DSLs (linguagens específicas de domínio) têm ganhado espaço como formas de formalizar e automatizar os conhecimentos capturados durante a modelagem (Vernon, 2016; Rapôso, 2025).

Destaca-se a prática de adoção incremental do DDD. A maioria dos relatos analisados indica que a implementação da abordagem raramente ocorre de forma integral ou simultânea. Em vez disso, observa-se um movimento de experimentação progressiva, iniciando-se geralmente por um contexto específico, como a reestruturação de um subsistema crítico ou a modelagem de um processo de negócio prioritário. Essa abordagem incremental permite a aprendizagem gradual da equipe, a validação empírica dos ganhos e a adaptação das práticas ao contexto organizacional. Conforme ressaltado por Khononov (2021), a adoção bem-sucedida do DDD é menos um evento pontual e mais um processo contínuo de transformação cultural e técnica.

A próxima subseção apresentará os benefícios relatados pela literatura e pelos relatos técnicos, detalhando os impactos positivos

da adoção do DDD em diferentes aspectos do desenvolvimento de software e da estrutura organizacional.

3.2 Benefícios Relatados em Diferentes Contextos

A análise exploratória da literatura e de relatos técnicos evidencia uma série de benefícios associados à implementação do Domain-Driven Design (DDD) em projetos de desenvolvimento de software. Esses benefícios, embora variem conforme o grau de maturidade da equipe, a complexidade do domínio e o contexto organizacional, convergem em torno de quatro dimensões principais: clareza conceitual e comunicacional, modularidade arquitetural, alinhamento organizacional e sustentabilidade evolutiva. Esta subseção detalha cada um desses eixos, com base em evidências relatadas por autores nacionais e internacionais.

A clareza conceitual e comunicacional é, segundo Evans (2003), a contribuição mais imediata e essencial do DDD. A ênfase na linguagem ubíqua e na modelagem colaborativa do domínio permite que todos os envolvidos (desenvolvedores, analistas, gestores e especialistas de negócio) compartilhem um vocabulário comum e preciso. Essa prática reduz ambiguidades, elimina ruídos na comunicação e promove maior assertividade na definição de requisitos e regras de negócio. Estudos como os de Vernon (2016) e Brandolini (2015) confirmam que a adoção sistemática da linguagem ubíqua melhora significativamente a qualidade da interação entre equipes e acelera o processo de entendimento dos problemas a serem resolvidos.

Além da linguagem, o próprio modelo do domínio, quando bem construído, passa a funcionar como um artefato de comunicação

organizacional. Modelos expressivos, coesos e alinhados ao vocabulário do negócio tornam-se legíveis não apenas para desenvolvedores, mas também para decisores estratégicos. Isso facilita a tomada de decisão, a validação de soluções e a transparência nos processos de desenvolvimento. Como destaca Khononov (2021), um modelo claro não apenas orienta o código, mas também educa a organização sobre seu próprio funcionamento, promovendo aprendizado coletivo.

Outro benefício amplamente relatado é a modularidade arquitetural proporcionada pelos bounded contexts e pelos context maps. A definição explícita de fronteiras semânticas e técnicas entre partes do sistema permite uma decomposição coerente, com alta coesão interna e baixo acoplamento entre módulos. Essa modularidade favorece a escalabilidade do sistema, a especialização das equipes e a flexibilidade na escolha de tecnologias e estratégias de implementação. Segundo Zimmermann et al. (2017), essa característica é particularmente vantajosa em arquiteturas baseadas em microserviços, onde a autonomia e a isolabilidade dos serviços são requisitos fundamentais.

Essa modularidade também contribui para a redução de complexidade técnica e organizacional. Ao segmentar o sistema em unidades menores e semanticamente consistentes, torna-se mais fácil compreender, manter e evoluir cada parte. Estudos de caso analisados por Jordanov (2023) indicam que equipes que adotaram o DDD conseguiram reduzir o tempo médio de desenvolvimento de novas funcionalidades, diminuir o retrabalho associado a erros de entendimento e aumentar a previsibilidade das entregas. Tais resultados são atribuídos, em grande medida, à clareza dos limites contextuais e à responsabilização explícita sobre os modelos.

No plano organizacional, o DDD tem sido associado a melhorias na estruturação das equipes e na articulação entre áreas de negócio e tecnologia. A associação entre bounded contexts e células autônomas de desenvolvimento (como as stream-aligned teams propostas por Skelton e Pais, 2019) promove alinhamento organizacional, clareza de papéis e redução de dependências cruzadas. Em organizações que adotaram essa abordagem, observa-se um aumento na responsabilidade técnica das equipes, maior proximidade com os stakeholders de negócio e maior autonomia na tomada de decisões técnicas. Essa configuração organizacional se alinha a princípios de DevOps e engenharia de fluxo, contribuindo para ciclos de entrega mais curtos e respostas mais ágeis às mudanças (Skelton & Pais, 2019; Vernon, 2016).

Um aspecto frequentemente mencionado como benefício indireto do DDD é a melhoria na qualidade do código e na manutenibilidade dos sistemas. A aplicação sistemática dos padrões táticos, como aggregates, value objects e domain services, promove estruturas de código mais coesas, testáveis e alinhadas ao domínio. Essa clareza estrutural reduz o acoplamento implícito, facilita a refatoração e incentiva a prática de testes orientados ao comportamento do domínio. Como destacado por Evans (2003) e Vernon (2013), quando o código reflete com fidelidade a lógica do domínio, torna-se mais robusto a mudanças e mais resiliente à obsolescência.

Além disso, o uso de domain events como mecanismo de comunicação entre contextos favorece a arquitetura orientada a eventos, que promove reatividade, desacoplamento e rastreabilidade. Sistemas que adotam esse padrão relatam maior flexibilidade na integração de novos serviços, maior visibilidade sobre os processos de negócio e maior resiliência a falhas locais. Em

ambientes distribuídos, a comunicação baseada em eventos contribui para a escalabilidade horizontal e para a orquestração assíncrona de fluxos complexos (Fowler, 2015; Jordanov, 2023).

Outro benefício relevante refere-se à sustentabilidade evolutiva dos sistemas construídos com DDD. A ênfase na modelagem contínua, na validação iterativa dos conceitos e na adaptação do modelo às mudanças do domínio permite que os sistemas evoluam de forma controlada e consistente ao longo do tempo. Essa característica é especialmente valiosa em contextos de alta incerteza, como mercados dinâmicos ou ambientes regulatórios em constante transformação. Rapôso (2025) argumenta que o DDD favorece a longevidade arquitetural, uma vez que os modelos são construídos com base em conhecimento de domínio e não em convenções técnicas passageiras.

Finalmente, cabe destacar o impacto positivo do DDD na aprendizagem organizacional. A prática de modelar coletivamente o domínio, de revisar regularmente os modelos e de alinhar código e negócio estimula o desenvolvimento de competências técnicas e analíticas nas equipes. A literatura indica que organizações que adotam o DDD tendem a desenvolver maior maturidade em engenharia de software, maior compreensão dos seus próprios processos e maior capacidade de inovar de forma orientada ao domínio (Khononov, 2021; Vernon, 2016; Özkan, 2023).

Na próxima subseção, será feita uma análise dos principais desafios e barreiras enfrentados durante a adoção e implementação do DDD, conforme reportado pela literatura recente e pelos estudos de caso avaliados.

3.3 Desafios Frequentes e Barreiras Organizacionais

Embora os benefícios da adoção do Domain-Driven Design (DDD) estejam bem documentados, a implementação efetiva dessa abordagem é marcada por uma série de desafios que se manifestam tanto no plano técnico quanto no organizacional. A literatura especializada e os relatos de casos práticos apontam para dificuldades recorrentes que limitam a adoção bem-sucedida do DDD, gerando frustrações, desvios conceituais e, em alguns casos, o abandono da iniciativa. Esta subseção apresenta os principais desafios relatados, agrupando-os em quatro eixos: complexidade conceitual, barreiras culturais e organizacionais, dificuldades técnicas e lacunas metodológicas.

O primeiro e talvez mais notório desafio refere-se à complexidade conceitual do DDD. Desde sua formulação por Evans (2003), a abordagem é reconhecida por sua densidade teórica e pela exigência de um pensamento abstrato voltado à modelagem conceitual. Muitos desenvolvedores, especialmente aqueles com formação técnica fortemente orientada a frameworks ou arquitetura procedural, apresentam dificuldade em assimilar os fundamentos do DDD, como a centralidade do domínio, a linguagem ubíqua, os bounded contexts e os padrões táticos. Conforme apontado por Khononov (2021), essa lacuna conceitual resulta na aplicação mecânica ou distorcida dos padrões do DDD, como a confusão entre entities e aggregates, o uso indevido de domain services ou a ausência de critérios para a definição de bounded contexts.

Essa complexidade teórica é agravada pela escassez de materiais de formação prática que abordem o DDD de forma didática e contextualizada. A maioria dos conteúdos disponíveis concentra-se

na implementação dos padrões táticos em frameworks específicos, negligenciando a fundamentação conceitual e estratégica da abordagem. Como consequência, muitos profissionais passam a adotar o DDD como um conjunto de receitas técnicas, sem internalizar seus princípios fundamentais, o que compromete a efetividade da abordagem (Vernon, 2016; Özkan, 2023).

O segundo eixo de dificuldades diz respeito às barreiras culturais e organizacionais. O DDD pressupõe uma cultura de colaboração intensa entre áreas técnicas e de negócio, baseada na escuta ativa, na co-construção do conhecimento e na experimentação contínua. No entanto, muitas organizações ainda operam sob modelos hierárquicos e fragmentados, nos quais a comunicação entre desenvolvedores e especialistas de negócio é mediada por múltiplos níveis de abstração, como analistas de sistemas, gerentes de produto ou equipes de requisitos. Essa estrutura dificulta a formação da linguagem ubíqua e impede que o modelo de domínio seja validado em tempo real com quem realmente compreende o negócio (Rapôso, 2025; Brandolini, 2015).

Além disso, o DDD exige tempo e espaço para reflexão coletiva, algo frequentemente incompatível com ambientes organizacionais orientados à entrega rápida, à maximização de produtividade e à minimização de custos. Sessões de modelagem colaborativa como EventStorming ou Domain Storytelling demandam investimento de horas e envolvimento de múltiplos stakeholders, o que pode ser percebido como custo improdutivo por lideranças não sensibilizadas quanto ao valor estratégico do domínio. Conforme destaca Vernon (2016), muitas tentativas de adoção do DDD fracassam não por razões técnicas, mas pela resistência cultural à colaboração interdisciplinar.

O terceiro conjunto de desafios está relacionado às dificuldades técnicas inerentes à aplicação dos padrões do DDD em ambientes de desenvolvimento reais. Um dos principais entraves refere-se à definição dos limites dos aggregates e à preservação de suas invariantes. Em sistemas complexos, com múltiplas entidades inter-relacionadas e regras de negócio cruzadas, torna-se difícil delimitar um aggregate que seja ao mesmo tempo coeso, performático e semanticamente estável. O risco de modelar aggregates excessivamente grandes, que comprometem o desempenho e a concorrência, ou excessivamente pequenos, que não protegem adequadamente as invariantes, é frequentemente reportado por equipes iniciantes (Evans, 2003; Vernon, 2013).

Outro problema técnico recorrente diz respeito à persistência e à serialização de objetos do domínio. Muitas plataformas de persistência relacional não se alinham naturalmente com os conceitos de value objects imutáveis ou com estruturas compostas como aggregates. Como resultado, os desenvolvedores enfrentam dificuldades na implementação de repositórios que respeitem as fronteiras do modelo e, ao mesmo tempo, sejam eficientes em termos de desempenho e escalabilidade. Essa tensão entre o modelo conceitual e o modelo de dados leva, em alguns casos, à duplicação de lógica, a workarounds não documentados e à degradação progressiva do modelo (Khononov, 2021; Jordanov, 2023).

As integrações entre bounded contexts também representam um ponto crítico. Embora o DDD ofereça um vocabulário rico para a definição de relações entre contextos como anticorruption layers, shared kernels e customer/supplier, a implementação desses padrões requer disciplina, infraestrutura e maturidade arquitetural.

Em muitos casos, as integrações são feitas de maneira direta, sem abstrações adequadas, por meio de chamadas síncronas de APIs, o que gera acoplamento excessivo, fragilidade e impacto em cascata nas mudanças. Essa situação compromete a autonomia dos contextos e vai de encontro aos princípios do design estratégico do DDD (Zimmermann et al., 2017; Özkan, 2023).

O quarto e último grupo de dificuldades diz respeito às lacunas metodológicas ainda existentes na abordagem do DDD. Embora diversas práticas tenham sido desenvolvidas para apoiar a modelagem colaborativa do domínio, como EventStorming, não há consenso sobre uma metodologia sistemática e replicável para a aplicação do DDD. A ausência de guias padronizados, métricas de avaliação e frameworks de maturidade dificulta a avaliação do progresso das equipes e a comparação entre diferentes implementações. Essa falta de estruturação metodológica leva a uma heterogeneidade de interpretações e à dificuldade de consolidar boas práticas (Rapôso, 2025).

Além disso, há uma escassez de estudos empíricos que investiguem de forma sistemática os efeitos do DDD em ambientes reais, especialmente com base em métricas quantitativas, como tempo de entrega, número de defeitos ou produtividade da equipe. A predominância de relatos qualitativos e estudos de caso isolados dificulta a generalização dos resultados e limita a construção de um corpo teórico robusto. Conforme aponta Özkan (2023), há uma lacuna na literatura científica que precisa ser preenchida com estudos comparativos, experimentos controlados e análises longitudinais que investiguem os impactos reais da adoção do DDD.

Na próxima subseção, será feita uma análise das variações metodológicas e adaptações contextuais identificadas na literatura, com foco em como o DDD tem sido reinterpretado e customizado em diferentes ambientes e estilos arquiteturais.

3.4 Variações Metodológicas e Adaptações Contextuais

O Domain-Driven Design (DDD), embora fundado em princípios bem definidos, não se configura como uma metodologia rígida ou prescritiva. Ao contrário, sua aplicação prática se caracteriza por um alto grau de flexibilidade, que permite que as equipes adaptem seus conceitos centrais às especificidades de seus contextos técnicos, organizacionais e culturais. Essa plasticidade metodológica, embora positiva por favorecer a aderência contextual, também gera uma ampla variedade de interpretações, variações e customizações. Esta subseção analisa essas variações metodológicas e adaptações contextuais com base nos achados da literatura técnica e científica, agrupando-as em quatro eixos: estilos arquiteturais, estratégias organizacionais, hibridizações com outras abordagens e ressignificações conceituais.

A primeira e mais visível linha de variação ocorre na relação entre o DDD e os estilos arquiteturais adotados pelas organizações. Em particular, a literatura aponta para uma forte associação entre DDD e arquiteturas de microserviços, conforme discutido anteriormente. Entretanto, essa associação nem sempre segue o modelo clássico proposto por Evans (2003). Em muitas implementações, o DDD é utilizado como um guia para a definição de fronteiras de serviços, sem que os demais elementos da abordagem sejam plenamente incorporados. Isso resulta em uma espécie de “DDD parcial”, focado na decomposição do sistema em serviços autônomos, mas com

pouca ênfase na linguagem ubíqua, na modelagem colaborativa ou nos padrões táticos (Vernon, 2016; Jordanov, 2023).

Além disso, há casos em que o DDD é aplicado em arquiteturas monolíticas ou híbridas, com o objetivo de modularizar subsistemas internamente por meio de bounded contexts, sem recorrer necessariamente à divisão em serviços distribuídos. Essa abordagem, embora menos visível nos discursos contemporâneos, tem demonstrado bons resultados em ambientes que não possuem maturidade ou infraestrutura para adotar microserviços. Estudos como o de Rapôso (2025) mostram que a modularização semântica promovida pelo DDD pode ser tão eficaz em sistemas monolíticos quanto em distribuídos, desde que os princípios da separação conceitual e da modelagem centrada no domínio sejam respeitados.

Outro eixo de variação metodológica refere-se às estratégias organizacionais adotadas para operacionalizar o DDD. Em contextos onde a organização já adota modelos de times autônomos, como os propostos por Skelton e Pais (2019) no Team Topologies, observa-se uma convergência natural entre bounded contexts e equipes stream-aligned, favorecendo a coerência entre a arquitetura do sistema e a estrutura da organização. No entanto, em ambientes mais tradicionais ou com estruturas matriciais, a adoção do DDD demanda adaptações significativas, como a formação de squads temporários para modelagem colaborativa ou a definição de papéis híbridos, como “especialista de domínio técnico”, que atuam como ponte entre desenvolvedores e áreas de negócio.

A prática revela também a hibridização do DDD com outras abordagens de design e desenvolvimento. Uma combinação frequente ocorre entre DDD e métodos ágeis, especialmente Scrum

e Kanban. Nessas configurações, o DDD é utilizado como suporte para o backlog refinement, a escrita de histórias de usuário e a definição de critérios de aceitação com base na linguagem ubíqua. A modelagem contínua do domínio torna-se, assim, parte do fluxo iterativo de entregas, contribuindo para que os incrementos de software reflitam com mais fidelidade as regras do negócio (Vernon, 2016; Brandolini, 2015).

Outra combinação crescente envolve DDD e práticas de DevOps. A separação de bounded contexts favorece o versionamento, a entrega contínua e o isolamento de falhas, enquanto a modelagem orientada a eventos contribui para a observabilidade e a rastreabilidade de processos. Em ambientes que utilizam pipelines de CI/CD, containers e infraestrutura como código, o DDD é reinterpretado como um framework organizacional para manter coerência e governança em arquiteturas altamente dinâmicas. Essa integração exige, contudo, disciplina metodológica e alinhamento entre arquitetura, operações e desenvolvimento, o que nem sempre está presente em organizações menos maduras (Zimmermann et al., 2017; Khononov, 2021).

Também são observadas ressignificações conceituais que adaptam os elementos do DDD a paradigmas específicos, como a programação funcional. Nesse contexto, conceitos como aggregates e domain services são reinterpretados como funções puras que operam sobre value objects imutáveis, promovendo clareza, previsibilidade e testabilidade. A abordagem funcional tende a enfatizar a transformação de dados em vez da manipulação de objetos com estado, o que leva à simplificação de modelos e à redução do acoplamento entre comportamentos e estrutura de dados. Autores como Khononov (2021) defendem que essa

abordagem torna o DDD mais compatível com arquiteturas orientadas a eventos e com ambientes de alta concorrência.

Em alguns casos, a ressignificação vai além da técnica e atinge o próprio entendimento dos objetivos do DDD. Algumas equipes optam por adotar apenas os princípios estratégicos da abordagem como a separação de contextos, o alinhamento com o negócio e a linguagem compartilhada, deixando de lado os padrões táticos em função de outras abordagens já consolidadas, como o Clean Architecture ou o CQRS. Essa prática não necessariamente descaracteriza o DDD, desde que os princípios fundamentais de modelagem orientada ao domínio sejam mantidos. Como aponta Vernon (2016), o DDD deve ser visto como um conjunto de orientações que podem ser aplicadas de forma gradual, seletiva e incremental.

A literatura também evidencia variações no uso das práticas colaborativas de modelagem. Enquanto algumas organizações adotam rigorosamente técnicas como EventStorming com envolvimento intenso de stakeholders, outras preferem abordagens mais enxutas, como a realização de entrevistas individuais com especialistas de negócio ou a construção incremental da linguagem ubíqua durante o desenvolvimento. Essa variação está fortemente ligada à cultura organizacional, à disponibilidade dos especialistas e à maturidade das equipes técnicas. O ponto comum, entretanto, é o reconhecimento da importância de envolver o conhecimento do domínio de forma estruturada no processo de design (Brandolini, 2015; Wicke et al., 2022).

Com esta subseção, conclui-se a etapa de desenvolvimento da pesquisa, que mapeou práticas, benefícios, desafios e variações da

implementação do DDD com base em fontes científicas e técnicas. A seguir, será apresentada a seção de Conclusão, na qual serão sintetizados os principais achados, discutidas suas implicações práticas e indicadas direções para futuras pesquisas.

4 Conclusão

A presente pesquisa exploratória teve como objetivo investigar, com base em literatura científica e técnica qualificada, como o Domain-Driven Design (DDD) tem sido implementado em diferentes contextos organizacionais, considerando suas práticas, benefícios, desafios e variações metodológicas. A partir da análise sistemática de estudos recentes e relatos empíricos, foi possível construir um panorama amplo e fundamentado da aplicação contemporânea do DDD, evidenciando tanto seu potencial transformador quanto as barreiras que limitam sua adoção plena.

Entre os principais achados, destaca-se a centralidade da linguagem ubíqua e da modelagem colaborativa como práticas estruturantes do DDD. Técnicas como EventStorming, Domain Storytelling e Example Mapping têm se mostrado eficazes para promover o alinhamento entre especialistas técnicos e de negócio, favorecer a construção de modelos expressivos e fortalecer a coesão entre código e domínio. Observa-se, contudo, que a eficácia dessas práticas está diretamente relacionada à maturidade das equipes e à cultura organizacional, exigindo espaços de diálogo, tempo para reflexão e compromisso com a aprendizagem interdisciplinar.

No plano arquitetural, a definição de bounded contexts e o uso de context maps revelaram-se fundamentais para a modularidade, a governança e a escalabilidade dos sistemas. Em especial, em

arquiteturas de microserviços, o DDD fornece princípios sólidos para a separação de responsabilidades, a redução de acoplamento e a clareza das integrações. No entanto, os dados analisados indicam que muitas implementações falham ao negligenciar os aspectos conceituais do DDD, aplicando a abordagem de forma parcial, fragmentada ou distorcida, o que compromete seus resultados.

Os benefícios relatados pela literatura incluem maior clareza conceitual, melhoria na comunicação entre equipes, aumento da manutenibilidade, fortalecimento da autonomia técnica e organizacional e sustentabilidade evolutiva dos sistemas. Tais impactos são observados em diferentes estilos arquiteturais, incluindo monolitos modulares, microserviços e sistemas baseados em eventos. Ressalta-se, ainda, a capacidade do DDD de fomentar a aprendizagem organizacional e de promover uma cultura orientada ao domínio, que valoriza o conhecimento tácito como ativo estratégico.

Por outro lado, a pesquisa identificou desafios relevantes que limitam a adoção efetiva do DDD, entre os quais se destacam: a complexidade conceitual da abordagem, a ausência de capacitação adequada, as limitações de ferramentas para modelagem, a dificuldade na definição e implementação de aggregates e integrações entre contextos, além de barreiras culturais e organizacionais à colaboração interdisciplinar. Também foi evidenciada a escassez de metodologias sistematizadas, métricas padronizadas e estudos empíricos robustos que orientem e avaliem a aplicação da abordagem em larga escala.

Em função dessa realidade, observam-se múltiplas variações e adaptações contextuais do DDD, que vão desde a sua hibridização

com práticas ágeis, DevOps e Clean Architecture, até sua ressignificação em paradigmas como a programação funcional. Essas variações, quando fundamentadas nos princípios originais da abordagem, demonstram sua flexibilidade e aplicabilidade em diferentes cenários. Contudo, também evidenciam a necessidade de diretrizes mais claras e instrumentos metodológicos que orientem sua adoção progressiva, especialmente em ambientes com baixa maturidade organizacional.

As implicações práticas deste estudo são múltiplas. Para profissionais e equipes técnicas, os resultados apontam para a importância de compreender o DDD como uma abordagem conceitual e estratégica, e não apenas como um conjunto de padrões técnicos. A adoção incremental, a modelagem colaborativa e a valorização da linguagem compartilhada são caminhos viáveis e sustentáveis para incorporar o DDD em diferentes realidades. Para gestores e líderes organizacionais, a pesquisa evidencia que o sucesso do DDD depende de mudanças culturais que promovam a integração entre tecnologia e negócio, bem como do investimento em capacitação, experimentação e tempo de modelagem.

Para a comunidade científica, esta pesquisa reforça a relevância do DDD como objeto de investigação e aponta lacunas que merecem ser exploradas em estudos futuros. Sugere-se a realização de pesquisas empíricas comparativas que avaliem o impacto do DDD em métricas de produtividade, qualidade e alinhamento estratégico, bem como estudos longitudinais que acompanhem a evolução de equipes na adoção da abordagem. Também são recomendadas investigações sobre práticas de formação, ferramentas de apoio à modelagem e formas de avaliação da maturidade em DDD.

Conclui-se, portanto, que o Domain-Driven Design permanece uma abordagem valiosa para o desenvolvimento de sistemas centrados no domínio, especialmente em contextos de alta complexidade e mudança constante. Sua implementação bem-sucedida, no entanto, exige não apenas domínio técnico, mas também sensibilidade organizacional, compromisso com a aprendizagem contínua e disposição para construir sistemas que reflitam, de forma precisa e evolutiva, os processos, regras e valores do mundo real que se pretende representar.

REFERÊNCIAS BIBLIOGRÁFICAS

Brandolini, A. (2015). Introducing EventStorming: An act of deliberate collective learning. Leanpub.

Conway, M. E. (1968). How do committees invent? Datamation, 14(4), 28–31.

Evans, E. (2003). Domain-Driven Design: Tackling complexity in the heart of software. Addison-Wesley.

Fowler, M. (2015). Microservices: A definition of this new architectural term. Retrieved from https://martinfowler.com/articles/microservices.html

Jordanov, J. (2023). Domain-Driven Design Approaches in Cloud-Native Applications. TEM Journal, 12(4), 1985–1994. https://doi.org/10.18421/TEM124-50

Khononov, V. (2021). Learning Domain-Driven Design: Aligning software architecture and business strategy. O'Reilly Media.

Özkan, O., Babur, Ö., & van der Brand, M. (2023). Domain-Driven Design in software development: A systematic literature review on implementation, challenges, and effectiveness. ACM Computing Surveys. https://doi.org/10.1145/3574886

Rapôso, C. F. L. (2025). Domain-Driven Design: revisão sistemática da literatura, lacunas e direções futuras. Revista Tópicos em Engenharia de Software, 11(1), 1–23.

Skelton, M., & Pais, M. (2019). Team Topologies: Organizing business and technology teams for fast flow. IT Revolution Press.

Vernon, V. (2013). Implementing Domain-Driven Design. Addison-Wesley.

Vernon, V. (2016). Domain-Driven Design Distilled. Addison-Wesley.

Wicke, C., Becker, S., & Döllner, J. (2022). Domain Storytelling in practice: Enabling collaborative domain modeling. Information and Software Technology, 141, 106746. https://doi.org/10.1016/j.infsof.2021.106746

Zimmermann, O., Zdun, U., Stocker, M., & Koehler, J. (2017). Context- and stakeholder-specific architecture viewpoints for software-intensive systems. Software and Systems Modeling, 16(1), 3–35. https://doi.org/10.1007/s10270-015-0498-8

---

[1] Bacharel em Engenharia de Produção pela Faculdade Estácio do Recife, Master in Business Administration pela Atlanta College of Liberal Arts and Sciences e Estudante em Master of Science in

Business Administration pela Must University. E-mail: acesse o artigo original para visualizar o e-mail