DOMAIN-DRIVEN DESIGN
DESTILADO

VAUGHN VERNON

ALTA BOOKS
GRUPO EDITORIAL
Rio de Janeiro, 2024

Sumário

Prefácio

Por que a construção de modelos é uma atividade tão divertida e compensadora? Eu amo construir modelos desde criança. Naquela época, geralmente construía modelos de carros e aviões. Não sei onde estava a LEGO naqueles tempos. Ainda assim, ela fez parte da vida do meu filho desde muito jovem. É tão fascinante idealizar e construir modelos com aquelas pequenas peças. É fácil criar modelos básicos, e parece possível estender a criatividade de forma quase ilimitada.

Você provavelmente deve se lembrar de algum tipo de construção de modelos na infância.

Os modelos se aplicam a muitas situações na vida. Se você gosta de jogos de tabuleiro, está usando modelos. Pode ser um modelo de bens imobiliários e proprietários de imóveis, de ilhas e sobreviventes, territórios e construção, e muito mais. Da mesma forma, videogames são modelos. Talvez eles modelem um mundo de fantasia com personagens extravagantes em papéis fantásticos. Um baralho de cartas e jogos desse tipo modelam poder. Utilizamos modelos o tempo todo e com tanta frequência que nem sequer lhes damos o devido reconhecimento. Os modelos simplesmente fazem parte da nossa vida.

Mas por quê? Todo mundo tem um estilo de aprendizagem. Existe uma grande quantidade de estilos de aprendizagem, mas três dos estilos mais mencionados são o auditivo, o visual e o tátil. Os aprendizes auditivos aprendem ouvindo e escutando. Os aprendizes visuais aprendem lendo ou vendo imagens. Os aprendizes táteis aprendem fazendo algo que envolva o toque. É interessante notar que cada estilo de aprendizagem é muito favorecido pelo indivíduo a partir do nível de dificuldade que ele ou ela tenha com os outros tipos de aprendizagem. Aprendizes táteis, por exemplo, têm facilidade de se lembrar do que fizeram, mas podem ter dificuldade de lembrar do que foi dito durante o processo. Com a construção de modelos, podemos imaginar que aprendizes visuais e táteis teriam uma enorme vantagem sobre os aprendizes auditivos, pois a construção de modelos parece envolver muito estímulo visual e de toque. Entretanto, isso nem sempre é verdade, especialmente se uma equipe de

construtores de modelos usa a comunicação audível no processo de construção. Em outras palavras, a construção de modelos nos dá a possibilidade de acomodar o estilo de aprendizagem da maioria dos indivíduos.

Com a afinidade natural que temos com o aprendizado por meio da construção de modelos, por que não teríamos o desejo natural de modelar os softwares que vêm nos ajudando e influenciando cada vez mais nossa vida? Na verdade, modelar softwares parece ser, bem, humano. E deveríamos fazer isso. Acho que os humanos devem ser construtores de modelos de software de elite.

É meu grande desejo ajudá-lo a ser o mais humano possível ao fazer a modelagem de software, utilizando algumas das melhores ferramentas disponíveis para isso. Essas ferramentas são inclusas em uma categoria que recebe o nome de "Domain-Driven Design" ou DDD. Essa caixa de ferramentas, que é, na verdade, um conjunto de padrões, foi codificada pela primeira vez por Eric Evans no livro *Domain-Driven Design: Atacando a complexidade no coração do software* [DDD]. Minha visão é trazer o DDD ao máximo de pessoas possível. Para deixar bem claro, se eu precisar dizer que quero levar o DDD às massas, que assim seja. É onde o DDD merece estar, e o DDD é a caixa de ferramentas que os seres humanos orientados a modelos merecem usar para criar modelos de software mais avançados. Com este livro, estou determinado a tornar o aprendizado e o uso do DDD simples e fácil e levar esse aprendizado ao público mais amplo possível.

Para aprendizes auditivos, o DDD oferece a perspectiva de aprendizagem pela comunicação da equipe de construção de um modelo baseado no desenvolvimento de uma *Linguagem Ubíqua*. Para alunos visuais e táteis, o processo de utilização das ferramentas do DDD é muito visual e prático, pois sua equipe modela tanto estratégica quanto taticamente. Isso se mostra verdade especialmente ao elaborarmos *Mapas de Contexto* e modelar o processo de negócio usando *Tempestade de Eventos*. Assim, acredito que o DDD pode apoiar a todos que queiram aprender e alcançar a grandeza por meio da construção de modelos.

Para Quem É Este Livro?

Este livro é para todos os interessados em aprender sobre os aspectos e as ferramentas mais importantes do DDD e em aprender rapidamente. Os leitores mais comuns são arquitetos e desenvolvedores de software

que colocarão o DDD em prática em seus projetos. Muito frequentemente, os desenvolvedores de software rapidamente descobrem a beleza do DDD e se sentem atraídos por suas poderosas ferramentas. Mesmo assim, tornei o assunto compreensível para os executivos, especialistas de domínio, gerentes, analistas de negócios, arquitetos de informação e testadores. Realmente, não há limite para aqueles que estão na indústria da tecnologia da informação (TI) e nos ambientes de pesquisa e desenvolvimento (P&D), que podem se beneficiar da leitura deste livro.

Se você for consultor e estiver trabalhando com um cliente para o qual recomendou o uso do DDD, recomende este livro como forma de acelerar o processo de atualização das principais partes interessadas. Se tiver desenvolvedores — talvez de nível júnior ou médio ou mesmo de nível sênior — trabalhando no seu projeto que não estão familiarizados com o DDD, mas que precisarão usá-lo muito em breve, certifique-se de que eles leiam este livro. Se fizerem isso, no mínimo, todos os interessados e desenvolvedores do projeto terão o vocabulário e compreenderão as principais ferramentas do DDD que estão sendo utilizadas. Isso lhes permitirá compartilhar as coisas de forma significativa à medida que o projeto avança.

Independentemente do seu nível de experiência ou papel, leia este livro e depois coloque o DDD em prática em um projeto. Em seguida, releia este livro e veja o que pode aprender com suas experiências e onde você pode melhorar no futuro.

O Que Este Livro Abrange

O primeiro capítulo, "O DDD para Mim", explica o que o DDD pode fazer por você e sua organização e fornece uma visão mais detalhada do que será aprendido e por que isso é importante.

O Capítulo 2, "Design Estratégico com Contextos Delimitados e Linguagem Ubíqua", apresenta o design estratégico e ensina sobre os fundamentos do DDD, os *Contextos Delimitados* e a *Linguagem Ubíqua*. O Capítulo 3, "Design Estratégico com Subdomínios", explica os *Subdomínios* e como você pode usá-los para lidar com a complexidade de integração com os sistemas existentes à medida que modela suas novas aplicações. O Capítulo 4, "Design Estratégico com Mapeamento de Contexto", ensina a variedade de formas em que as equipes trabalham em conjunto e de modo estratégico e como seu software pode se integrar. Isso é chamado de *Mapeamento de Contexto*.

O Capítulo 5, "Design Tático com Agregados", volta sua atenção para a modelagem tática com *Agregados*. Uma importante e poderosa ferramenta de modelagem tática a ser usada com os *Agregados* são os *Eventos de Domínio*, tema do Capítulo 6, "*Design Tático com Eventos de Domínio*".

Finalmente, no Capítulo 7, "Ferramentas de Aceleração e Gestão", o livro destaca algumas ferramentas de aceleração e gestão de projeto que podem ajudar as equipes a estabelecer e manter um bom ritmo. Esses dois tópicos raramente são discutidos em outras fontes de conteúdo sobre DDD e são extremamente necessários para quem está determinado a colocar o DDD em prática.

Convenções

Há apenas algumas convenções que devemos ter em mente ao ler este livro. Todas as ferramentas de DDD sobre as quais discorro estão impressas em itálico. Por exemplo, você lerá sobre *Contextos Delimitados* e *Eventos de Domínio*. Outra convenção é que todo código-fonte é apresentado na fonte Courier monoespaçada. Siglas e abreviaturas para as obras listadas nas Referências aparecem entre colchetes ao longo dos capítulos.

Mesmo assim, o que este livro mais enfatiza, e do que seu cérebro gostará muito, é a aprendizagem visual com muitos diagramas e figuras. Você notará que não há números de figuras no livro, pois não queria distraí-los com eles. Em todos os casos, as figuras e os diagramas precedem o texto que as discute, o que significa que os visuais gráficos introduzem pensamentos à medida que você lê o livro. Isso significa que, quando estiver lendo o texto, você poderá contar com a referência à figura ou ao diagrama anterior como auxílio visual.

Agradecimentos

Este já é meu terceiro livro dentro do estimado selo da Addison-Wesley. É também a terceira vez que trabalho com meu editor, Chris Guzikowski, e com o editor de desenvolvimento, Chris Zahn, e tenho o prazer de dizer que a terceira vez foi tão encantadora quanto as duas primeiras. Mais uma vez, obrigado por ter escolhido publicar os meus livros.

Como sempre, um livro não pode ser escrito e publicado com sucesso sem feedback crítico. Desta vez, recorri a uma série de profissionais de DDD que não necessariamente ensinam ou escrevem sobre isso, mas que trabalham em projetos ao passo que ajudam outros a utilizar essa poderosa caixa de ferramentas. Senti que esse tipo de profissionais era crucial para garantir que este material agressivamente destilado dissesse exatamente o que fosse necessário e da forma correta. A ideia é que, se quiser que eu fale por 60 minutos, dê-me 5 minutos para me preparar; se quiser que eu fale por 5 minutos, dê-me algumas horas para me preparar.

Em ordem alfabética de sobrenome, as pessoas que mais me ajudaram foram Jérémie Chassaing, Brian Dunlap, Yuji Kiriki, Tom Stockton, Tormod J. Varhaugvik, Daniel Westheide e Philip Windley. Muito obrigado!

Sobre o Autor

Vaughn Vernon é um desenvolvedor de software veterano e líder de pensamento na simplificação do design e implementação de software. Ele é o autor dos best-sellers *Implementando Domain-Driven Design* e *Reactive Messaging Patterns with the Actor Model* [sem publicação no Brasil]. Já ministrou sua Oficina IDDD em todo o mundo para centenas de desenvolvedores de software. Vaughn é palestrante frequente em conferências do setor. Seu maior interesse é na computação distribuída, em mensagens e, principalmente, no Modelo de Atores. Vaughn é especialista em consultoria de Domain-Driven Design e DDD utilizando o Modelo de Atores com Scala e Akka. Você pode acompanhar os trabalhos mais recentes de Vaughn lendo seu blog no endereço: www.VaughnVernon.com [conteúdo em inglês] e seguindo sua conta no Twitter: @VaughnVernon.

Capítulo 1

O DDD para Mim

Você quer aprimorar seu ofício e ser mais bem-sucedido nos projetos. Está ansioso para ajudar sua empresa a competir em novos patamares usando os softwares que cria. Quer implementar um software que não só modele corretamente as necessidades de sua empresa, mas que também atue em grande escala utilizando as mais avançadas arquiteturas de software. Aprender sobre o *Domain-Driven Design* (DDD) pode ajudá-lo a realizar rapidamente tudo isso e mais.

O DDD é um conjunto de ferramentas que auxiliam na concepção e implementação de softwares que oferecem alto valor, tanto estratégica como taticamente. Sua organização não pode ser a melhor em tudo, por isso é melhor escolher cuidadosamente no que deve se destacar. As ferramentas de desenvolvimento estratégico do DDD ajudam você e sua equipe a fazer as melhores escolhas de design de software de forma competitiva e a tomar boas decisões de integração para sua empresa. Sua organização se beneficiará mais dos modelos de software que refletem explicitamente suas competências centrais. As ferramentas de desenvolvimento tático do DDD podem ajudar você e sua equipe a projetar softwares úteis que modelam com precisão as operações específicas da empresa. Sua organização deverá se beneficiar das amplas opções para implantar suas soluções em uma variedade de infraestruturas, seja localmente ou na nuvem. Com o DDD, você e sua equipe poderão realizar os projetos e implementações de software mais eficazes necessários para o sucesso no atual cenário empresarial competitivo.

Neste livro, eu destilei o DDD para você, com o tratamento condensado das ferramentas de modelagem estratégica e tática. Compreendo as demandas particulares do desenvolvimento de software e os desafios que enfrentamos ao trabalhar para melhorar nosso ofício em um setor cujo ritmo é acelerado. Nem sempre é possível passar meses lendo sobre um assunto como o DDD, ainda mais se queremos pôr o DDD em prática o mais rápido possível.

Sou o autor do best-seller *Implementando Domain-Driven Design* [IDDD], e também criei e ministro a Oficina IDDD, que dura três dias. E agora também escrevi este livro para lhe trazer o DDD de forma agressivamente condensada. Tudo isso faz parte do meu compromisso de levar o DDD a todas as equipes de desenvolvimento de software, onde merece estar. Meu objetivo, naturalmente, inclui trazer o DDD até vocês.

O DDD Vai Doer?

Você já deve ter ouvido falar que o DDD é uma abordagem complicada ao desenvolvimento de software. Complicada? Decerto, não necessariamente complicada. Na verdade, o DDD é um conjunto de técnicas avançadas a serem utilizadas em projetos de software complexos. Devido ao seu poder e ao quanto é preciso aprender, sem instruções de especialistas, pode ser assustador pôr o DDD em prática por conta própria.

Você provavelmente já descobriu também que alguns dos outros livros de DDD têm muitas centenas de páginas e estão longe de ser fáceis de consumir e executar. Foram necessárias muitas palavras para explicar o DDD em detalhes, a fim de fornecer uma referência de implementação abrangente sobre mais de uma dúzia de tópicos e ferramentas do DDD. Esse esforço resultou no *Implementando Domain-Driven Design* [IDDD]. Este livro novo e mais condensado foi escrito para familiarizá-lo com as partes mais importantes do DDD da maneira mais simples e rápida possível. Por quê? Porque algumas pessoas se sentem sobrecarregadas com os textos maiores e precisam de um guia destilado para ajudá-las a dar os primeiros passos para aderir ao DDD. Descobri que quem utiliza o DDD revisita a literatura sobre o assunto várias vezes. Na verdade, você pode até concluir que nunca aprenderá o suficiente, de modo que precisará utilizar este livro como referência rápida e consultar outros livros para maiores detalhes conforme aprimora seu trabalho. Outras pessoas têm tido dificuldade de convencer seus colegas e sua equipe administrativa a usar o DDD. Este livro o ajudará, não apenas por explicar o DDD em um formato condensado, mas também mostrando que existem ferramentas disponíveis para acelerar e administrar seu uso.

Naturalmente, não é possível ensinar tudo sobre o DDD neste livro, pois destilei propositadamente as técnicas de DDD para você. Para uma abordagem mais aprofundada, veja meu livro *Implementando Domain-Driven Design* [IDDD] e considere participar da minha Oficina IDDD. Esse curso intensivo de três dias, que tenho ministrado no mundo todo para um amplo público de centenas de desenvolvedores, vai ajudá-lo a se informar rapidamente sobre o DDD. Também realizo treinamento de DDD online pelo site http://ForComprehension.com [conteúdo em inglês].

A boa notícia é que o DDD não tem que doer. Como você provavelmente já lida com a complexidade de seus projetos, poderá aprender a usar o DDD para reduzir a dor de vencer a complexidade.

O Design Bom, o Ruim e o Eficiente

Muitas vezes, as pessoas falam sobre design bom e design ruim. De qual tipo você projeta? Diversas equipes de desenvolvimento de software nem sequer pensam no design. Em vez disso, elas executam o que chamo de "embaralhamento do quadro de tarefas". Nele, a equipe tem uma lista

de tarefas de desenvolvimento, com backlog de tarefas pendentes de produtos Scrum, onde simplesmente movem uma nota adesiva da coluna "Para fazer" para a coluna "Em andamento". A criação do backlog de tarefas pendentes e a realização do "embaralhamento do quadro de tarefas" constituem a totalidade dos cuidadosos insights sobre o assunto, e o resto é deixado para os heróis da codificação, à medida que os programadores criticam a fonte. Isso raramente funciona tão bem quanto deveria, e o custo para o negócio é geralmente o preço mais alto pago por tais designs inexistentes.

Isso costuma acontecer devido à pressão para entregar versões de software em um prazo inflexível, em que a gerência usa Scrum principalmente para controlar o cronograma em vez de permitir um dos princípios mais importantes do Scrum: *a aquisição de conhecimento*.

Quando faço consultoria ou dou palestras, algumas em empresas individuais, geralmente encontro as mesmas situações. Os projetos de software estão em perigo, e equipes inteiras são contratadas para manter os sistemas em funcionamento, corrigindo o código e os dados diariamente. Seguem alguns dos problemas insidiosos que encontro e que, curiosamente, o DDD pode facilmente ajudar as equipes a evitar. Vou começar com os problemas comerciais de nível mais alto e passar para os mais técnicos:

- O desenvolvimento de software é considerado um centro de custos em vez de um centro de lucro. Geralmente, isso acontece porque as empresas veem os computadores e o software como incômodos necessários em vez de fontes de vantagem estratégica. (Infelizmente pode não haver cura para isso se a cultura empresarial for firmemente fixada.)
- Os desenvolvedores estão muito envolvidos com a tecnologia e, em vez do design e do pensamento detalhado, preferem tentar usá-la para resolver problemas. Isso faz com que os desenvolvedores estejam constantemente perseguindo novos "objetos brilhantes", que são os mais recentes modismos da tecnologia.
- O banco de dados tem prioridade exagerada, e a maioria das discussões sobre as soluções giram em torno do banco de dados e de um modelo de dados em vez de processos e operações comerciais.
- Os desenvolvedores não dão a devida ênfase à nomeação de objetos e operações de acordo com o propósito do negócio que atendem. Isso leva a um grande abismo entre o modelo mental que o negócio possui e o software que os desenvolvedores entregam.

- O problema anterior geralmente é resultado da má colaboração com o negócio. Muitas vezes, as partes interessadas dos negócios também gastam muito tempo em trabalhos isolados, produzindo especificações que ninguém utiliza ou que são apenas parcialmente consumidas pelos desenvolvedores.
- As estimativas do projeto estão em alta demanda e, muito frequentemente, sua produção pode acrescentar tempo e esforço consideráveis, resultando no atraso de entrega do software. Os desenvolvedores usam o "embaralhamento do quadro de tarefas" no lugar de um design cuidadoso. Eles produzem uma *Grande Bola de Lama* (discutida nos capítulos seguintes) em vez de segregar os modelos de forma apropriada, de acordo com os drivers de negócios.
- Os desenvolvedores incorporam a lógica de negócios nos componentes da interface do usuário e nos componentes de persistência. Além disso, os desenvolvedores frequentemente realizam operações de persistência no meio da lógica de negócios.
- Há consultas quebradas, lentas e bloqueadas de bancos de dados que impedem os usuários de realizar operações de negócio urgentes.
- Há abstrações erradas, em que os desenvolvedores tentam atender a todas as necessidades atuais e imaginárias futuras, generalizando excessivamente soluções em vez de atender às necessidades concretas e reais do negócio.
- Existem serviços fortemente acoplados, em que uma operação é realizada em um serviço, que liga diretamente para outro serviço para causar uma operação de equilíbrio. Esse acoplamento costuma resultar em processos de negócios quebrados e dados não reconciliados, sem contar os sistemas que são muito difíceis de manter.

Isso tudo parece acontecer no espírito de "nenhum design rende um software de baixo custo". E, muitas vezes, é simplesmente uma questão de negócios e de os desenvolvedores de software não saberem que existe uma alternativa muito melhor. "O software está devorando o mundo" [WSJ], e o fato de que o software também pode devorar seus lucros ou alimentá-los com um banquete também deve ser importante para você.

É importante entender que a economia imaginada do "No Design" é uma falácia que astutamente enganou as pessoas que insistem na produção de software sem um design cuidadoso. Isso porque o design ainda flui do cérebro dos desenvolvedores individuais pelas pontas de seus dedos, enquanto eles se confrontam com o código, sem nenhuma