Resumo do livro "Domain-Driven Design"

Leonardo Leite

14 de julho de 2016

Atenção: este documento trata-se ainda de uma versão em desenvolvimento.

1 Introdução

Este documento é um resumo do livro "Domain-Driven Design, Atacando as Complexidades no Coração do Software", escrito por Eric Evans em 2003. Os objetivos deste resumo são:

- Em primeiro lugar, fazer um exercício de fixação e assimilação de minha leitura do livro.
- Servir de referência rápida para que eu tente aplicar os conceitos do DDD no desenvolvimento de software.
- Fornecer a outras pessoas uma ideia rápida sobre o que trata o livro Domain-Driven Design, assim como apresentar alguns de seus principais conceitos.

Uma consideração importante ao leitor: embora minha prática do desenvolvimento de software já tenha vários pontos de acordo com o DDD, este resumo foi escrito logo após minha leitura do livro. Ou seja, não se trata de um texto baseado em minha experiência com DDD, mas somente na leitura do livro.

Destaco também que o livro que eu li foi a primeira edição da tradução para o português (2009). Aliás, uma tradução não tão boa. O pior exemplo de tradução que achei foi a tradução de *graphs* (grafos) e *edges* (arestas) respectivamente por gráficos e bordas. Não sei dizer se a tradução em si foi melhorada na segunda edição da tradução (2011).

Minha impressão geral sobre o livro: gostei bastante da mentalidade do autor, bem condizente com princípios da Programação Extrema (XP) e ao mesmo tempo com uma boa dose de pragmatismo. No entanto o livro é um tanto quanto grande e verboso. Embora seja definitivamente um clássico, já é relativamente antigo (2003). Dessa forma, acho que talvez possam existir por aí livros mais resumidos que possam transmitir a essência do DDD sem tantas considerações e mais direto ao ponto considerando o atual estado-da-arte do desenvolvimento de software. Não que o livro tenha deixado de ser atual, mas algumas considerações menores poderiam hoje ser deixadas de lado. Ao mesmo tempo, já surgiram alguns paradigmas modernosos como *micro-serviços*, que são bem relacionados com o tema do livro. Portanto, sua decisão de ler ou não o livro vai depender bastante do quanto você deseja se aprofundar no assunto.

Outra consideração bem interessante é, pra mim, foi bem positivo ter demorado pra ler esse livro. Já tinha ouvido falar do livro há alguns anos, mas somente agora finalmente

o priorizei. E isso foi bom porque a experiência recente que eu tive desenvolvendo um sistema com um complicado modelo de domínio me ajudou bastante a ler o livro com melhor proveito. Outros sistemas em que trabalhara antes não tinham um modelo de domínio assim tão complexo. Ler o livro e comparar o que o autor descreve com suas próprias experiências é de grande valor para uma melhor leitura do livro. Dessa forma, dou a ousada sugestão de que se você ainda é um desenvolvedor iniciante leia apenas um livro básico (ou mesmo esse resumo) sobre o assunto e espere alguns anos até ler o livro Domain-Driven Design.

Bom, chega de enrolação. Vamos ao que interessa...

2 A linguagem onipresente

Um modelo é uma simplificação da realidade. O modelo do domíno de um sistema é a forma como entedemos a realidade de um dado negócio. Esse modelo é algo abstrato, que tem impacto em diversos artefatos concretos, como requisitos, diagramas, o código-fonte e a estrutura do banco de dados. Todos esses artefatos devem estar alinhados com o modelo do domínio. E parte desse alinhamento diz respeito às palavras utilizadas.

Evans defende extensivamente ao longo do livro a utilização de uma *linguagem onipresente* para falar do modelo e para se aplicar nos artefatos concretos. Assim, ocorre uma melhor assimilação dos desenvolvedores sobre o que dizem os especilistas dos negócios. Assim, fica evidente desvios que a arquitetura esteja tomando em relação ao negócio. A utilização de uma linguagem onipresente é uma ferramenta realmente poderosa.

3 Camadas de um software

As camadas de um software geralmente se resumem a interface, aplicativo, domínio e infra.

Interface:

Aplicativo: não contém regras de negócio. Tende a ser bem magra. Essa camada utiliza a camada do domínio.

Domínio: é o coração do sistema, onde estão as regras do negócio. É a parte do sistema onde menos os *frameworks* e plataformas podem ajudar. É onde mais a criatividade e as capacidades de análise e design se fazem necessárias. E é nessa camada que o livro Domain-Driven Design é focado.

Infra: ecursos técnicos como mensagens, persistência etc.

Muitas pessoas exaltam muito a importância da infra-estrutura na definição de arquitetura de um software. Alguns acham que a escolha do Sistem Gerenciador de Bancos de Dados é a mais importante. Algumas outras se degladiam pela escolha do *framework* de desenvolvimento. Mas o Uncle Bob é bem feliz ao explicar como essas coisas na realidade importam muto pouco[1]. Assim, entendemos que o núcleo do sistema é composto pelas regras de negócio. Essas regras tendem a ser mais duradouras que as tecnologias empregadas. E são essas regras que, no fim, das contas, entregam valor ao negócio. Por

[1] http://blog.cleancoder.com/uncle-bob/2016/01/04/ALittleArchitecture.html

isso a importância do estudo do DDD, que consiste no aprimoramento de nossa capacidade em melhor desenvolver essa cadama do domínio.

Uncle bob também esclare como a camada de domínio não deve depender da camada de infra. O negócio não pode depender da tecnologia. Isso fica evidente na apresentação da arquitetura hexagonal[2].

4 Elementos do domínio

entidade (tem identidade), objetos de valor e serviços.

5 Padrões do domínio

agregados, fábricas e repositórios.

6 Arquitetura de larga escala

7 Objetos não são estruturas de dados

Modelo anêmico (post caelumn)

[2]https://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html