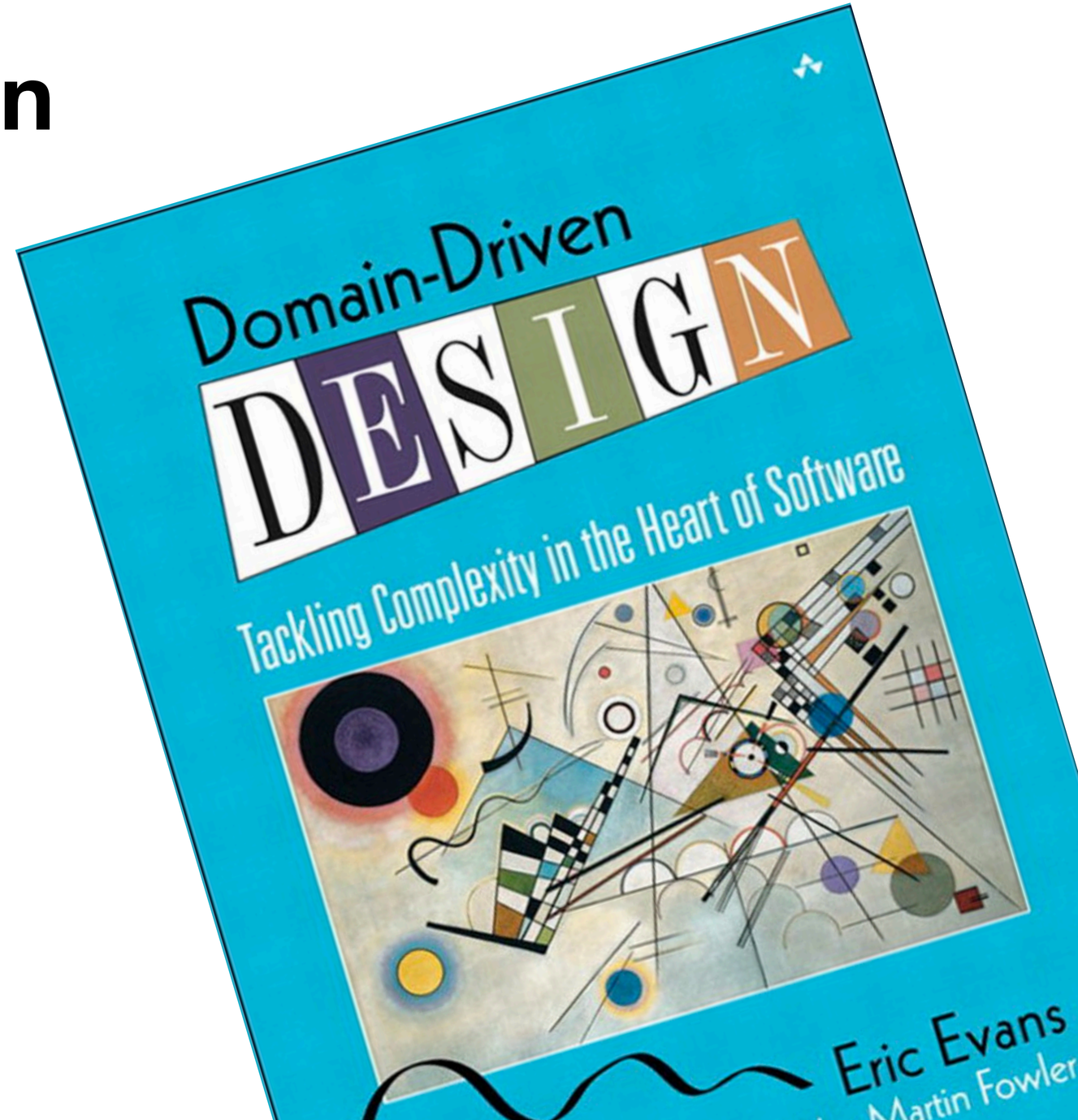
Domain-Driven Design

to the moon and back



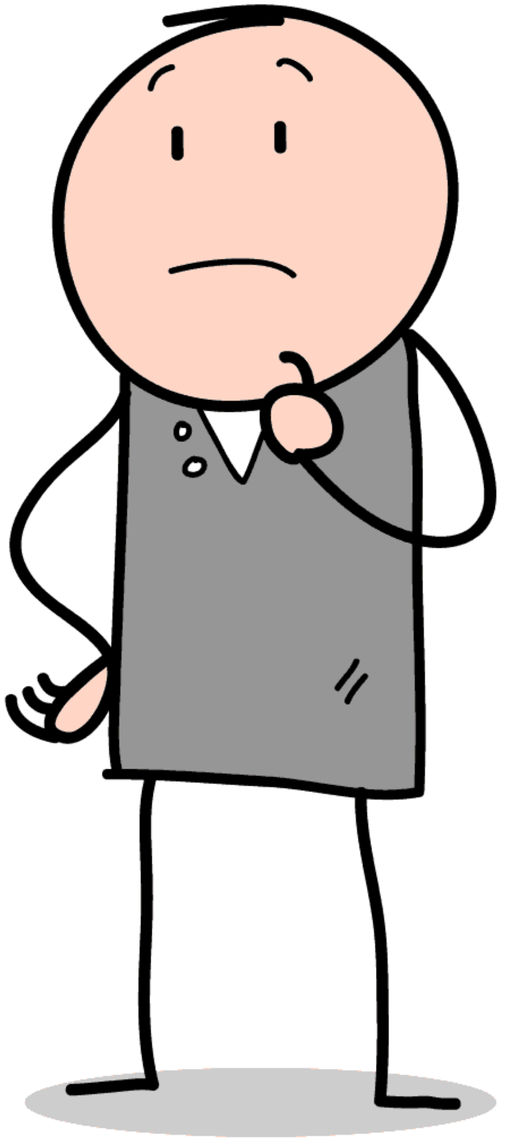
Marcelo T. dos Santos, 2021

Agenda

- Introdução à Domain-Driven Design
- Detalhamento do nosso contexto
- Apresentação de nossos problemas
- Aprofundamento nos “building blocks” que auxiliam na expressão de um modelo em código
- Demonstração dos conceitos através de uma aplicação

O que é DDD?

Domain-Driven Design define uma abordagem para o desenvolvimento de sistemas de software que expressam grande conhecimento dos processos e regras do domínio sendo modelado. Esta abordagem é descrita através de padrões independentes da tecnologia e paradigma a serem utilizados na sua implementação.



“Domain-Driven Design is an approach to software development that centers the development on programming a domain model that has a rich understanding of the processes and rules of a domain.”

Martin Fowler

“Um padrão é uma regra de três partes que expressa a relação entre um contexto (1), um problema (2) e uma solução (3).”

Daniel cukier

Define o escopo de um

[Problema]

Nós gostaríamos de ter Y, ou estar numa posição Y.

Impõe uma

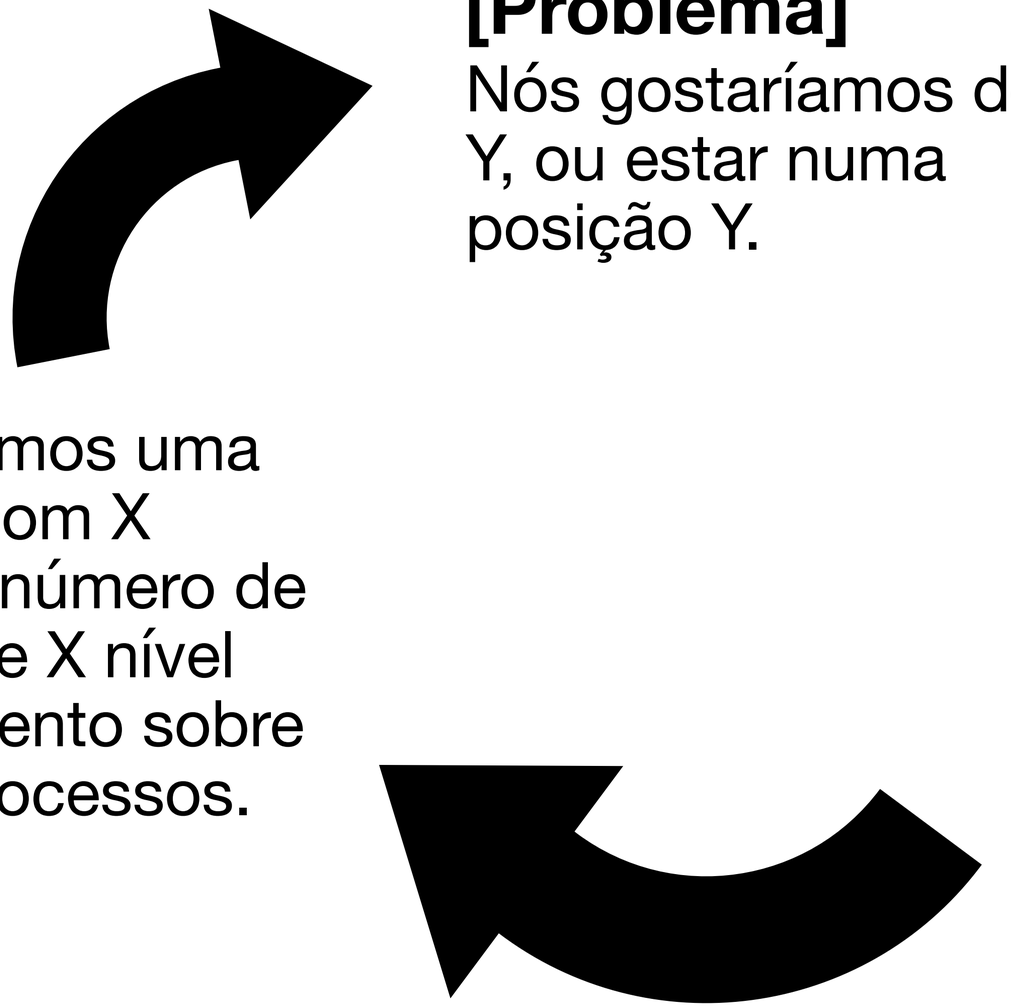
[Solução]

E, portanto, iremos implementar Z para atingir Y dado o contexto X em que estamos.

Traz um novo

[Contexto]

Dado que somos uma empresa X, com X ambições, X número de funcionários e X nível de conhecimento sobre os nossos processos.



PONTOS CHAVE DO LIVRO

- O código tem valor em tempo de execução e em tempo de análise (implementação).
- Unificar o time de desenvolvimento e negócios — adotando uma linguagem comum aos especialistas e desenvolvedores que será utilizada para descrever o domínio, seus problemas e soluções.
- Estabelecer o foco na complexidade do problema a ser solucionado. Decisões técnicas são então uma consequência da evolução do modelo de negócio e definem formas concretas de resolver problemas que surgem durante este processo.
- Modelos serão frequentemente substituídos, modificados, e até mesmo completamente descartados durante a evolução de um projeto.
- Organizar o software de forma alinhada com a estrutura organizacional que o sustenta para comportar a evolução do projeto.

CONTEXTO DO PROJETO

- Nossos sistemas são utilizados como fonte de informação para diversos grupos de colaboradores e também outros sistemas de software.
- Usuários acessam os nossos sistemas de diferentes localidades e fuso-horários, e apresentam diferentes níveis de experiência com determinadas tecnologias.
- A nossa equipe de desenvolvedores e especialistas está em constante rotação e, portanto, a fonte de conhecimento sobre os processos não deve residir somente nas pessoas.
- Já existe uma estrutura organizacional bem estabelecida na colaboração para conduzir implementações em diferentes setores de interesse (i.e., Authorship Committee, Publication Committee, Speakers Committee).

SECRETARIAT

- Envolvido em praticamente todos os processos burocráticos envolvendo membros e instituições.
- Têm pouca participação na tomada de decisão, porém são ávidos usuários dos nossos sistemas.
- Registro de membros, instituições, extensão de contratos, entre outras atividades são parte do dia-a-dia deste setor.

PUBLICATION COMMITTEE

- Responsável, principalmente, pelo desenvolvimento dos sistemas da família “Analysis”.
- Estão envolvidos no processo de publicação de resultados dos trabalhos de diversos grupos da colaboração.
- Composto por diferentes iniciativas como, por exemplo, o comitê Physics Office — responsável pelo desenvolvimento das ferramentas de integração contínua que utilizam o GitLab como plataforma.
- Organização, operação e comunicação sobre as atividades de publicação são o foco deste departamento.

AUTHORSHIP COMMITTEE

- Responsável pelas policies que governam o processo de qualificação e gerenciamento de autorias dentro da colaboração.
- Em recente contato com o time, apresentaram diversas ideias de projetos que podem se tornar prioridade.
- Apresenta um domínio super interessante e pouco explorado :(
- Estão sempre disponíveis para discussões e se mostram bastante ativos nas nossas atividades.

SPEAKERS COMMITTEE

- O domínio mais complexo na minha opinião :s
- Em resumo, se apoia em ferramentas para organização de talks e conferências, bem como a produção de pesquisas sobre a produção de conteúdo dentro de colaboração.
- Dependem fortemente em nossas ferramentas de busca para analisar a equidade na escolha de palestrantes.
- Alguns grupos de física se organizam em sub-comitês focados em conferências internas e apresentam diferentes necessidades do comitê geral.

DESAFIOS

- Regras e processos importantes para o bom funcionamento dos sistemas estão escondidos na base de código.
- A comunicação entre desenvolvedores e especialistas enfrenta grandes barreiras de tradução.
- Pouca interação com os usuários sobre a relevância das funcionalidades de cada sistema.
- Alto nível de acoplamento entre as implementações e o framework utilizado (FENCE).
- Dificuldade em implementar novas soluções devido à necessidade de grandes modificações de código.

PRIMEIROS PASSOS

- Adotar uma linguagem publicada e largamente utilizada pela comunidade de software.
- Incluir especialistas em todo o ciclo de desenvolvimento.
- Evidenciar a complexidade essencial dos problemas a serem resolvidos.
- Estabelecer uma linguagem comum para tratamento de problemas e soluções.
- Buscar mais informações sobre regras e processos que governam a colaboração TODOS OS DIAS!
- Trabalhar em colaboração com o time :D

UTILIZAR UMA LINGUAGEM CONHECIDA

- Estabelecer uma base de conhecimento acessível à todos torna o processo de compartilhamento muito mais simples. A linguagem publicada pelo DDD, seus termos e conceitos, vem sido largamente utilizada pela comunidade de desenvolvimento de software nos últimos 5 anos.
- É possível encontrar referências à “bounded context” e “domains" na documentação de diversas ferramentas, incluindo frameworks como o Symfony e Doctrine.
- Novos membros do projeto serão capazes de consumir material disponível online sem a necessidade de intervenção direta de outros membros.
- Alinhamento com o mercado de desenvolvimento?!

ENGAJAR ESPECIALISTAS

- Especialistas de domínio detêm a razão para existência do software e, portanto, devem ser incluídos ao ciclo de desenvolvimento desde a concepção até a validação e monitoramento das soluções a serem implementadas.

- Entendimento mais profundo sobre cada solução e acesso à conceitos ~~escondidos~~ implícitos em nossa base de código.

- Capacidade de negociar mudanças de planos para atender imprevistos sem comprometer o potencial de satisfação das pessoas interessadas.

EXPRESSAR A COMPLEXIDADE ESSENCIAL

- A complexidade de uma solução é influenciada primariamente pela complexidade essencial do problema a ser resolvido. Está deve ser entendida e considerada a todo o momento, pois é a razão para o software existir.
- A solução técnica deve ser tão complexa quanto for necessário para suprir as necessidades do domínio.

Complexidade do Software = Complexidade Essencial + Complexidade Acidental

- Deixar claro “o que é" o problema que está sendo resolvido é mais importante do que deixar claro “como" o problema está sendo resolvido.

DEFINIR UMA LINGUAGEM COMUM

- Compartilhar um mesmo conjunto de abstrações para descrição dos problemas e soluções evita confusões sobre termos específicos do domínio e nos dá a possibilidade de agir logo quando divergências acontecem.
- Novos membros do projeto serão capazes de se apropriar das histórias contadas pelo código para entender melhor os diferentes domínios modelados pelo software no qual irão trabalhar.
- Discussões sobre o software serão simplificadas ao nível de mapeamento do fluxo dos processos a serem modelados, sem necessidade de traduções e detalhamentos técnicos.

BLOCOS DE CONSTRUÇÃO

Casos de Uso

- Objetos que descrevem as requisições (casos de uso) do sistema, e.g, “Nomear um membro para a posição de supervisor”.
- Agregam a intenção juntamente com as pré-condições e o detalhamento dos passos necessários para a conclusão de cada uma das funcionalidades oferecidas pelo software.
- É uma boa maneira de separar o mundo exterior (infraestrutura) do mundo interior (domínio).

BLOCOS DE CONSTRUÇÃO

Agregados

- Objetos que agregam entidades e “value objects” para realizar uma operação.
- A delimitação dessa agregação é, muitas vezes, definida como sendo uma “barreira transacional”.
- Quando descrevemos as operações dos sistemas, este é, geralmente, o ator principal, responsável por iniciar mudanças de estado.

Blocos de Construção

Entidades

- Elementos do domínio que possuem um ciclo de vida dentro da aplicação, diferenciados através de uma identidade única.
- É importante notar que a identidade está estritamente relacionado ao contexto, podendo não se aplicar em todos os casos onde uma entidade nomeada X aparece.

Blocos de Construção

Value Objects

- Elementos do domínio que não possuem distinção por identidade, porém, representam conceitos inerentes ao negócio. Estes objetos carregam grande parte dos comportamentos (computações) que serão necessárias para atingir diferentes casos de uso.

- Por exemplo, um “Email" poderia ser implementado como um objeto de valor e definir restrições que correspondem ao contexto em que se trabalha — somente emails internos, que apresentam o domínio “cern.ch", são considerados válidos.

Blocos de Construção

Repositórios

- Definem um mecanismo para carregar entidades em memória e persistir as suas mudanças de estado.
- Se comunicam com bases de dados (no nosso caso, Oracle), APIs externas, arquivos em memória, etc.
- Geralmente são definidos apenas para os agregados da aplicação.

BLOCOS DE CONSTRUÇÃO

Módulos

- **A**grupamento de código ao redor da funcionalidade que estes oferecem juntos.
- Pode ser expressado em código através de namespaces e diretórios.
- Encapsulando as associações entre objetos, minimizando a complexidade de entendimento do modelo.

Demo ❤️

https://gitlab.cern.ch/fence/ddd-to-the-moon-and-back

Heurísticas

- Identifique as barreiras entre contextos através da linguagem utilizada para descrever as histórias de cada um deles — quando um mesmo termo é utilizado para descrever dois elementos distintos, existe uma grande possibilidade de se tratarem de contextos diferentes.
- Procure alinhar um grupo de features (bounded context) ao grupo de pessoas que às influenciam (i.e., Authorship Committee teria um contexto dedicado no código que restringiria a implementação de suas policies e requisições à um mesmo conjunto de abstrações)
- Busque modelos que irão nos ajudar a expressar o nosso entendimento do domínio ou nos dar tempo para entendê-lo melhor.
- Faça conceitos implícitos se tornarem explícitos — dê nomes à elementos indefinidos que facilitariam a explicação de conceitos inerentes ao domínio.
- Alinhe a organização do código à das pessoas que tomam decisões sobre suas funcionalidades.
- Procure consenso sobre termos duplicados que se referem ao mesmo elemento dentro de um mesmo contexto.
- Não é necessário se restringir a somente uma base de dados.
- Se dois módulos da sua aplicação sabem muito sobre seus detalhes internos, eles deveriam se se tornar um só (“Marry-me heuristic").
- Isole componentes dos seus sistemas de falhas em outros componentes colaboradores (“Erlang heuristics").

Bibliografia

- Advanced Web Application Architecture - Mattias Noback
- Domain-Driven Design: Tackling Complexity in the Heart of Software - Eric Evans
- Implementing Domain-Driven Design - Vaughn Vernon
- Introducing Event Storming - Alberto Brandolini

Apresentações

- Domain-Driven Design (PHP Experience 2017) - Jefersson Nathan (em português)
- Domain Driven Design do Jeito Certo - Weslley Williand e Elemar Júnior (em português)
- DDD – Introdução a Domain Driven Design - Daniel Cukier (em português)
- Domain-Driven Design: Hidden Lessons from the Big Blue Book - Nick Tune (em inglês)
- Domain-Driven Design: The good parts - Jimmy Bogard (em inglês)
- Domain Driven Design and Onion Architecture in Scala - Wade Waldron (em inglês)

Projetos para se inspirar

- https://github.com/Sylius/Sylius
- https://github.com/matthiasnoback/decoupling-from-infrastructure-workshop
- https://github.com/CodelyTV/php-ddd-example
- https://github.com/matthewrenze/clean-architecture-demo/tree/repo-and-uow
- https://github.com/kgrzybek/modular-monolith-with-ddd