Engenharia de Softwares Escaláveis

Design Patterns e Domain-Driven Design com Java

Agenda

Etapa 3: Introdução ao Domain-Driven Design.

- Domain-Driven Design.
- Projeto Estratégico.

Domain-Driven Design

Atividades de Engenharia de Software

1 Requisitos

2 Análise

3 Arquitetura

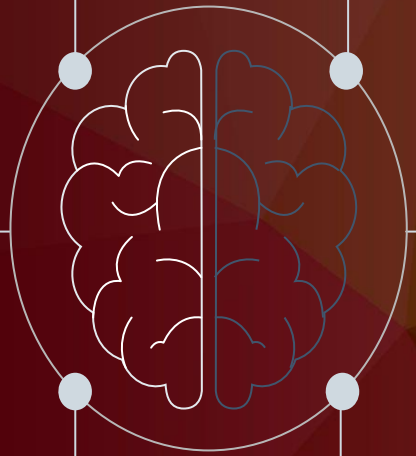
4 Projeto

5 Construção

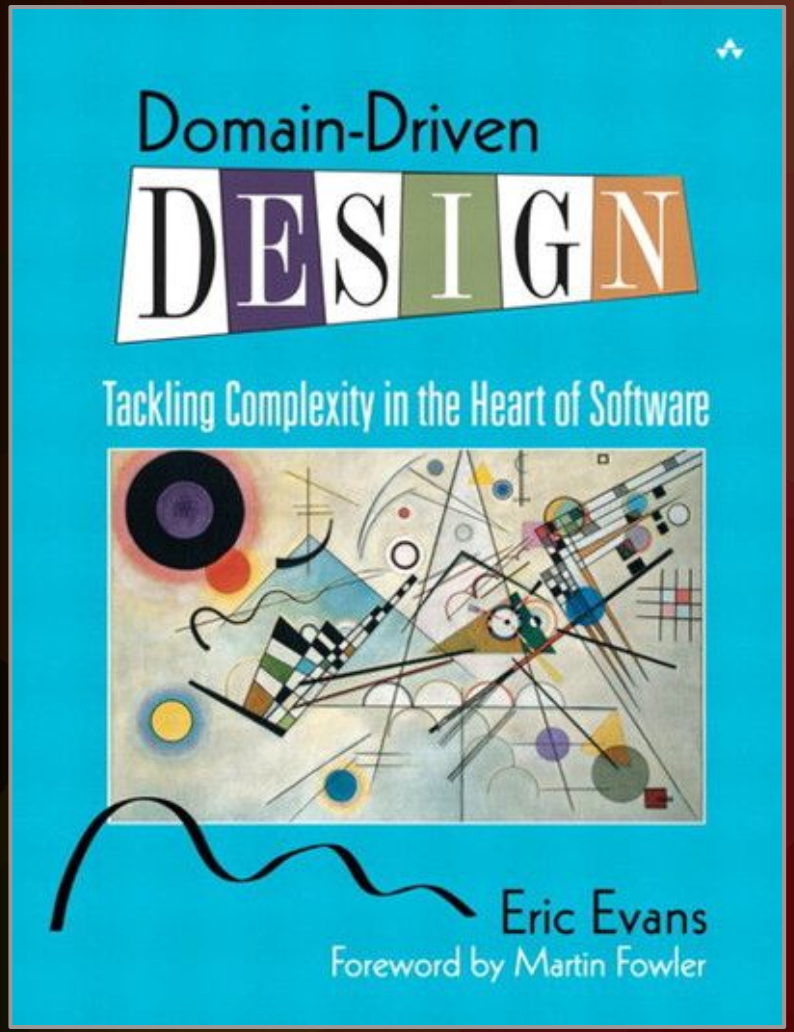
6 Testes

7 Implantação

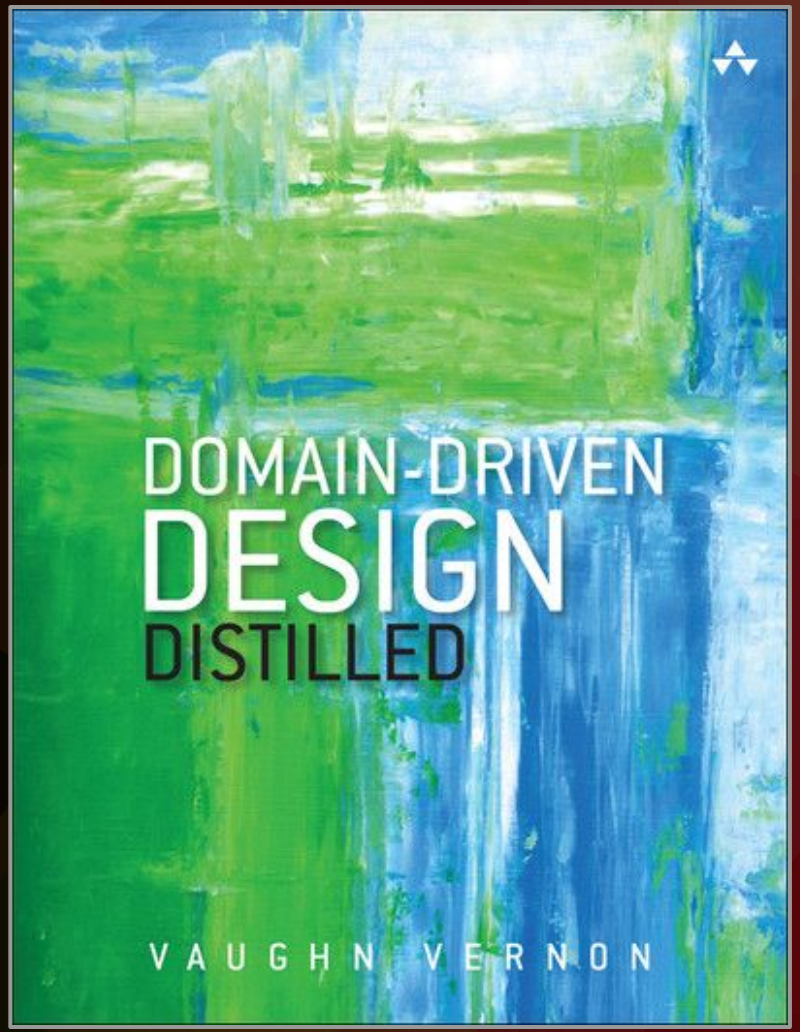
8 Manutenção

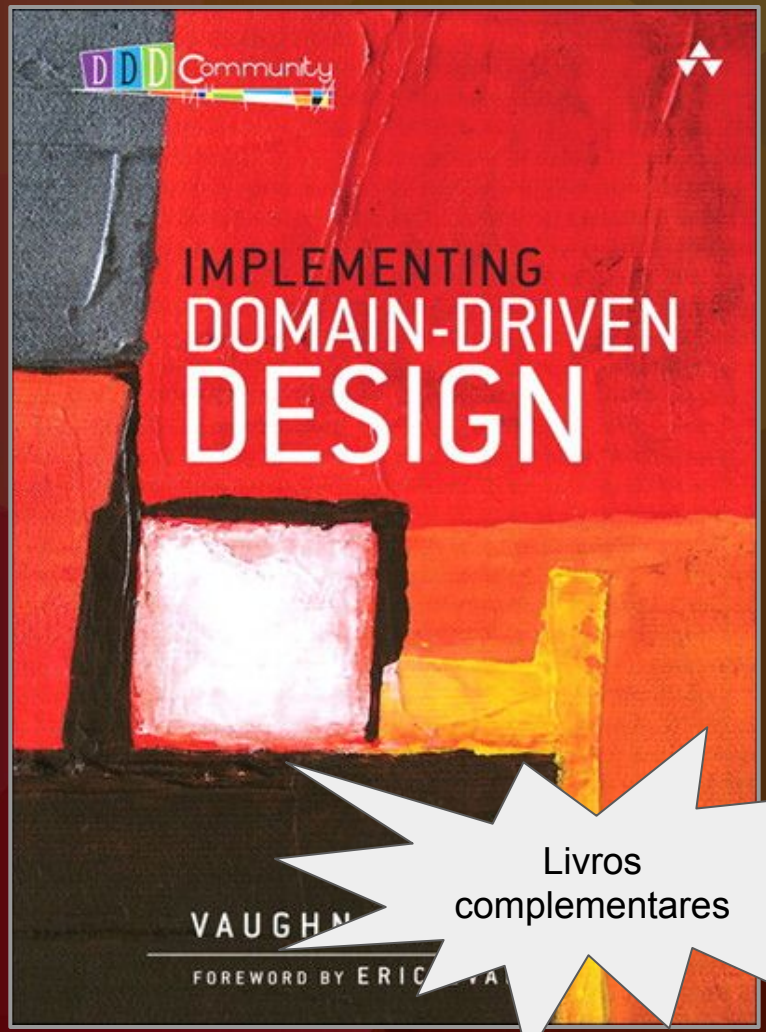


Nosso Foco



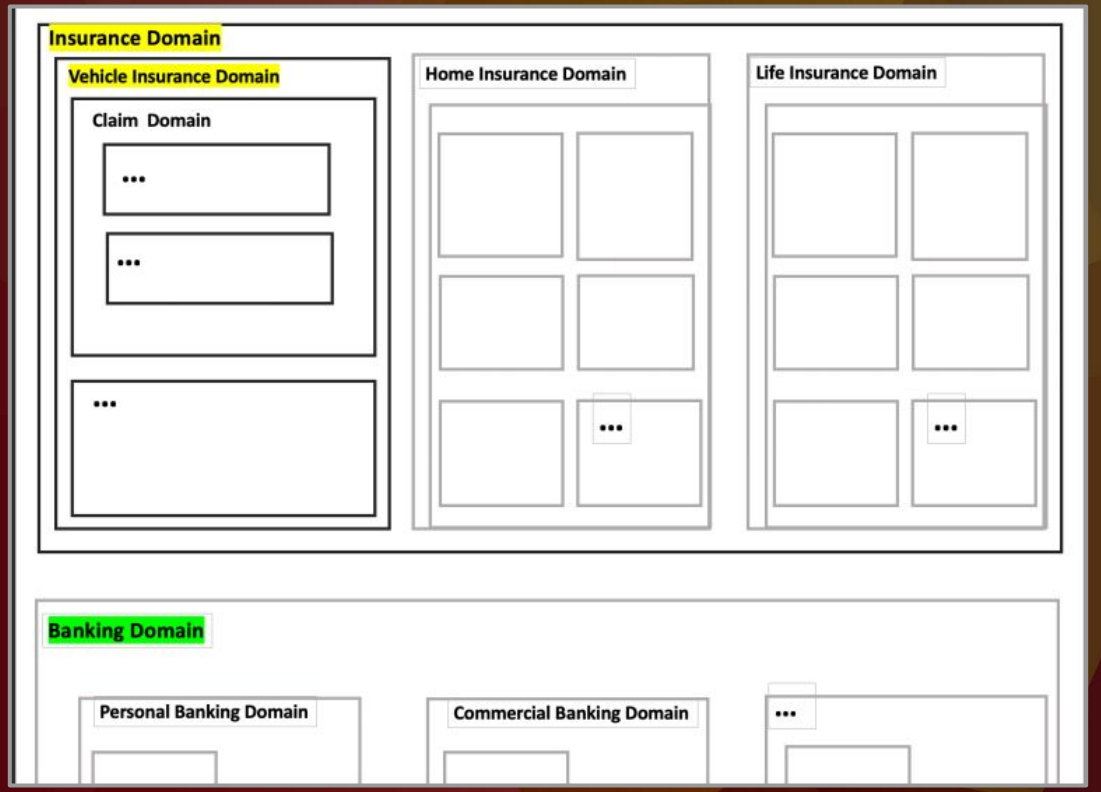
Criado por **Eric Evans** no livro “Design Orientado a Domínio: Enfrentando a Complexidade no Coração do Software”.





Livros complementares

Domain-Driven Design é uma abordagem de design de software (projeto) que se concentra na **modelagem do domínio do problema** como o principal foco do desenvolvimento.



Domínio refere-se à **área de conhecimento ou negócio** ao qual o software se aplica.

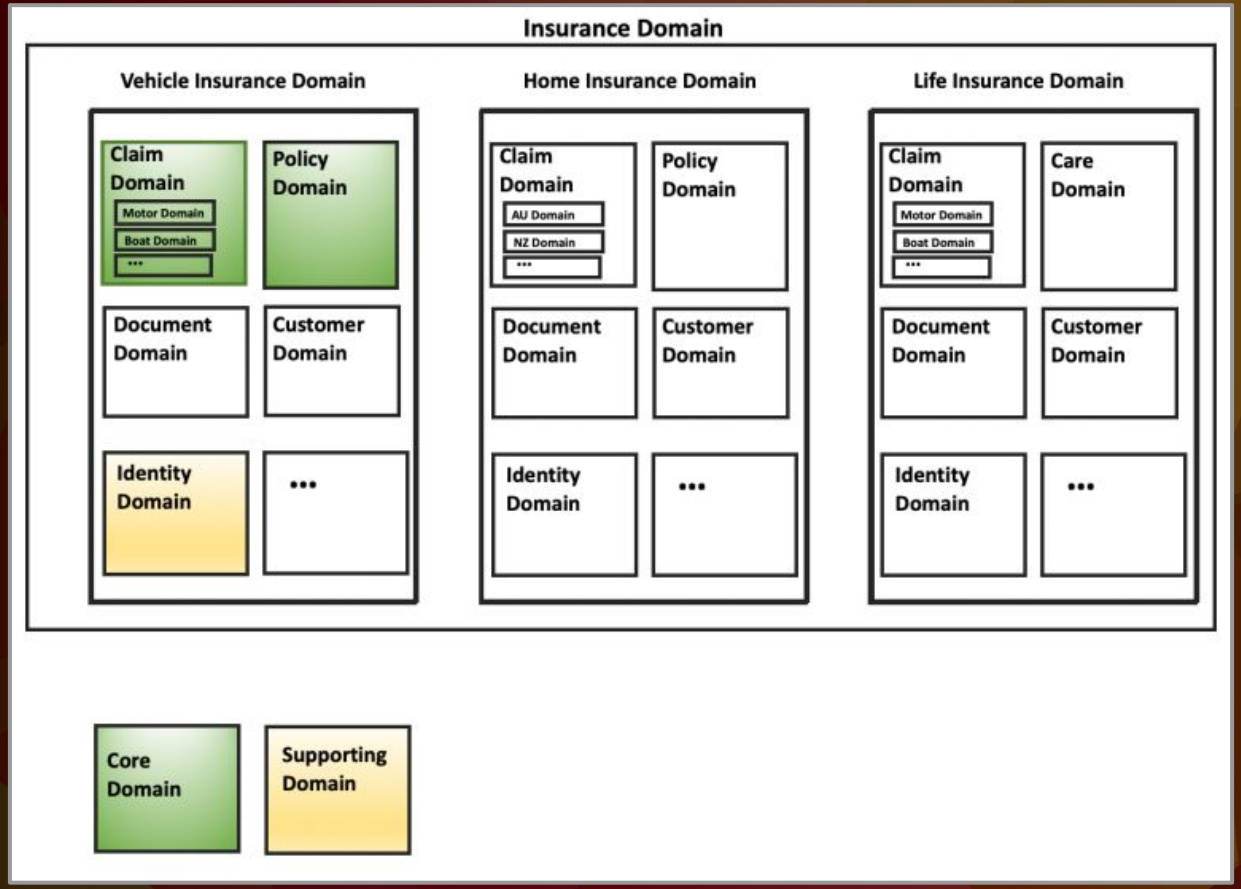
É o conjunto de problemas, regras e processos que o software visa resolver ou suportar.

Por exemplo, em um sistema de e-commerce, o **domínio** inclui todas as atividades relacionadas à compra e venda de produtos online.

Subdomínios são partes menores e mais específicas do domínio geral.

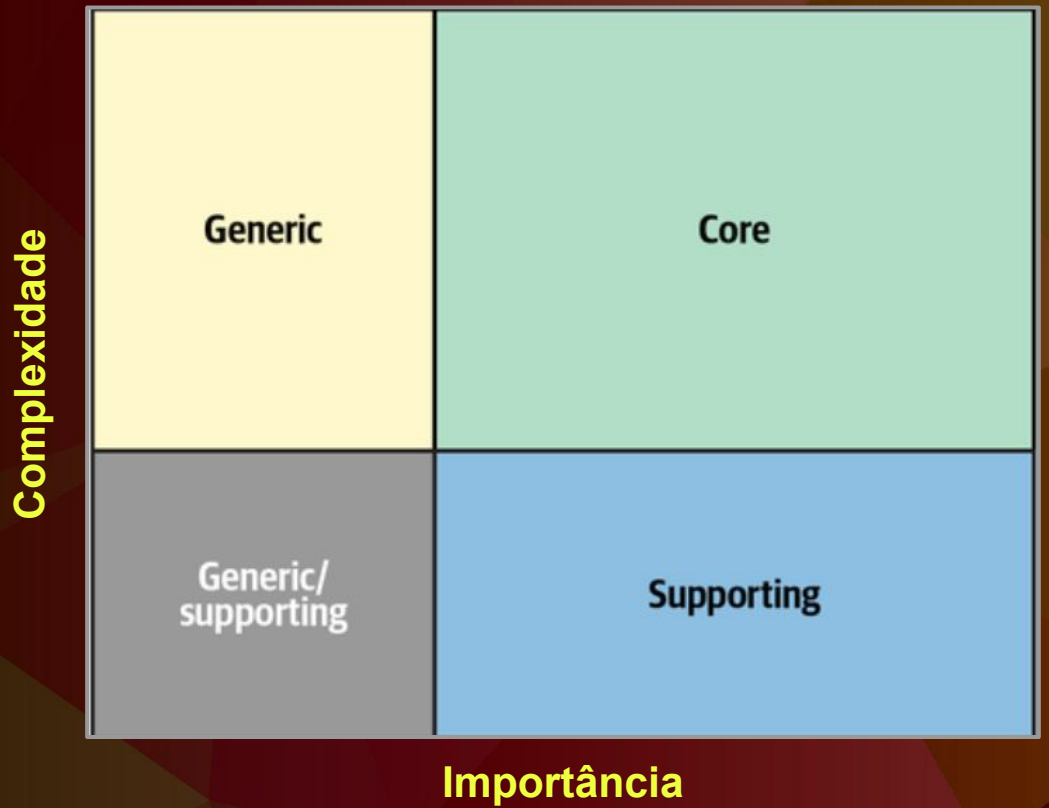
Eles representam **áreas de conhecimento ou funcionalidades** dentro do domínio maior.

Por exemplo, no domínio de e-commerce, os **subdomínios** podem incluir gestão de inventário, processamento de pedidos, gestão de clientes, pagamentos online, entre outros.



O negócio de **Seguros** está dividido em vários **domínios** que contam com **subdomínios** colaborativos.

Domínios e **Subdomínios** podem ser classificados, considerando a sua importância e complexidade.



Domínio Principal é o que diferencia o negócio no mercado e representa sua principal fonte de vantagem competitiva.

É onde o maior esforço de design e desenvolvimento deve ser concentrado.

Gestão de Pedidos (balcão, drive-thru, app).

Preparação de Alimentos (Cozinha).

Subdomínio de Suporte dá suporte ao **Core Domain**, mas não é por si só o diferencial competitivo.

Sistemas Internos de TI (POS, apps internos).

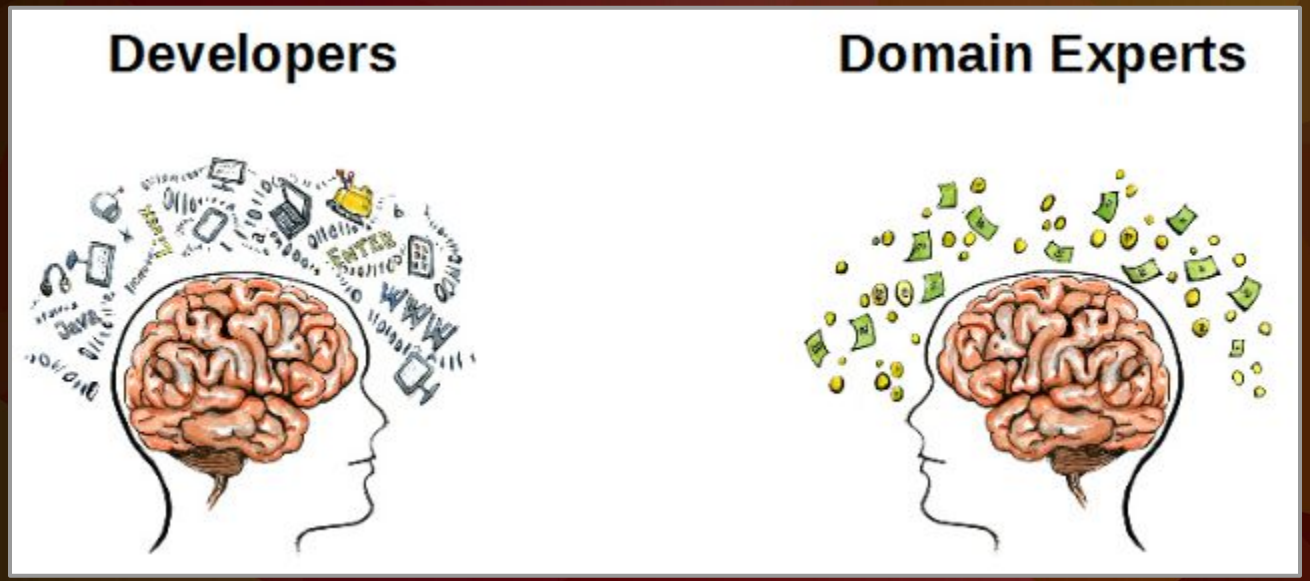
Recrutamento, Treinamento e Gestão de Equipes.

Subdomínio Genérico são funcionalidades comuns a muitas empresas, que podem ser compradas prontas ou terceirizadas.

Folha de Pagamento e Benefícios.

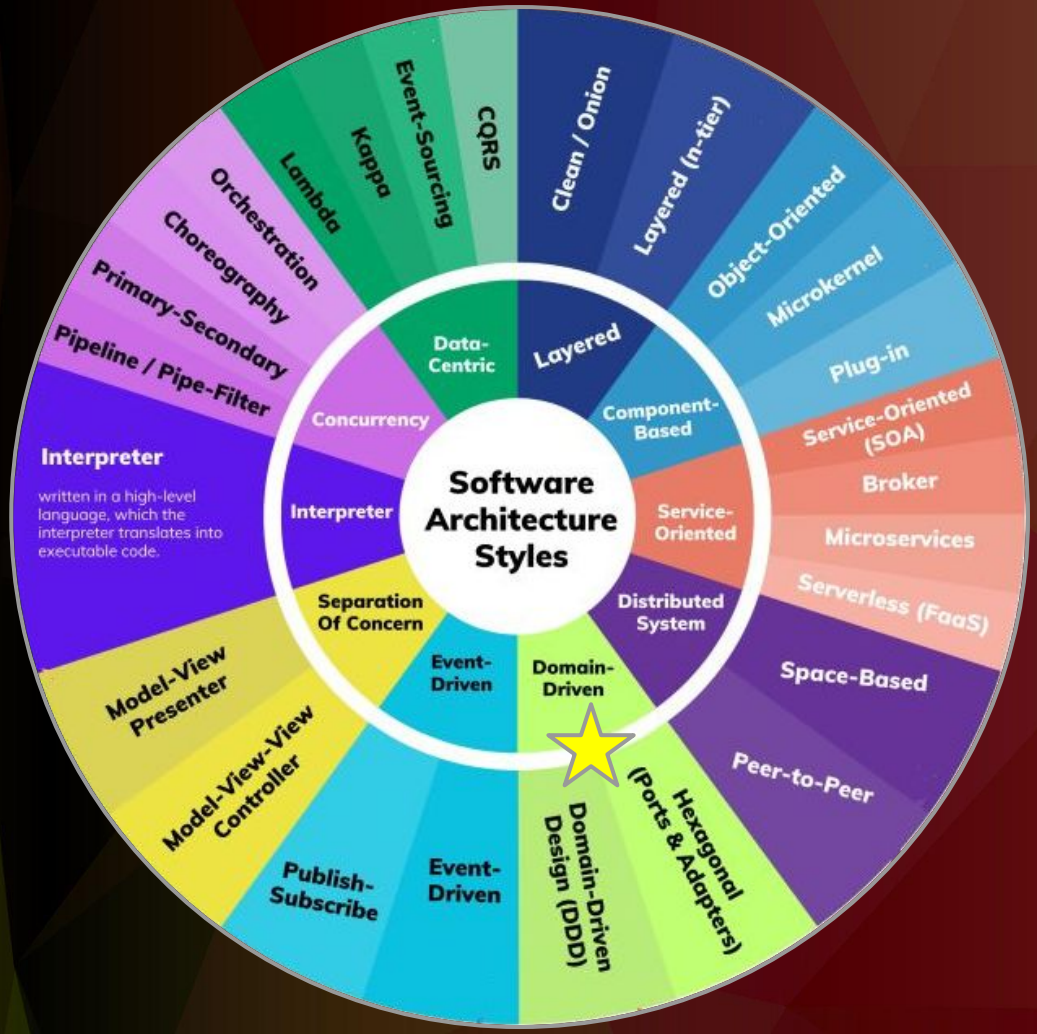
Contabilidade e Finanças.

Linguagem Ubíqua é uma **linguagem comum** entre desenvolvedores e especialistas do **domínio**, que descreve **conceitos e processos de negócio** e garante uma comunicação clara e consistente.



Separação de Responsabilidades identifica e define os **contextos delimitados**, separando as diferentes partes do sistema em componentes coesos e independentes, o que facilita a manutenção, a evolução e a escalabilidade.

Iteração e Refinamento Contínuos é a **abordagem iterativa e incremental** para o desenvolvimento de software, permitindo que o modelo de domínio evolua e seja refinado ao longo do tempo à medida que novos *insights* são adquiridos e os requisitos do negócio mudam.



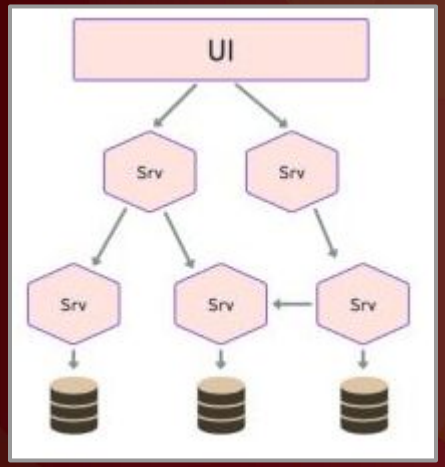
Este é um resumo do blog **Bytebytego** que relaciona **padrões de arquitetura de software**.

Deve-se dar um destaque especial para os seguintes itens:

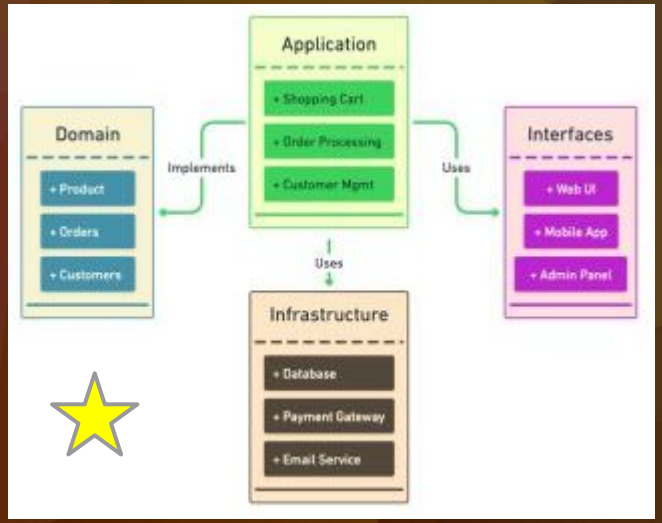
- Layered.
- Service-Oriented.
- **Domain-Driven**.
- Event-Driven.
- Separation of Concern.
- Data-Centric.

Layered

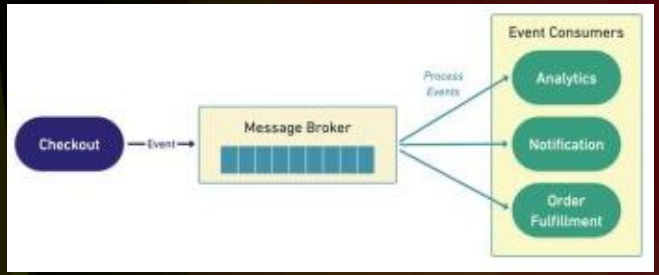
Microsserviços



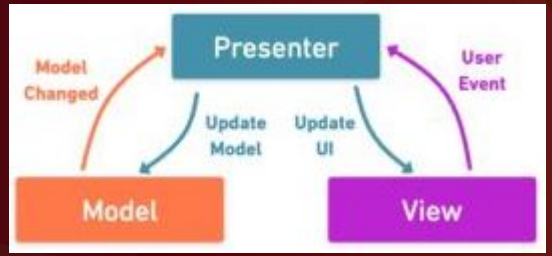
Domain-Driven Design



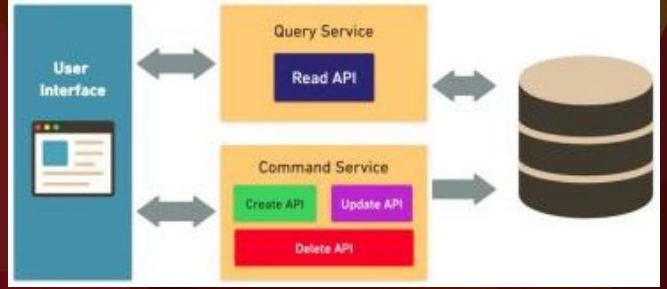
Event-Driven Architecture



Model View Presenter



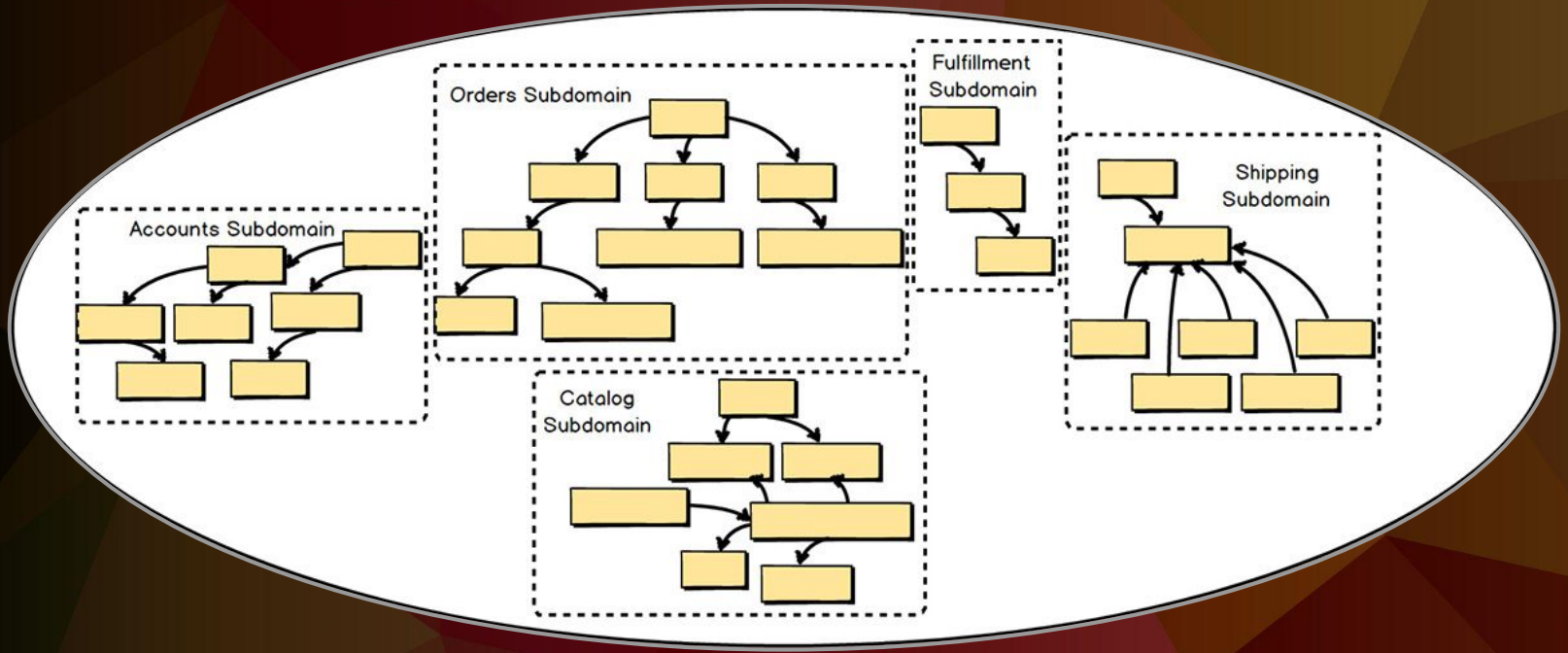
CQRS Architecture



Projeto Estratégico

Visão Holística: em vez de focar apenas em partes isoladas, a abordagem estratégica no **DDD** envolve uma compreensão profunda e holística do domínio do problema como um todo.

Isso inclui identificar os diferentes subdomínios, seus relacionamentos e contextos delimitados.

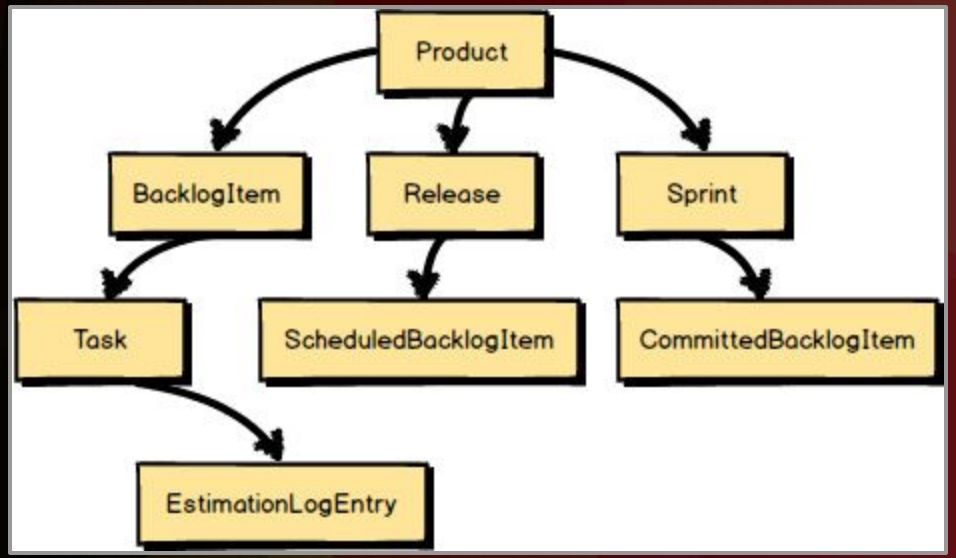


Bounded Contexts são limites explícitos dentro dos quais um modelo específico é definido e válido.

Projetar software de forma estratégica no **DDD** envolve a identificação e definição adequada desses **Bounded Contexts**, garantindo que cada parte do sistema tenha uma **linguagem ubíqua** e um **modelo de domínio** claro.

Context Mapping é uma técnica do **DDD** que ajuda a visualizar e gerenciar os relacionamentos entre os diferentes contextos de um sistema.

Isso é crucial para projetar a arquitetura de sistemas complexos, garantindo uma integração adequada entre os contextos e a comunicação clara entre as partes interessadas.



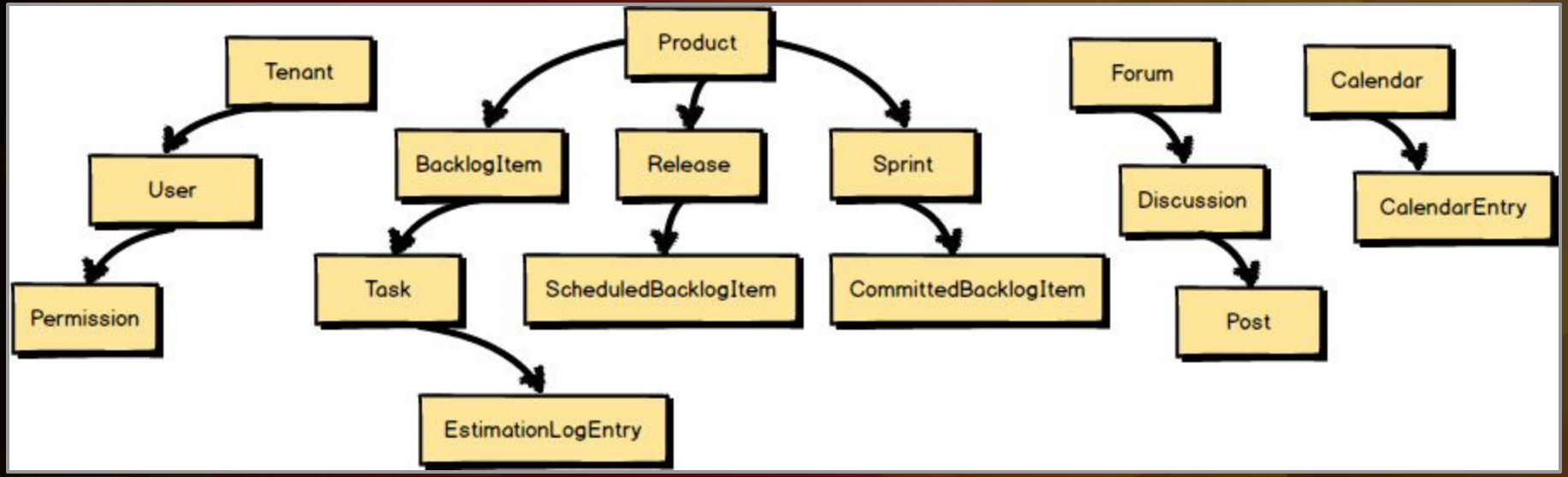
O exemplo é de uma **aplicação ágil de gerenciamento de projetos** baseada em Scrum.

Um conceito central é **Produto**, que representa o software que será construído e que será refinado ao longo de talvez anos de desenvolvimento.

O produto possui **Itens de Backlog**, **Lançamentos** e **Sprints**.

Cada **Item do Backlog** possui diversas **tarefas** e cada **tarefa** pode ter uma coleção de **entradas de registro de estimativa**.

Lançamentos têm **itens de backlog agendados** e **Sprints** têm **itens de backlog comprometidos**.

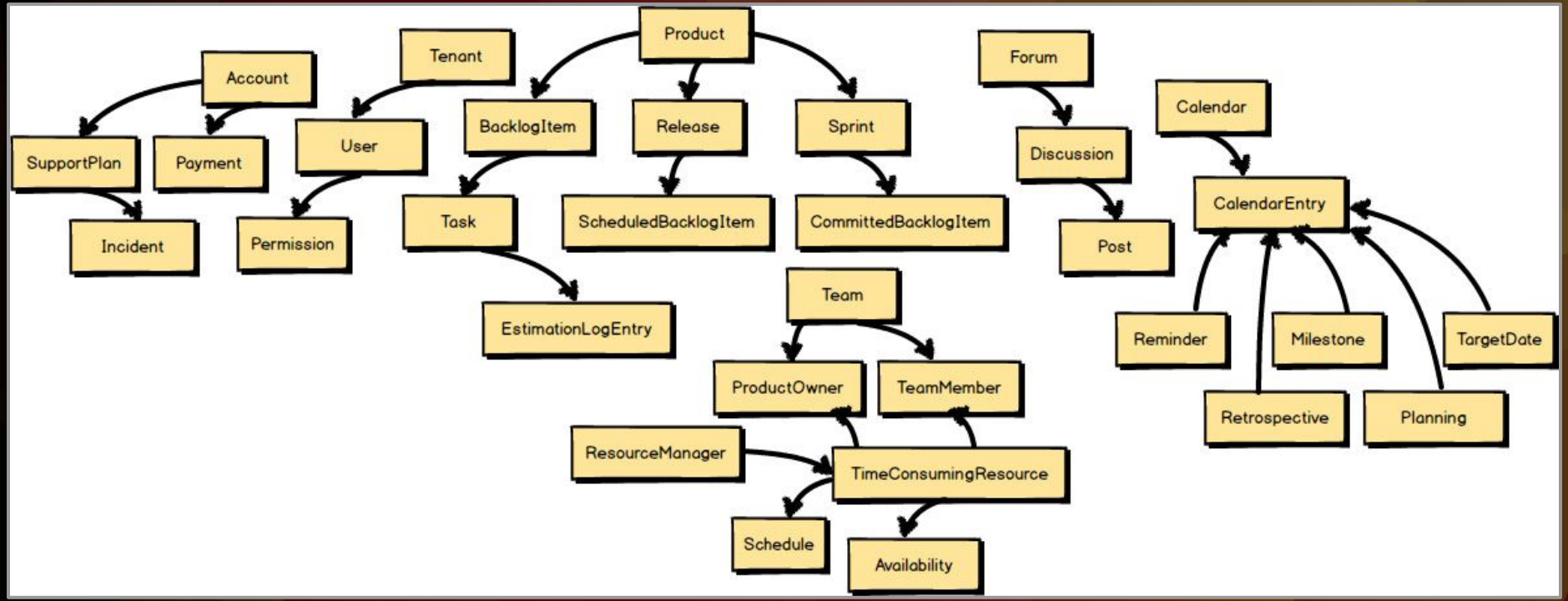


Vamos representar cada organização cliente como um **Inquilino**.

Aos **Inquilinos** permitiremos o registro de qualquer número de **usuários**, e os **usuários** terão **permissões**.

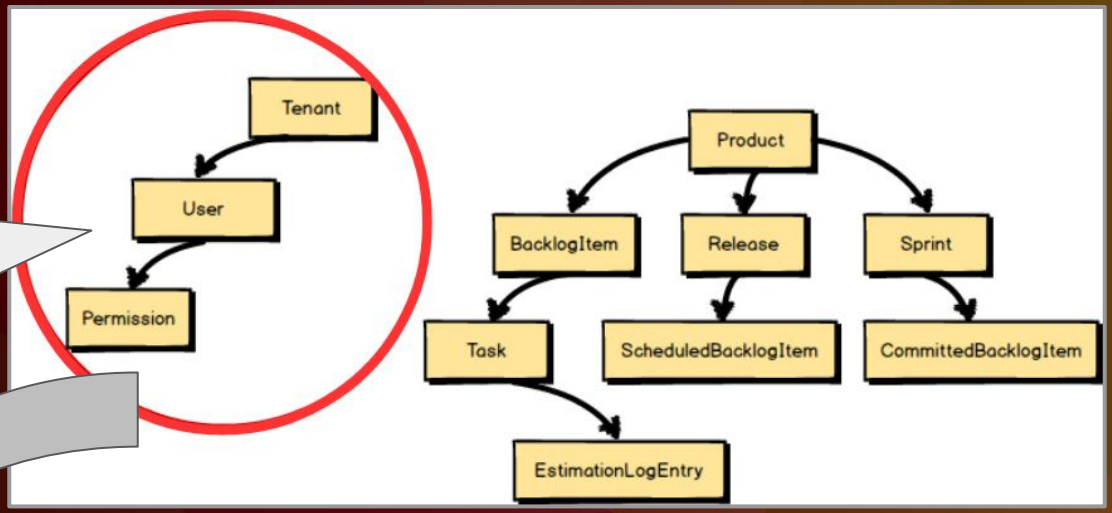
E vamos adicionar um conceito chamado **Fórum** para representar uma das ferramentas colaborativas que iremos apoiar.

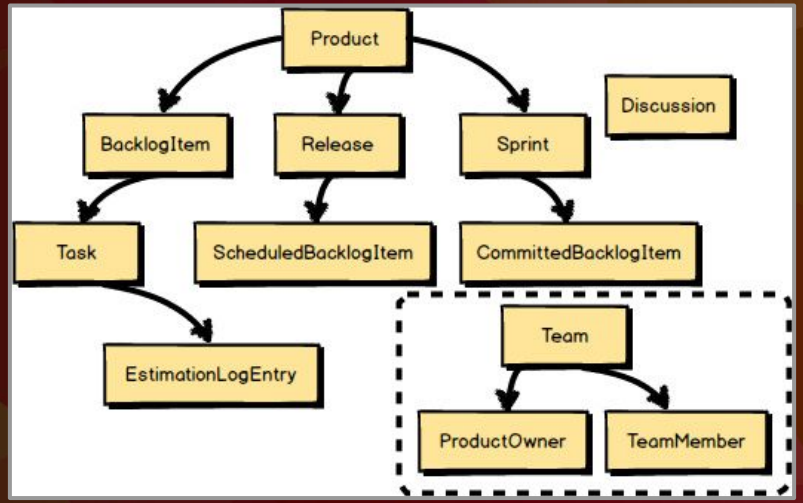
Também queremos oferecer suporte a **Calendários** compartilhados.



E assim, vamos agregando novas entidades ao sistema, conforme os requisitos funcionais definidos

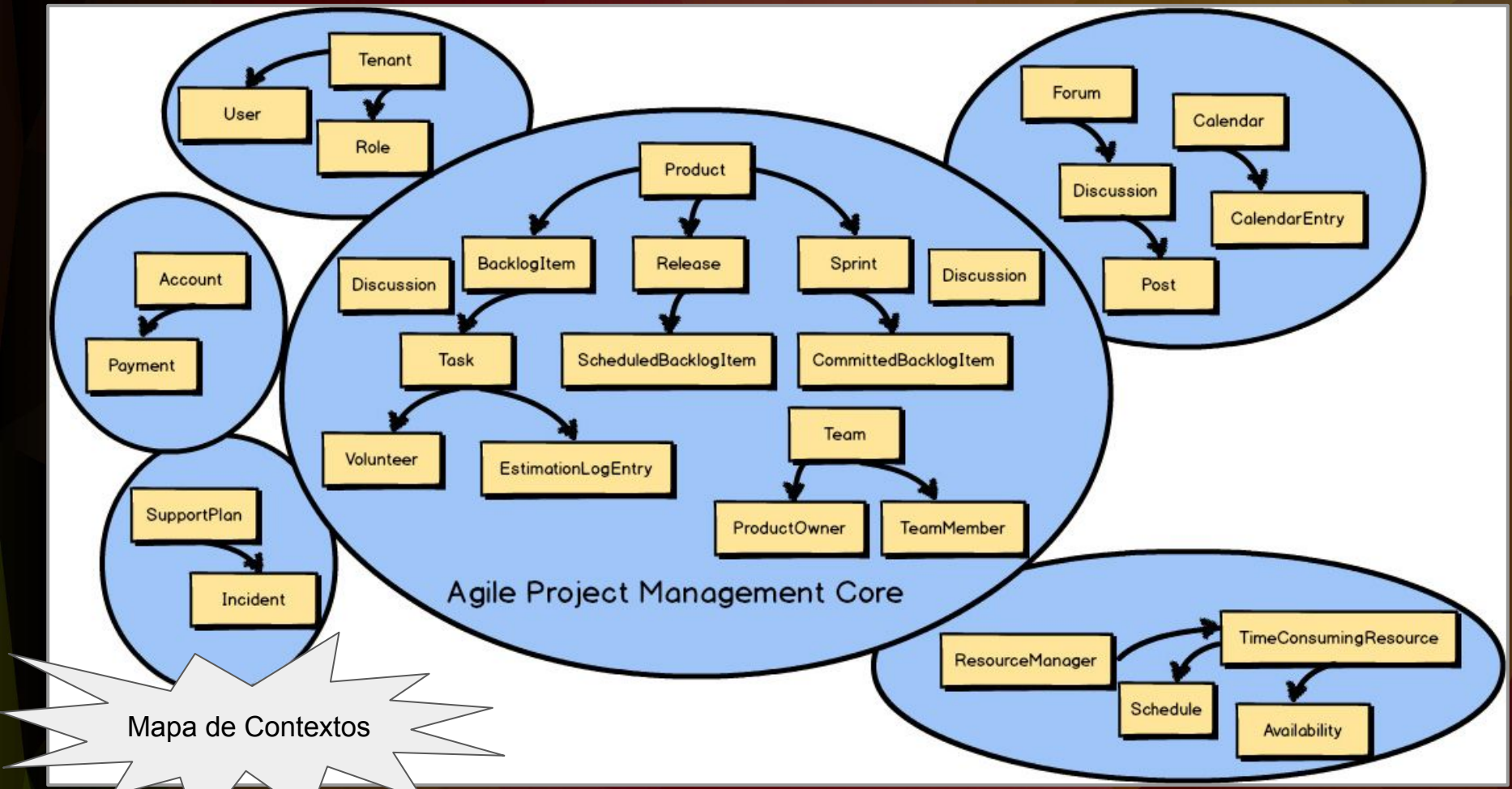
Esses termos não são conhecidos no contexto do Scrum





Linguagem Ubíqua

Procuramos substituir termos que não são adequados ao contexto da aplicação.



Mapa de Contextos