2021

Domain-Driven Design

Uma Introdução

Johnathan Fercher da Rosa
Renan Da Silva Passos Lopes

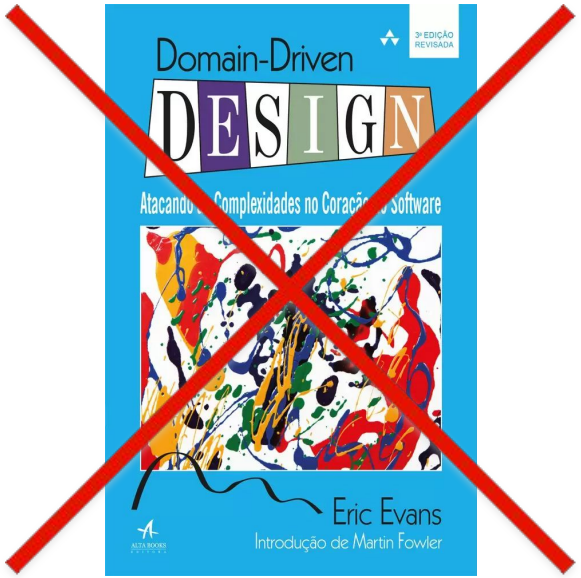
2021

- Introdução
- Design Estratégico
- Model-Driven Design
- Recapitulando
- Referências

Introdução

O que NÃO é DDD?

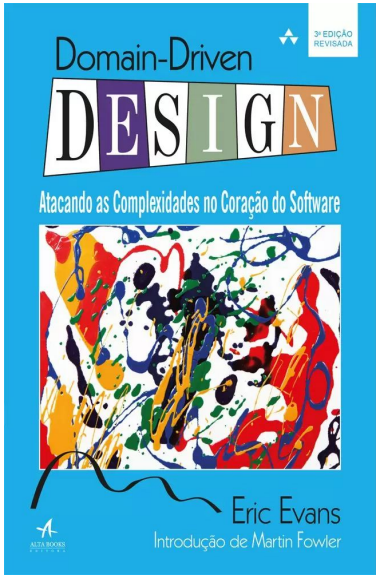
- Não é uma arquitetura
- Não é um framework
- Não é uma tecnologia
- Não está limitado a uma linguagem de programação
- Não está limitado ao paradigma orientado a objetos



O que é DDD?

É uma abordagem de desenvolvimento de software focada em resolver problemas complexos.

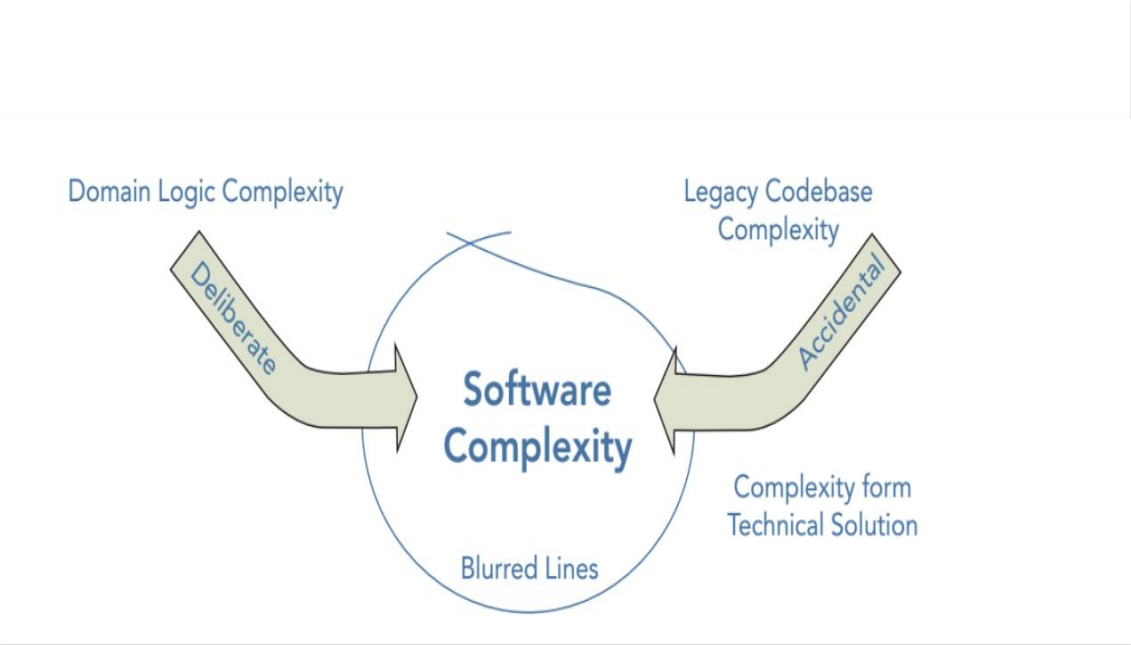
Proposto pela primeira vez por Eric Evans.



O que torna um software complexo?

Mistura da complexidade do domínio com a complexidade tecnológica

Complexidade do Domínio x Complexidade Acidental

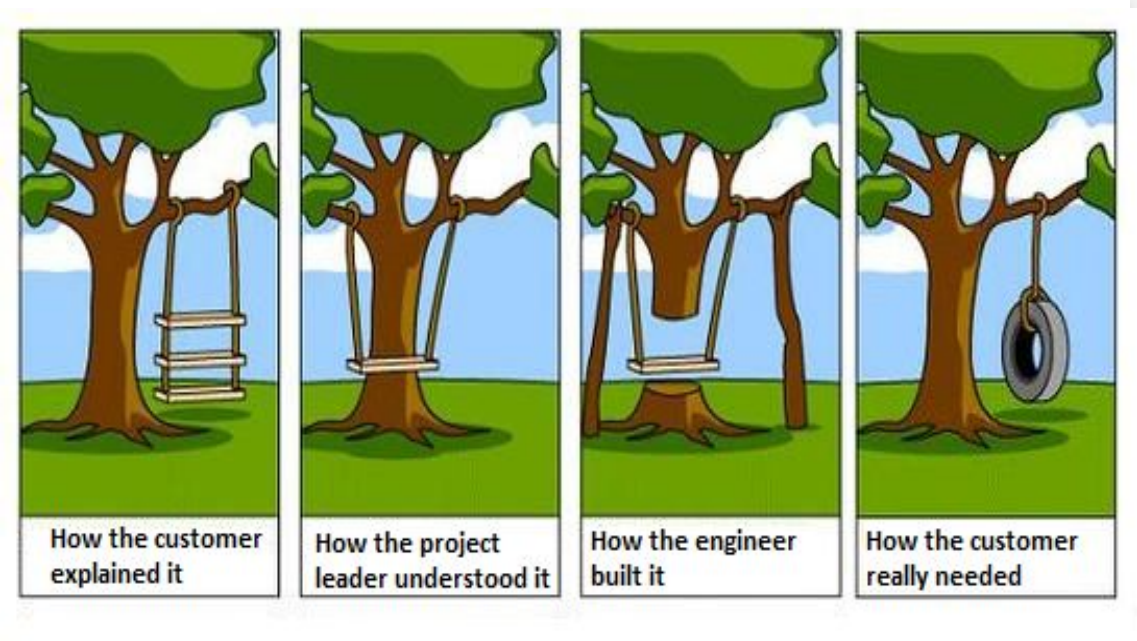


O que torna um software complexo?

Código que funciona mas não revela a intenção de negócio.

Código difícil de ler e manter porque as traduções entre o modelo do negócio e o código são custosas e suscetíveis a erro

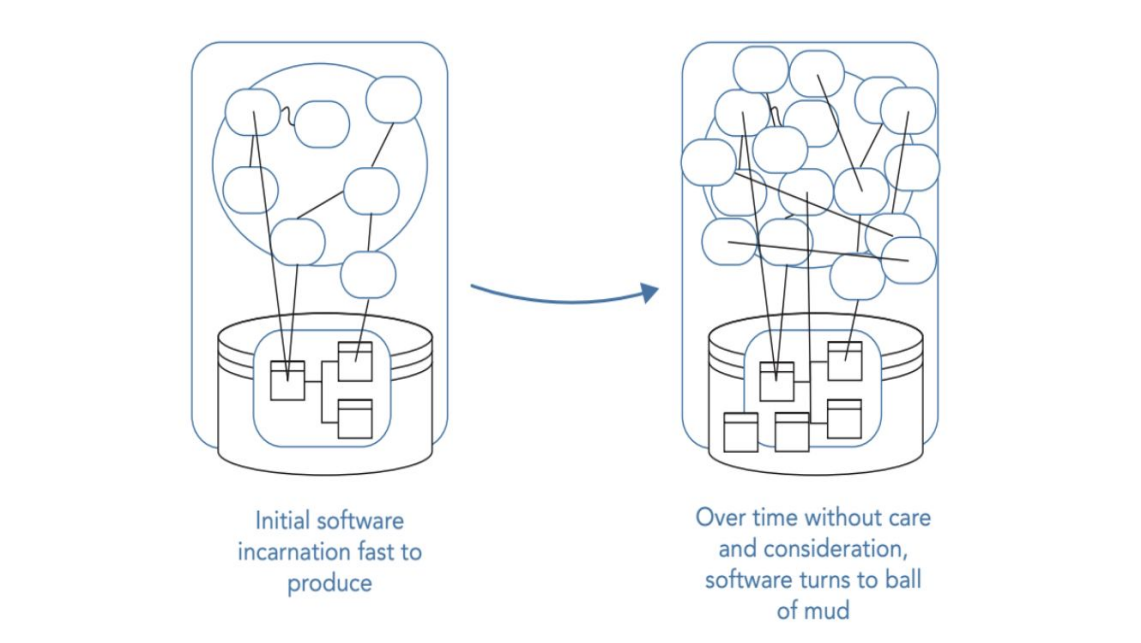
Ausência de uma linguagem comum



O que torna um software complexo?

Pouco foco no design em torno do domínio do problema, melhorias subsequentes são difíceis e problemáticas

Falta de Organização



Problemas

- Desenvolvedores com dificuldade de trabalhar no código
- Desenvolvedores infelizes por trabalhar numa base de código confusa e propensa a erros
- Velocidade de desenvolvimento não atende o negócio, mesmo adicionando mais pessoas no time
- Lentidão em agregar valor ao negócio
- No final, o pedido para reescrever a aplicação é garantido

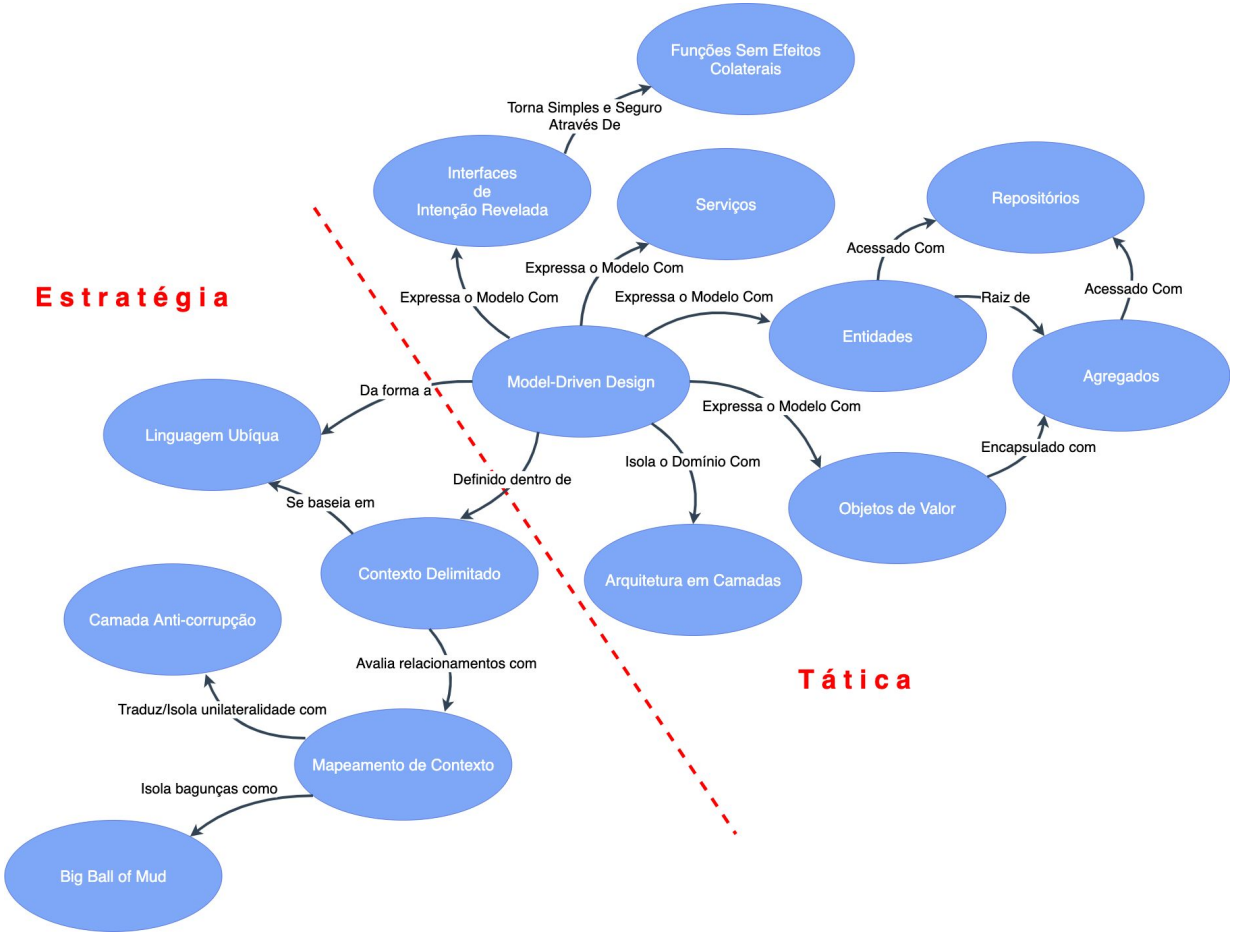
Domain-Driven Design

- Separar domínio e tecnologia
- Habilitar colaboração entre desenvolvedores e analistas de negócio através de uma linguagem compartilhada
- Revelar conceitos importantes do negócio
- Criar modelo para resolver problema do domínio
- Isolar modelo da ambiguidade e corrupção
- Entender os relacionamentos entre contextos

Padrões Táticos e Estratégicos

Padrões Estratégicos refinam o problema do domínio e moldam a arquitetura da aplicação.

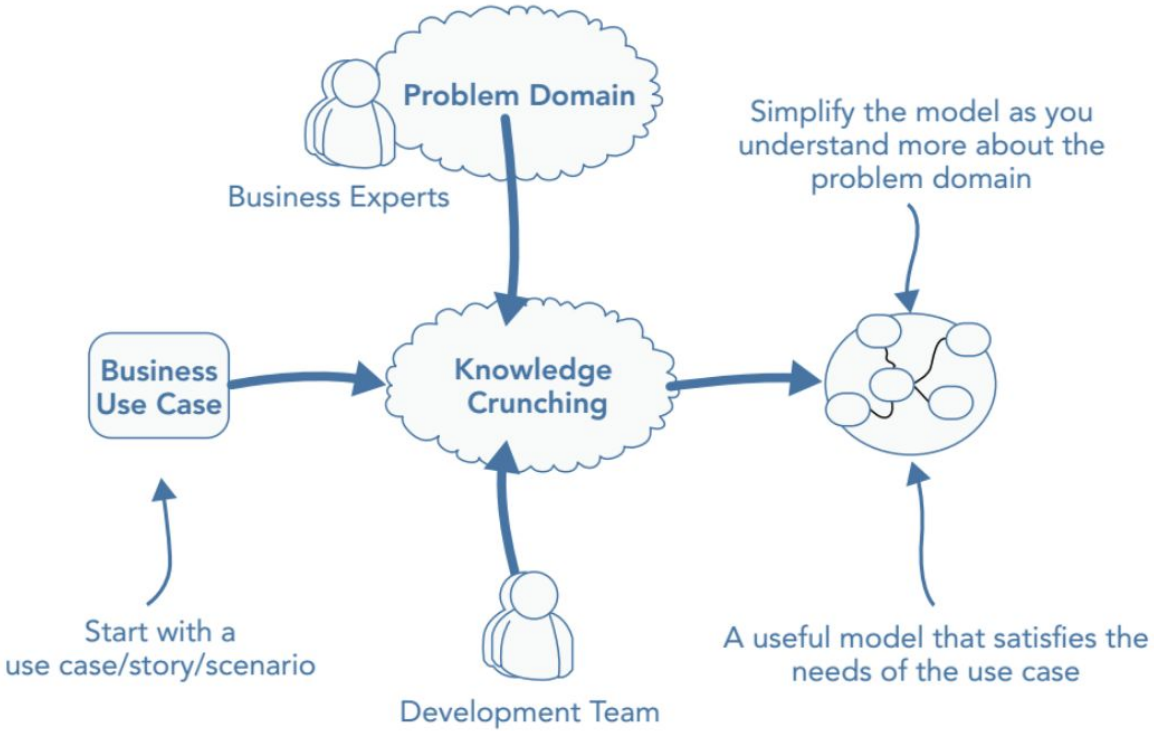
Padrões Táticos ajudam a criar modelos efetivos para contextos delimitados complexos.



2021

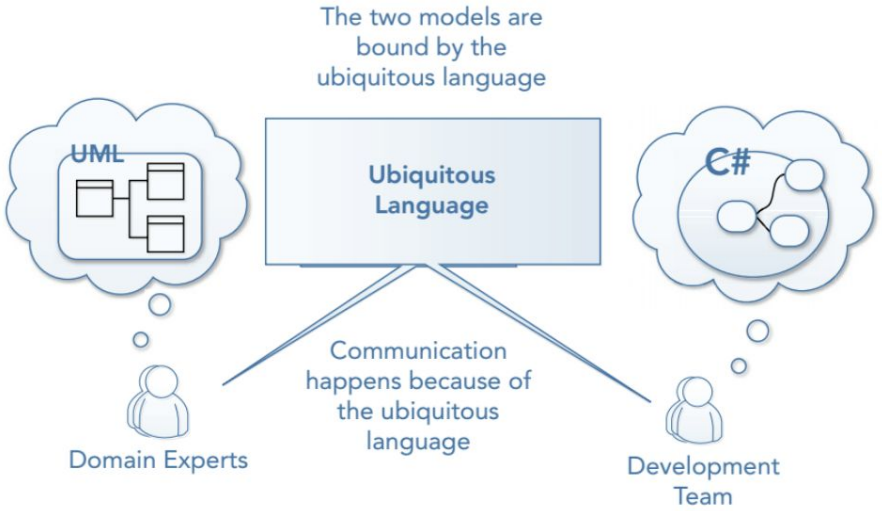
Design Estratégico

Destilando o problema do domínio



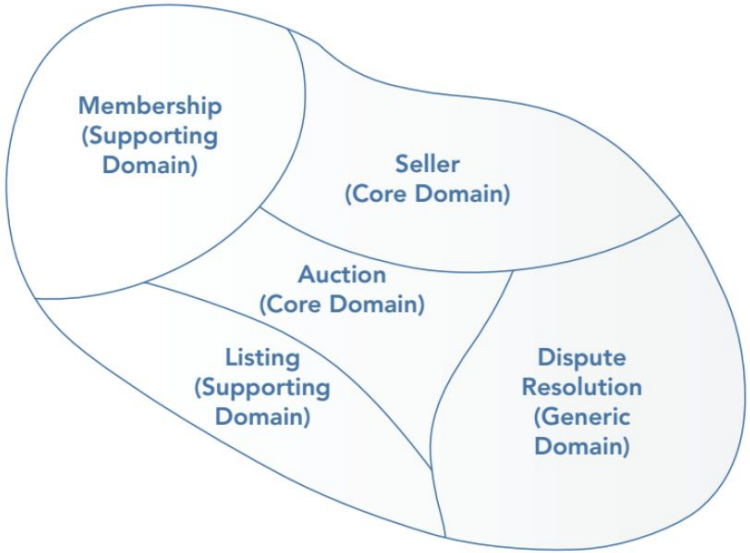
Linguagem Ubíqua

- Permite a comunicação efetiva entre os experts do domínio e desenvolvedores
- Rica em terminologia específica do domínio.
- Permite os times organizar o modelo mental e de código com facilidade
- Sobrevive ao Software
- Constante construção



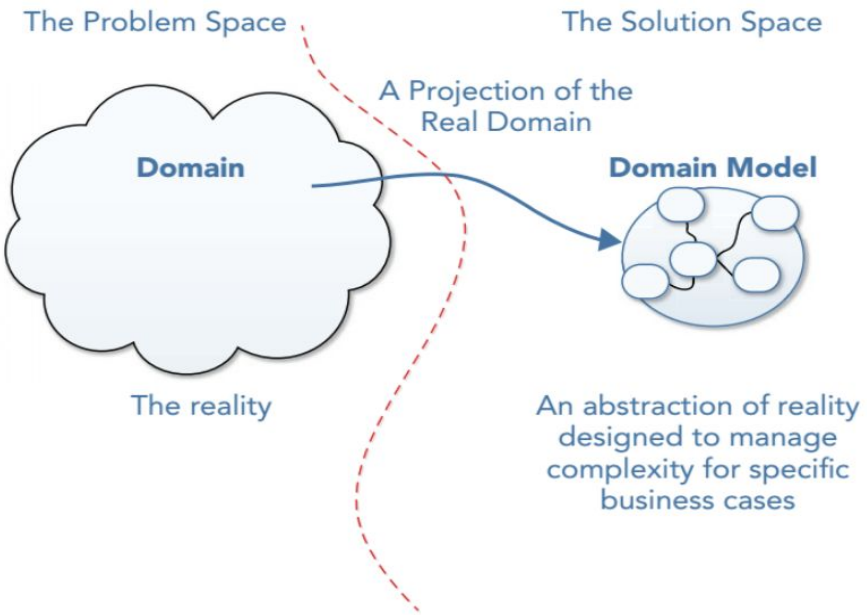
Core Domain - Por que decompor o domínio?

- Algumas partes do sistema são mais importantes do que outras
- Nem todas as partes do sistema necessitam do mesmo grau de qualidade
- Limites explícitos entre diferentes contextos



Modelo de Domínio

- Abstração do domínio real do problema
- Não é a realidade
- Claro e livre de complexidades tecnológicas



Modelo de Domínio

Conceito

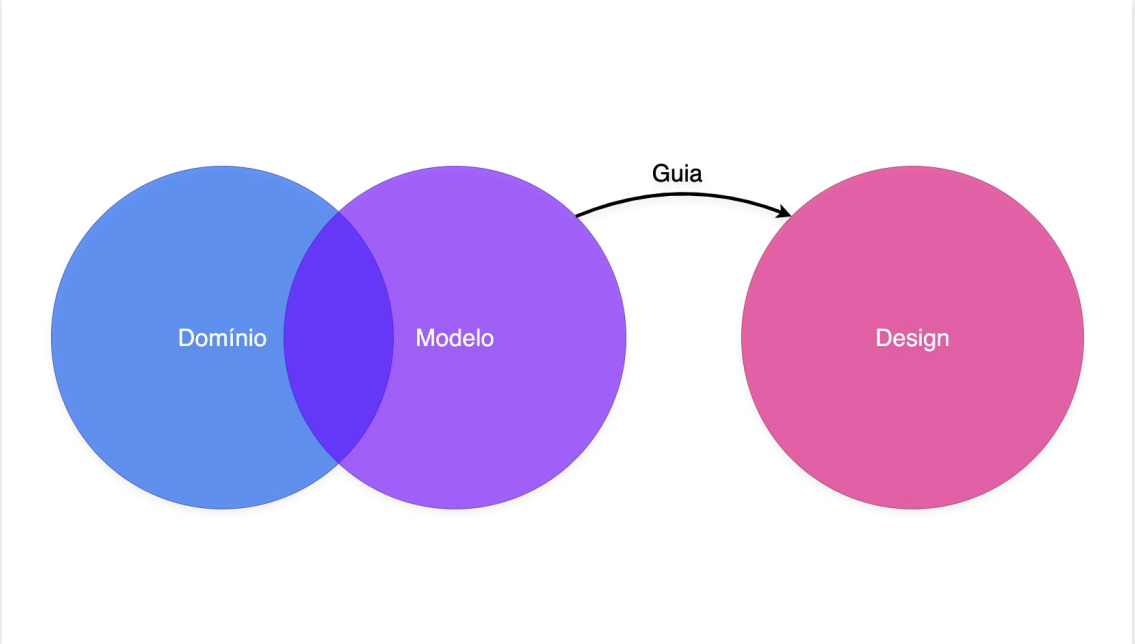
Design

Definição

Uma implementação do modelo levando em consideração os requisitos técnicos.

"Um domínio pode ser expressado por diferentes modelos, e um modelo pode ser implementado de diferentes formas."

- Domain-Driven Design: Quickly



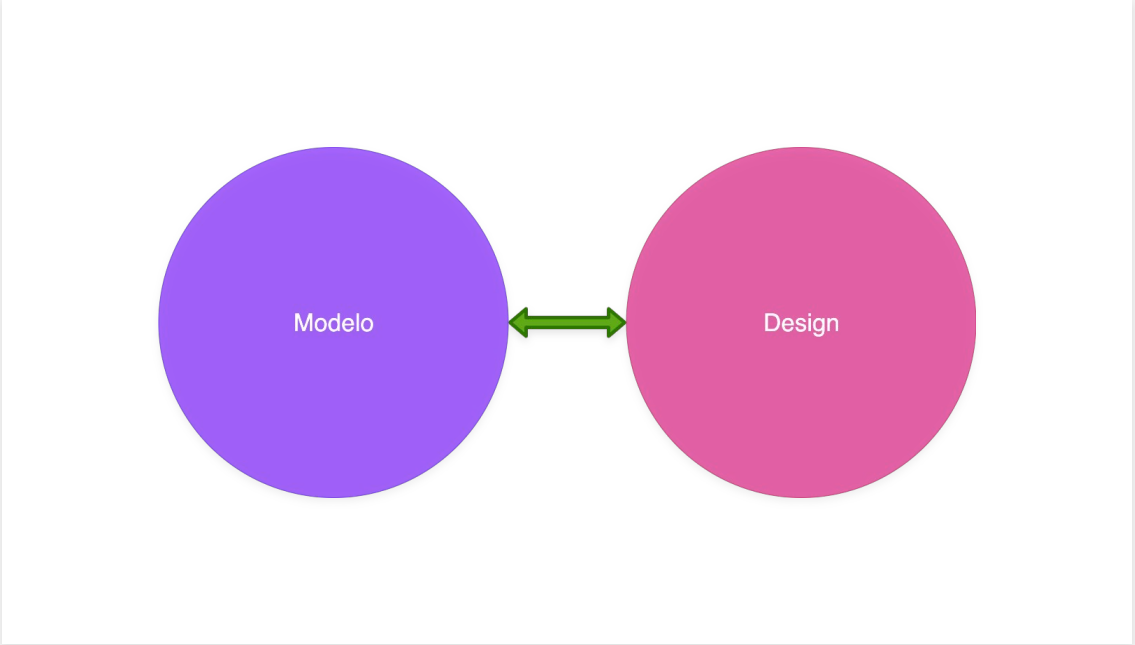
Conceito

Conexão Modelo e Design

"Um modelo que não leva em consideração o design pode ser péssimo de ser implementado e mantido;

Uma implementação que não leva em consideração o modelo pode não resolver o problema;"

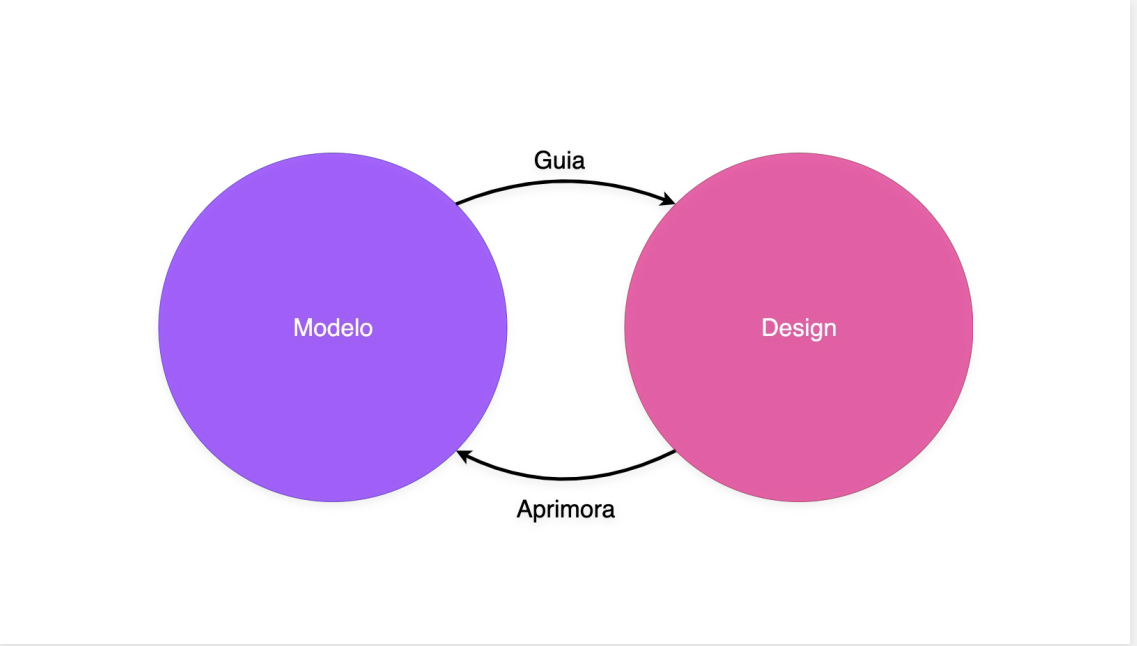
- Domain Driven Design (DDD) - Daniel Cukier



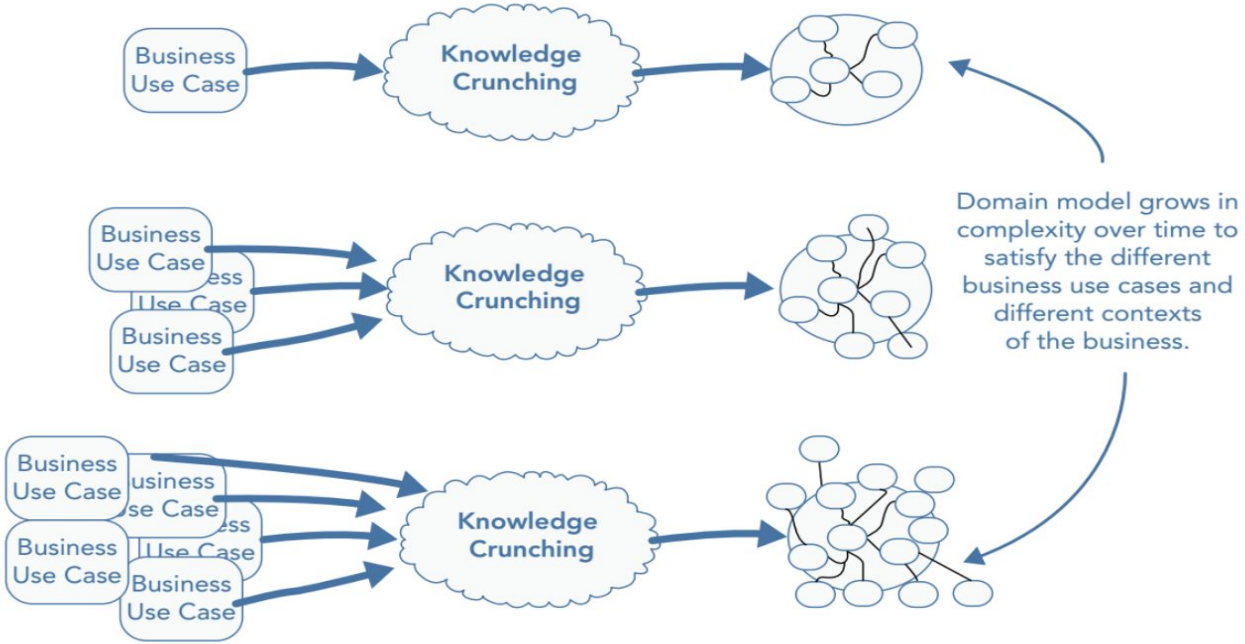
Conceito

Conexão Modelo e Design

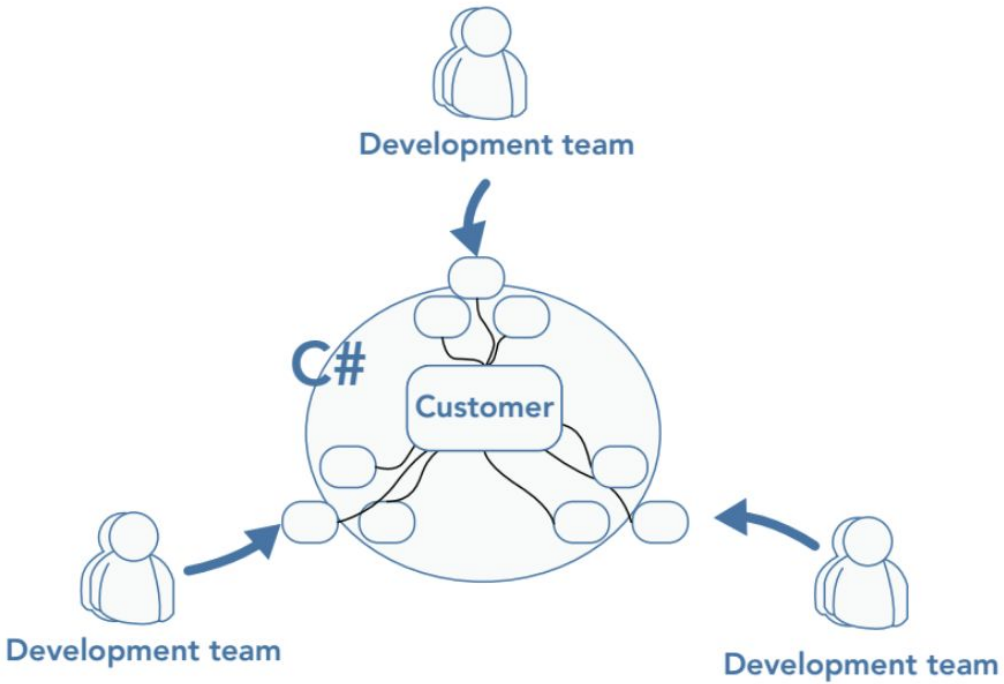
O modelo guia a implementação para resolver um problema do domínio, e a implementação fornece ideias para aprimorar o modelo.



Contextos Delimitados - Problemas com modelo único

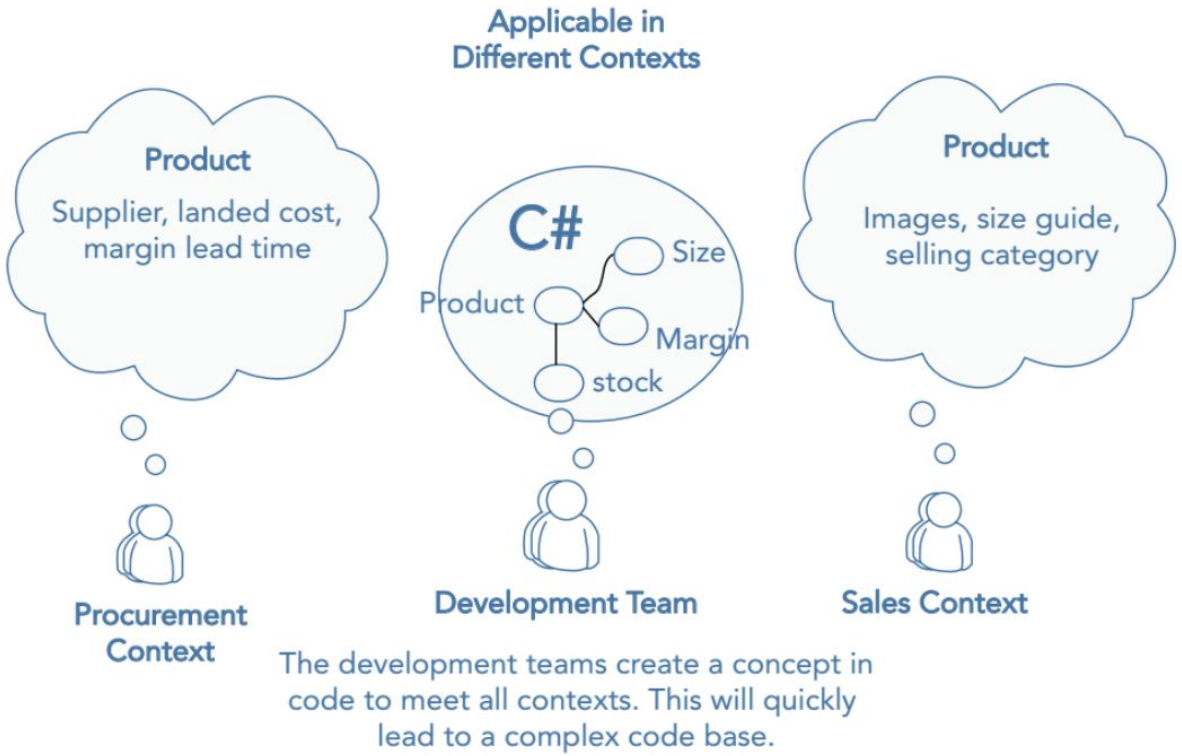


Contextos Delimitados - Problemas com modelo único

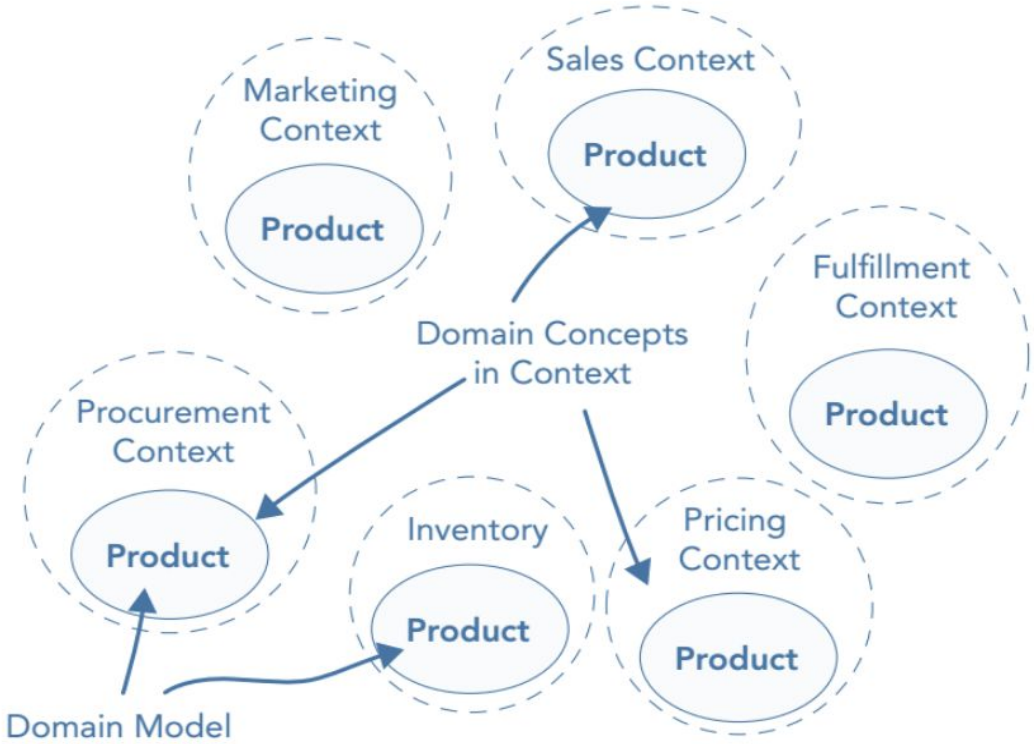


In a single model, multiple teams will dilute the explicitness.

Contextos Delimitados - Problemas com modelo único



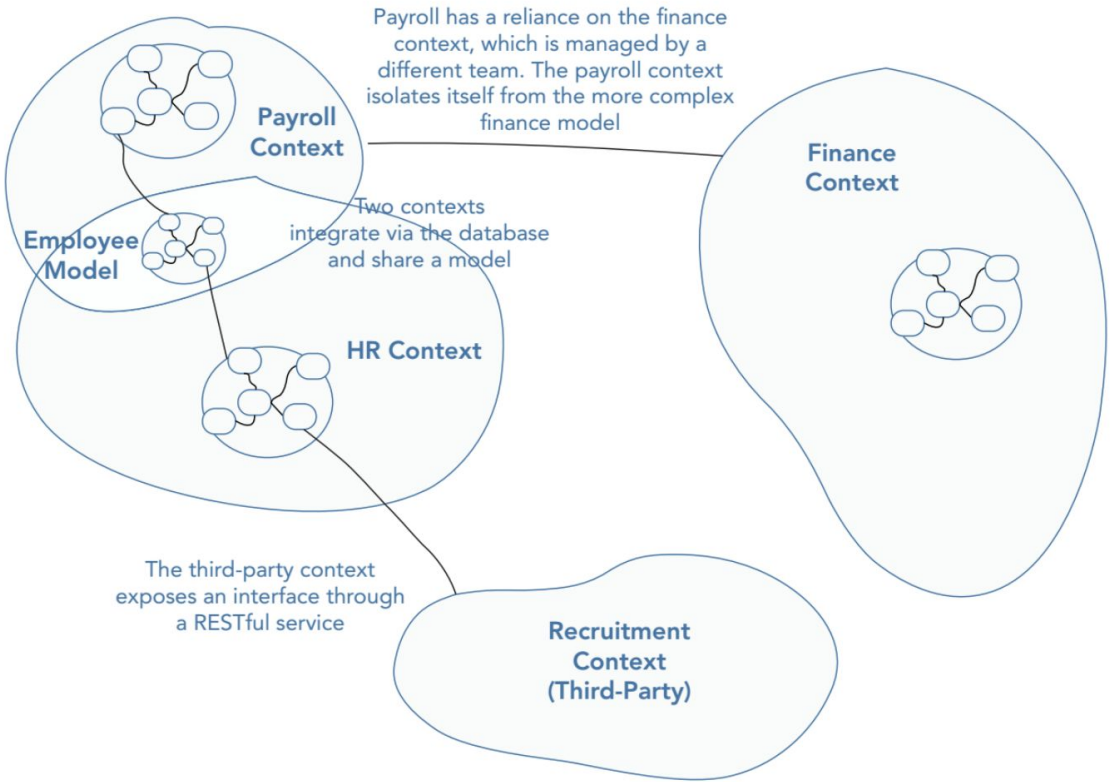
Contextos Delimitados



Definição de Contextos Delimitados

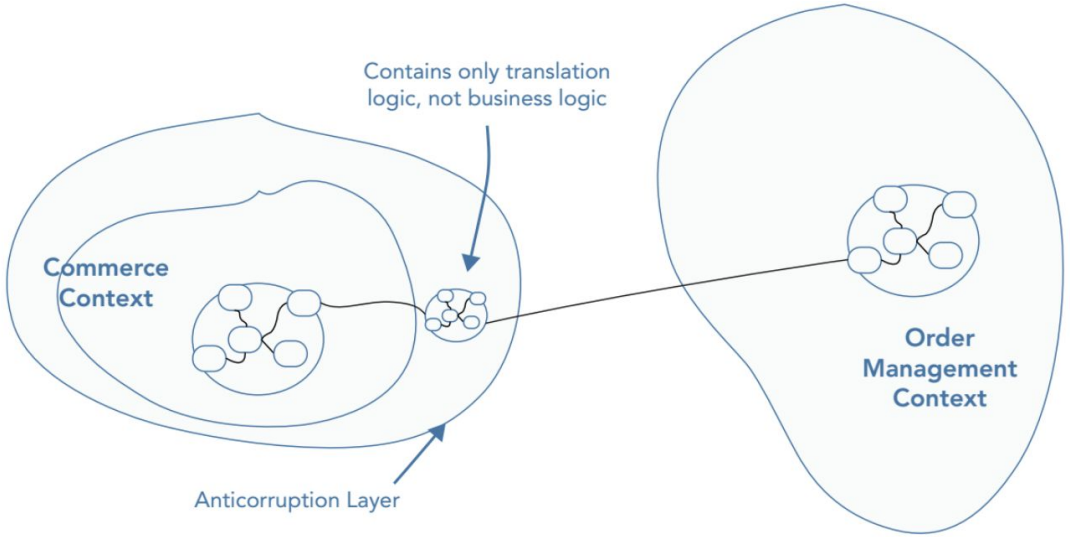
- ~~Tamanho do módulo~~
- Ambiguidade na terminologia e conceitos do negócio
- Capacidades de negócio
- Localização física
- Código Legado
- Integração com códigos terceiros

Mapeamento de Contextos



Camada de Anti-Corrupção

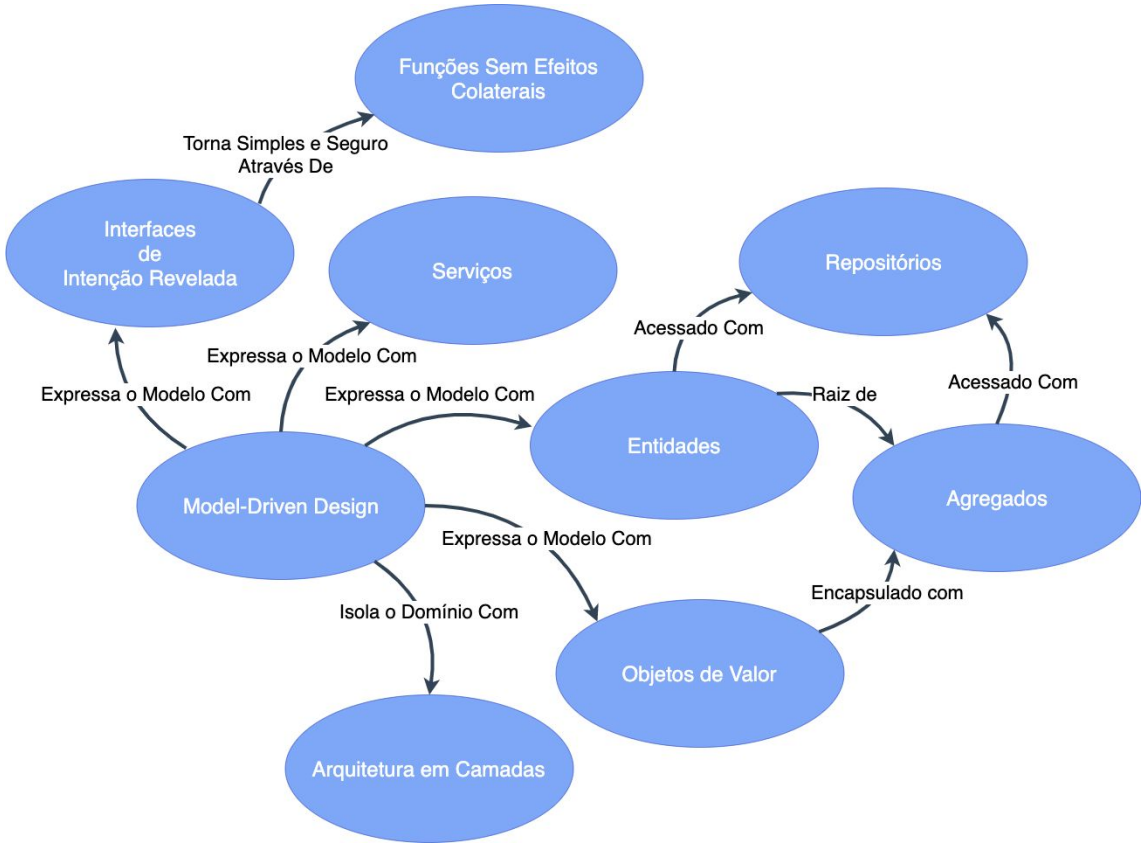
Usada para integrar com códigos que não nos pertencem ou não podemos mudar



2021

Model-Driven Design

Model-Driven Design



Entidades

- Parte dos nomes da linguagem ubíqua;
- Objetos que possuem identidade;
- Ciclo de vida longo, mutáveis e possuem estado;
- **Não é a entidade do banco;**


```
{
   "container_id": "56d27415-7887-48ed-9e45-c9728701bbbb",
   "logistic_center_id": "BRSP01",
   "site_id": "MLB",
   "created_date": "2021-10-13T16:34:45Z",
   "status": {
      "value": "pending"
   }
}
```

Objetos de Valor

- Parte dos nomes da linguagem ubíqua;
- Objetos que não possuem identidade;
- Fornecem informações adicionais sobre algo;
- Ciclo de vida curto, imutáveis e possuem estado;

```
{
   "route_regeneration": {
      "date": "2021-10-13T16:34:45Z",
      "type": "wrong_logistic_center",
      "last_mile_service_id": 21556
   }
}
```

Agregados

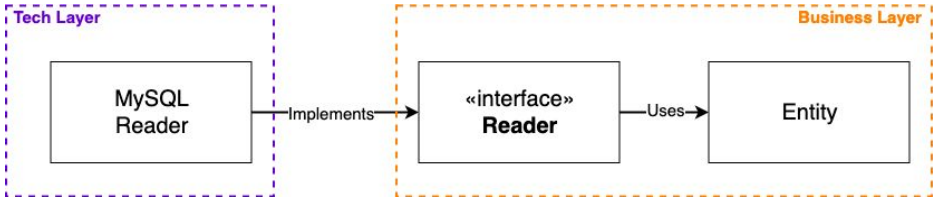
- Um agregado de entidades e/ou objetos de valor;
- Uma espécie de "view";
- Partes do agregado não podem ser manipuladas fora do agregado;


```
{
   "container_id": "56d27415-7887-48ed-9e45-c9728701bbbb",
   "logistic_center_id": "BRSP01",
   "site_id": "MLB",
   "created_date": "2021-10-13T16:34:45Z",
   "finished_date": "2021-10-13T16:34:45Z",
   "route_regeneration": {
      "date": "2021-10-13T16:34:45Z",
      "type": "wrong_logistic_center",
      "last_mile_service_id": 21556
   },
   "authorization": {
      "date": "2021-10-13T16:34:45Z"
   },
   "status": {
      "value": "finished",
      "success": true
   }
}
```

Repositórios

- Encapsula lógica de leitura/escrita de dados, desacoplando detalhes de tecnologia;
- Pode servir para realizar operações de escrita/leitura de entidades, objetos de valor e agregados inteiros;

```
type ShipmentReader interface {
   GetByID(ctx context.Context, id string) (*entities.Shipment, error)
}
```



Serviços

- Parte dos verbos da linguagem ubíqua;
- Não possuem estado;
- Objetos que carregam comportamento;
- **Nem todo serviço é de domínio;**

```
type ShipmentRouteRegenerator interface {
    Regenerate(ctx context.Context, id string) error
}
```

Interfaces De Intenção Revelada

```
type ShipmentService interface {
    Regenerate(ctx context.Context, id string) error
    GetUnfinisheds(ctx context.Context, id string) (*entities.Shipment, error)
    Add(ctx context.Context, positionID string) error
}
```

```
type ShipmentRouteRegenerator interface {
    Regenerate(ctx context.Context, id string) error
}
```

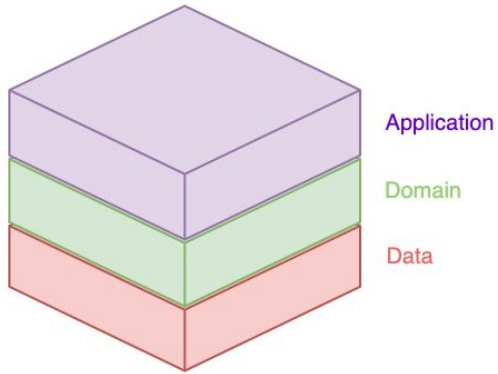
Funções Sem Efeitos Colaterais

```
func (f *Foo) Do(obj *Object) {
    obj.Field = "Pending"
}
```

```
func (f *Foo) Do(obj *Object) {
    f.Counts++
}
```

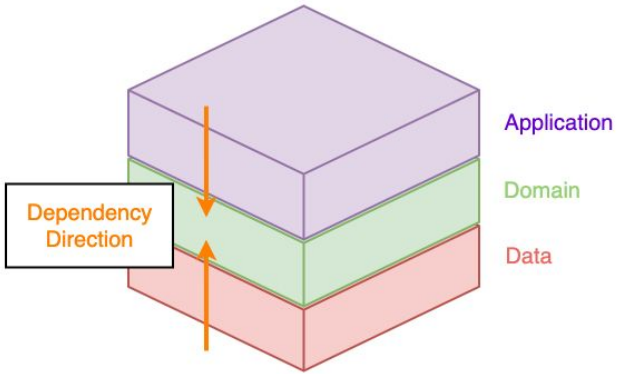
Arquitetura em Camadas

- **Qual Arquitetura?**
  - Qualquer uma que isole o domínio de detalhes técnicos.
  - Exemplos: **Clean Architecture** e **Hexagonal Architecture**



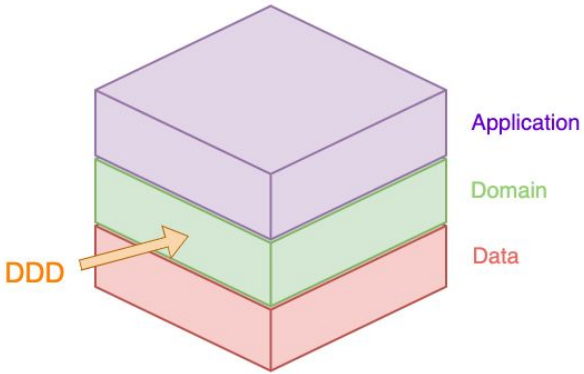
Arquitetura em Camadas

- **Qual Arquitetura?**
  - Qualquer uma que isole o domínio de detalhes técnicos.
  - Exemplos: **Clean Architecture** e **Hexagonal Architecture**

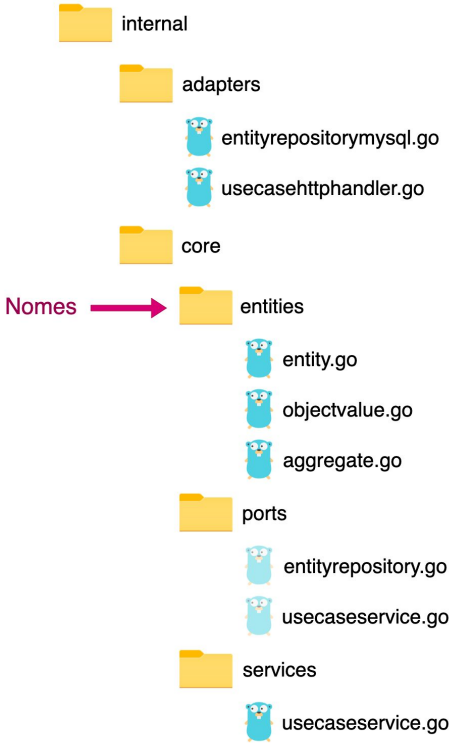


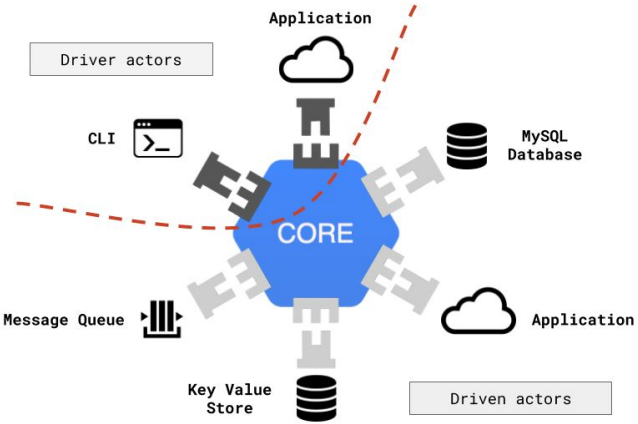
Arquitetura em Camadas

- **Qual Arquitetura?**
  - Qualquer uma que isole o domínio de detalhes técnicos.
  - Exemplos: **Clean Architecture** e **Hexagonal Architecture**

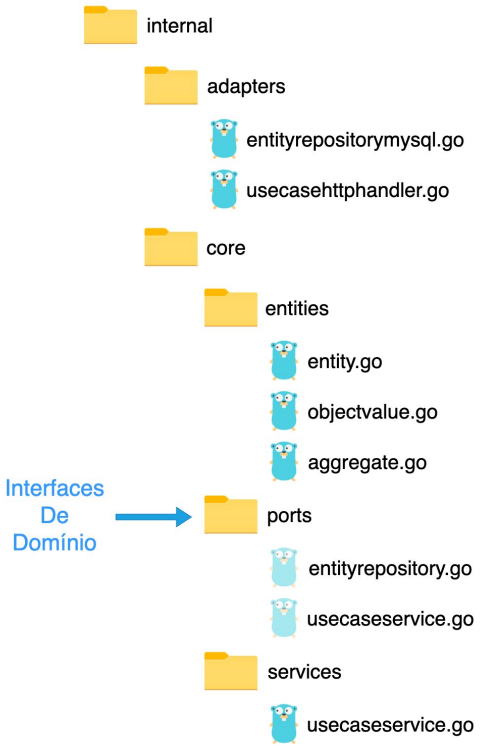


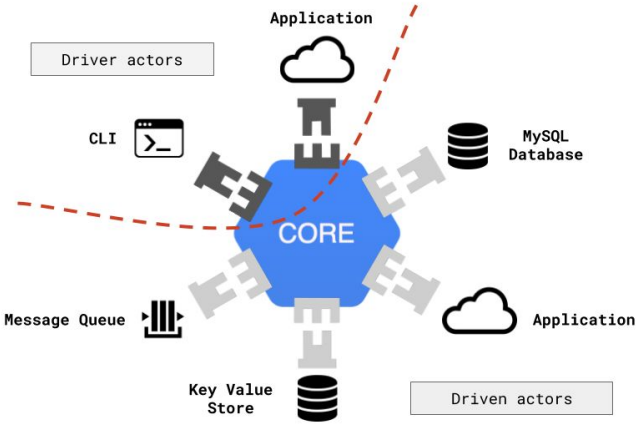
Arquitetura Hexagonal + Model-Driven Design



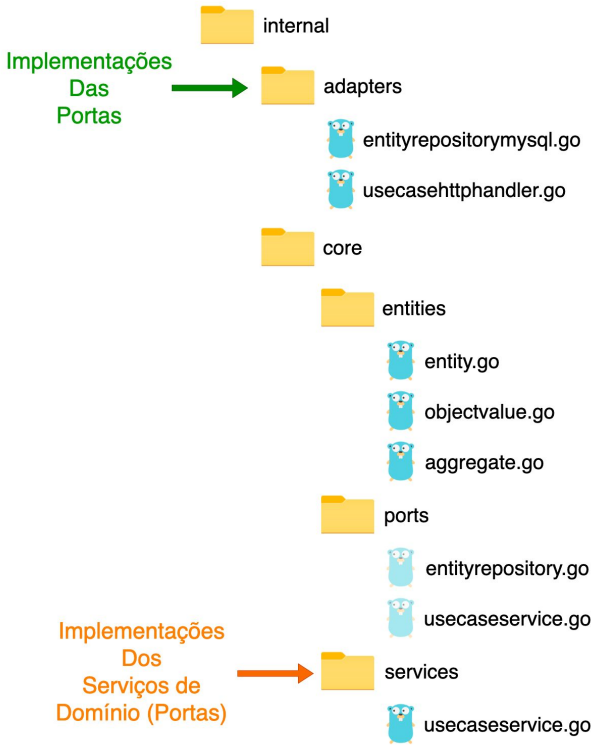


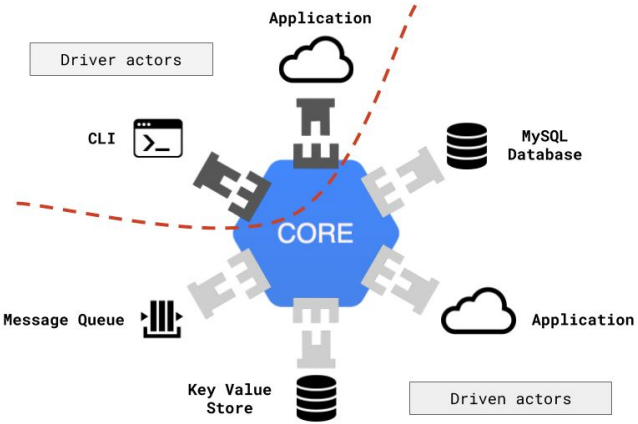
Arquitetura Hexagonal + Model-Driven Design





Arquitetura Hexagonal + Model-Driven Design





Recapitulando

Prototipagem

1. Surge um problema para resolver;
2. Parte do time de desenvolvimento busca entender o problema junto aos **domain experts**;
3. Cria-se uma RFC expondo de forma clara os requisitos técnicos e funcionais;
4. Constrói-se o primeiro modelo do sistema utilizando Model-Driven Design dentro do contexto delimitado;
5. Valída-se junto aos **domain experts** se o problema foi definido de maneira correta e se a solução proposta é de fato uma solução;

Execução

1. Durante a implementação surgem novas questões que direcionam a solução para um caminho mais robusto;
2. O código é refatorado;
3. O modelo da RFC é atualizado;
4. A solução é validada;

- **1, 2, 3 e 4 são realizados de forma cíclica;**
- **Os ciclos podem ocorrer durante melhorias contínuas;**
- **É totalmente aplicável Agile (sprints, feedback do usuário, definições de MVPs);**
- **As entregas passam por um processo de CI e testes automatizados são necessários para validação;**

Referências

[1] - Domain-Driven Design: Quickly (https://www.infoq.com/minibooks/domain-driven-design-quickly/)
[2] - Domain Driven Design (DDD) - Daniel Cukier (https://www.youtube.com/watch?v=bsDwFWuFveo)
[3] - Domain-Driven Design: Tackling Complexity In The Heart of Software - Eric Evans (https://www.amazon.com.br/Domain-Driven-Design-Tackling-Complexity-Software/dp/0321125215)
[4] Hexagonal Architecture in Go - Matías Varela (https://medium.com/@matiasvarela/hexagonal-architecture-in-go-cfd4e436faa3)
[5] - Implementing Domain-Driven Design - Vaughn Vernon (https://www.amazon.com.br/Implementing-Domain-Driven-Design-Vaughn-Vernon/dp/0321834577)
[6] - Patterns, Principles, and Practices of Domain-Driven Design - Scott Millett (https://www.amazon.com.br/Patterns-Principles-Practices-Domain-Driven-Design/dp/1118714709)

2021

Obrigado!

mercado envíos