**Domain Driven Design ou Projeto Orientado a Domínio.**

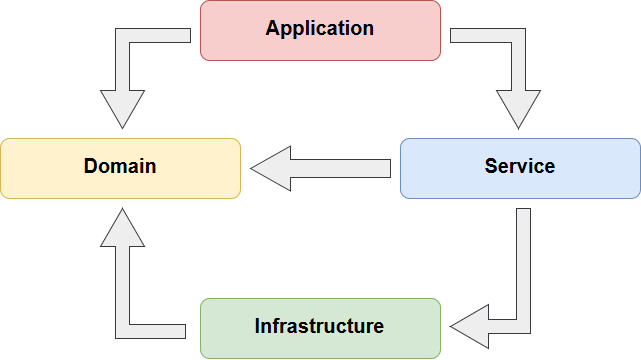
**DDD não é arquitetura em camadas.**

O DDD é uma abordagem de modelagem de software que segue um conjunto de práticas com objetivo de facilitar a implementação de complexas regras / processos de negócios que tratamos como **domínio**.

## Vantagens de usar DDD

Escalável,testável, de manutenção fácil e tranquila, escrito com boas práticas.

# **Entendendo a arquitetura utilizada**



Derivação da arquitetura DDD

# Entendendo a arquitetura com o DDD

## Todas as camadas:



## 01- Apresentação



Camada responsável por abranger tudo o que diz respeito a interface de usuário (UI):  
- Aplicação Desktop (WinForms)  
- Aplicação Web (Angular, React, Vue)  
- Aplicação Mobile (Android)

## 02 - Serviços



Toda forma de comunicação remota acontecerá aqui, muito usada em aplicação web, e nem sempre é implementada quando se trata de uma aplicação que consegue e suporta comunicar com a camada de **Aplicação**(a seguir).  
Algumas das implementações mais comuns:  
- Web API (Http)  
- SignalR  
- WebSockets

Camada de serviço: seria o “coração” do projeto, pois é nela que é feita todas as regras de negócio e todas as validações, antes de persistir os dados no banco de dados.  
Possui referências das camadas Domain, Infra.Data e Infra.CrossCutting.

## 03 - Aplicação



Camada responsável por fazer a(s) aplicação(s) se comunicar diretamente com o Domínio. Nela são implementados:  
- Classes dos Serviços da aplicação  
- Interfaces (contratos)  
- DataTransferObjects (DTO)  
- AutoMapper

Responsável pelo projeto principal, pois é onde será desenvolvido os controladores e serviços da API. Tem a função de receber todas as requisições e direcioná-las a algum serviço para executar uma determinada ação.  
Possui referências das camadas Service e Domain.

## 04 - Domínio



Aqui é onde o **DDD** acontece! E nela nós temos:  
- Entidades (classes/modelos)  
- Interfaces (contratos) para Serviços e Repositórios  
- Classes dos Serviços do domínio  
- Validações (se necessário)

Camada de domínio: responsável pela implementação de classes/modelos, as quais serão mapeadas para o banco de dados, além de obter as declarações de interfaces, constantes, DTOs (Data Transfer Object) e enums.

## 05 - Infraestrutura



Camada que da suporte as demais camadas. Que atualmente é dividida por duas camadas com seus respectivos conteúdos:

**Data:**- Repositórios  
- DataModel (Mapeamento)  
- Persistência de dados

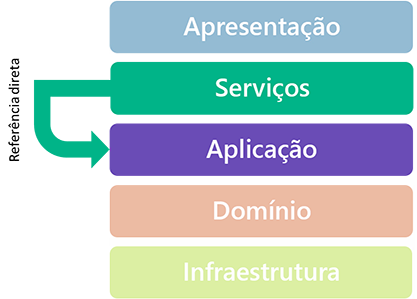
**CrossCutting (camada que atravessa todas as outras, portando possui referência de todas elas):**- IoC (Inversão de controle)

Camada de infraestrutura é dividida em duas sub-camadas  
- Data: realiza a persistência com o banco de dados, utilizando, ou não, algum ORM.  
- Cross-Cutting: uma camada a parte que não obedece a hierarquia de camada. Como o próprio nome diz, essa camada cruza toda a hierarquia. Contém as funcionalidades que pode ser utilizada em qualquer parte do código, como, por exemplo, validação de CPF/CNPJ, consumo de API externa e utilização de alguma segurança.  
Possui referências da camada Domain.

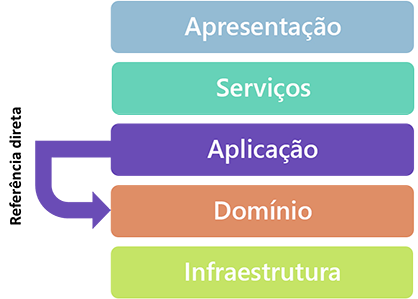
# Quais as camadas que se comunicam umas com as outras



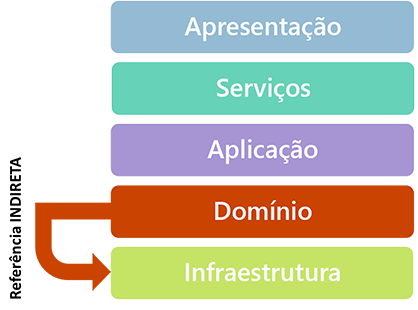
**01 - Apresentação:**recebe referência de “Aplicação”, “Domínio” e da “CrossCutting” (em “Infraestrutura”), em casos de aplicações front-end a comunicação é feita unicamente com a cama de “Serviços” (API, por exemplo).



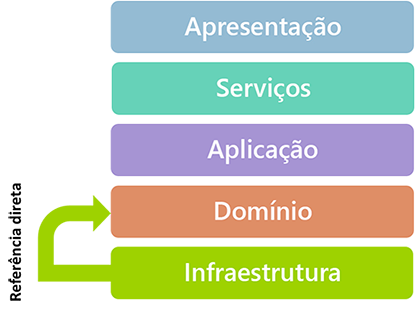
**02 - Serviços:**recebe referência de “Aplicação”.



**03 - Aplicação:**recebe referência de “Domínio”.



**04 - Domínio:**embora seja a camada que mais da suporte às outras camadas, ela é a única que não recebe referência de ninguém, logo ela não depende de nada! Porem como mostra a imagem, ela se comunica de forma “indireta” com a camada Data (Infraestrutura), e isso só é possível graças à interfaces (sim, aquela que você assina o nome dos métodos, coisa e tal).



**05 - Infraestrutura:**por último, e não menos importante, temos esta camada que (como dito anteriormente) possui “subcamadas” **Data**e **CrossCutting**, onde recebem referência do domínio.

**Data**: tem o objetivo de persistir dados ou qualquer outra comunicação externa.  
**CrossCurring (Ioc)**: é onde registrados todas as interfaces e classes existente no projeto, para que o mesmo seja responsável por instanciar a árvore de dependências de toda a arquitetura.

## Uma observação a arquitetura:

Como podemos ver, todas as camadas possuem uma numeração sequencial, e isso é muito importante, pois é exatamente desta forma em que o fluxo da arquitetura funciona, desde a interface gráfica até a persistência de informações no banco de dados.