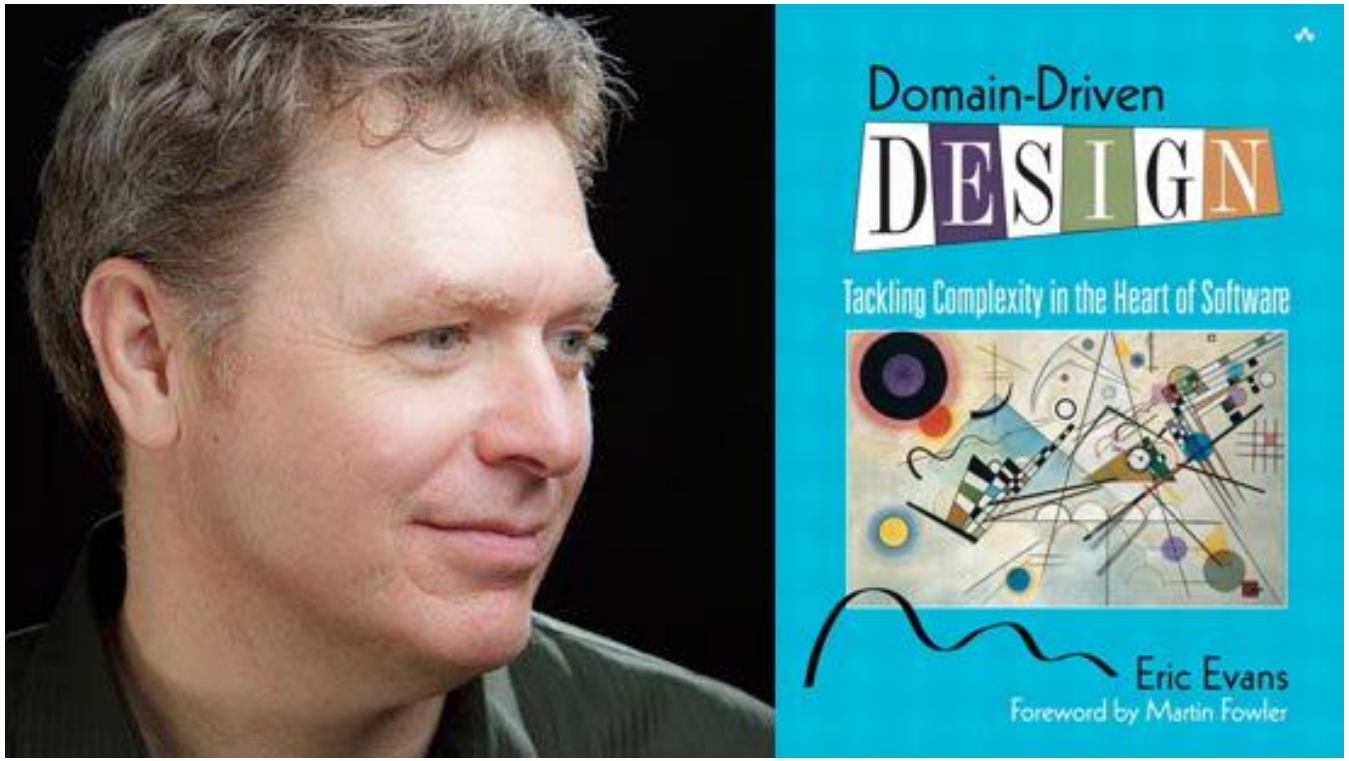
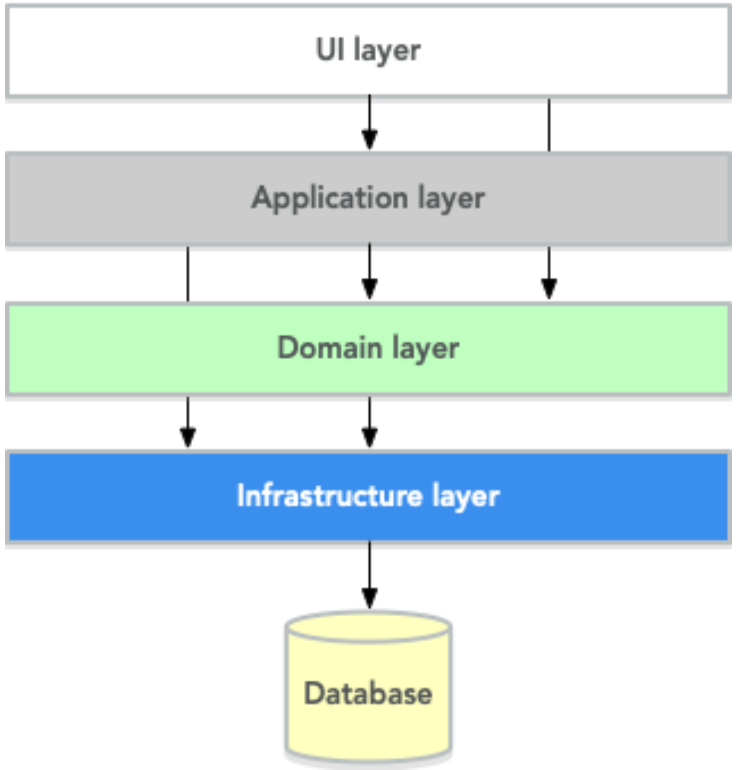
DDD – DOMAIN DRIVEN DESIGN

www.cotiinformatica.com.br

O que é DDD?

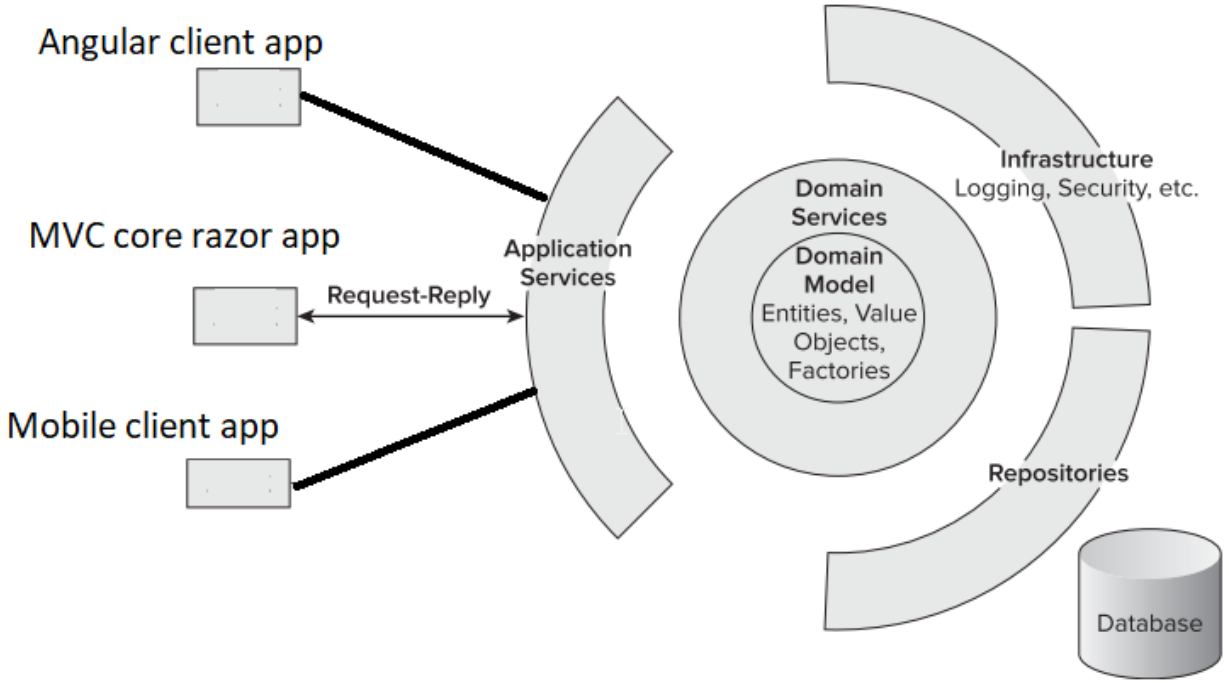
Domain Driven Design significa Projeto Orientado a Domínio. Ele veio do título do livro escrito por Eric Evans, dono da DomainLanguage, uma empresa especializada em treinamento e consultoria para desenvolvimento de software. O livro de Evans é um grande catálogo de Padrões, baseados em experiências do autor ao longo de mais de 20 anos desenvolvendo software utilizando técnicas de Orientação a Objetos.





O que é DDD?

A principal ideia do DDD (Domain Driven Design) é a de que o mais importante em um software não é o seu código, nem sua arquitetura, nem a tecnologia sobre a qual foi desenvolvido, mas sim o problema que o mesmo se propõe a resolver, ou em outras palavras, a regra de negócio.
Ela é a razão do software existir, por isso deve receber o máximo de tempo e atenção possíveis.
Em praticamente todos os projetos de software, a complexidade não está localizada nos aspectos técnicos, mas sim no negócio, na atividade que é exercida pelo cliente ou problema que o mesmo possui.



O que é DDD?

Alinhamento do código com o negócio

- Contato dos desenvolvedores com os especialistas do domínio é algo essencial quando se faz DDD (o pessoal de métodos ágeis já sabe disso faz tempo)

Favorecer reutilização

- A codificação é voltada para aproveitar um mesmo conceito de domínio ou um mesmo código em vários lugares

O que é DDD?

Mínimo de acoplamento

- Com um modelo bem feito, organizado, as várias partes de um sistema interagem sem que haja muita dependência entre módulos ou classes de objetos de conceitos distintos

Independência da Tecnologia

- DDD não foca em tecnologia, mas sim em entender as regras de negócio e como elas devem estar refletidas no código e no modelo de domínio. Não que a tecnologia usada não seja importante, mas essa não é uma preocupação de DDD

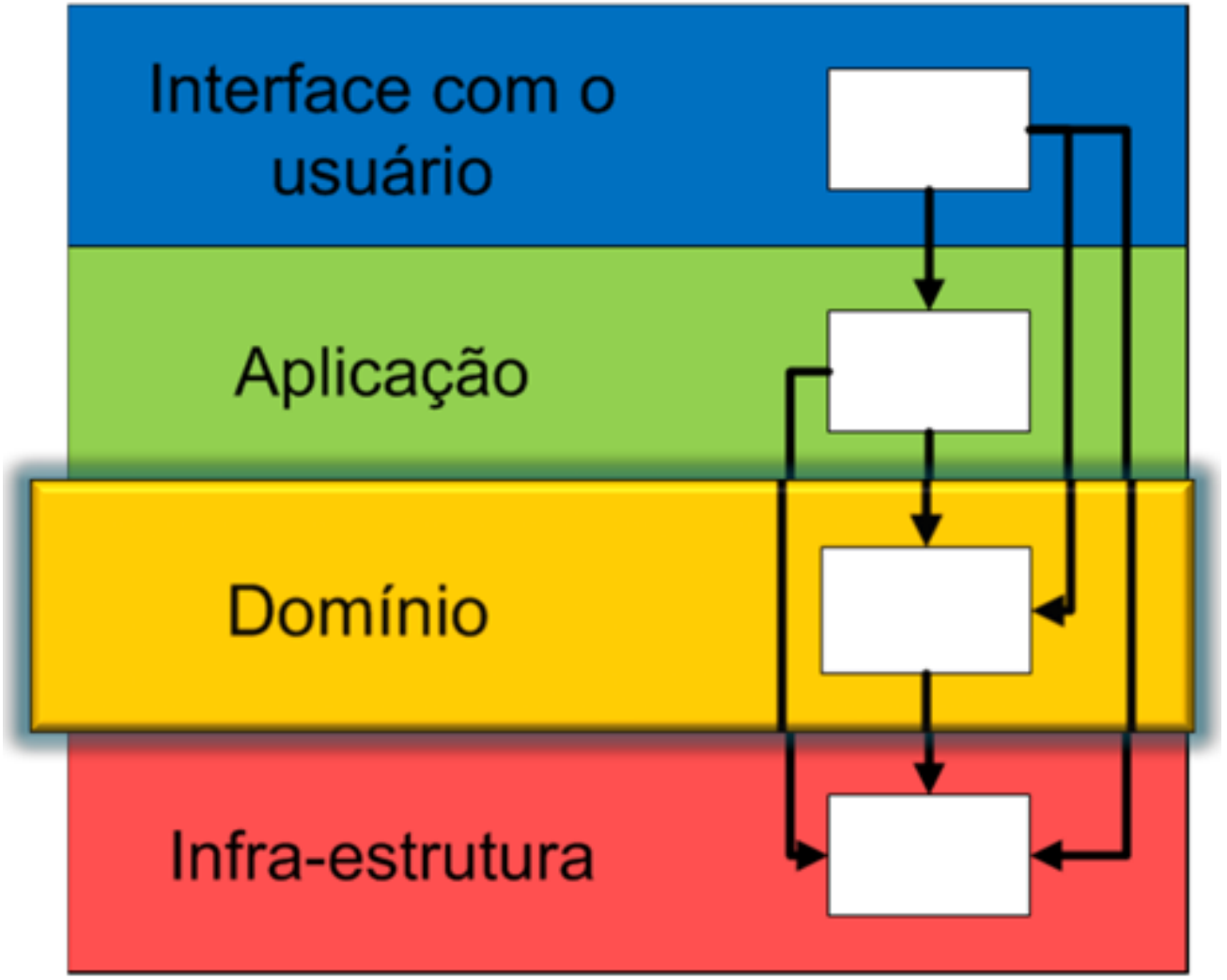
O que é DDD?

Uma vez que decidimos criar um modelo usando MDD, precisamos, inicialmente, isolar o modelo de domínio das demais partes que compõem o sistema.

Extrair a essência do domínio, dentre milhares de linhas de código de um sistema complexo nem sempre é fácil.

O trabalho de refinamento e busca de uma visão clara é contínuo.

A refatoração é um processo incessante de busca por melhorias de projeto. Aplicar DDD é utilizar Padrões com o objetivo de extrair e reconhecer a essência de um sistema.



O que é DDD?

Essa separação pode ser feita utilizando-se uma arquitetura em camadas, que dividirá nossa aplicação em quatro partes:

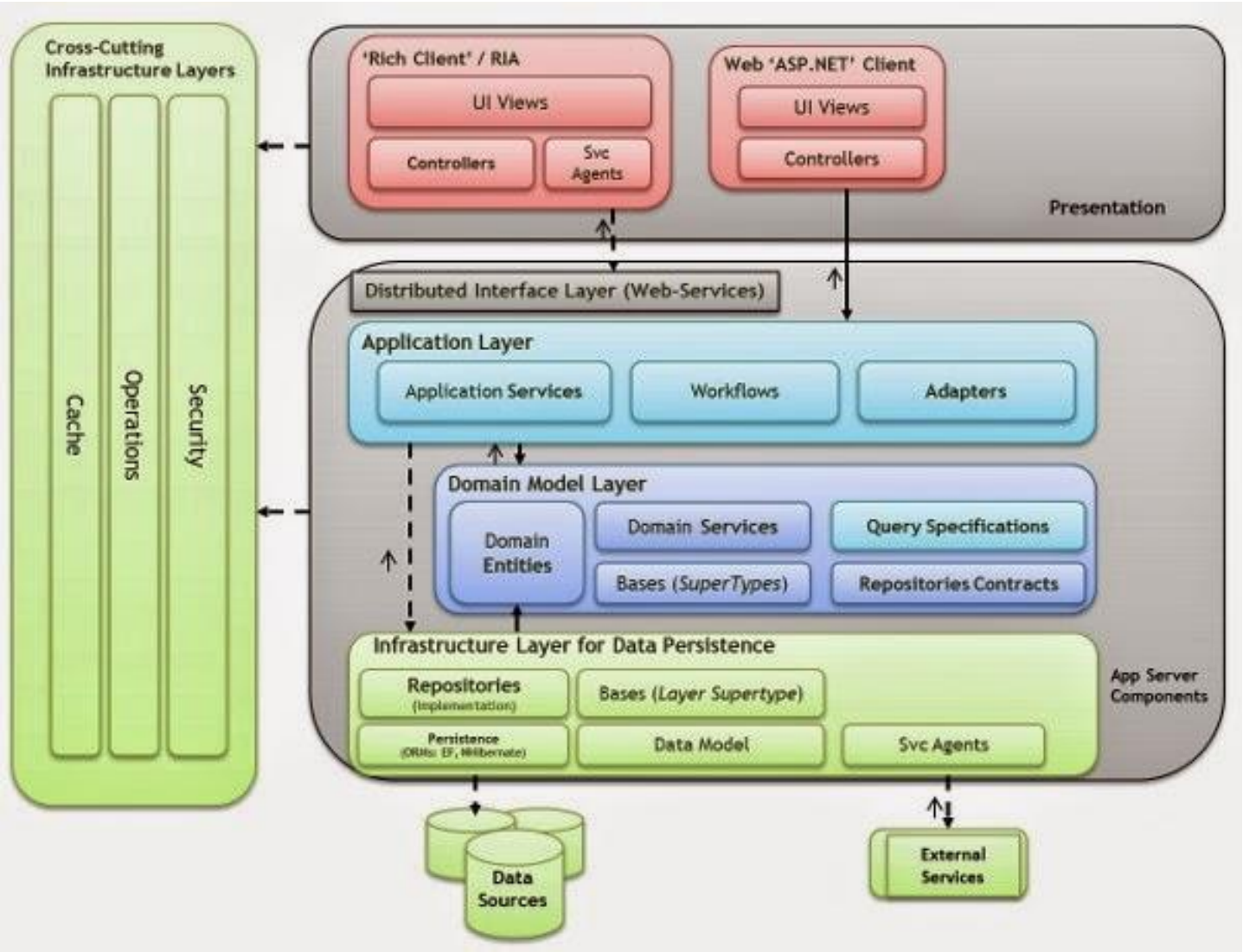
Interface de Usuário – parte responsável pela exibição de informações do sistema ao usuário e também por interpretar comandos do usuário;

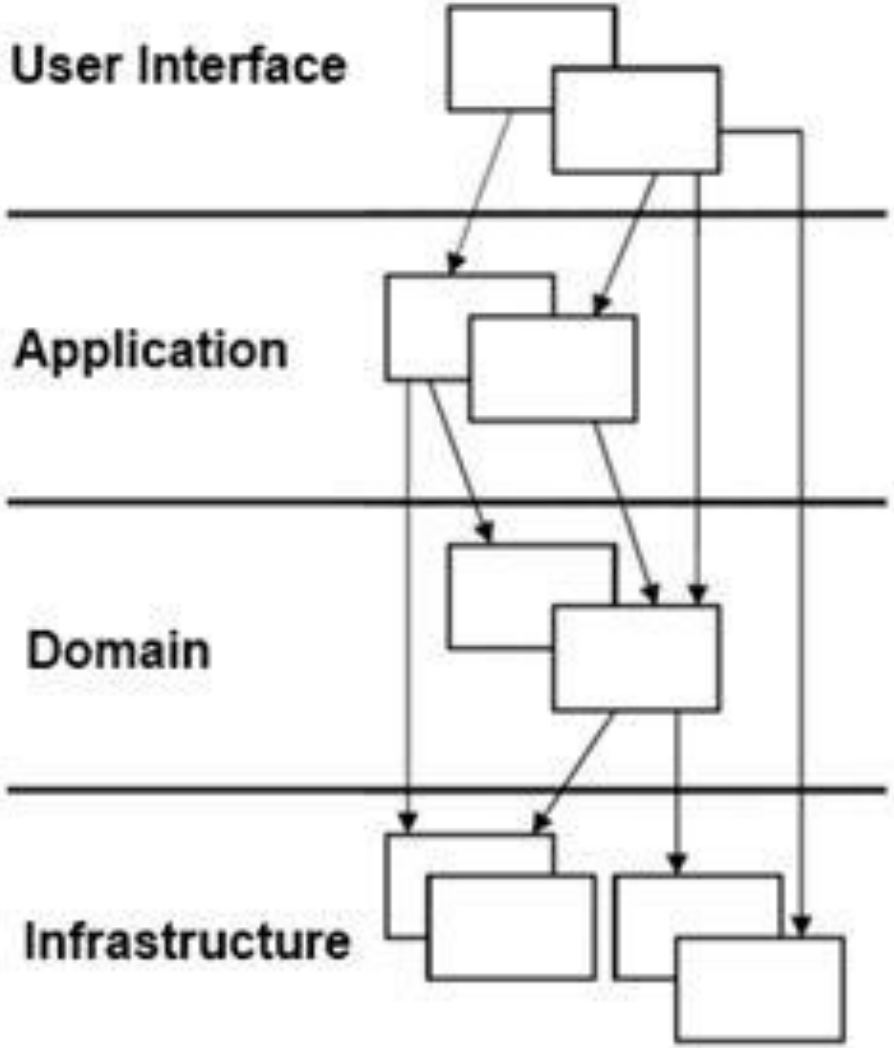
Aplicação – essa camada não possui lógica de negócio. Ela é apenas uma camada fina, responsável por conectar a Interface de Usuário às camadas inferiores;

Domínio – representa os conceitos, regras e lógicas de negócio. Todo o foco de DDD está nessa camada. Nosso trabalho, daqui para frente, será aperfeiçoar e compreender profundamente essa parte;

Infra-estrutura – fornece recursos técnicos que darão suporte às camadas superiores. São normalmente as partes de um sistema responsáveis por persistência de dados, conexões com bancos de dados, envio de mensagens por redes, gravação e leitura de discos, etc.

O que é DDD?


Cross-Cutting Infrastructure Layers
Cache
Operations
Security
'Rich Client' / RIA
UI Views
Controllers
Svc Agents
Web 'ASP.NET' Client
UI Views
Controllers
Presentation
Distributed Interface Layer (Web-Services)
Application Layer
Application Services
Workflows
Adapters
Domain Model Layer
Domain Entities
Domain Services
Query Specifications
Bases (SuperTypes)
Repositories Contracts
Infrastructure Layer for Data Persistence
Repositories
(Implementation)
Bases (Layer Supertype)
Persistence
(ORMs: EF, NHibernate)
Data Model
Svc Agents
App Server Components
Data Sources
External Services


User Interface
Application
Domain
Infrastructure

DOMAIN

www.cotiinformatica.com.br

DOMAIN

Modelo do domínio, serviços e interfaces dos repositórios. Esta é a parte central do aplicativo. A linguagem obiqua é usada nessas classes, interfaces e assinaturas dos métodos, e todo conceito dentro dessa camada é familiar para o especialista no domínio.

O Domínio é composto de:

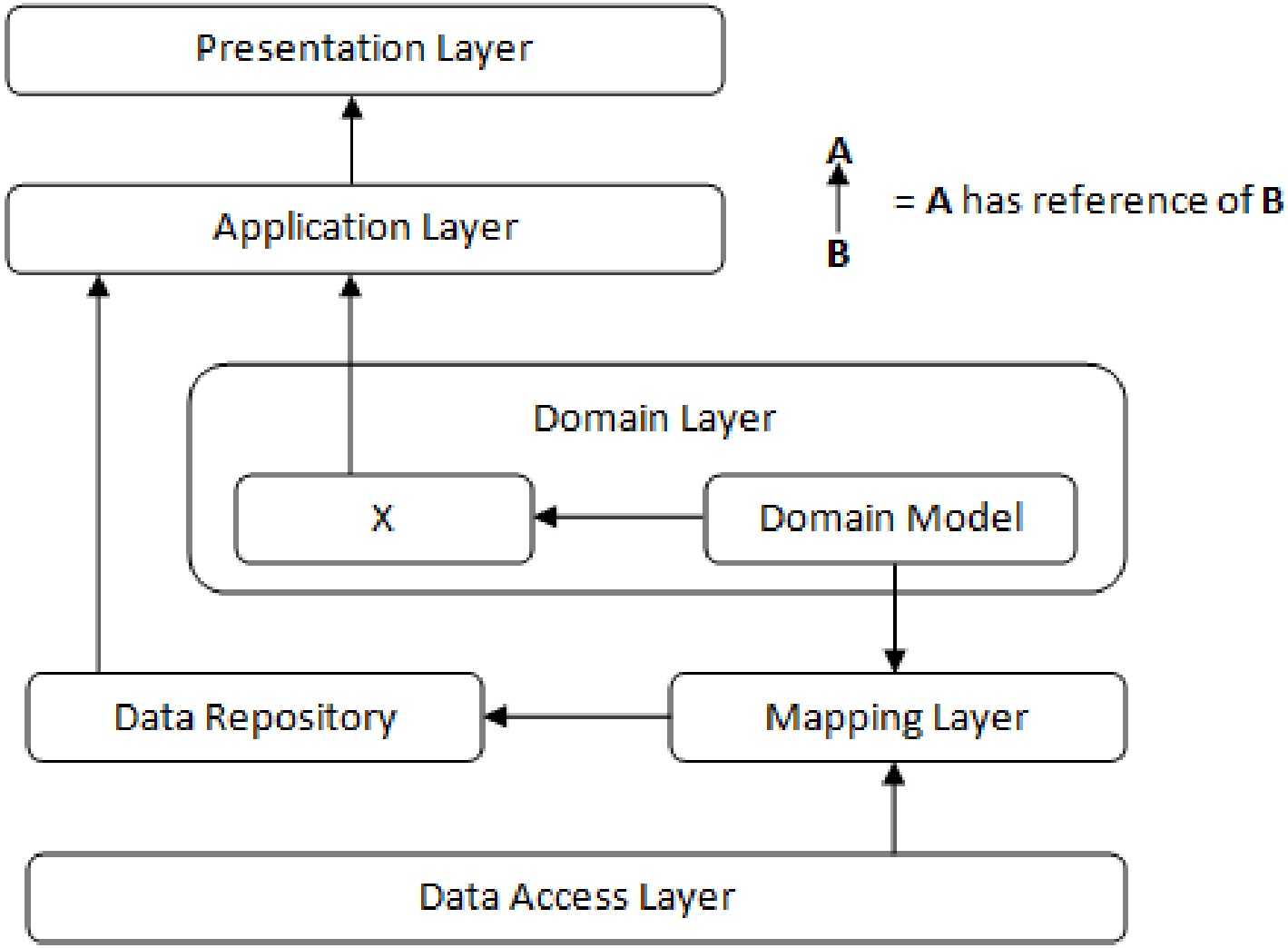
Domain Models (Modelos de domínio)

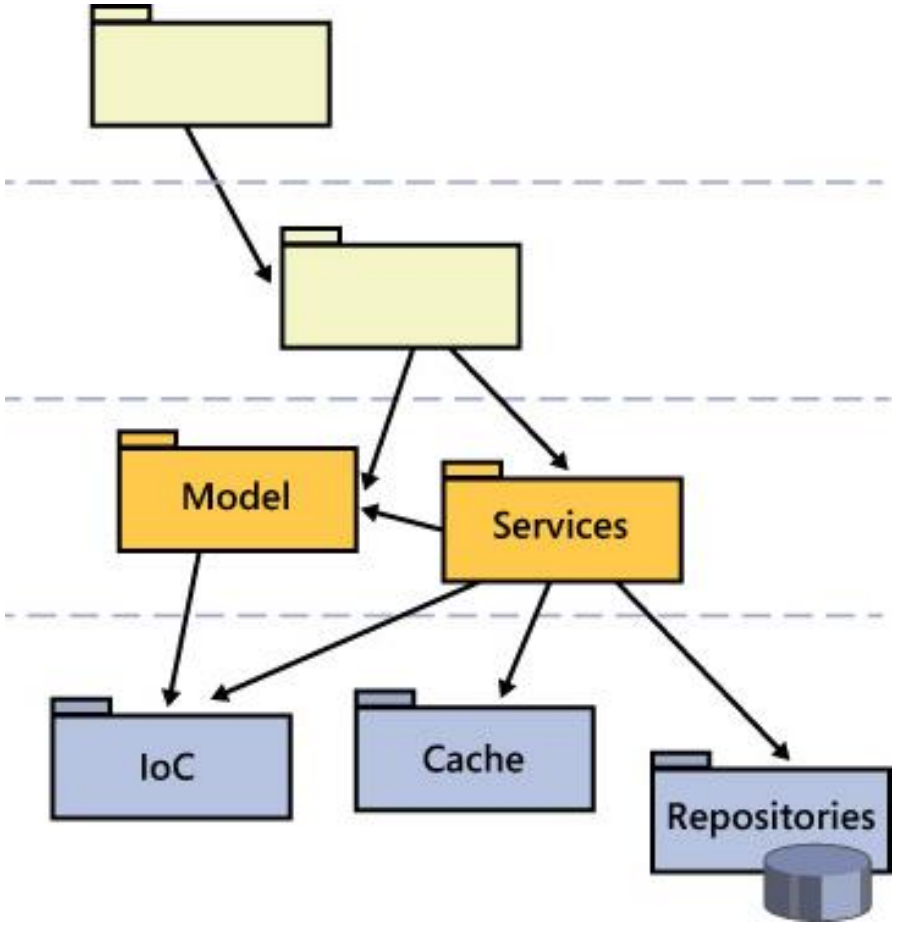
O padrão *Entity* define que todo sistema possui ao menos uma entidade, ou seja, todo sistema possui ao menos um objeto o qual irá refletir um conceito do domínio, **com sua respectiva identidade e estado**.

Domain Services (Serviços de domínio)

Os serviço do domínio (*Domain Services*) são classes que tem como objetivo serem uma alternativa para o **desacoplamento de código**. Os serviços do domínio surgem em cenários onde a escolha de dar responsabilidade a uma classe ou outra poderia causar problemas com **acoplamento do código**.
Ou ainda, quando uma responsabilidade nova não se encaixa em nenhuma das Entidades já definidas.

DOMAIN

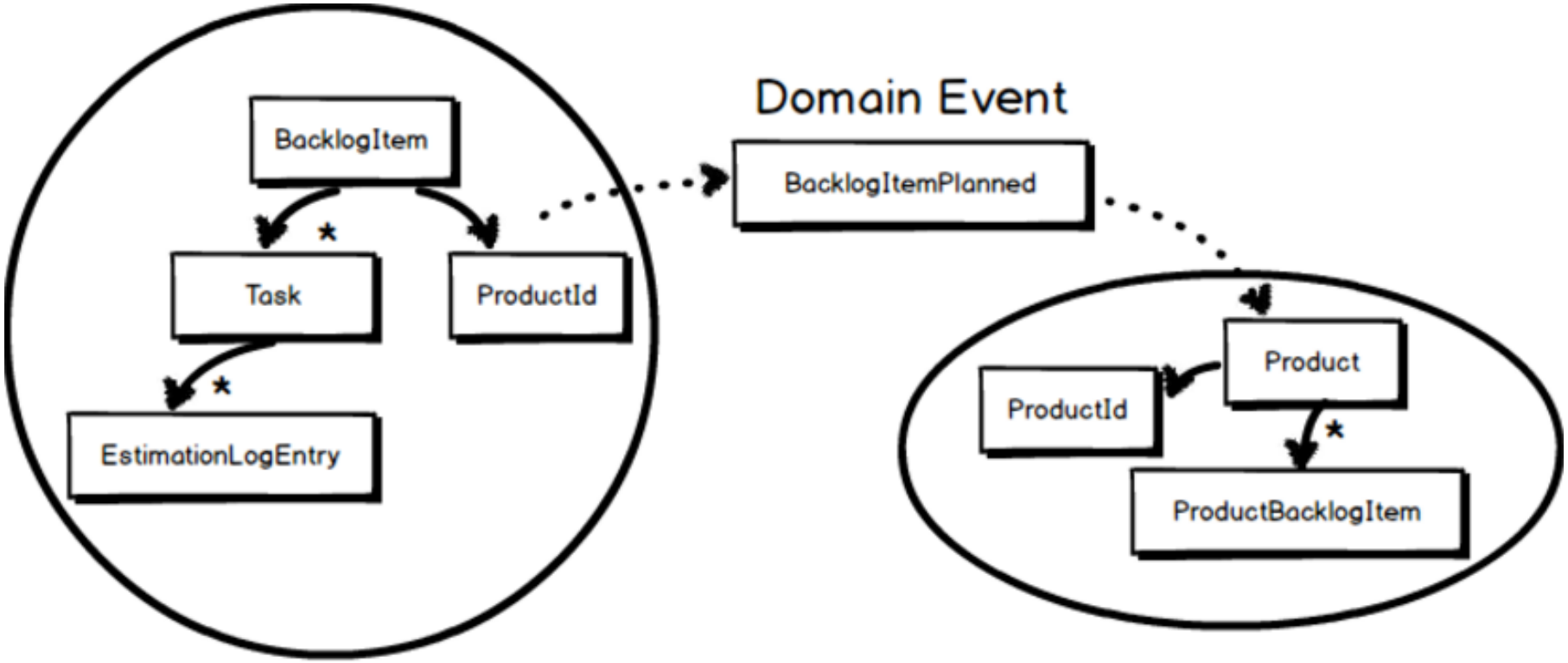

Presentation Layer
Application Layer
Domain Layer
X
Domain Model
Data Repository
Mapping Layer
Data Access Layer
A
B
= A has reference of B


Model
Services
IoC
Cache
Repositories

DOMAIN

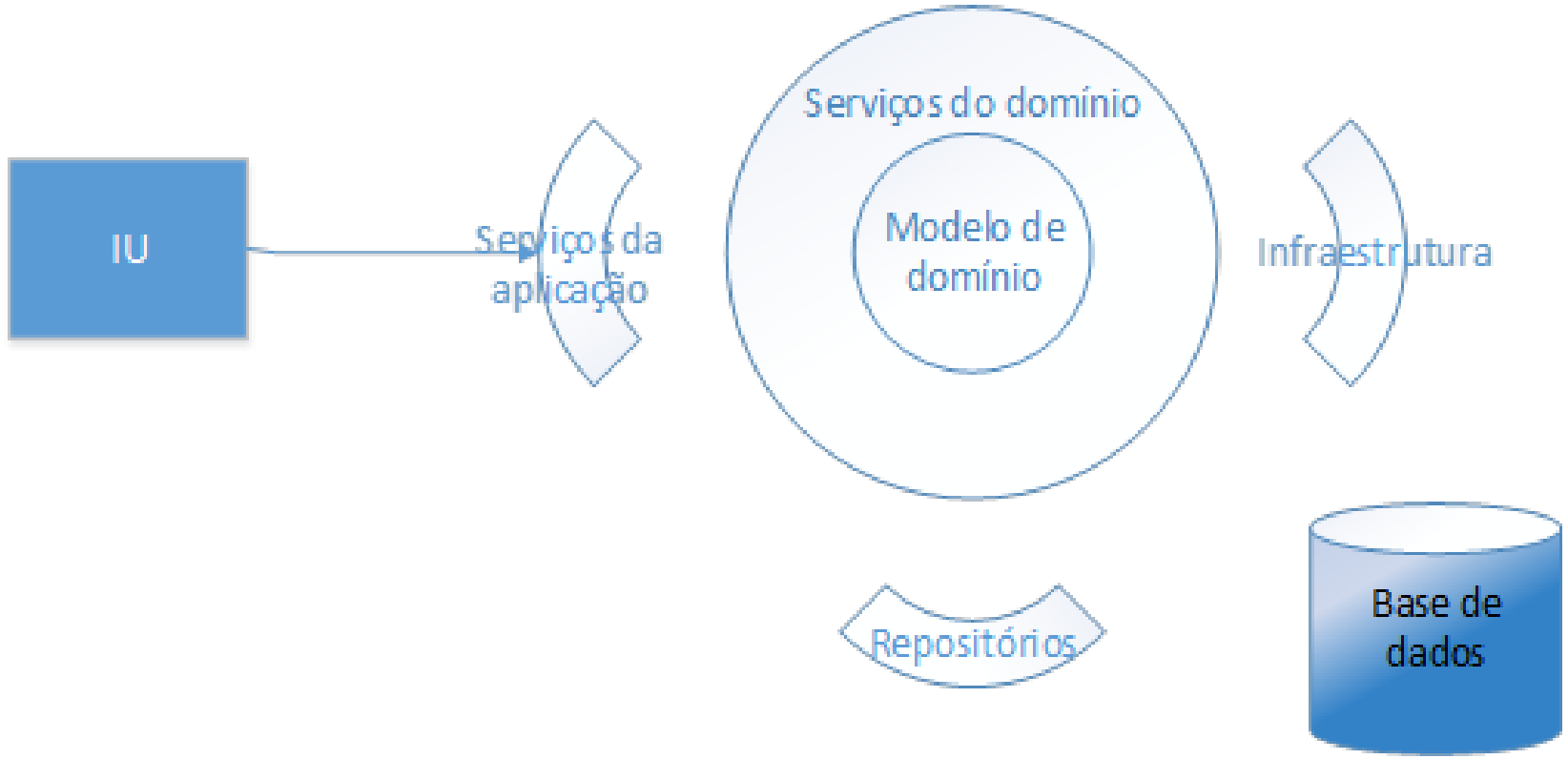
Aggregates

Os agregados *(aggregate)* são grupos de objetos associados, onde apenas um deles é visível à chamadas externas. Este único objeto visível externamente é chamado de **raíz** do agregado. Ainda, cada agregado, além de possuir uma entidade raiz, possui também um limite (o que está dentro do agregado).

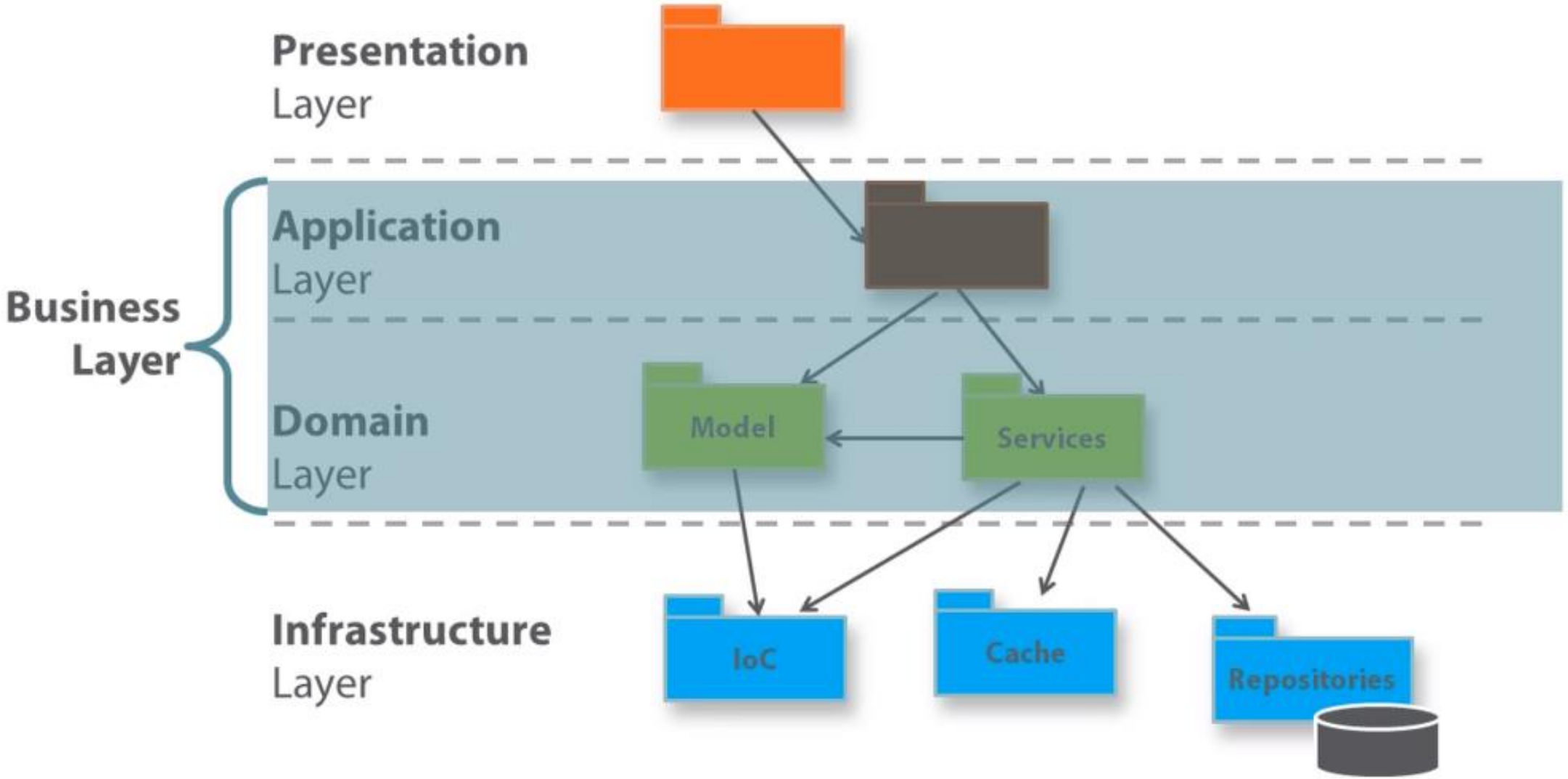


DOMAIN

A ideia básica do DDD está centrada no conhecimento do problema para o qual o software foi proposto. Na prática, trata-se de uma coleção de padrões e princípios de design que buscam auxiliar o desenvolvedor na tarefa de construir aplicações que reflitam um entendimento do negócio. É a construção do software a partir da modelagem do domínio real como classes de domínio que, por sua vez, possuirão relacionamentos entre elas.



DOMAIN

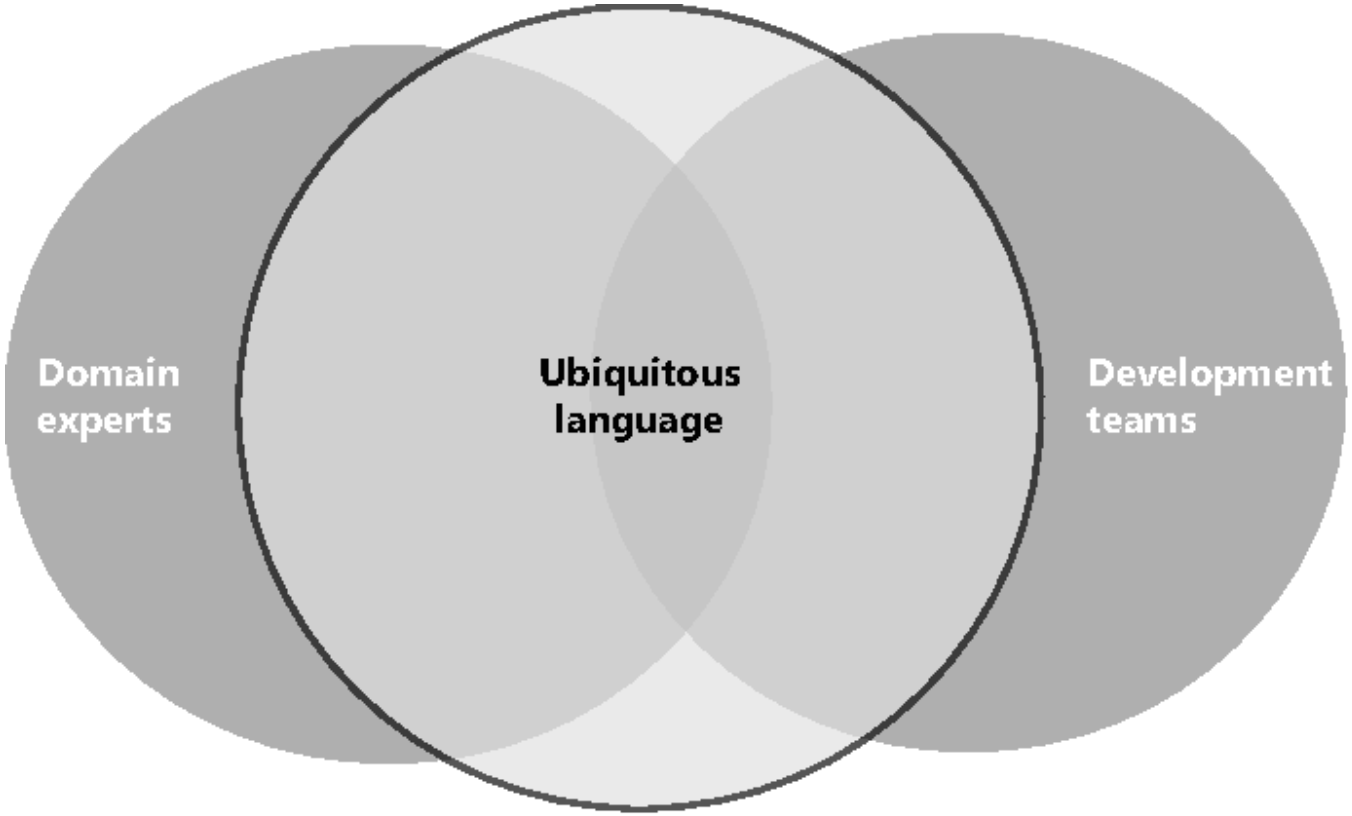

Presentation
Layer
Application
Layer
Business
Layer
Domain
Layer
Model
Services
Infrastructure
Layer
IoC
Cache
Repositories

Ubiquitous Language

Ubiquitous Language é um conceito muito comum em DDD(Domain-driven Design).
Basicamente é um dos conceitos que o DDD utiliza, que tem como finalidade: “falar” a língua do usuário/cliente . Manter uma única linguagem de domínio que seja entendível tanto para os desenvolvedores quanto para o cliente.

O que o DDD com o Ubiquitous Language fala é bem simples:

“Todo comportamento do seu sistema deveria estar implementado em classes cujos nomes devem fazer parte do domínio do problema, do domínio do cliente.”



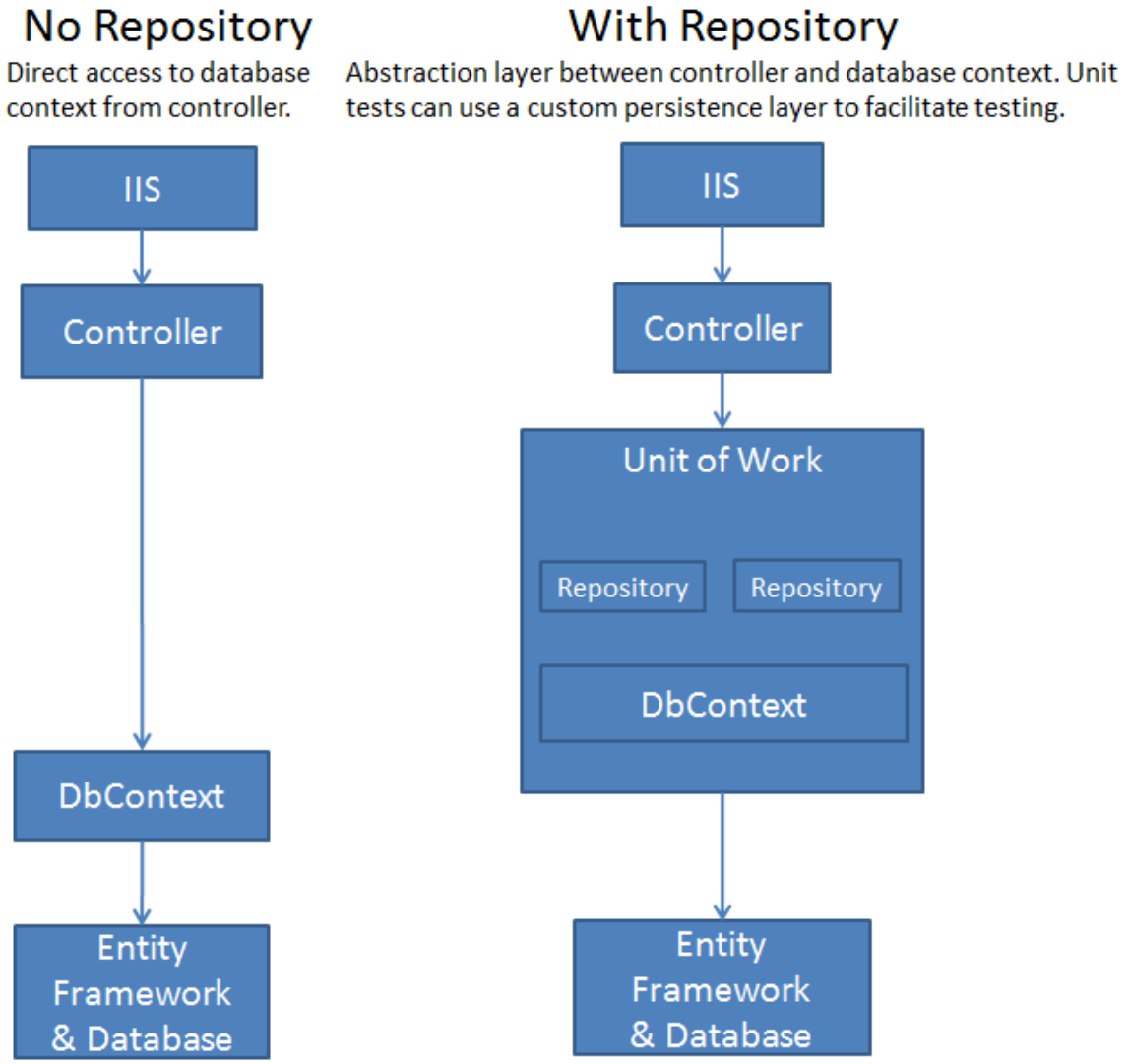
INFRA STRUCTURE

www.cotiinformatica.com.br

INFRA STRUCTURE

Os componentes de persistência de dados fornecem acesso aos dados hospedados dentro dos limites de um microserviço (ou seja, o banco de dados de um microserviço). Eles contêm a implementação real dos componentes, como repositórios e classes Unidade de Trabalho, como objetos DbContext personalizados do EF (Entity Framework). O DbContext do EF implementa os padrões de Repositório e de Unidade de Trabalho.

Repositórios são classes ou componentes que encapsulam a lógica necessária para acessar fontes de dados. Eles centralizam a funcionalidade comum de acesso a dados, melhorando a sustentabilidade e desacoplando a infraestrutura ou a tecnologia usada para acessar os bancos de dados da camada do modelo de domínio.

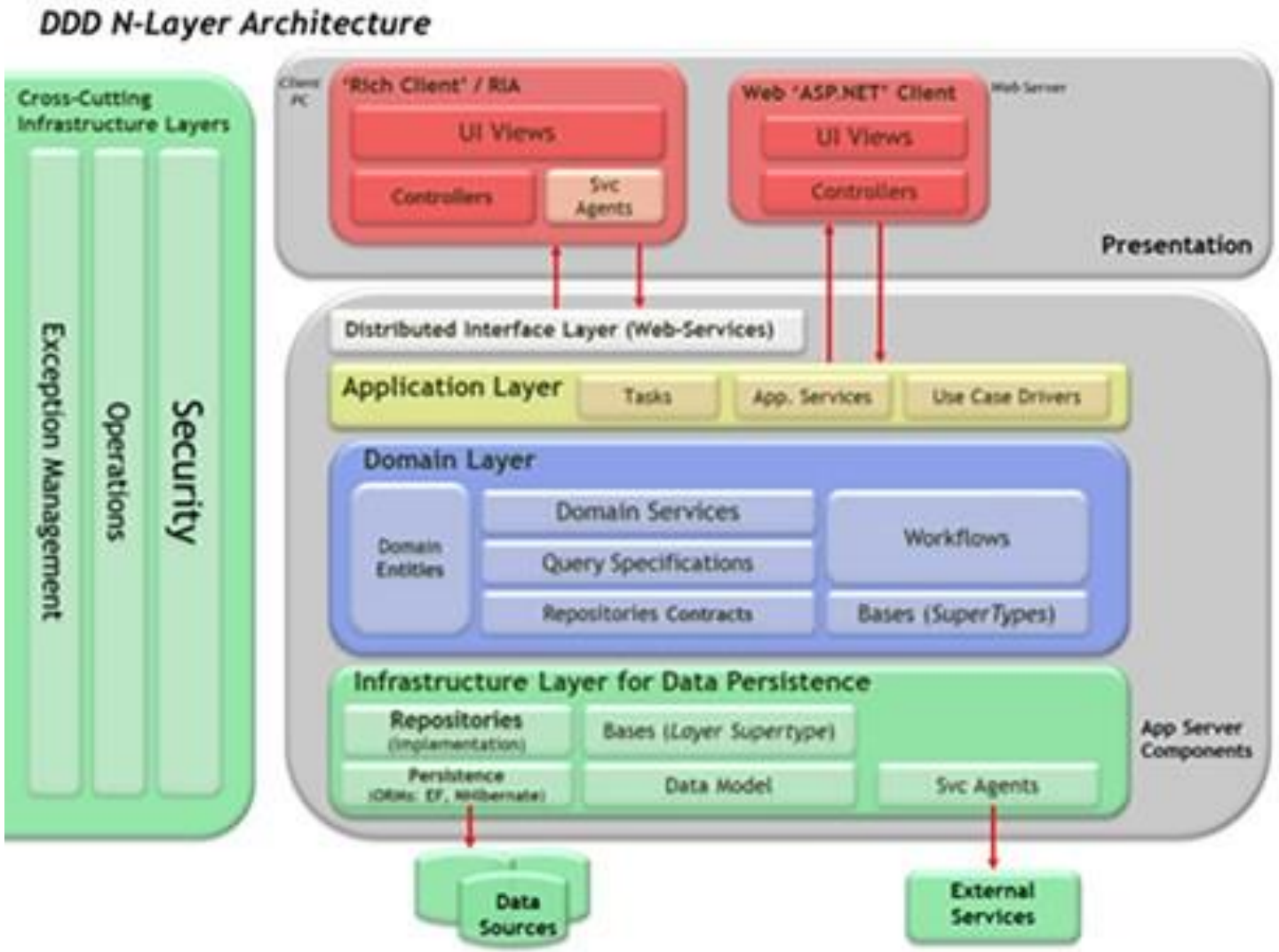


INFRA STRUCTURE

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.



APPLICATION

www.cotiinformatica.com.br

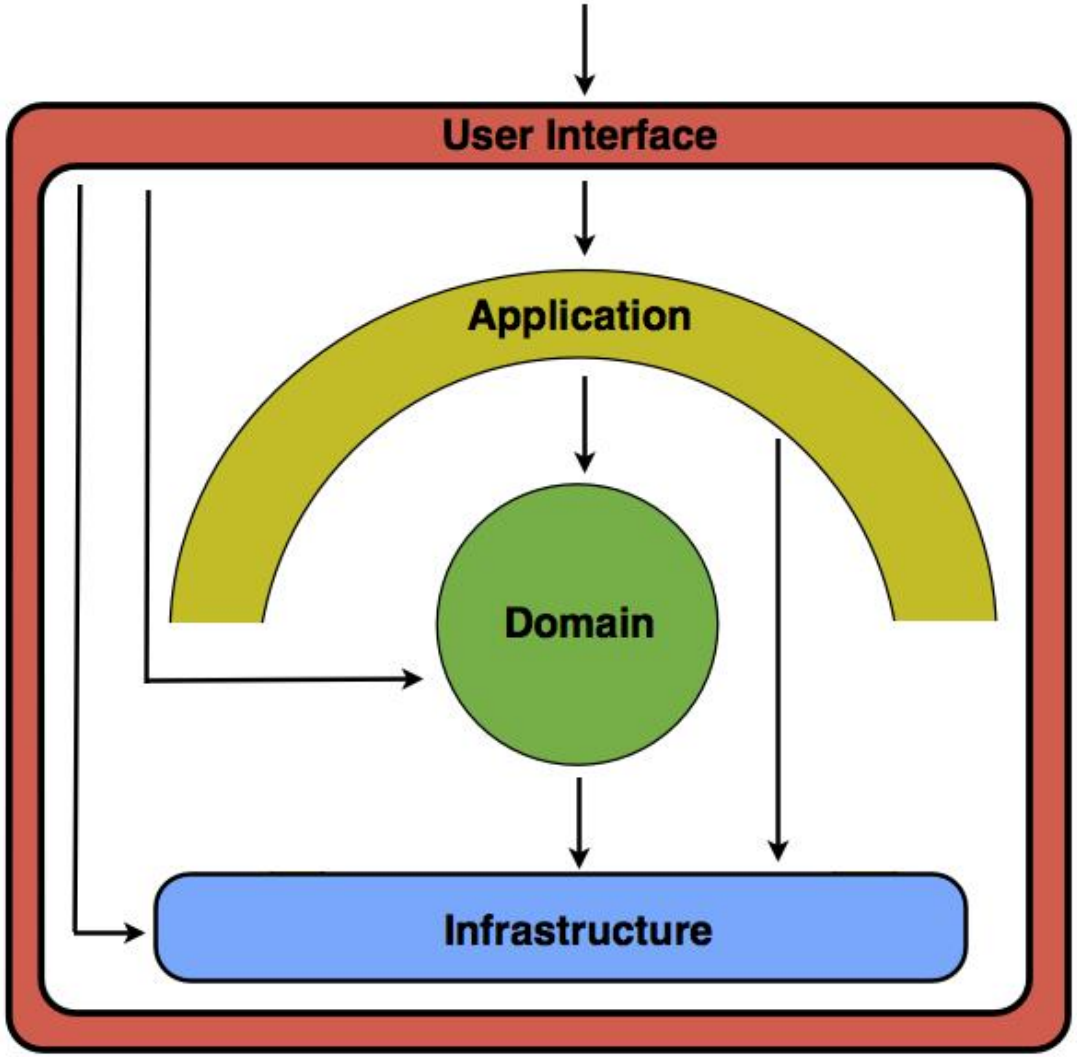
APPLICATION

Na camada de aplicação não é implementada nenhuma regra de negócio, ela somente coordena a execução de uma tarefa e delega para os objetos de domínio na camada inferior.

Camada responsável por fazer a(s) aplicação(s) se comunicar diretamente com o Domínio.

Nela são implementados:

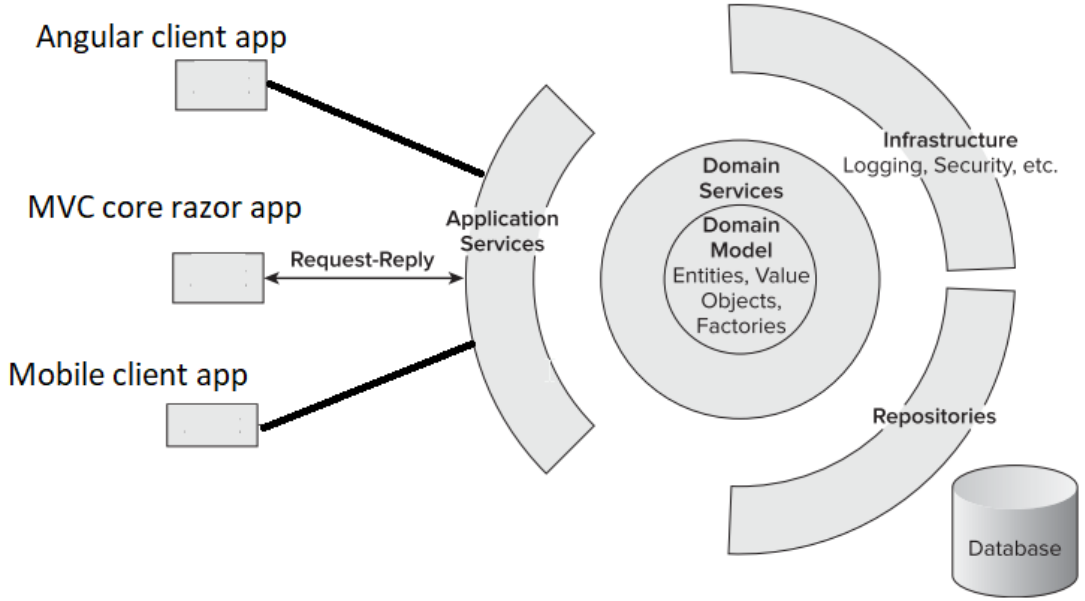
- Classes dos Serviços da aplicação
- Interfaces (contratos)
- DataTransferObjects (DTO)
- AutoMapper

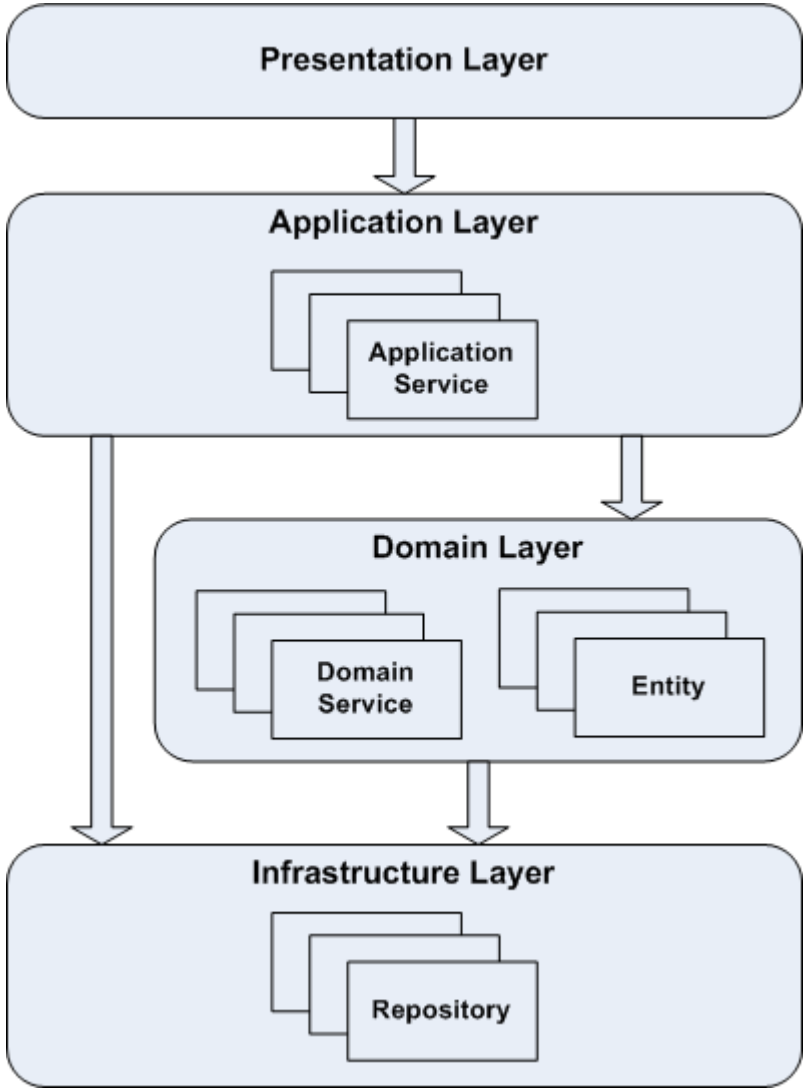


APPLICATION

A **camada de aplicação** (*Application Layer*) fornece um conjunto de serviços de aplicação (*application services*), os quais expressam as *user stories* (ou *use cases*) do software.

De modo simples, um serviço de aplicação recebe dados de seus clientes, como a interface de usuário, trata esses dados se necessário e chama um objeto do domínio para executar a operação de negócio.





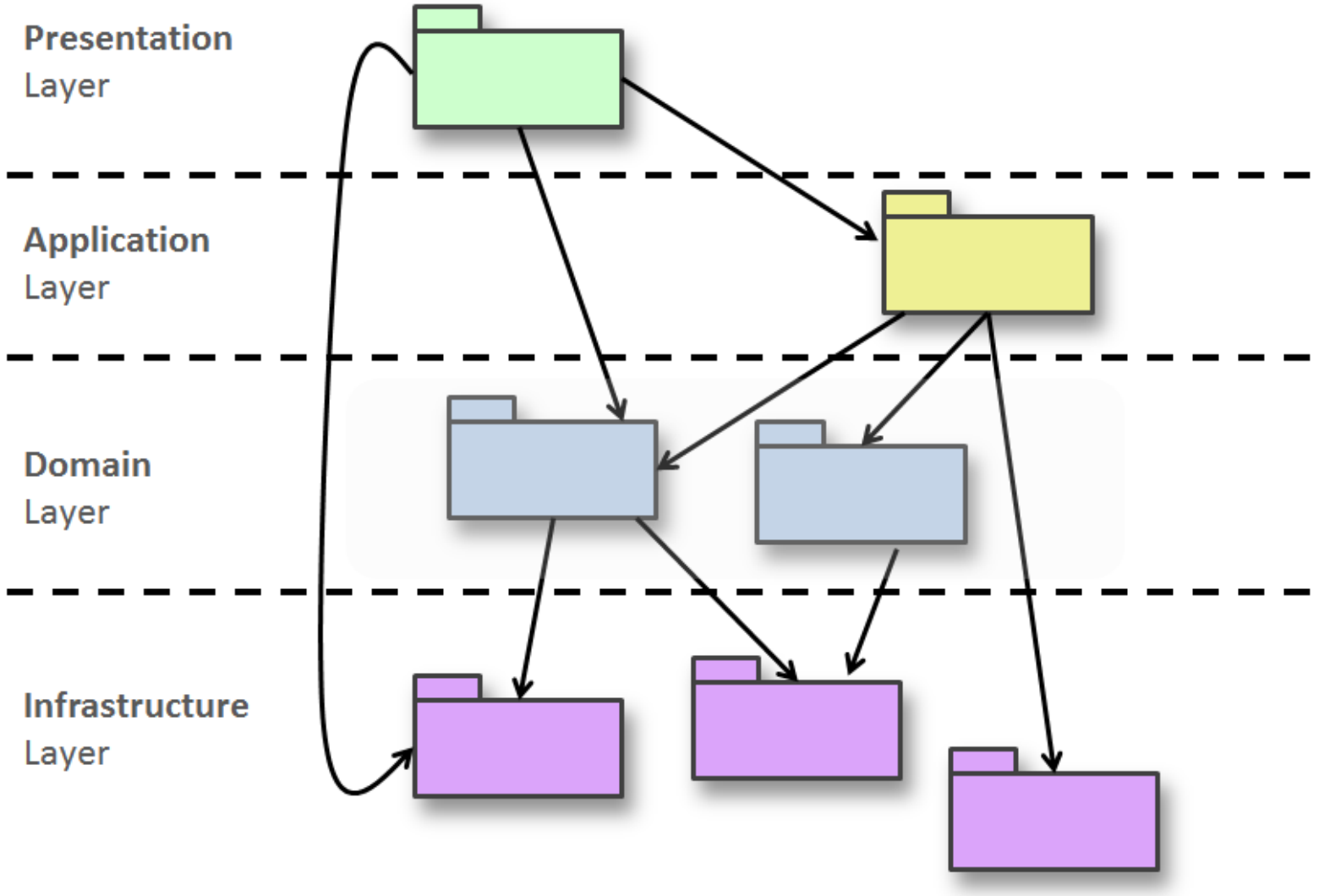
PRESENTATION

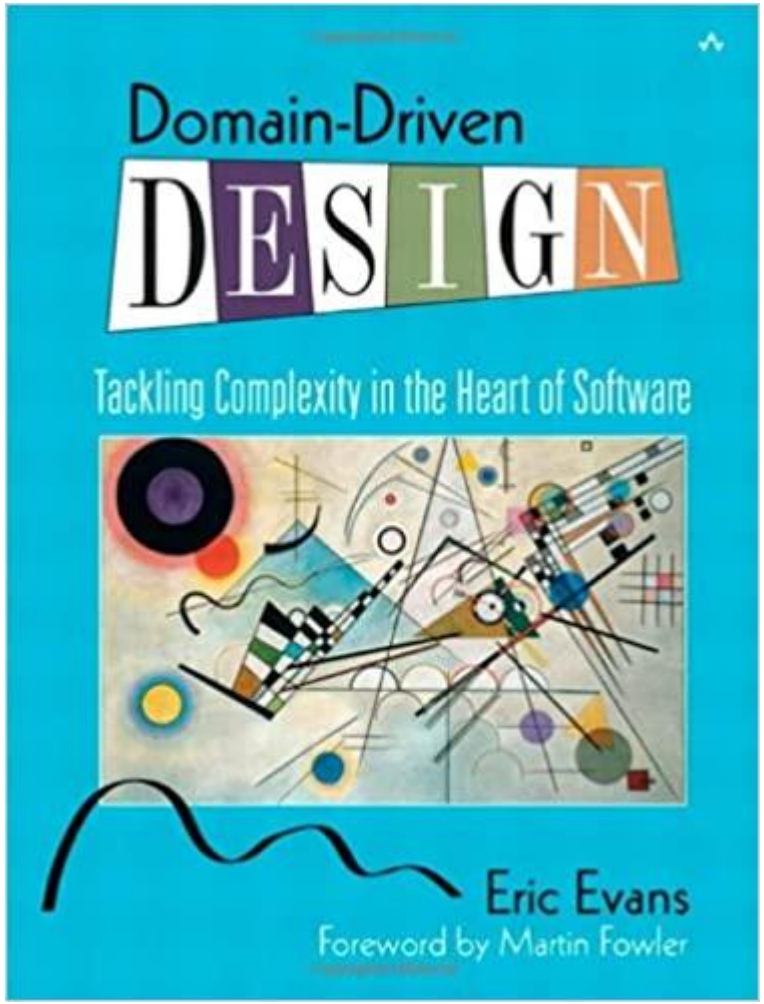
www.cotiinformatica.com.br

PRESENTATION

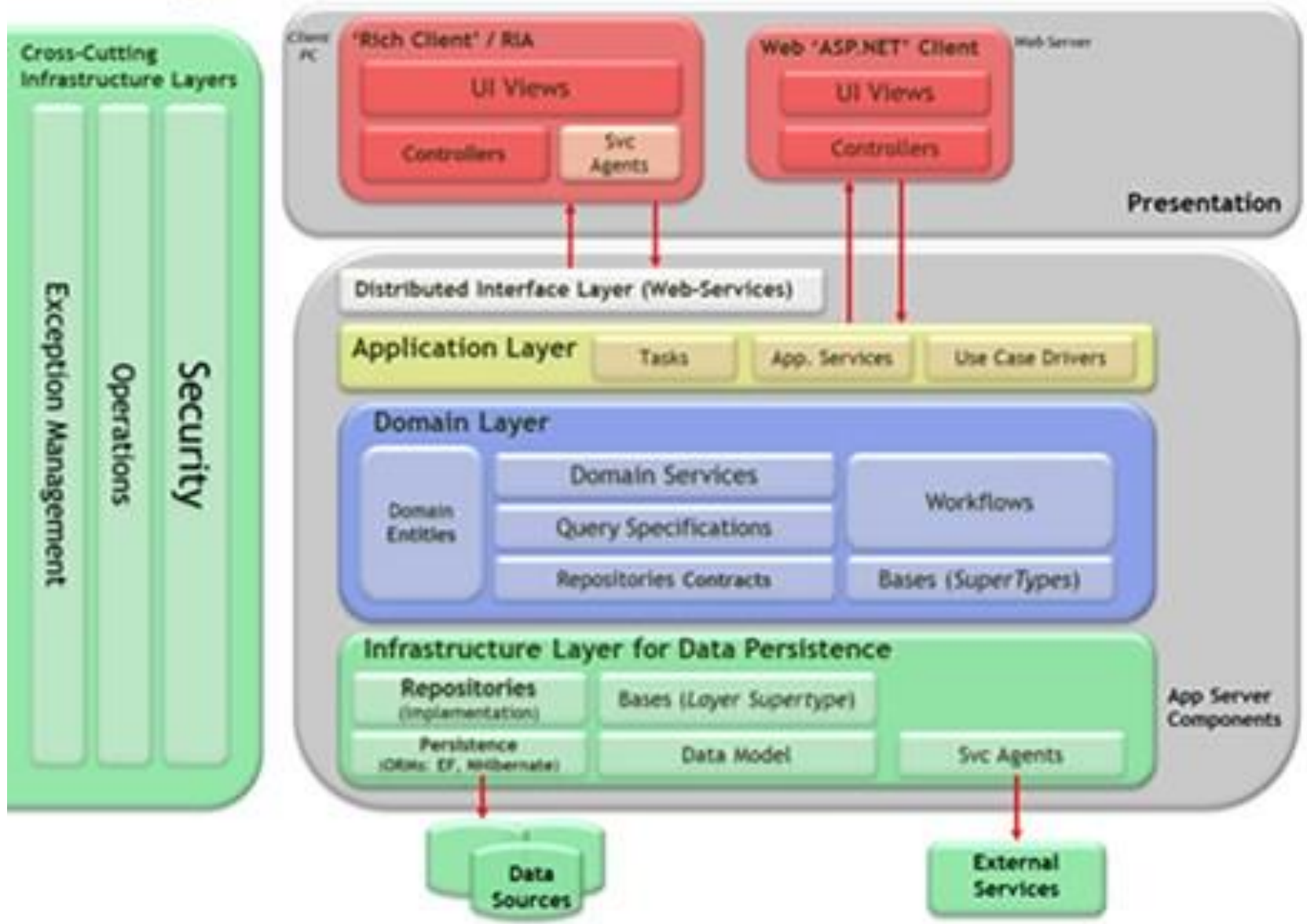
Camada onde reside o código que define o que será efetivamente apresentado ao usuário,
seja por meio de páginas web ou API de serviços.
A camada de apresentação é a responsável por apresentar as páginas de uma aplicação Web.

Os dados que devem ser apresentados na camada de apresentação são providos pela camada de modelo que envia informações para a camada de apresentação. Em resumo, o que esta camada deve definir é como estes dados serão apresentados para o usuário da aplicação.





DDD N-Layer Architecture



www.cotiinformatica.com.br