Design de código versus Arquitetura de software

**Design de código =>** disciplina responsável por definir os objetos e suas interações para resolver um problema que foi identificado e documentado durante a análise orientada a objetos.

**Arquitetura de software =>** consiste na definição dos componentes de software, suas propriedades externas, e seus relacionamentos com outros softwares.

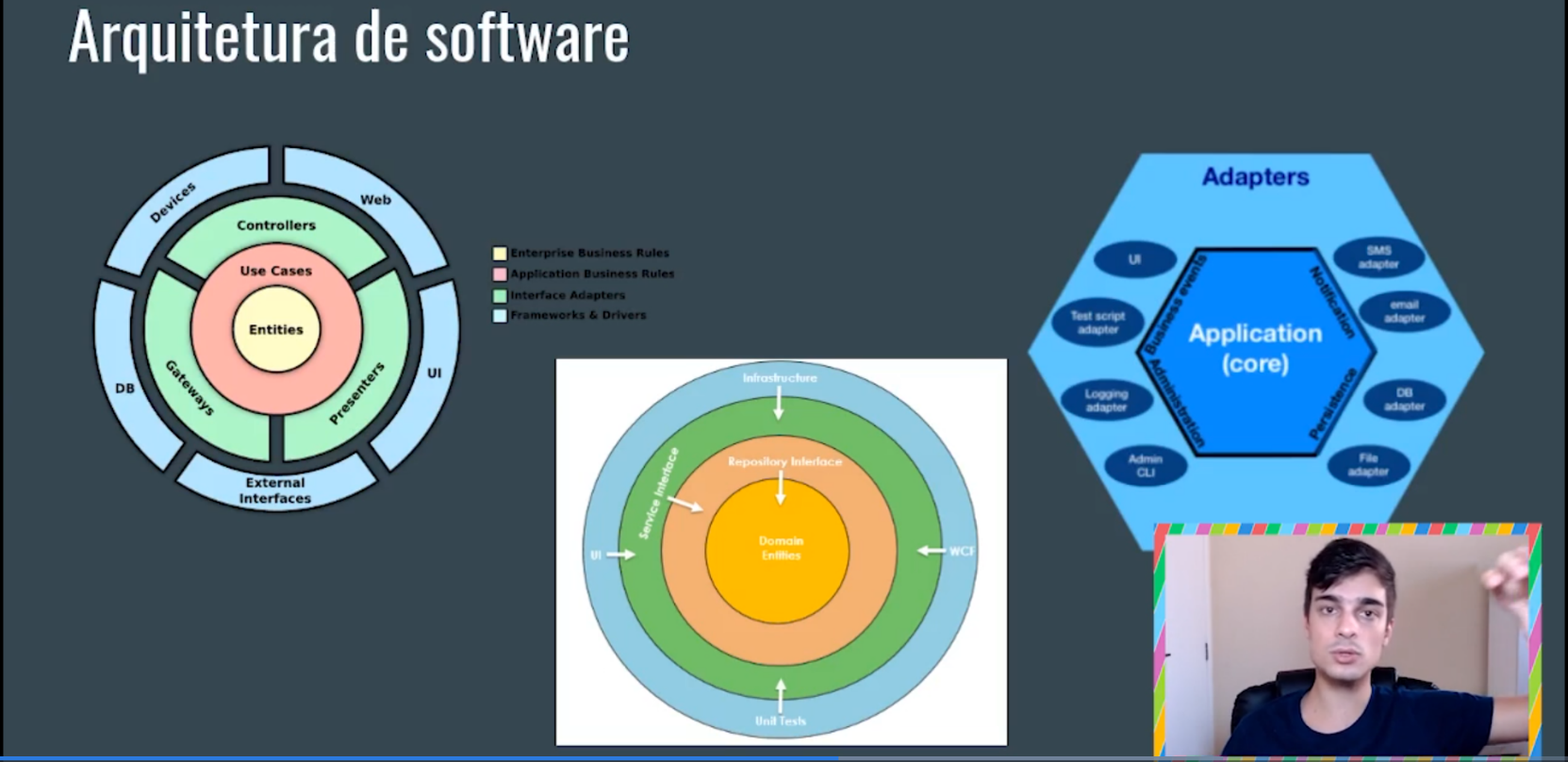
Analogia com uma casa

Um arquiteto faz a planta da casa.

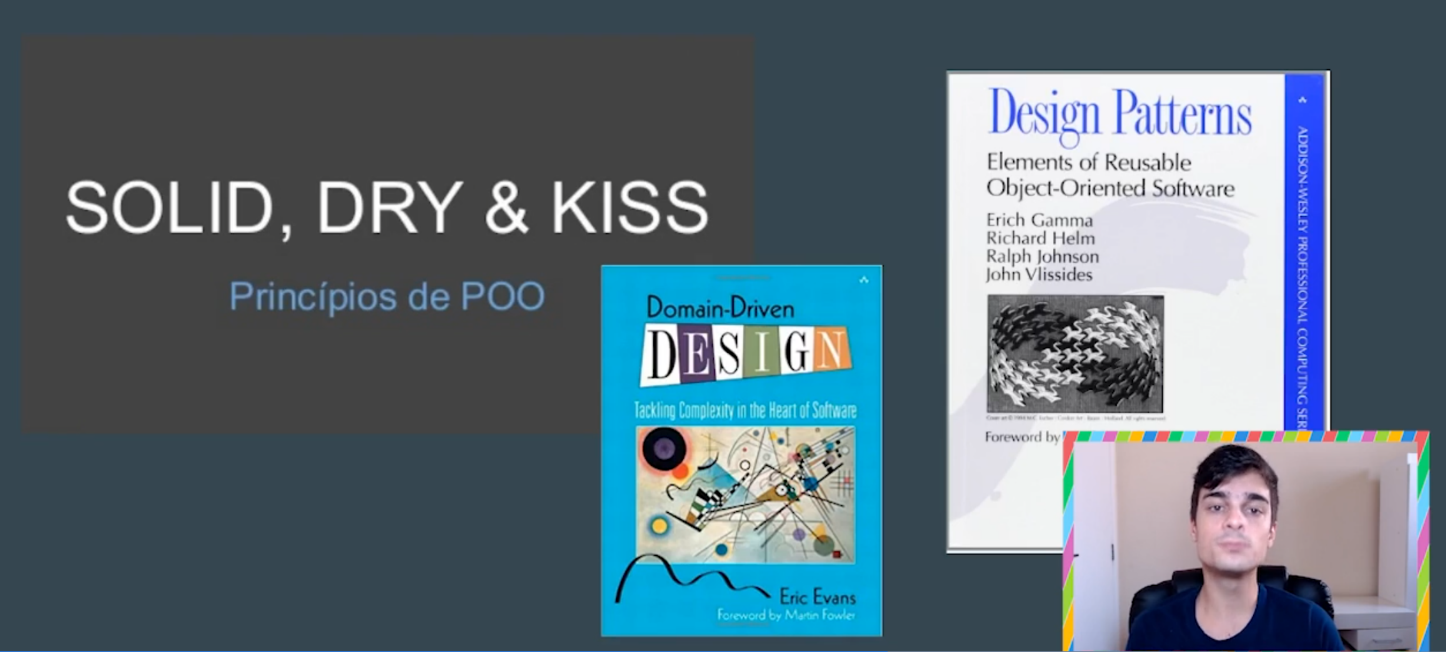
Um designer de interiores define como os moveis vão estar.

Trazendo para o nosso mundo de TI!!!

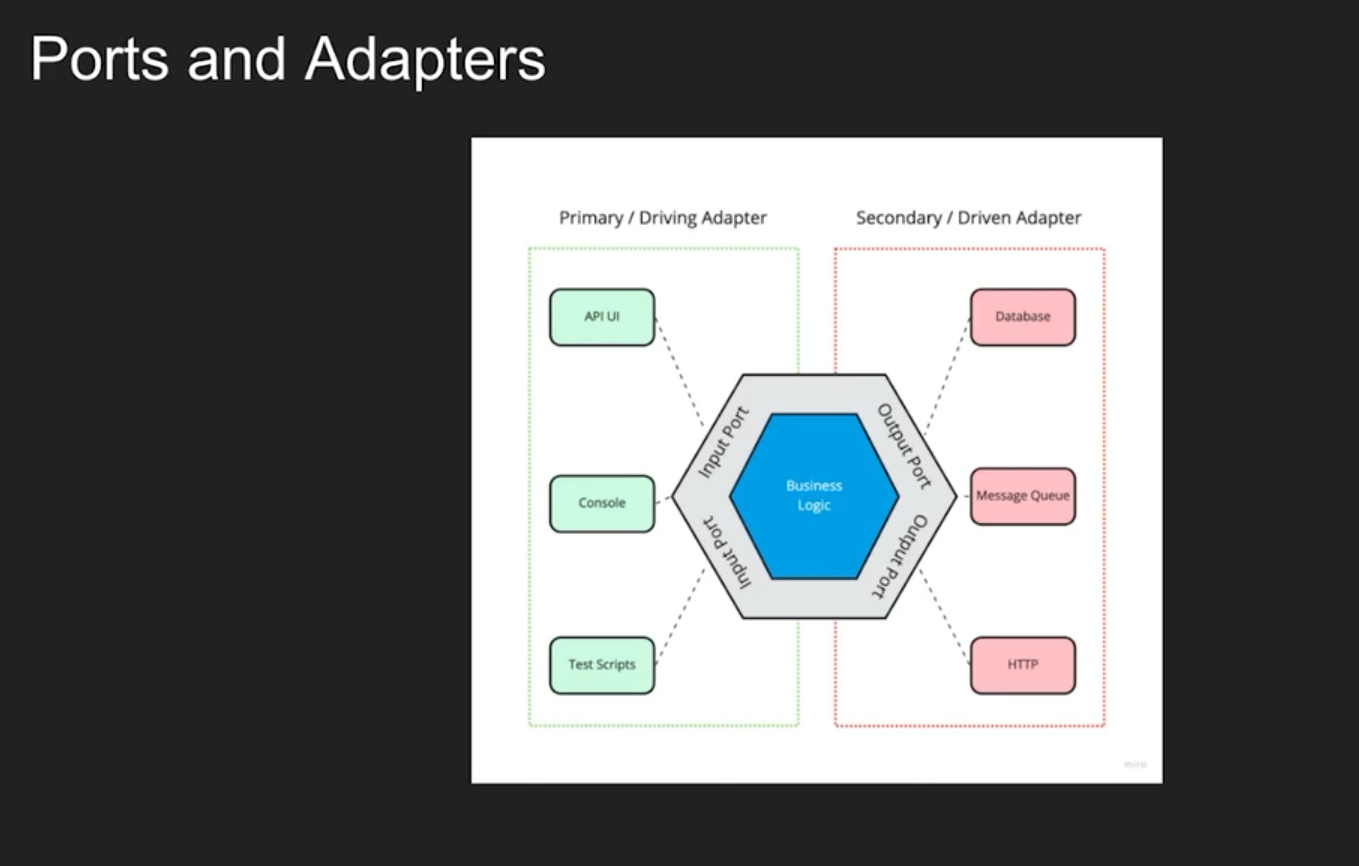
Basicamente a arquitetura é a visão de alto nível de como iremos modelar o nosso sistema



1. Exemplo de arquitetura de software

Já o design de código está ligado a maneira de como é construído (visão de baixo nível) a nossa aplicação, temos diversos tipos de exemplos de design de código como os padrões de projetos, DDD, SOLID, POO e por ai vai...

1. Design de código

ARQUITETURA HEXAGONAL

Como funciona?

A ideia é isolar o nosso domínio (regra de negócio) deixando em um local centralizado e toda comunicação do mundo externo vai chegar por alguma porta que nossa aplicação vai abrir e essa porta vai enviar os dados e algum adaptador vai ler as informações que chega nessa porta.

Teremos várias portas e vários adaptadores e teremos tratamentos diferentes para este cenário.

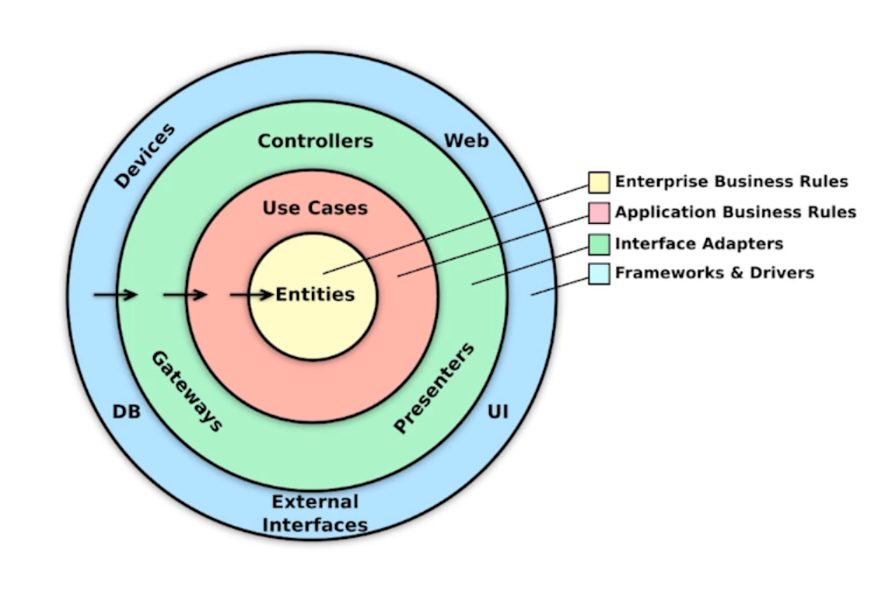
Informações úteis:

Adapter == controller.

Varias entradas que levam para o mesmo caminho (domínio).

Nosso domínio conhece apenas interfaces de comunicação ou seja quanto mais próximo da nossa porta de saída teremos o adaptadores, um que se comunica com o mongo, sql etc.. e minha regra de negócio n precisa saber qual destes adapter irão ser utilizados pois ambos irão implementar da mesma interface.

CLEAN ARCHITECTURE



Como organizar a camada Model (domínio)?

Entities => é a camada mais interna em que todas as outras podem ver porém a mais isolada pois n conversa com niguem e como o nome já fala representa uma entidade de fato DEVOLUÇÃO, PIX-OUT, TED-IN.

Use Cases (services) => basicamente é a camada que vai organizar o fluxo da regra de negócio, geralmente ele é chamado através de uma controller ou serviço de menssageria.

Interface adapters => é a camada que permite adaptar uma ferramenta externa (web, banco de dados, dados de infra) e adaptar de uma forma que o nosso use case vai entender

Frameworks & Drivers => é a camada que tem só detalhes de infraestrutura (frame web, interface com user, endpoints de uma api, banco de dados) e não temos regra nenhuma do nosso negocio

A Web o DB tem acesso as camadas mais internas porém a nossa entidade jamais deve conhecer o nosso DB pois caso eu precise mudar de sql pra potsgress a minha entidade n deve ser alterada ela não pode ficar dependente de detalhes de infra!

Fluxo:

web chama => controler que chama => user case que chama => entidades

o inverso n acontece

é claro que nem tudo é perfeito e temos algumas falhas nesse processo é ai que entra o DDD

Principais vantagens observadas no uso de Clean Architecture:

* Isolar o código de domínio da aplicação
* Inversão de dependências
* Favorece boas práticas de Orientação a Objetos
* Código de domínio com alta "testabilidade"
* Independência de frameworks e detalhes de infraestrutura