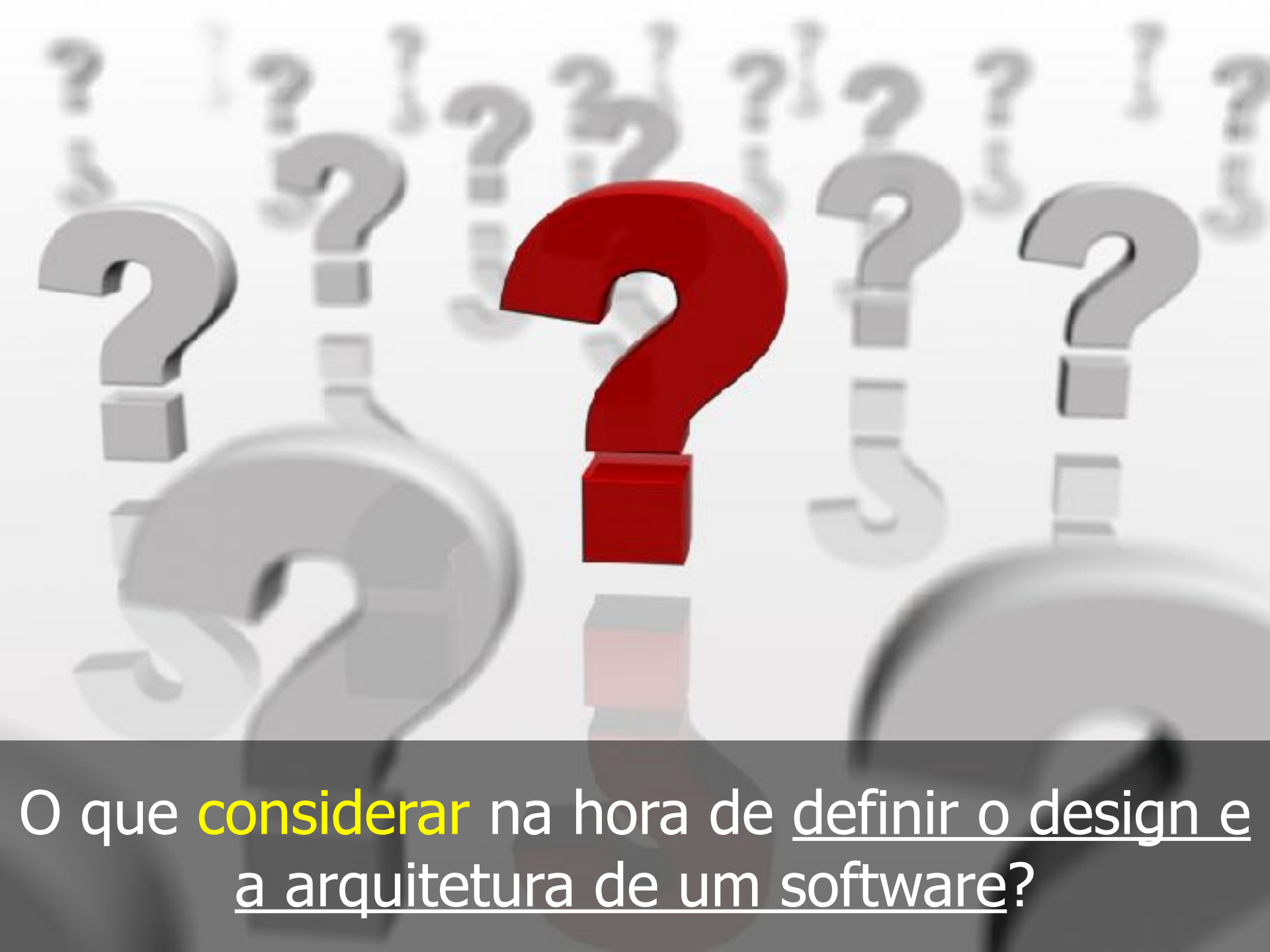

Clean Architecture

Talvez o nome Clean Design faça mais sentido que Clean Architecture

"Design is inevitable, the alternative to good design is bad design, not no design at all"

Douglas Martin



O que considerar na hora de definir o design e a arquitetura de um software?

Escopo do produto

Quem é e qual é o tamanho da equipe

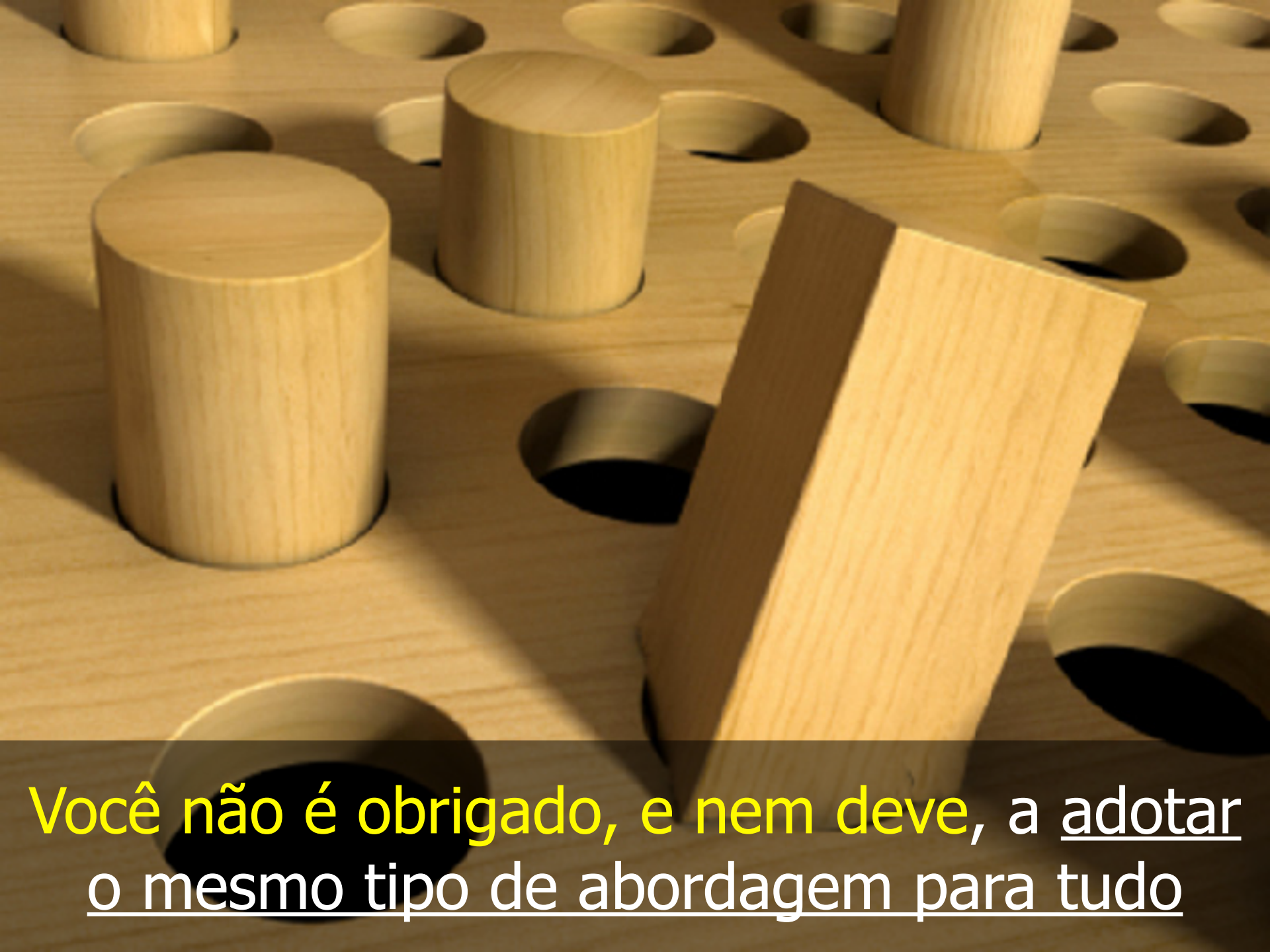

Prazo de entrega


Tipo de dispositivo


Volume de requisições

Orçamento

O design e a arquitetura de um ERP provavelmente será diferente de uma plataforma de streaming, que também terá necessidades diferentes de uma rede social

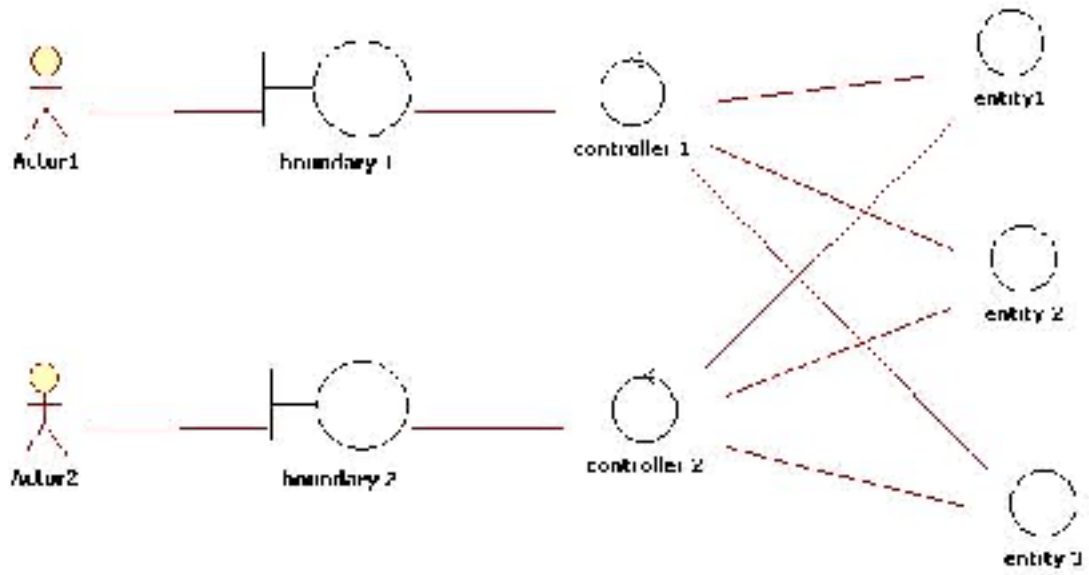


Você não é obrigado, e nem deve, a adotar o mesmo tipo de abordagem para tudo

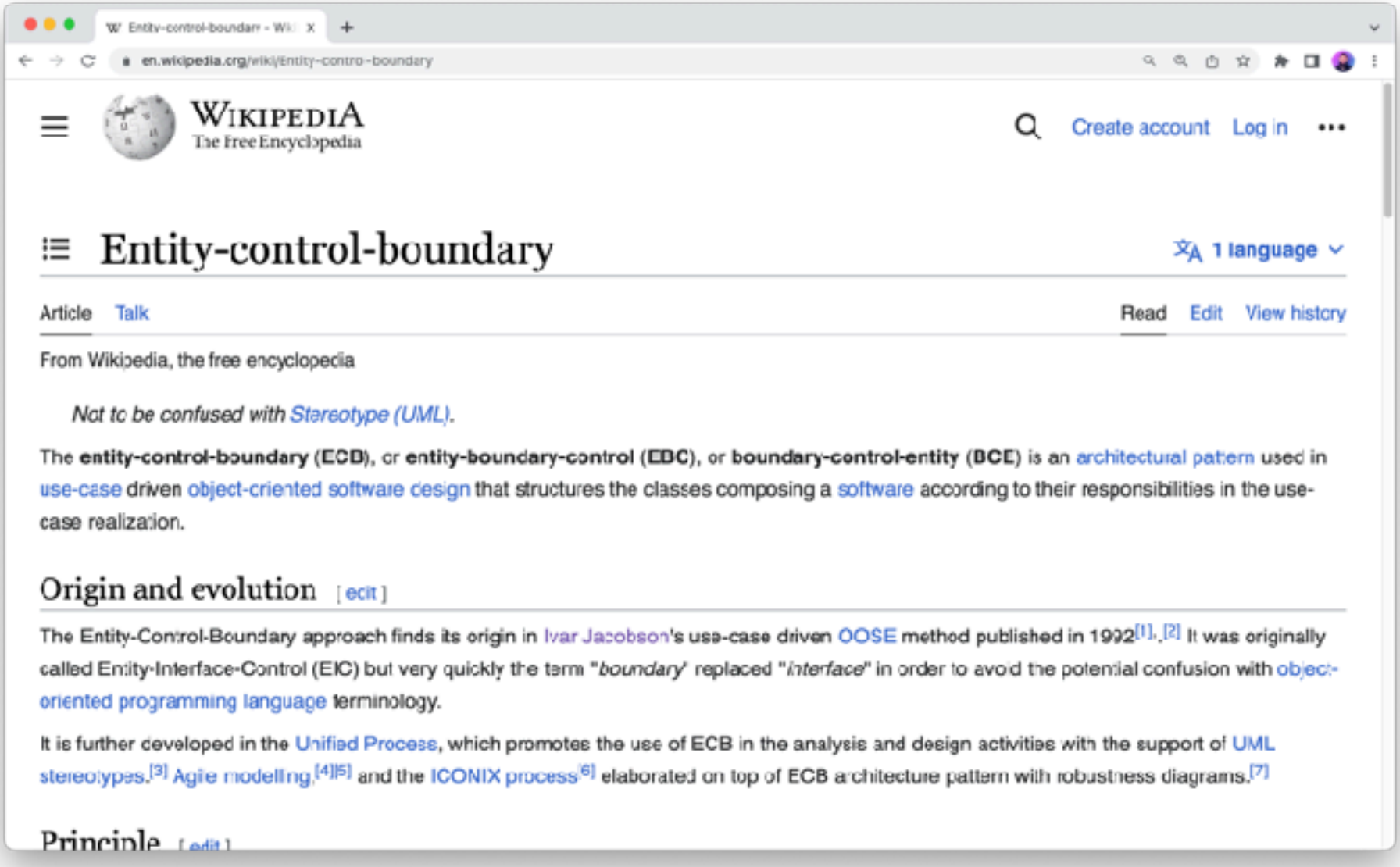
There's no one size fits all

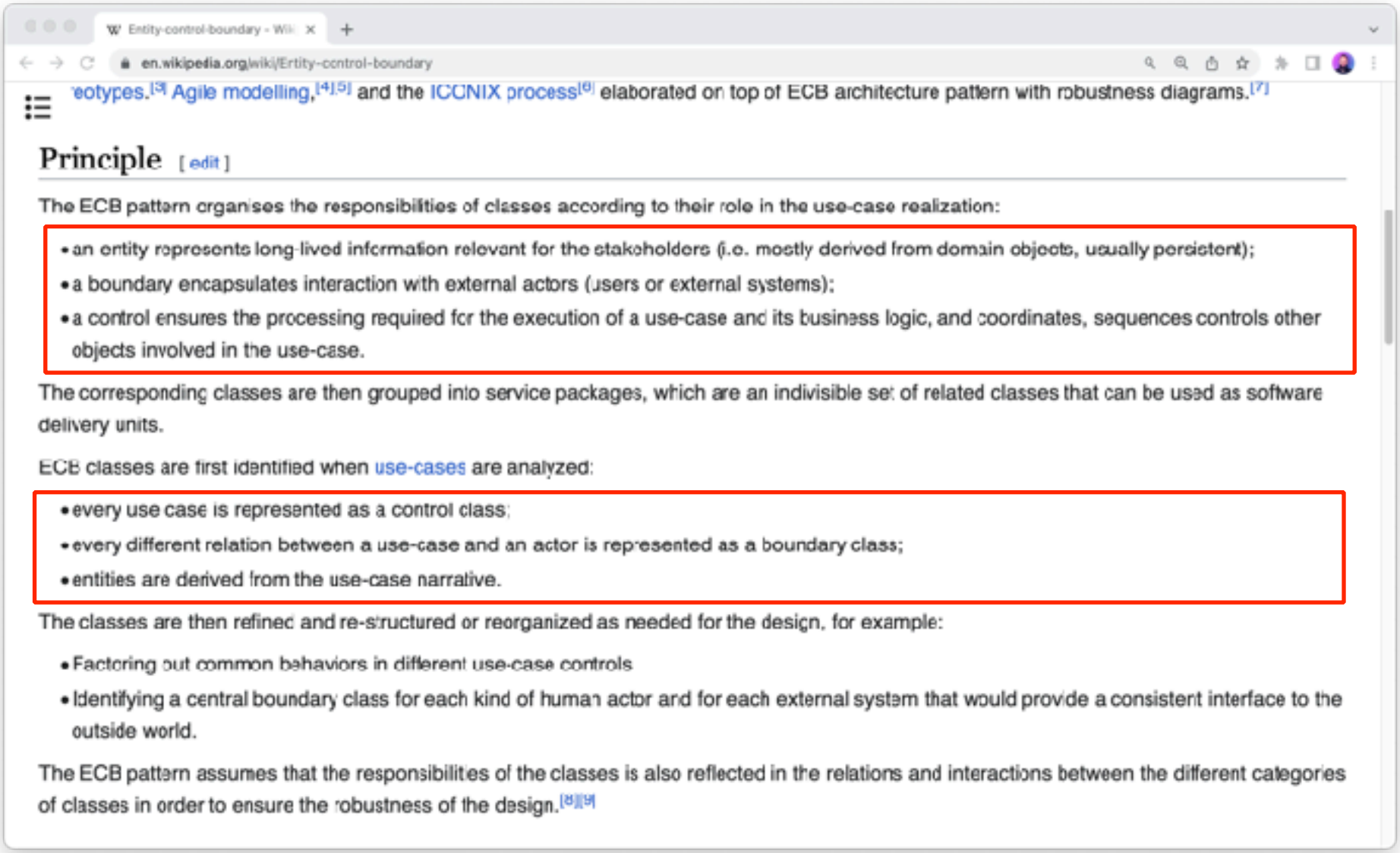
Mas isso também não é justificativa para implementar cada coisa de uma forma

Nas últimas décadas, vários modelos influenciaram o design e arquitetura...



Entity-Control-Boundary de Ivar Jacobson (1992)


Entity-control-boundary - Wi
en.wikipedia.org/wiki/Entity-contro-boundary
WIKIPEDIA
The Free Encyclopedia
Create account
Log in
Entity-control-boundary
1 language
Article
Talk
Read
Edit
View history
From Wikipedia, the free encyclopedia
Not to be confused with Stereotype (UML).
The entity-control-boundary (ECB), or entity-boundary-control (EBC), or boundary-control-entity (BCE) is an architectural pattern used in use-case driven object-oriented software design that structures the classes composing a software according to their responsibilities in the use-case realization.
Origin and evolution [edit]
The Entity-Control-Boundary approach finds its origin in Ivar Jacobson's use-case driven OOSE method published in 1992[1],[2] It was originally called Entity-Interface-Control (EIC) but very quickly the term "boundary" replaced "interface" in order to avoid the potential confusion with object-oriented programming language terminology.
It is further developed in the Unified Process, which promotes the use of ECB in the analysis and design activities with the support of UML stereotypes.[3] Agile modelling,[4][5] and the ICONIX process[6] elaborated on top of ECB architecture pattern with robustness diagrams.[7]
Principle [edit]


Entity-control-boundary - Wik
en.wikipedia.org/wiki/Entity-control-boundary
eotypes.[3] Agile modelling,[4][5] and the ICONIX process[6] elaborated on top of ECB architecture pattern with robustness diagrams.[7]
Principle [edit]
The ECB pattern organises the responsibilities of classes according to their role in the use-case realization:
• an entity represents long-lived information relevant for the stakeholders (i.e. mostly derived from domain objects, usually persistent);
• a boundary encapsulates interaction with external actors (users or external systems);
• a control ensures the processing required for the execution of a use-case and its business logic, and coordinates, sequences controls other objects involved in the use-case.
The corresponding classes are then grouped into service packages, which are an indivisible set of related classes that can be used as software delivery units.
ECB classes are first identified when use-cases are analyzed:
• every use case is represented as a control class;
• every different relation between a use-case and an actor is represented as a boundary class;
• entities are derived from the use-case narrative.
The classes are then refined and re-structured or reorganized as needed for the design, for example:
• Factoring out common behaviors in different use-case controls
• Identifying a central boundary class for each kind of human actor and for each external system that would provide a consistent interface to the outside world.
The ECB pattern assumes that the responsibilities of the classes is also reflected in the relations and interactions between the different categories of classes in order to ensure the robustness of the design.[8][9]

Entity

An entity is a long-lived, passive element that is responsible for some meaningful chunk of information. This is not to say that entities are "data," while other design elements are "function." **Entities perform behavior organized around some cohesive amount of data**.

An example of an entity for a customer service application is a Customer entity that manages all information about a customer. **A design element for this entity would include data about the customer, behavior to manage the data, behavior to validate customer information and to perform other business calculations, such as "Is this customer allowed to purchase that product?**"

The identification of the entities as part of this pattern can be done many times at different levels of abstraction from the code, at different levels of granularity in size, and from the perspectives of different contexts.

Control

A control element manages the flow of interaction of the scenario. A control element could manage the end-to-end behavior of a scenario. or it could manage the interactions between a subset of the elements. Behavior and business rules relating to the information relevant to the scenario should be assigned to the entities; **the control elements are responsible only for the flow of the scenario**.

CreateMarketingCapmpaign is an example of a control element for a customer service application. **This design element would be responsive to certain frontend boundary elements and would collaborate with other entities, control elements, and backend boundary elements to support the creation of a marketing campaign**.

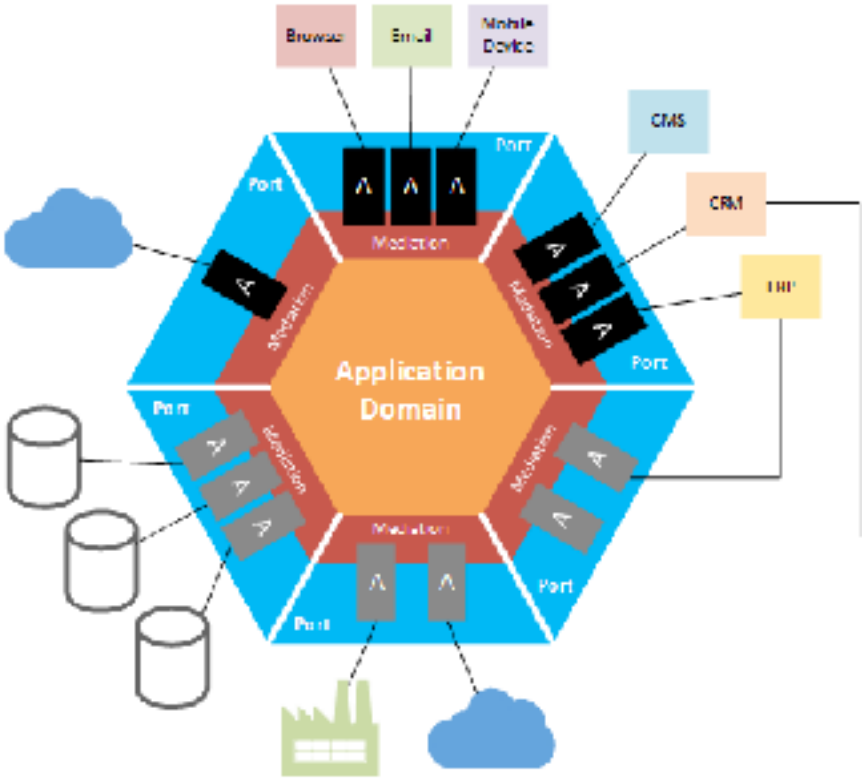
Boundary

A boundary element lies on the periphery of a system or subsystem and will be the "front end" that accept input from outside of the area under design, and other elements will be "back end," **managing communication to supporting elements outside of the system or subsystem**.

Two examples of boundary elements for a customer service application might be a front end MarketingCampaignForm and a back end BugdetSystem element. The MarketingCampaignForm would manage the exchange of information between a user and the system, and the BugdetSystem would manage the exchange of information between the system and an external system that manages budgets.

"Beginners may sometime only use entity object as data carriers and place all dynamic behaviour in control objects. This should, however be avoided. Instead, quite a lot of behaviour should be placed in the entity objects"

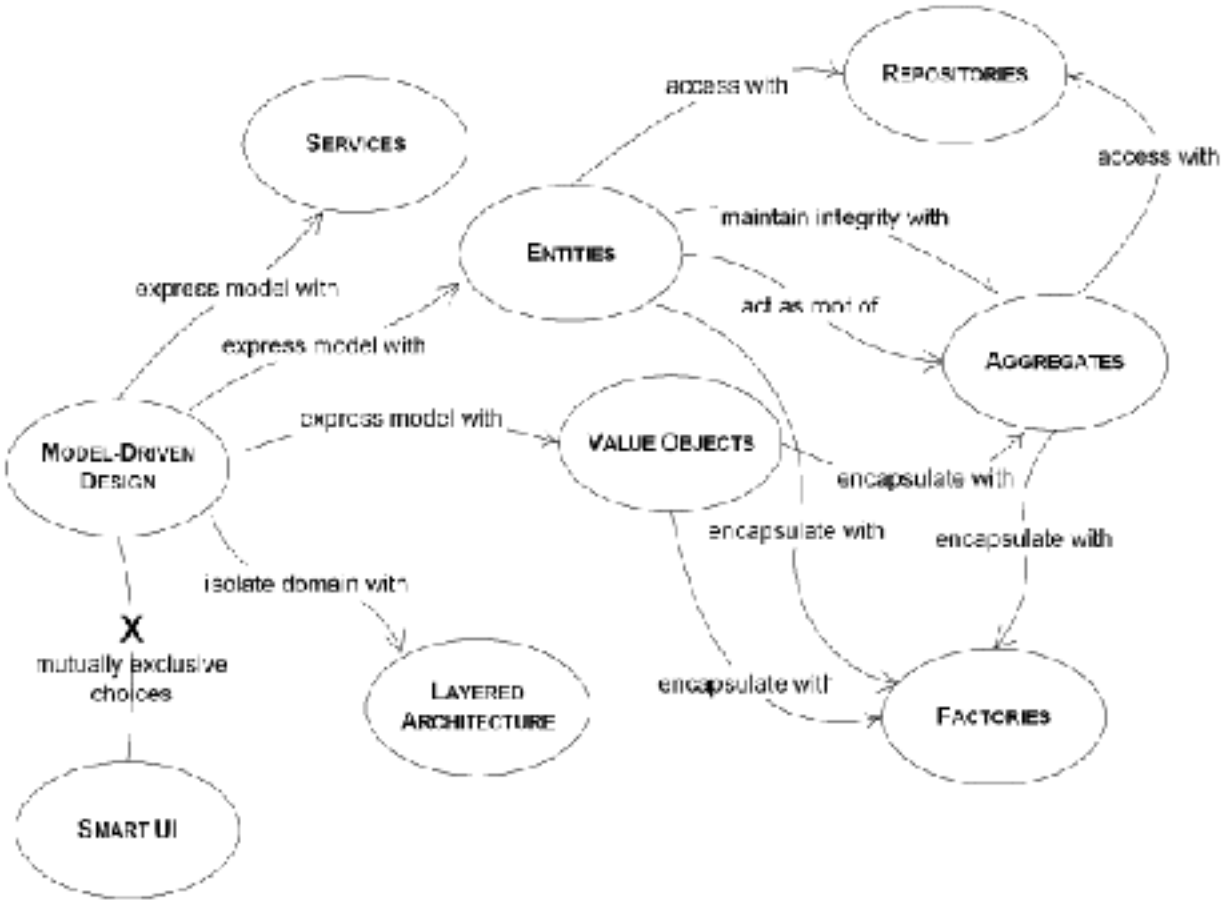
Ivar Jacobson



Hexagonal Architecture ou Ports and Adapters, Alistair Cockburn (2005)

"Allow an application to equally be driven by users, programs, automated test or batch scripts, and to be developed and tested in isolation from its eventual run-time devices and databases"

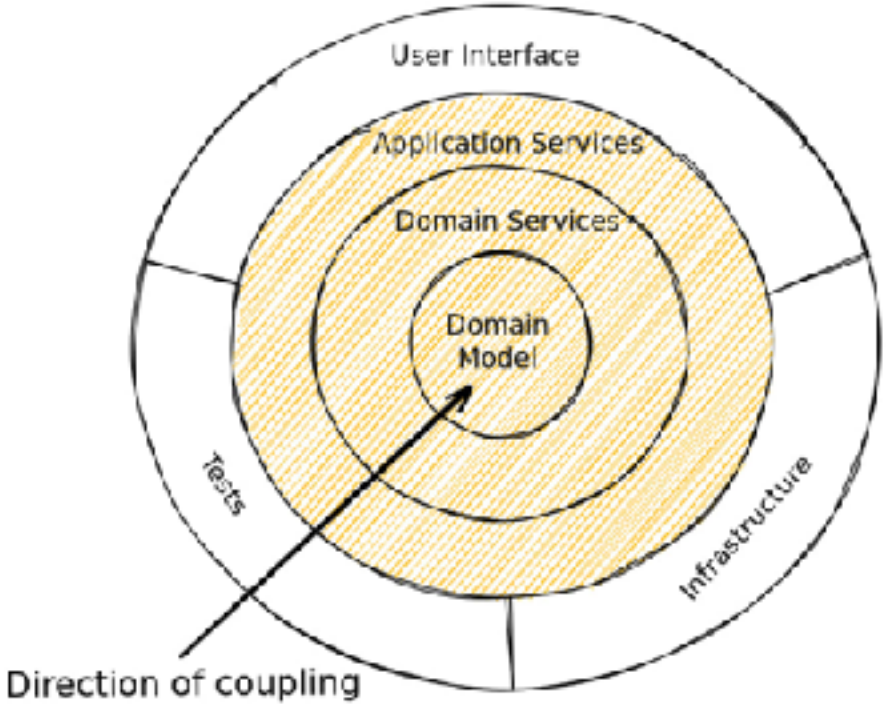
Alistair Cockburn



Domain-Driven Design, Eric Evans (2003)

"The heart of software is its ability to solve domain-related problems for its user"

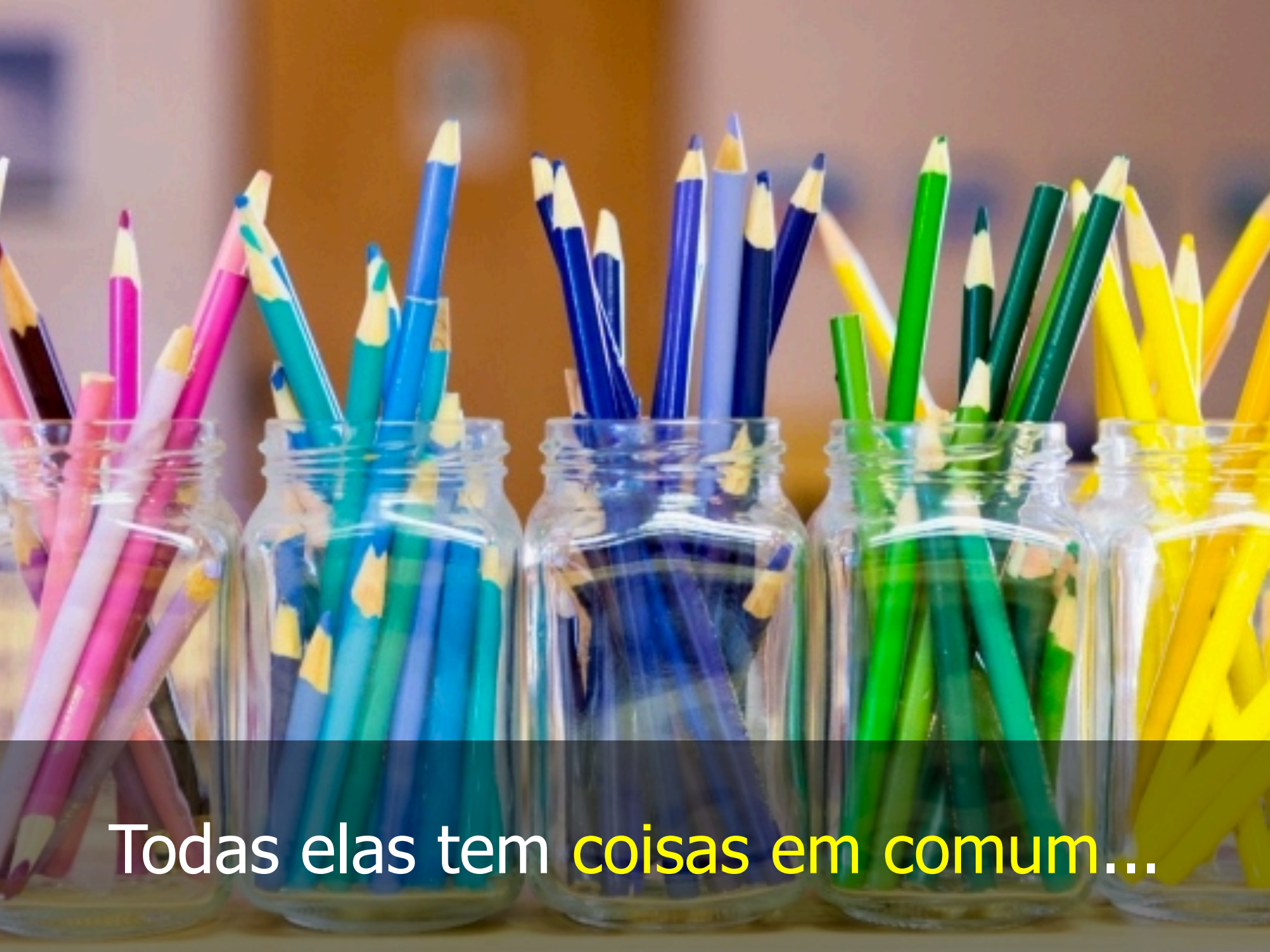
Eric Evans



Onion Architecture, Jeff Palermo (2008)

"The biggest offender (and most common) is the coupling of UI and business logic to data access"

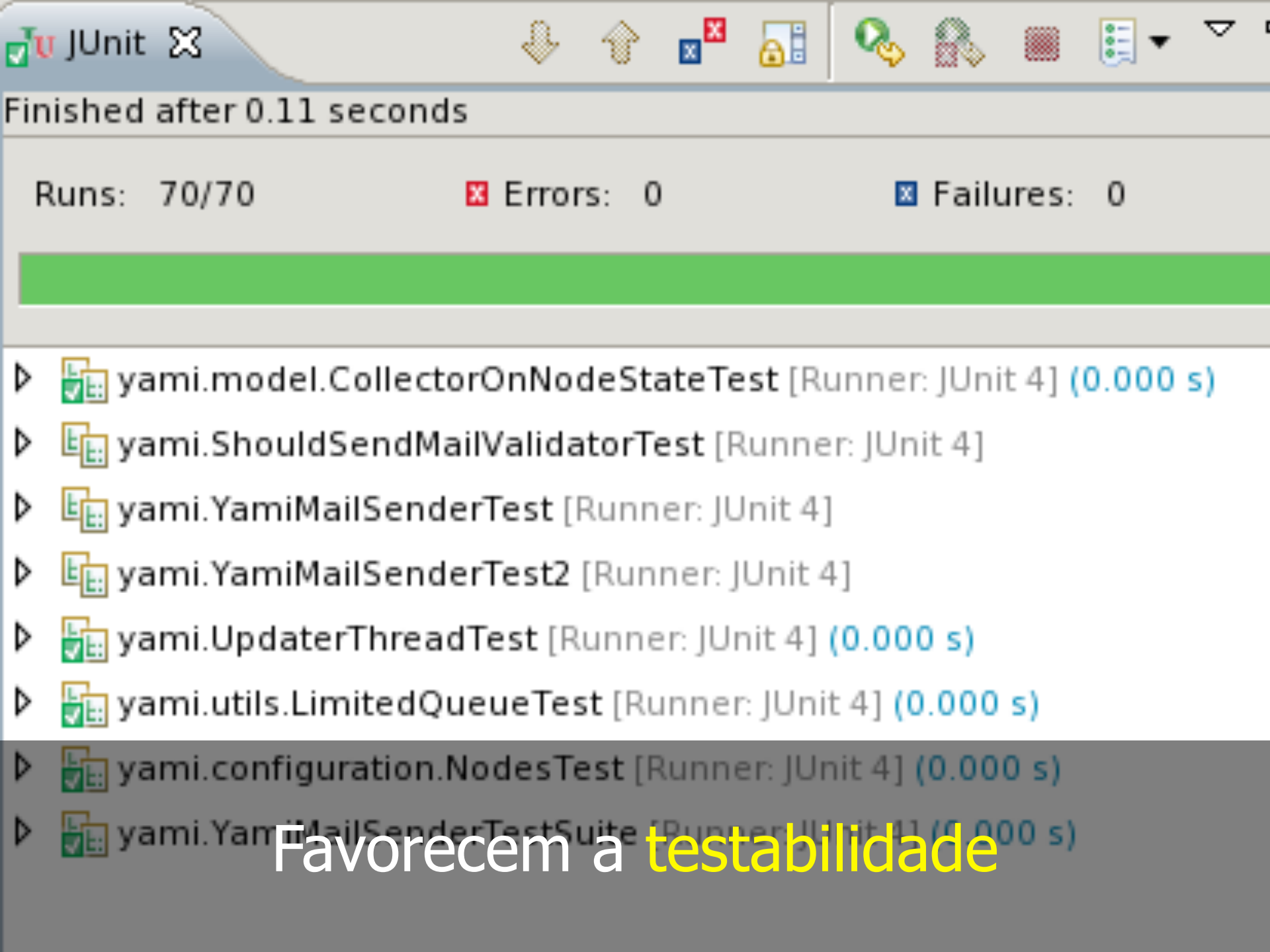
Jeff Palermo


Todas elas tem coisas em comum...

Isolam as regras de negócio

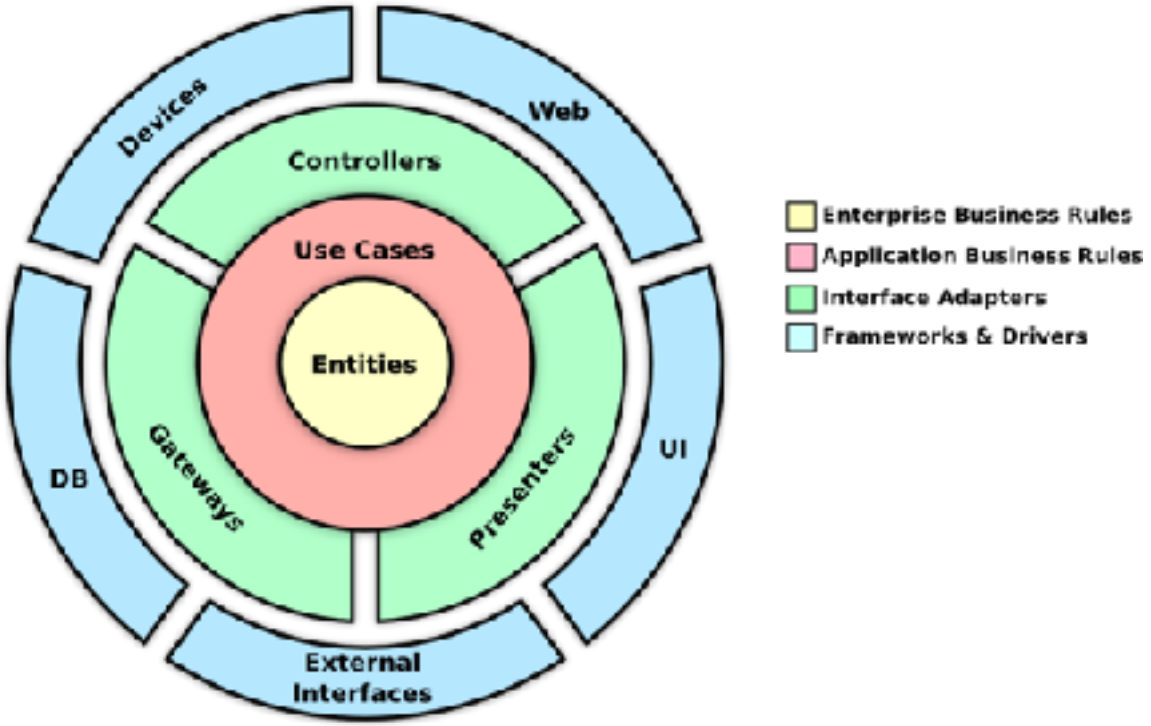

Definem camadas e suas responsabilidades

Criam um fluxo de controle e dependência ordenado e direcional


JUnit
Finished after 0.11 seconds
Runs: 70/70
Errors: 0
Failures: 0
yami.model.CollectorOnNodeStateTest [Runner: JUnit 4] (0.000 s)
yami.ShouldSendMailValidatorTest [Runner: JUnit 4]
yami.YamiMailSenderTest [Runner: JUnit 4]
yami.YamiMailSenderTest2 [Runner: JUnit 4]
yami.UpdaterThreadTest [Runner: JUnit 4] (0.000 s)
yami.utils.LimitedQueueTest [Runner: JUnit 4] (0.000 s)
yami.configuration.NodesTest [Runner: JUnit 4] (0.000 s)
Favorecem a testabilidade

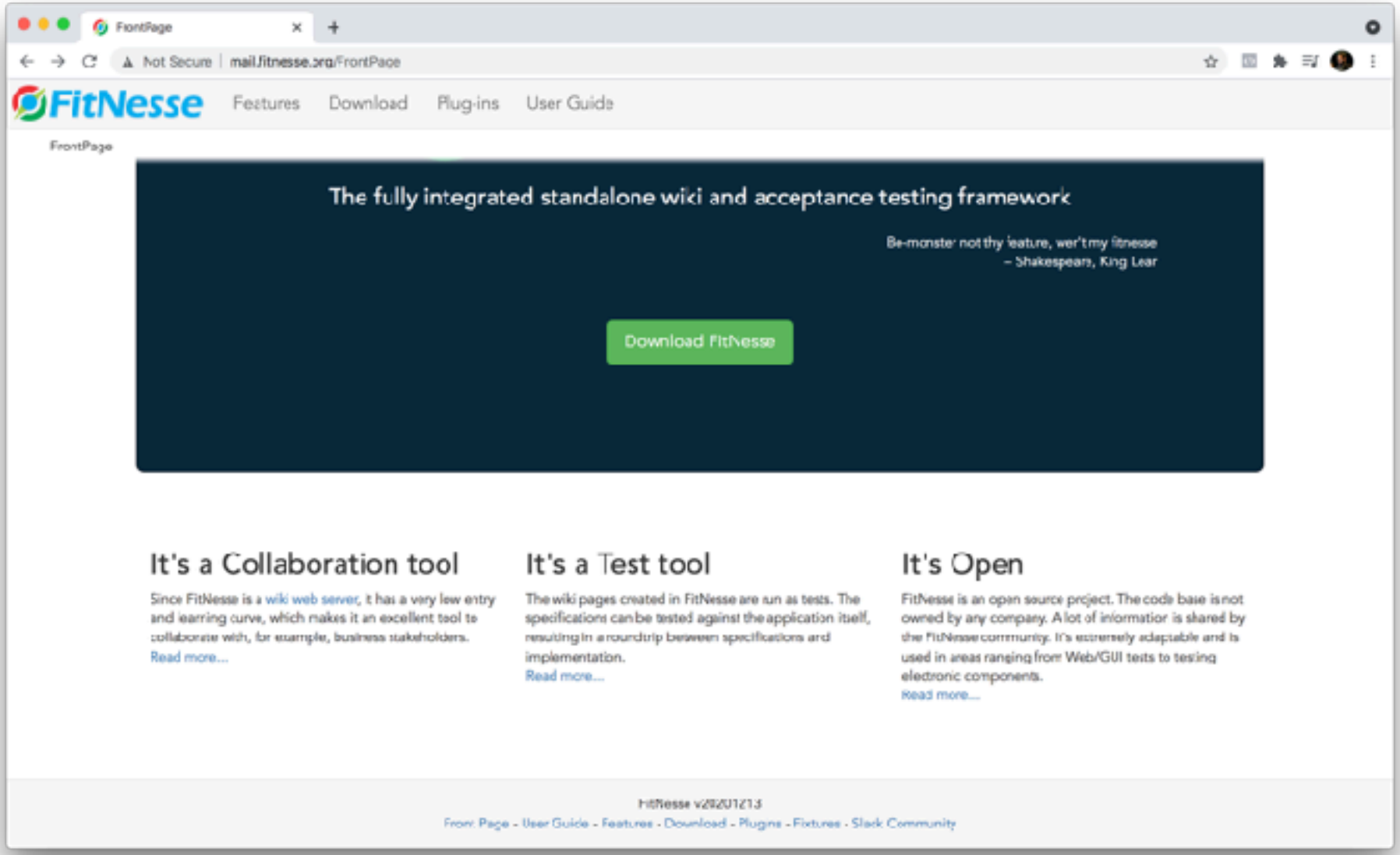
São independentes de recursos externos

Favorecem a evolução tecnológica



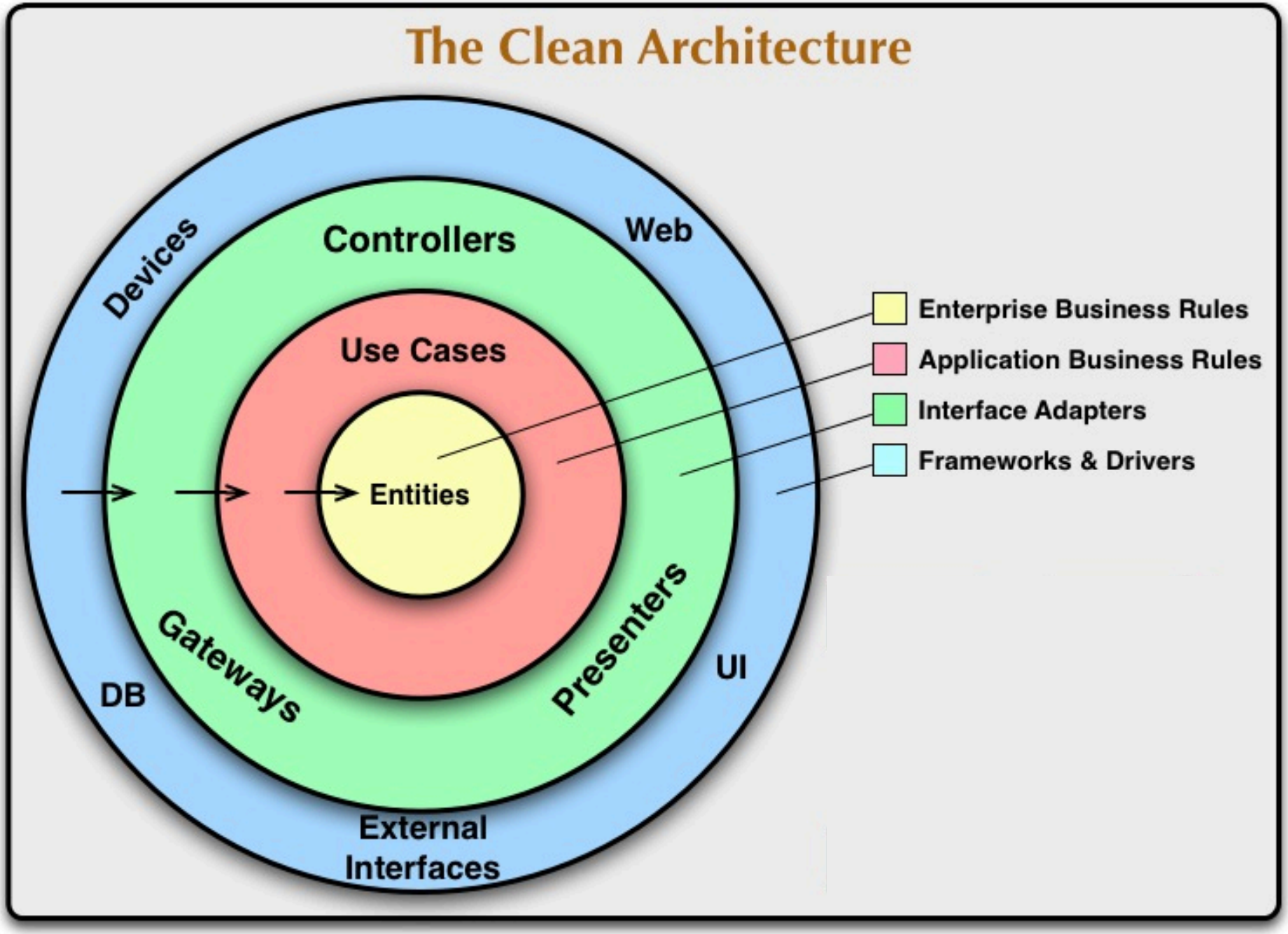
Clean Architecture, Robert Martin (2012)

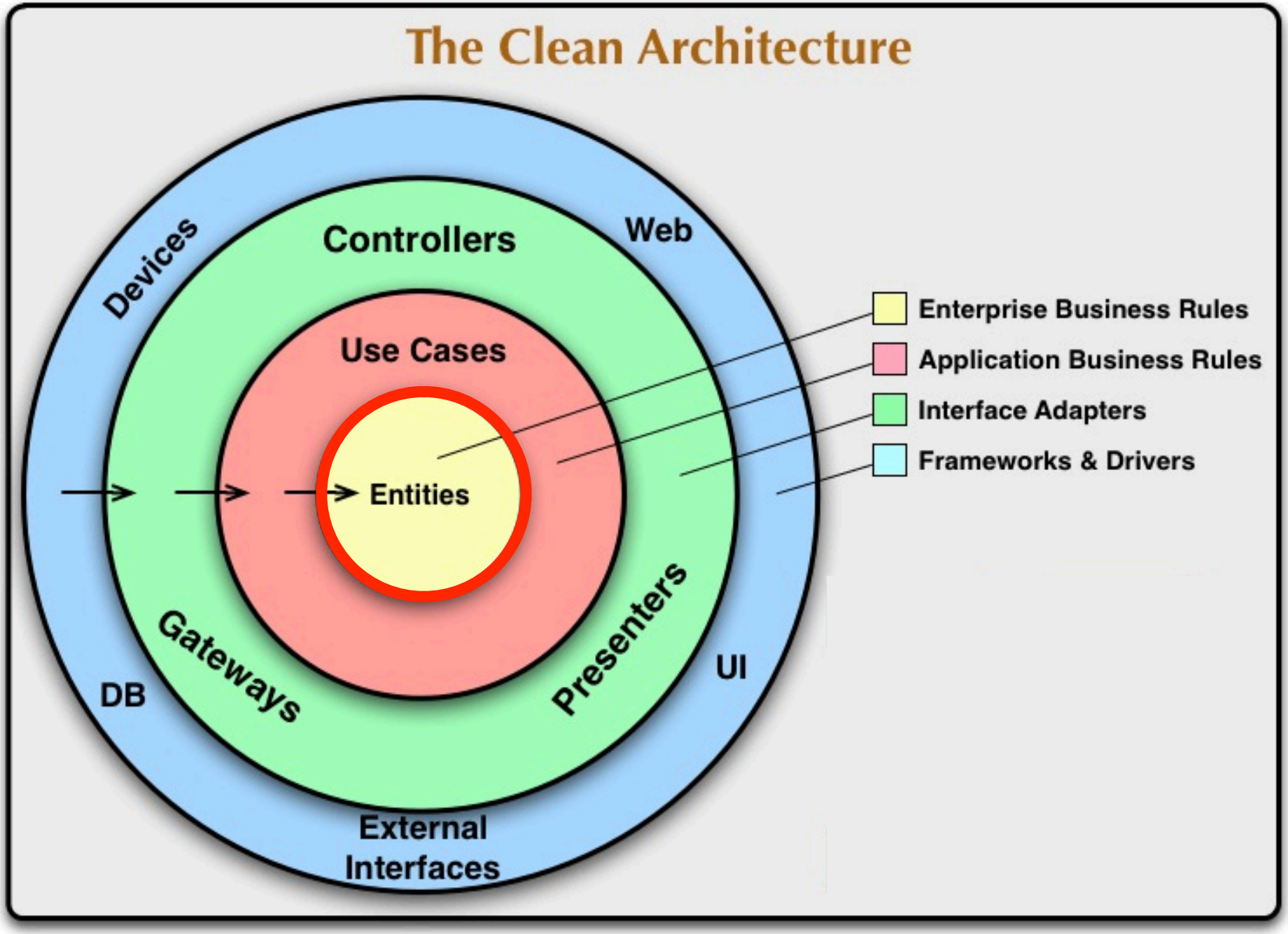
A Clean Architecture é um modelo que tem como objetivo o desacoplamento entre as regras de negócio, ou domínio, da aplicação e os recursos externos como frameworks e banco de dados


FrontPage
Not Secure | mail.fitnesse.org/FrontPage
FitNesse
Features
Download
Plug-ins
User Guide
FrontPage
The fully integrated standalone wiki and acceptance testing framework
Be-monster not thy feature, wer't my fitnesse
– Shakespears, King Lear
Download FitNesse
It's a Collaboration tool
Since FitNesse is a wiki web server, it has a very low entry and learning curve, which makes it an excellent tool to collaborate with, for example, business stakeholders.
Read more...
It's a Test tool
The wiki pages created in FitNesse are run as tests. The specifications can be tested against the application itself, resulting in a roundtrip between specifications and implementation.
Read more...
It's Open
FitNesse is an open source project. The code base is not owned by any company. A lot of information is shared by the FitNesse community. It's extremely adaptable and is used in areas ranging from Web/GUI tests to testing electronic components.
Read more...
FitNesse v20201213
Front Page - User Guide - Features - Download - Plugins - Fixtures - Slack Community

"The center of your application is not the database, nor is it one or more of the frameworks you may be using. The center of your application is the use cases of your application"

Robert Martin


The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers


The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers

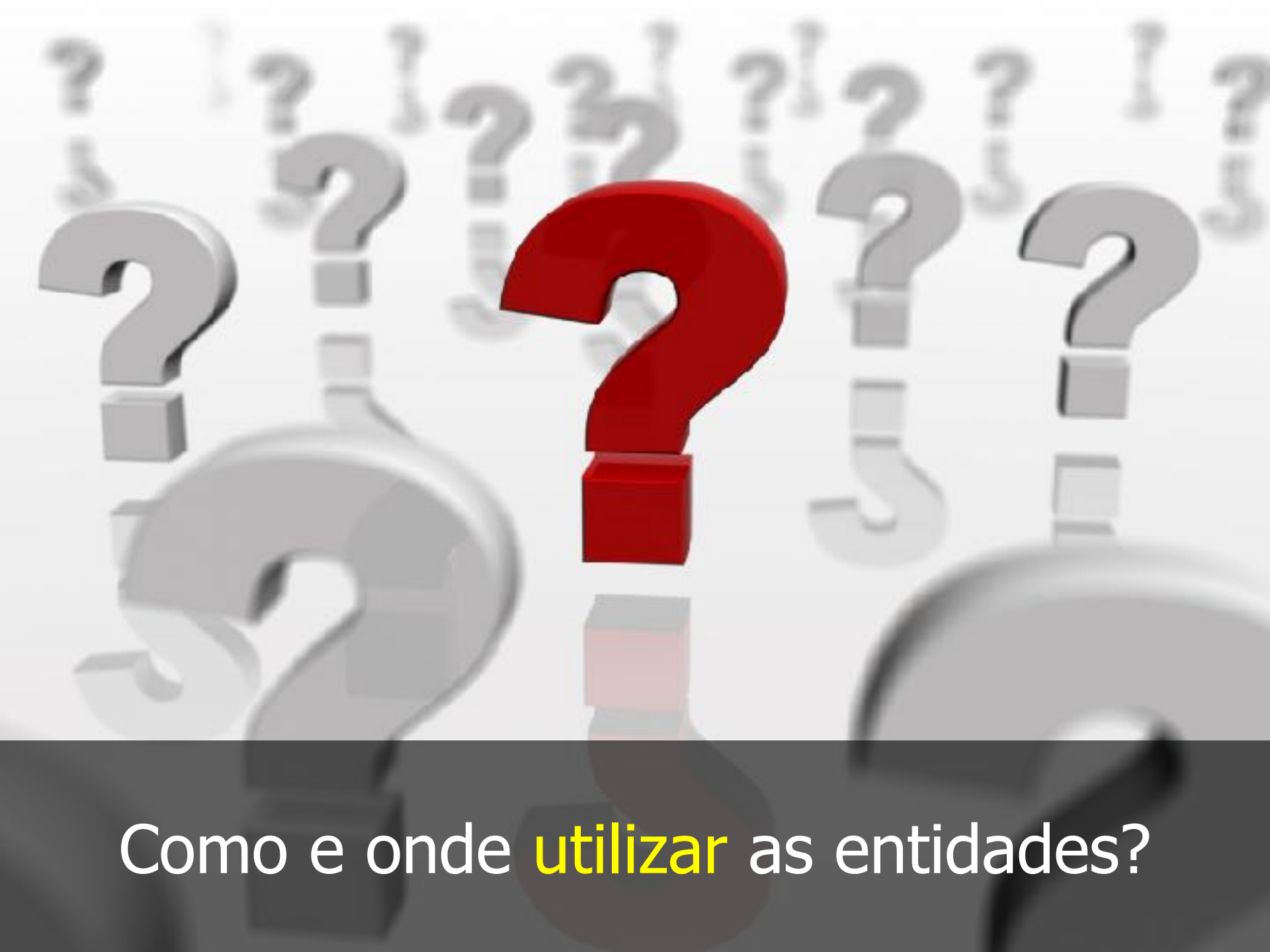
Entidades são responsáveis por modelar as regras de negócio independentes, aplicadas em qualquer contexto e que podem ser desde um objeto com métodos até mesmo um conjunto de funções

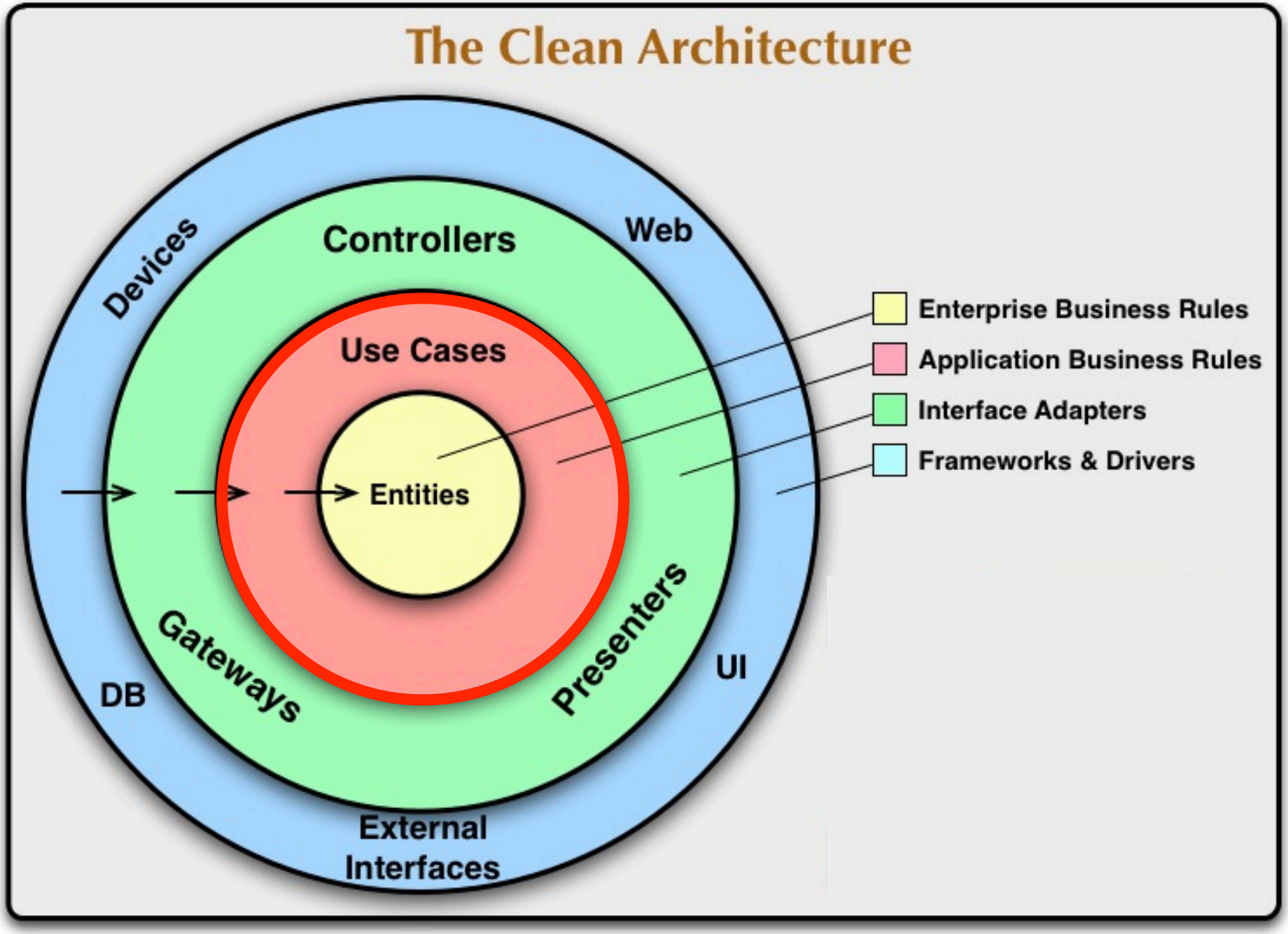
Regras de negócio independentes

- O CPF do cliente é válido?
- Qual é o valor total do pedido, considerando os itens e a quantidade de cada um?
- Quanto desconto deve ser aplicado ao pedido?
- O cupom de desconto é válido?
- Qual é o volume de um item?
- Qual é a densidade de um item?
- Quanto é o frete de um item?

Essas entidades não são as mesmas que utilizamos em um ORM


O problema do domínio anêmico


Como e onde utilizar as entidades?

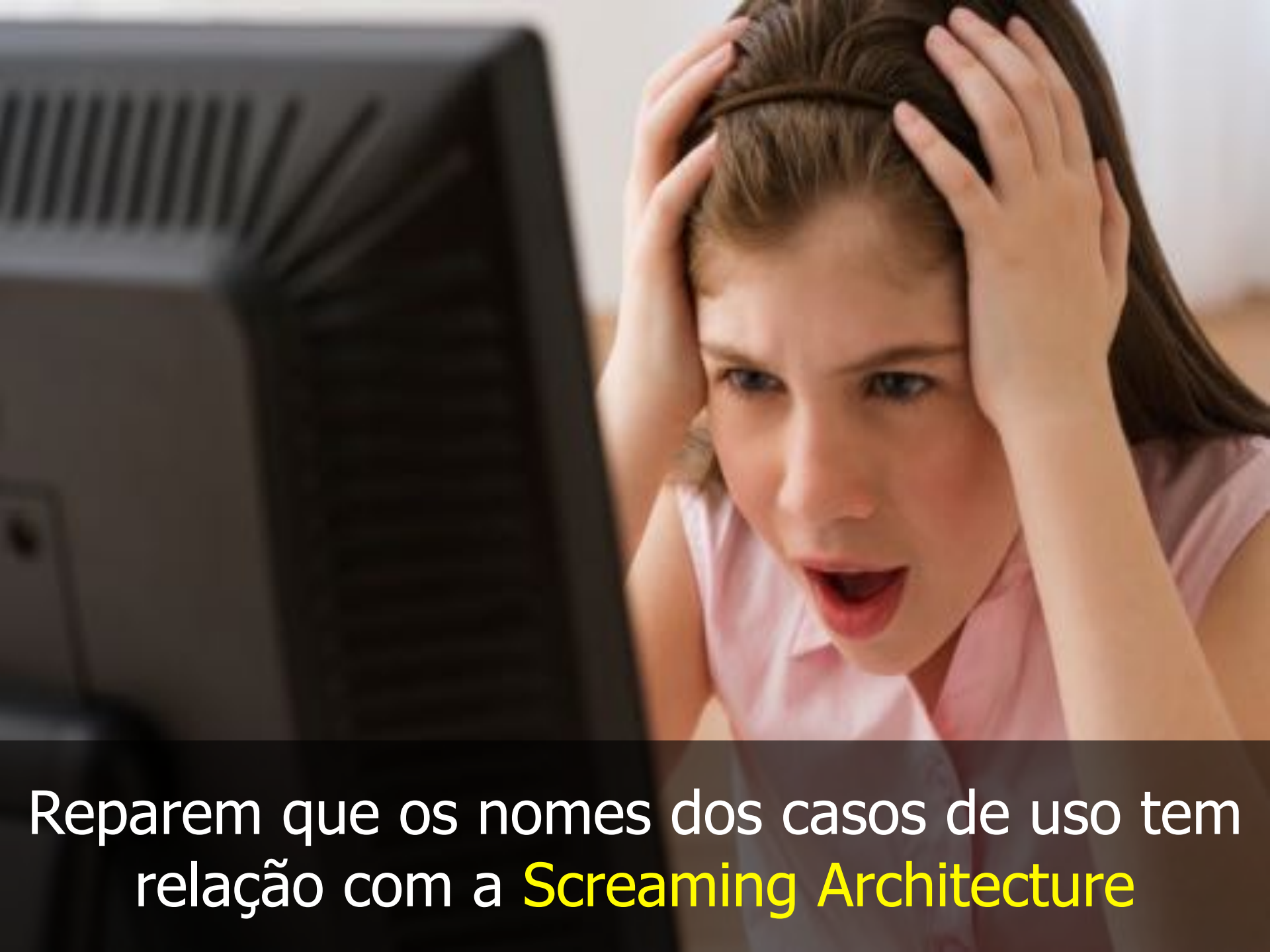

The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers

Na Clean Architecture, os casos de uso contém a aplicação das regras de negócio independente em um contexto específico

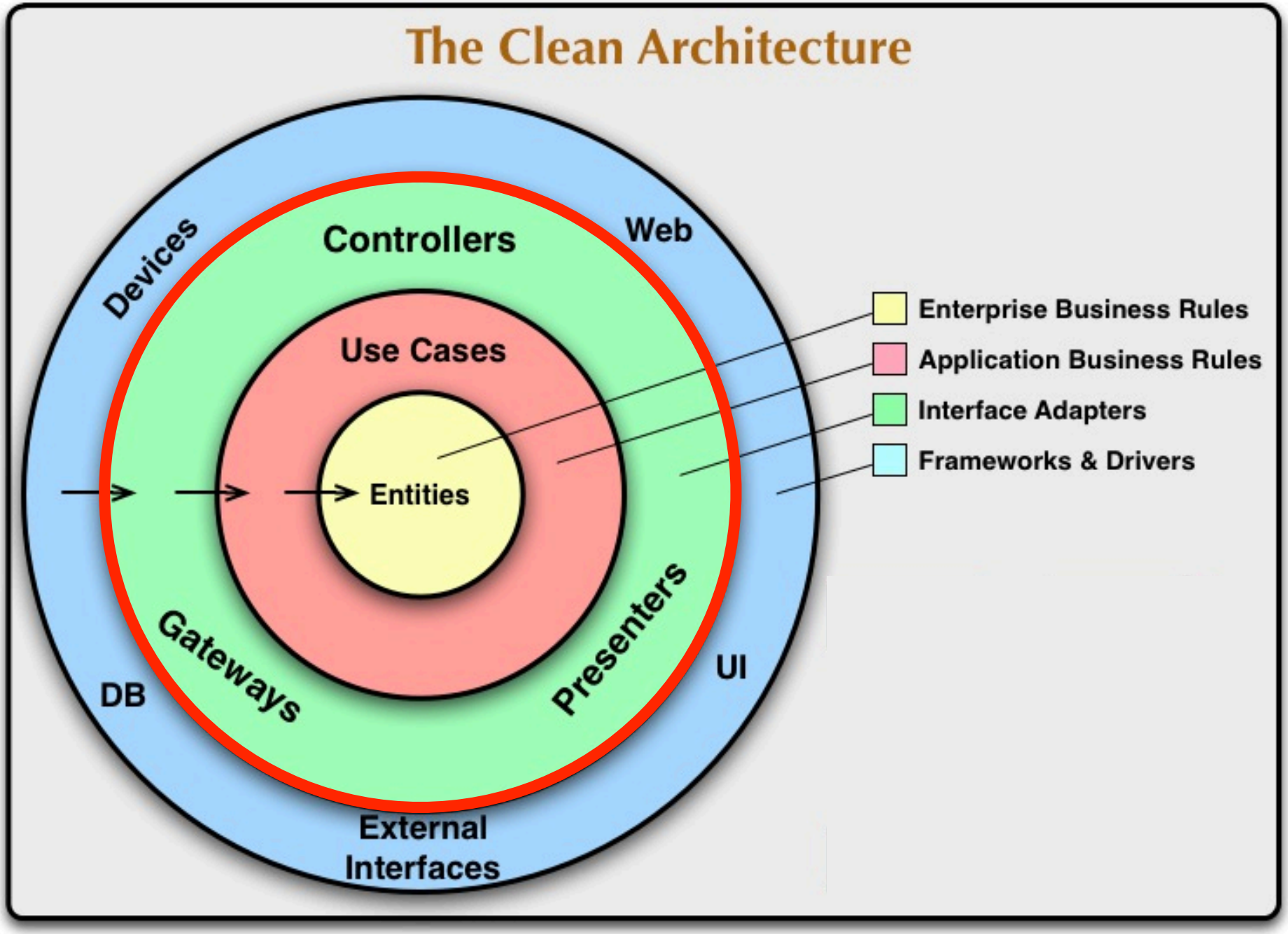
Eles realizam a orquestração das entidades, executando regras de negócio independentes

Casos de uso

- Fazer um pedido
- Cancelar um pedido
- Simular o frete
- Validar um cupom de desconto
- Realizar um pagamento
- Emitir uma nota fiscal


Reparem que os nomes dos casos de uso tem
relação com a Screaming Architecture


Caso de uso é diferente de CRUD?

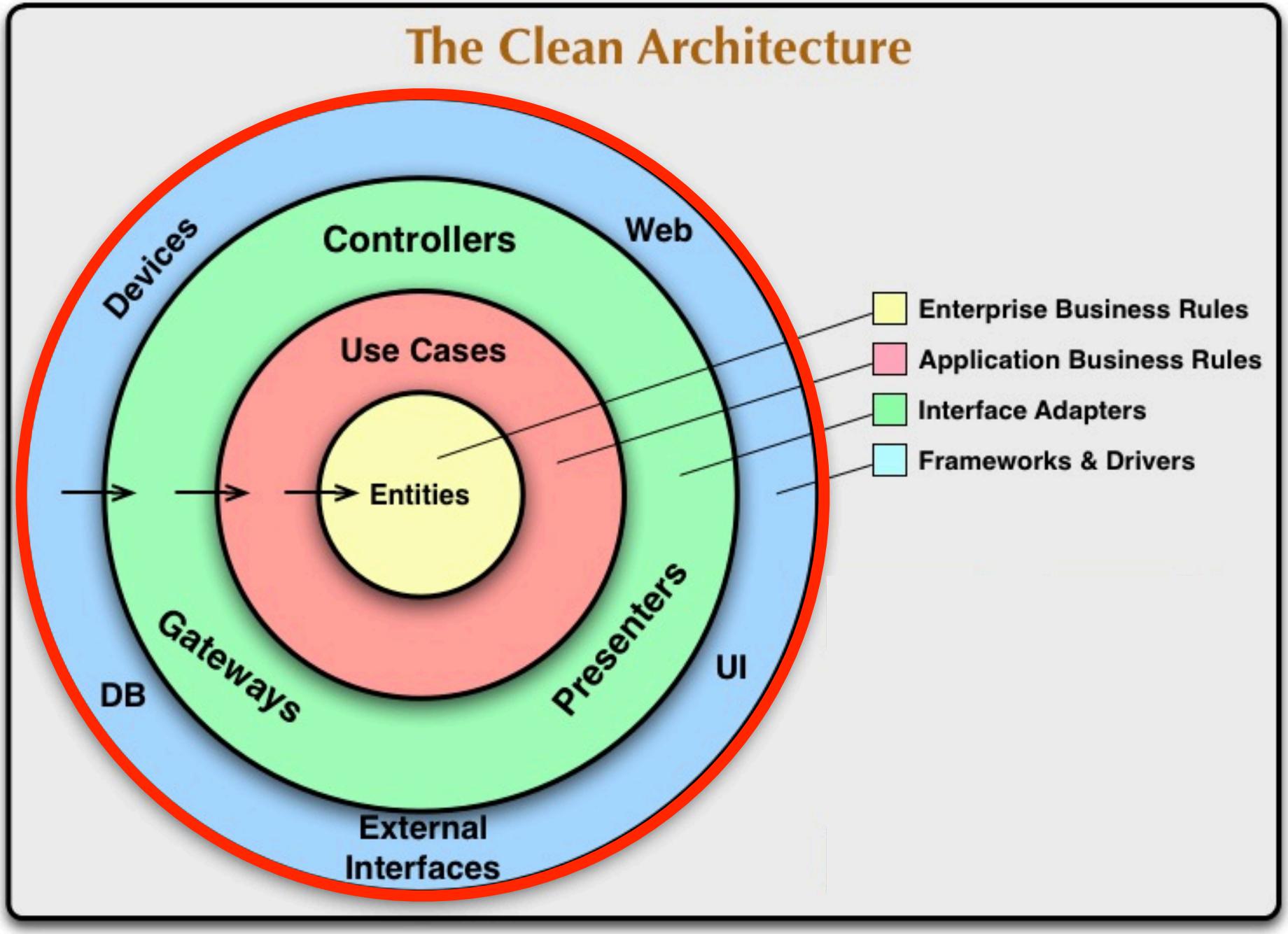

The Clean Architecture
Devices
Web
Controllers
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers


Os interface adapters fazem a ponte entre
os casos de uso e os recursos externos

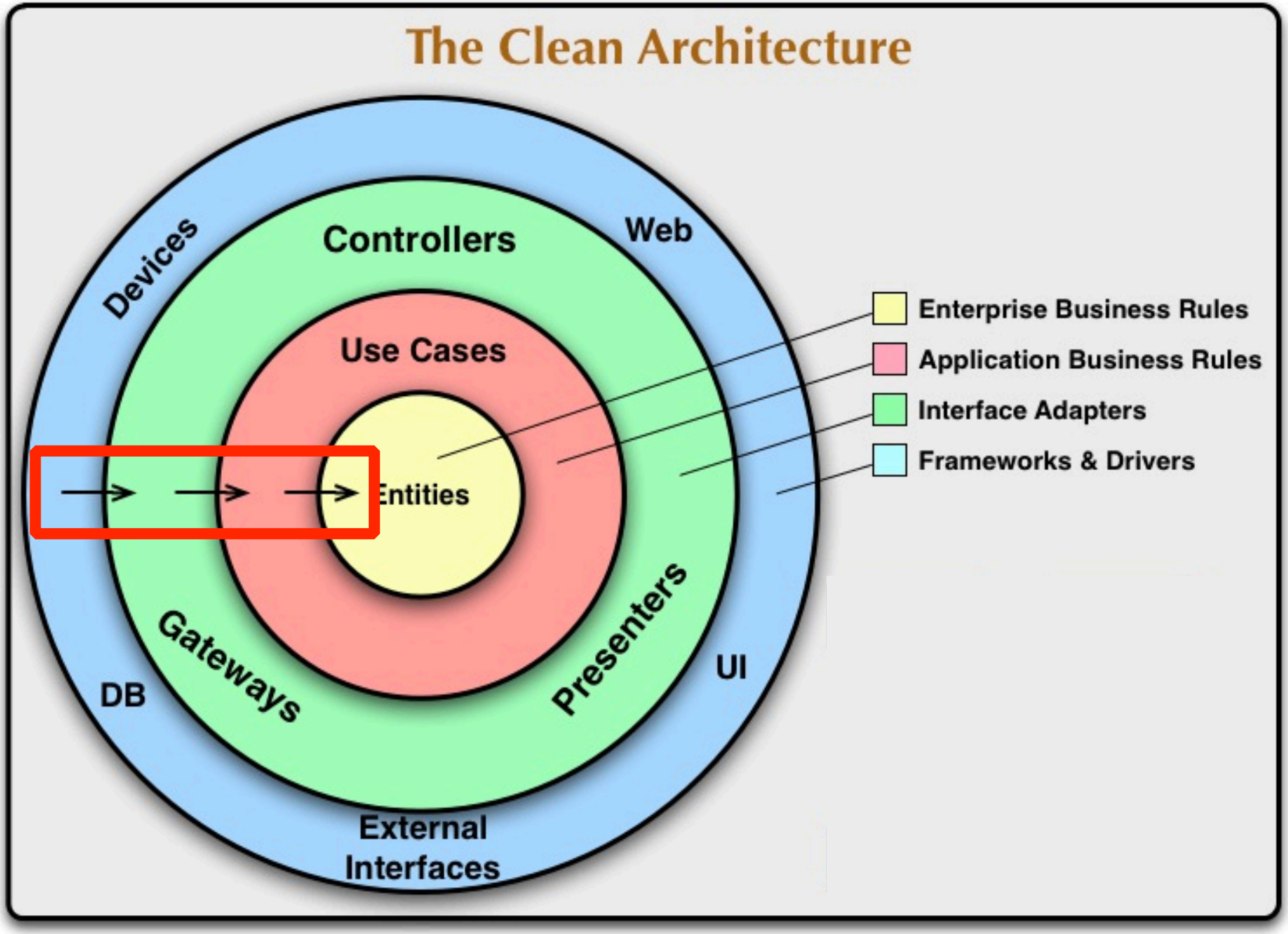
São responsáveis por realizar a conversão de dados de uma tecnologia

Todo código SQL pertence à este camada

Interação com sistema de arquivos e qualquer API externa também

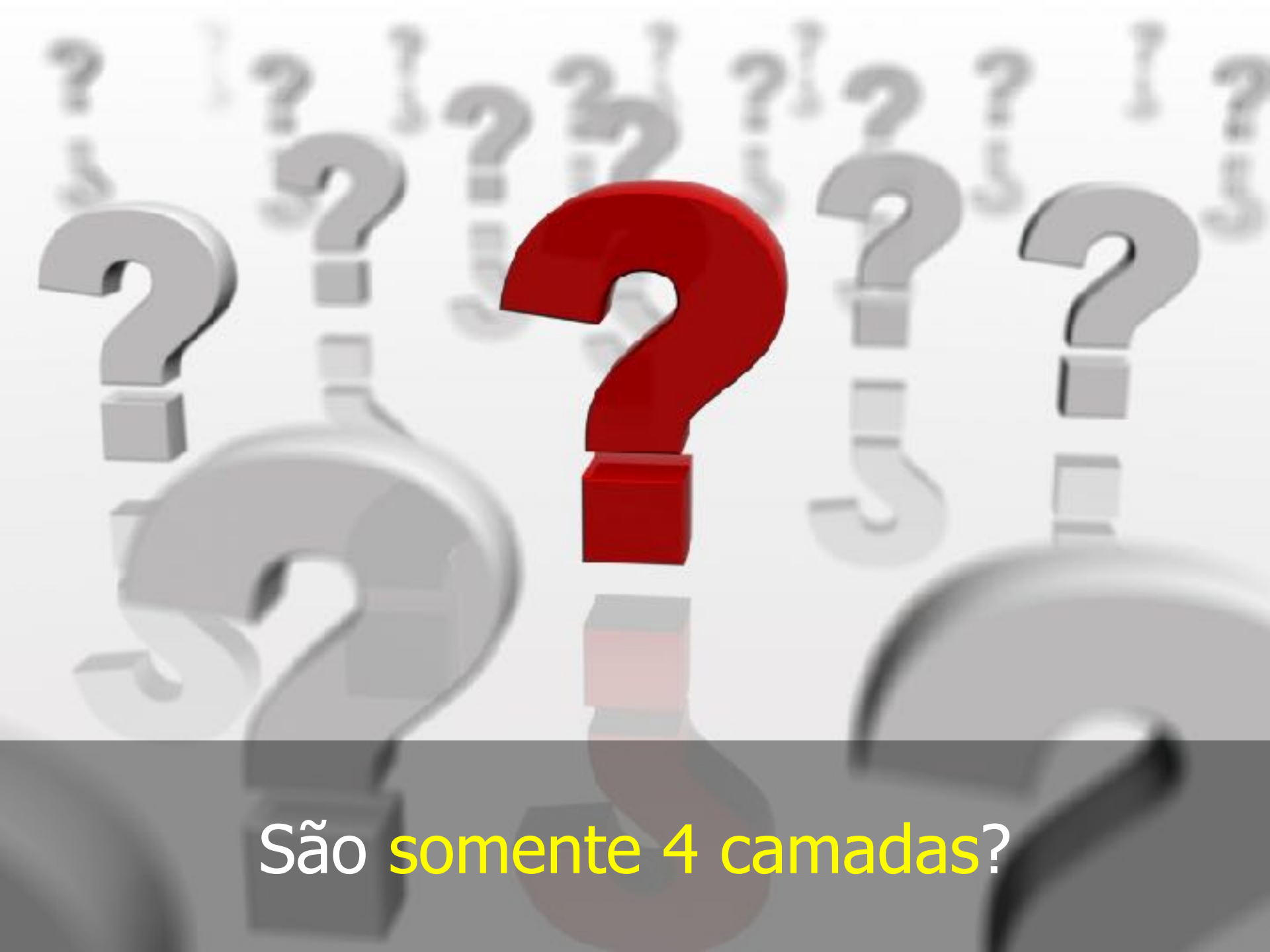

The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers

Por fim, os frameworks and drivers são o nível mais baixo de abstração, é o componente que realiza a conexão com o banco de dados, requisições HTTP, interage com o sistema de arquivos ou acessa recursos do SO


The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers


Dependency Rule

Quem está fora conhece quem está dentro, mas quem está dentro não conhece quem está fora


São somente 4 camadas?

Uma camada é uma fronteira lógica entre um conjunto de componentes, que tem uma responsabilidade bem definida

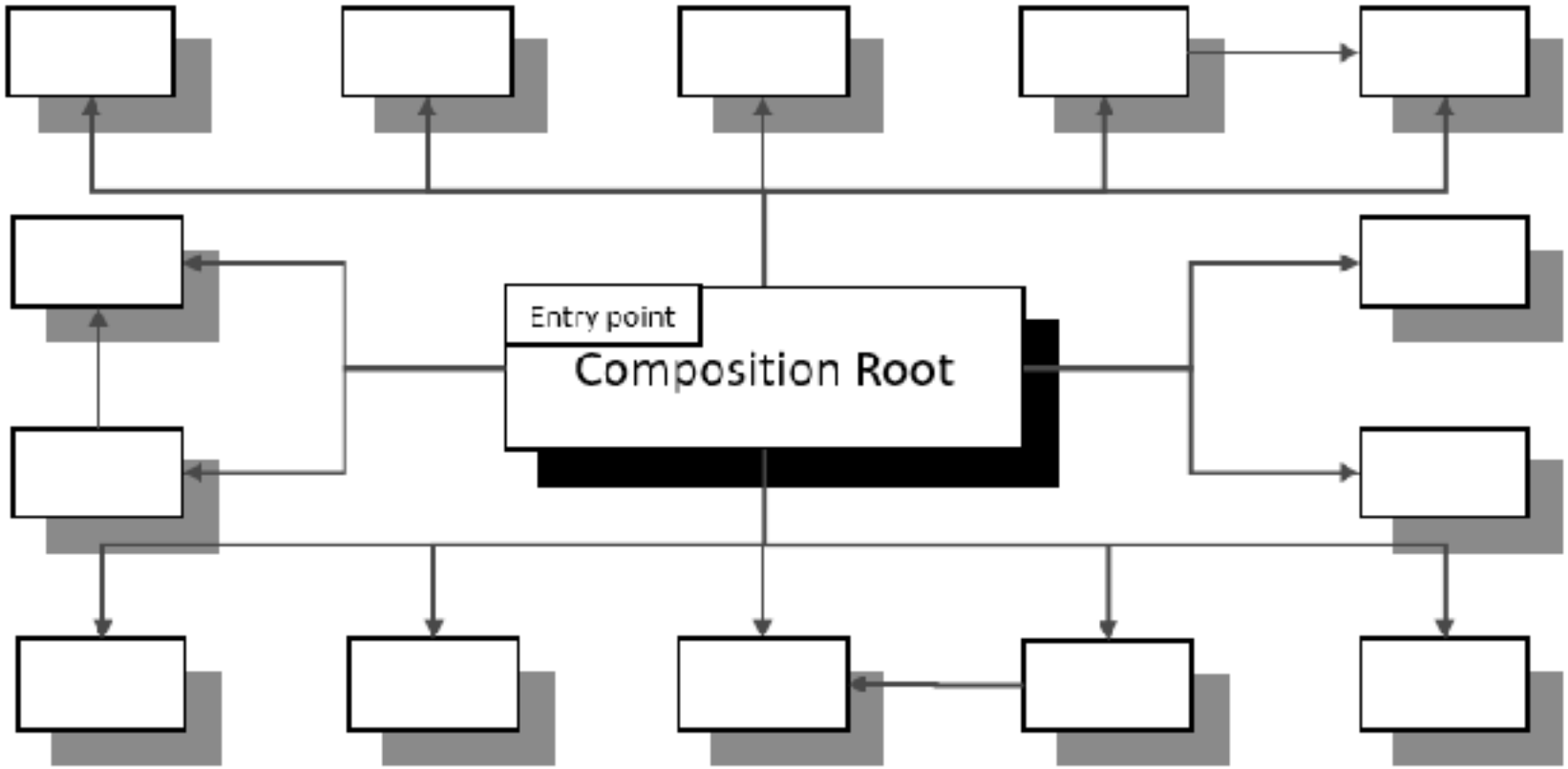

As camadas não tem necessariamente
relação física com as pastas


Como incializamos a aplicação?

O main é o ponto de entrada da aplicação (HTTP, CLI, UI, Testes), é lá que as fábricas e estratégias são inicializadas e as injeções de dependência são realizadas durante a inicialização

When composing an application from many loosely coupled classes, **the composition should take place as close to the application's entry point as possible**. The Main method is the entry point for most application types. The Composition Root composes the object graph, which subsequently performs the actual work of the application.

https://freecontent.manning.com/dependency-injection-in-net-2nd-edition-understanding-the-composition-root/


Entry point
Composition Root