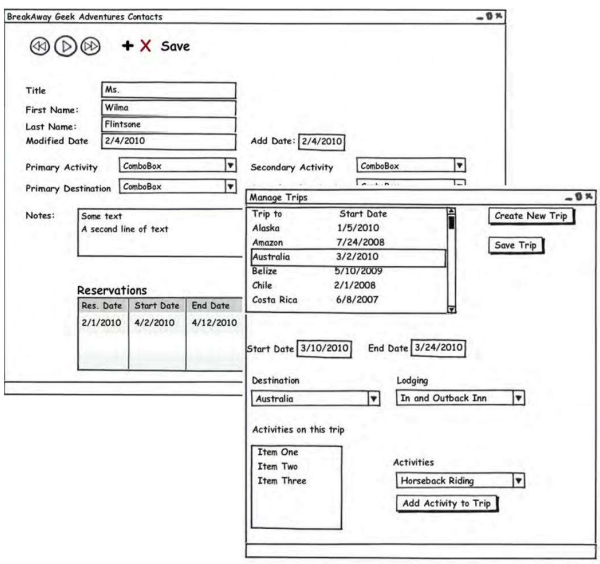
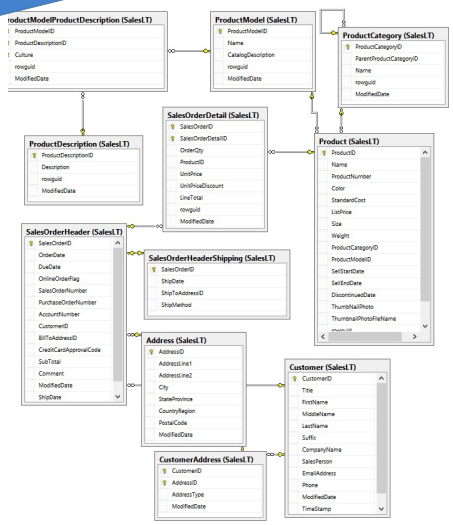
ARCHITECTURAL REFACTORING: CASE STUDY

CRUD-based Workflow

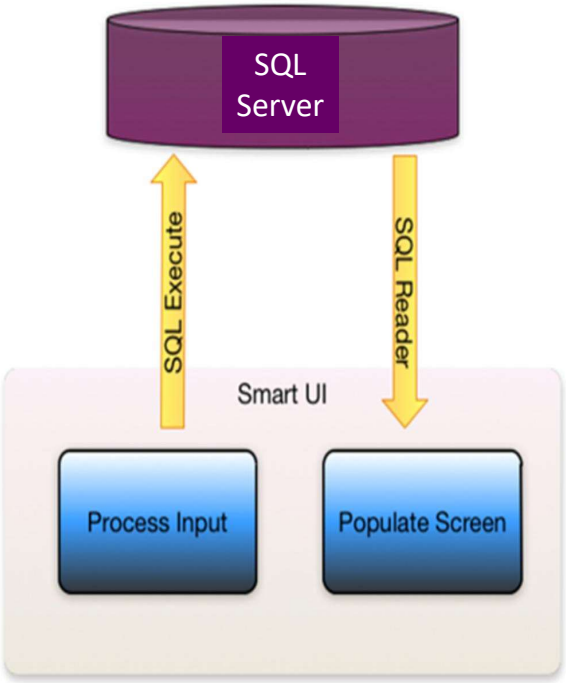


Transaction Script
Active Record

Software Solution

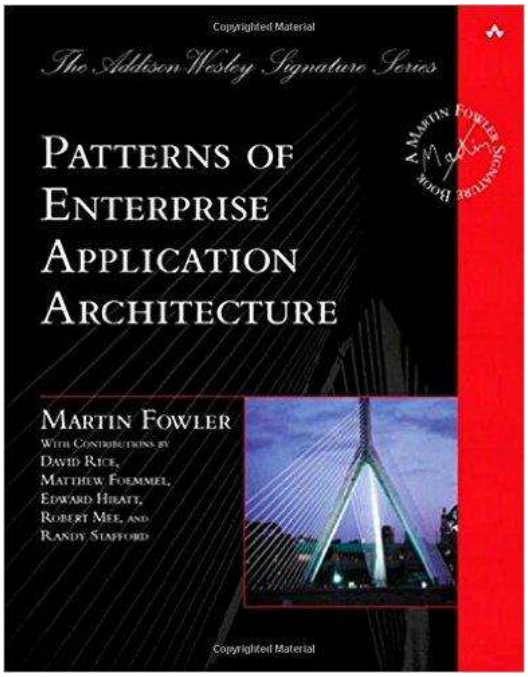


Smart UI Anti-Pattern

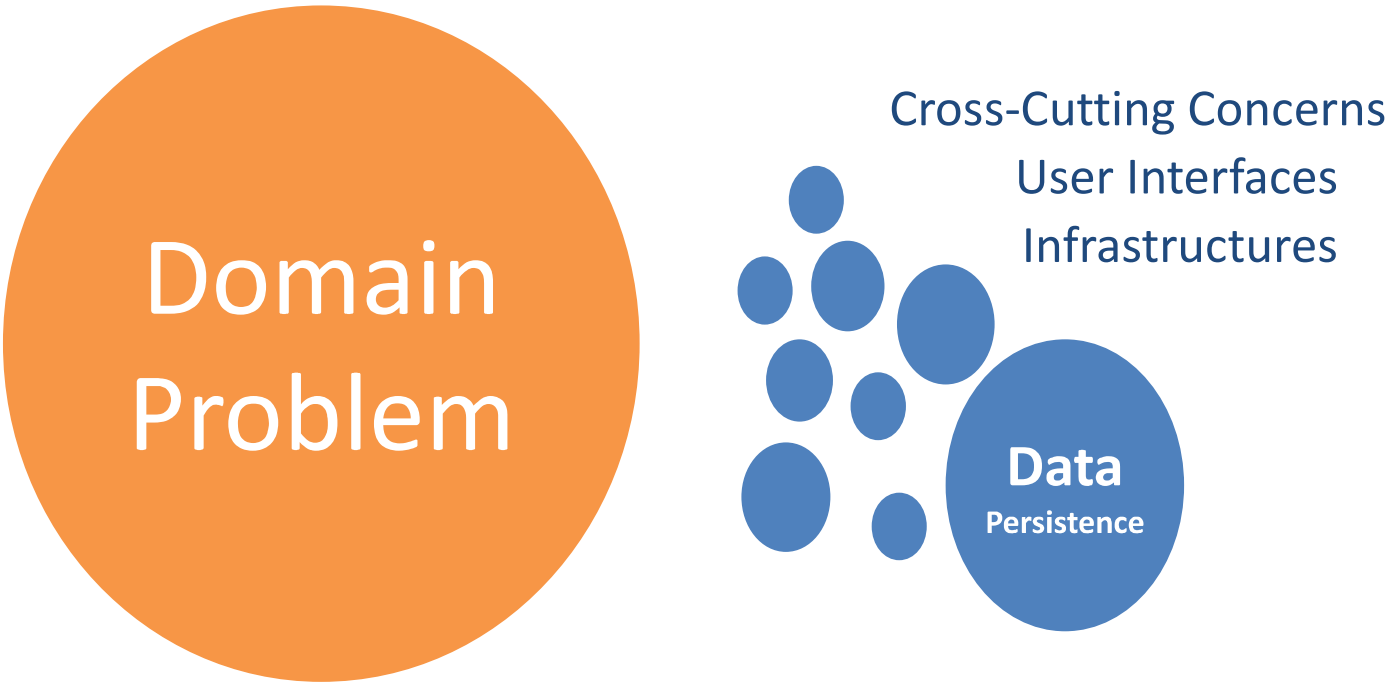


From Simple to Complex, but Flexible

- Transactional Script
  - GUI talks to DB directly
  - No Domain Model
- Active Record
  - An object is a wrapper for row in DB with minimal logic and basic CRUD.
  - Anemic Domain Model
- Data Mapper
  - Complete separation between domain and data source
  - On to “Rich Domain Model”



Focus On The Domain Problem



Isolating the domain

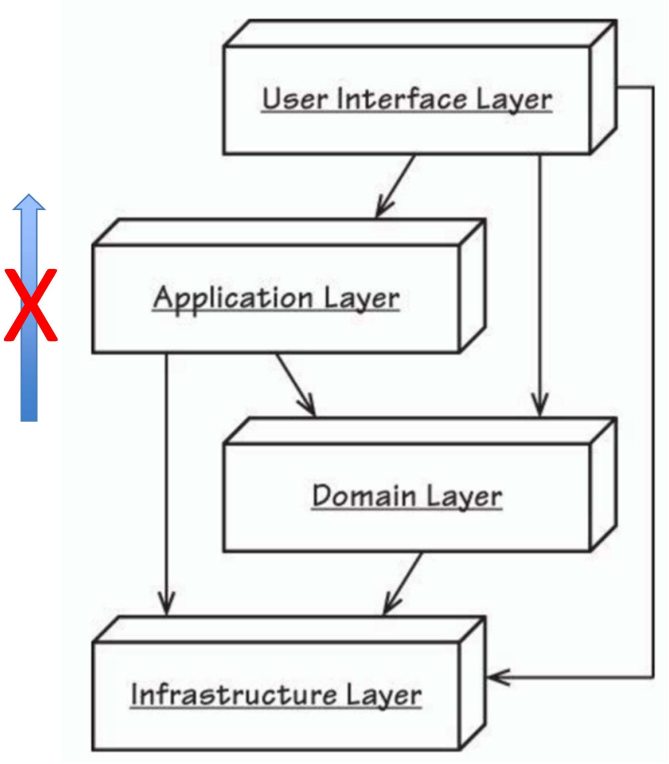
If an architecture isolates domain related code and logic in a single layer such that ...

- It is strongly cohesive
- Loosely coupled with the rest of the system
- Can perform tasks related to the domain concepts or policies delegated from other layers/interfaces.

Then such an architecture is capable of support domain driven designs.

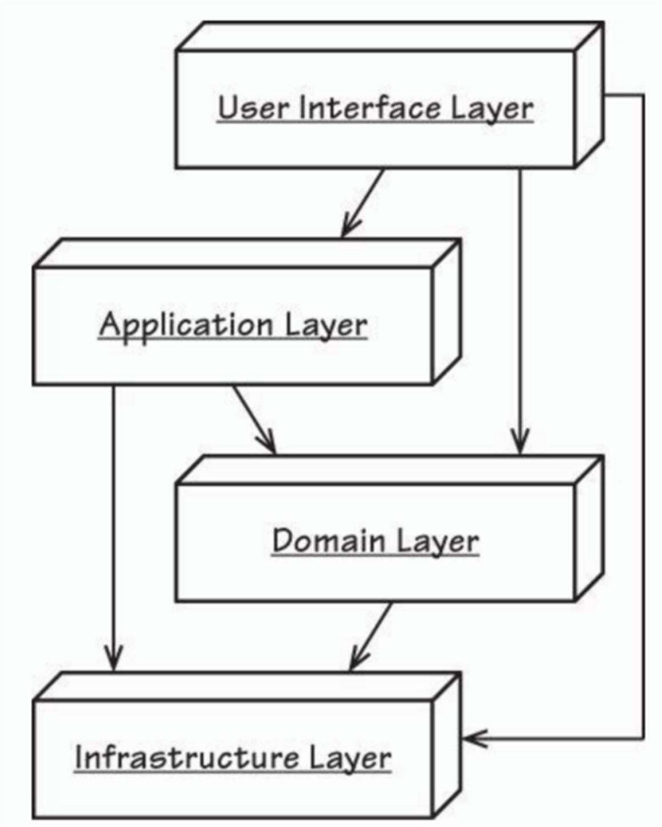
Traditional Layered Architecture

- Each layer should be strongly cohesive.
- Each layer may couple only to itself and below.
- There is never a direct reference from lower to higher
- Lower layers may actually loosely couple to higher layers, but this is only by means of a mechanism such as callbacks or Observer pattern.



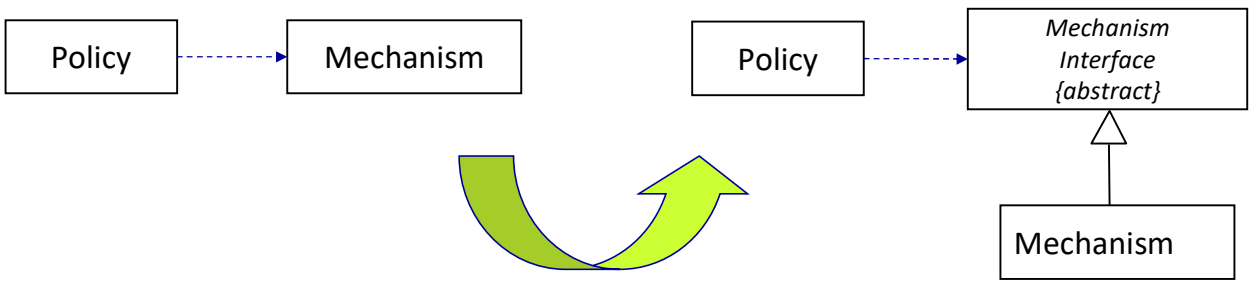
Typical Layers

User Interface (Presentation Layer)	Responsible for presenting information to the user and interpreting user commands.
Application Layer	This is a thin layer which coordinates the application activity. It does not contain business logic. It does not hold the state of the business objects, but it can hold the state of an application task progress.
Domain Layer	This layer contains information about the domain. This is the heart of the business software. The state of business objects is held here. Persistence of the business objects and possibly their state is delegated to the infrastructure layer.
Infrastructure Layer	This layer acts as a supporting library for all the other layers. It provides communication between layers, implements persistence for business objects, contains supporting libraries for the user interface layer, etc.



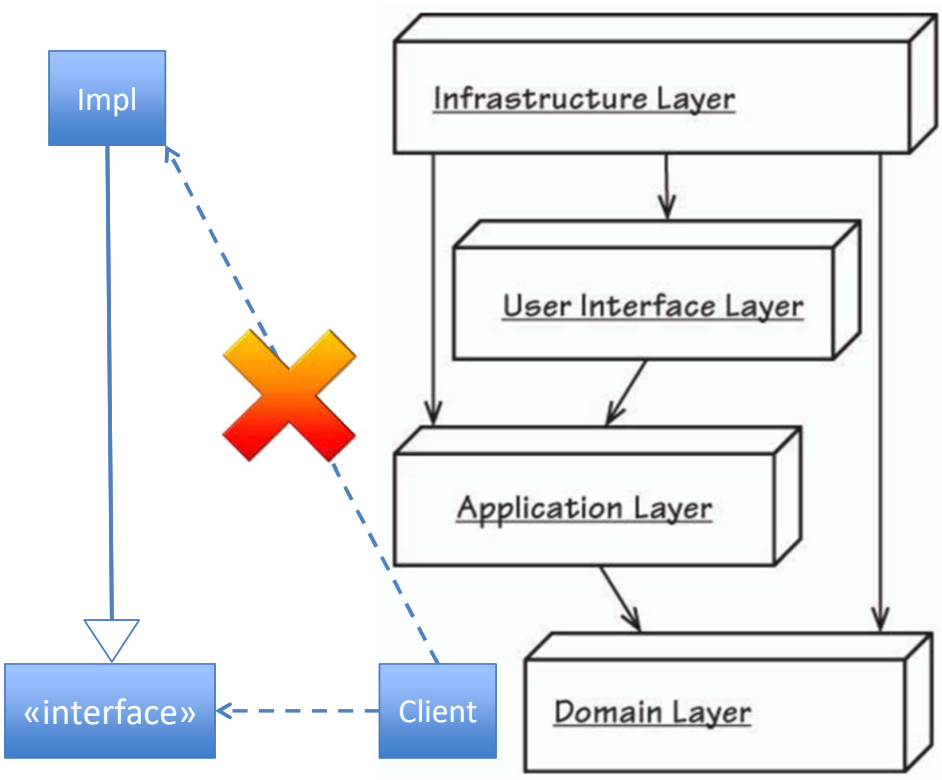
Dependency Inversion Principle (DIP)

High level modules should not depend upon low level modules. Both should depend on abstractions.
Abstractions should not depend upon details. Details should depend upon abstraction.



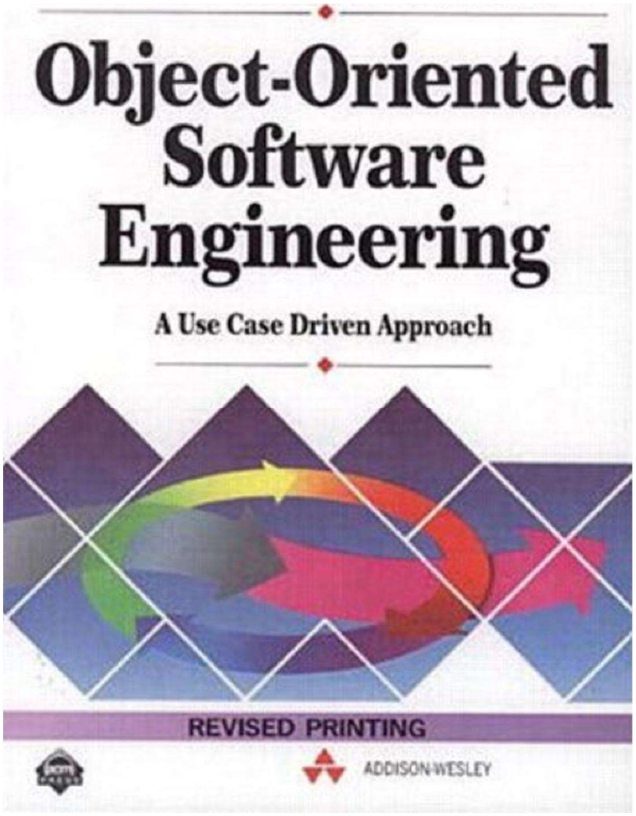
Flexible, extensible, reusable

DIP Applied Layered Architecture



CLEAN ARCHITECTURE

What Happened?



Martin's slides

Use Cases

Create Order

Data:

<Customer-id>, <Customer-contact-info>,
<Shipment-destination>, <Shipment-mechanism>,
<Payment-information>

Primary Course:

1. Order clerk issues "Create Order" command with above data.
2. System validates all data.
3. System creates order and determines order-id.
4. System delivers order-id to clerk.

Exception Course: Validation Error

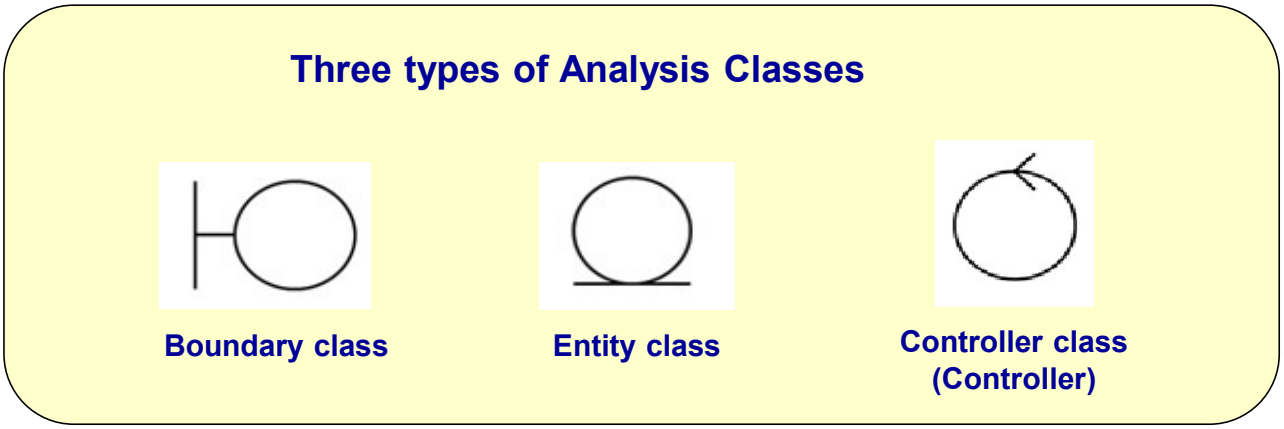
1. System delivers error message to clerk

Martin's slides

Robustness Analysis: What objects participate in each use case?

Draw an "object picture" of the use case

Use the boundary/control/entity stereotypes

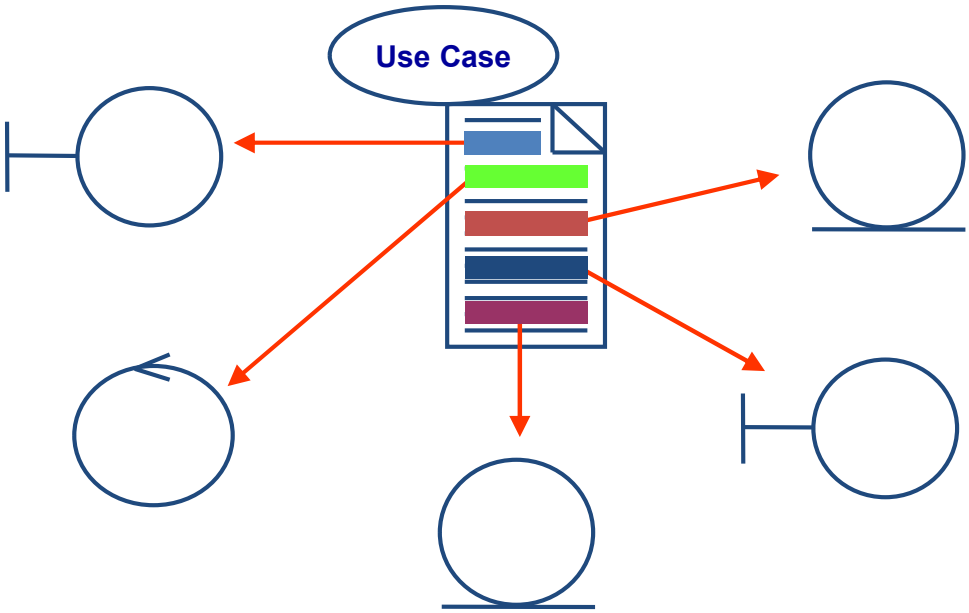


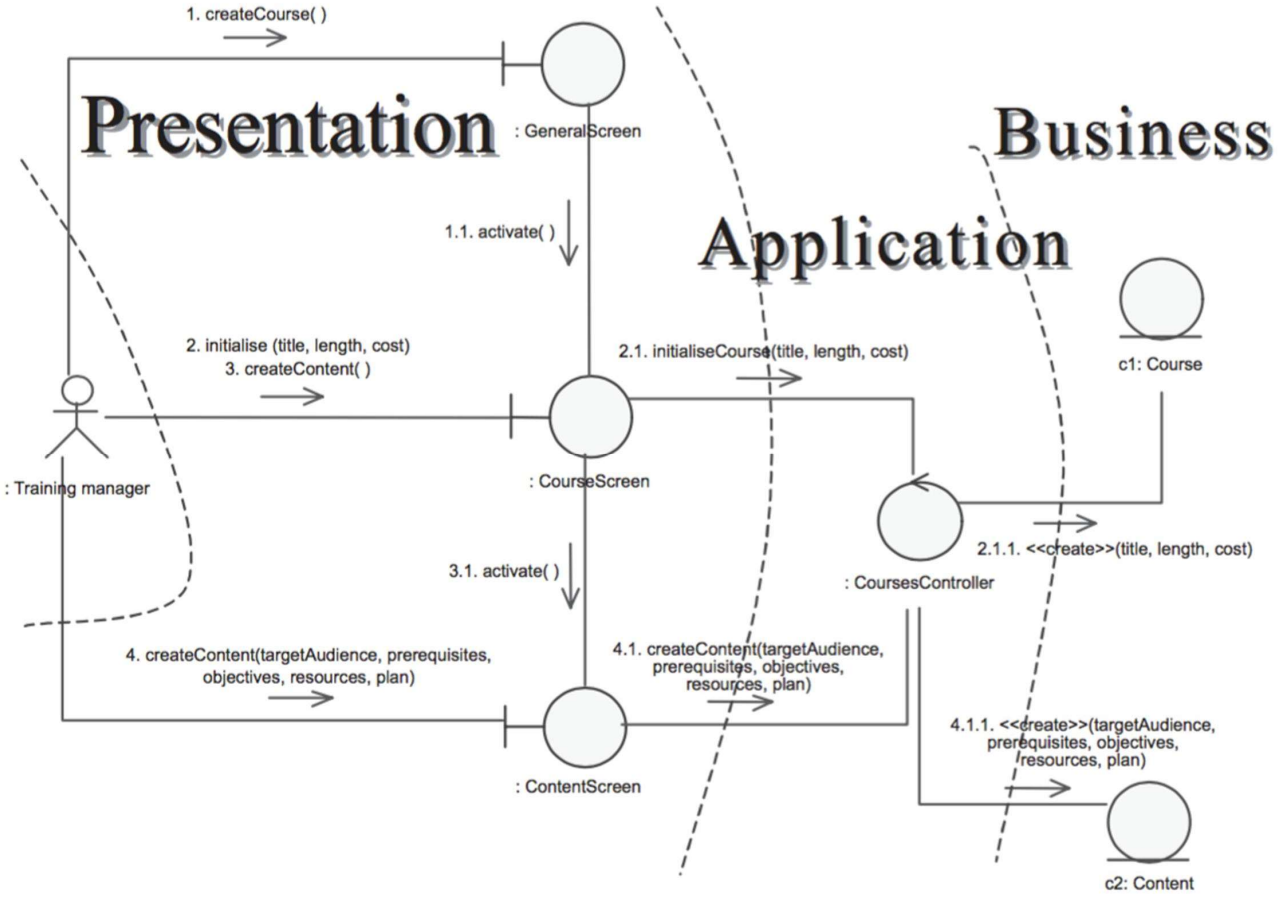
Analysis Objects

- Boundary class
  - Interfaces between system and its environment
- Entity Class
  - Application-independent business rules
  - Represents the key concepts of the system
- Controller Class (aka, Controller)
  - Application-specific business rules
  - Use-case behavior coordinator

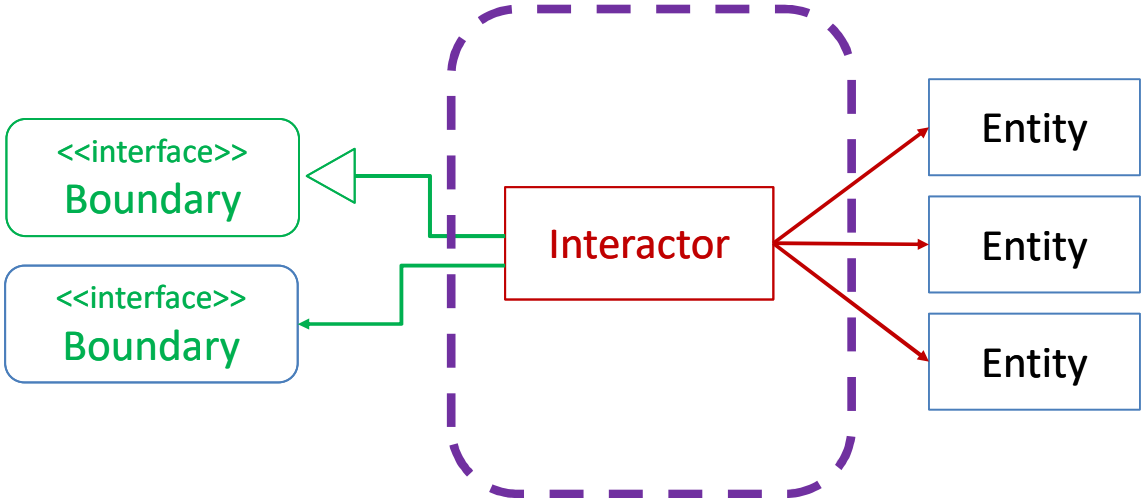
Find Analysis Classes from Use Case

The complete behavior of a use case has to be distributed to analysis classes



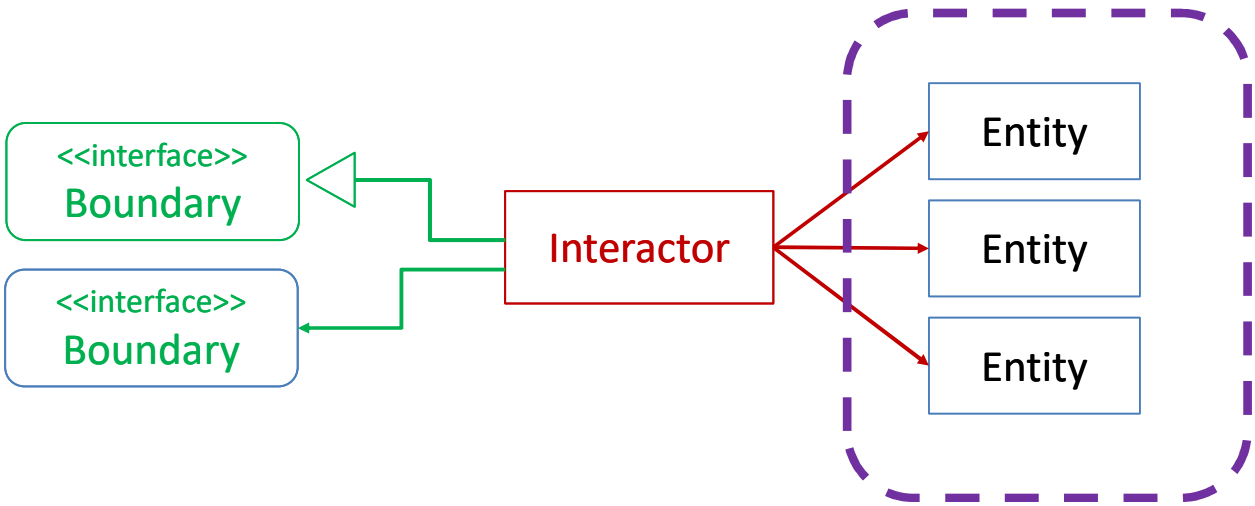


Interactors



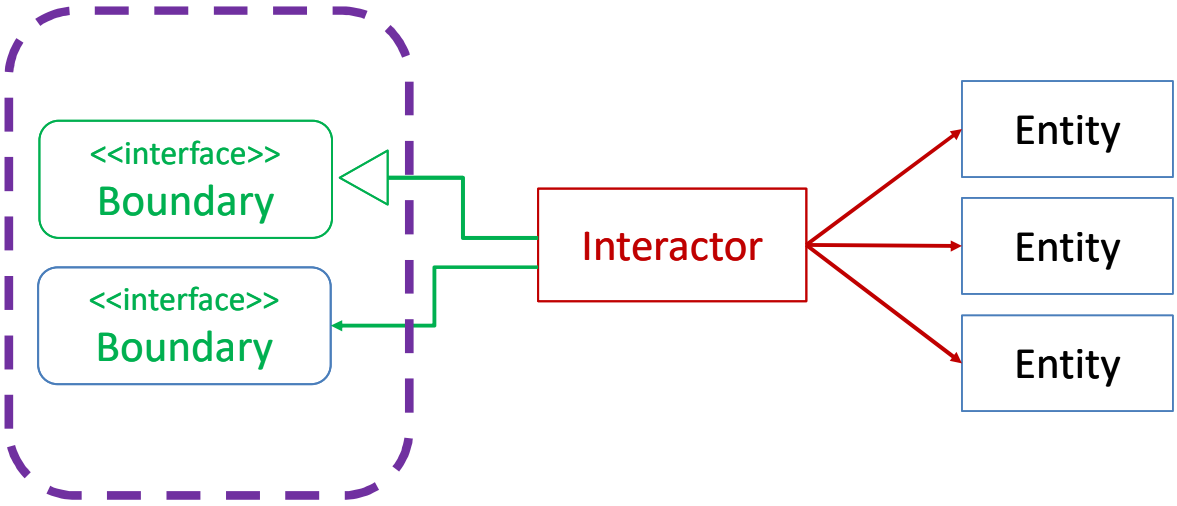
Interactors have: Application specific business rules

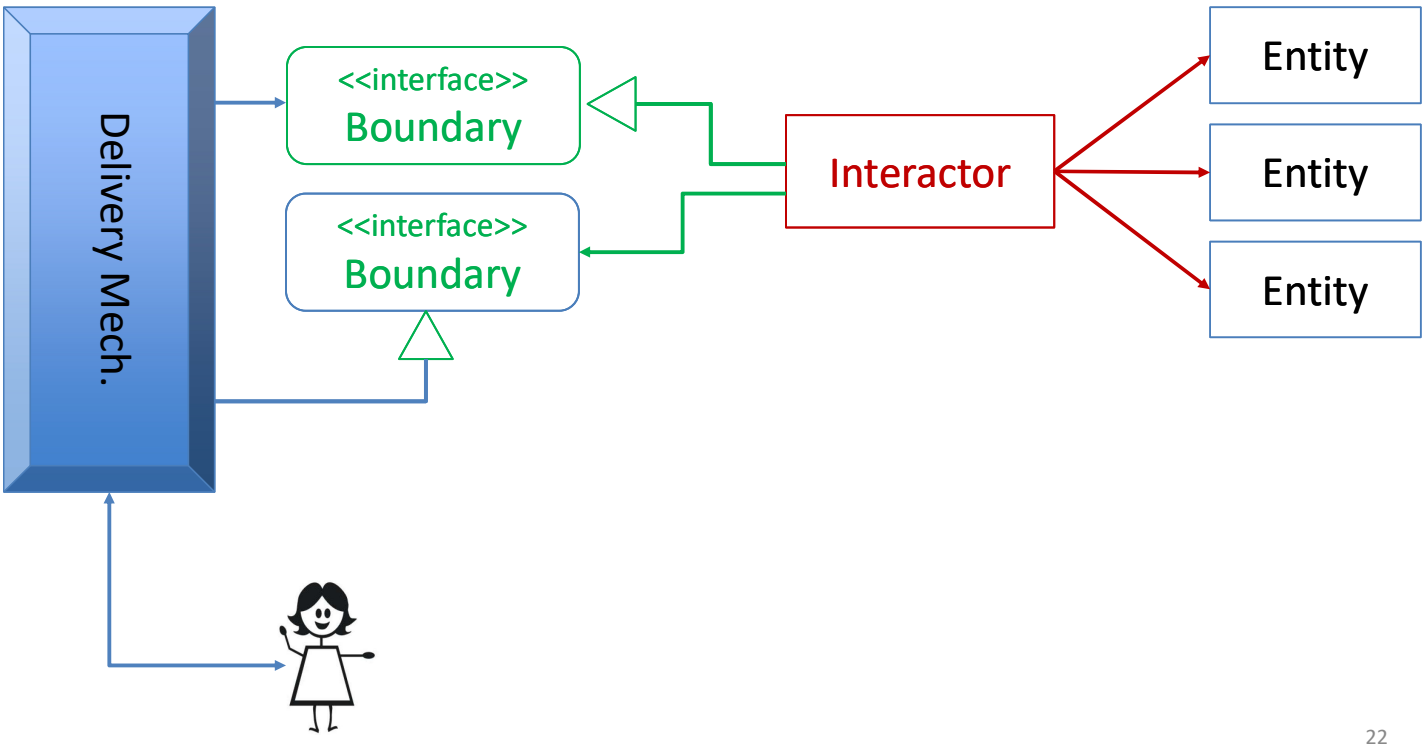
Entities

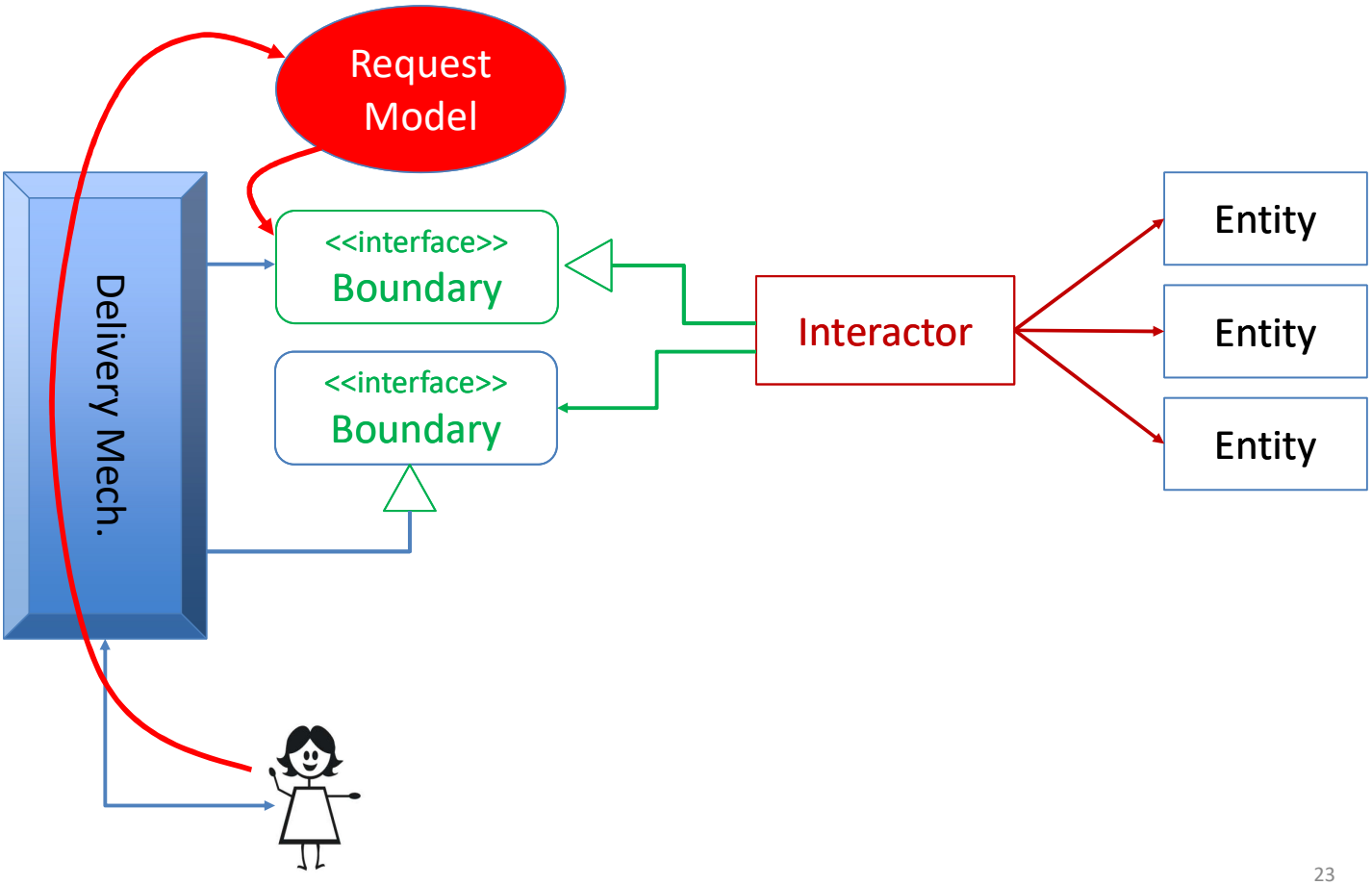


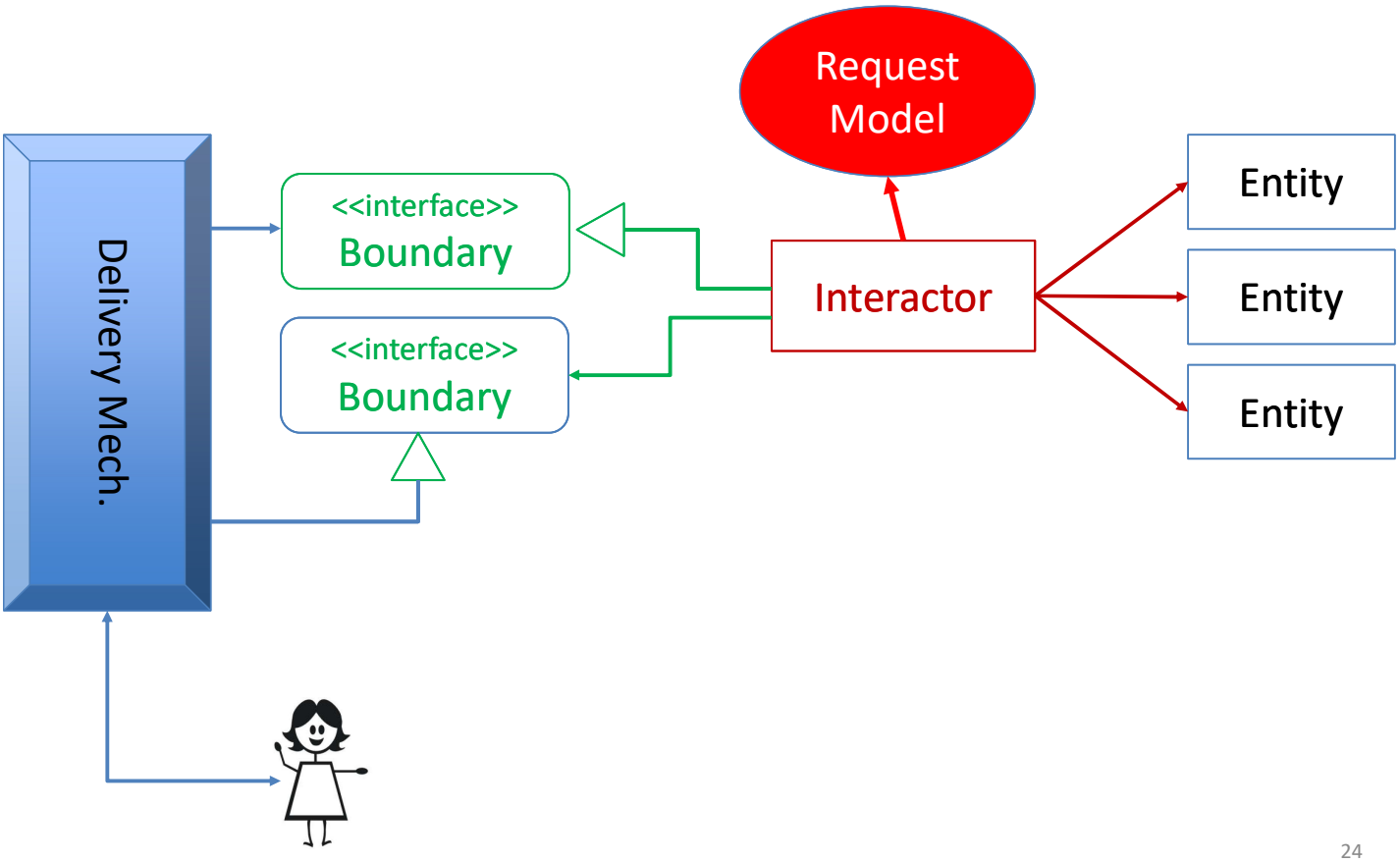
Entities have: Application independent business rules

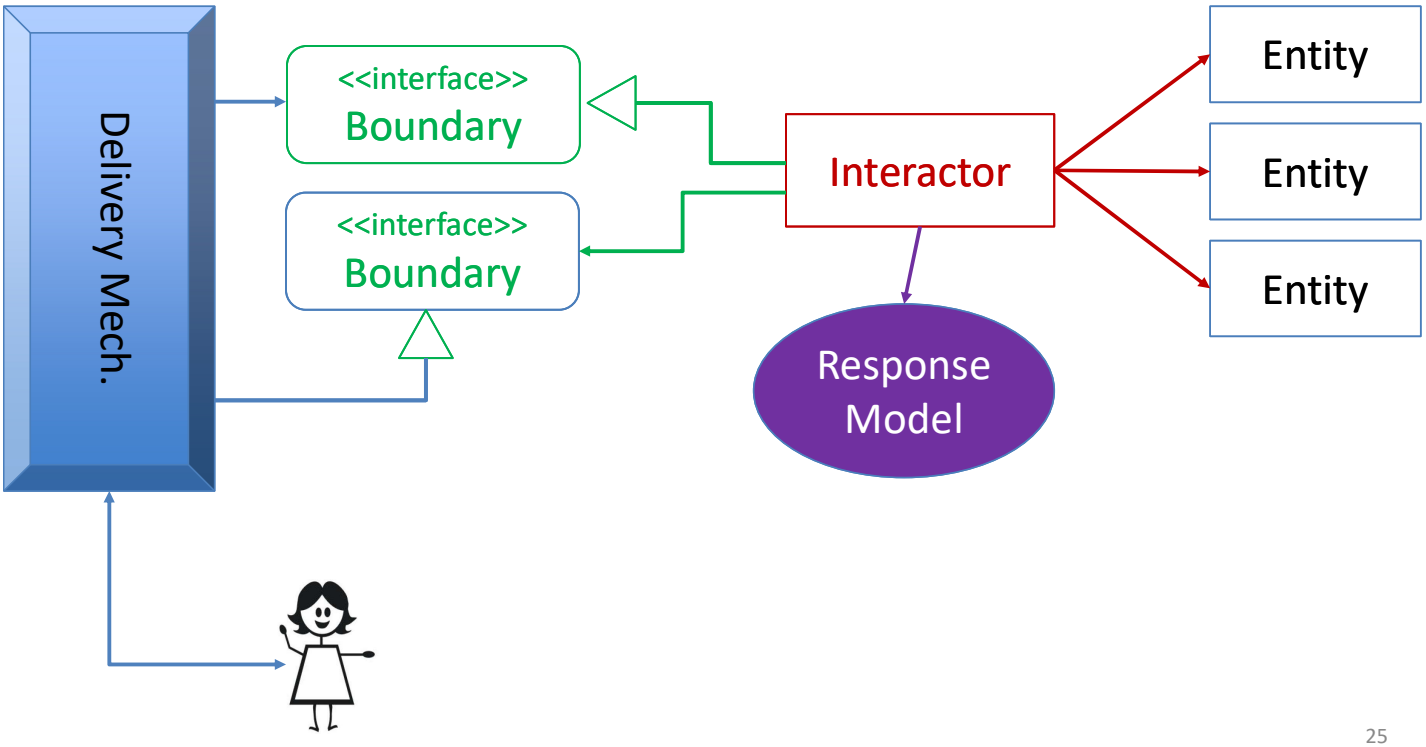
Boundaries

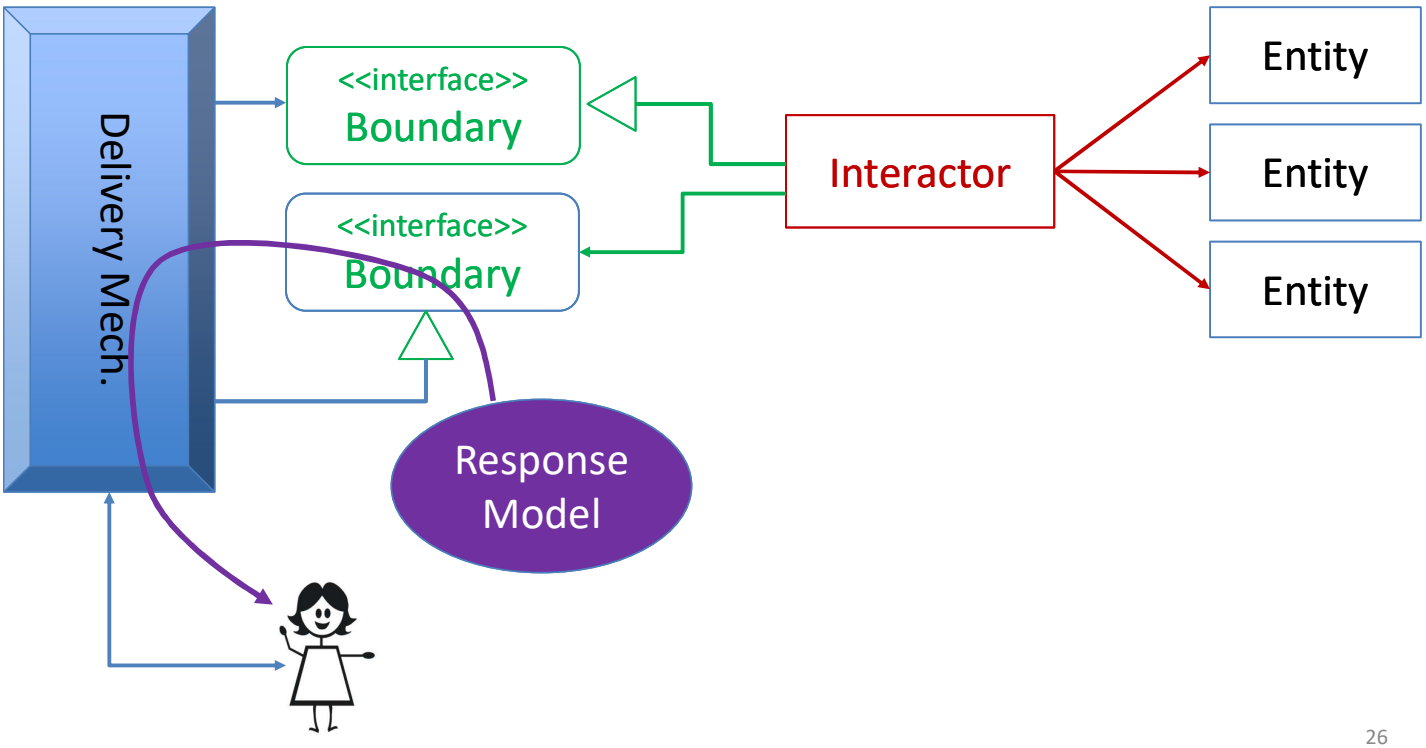

<<interface>>
Boundary
<<interface>>
Boundary
Interactor
Entity
Entity
Entity


Delivery Mech.
<<interface>>
Boundary
<<interface>>
Boundary
Interactor
Entity
Entity
Entity

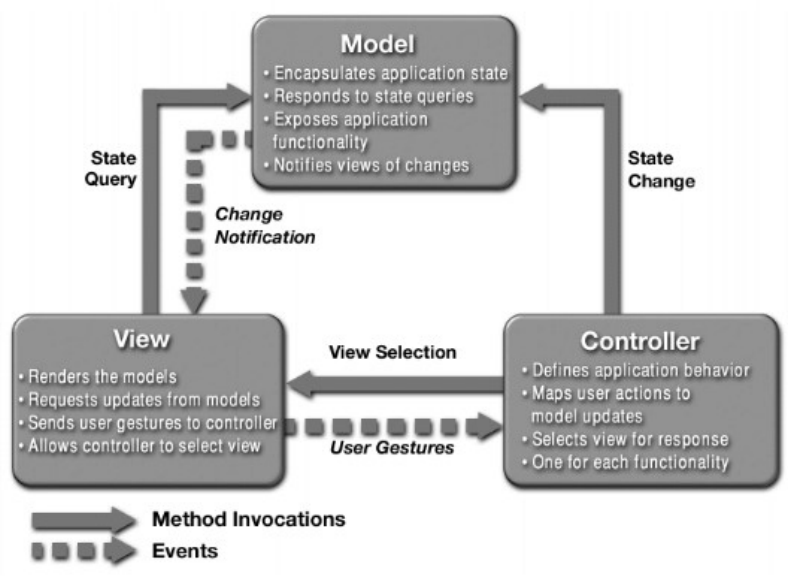

Request
Model
Delivery Mech.
<<interface>>
Boundary
<<interface>>
Boundary
Interactor
Entity
Entity
Entity


Request
Model
Delivery Mech.
<<interface>>
Boundary
<<interface>>
Boundary
Interactor
Entity
Entity
Entity


Delivery Mech.
<<interface>>
Boundary
<<interface>>
Boundary
Interactor
Entity
Entity
Entity
Response
Model


Delivery Mech.
<<interface>>
Boundary
<<interface>>
Boundary
Interactor
Entity
Entity
Entity
Response
Model

Model-View-Controller



Model

Contains business logic

View

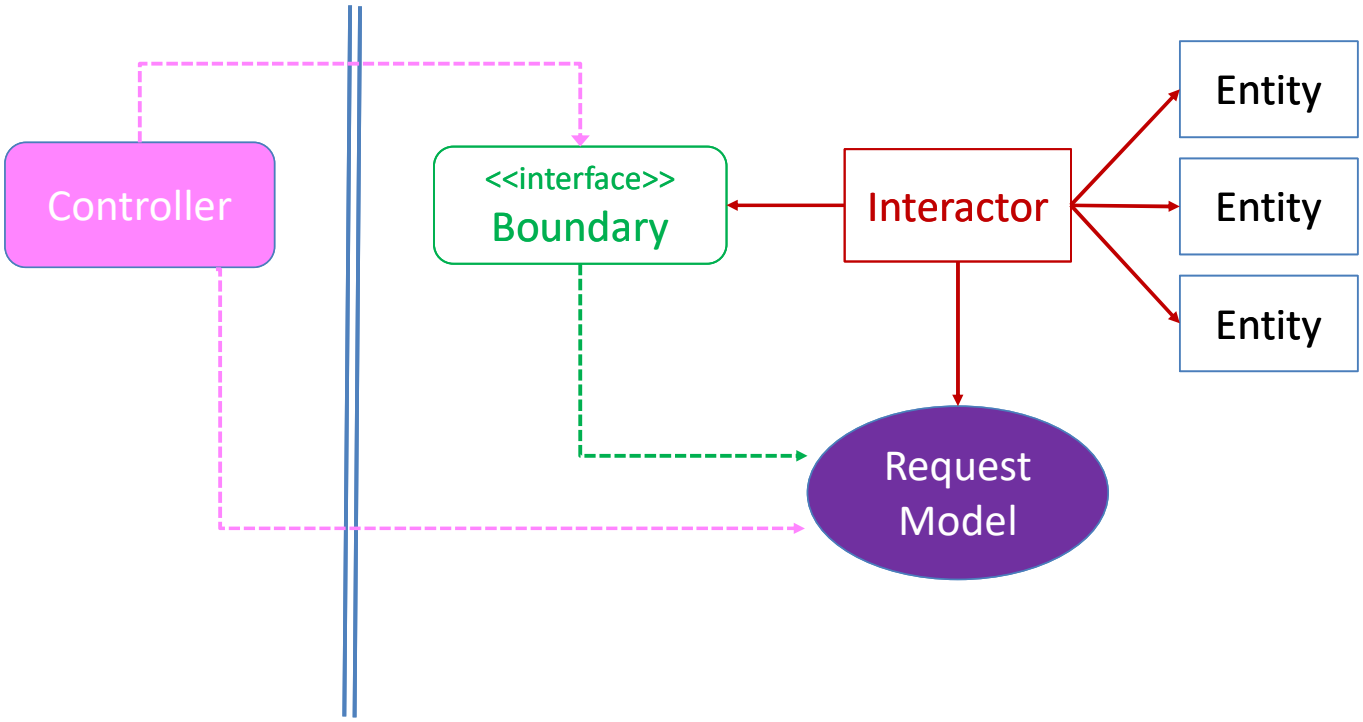
Interacts with **Model** to create data for **View**

Controller

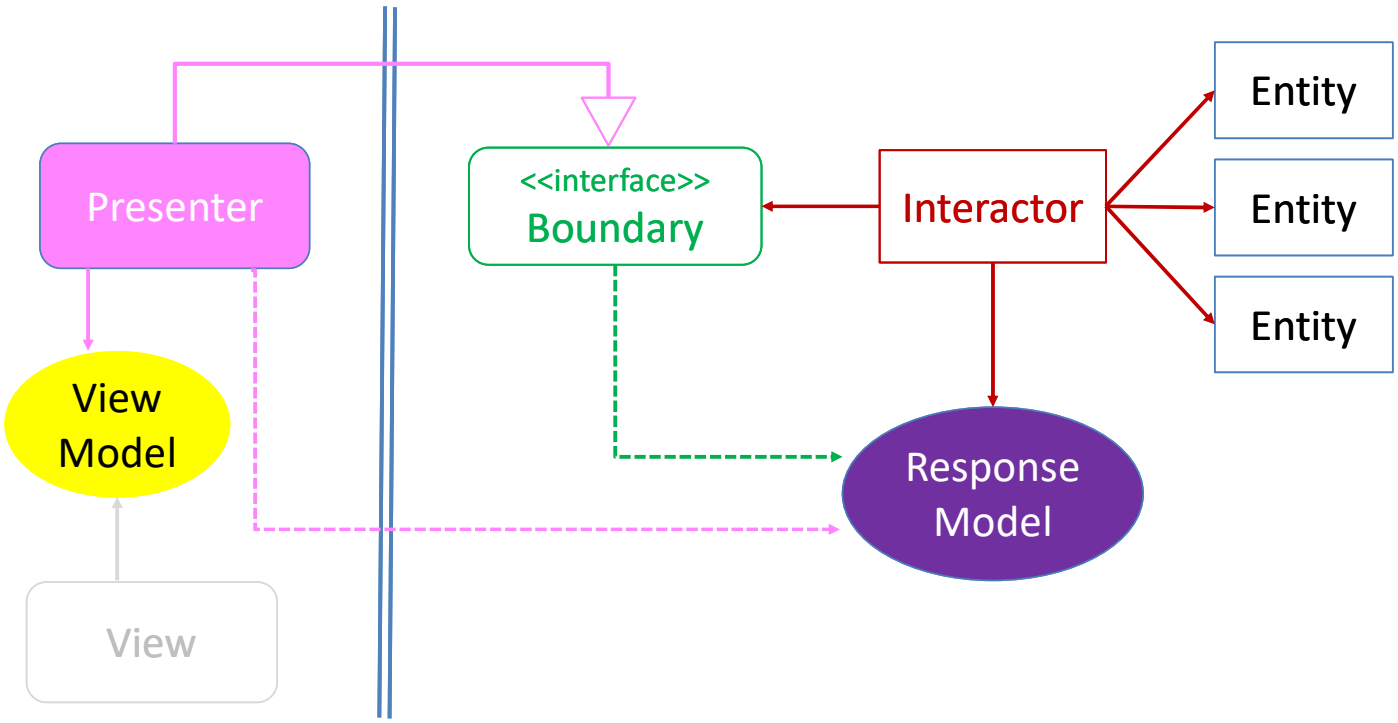
Renders contents and relays use commands to the **Controller**

MVC is a delivery mechanism

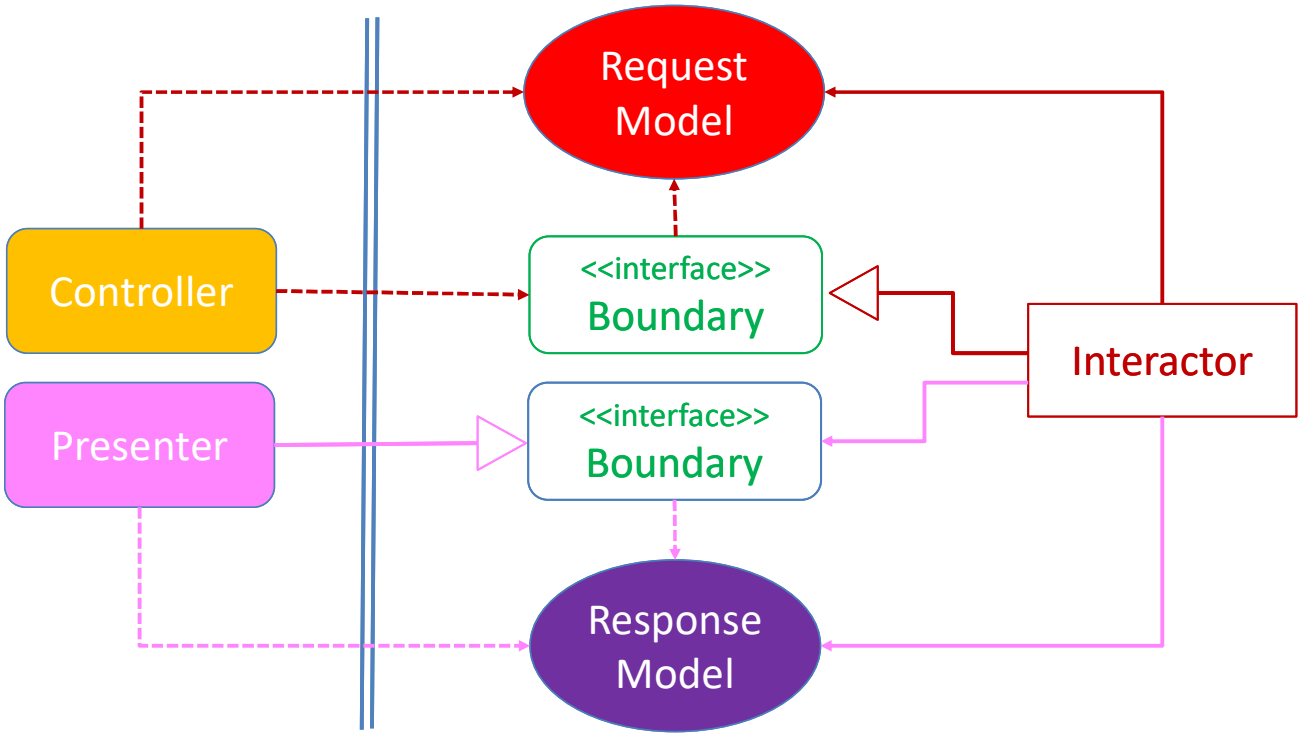
Model View Controller



Model View Presenter

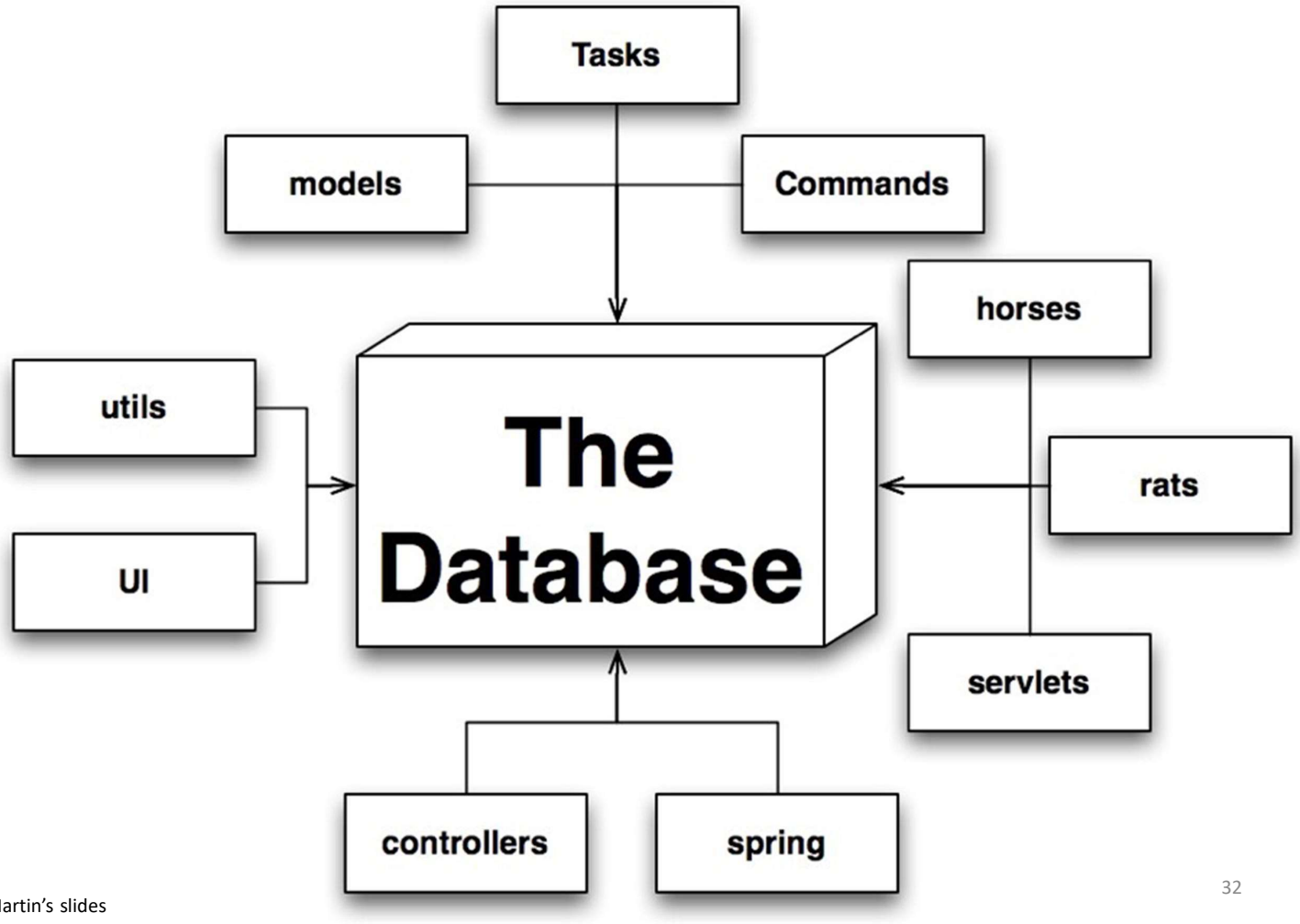


WHOLE ENCHILADA!



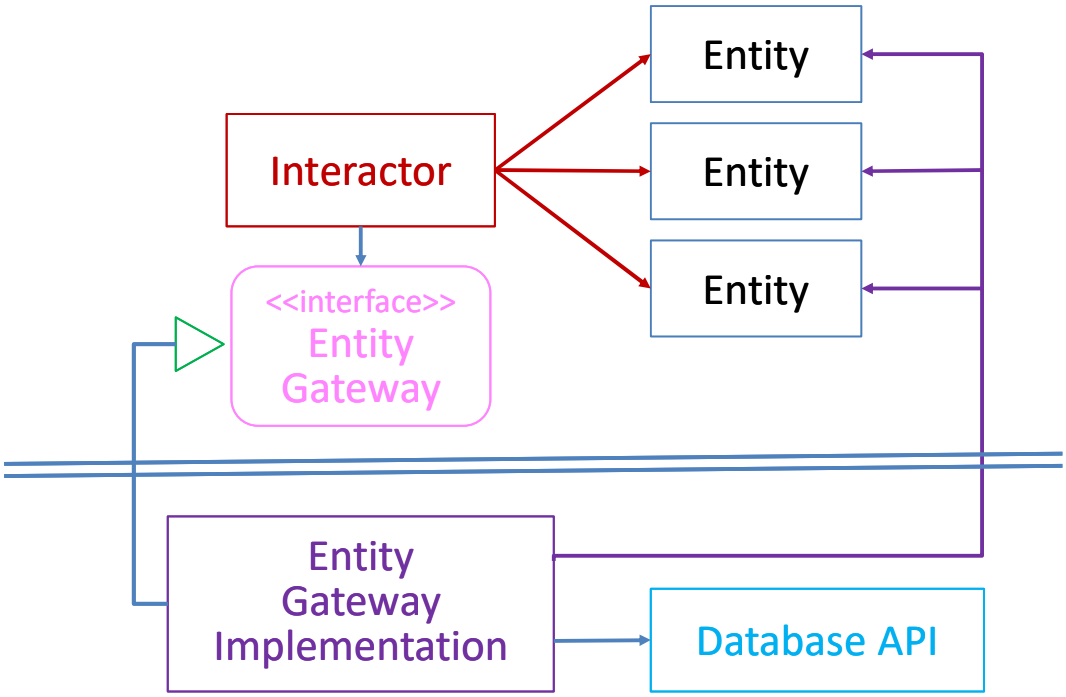
What about Database?

Martin's slides



Martin's slides

DB is a detail… Isolate It!



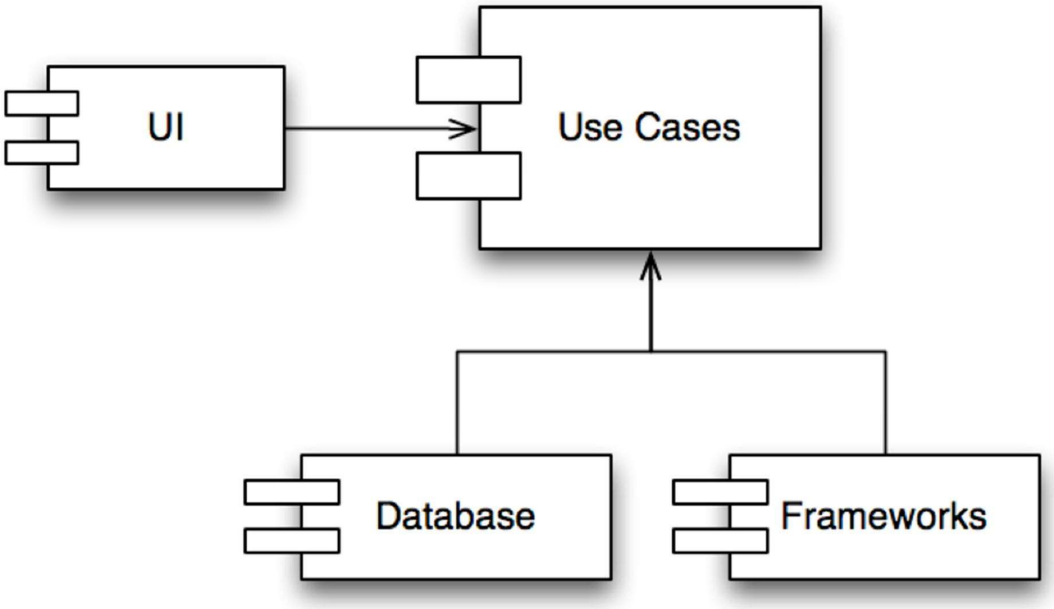
Martin's slides

A good architecture allows major decisions to be deferred!

A good architecture maximizes the number of decisions NOT made!

Martin's slides

PLUG-IN Model



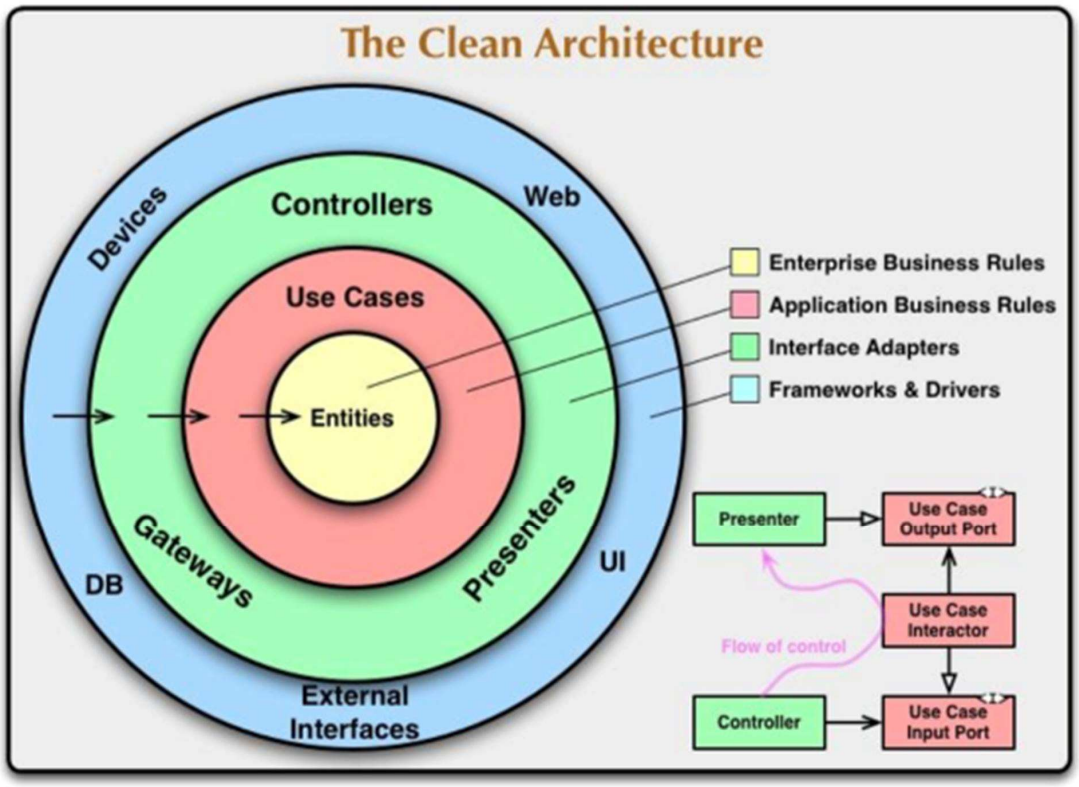
Martin's slides

Definition of Clean Architecture

- Independent of Frameworks
- Testable
- Independent of UI
- Independent of Databases
- Independent of Any External Agencies

http://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html

Clean Architecture



Layers

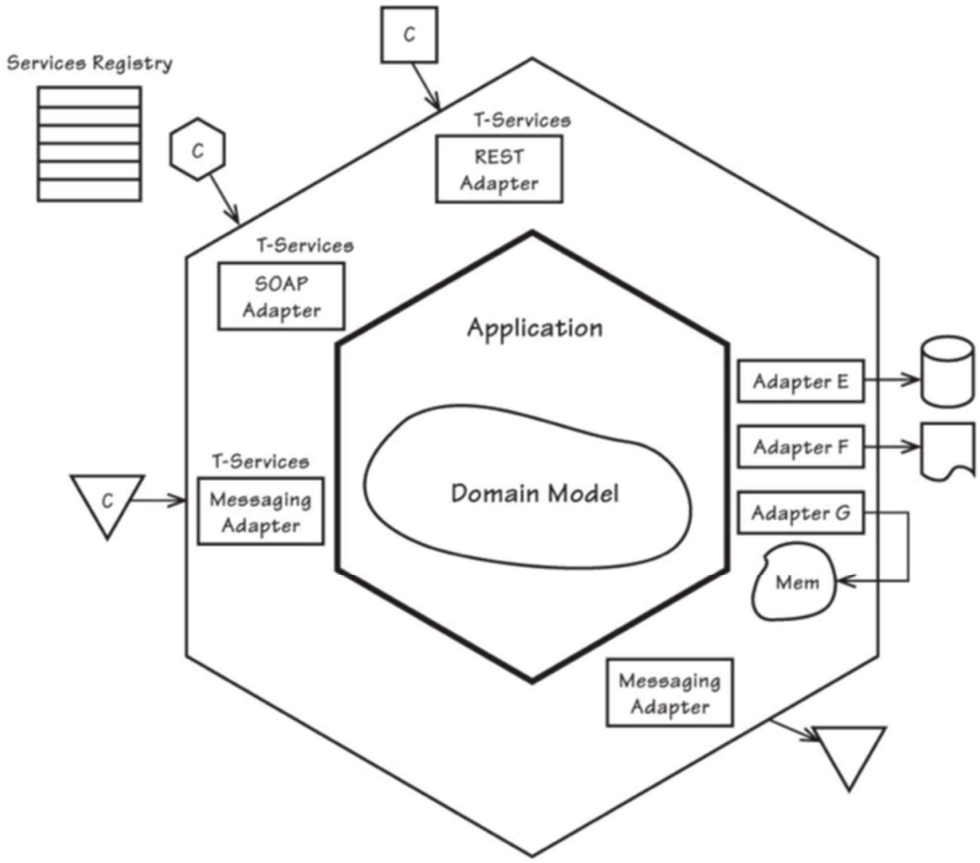
As you move inwards the level of abstraction increases.

- Entities
  - Domain Layer
- Use Cases
  - Application Layer
- Interface Adapters
  - Presentation Layer (UI-level MVC)
- Frameworks and Drivers
  - Infrastructure Layer

Cousins

All based on Separation of Concerns

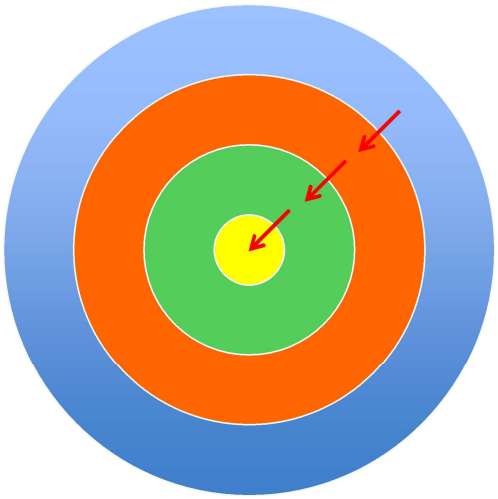
- Hexagonal Architecture (a.k.a Ports & Adapters)
  - Alistair Cockburn
- Onion Architecture
  - Jeffrey Palermo
- Scream Architecture
  - Uncle Bob
- DCI (Data, Context, and Interaction)
  - James Coplien, and Trygve Reenskaug
- BCE (Boundary-Controller-Entity)
  - Ivar Jacobson

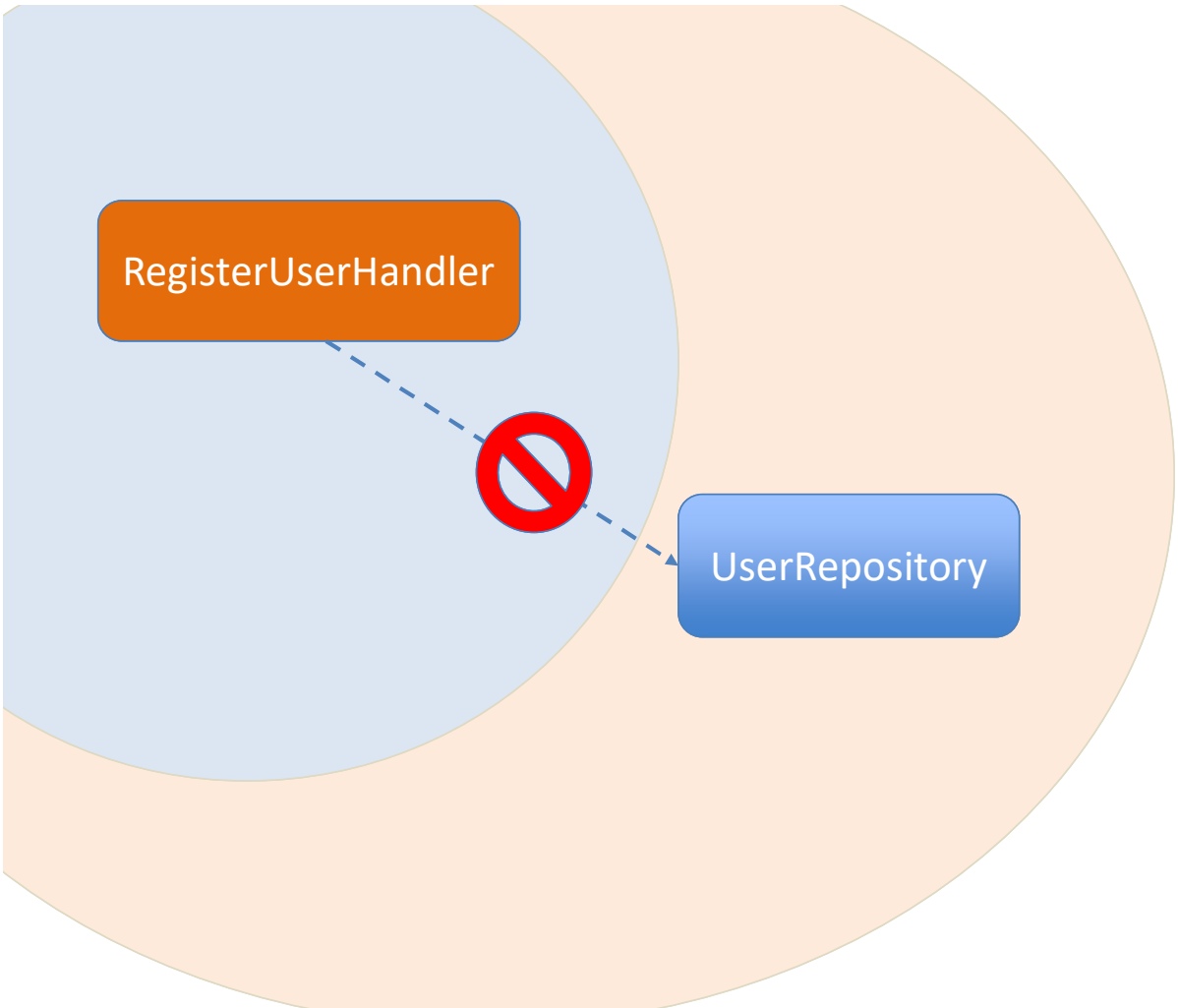


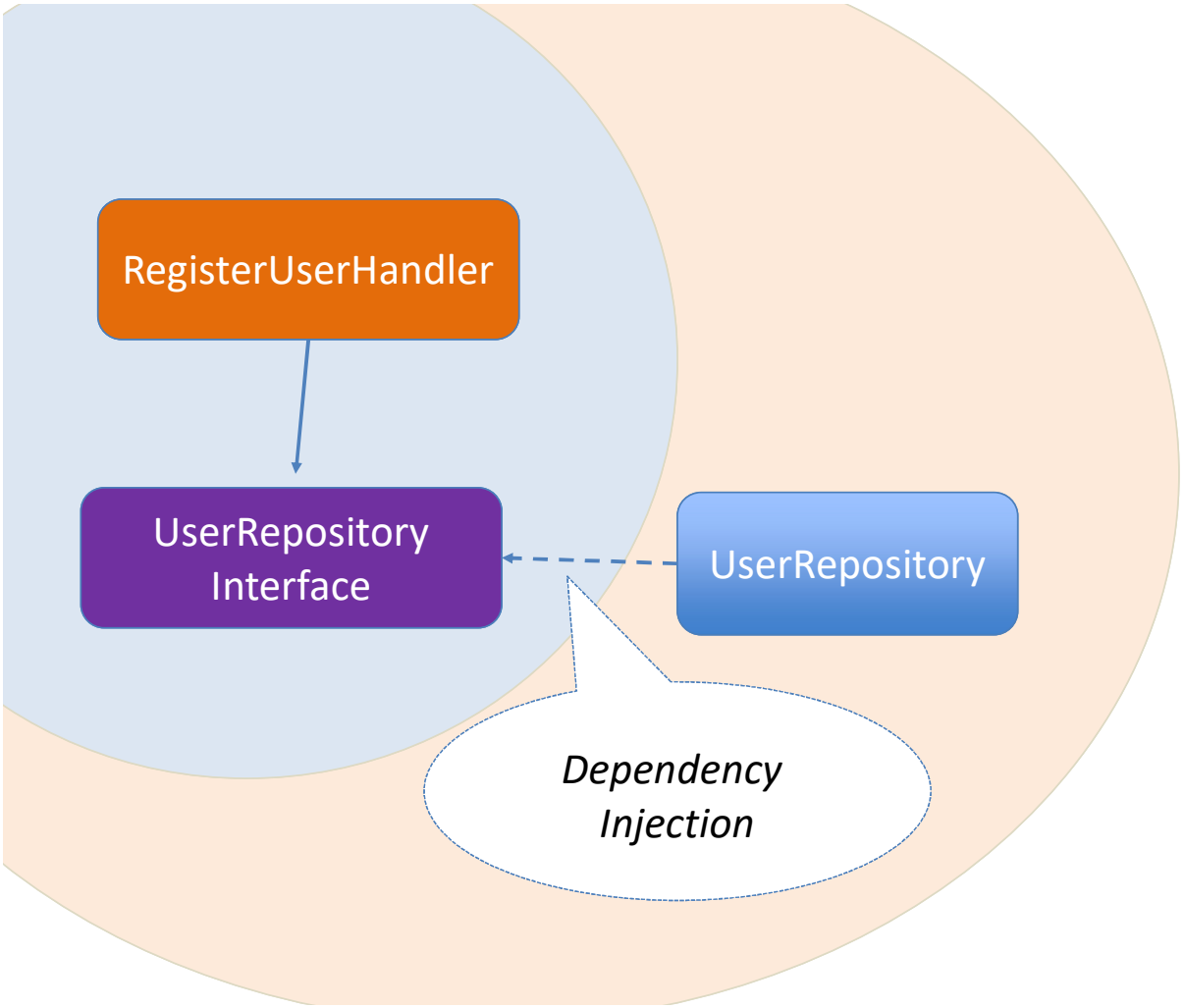
Dependency Rule

"Source code dependencies can only point inwards.

Nothing in an inner circle can know anything at all about something in an outer circle."






RegisterUserHandler
UserRepository Interface
UserRepository
Dependency Injection