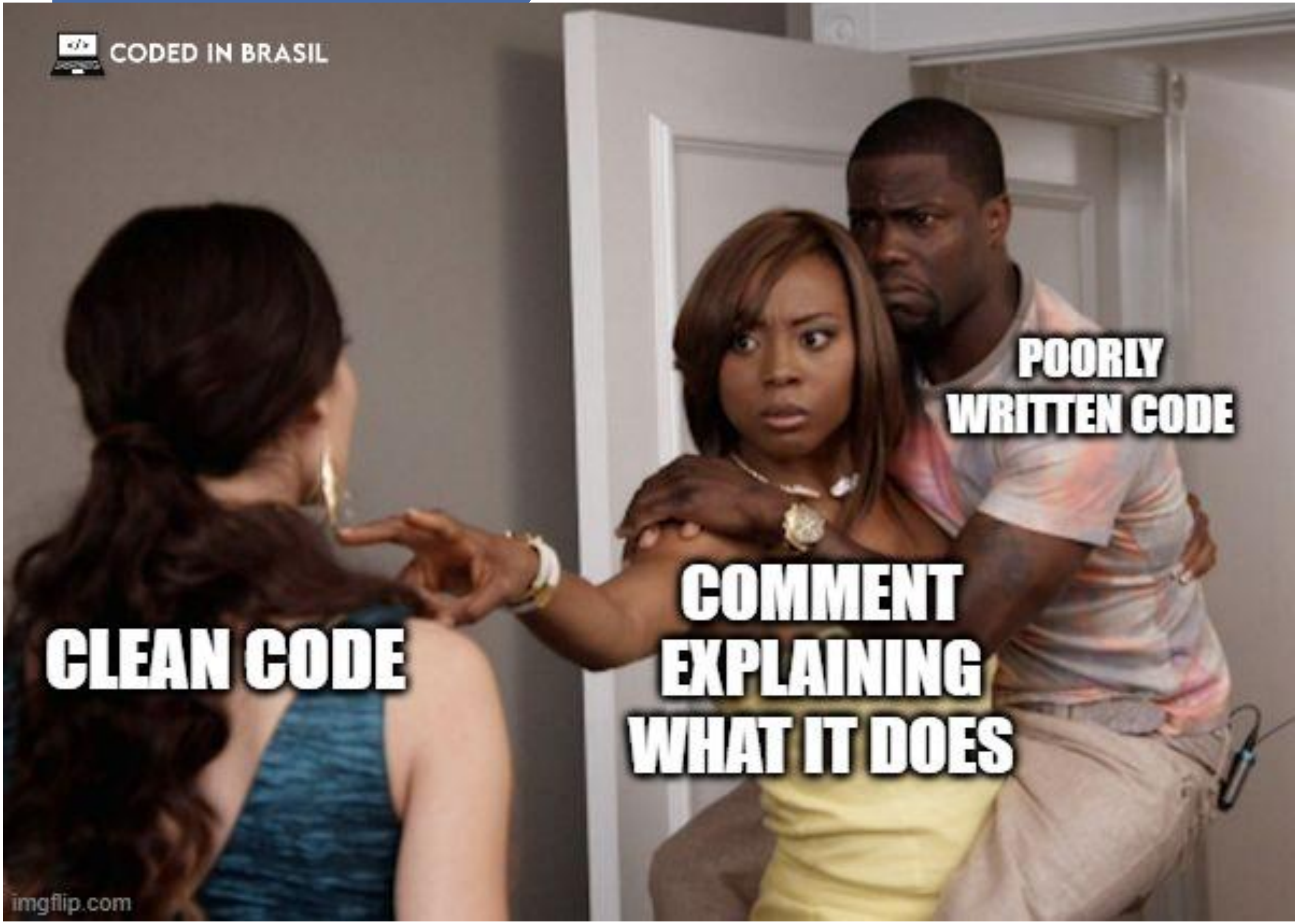
CLEAN architecture

Paolo Burgio
paolo.burgio@unimore.it

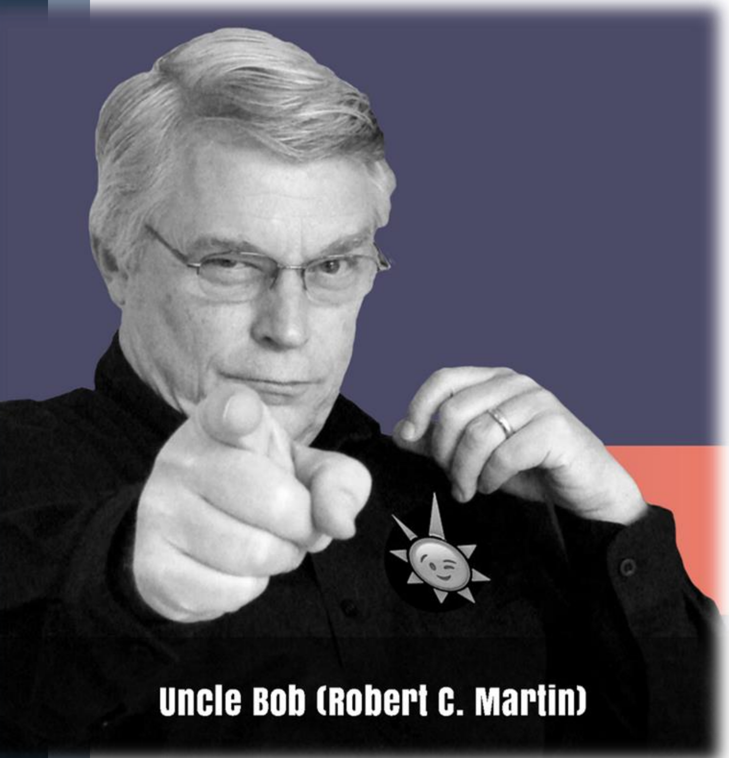
UNIMORE UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA High Performance Real Time Lab


CODED IN BRASIL
POORLY WRITTEN CODE
COMMENT EXPLAINING WHAT IT DOES
CLEAN CODE
imgflip.com

What is it?

A code architectural pattern

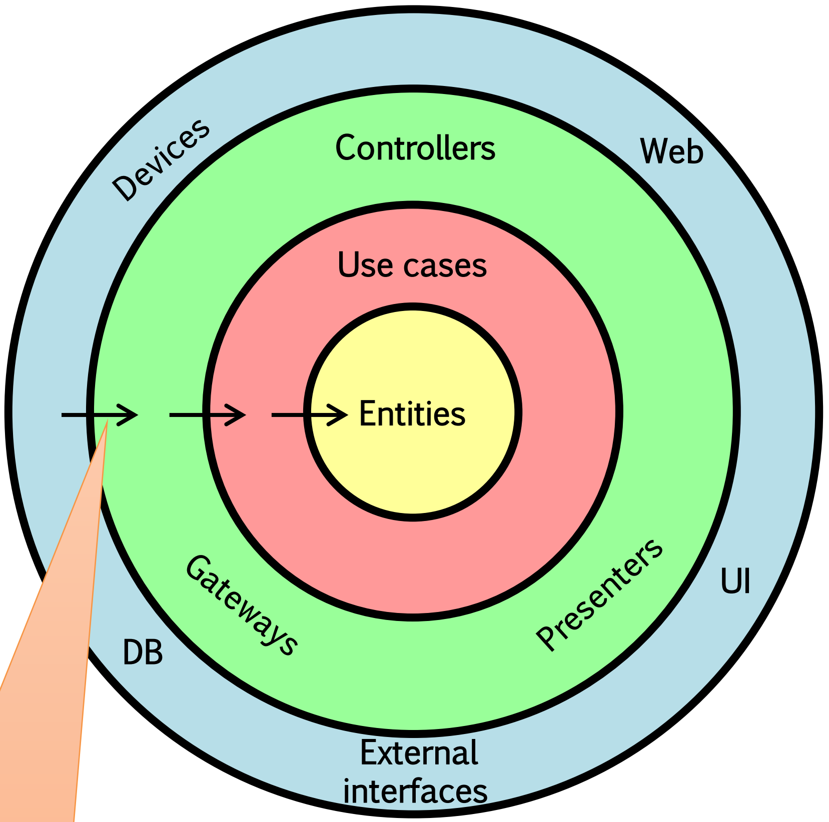
- A structure that enables building software that is more scalable, testable, maintainable
- Built upon/heavily relies on good coding practices (e.g., SOLID, design patterns..)
- Disclaimer: +15-20% dev time overhead



- Formalized by “Uncle Bob”
- Started his blog in 2011
- Adopted by nearly all mid- and large-scale projects

As simple as this

- Aka: "Onion Architecture"



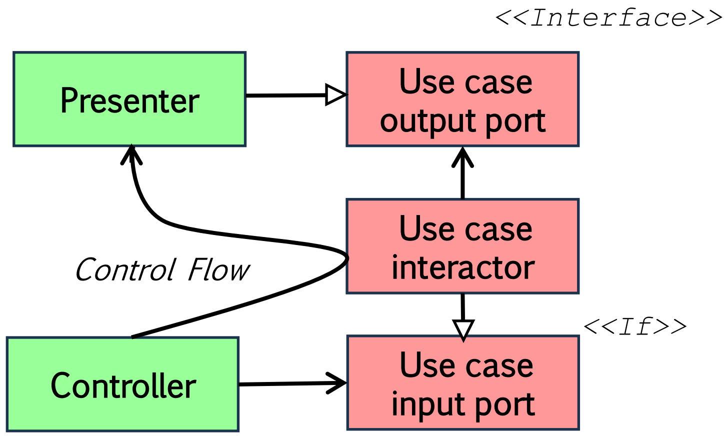
Enterprise business rule

Application business rule

Interface Adapters

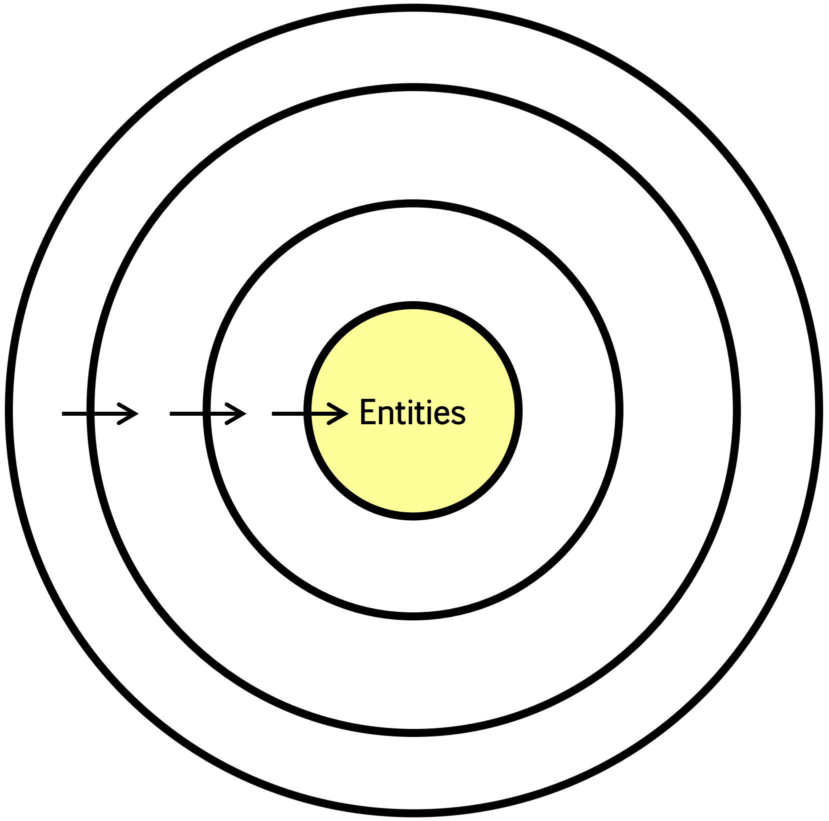
Frameworks & Drivers

Dependencies go from "out" to "in"



The Model

- Our view of the world: just field, and basic operations (get, set..)



Enterprise business rule

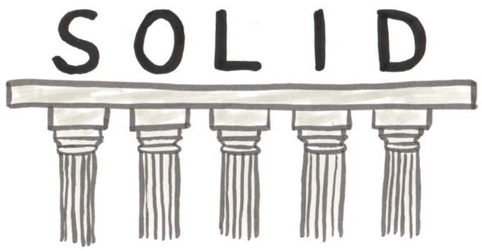
- Everything depends on them/includes them, they do not depend on anything
- Why is this so important?

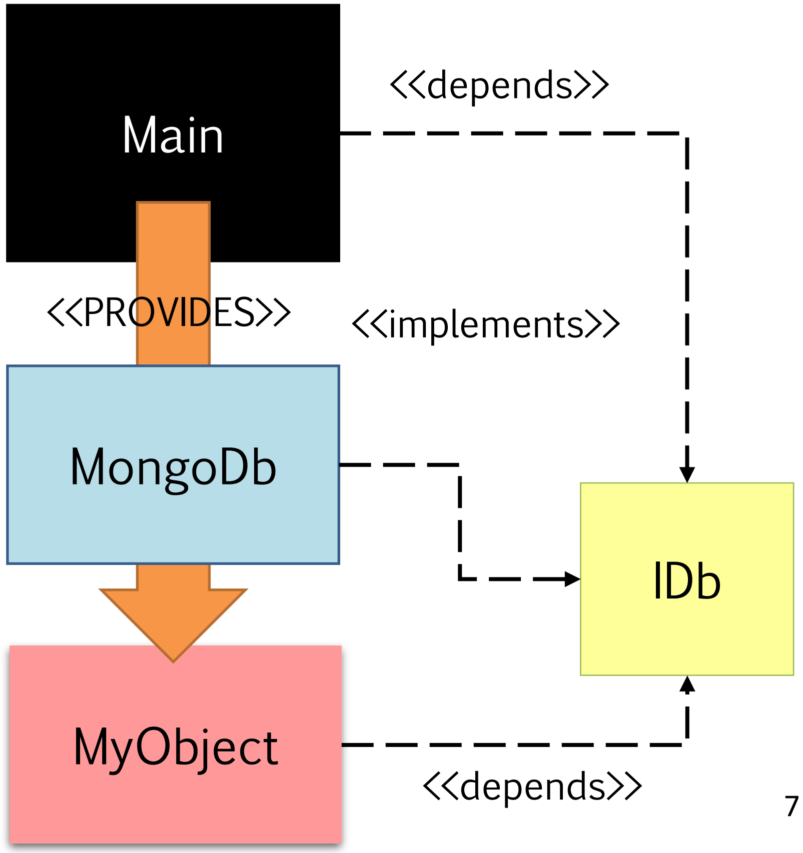
Dependency Inversion

- Reduce coupling
  - Avoids unnecessary dependencies that ultimately make the code hard to modify
- Enables fast testing and debugging
- Wraps functionalities (Interface Segregation)

(Only one issue)

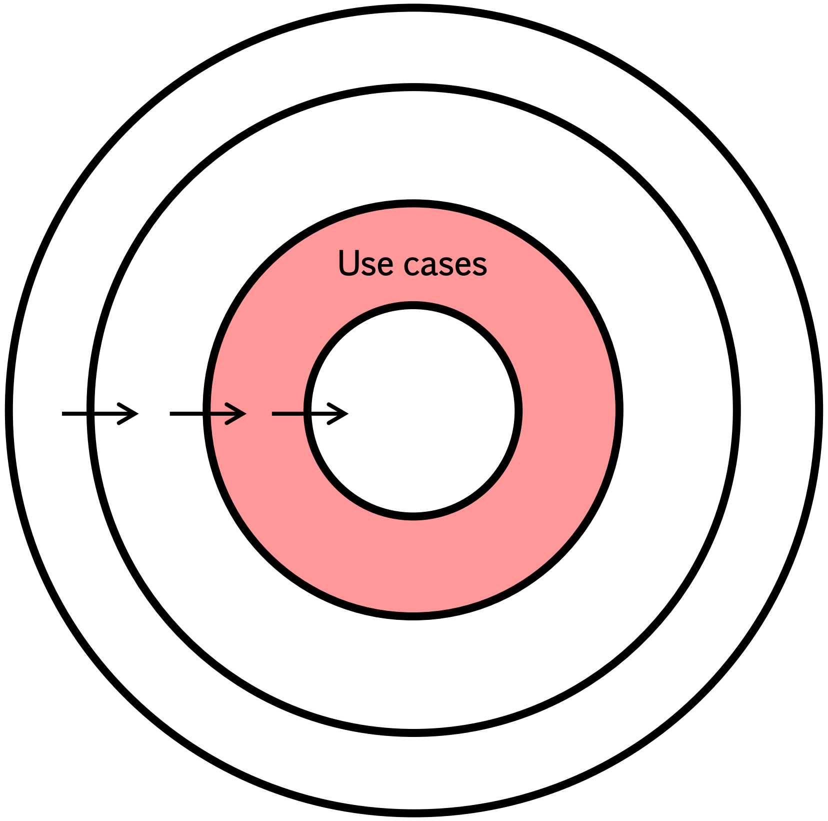
- You need to find a (elegant) way to provide the required services
- Dependency Injection!





Straight from requirements

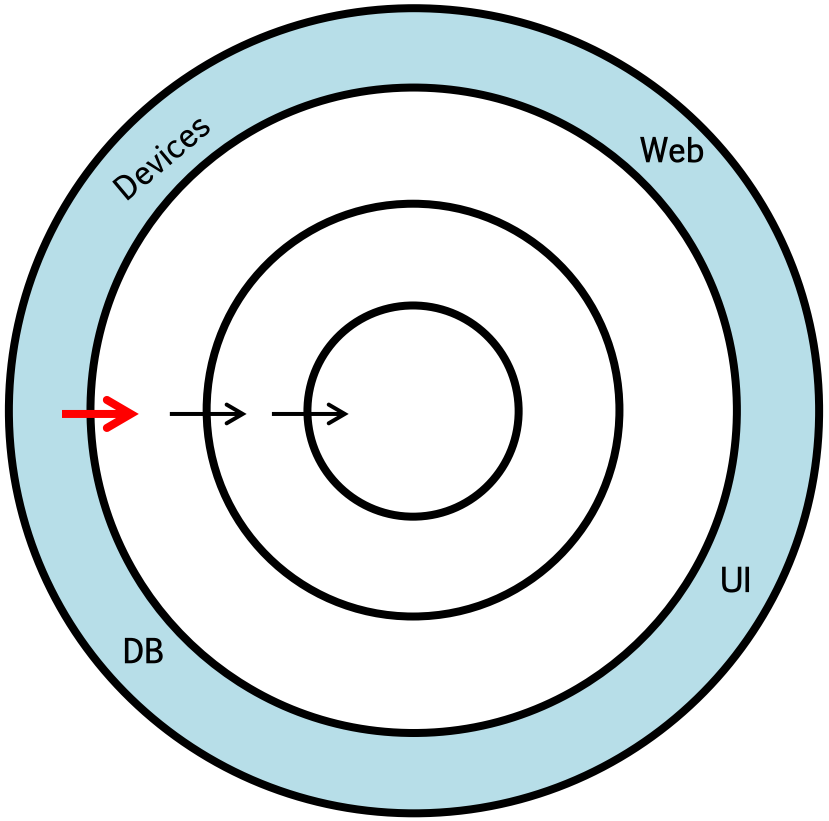
› Application specific logics: functionalities



Application business rule

“The bad world”

› This layer represents, and wraps, “external” dependencies, e.g., DTOs, MongoDb…

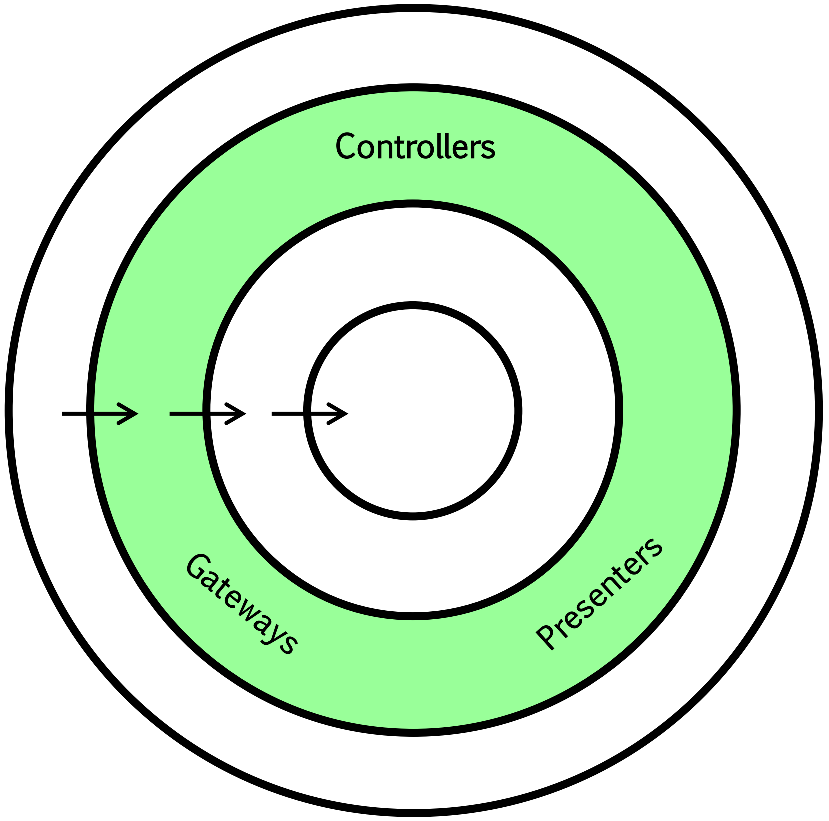


Frameworks & Drivers

› How do we implement the dependency?

Our good old friend

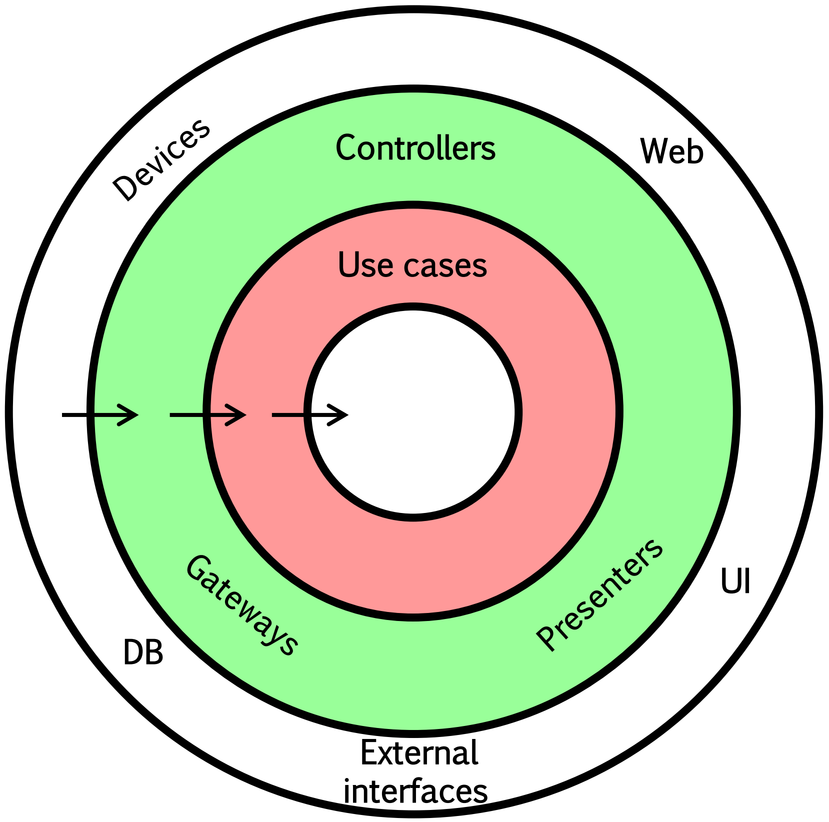
- Aka: “Onion Architecture”



Interface Adapters

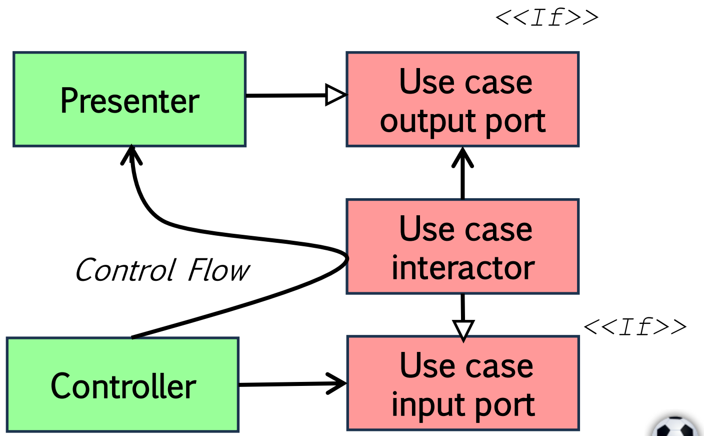
Control flow, and class diagram

› Note how we use Interfaces, and (consequently) Dependency Injection



Application business rule

Interface Adapters



Dependency Injection in dotNet

Example: WebApp

› We build and run the actual program, explicitly, in `Program.cs`

› `WebApplicationBuilder` is the class that performs (Web)Application startup

› It has features to inject services

```
// 'Transient' means that you create a new instance every time
// it is injected
builder.Services.AddTransient<IService, ConcreteImplementation>();

// Scoped' services are created only once for every HTTP request
// we are serving (hence, useful for keeping states within a request
builder.Services.AddScoped<IService, ConcreteImplementation>();

// ...
builder.Services.AddSingleton<IService, ConcreteImplementation>();
```

Exercise (C#)

Let's
code!

Take any "basic" application, and refactor it following the clean architecture

..or...

Refactor the basic example of C# WebApi

```
$ dotnet new webapi --use-controllers [-o MyApi]
```

Use dependency injection with builder.Services.Add in "

```
builder.Services.AddScoped<IService, ConcreteImplementation>();
```

Remember to create a basic UML scheme for its structure, to identify the four layers

› Bonus: check `AutoMapper`

Dependency Injection in Java

Java does not natively support DI

› Use external FWK, such as *Spring* or Google Guice

› Typically, based on annotations

› `@AutoWired` tells Spring to search for a Spring bean that implements the `IWriter` interface and place it automatically into the setter.

```
@Service
public class MySpringBeanWithDependency {
  private IWriter writer;

  @Autowired
  public void setWriter(IWriter writer) {
    this.writer = writer;
  }

  public void run() {
    String s = "This is my test"; writer.writer(s);
  }
}
```

Dependency Injection in Java

- `@Service` tells Spring this is something that implements business logic, and we can inject it

```
// public interface IWriter {
// void writer(String s);
// }

@Service
public class MyWriter implements IWriter {
  @Override
  public void writer (String s){
    System.out.println("The string is " + s);
  }
}
```

Dependency Injection in Java

› Also `MySpringBeanWithDependency` implements `@Service` ...of course

```java
@Service
public class MySpringBeanWithDependency {
  private IWriter writer;

  @Autowired
  public void setWriter(IWriter writer) {
    this.writer = writer;
  }

  public void run() {
    String s = "This is my test"; writer.writer(s);
  }
}
```

Spring annotations

Basically, every class you saw before was a Java Bean

- You could use the “generic” `@Bean` annotation
- Used for Classpath Scanning
- In C# it's called Reflection, but it's basically the same principle

We can even be more precise, specifying

- `@Component`, a generic Spring-managed component.
- `@Service`, which we saw, annotates classes at the business logic/services layer
- `@Repository` annotates classes at the persistence layer, i.e., (database) repositories

Exercise (Java)

Let's code!

- Take the basic WebAPi example

  ...or...

- Take any application (the simpler, the better)

- ...and refactor it following CLEAN architecture

References

Course website

- http://hipert.unimore.it/people/paolob/pub/ProgSW/index.html

Uncle Bob

- https://blog.cleancoder.com/uncle-bob/2011/11/22/Clean-Architecture.html

My contacts

- paolo.burgio@unimore.it
- http://hipert.mat.unimore.it/people/paolob/