SPRING IO

#springio19

Clean Architecture with Spring

Tom Hombergs
@TomHombergs

Set status

Tom Hombergs

thombergs

Edit profile

Software Engineer & Open Source Enthusiast

@adessoAG

Germany

tom.hombergs@gmail.com

https://reflectoring.io

Organizations

adesso

Overview | Repositories 73 | Projects 0 | Stars 122 | Followers 204 | Following 32

Pinned

Order updated.

Customize your pins

code-examples

A collection of code examples from blog posts etc.

Java ★ 165 168

adessoAG/wicked-charts

Beautiful and interactive javascript charts for Java-based web applications.

JavaScript ★ 64 45

docx-stamper

Easy-to-use template engine for creating docx documents in Java.

Java ★ 73 34

diffparser

Parse unified diffs with Java.

Java ★ 38 20

adessoAG/budgeteer

Simple Project Budget Monitoring via Web Browser

JavaScript ★ 26 17

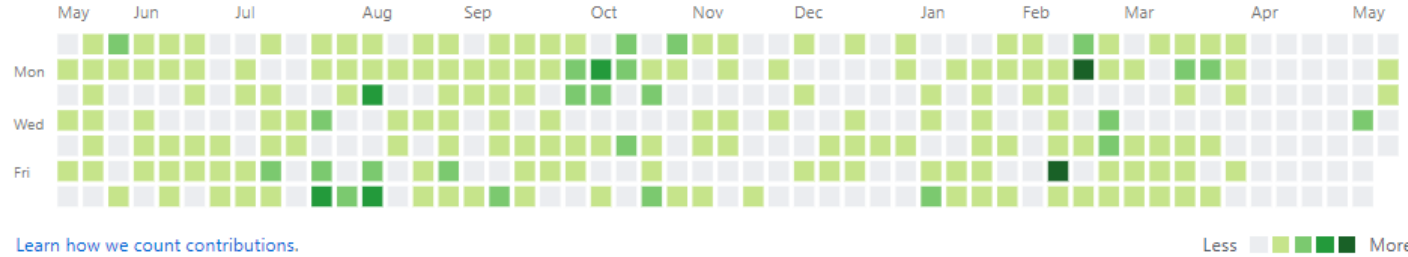
reflectoring/coderadar

Continuous code analysis server.

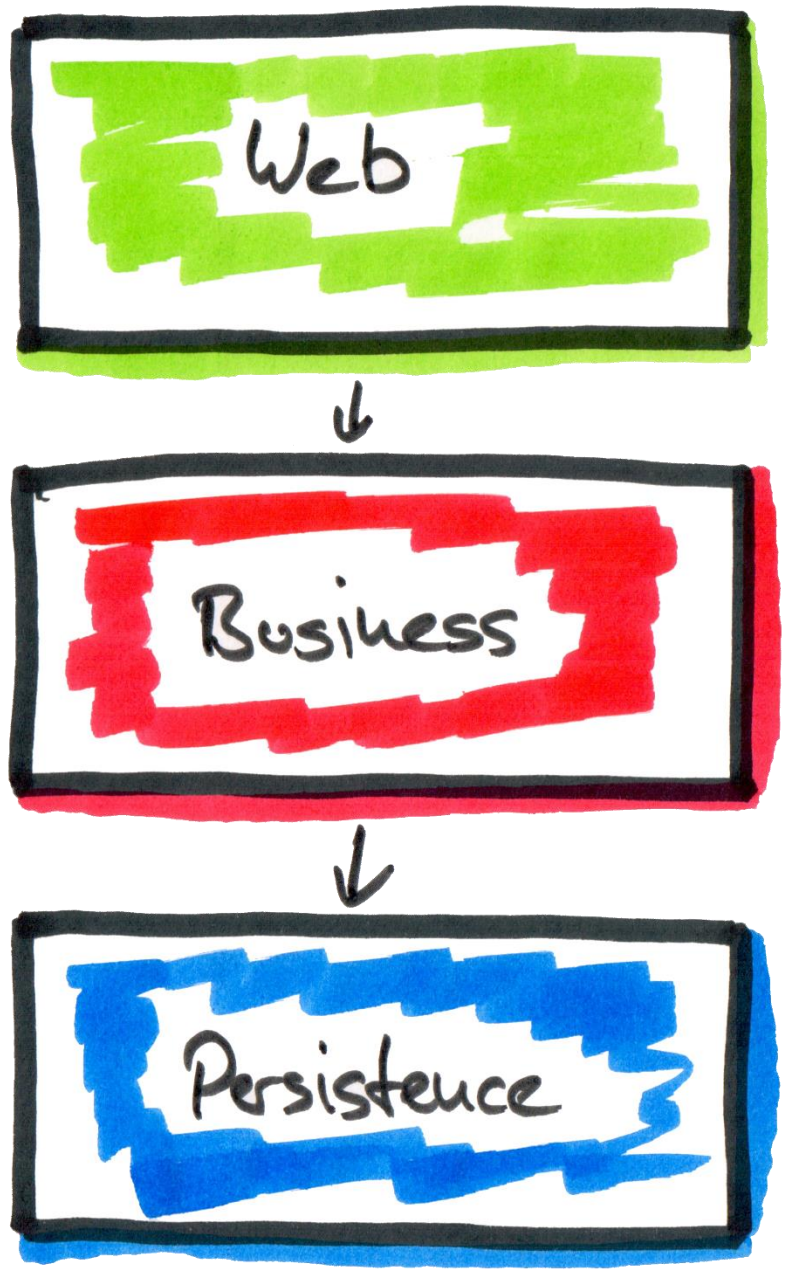
Java ★ 29 14

761 contributions in the last year

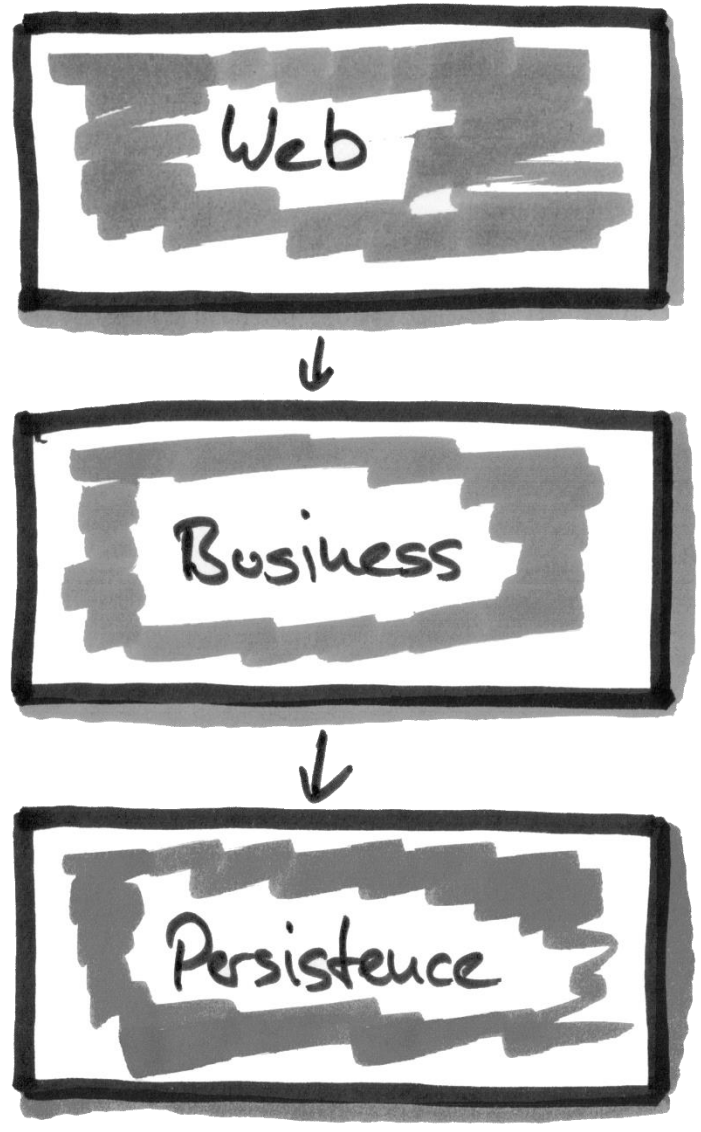
Contribution settings



Layers



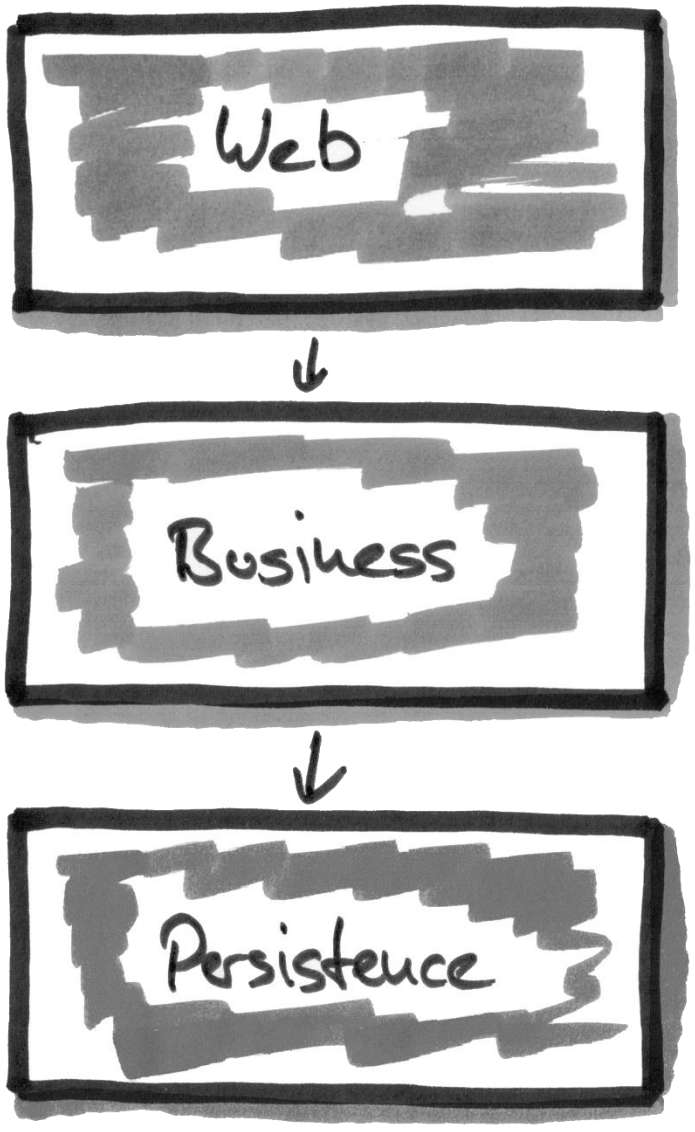
Why bother with Architecture?



Architecture Goals

Architecture Goals

The goal of software architecture is to minimize the lifetime cost of the software

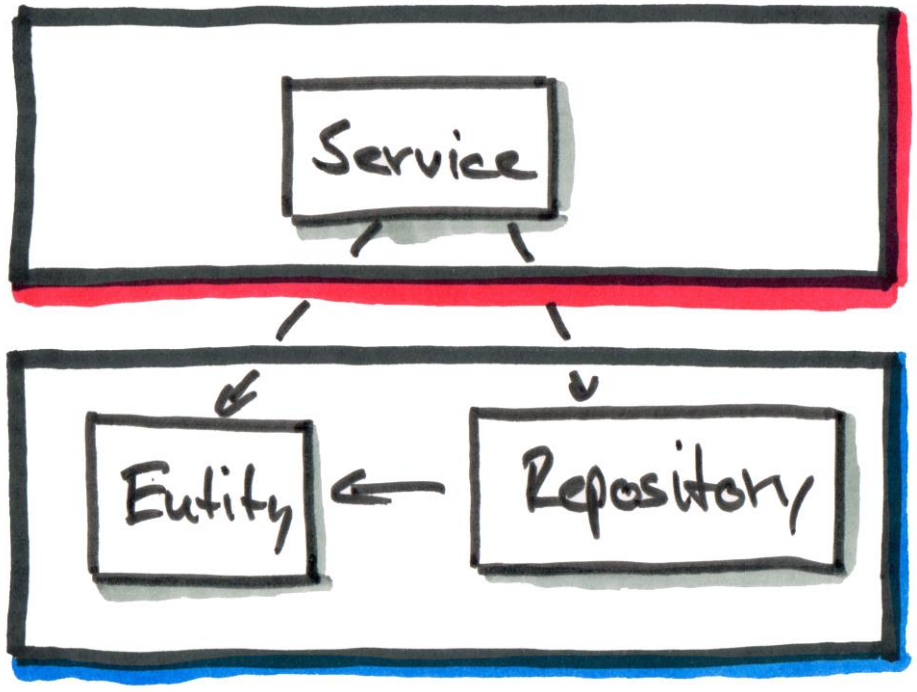


So …
What‘s Wrong With Layers?

Layers are a solid architecture pattern!

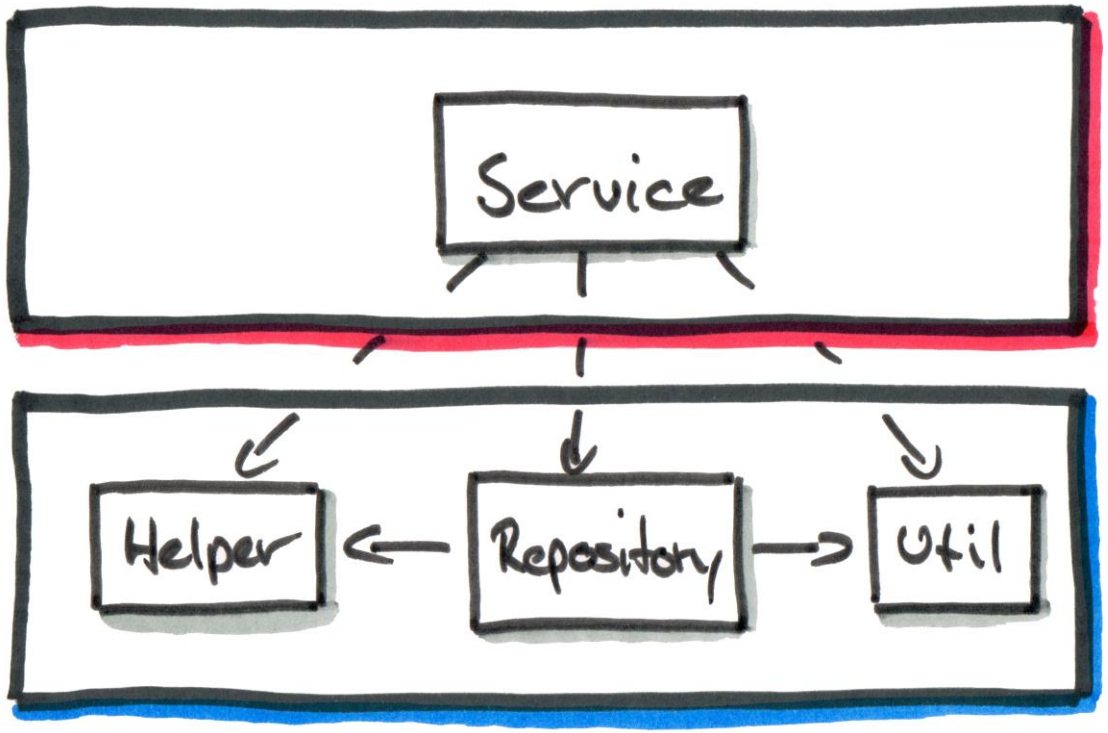
But without additional restrictions they are prone to design flaws

Database-Driven Design



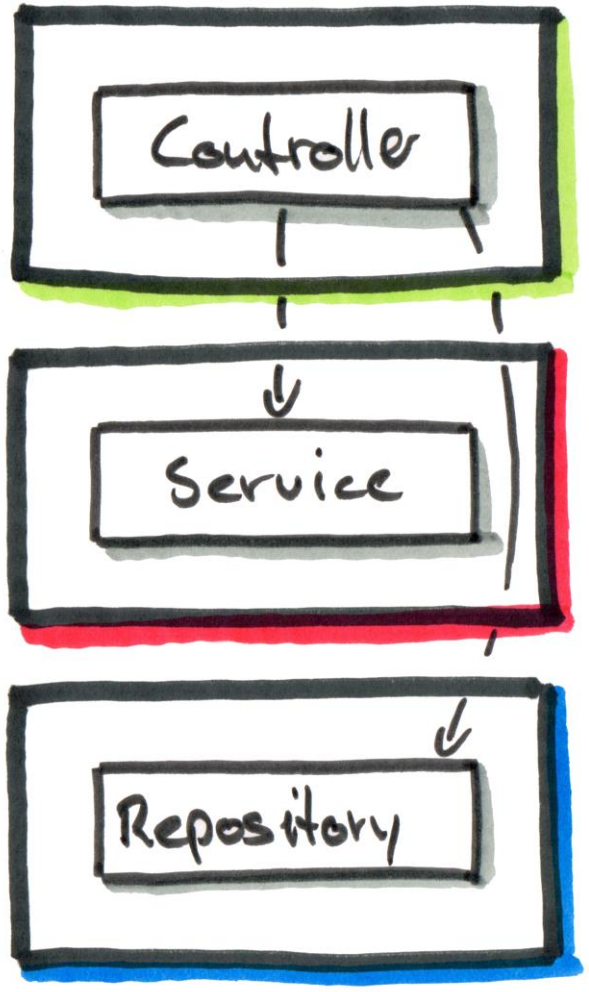
hard to change things due to coupling to the database

Blurred Boundaries



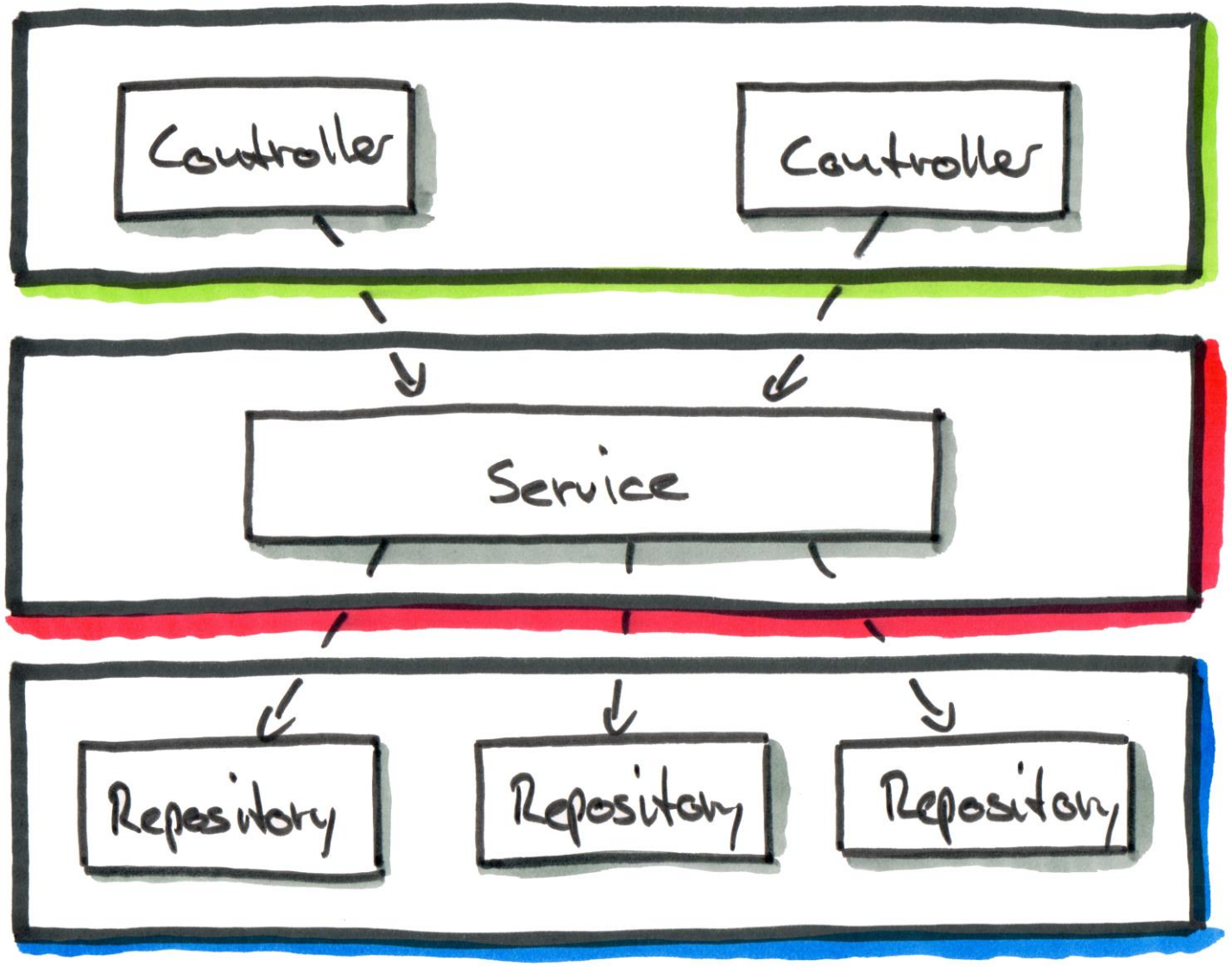
easily blurred boundaries

Hard-to-test Shortcuts



hard to test due to many dependencies

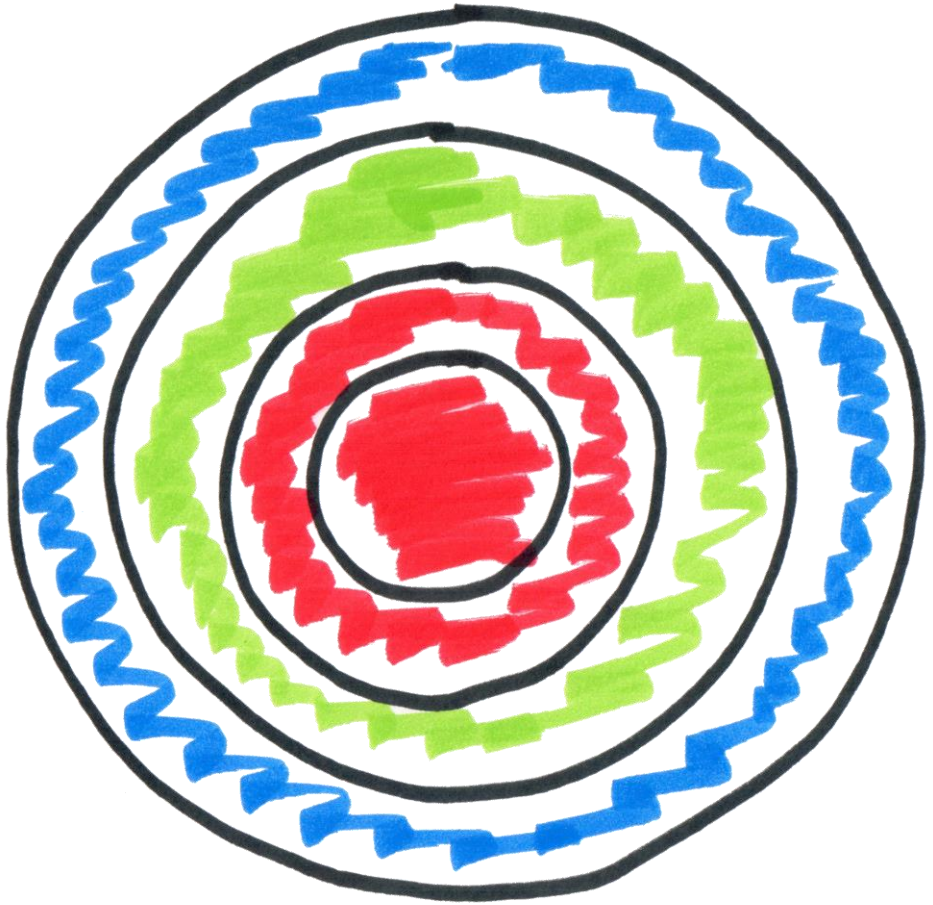
Hidden Functionality



hard to find the use cases in the code

hard to parallelize work on the same feature

Do Circles Instead!



Single Responsibility Principle

Open-Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

Single Responsibility Principle

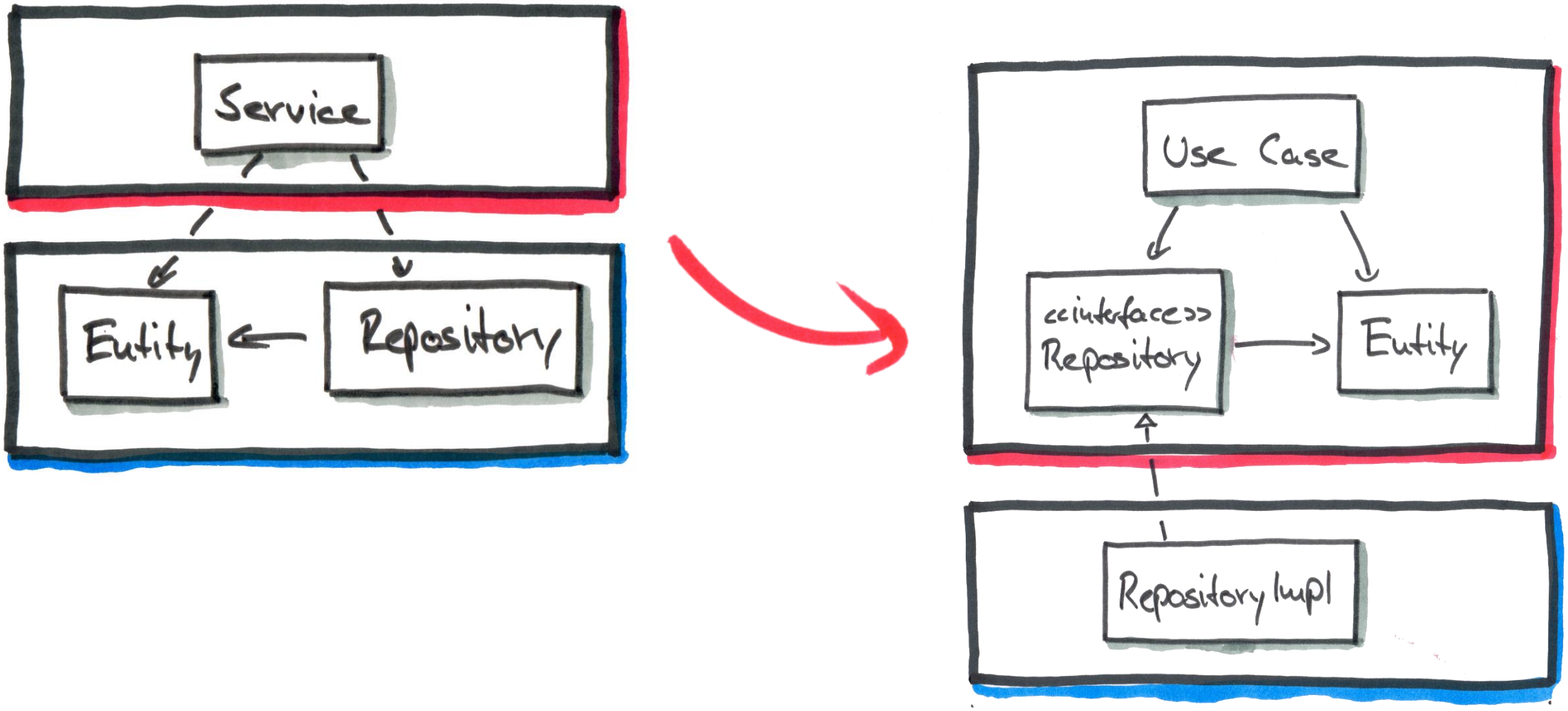
Open-Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

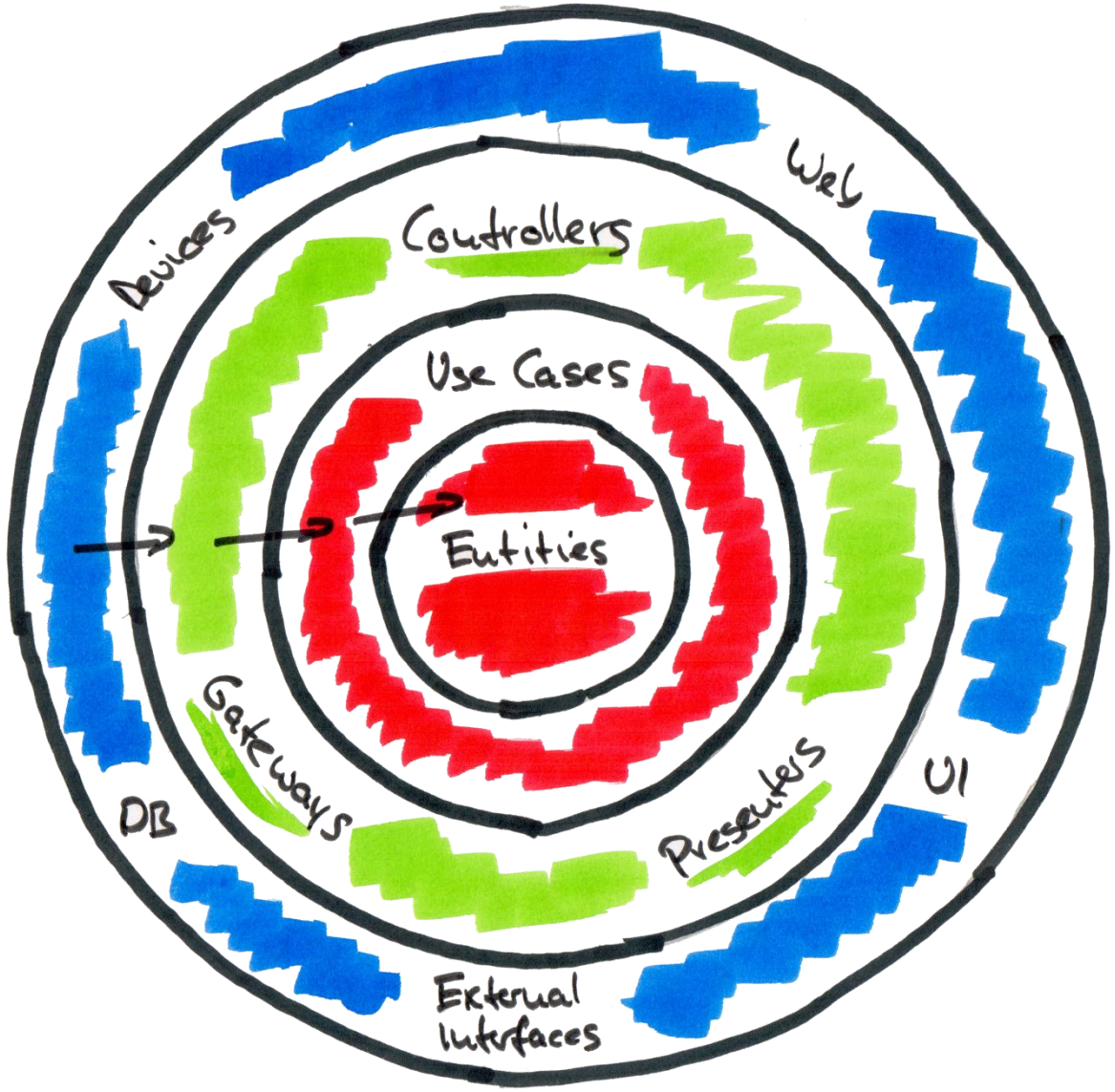
Dependency Inversion Principle



We can choose the direction of any code dependency*

* As long as we have control over the code

Clean Architecture (Robert C. Martin)



Dependency Rule

The DB is a Detail!

Domain Logic is the Core!

The Web is a Detail!

Single Responsibility Principle

Open-Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

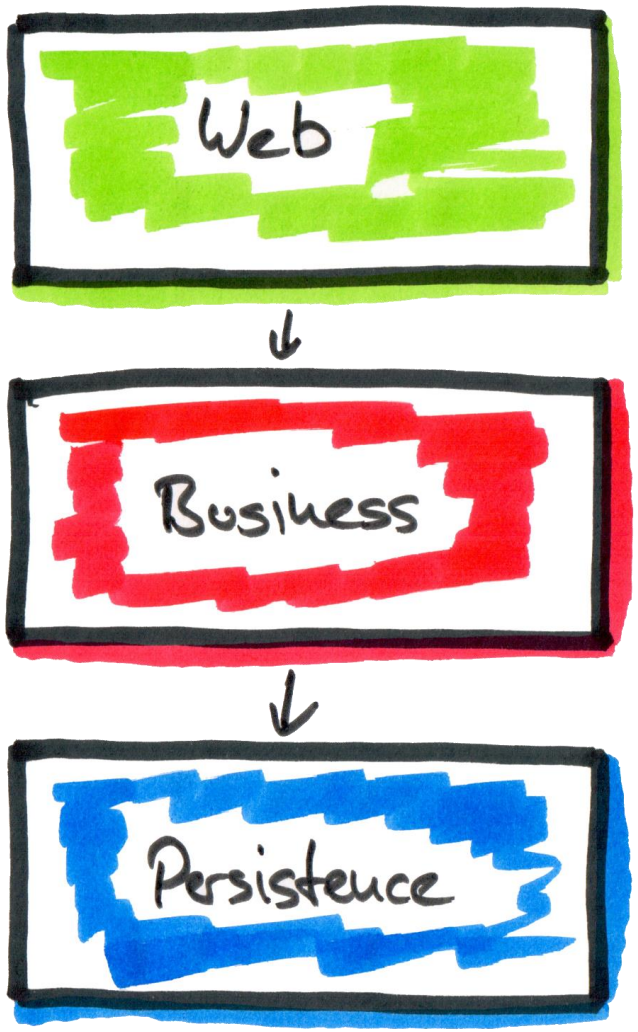
Dependency Inversion Principle

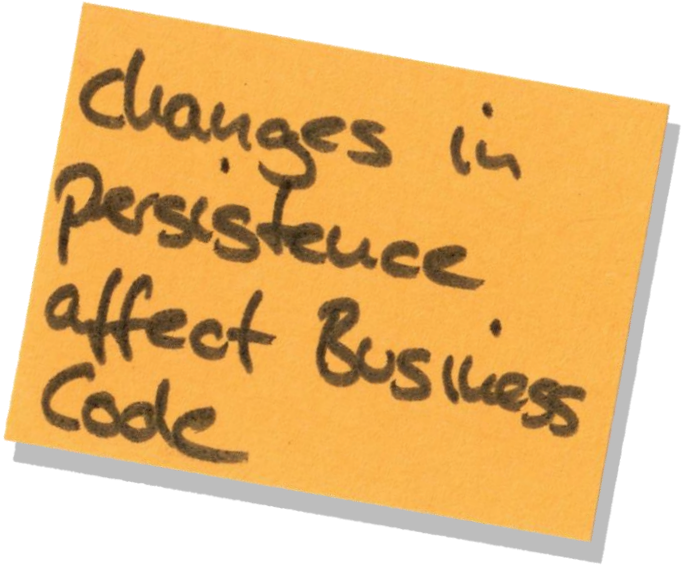
Single Responsibility Principle

A module* should have only one reason to change

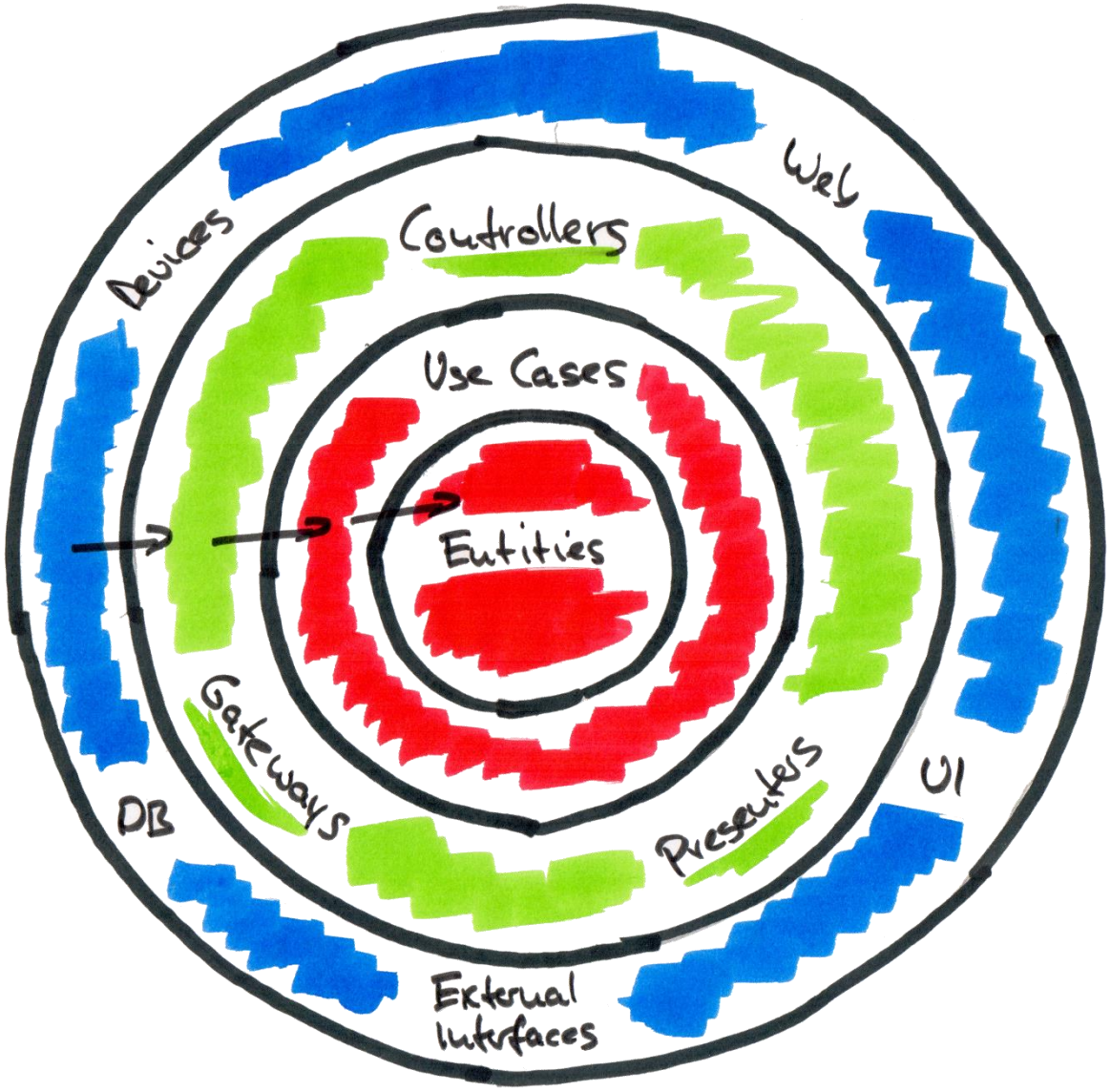
* Read: class, package, component, architecture element, software entity, …

Single Responsibility Principle – Macro Level





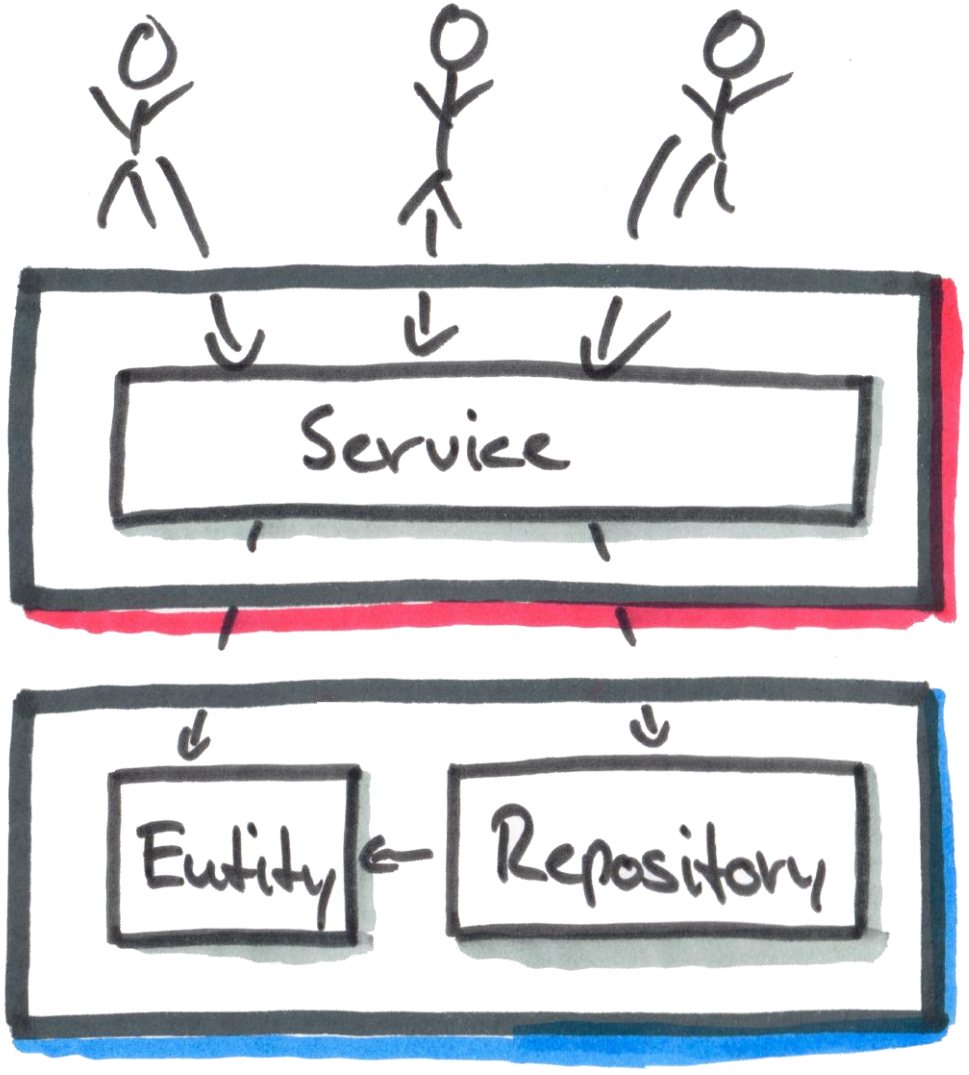
Single Responsibility Principle – Macro Level



Business Code is isolated from changes

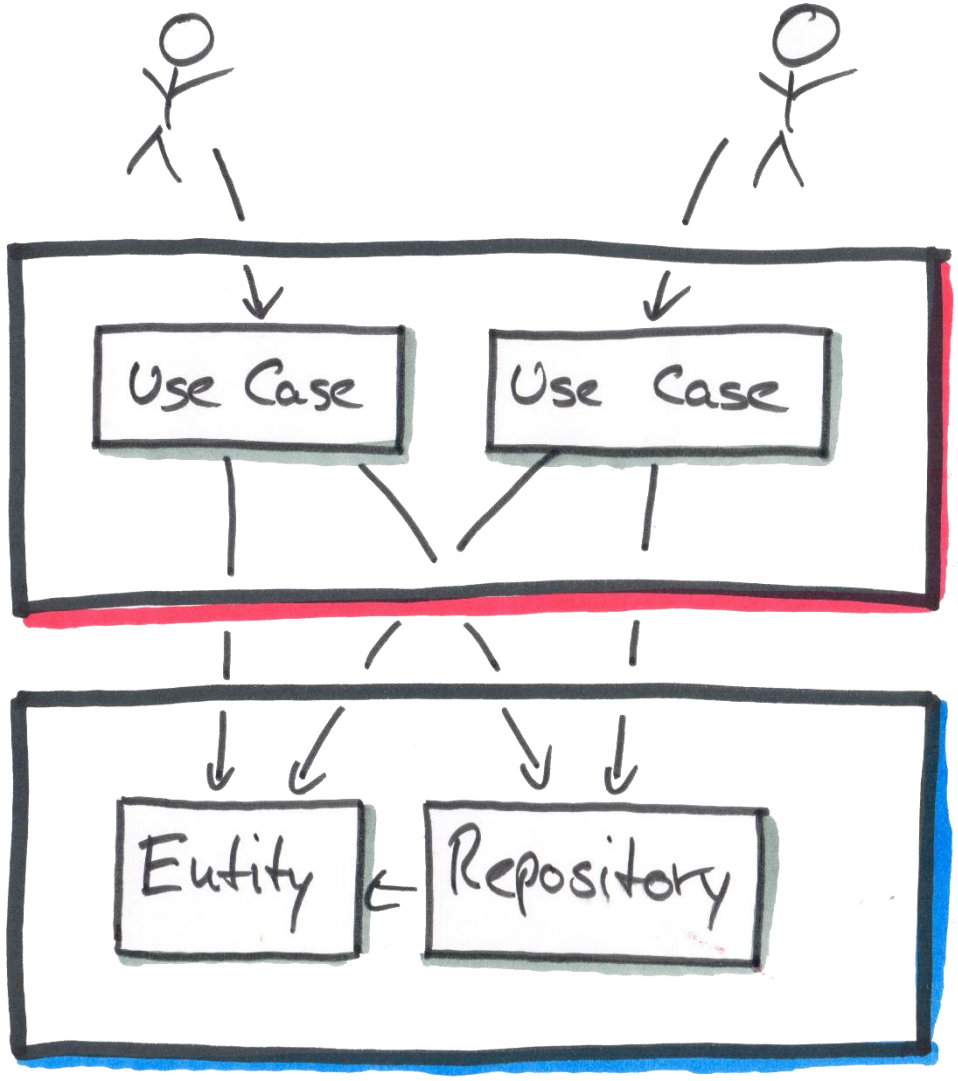
Only a Domain-Centric Architecture allows Domain-Driven Design

Single Responsibility Principle – Micro Level

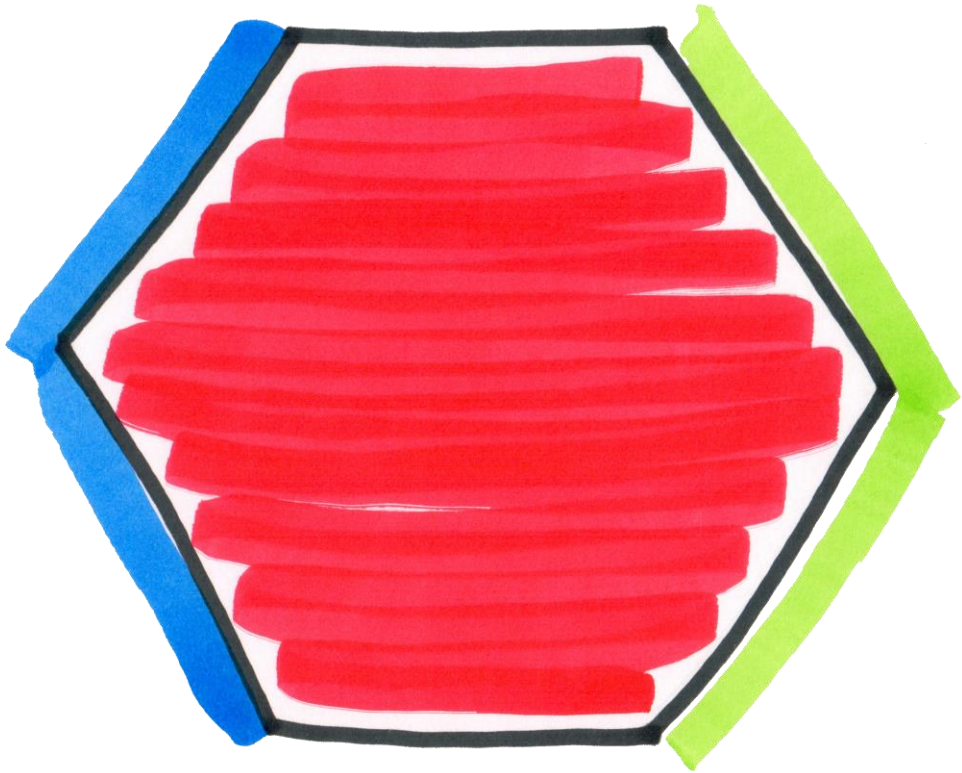


Single Responsibility Principle – Micro Level

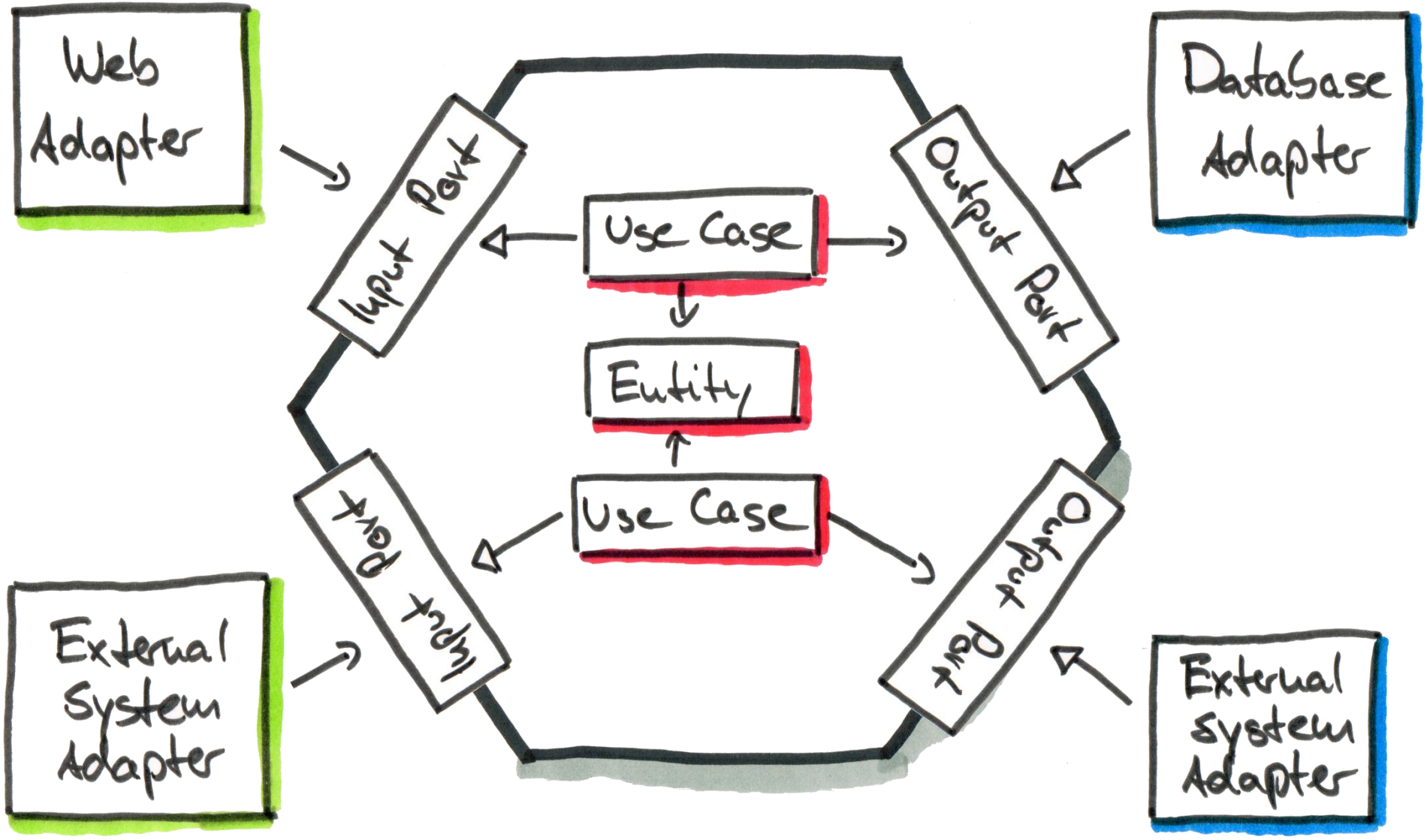
Service is responsible for one actor



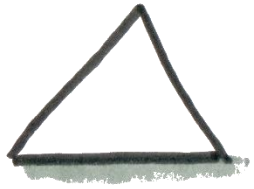
Do Hexagons Instead!



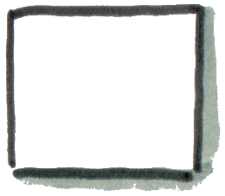
Hexagonal Architecture / Ports & Adapters (Alistair Cockburn)



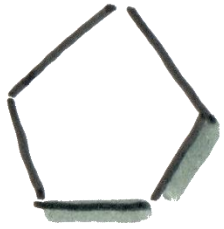
Choose your Polygonal Style



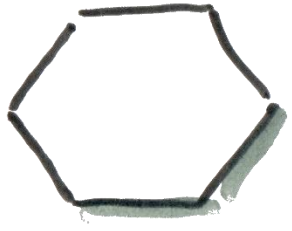
Triangle



Quadrilateral

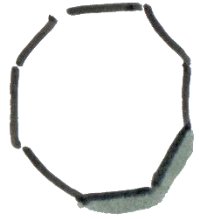


Pentagon



Hexagon

Heptagon

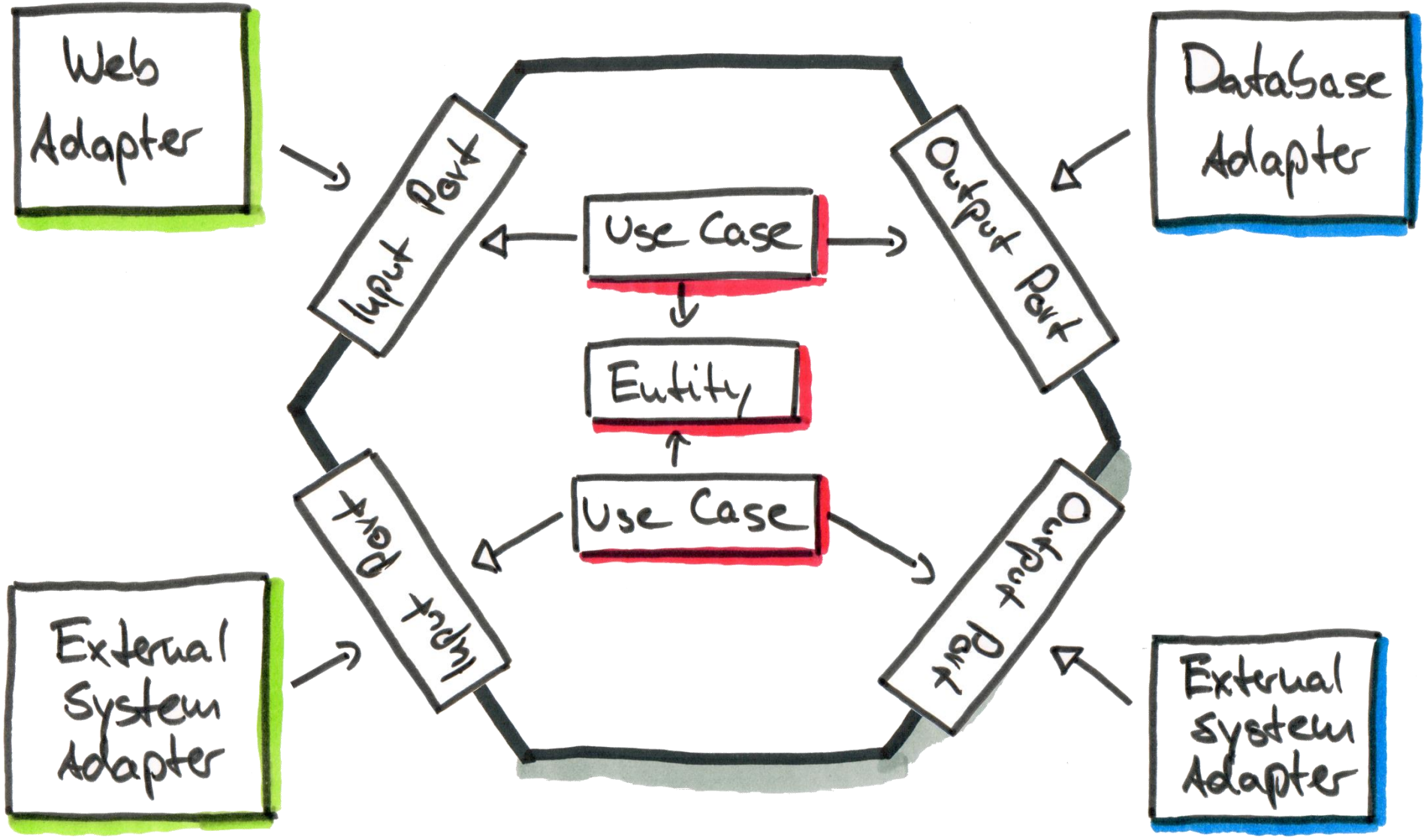


Octagon

Nonagon

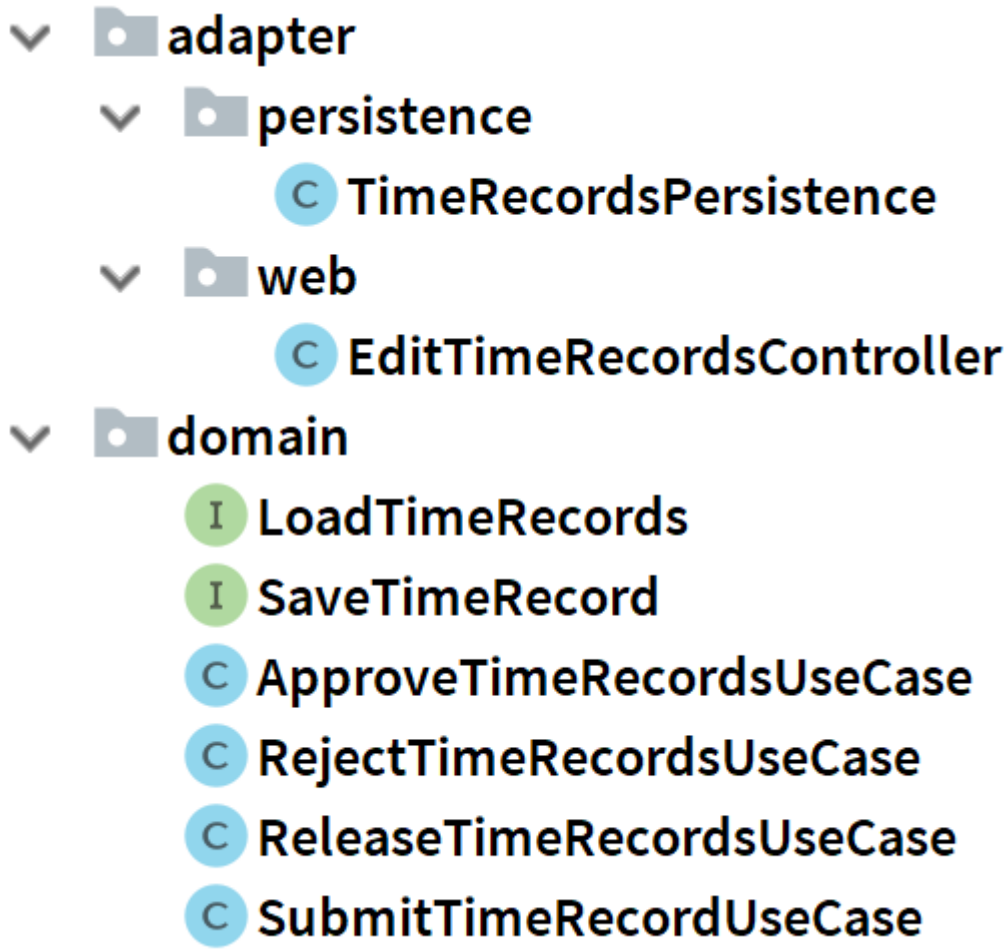
Decagon

Use Cases with a Single Responsibility in a Hexagonal Architecture

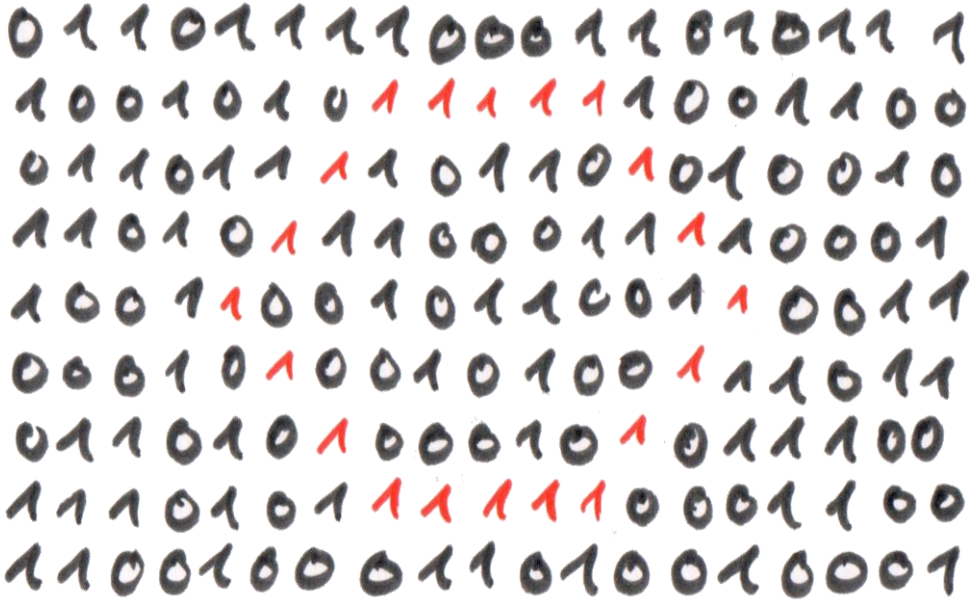


Quiz: What does this Application do?

Quiz: What does this Application do?

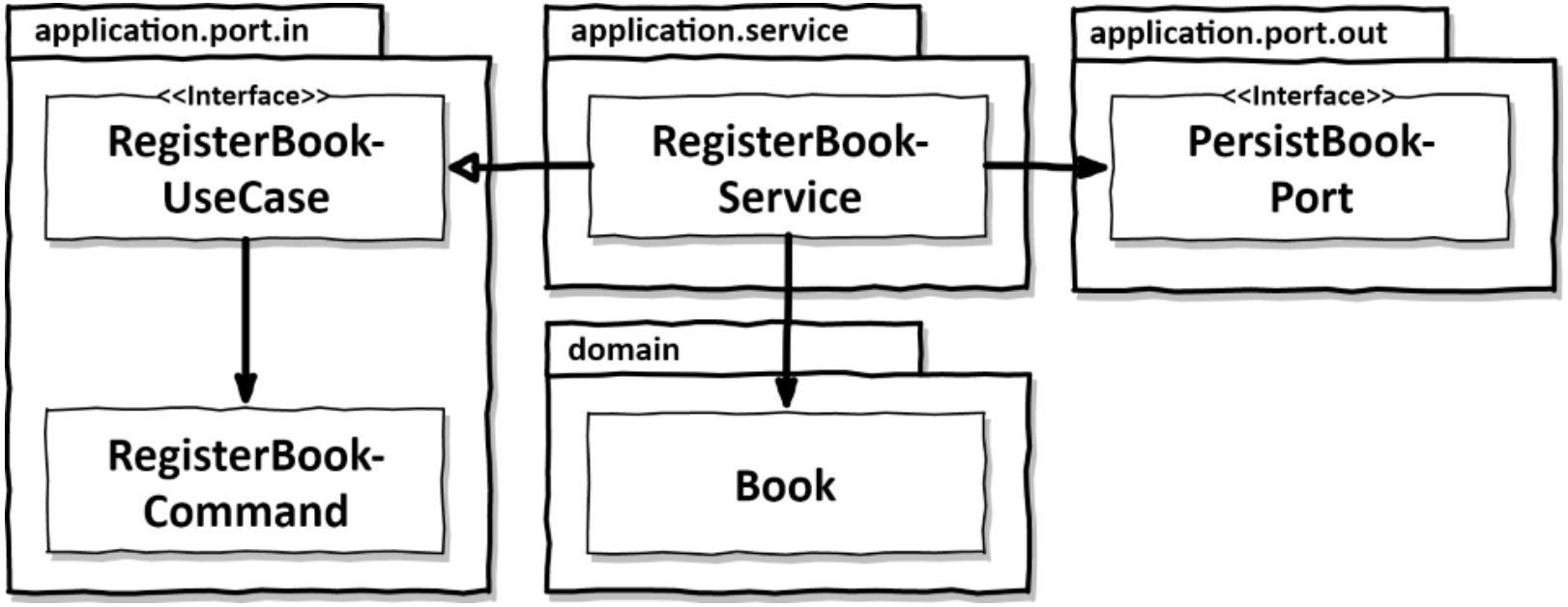


Let‘s Get Our Hands Dirty!



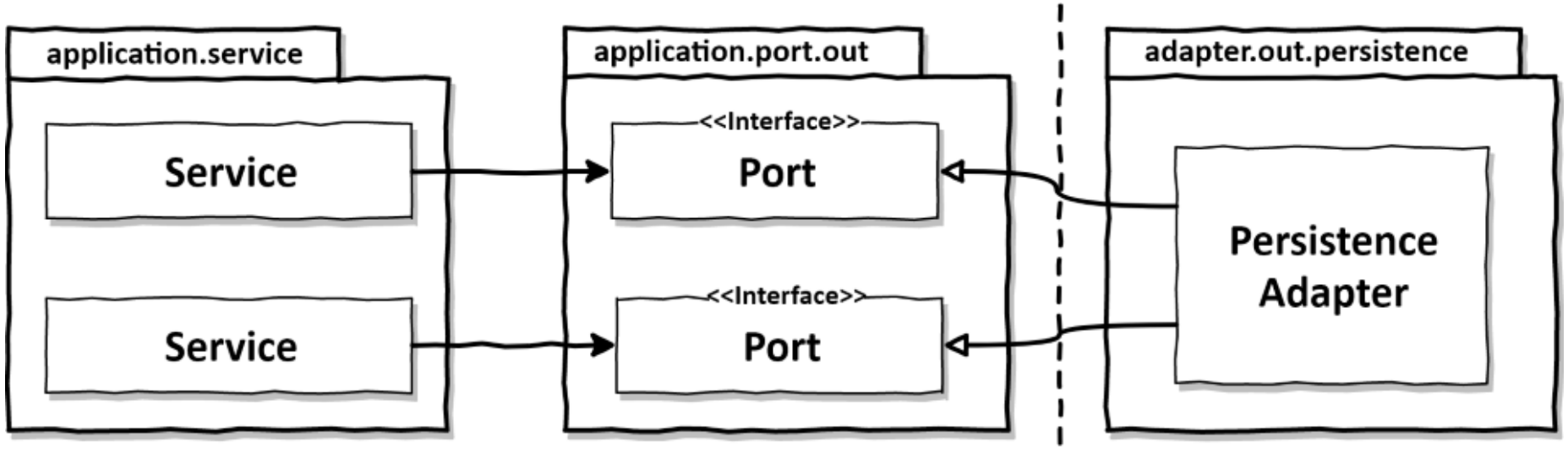
Code Example: https://github.com/thombergs/clean-architecture-example

Implementing a Use Case



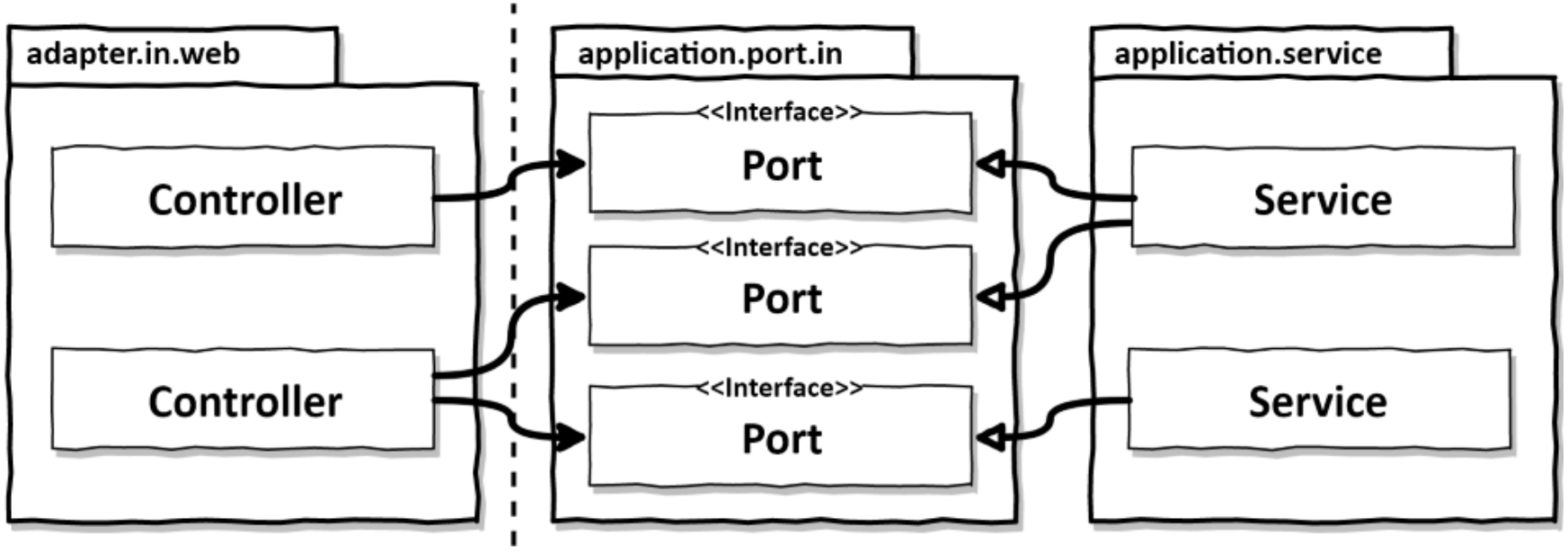
Code Example: https://github.com/thombergs/clean-architecture-example

Implementing a Persistence Adapter with Spring Boot



Code Example: https://github.com/thombergs/clean-architecture-example

Implementing a Web Adapter with Spring Boot



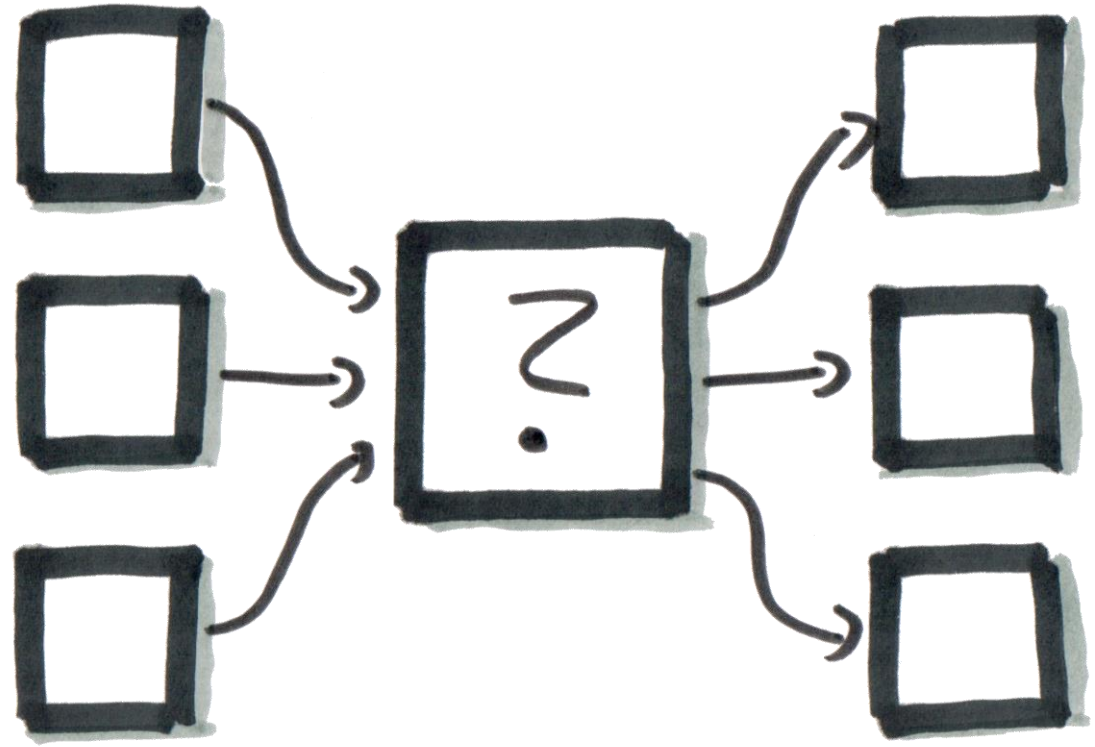
Code Example: https://github.com/thombergs/clean-architecture-example

Spring doesn't care about which architecture we're implementing

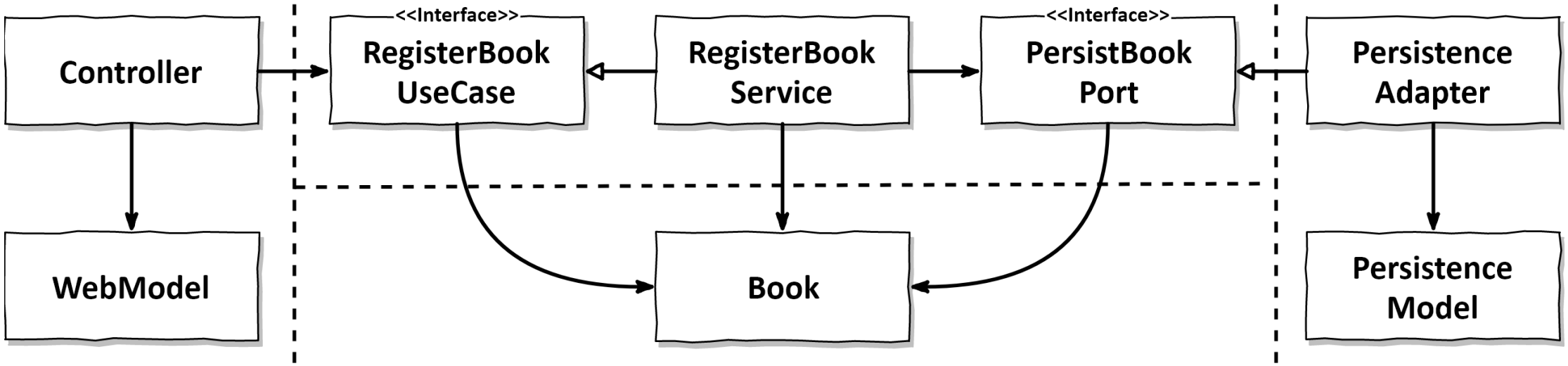
Spring helps to test isolated parts of our architecture

Spring is easily held at arm's length

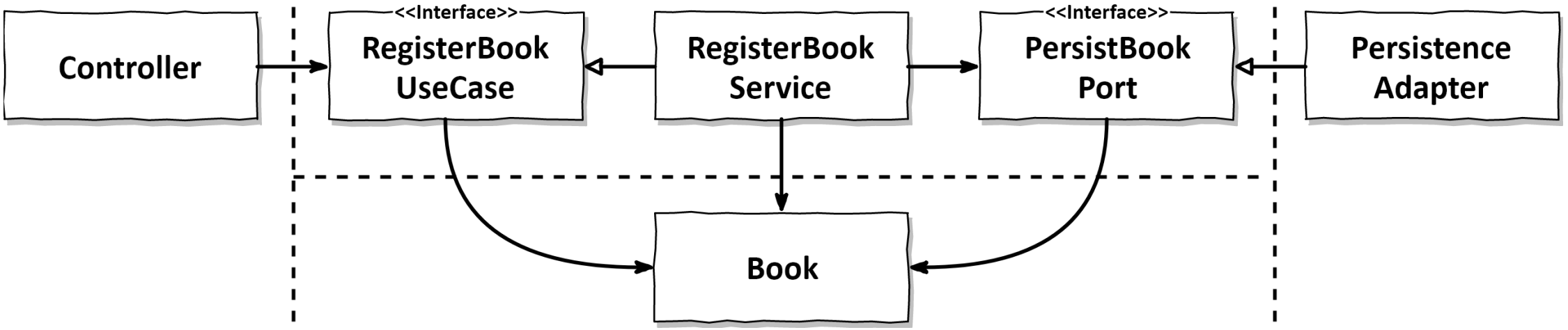
Mapping Strategies



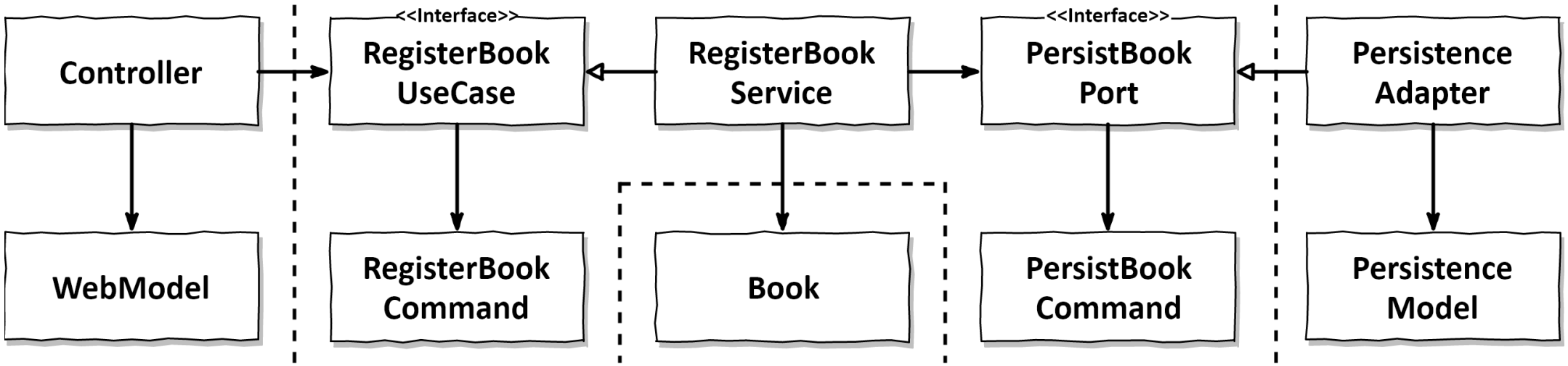
Mapping Strategies – Two Way Mapping



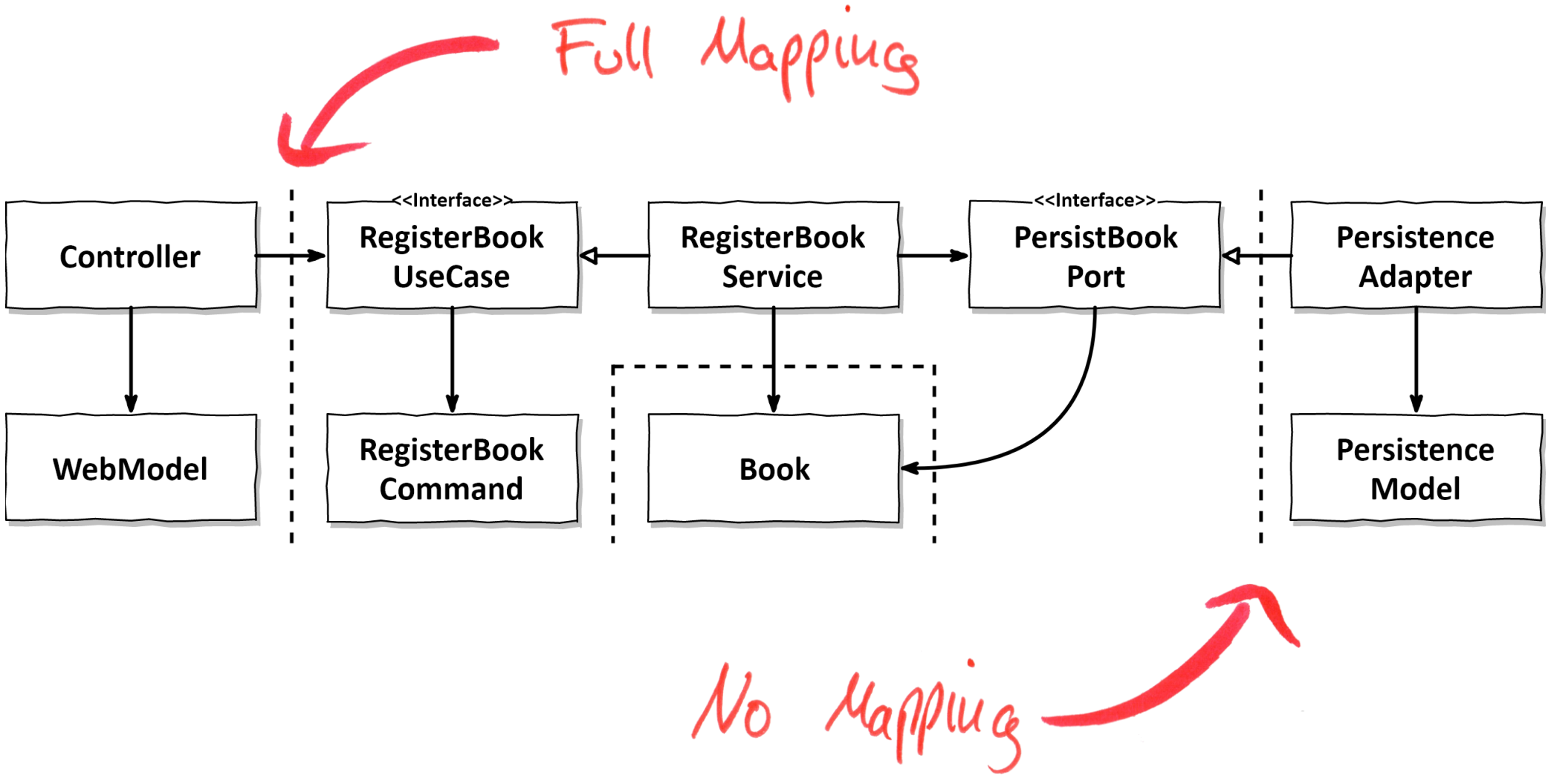
Mapping Strategies – No Mapping



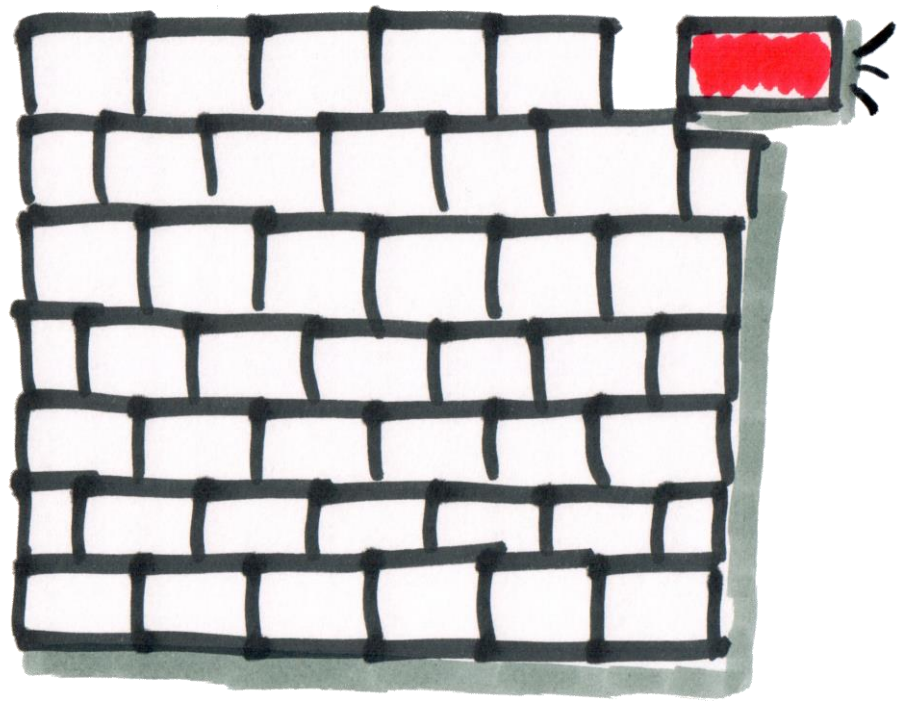
Mapping Strategies – Full Mapping (Command Pattern per Use Case)



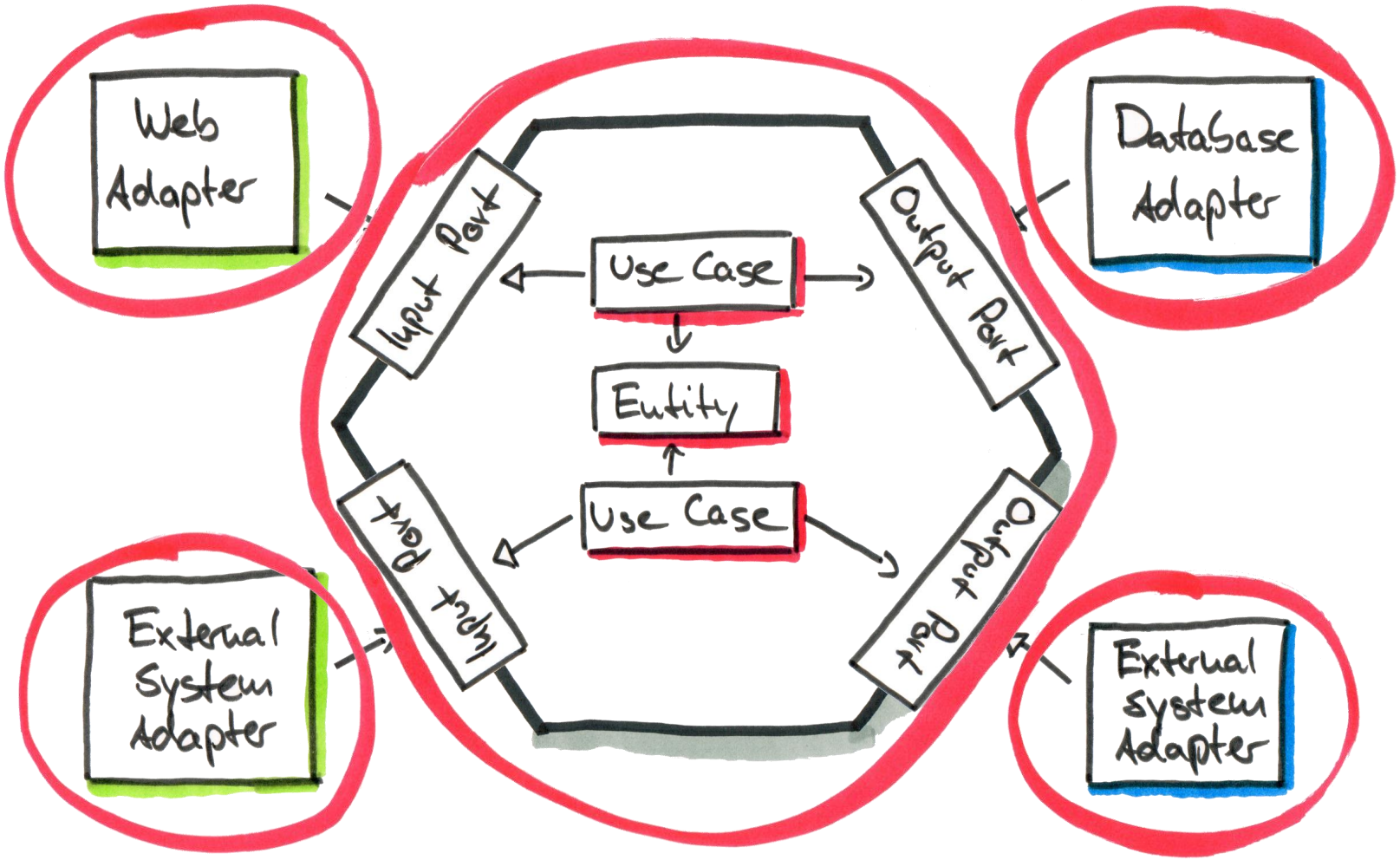
Mapping Strategies – Mix & Match!



Enforcing the Architecture



Enforcing the Architecture - Separate Build Artifacts



Enforcing the Architecture - ArchUnit

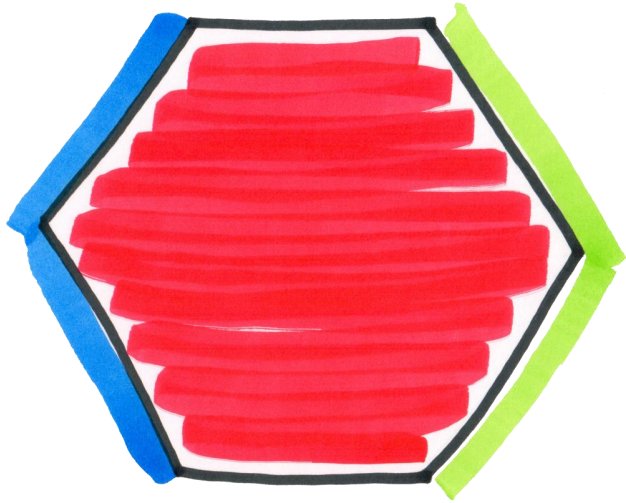
```
@Test
void validateRegistrationContextArchitecture() {
        HexagonalArchitecture
                .boundedContext("io.reflectoring.copyeditor.registration")
                .withDomain("domain")
                .withAdapters("adapter")
                        .incoming("in.web")
                        .outgoing("out.persistence")
                        .and()
                .withApplicationLayer("application")
                        .services("book")
                        .services("invitation")
                        .incomingPorts("port.in")
                        .outgoingPorts("port.out")
                        .and()
                .withConfiguration("configuration")
                .check(allClasses());
}
```

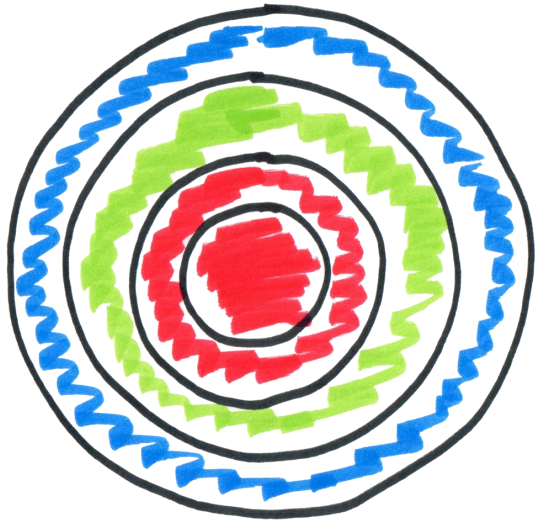
https://github.com/odrotbohm/moduliths

Yes, it will probably work …

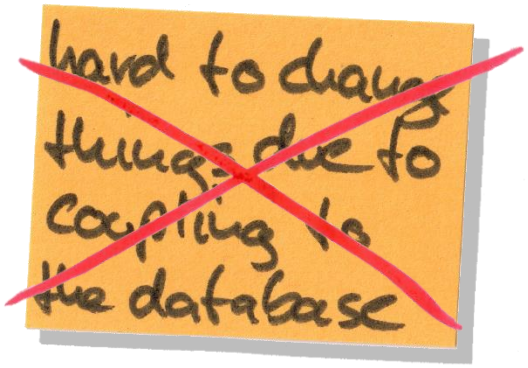
… but I'm used to having 5 years of time to learn each new Java version.

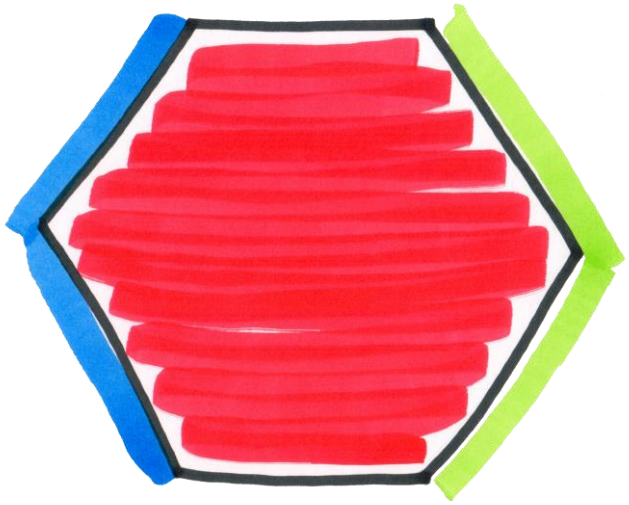
Conclusion

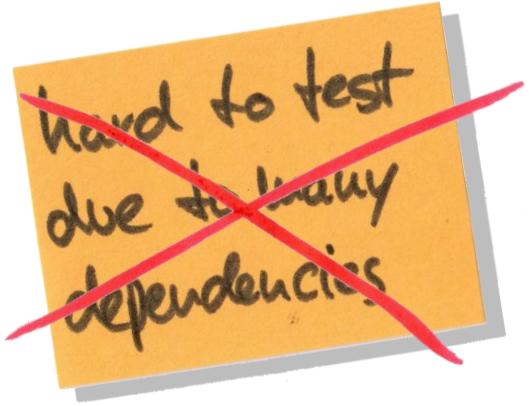


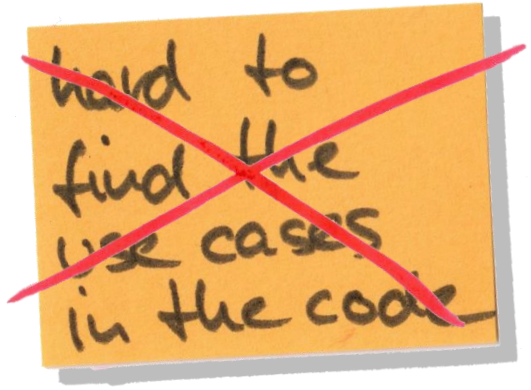


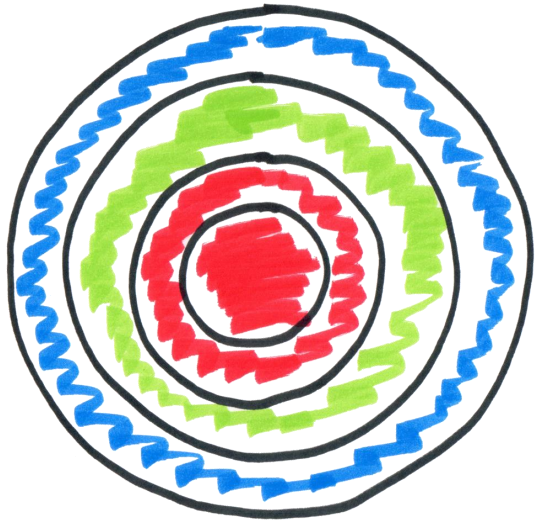
Benefits of a Domain-Centric Architecture





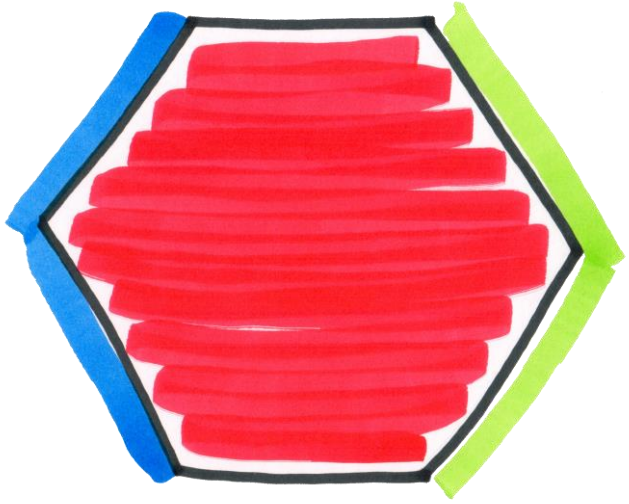






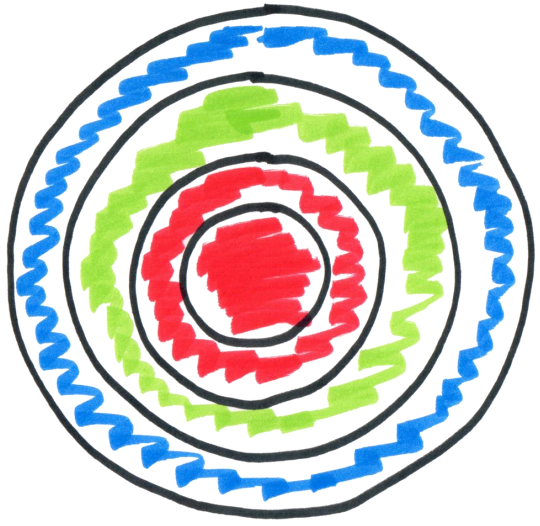
Drawbacks of a Domain-Centric Architecture

a lot of mapping



needs "real" business logic

a lot of perceived redundancy



SPRING IO

#springio19

Thanks for your Patience!

Tom Hombergs
@TomHombergs

Grab the e-book for free ...

https://geni.us/dirty-hands-springio

BCN IO 2019