betsson group

Clean Architecture Essentials

Ivan Paulovich

Betsson Dev'Talk #4
Stockholm - 23rd May 2019

@ivanpaulovich

https://paulovich.net

Ivan Paulovich

Software craftsman and GitHub addicted.
Interested in TDD, SOLID and
Clean Architecture.

betsson group

MVP Microsoft® Most Valuable Professional Reconnect

@ivanpaulovich

The “Software Architecture”

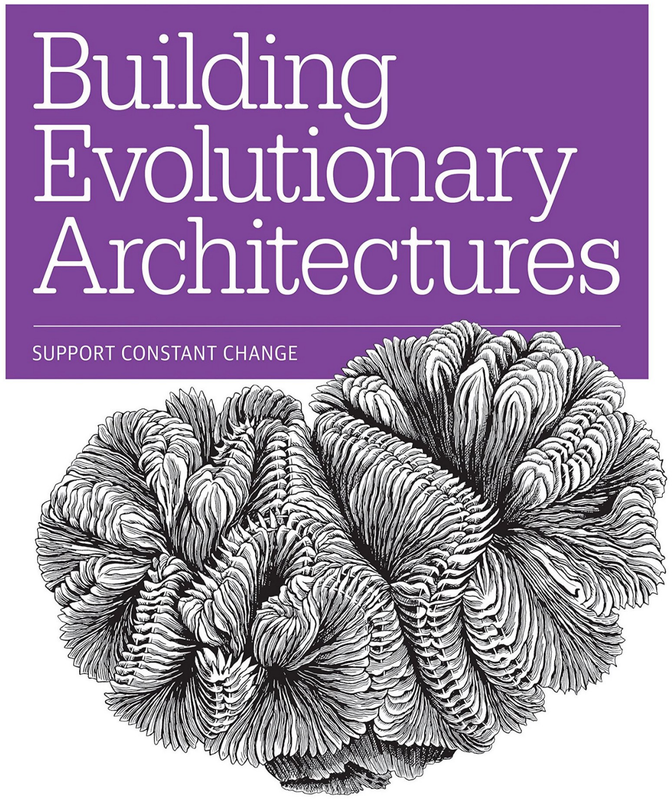
Presentation Layer	ASP.NET MVC and Autofac
Business Layer	MediatR and AutoMapper
Data Layer	Entity Framework and Dapper
Database and Messaging	SQL Server and RabbitMQ
IDE and Tools	Visual Studio, Resharper and .NET Framework

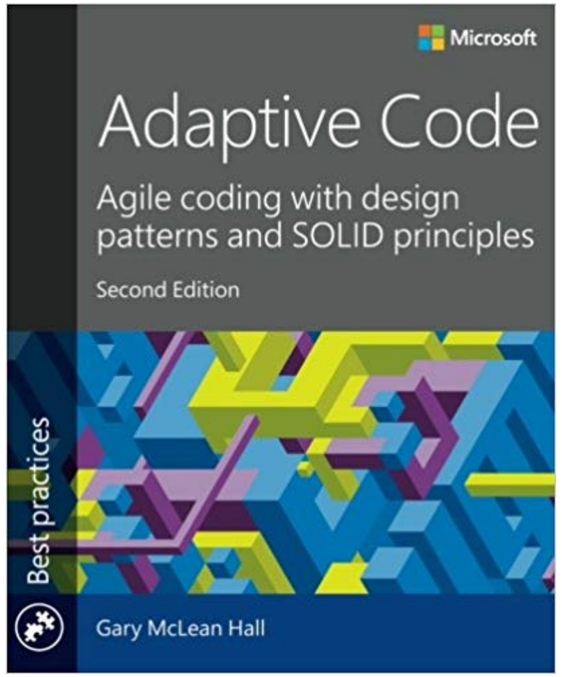
The “Software Architecture”

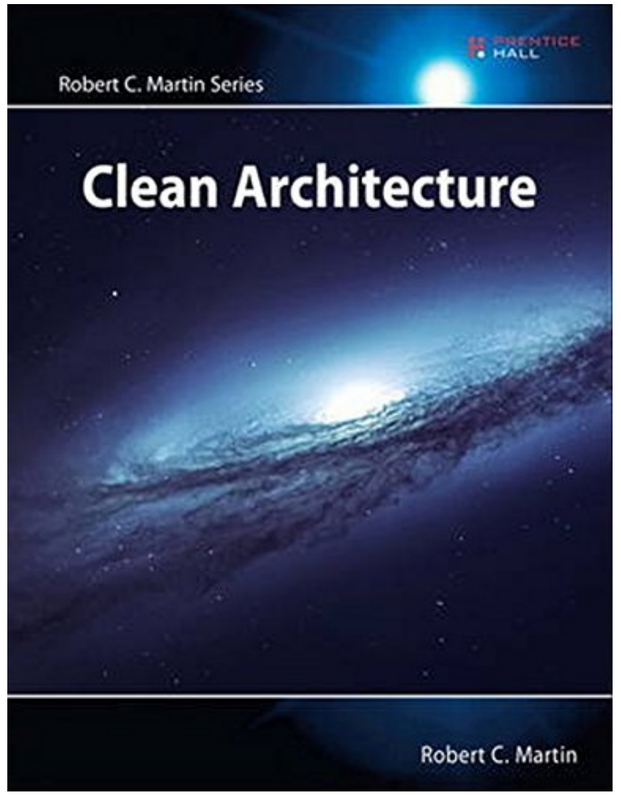
Presentation Layer	ASP.NET MVC and Autofac
Business Layer	MediatR and AutoMapper
Data Layer	Entity Framework and Dapper
Database and Messaging	SQL Server and RabbitMQ
IDE and Tools	Visual Studio, Resharper and .NET Framework

Not a Software Architecture!

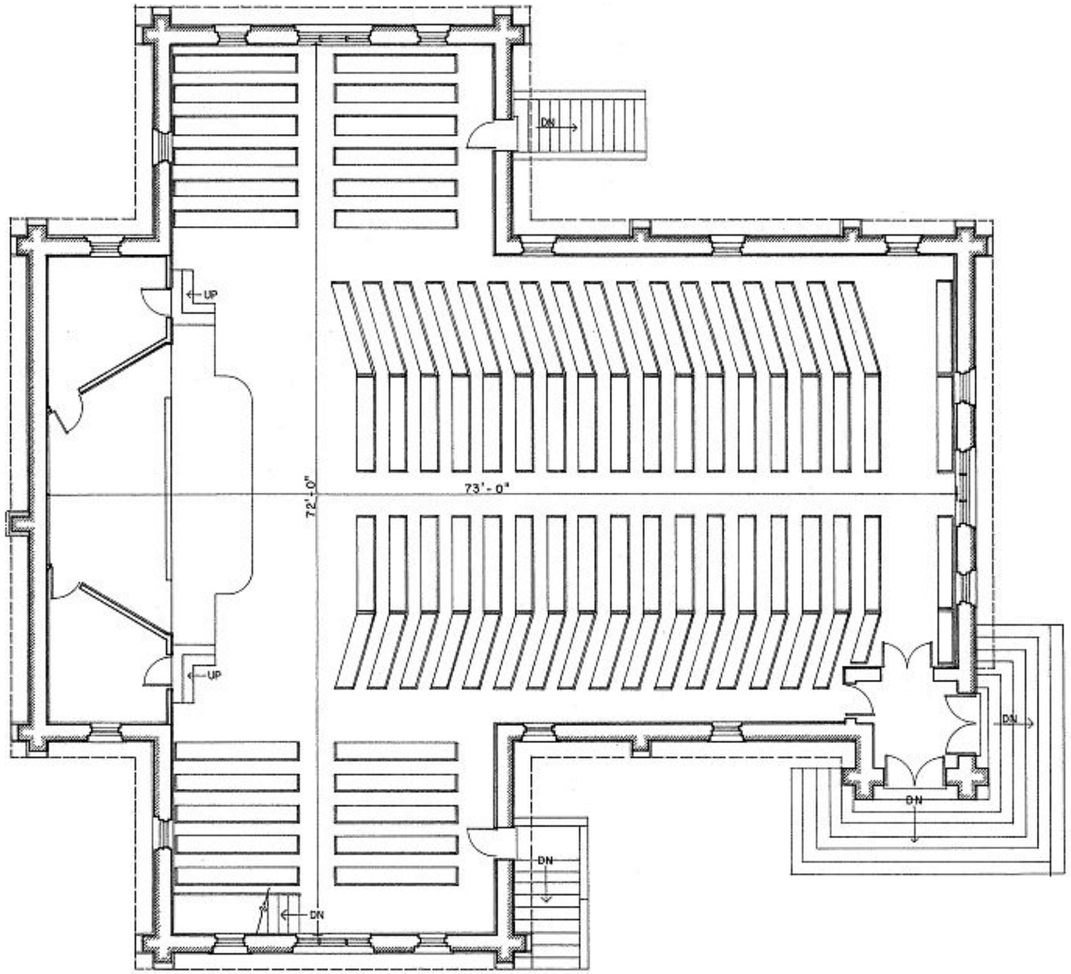
Resources I Like



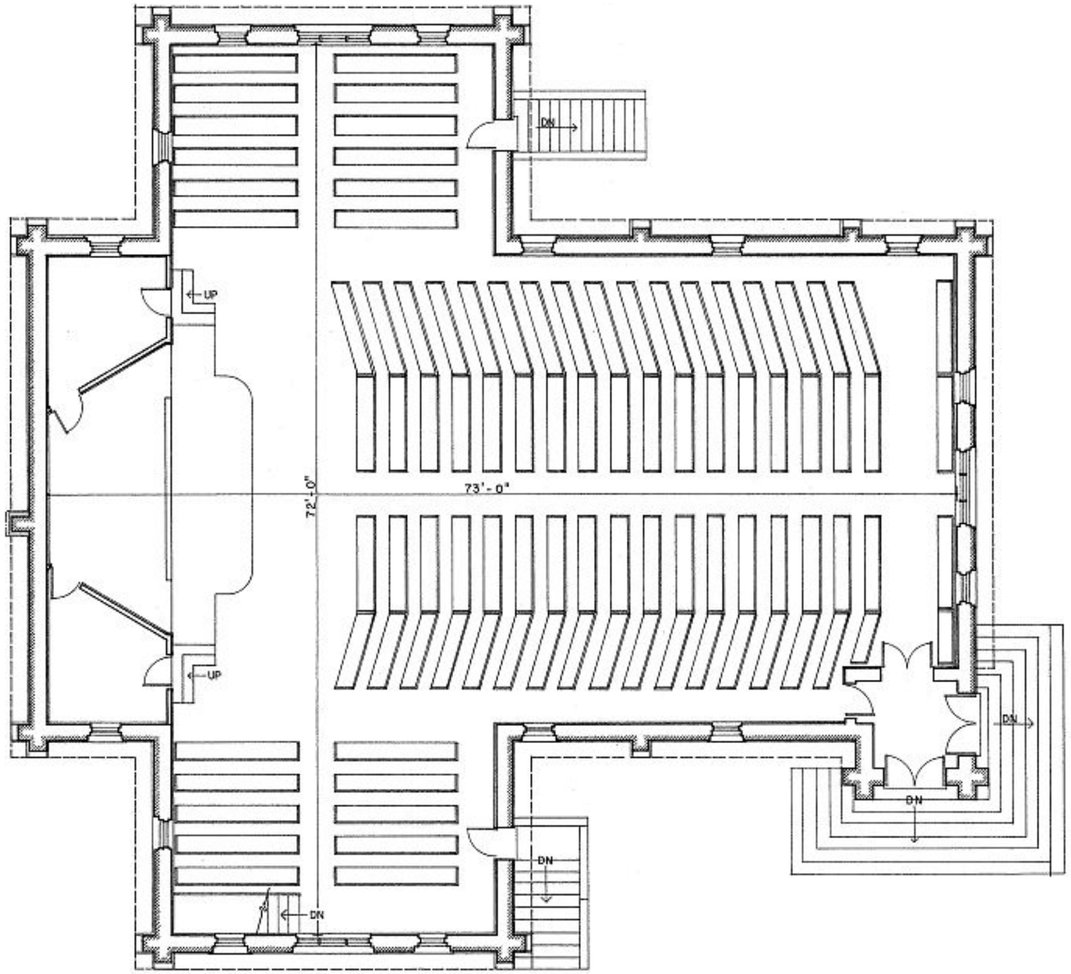




Architecture is About Usage

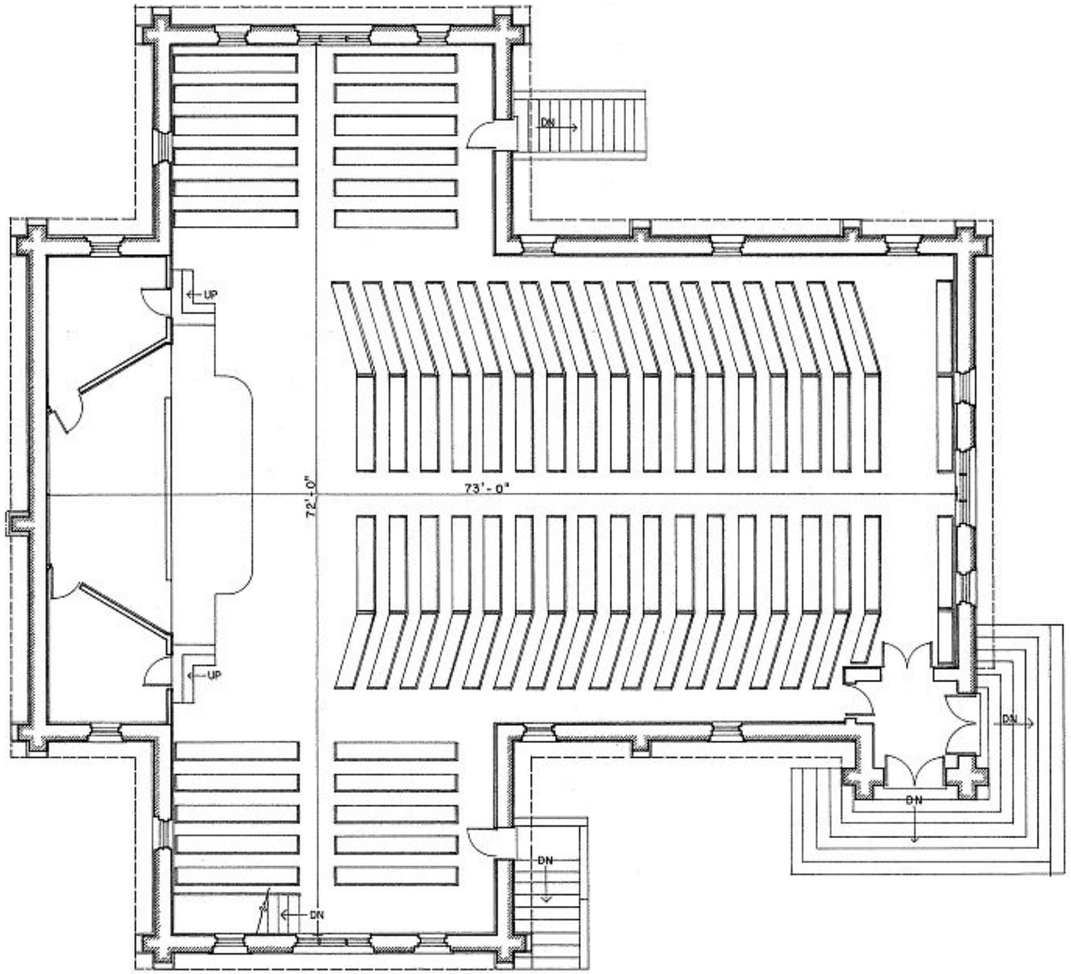


Architecture is About Usage



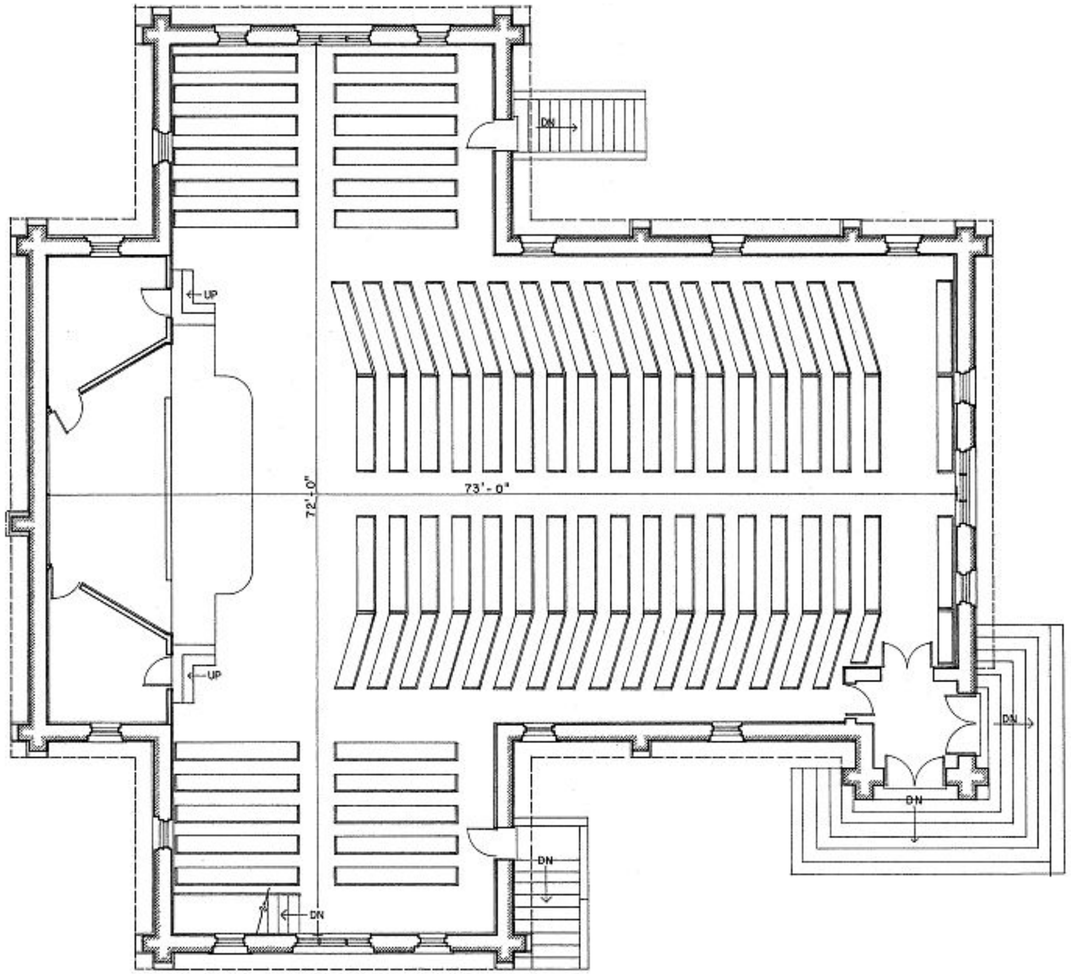
- The floor plan shows where people enter the building and sit. It shows where the focus is.

Architecture is About Usage



- The floor plan shows where people enter the building and sit. It shows where the focus is.
- When looking at the image it is easy to notice that it is a church floor plan.

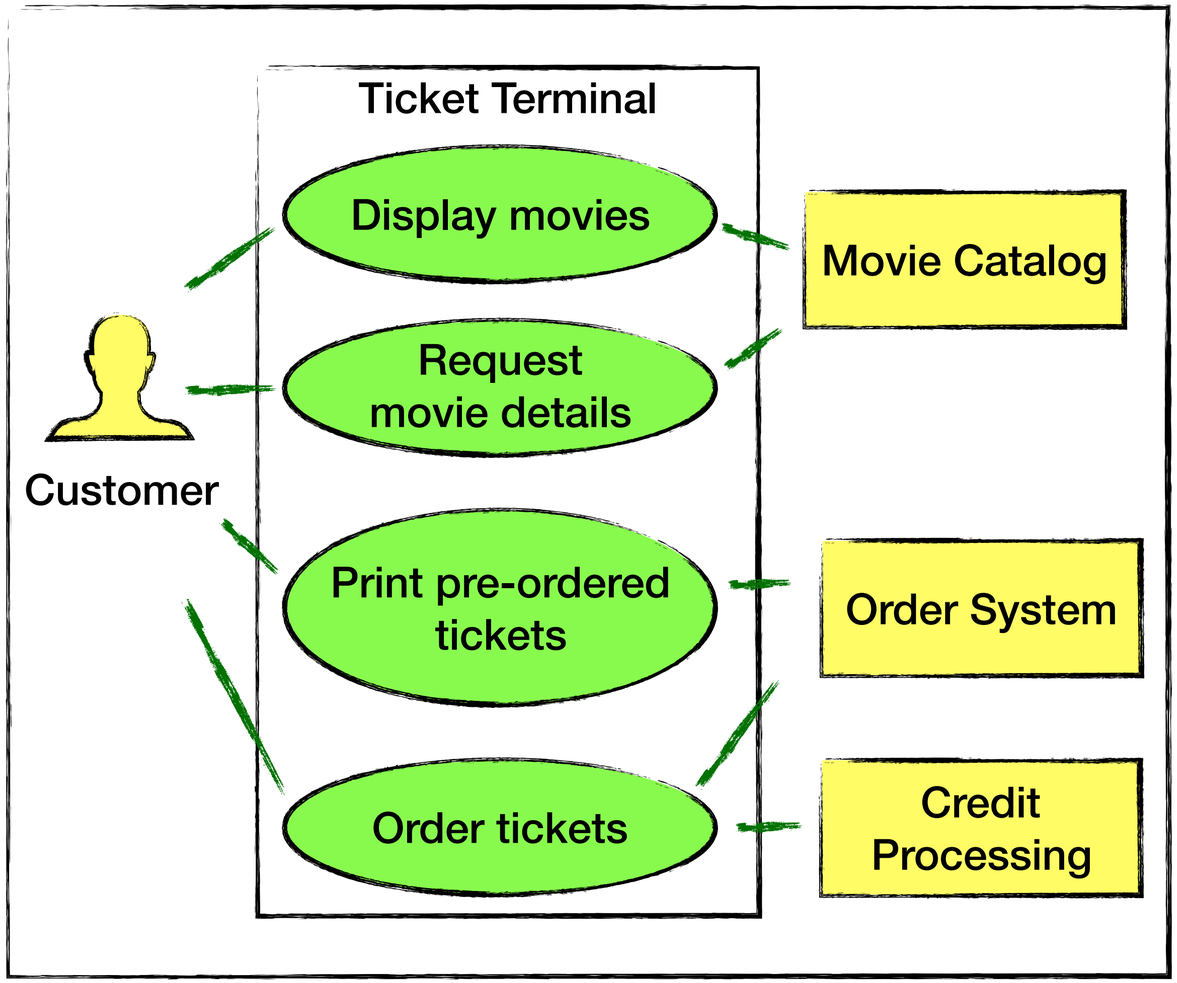
Architecture is About Usage



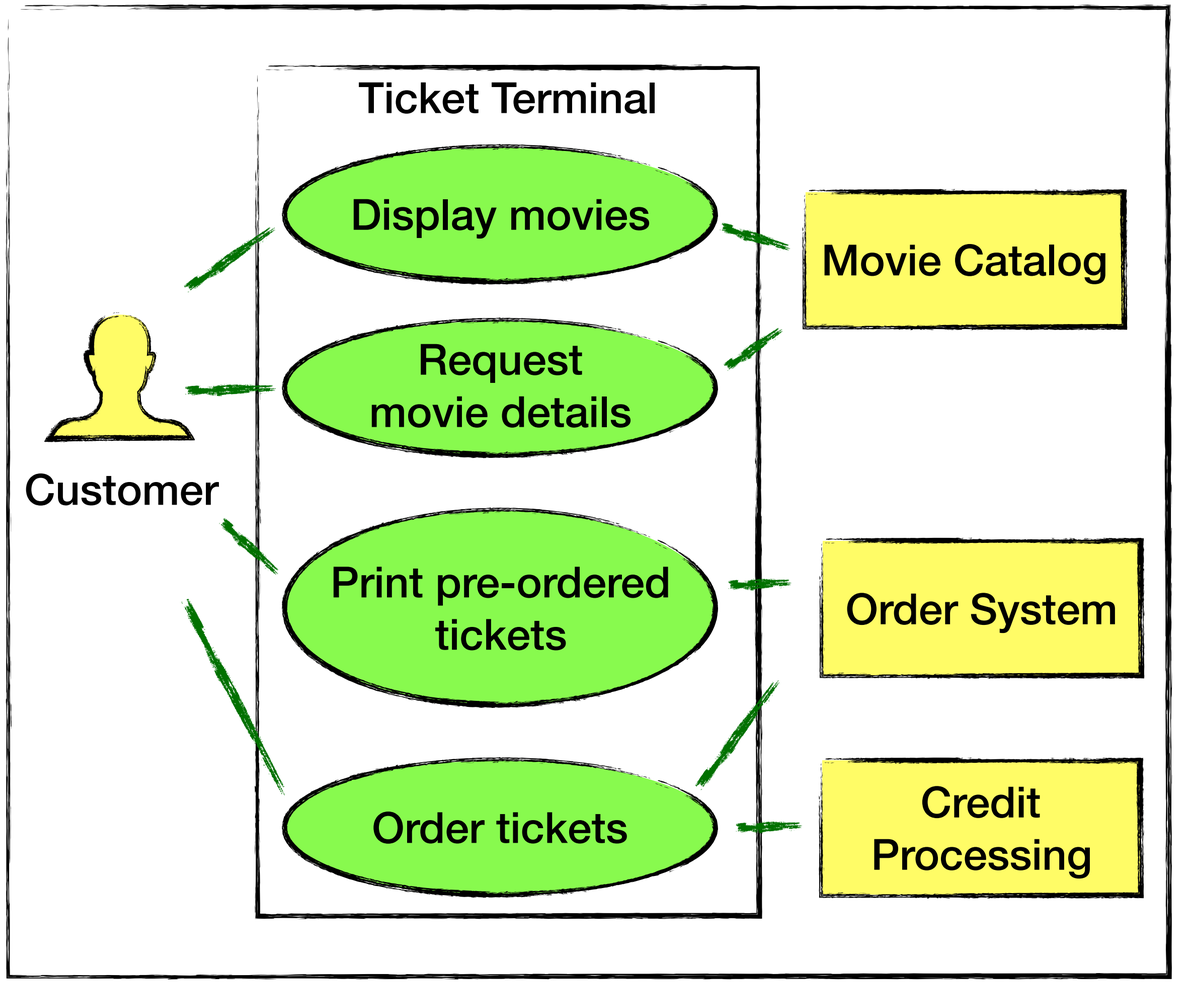
- The floor plan shows where people enter the building and sit. It shows where the focus is.
- When looking at the image it is easy to notice that it is a church floor plan.

How to make the software architecture scream its usage?

Use Cases

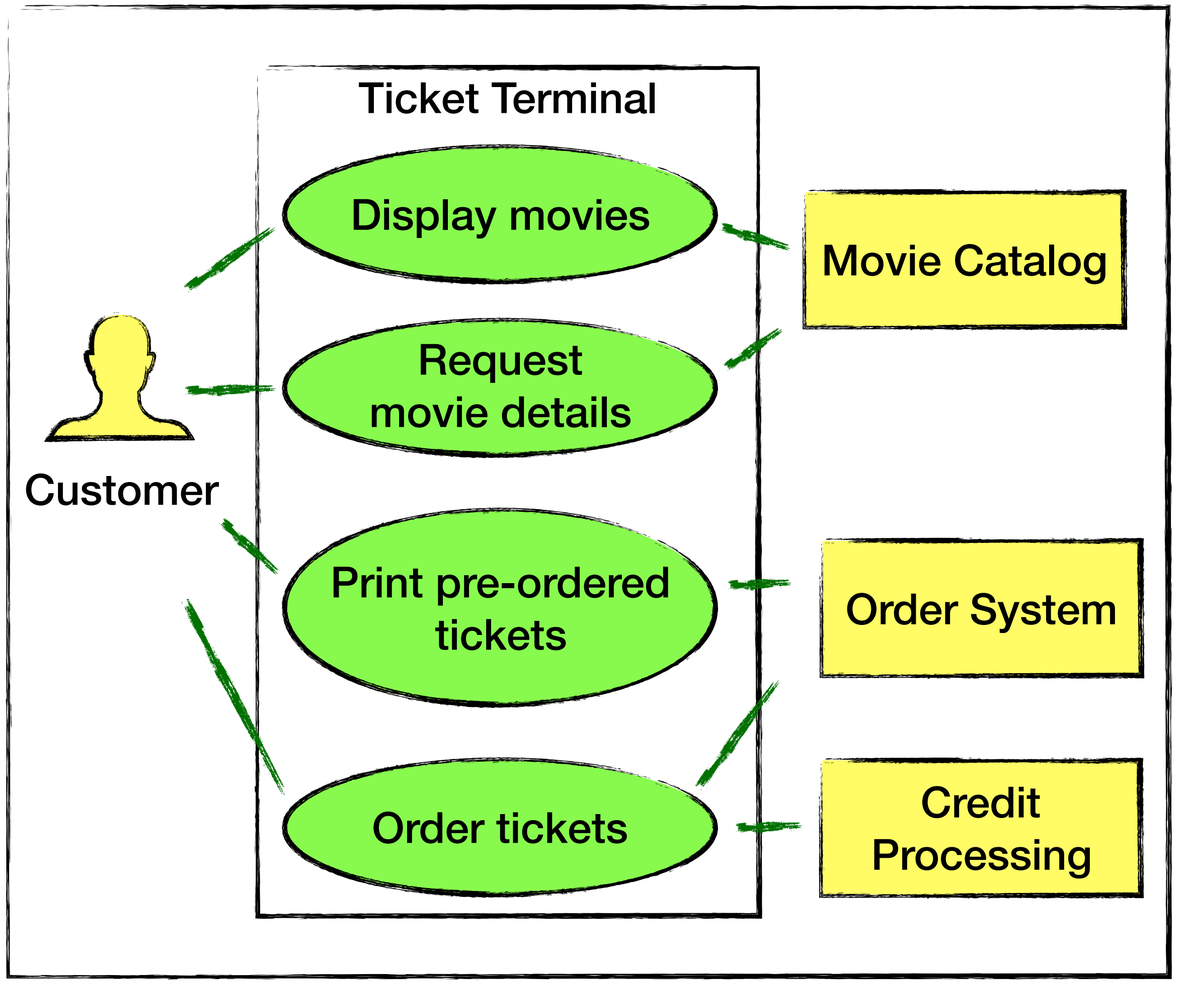

Ticket Terminal
Display movies
Request movie details
Print pre-ordered tickets
Order tickets
Customer
Movie Catalog
Order System
Credit Processing

Use Cases



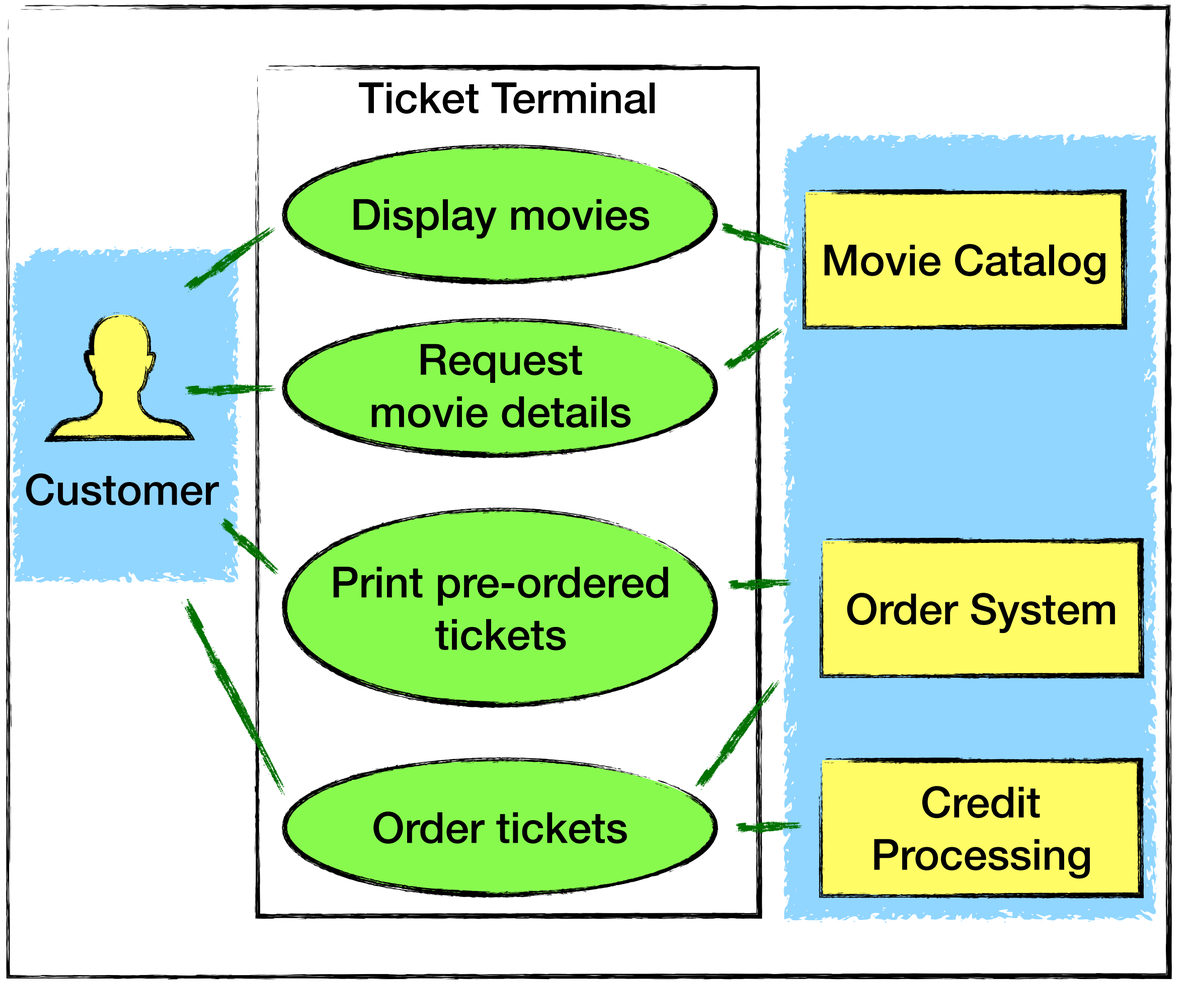
- Use Cases are delivery independent, they show the intent of a system.

Use Cases



- Use Cases are delivery independent, they show the intent of a system.
- Use Cases are algorithms which interpret the input to generate the output data.

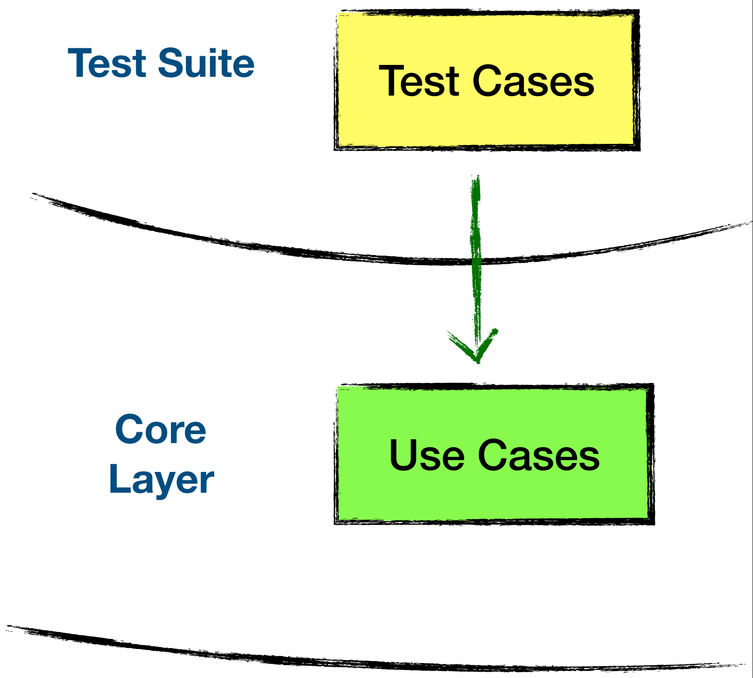
Use Cases



- Use Cases are delivery independent, they show the intent of a system.
- Use Cases are algorithms which interpret the input to generate the output data.
- Primary and secondary actors.

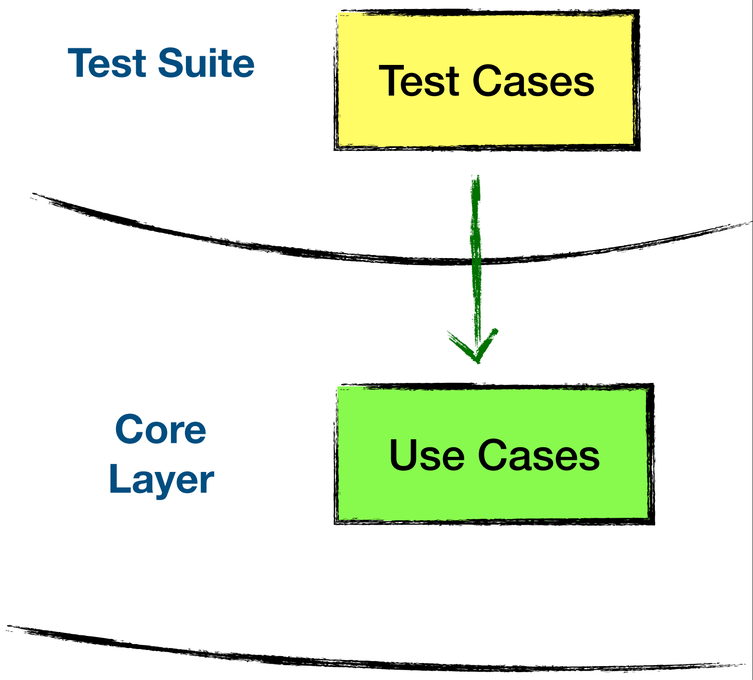
Use Cases Implementation Guided by Tests

- Test Cases should be the first consumers implemented.



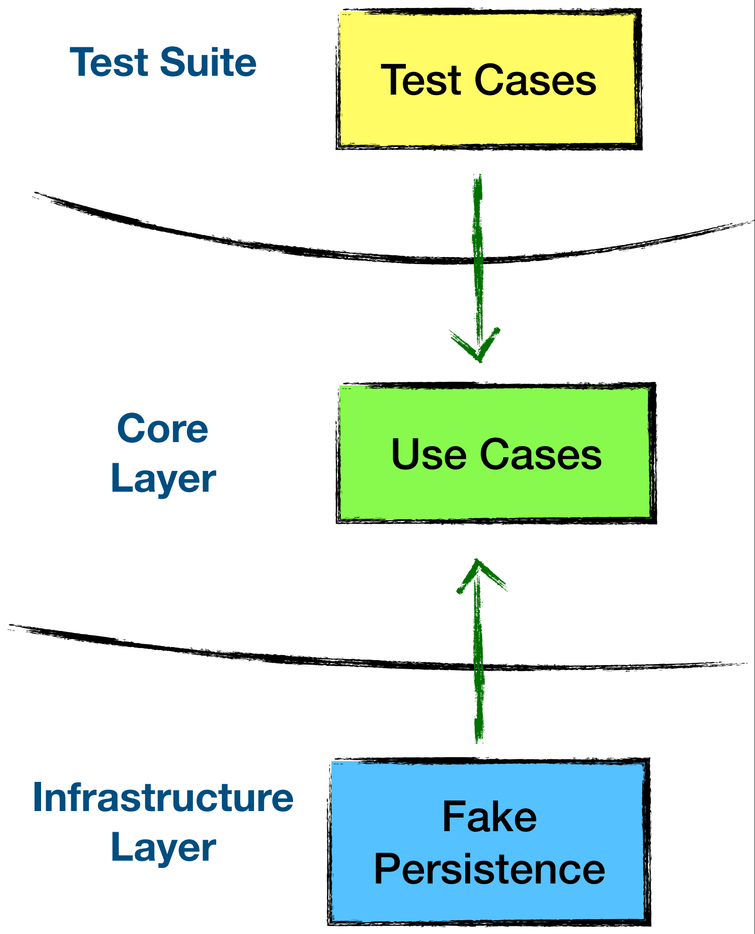
Use Cases Implementation Guided by Tests

- Test Cases should be the first consumers implemented.
- You should design the Use Cases by following the Single Responsibility Principle.



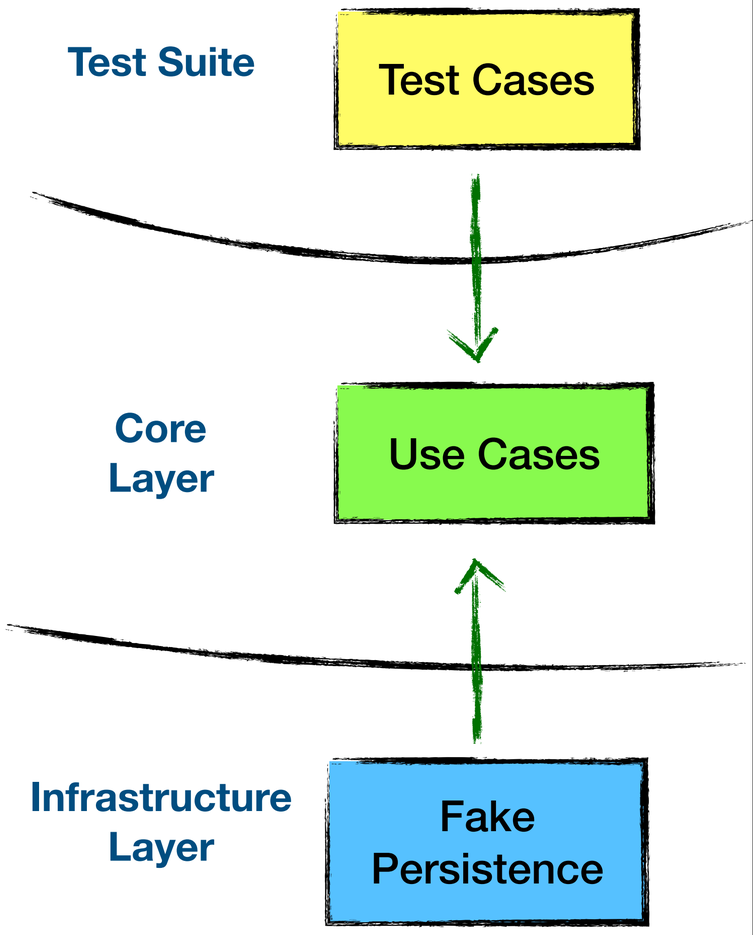
Use Cases Implementation Guided by Tests

- Test Cases should be the first consumers implemented.
- You should design the Use Cases by following the Single Responsibility Principle.
- Fake implementations should be the first delivery mechanisms implemented.



Use Cases Implementation Guided by Tests

- Test Cases should be the first consumers implemented.
- You should design the Use Cases by following the Single Responsibility Principle.
- Fake implementations should be the first delivery mechanisms implemented.
- Test Cases help you design fine grained Use Cases (Interface Segregation Principle).

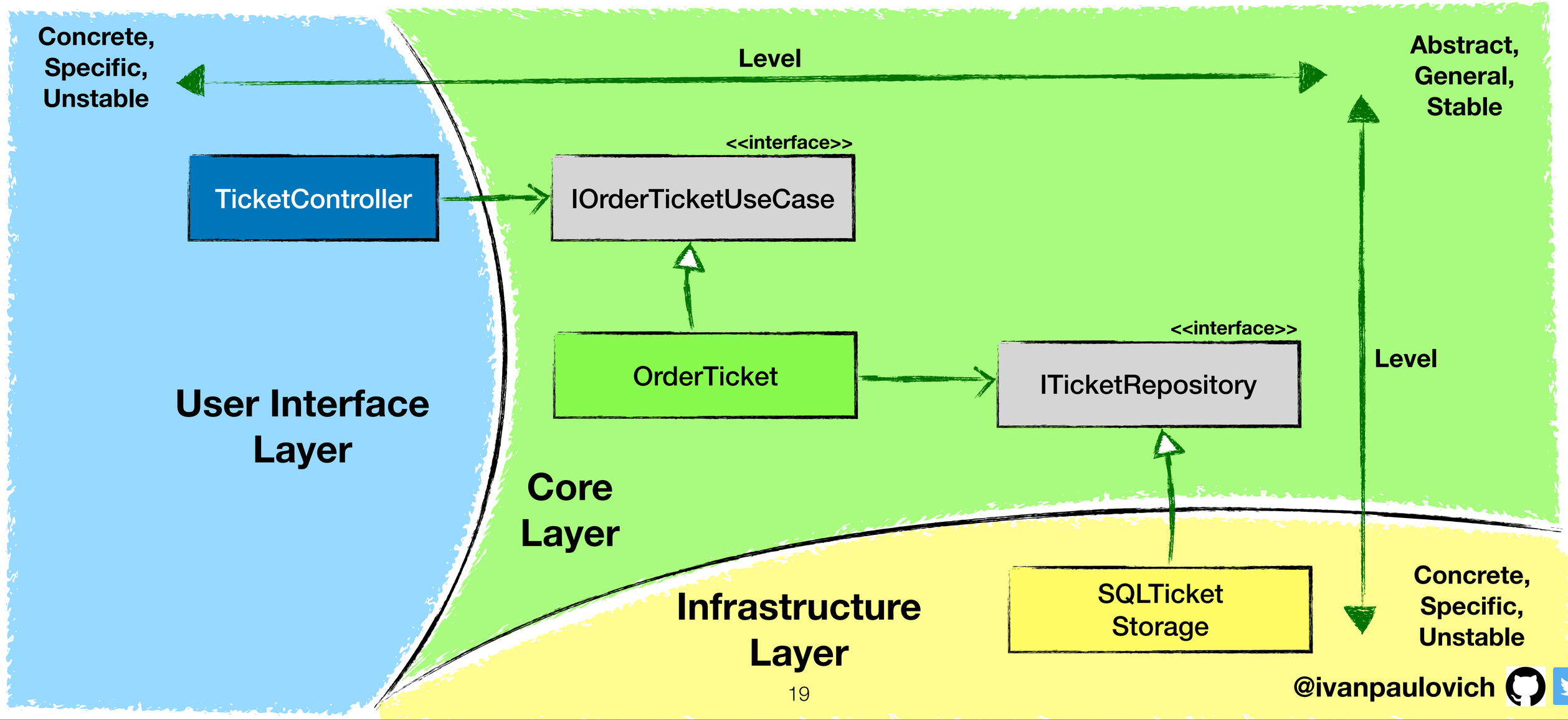


Software Architecture

“A good architecture **allows major decisions to be deferred**. A good architect maximizes the number of **decisions not made**.

Uncle Bob.

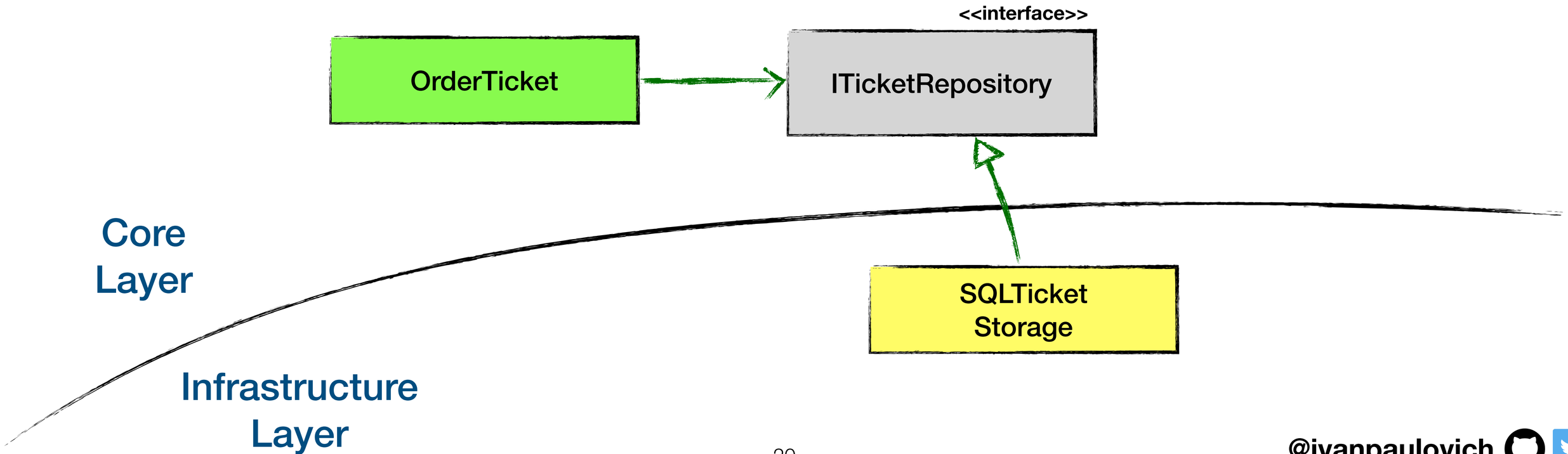
Decoupling
User Interface, Use Cases and Repositories



@ivanpaulovich

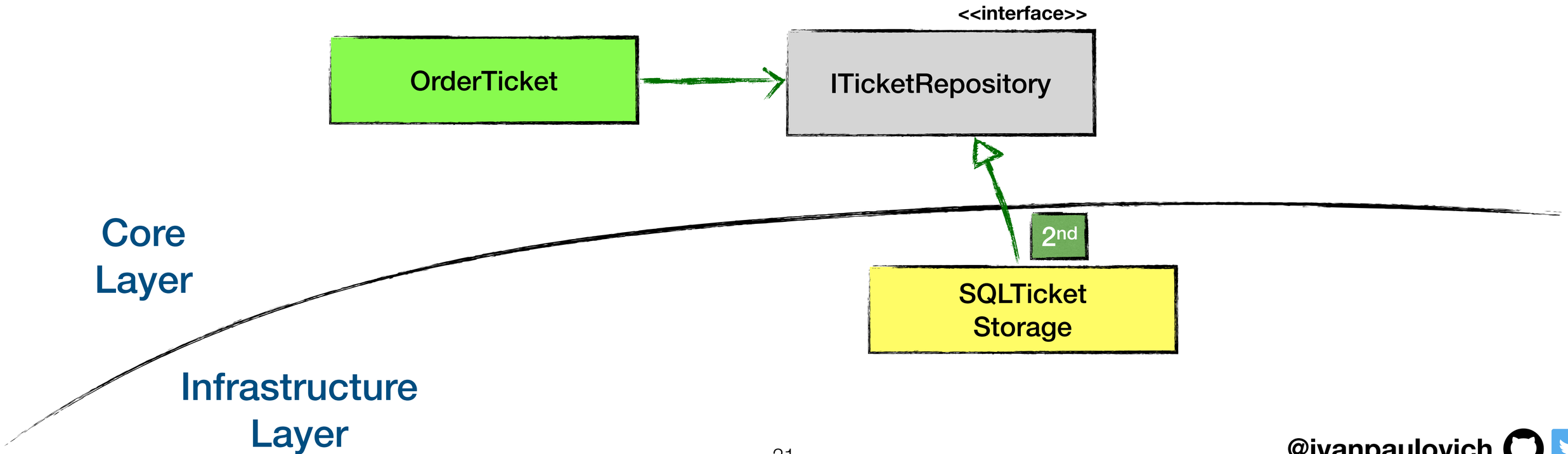
Plugin Architecture

- The dependencies point in the direction of the abstract components.



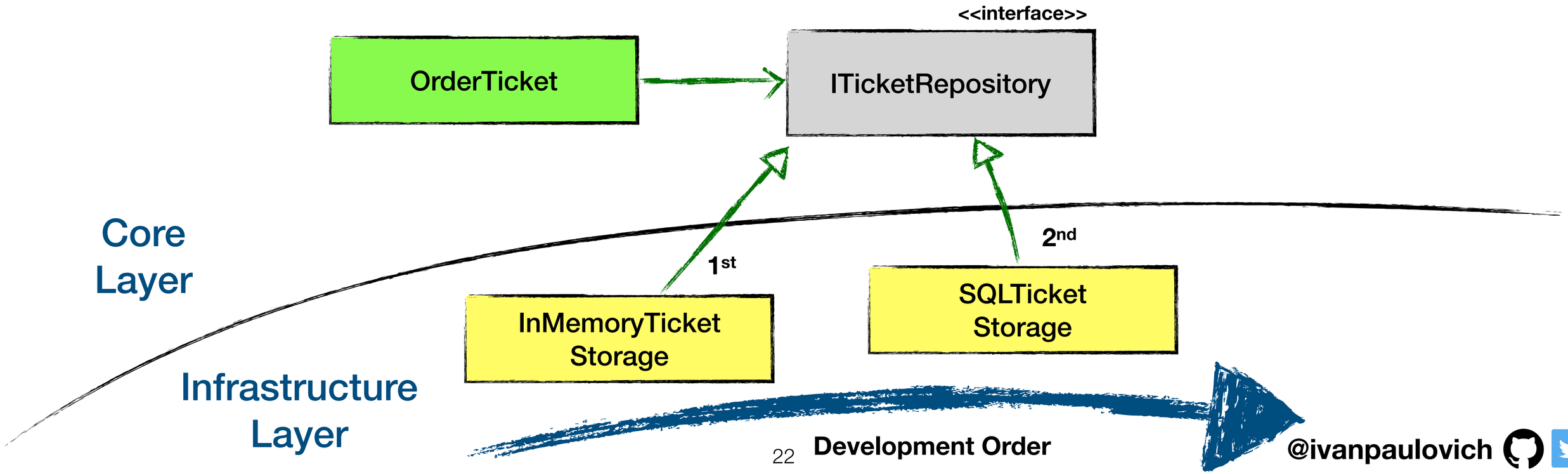
Plugin Architecture

- The dependencies point in the direction of the abstract components.



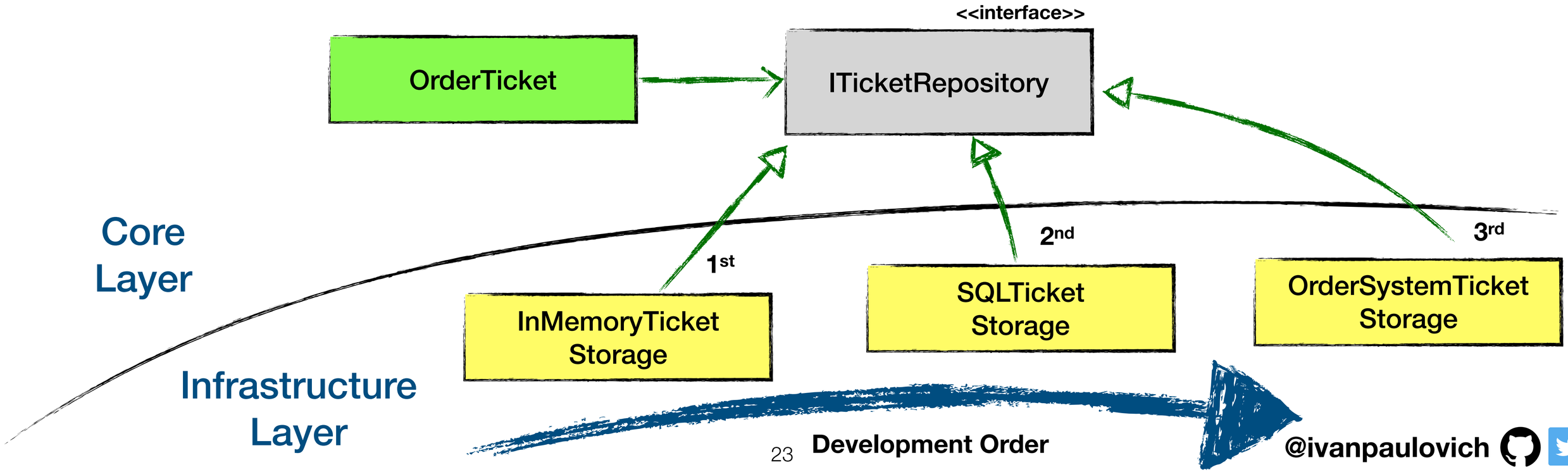
Plugin Architecture

- The dependencies point in the direction of the abstract components.
- First implement the simplest component version!



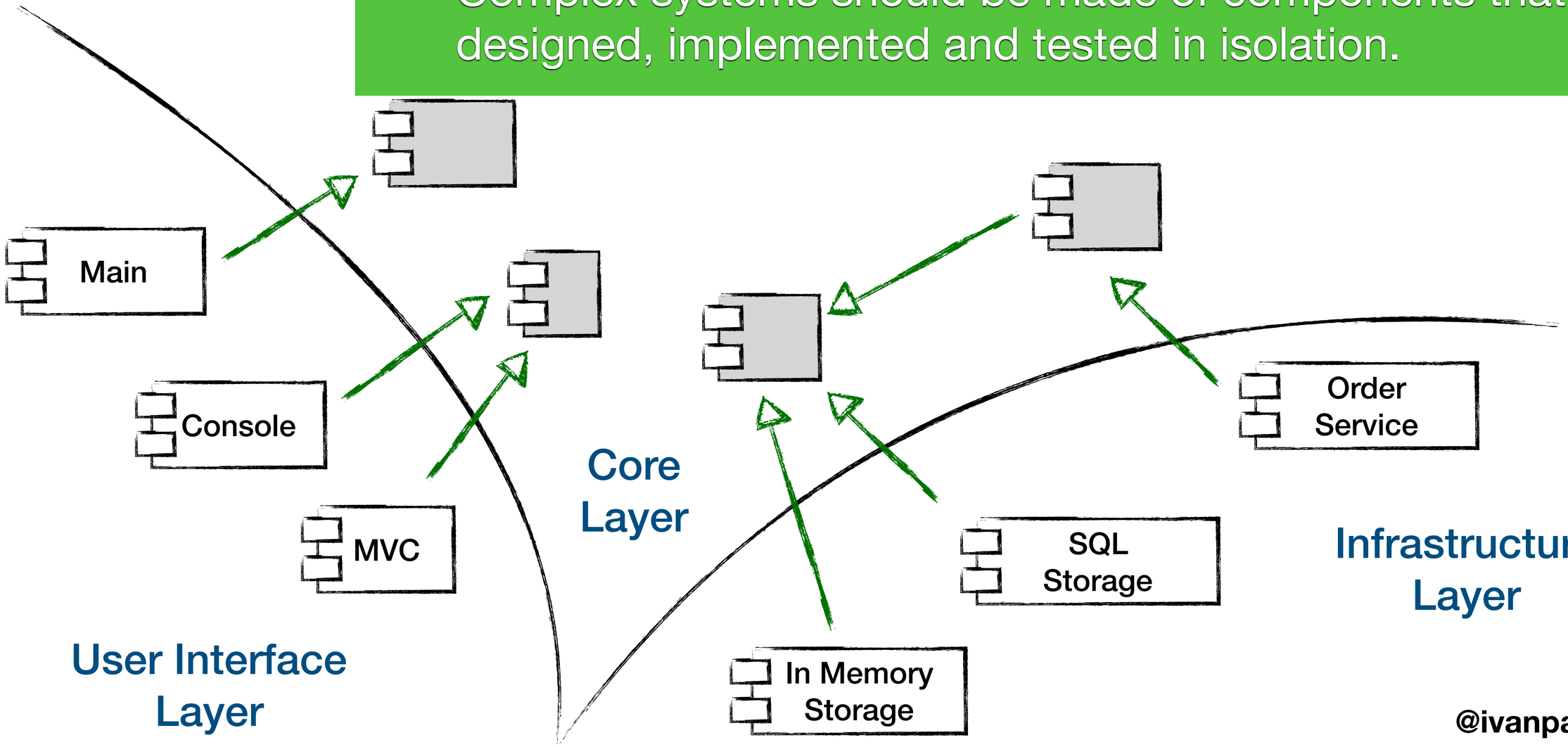
Plugin Architecture

- The dependencies point in the direction of the abstract components.
- First implement the simplest component version!
- Defer decisions. Keep the options open as long as possible!



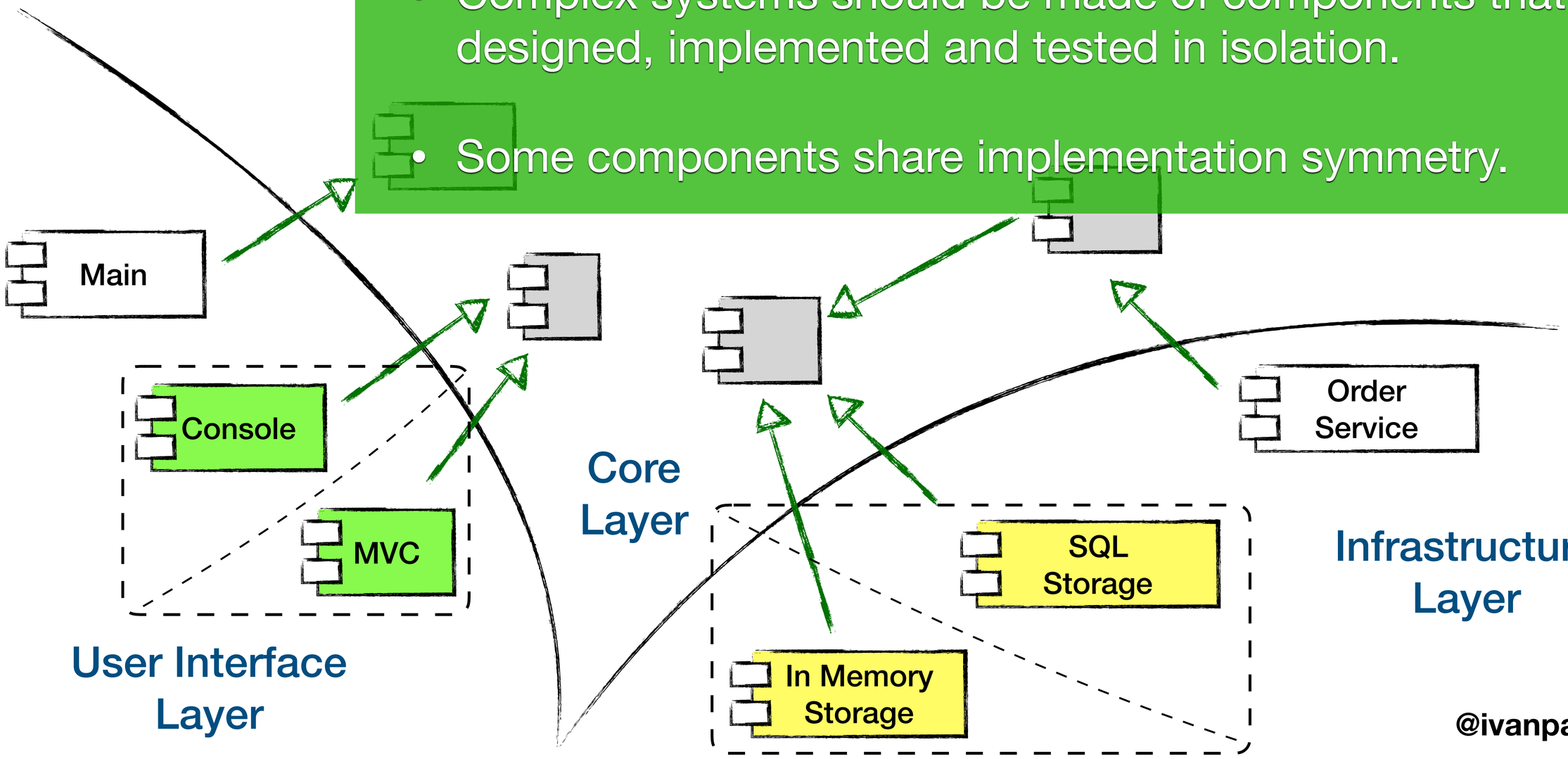
High Level Design

- Complex systems should be made of components that are designed, implemented and tested in isolation.



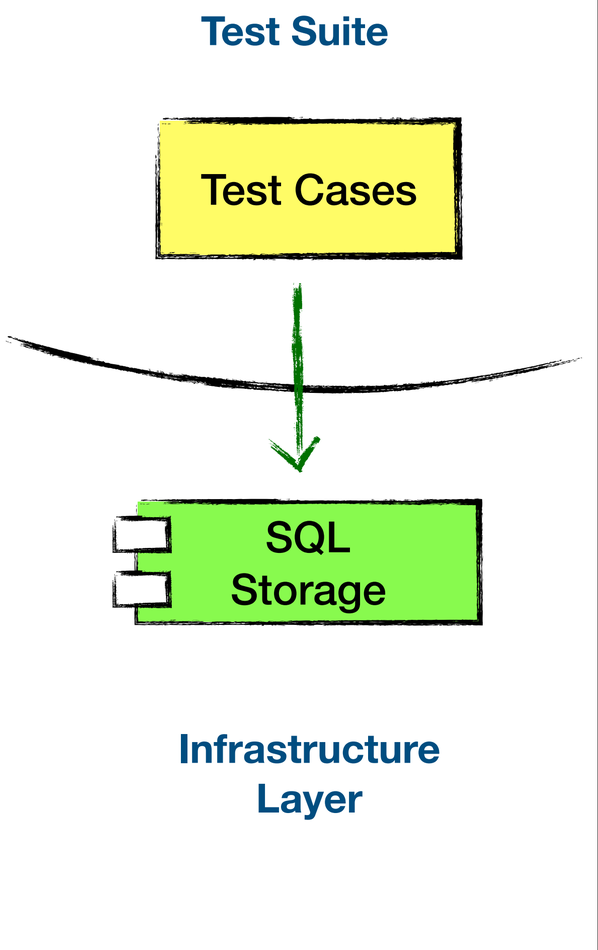
High Level Design

- Complex systems should be made of components that are designed, implemented and tested in isolation.
- Some components share implementation symmetry.



Components Overview

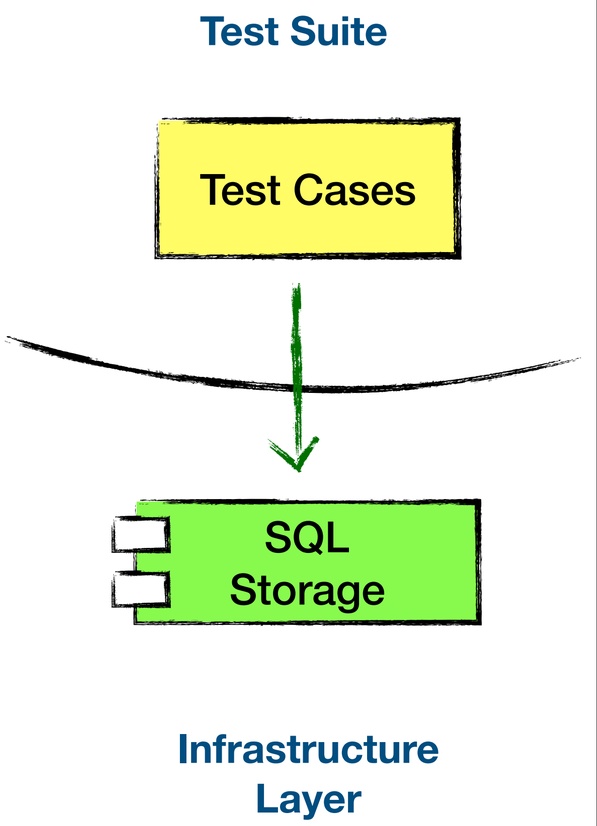
1. Components are designed to fulfill the Core needs and are implemented and tested in isolation.



Components Overview

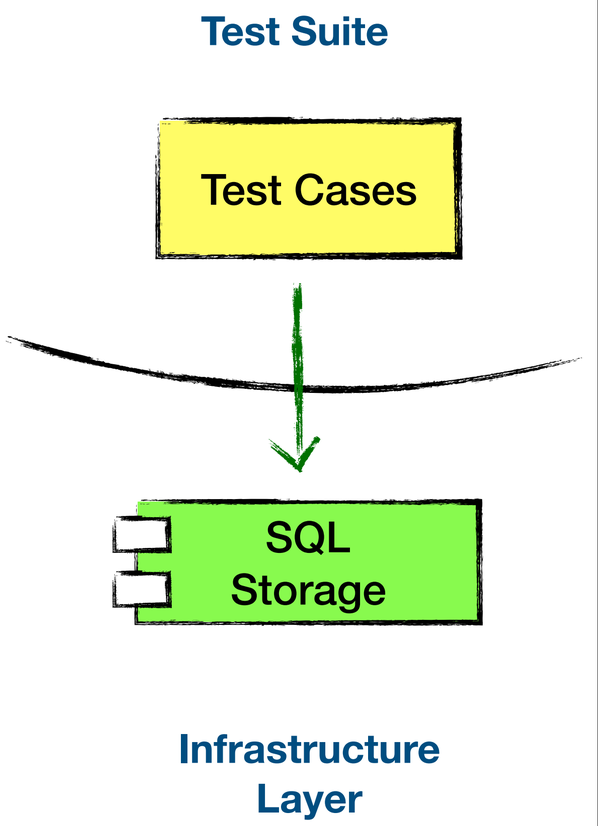
1. Components are designed to fulfill the Core needs and are implemented and tested in isolation.

2. Components could be replaced, upgraded or decommissioned with minimum Core business impact.



Components Overview

1. Components are designed to fulfill the Core needs and are implemented and tested in isolation.
2. Components could be replaced, upgraded or decommissioned with minimum Core business impact.
3. Good components are loosely coupled.



Frameworks are *Details*

- Application code called by frameworks should be under control.

Frameworks are *Details*

- Application code called by frameworks should be under control.
- Hide 3rd party libraries, use them behind core interfaces.

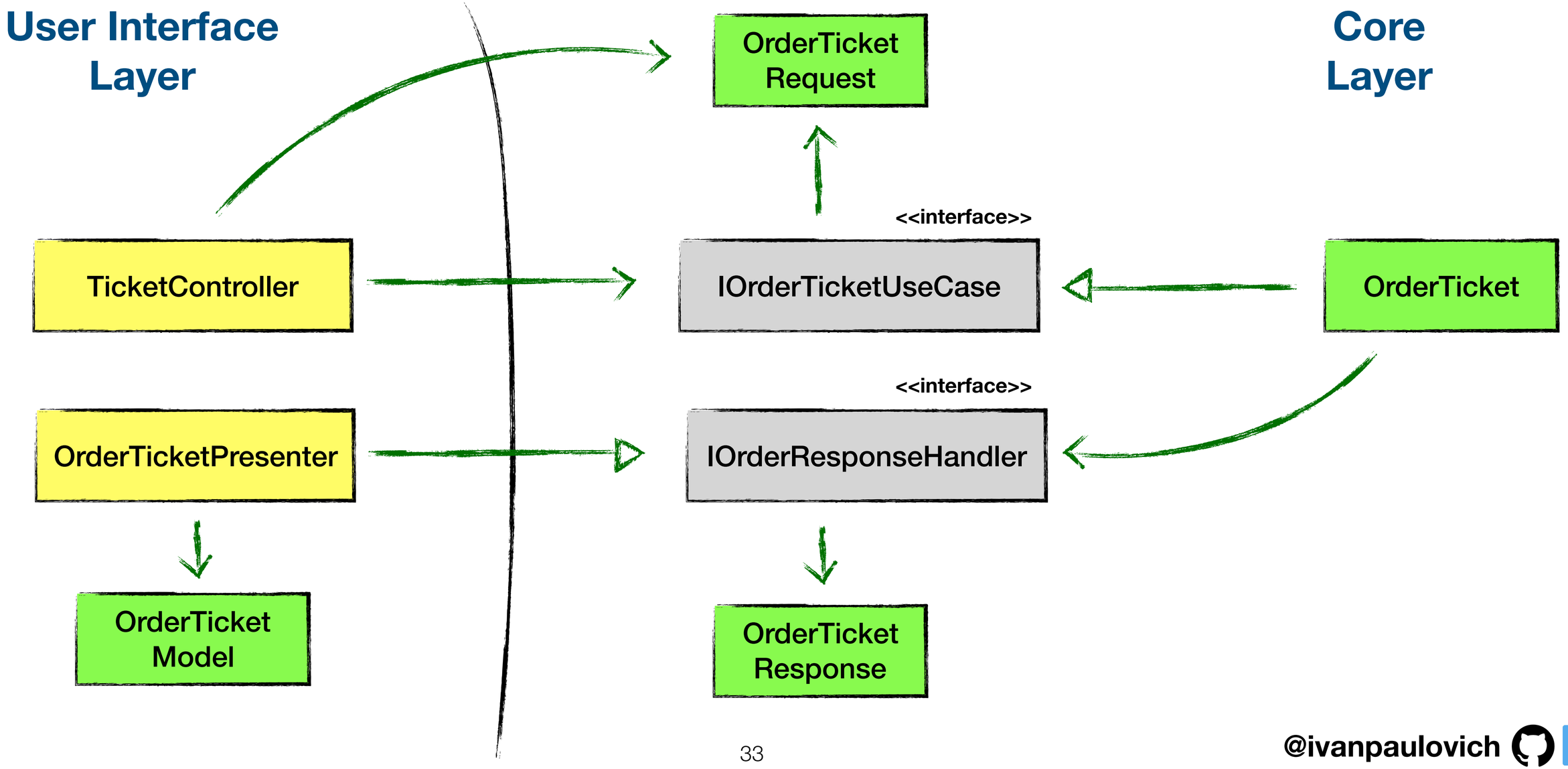
Frameworks are *Details*

- Application code called by frameworks should be under control.
- Hide 3rd party libraries, use them behind core interfaces.
- The .NET Framework is a detail.

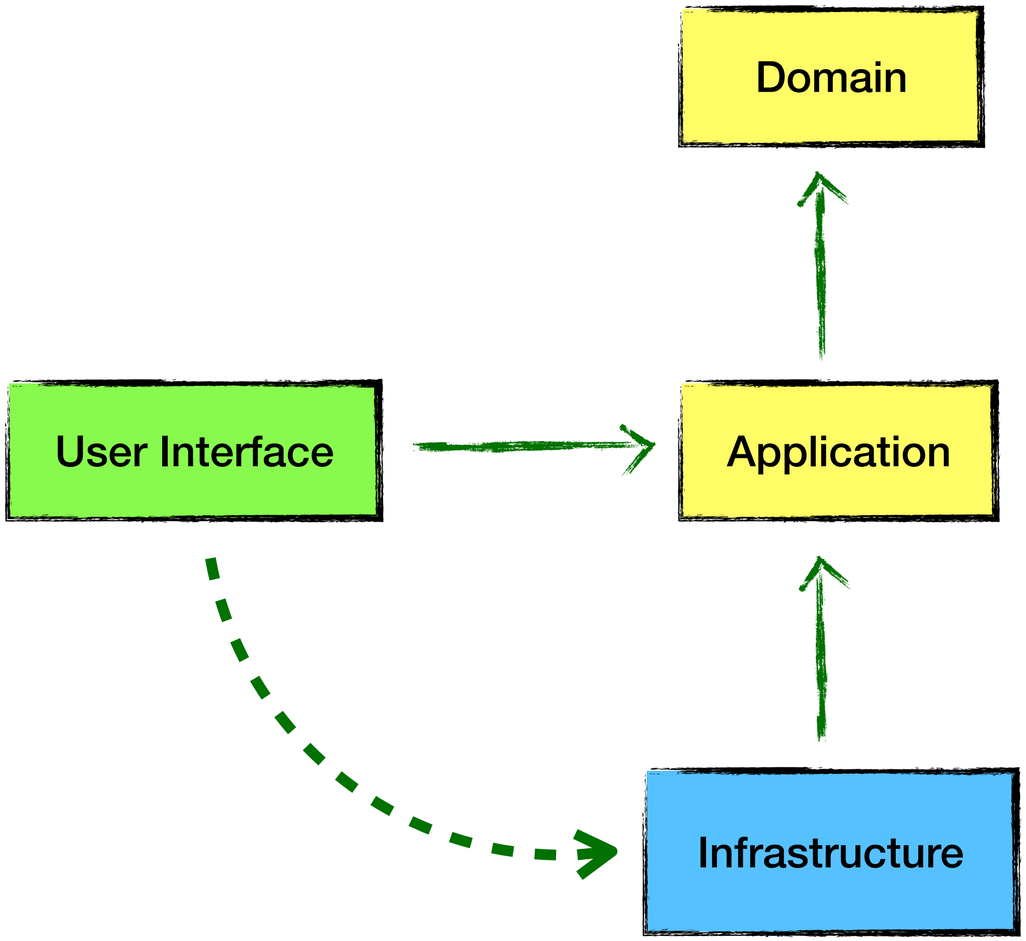
Frameworks are *Details*

- Application code called by frameworks should be under control.
- Hide 3rd party libraries, use them behind core interfaces.
- The .NET Framework is a detail.
- Keep outside the application: Reflection, Linq, Entity Framework, MVC, Data Annotations, WCF.

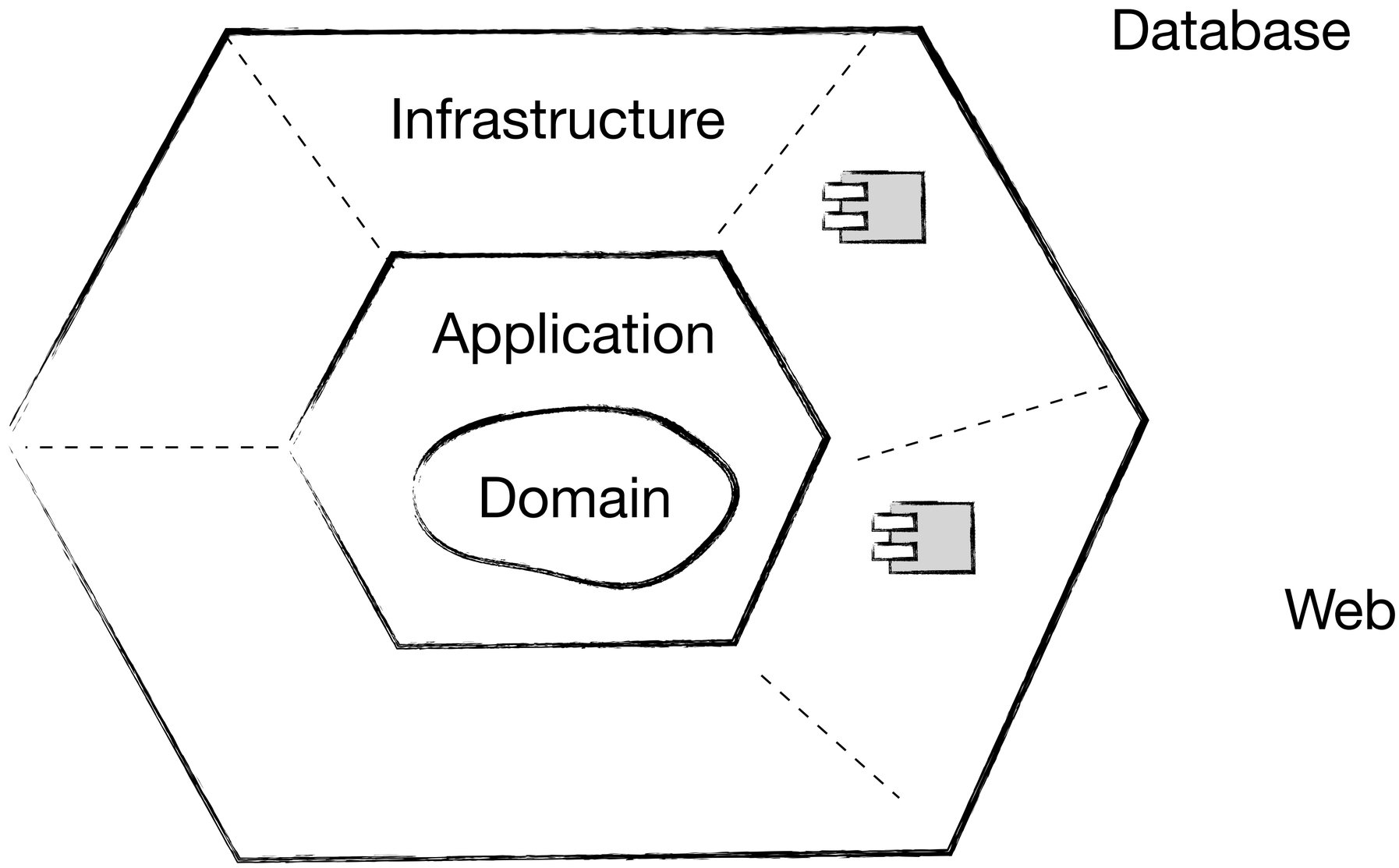
The UI is a *detail*



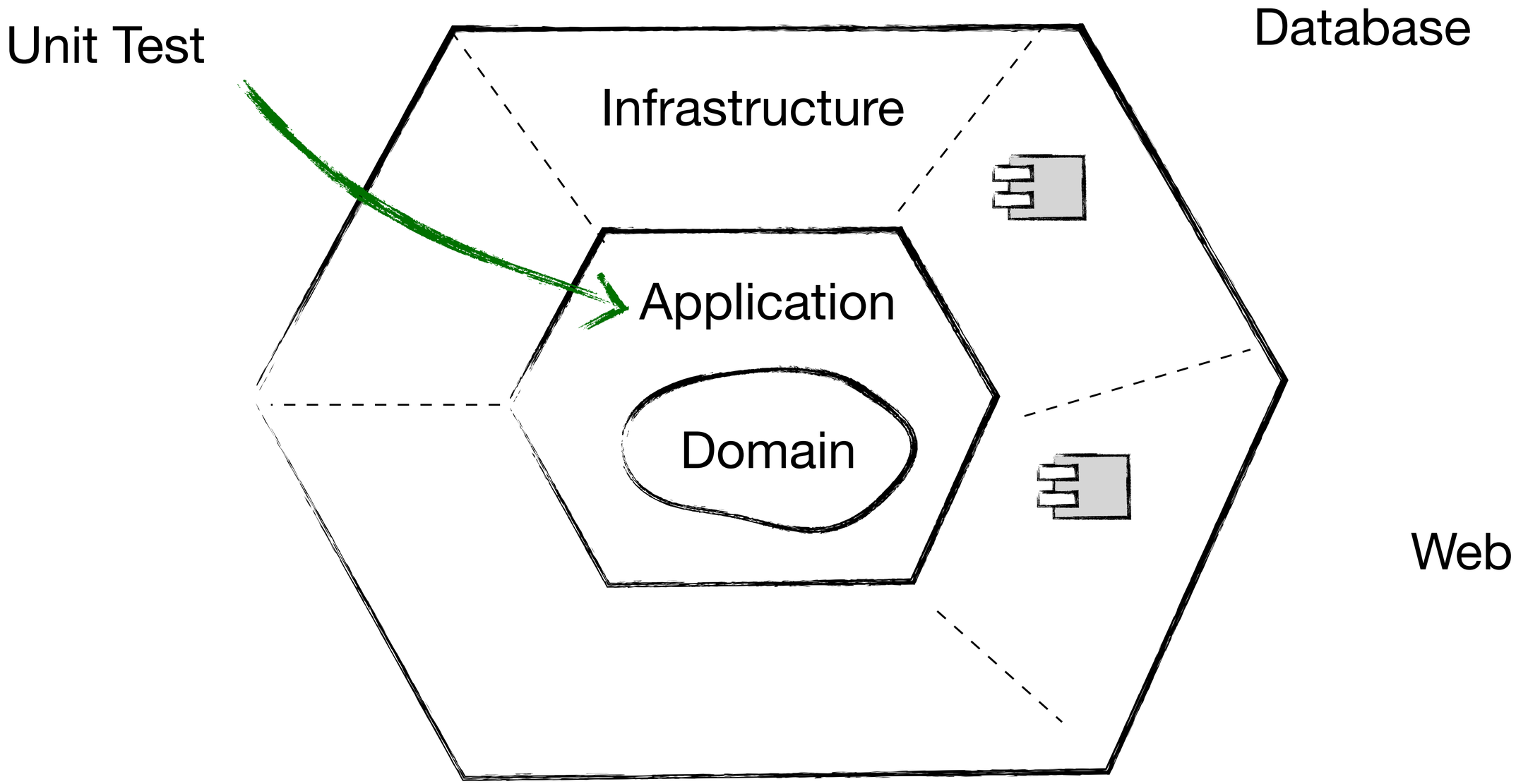
Layered Diagram



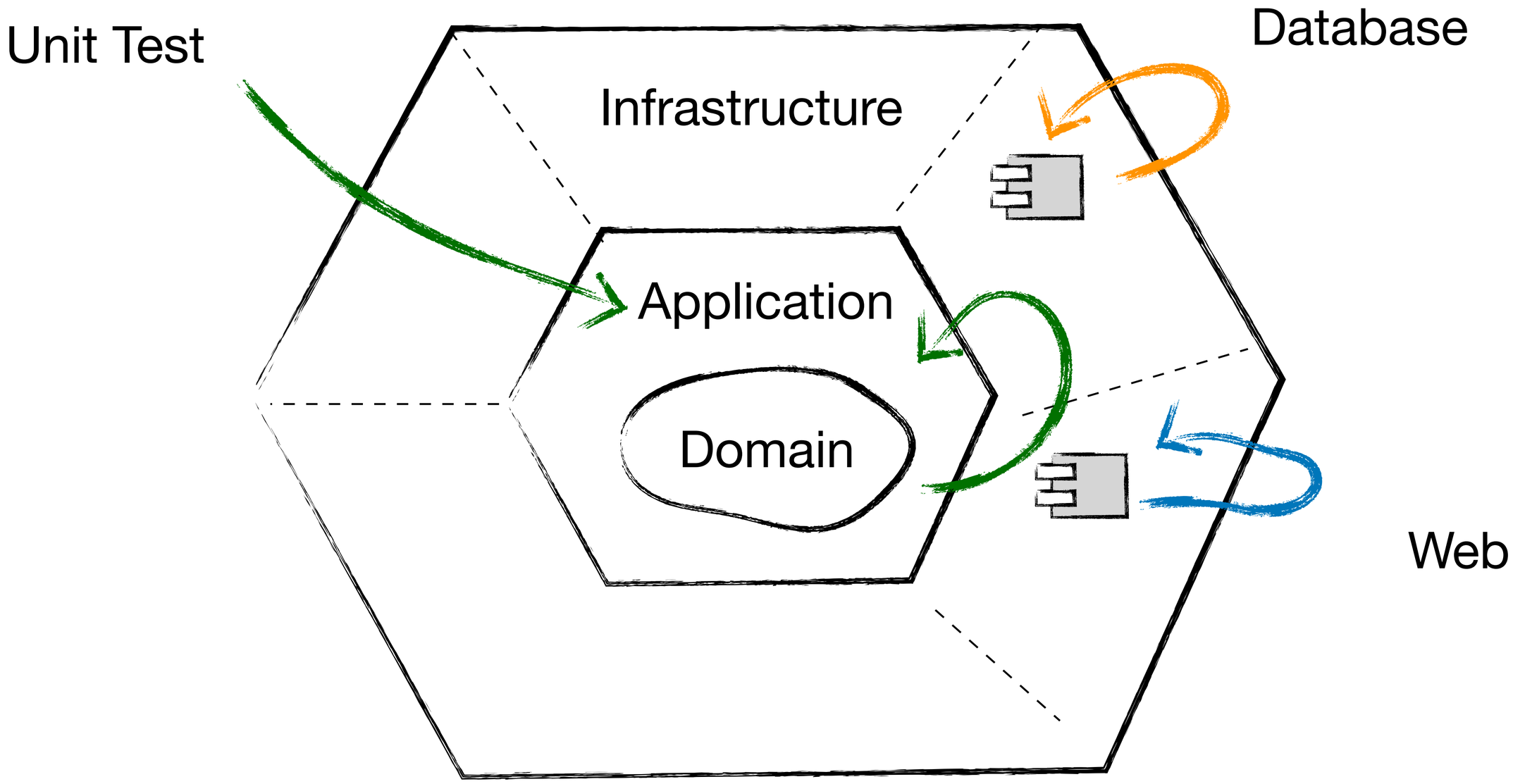
Ports and Adapters



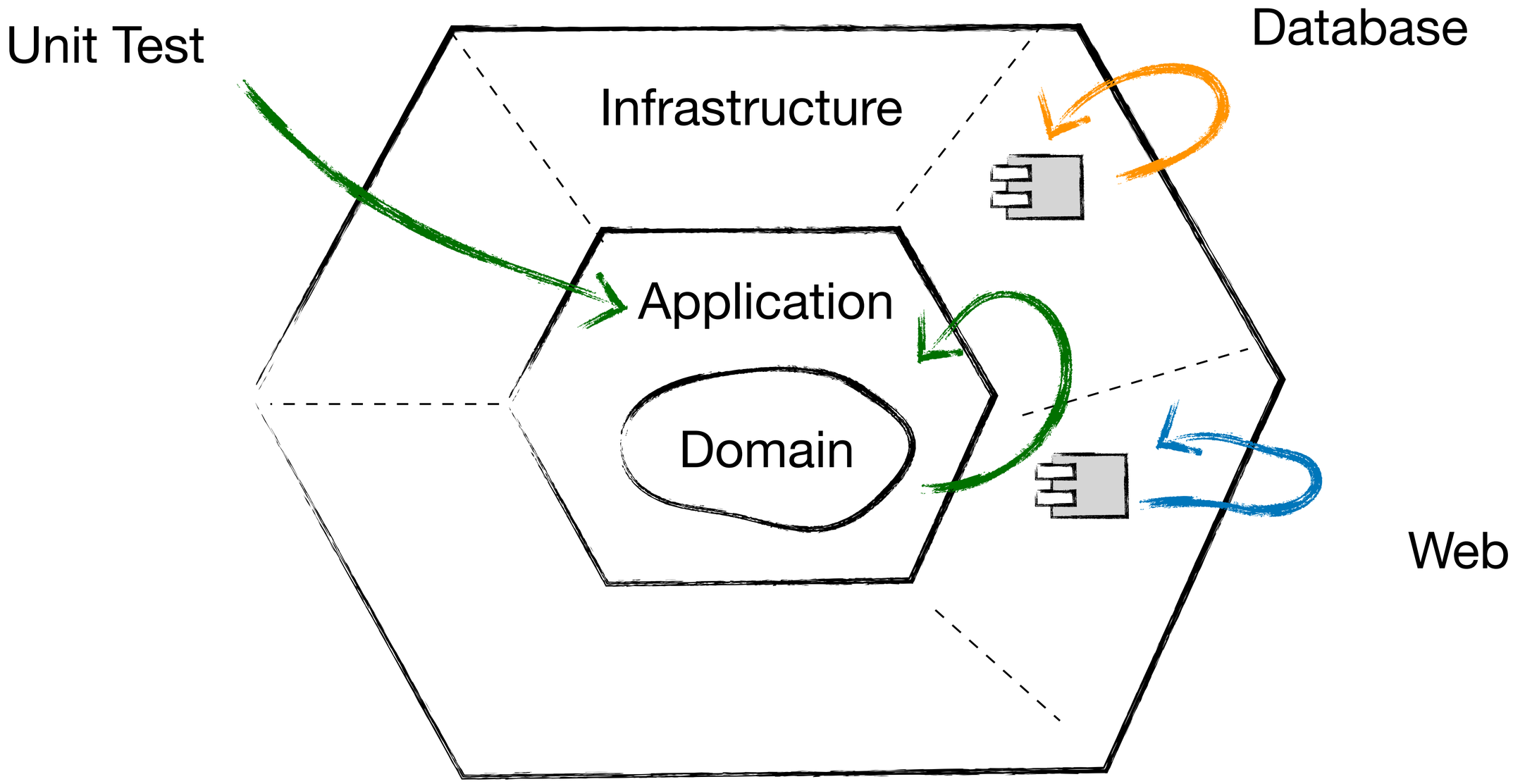
Ports and Adapters



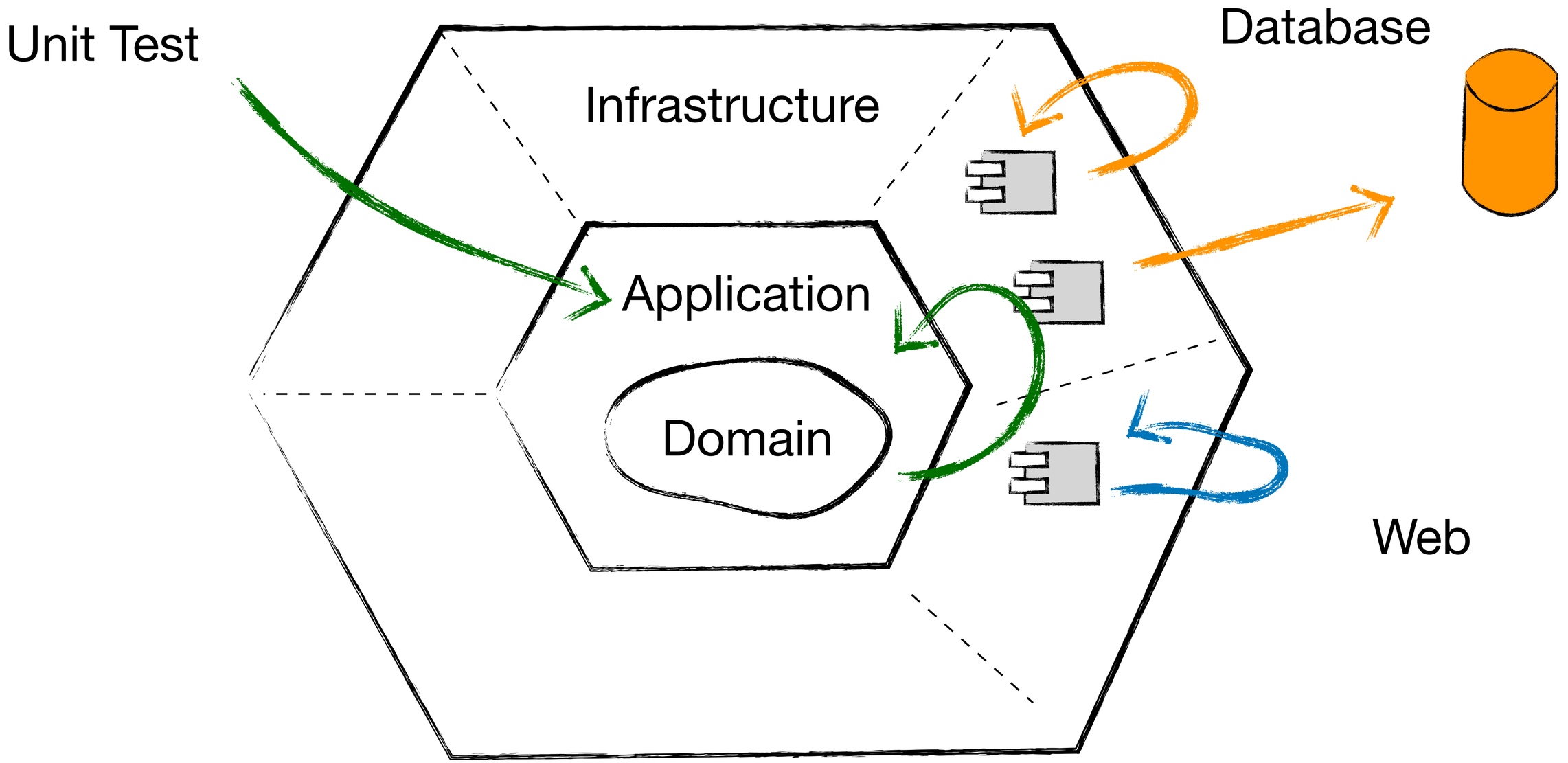
Ports and Adapters



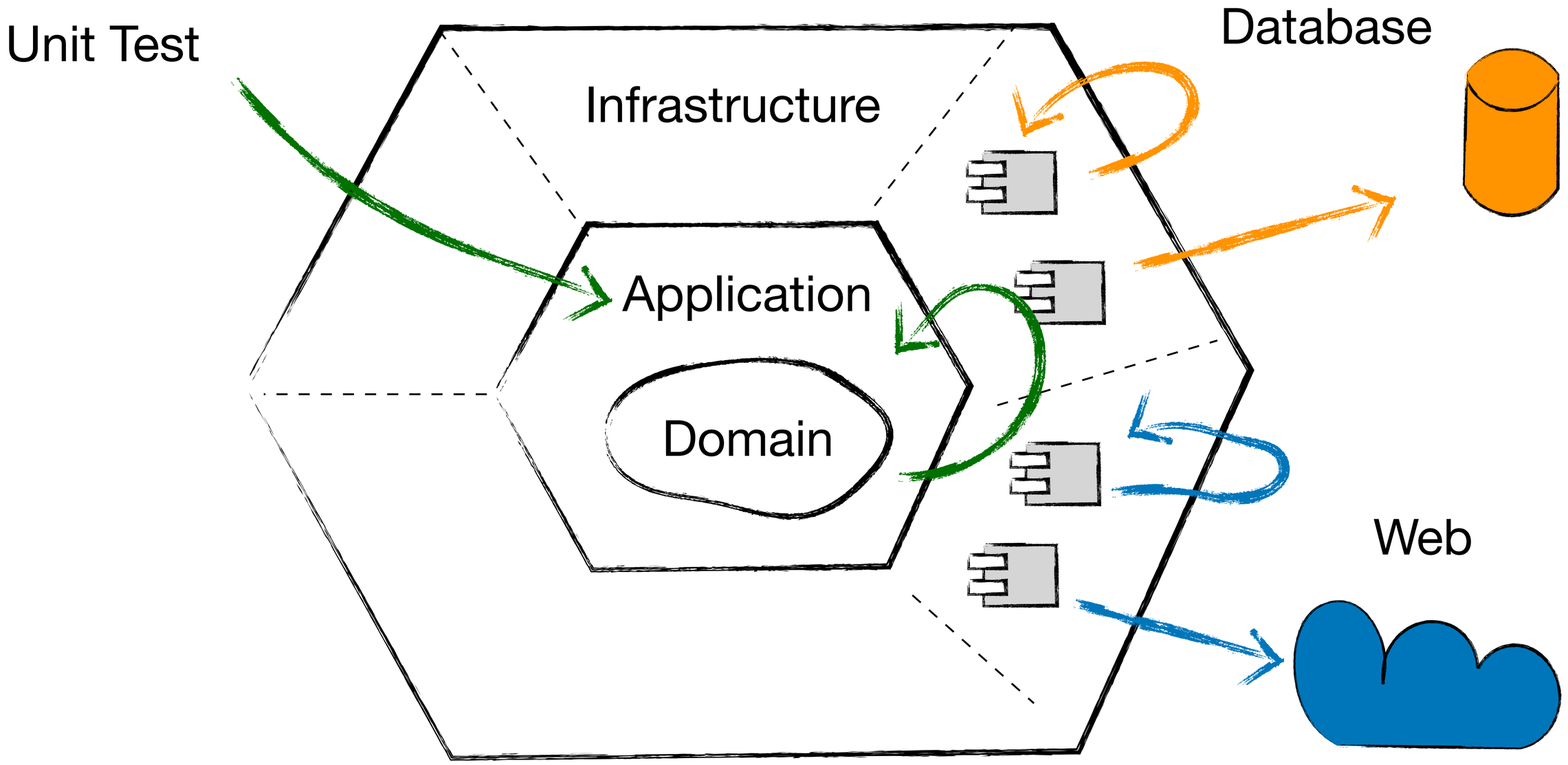
Ports and Adapters



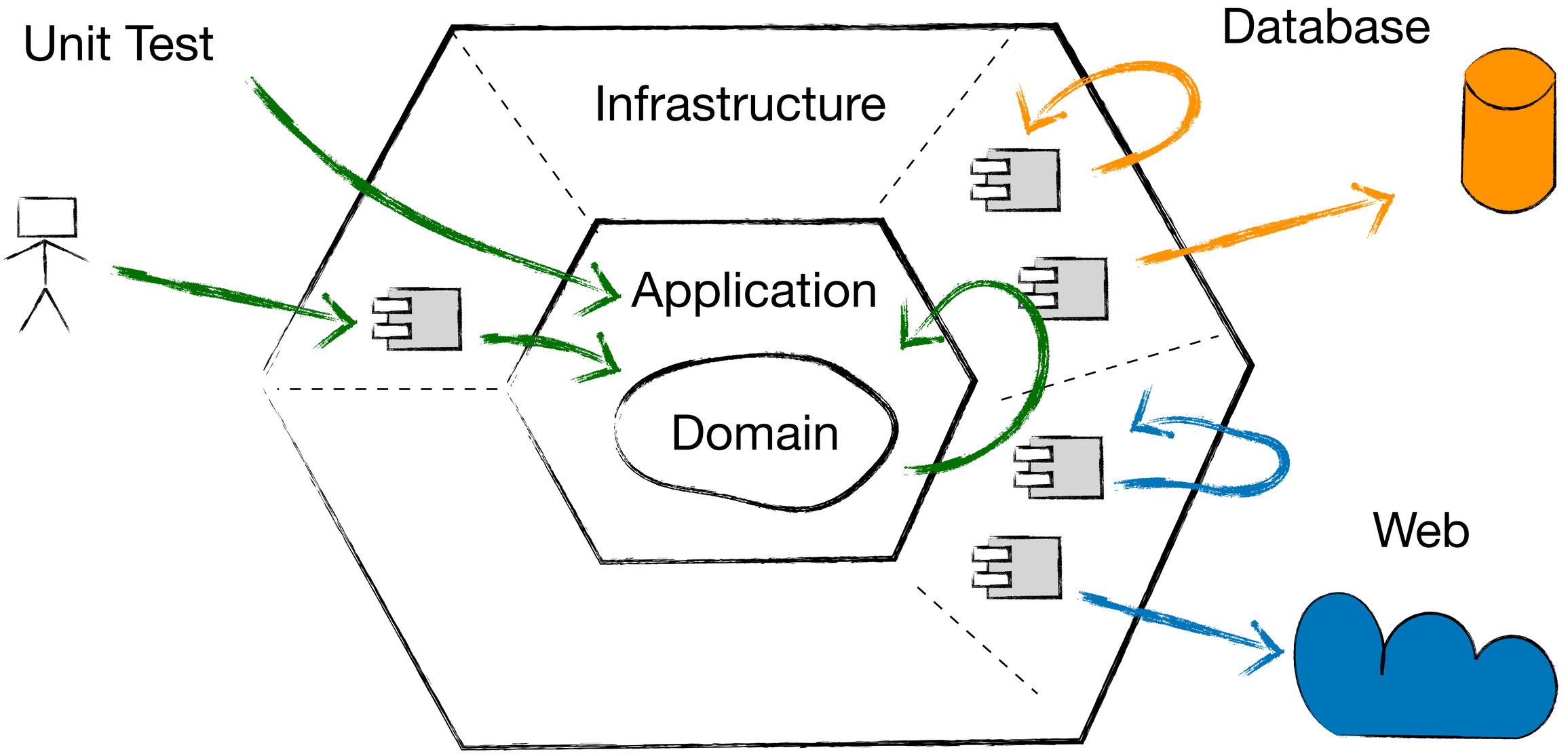
Ports and Adapters



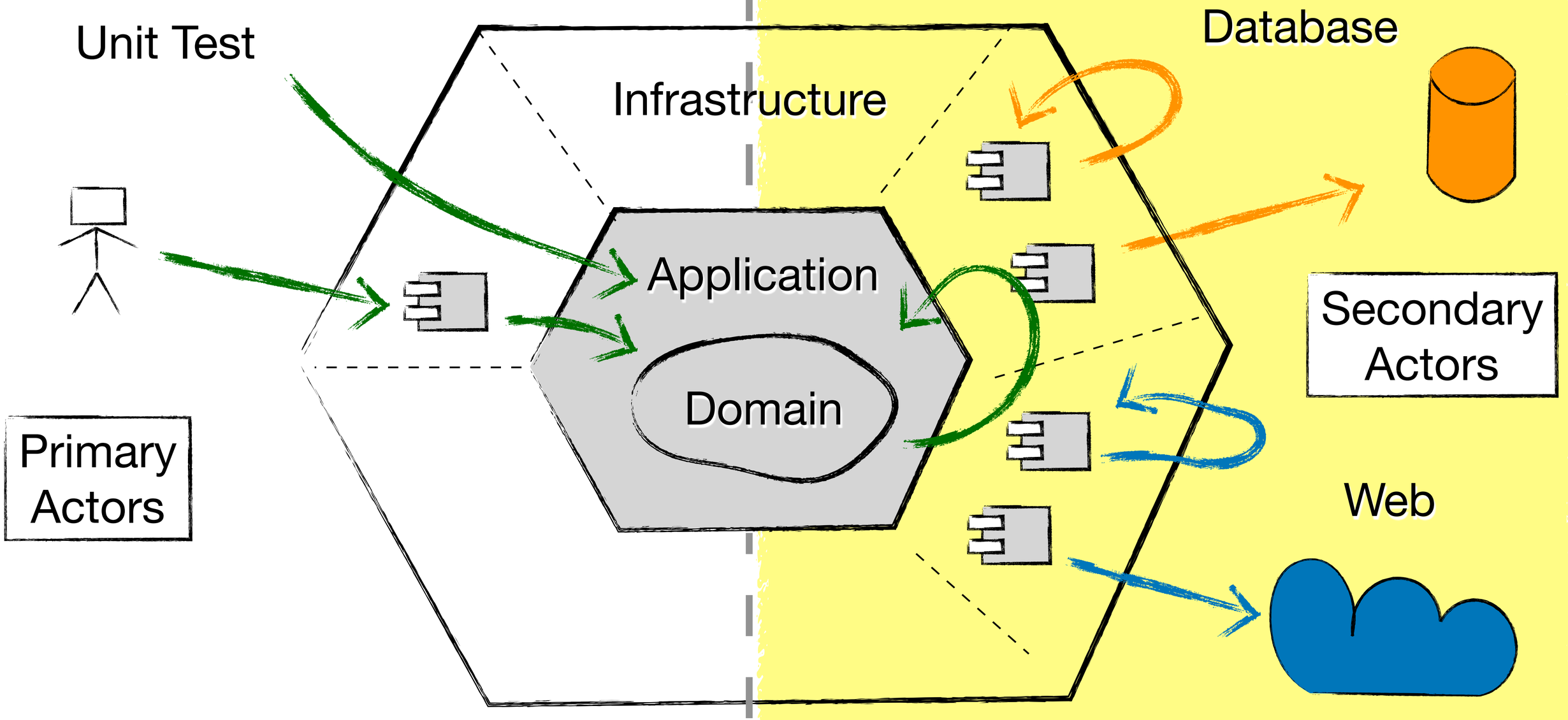
Ports and Adapters



Ports and Adapters



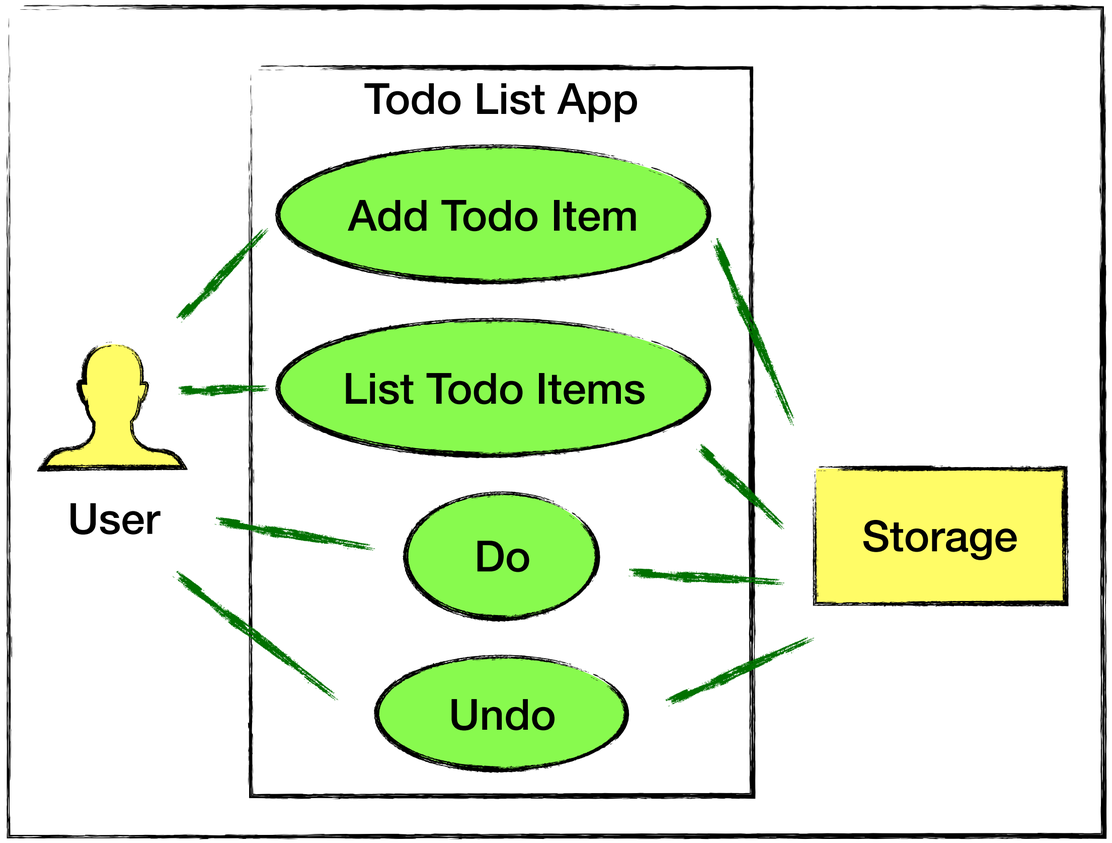
Ports and Adapters



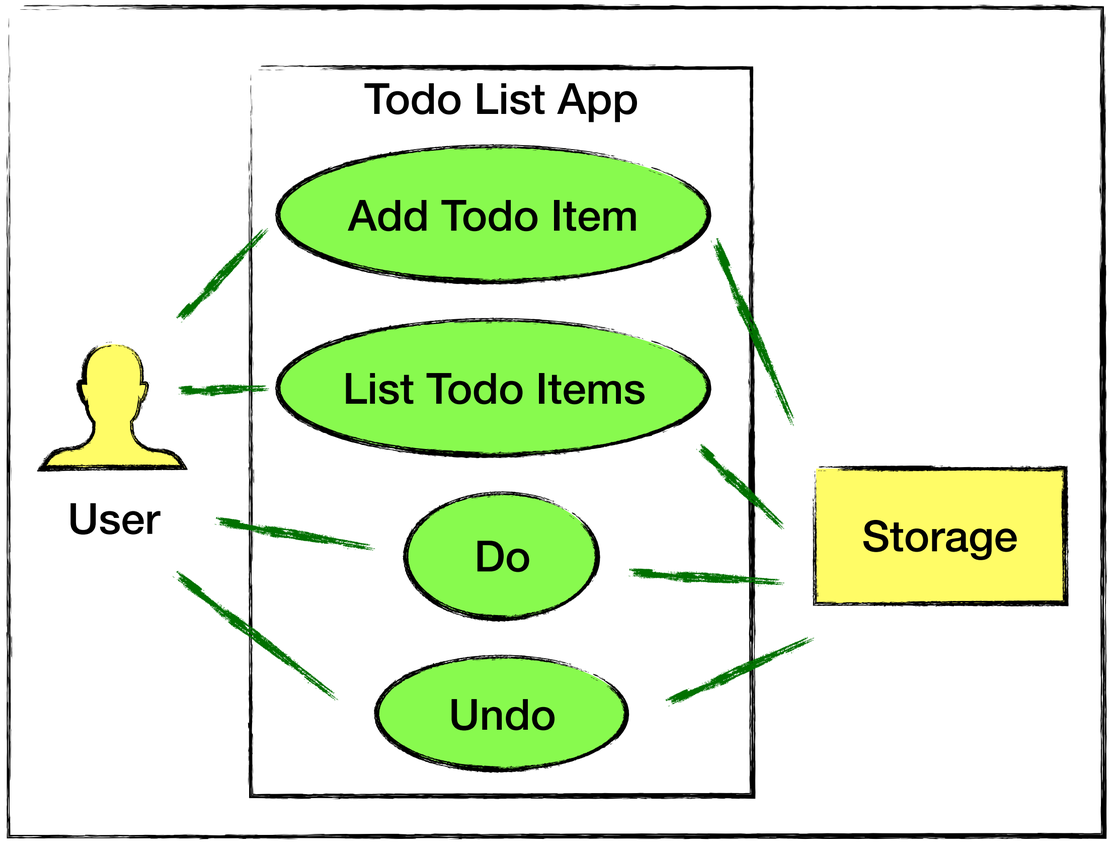
Evolutionary Architecture Definition:

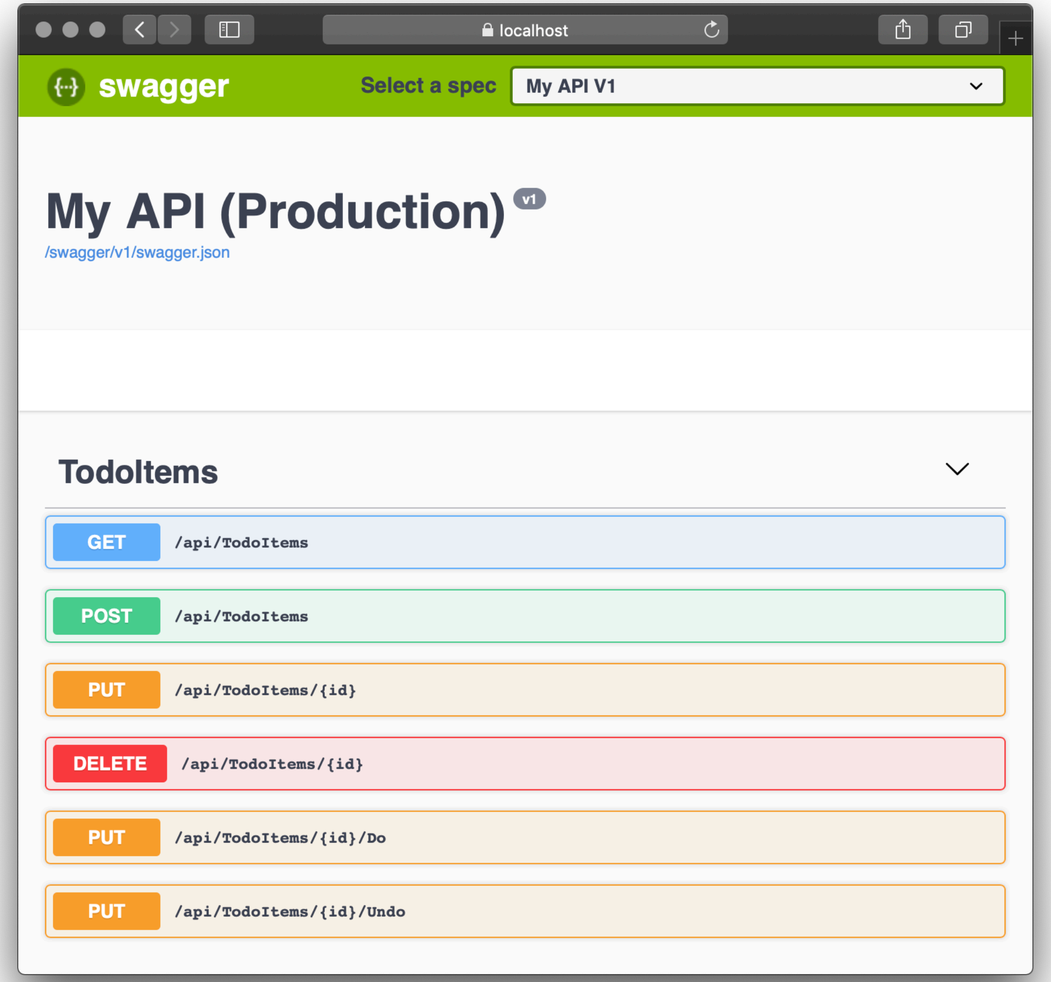
An evolutionary architecture supports guided, incremental changes as a first principle across multiple dimensions.

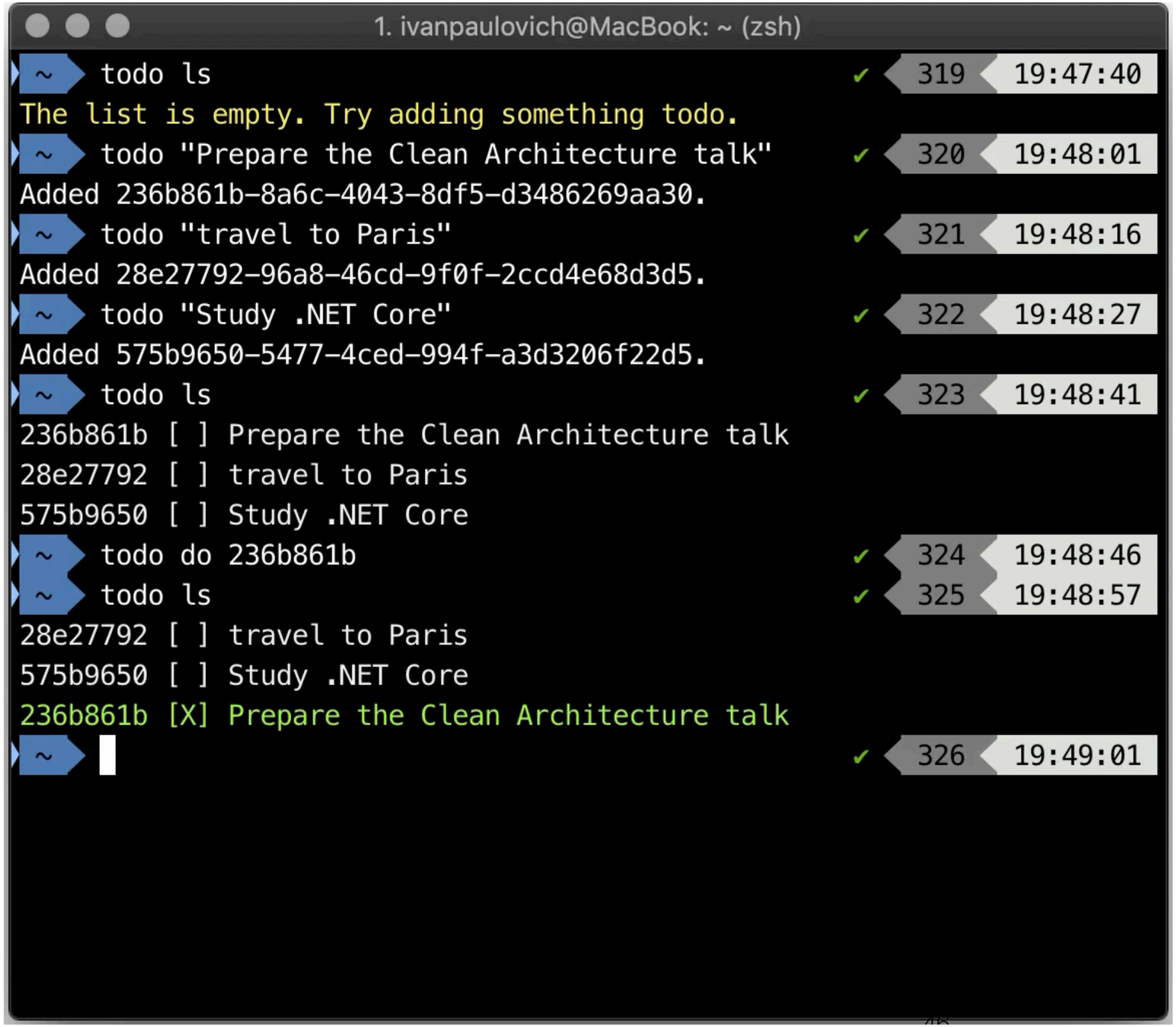
Sample Application



Sample Application







✅ Commad-Line Task management with storage on your GitHub 🔥

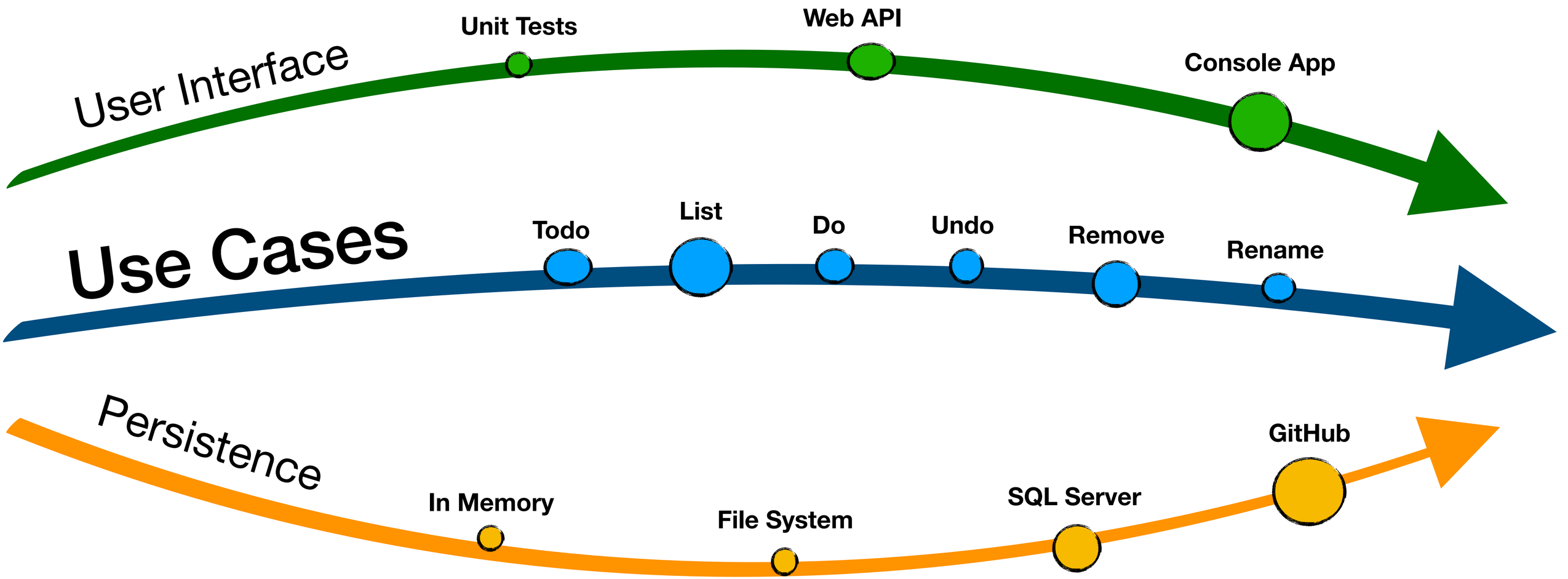
$ dotnet tool install -g todo

nuget v1.0.41 1.2k build passing

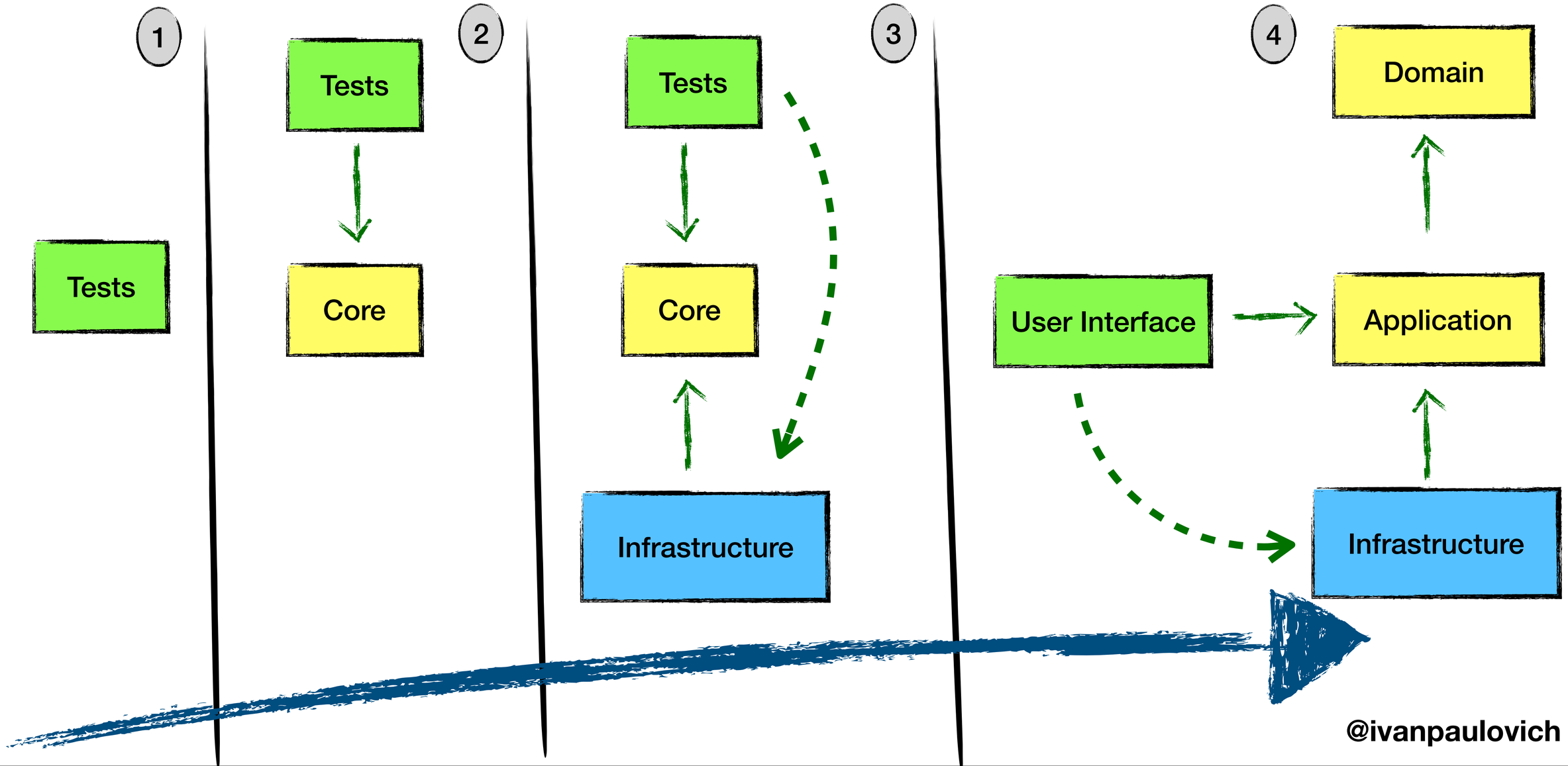
https://github.com/ivanpaulovich/todo

@ivanpaulovich

Evolutionary Architecture In Action



Packages on the software lifetime



Wrapping up Clean Architecture

- Clean Architecture is about usage and the use cases are the central organising principle.

Wrapping up Clean Architecture

- Clean Architecture is about usage and the use cases are the central organising principle.
- Use cases implementation are guided by tests.

Wrapping up Clean Architecture

- Clean Architecture is about usage and the use cases are the central organising principle.
- Use cases implementation are guided by tests.
- The User Interface and Persistence are designed to fulfil the core needs (not the opposite!).

Wrapping up Clean Architecture

- Clean Architecture is about usage and the use cases are the central organising principle.
- Use cases implementation are guided by tests.
- The User Interface and Persistence are designed to fulfil the core needs (not the opposite!).
- Defer decisions by implementing the simplest component first.

Fork me on GitHub

✓ coding the `todo` CLI tool

Ivan Paulovich
ivanpaulovich

Edit profile

Software craftsman and speaker. TDD, DDD and SOLID on daily basis. Interested in Event Sourcing, CQRS, Hexagonal, Clean Architecture, Docker and Cloud

Betsson Group
Stockholm, Sweden
ivan@paulovich.net
https://paulovich.net

Overview | Repositories 50 | Projects 0 | Stars 229 | Followers 275 | Following 241

Pinned — Customize your pins

todo
✅ Commad-Line Task management with storage on your GitHub 🔥
C# ★ 63 8

dotnet-new-caju
This dotnet-new template for .NET Back-ends increases productivity on building applications with the Hexagonal, Clean or Event Sourcing architectures styles. This tool generates a .NET back-end wit...
C# ★ 139 19

event-sourcing-jambo
An Hexagonal Architecture with DDD + Aggregates + Event Sourcing using .NET Core, Kafka e MongoDB (Blog Engine)
C# ★ 116 48

clean-architecture-manga
Clean Architecture service template for your Microservice with DDD, TDD and SOLID using .NET Core 2.0. The components are independent and testable, the architecture is evolutionary in multiple dime...
C# ★ 424 91

clean-architecture-webapi-ef-core
❤️ The simplest Clean Architecture demo on how to implement a Web Api using .NET Core and Entity Framework
C# ★ 43 6

hexagonal-architecture-acerola
An Hexagonal Architecture service template with DDD, CQRS, TDD and SOLID using .NET Core 2.0. All small features are testable and could be mocked. Adapters could be mocked or exchanged.
C# ★ 161 33

791 contributions in the last year — Contribution settings — 2019

Ask me 2 Questions!

betsson group

@ivanpaulovich