

# Clean Architecture in practice

Hannes Lowette

Axes\_

# Who am I?

- Father of Arne (12), Joren (8) and Marit (6)
- Partner of Barbara (?)
- Head of L&D @ Axxes
- .NET backend dev
- Loves knowledge sharing
- Amateur guitar builder
- Guitarist @  
Dylan Beattie & the Linebreakers
- Mountain biker
- Bad chess player
- Microsoft MVP



# Who are you?

- What's your name?
- What is your background?
- What are you hoping to learn?
- Tell me 1 cool fact about you!

... or if you don't feel like sharing:

*I'm X and I just want to learn.*

AXXES





# Disclaimer

I am here for you,  
not the other way around

This means:

- If you have questions, ask!
- If you feel we can do things better/differently, speak up!

The workshop is different every time

AXXES

# PSA

1. Need a break? Others probably need one too.  
→ Let me know! I tend to ramble on.
2. Learning is best when you feel comfortable.  
→ Eat snacks, drink, go to the bathroom, ...
3. I can talk for a day, but I get hoarse.  
→ Let's turn the coding into a conversation!  
→ You learn, I learn something too
4. Need me to commit and push?  
→ Shout out!

AXXES



# Agenda

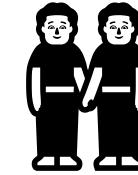
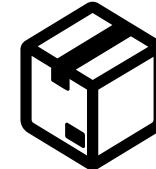
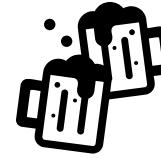
- 1 \_ What are we building?
- 2 \_ Clean Code & SOLID
- 3 \_ Clean Architecture
- 4 \_ Practical Implementation

# What are we building?

---

BeerSender.NET

# Lockdown boredom



AXXES

# Execution



AXXES

# Expectation



AXXES

# Reality

12/01/2022  
15:41

**Returned**  
Returned to shipper  
MOL, BE

11/01/2022  
19:07

**Returning to Sender**  
UPS initiated contact with receiver or importer for clearance information. Once received, UPS will submit for clearance. / The package will be returned to the sender.  
Lummen, Belgium

11/01/2022  
13:02  
UPS initiated contact with the sender to obtain clearance information. Once received, UPS will submit for clearance. / The information requested has been obtained and the hold has been resolved.  
Lummen, Belgium

10/01/2022  
22:22  
UPS initiated contact with the sender to obtain clearance information. Once received, UPS will submit for clearance.  
Lummen, Belgium

10/01/2022  
22:21  
A missing commercial invoice is causing a delay. We are currently waiting for information from the sender.  
Lummen, Belgium

10/01/2022  
22:20  
Warehouse Scan  
Lummen, Belgium

10/01/2022  
19:40  
Origin Scan  
Lummen, Belgium

10/01/2022  
14:00  
Collection Scan  
Lummen, Belgium

08/01/2022  
11:41  
Your parcel is currently at the UPS Access Point™ and is scheduled to be tendered to UPS.  
Lummen, Belgium

08/01/2022  
11:41  
Drop-Off  
Lummen, Belgium

07/01/2022  
22:21  
Your parcel is pending release from a Government Agency. Once they release it, your parcel will be on its way.  
Lummen, Belgium

07/01/2022  
22:21  
Your parcel is on the way

07/01/2022  
22:20  
Shipper created a label, UPS has not received the package yet.  
Belgium





# Ended up being

- Tracking
- Providing extra customs info
- Emails
- UPS helpdesk ☹
- Packages returned
- Recuperating shipping costs
- Re-sending
- No delivery attempts
- Re-re-sending

AXXES

# Sending beer is tricky

- Making & filling boxes is fun
- The rest, not so much
- Follow-up is key!

→ We need an app for that!

AXXES



# BeerSender.NET

---

Because it's not written in VB6!

# Clean Code & SOLID

---

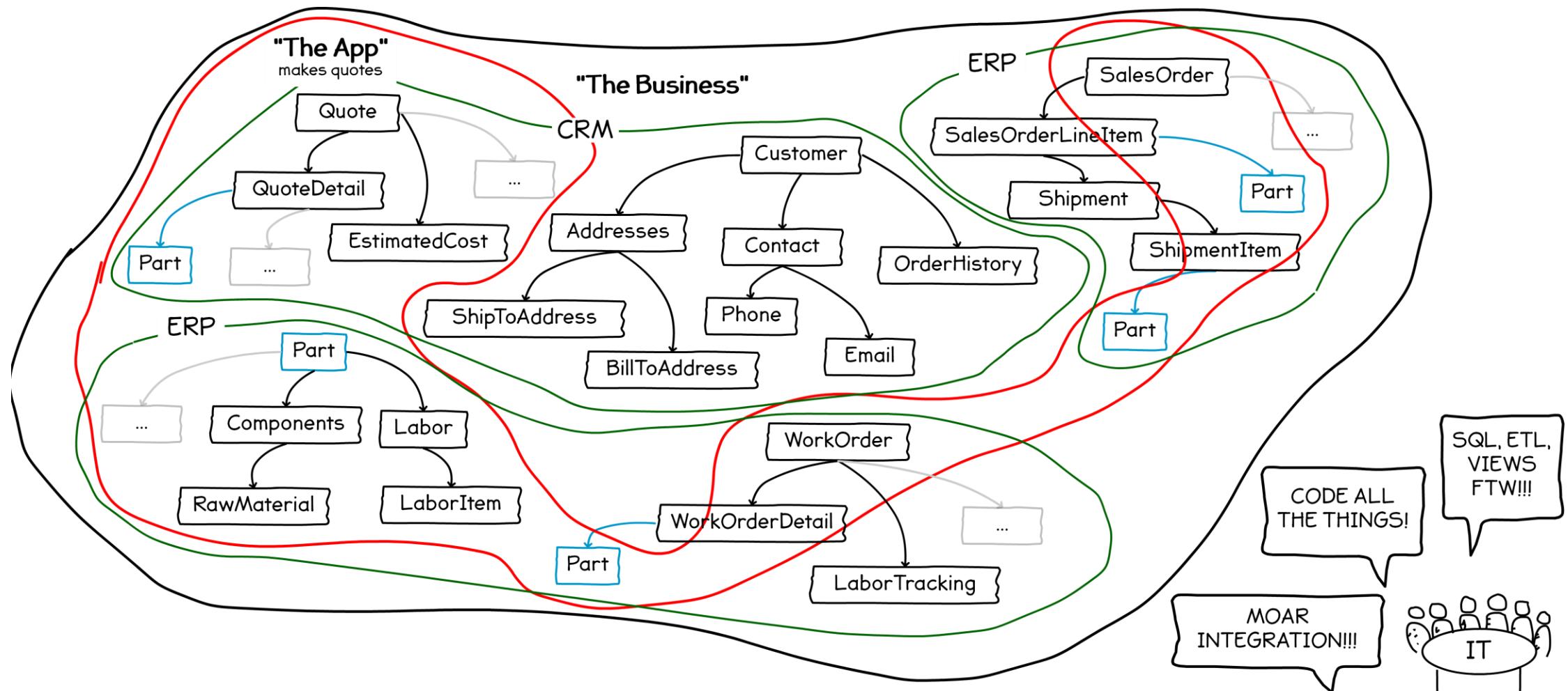
Uncle Bob's legacy

# An OO dev's evolution

---

As perceived by me

# Step 1: Demoware



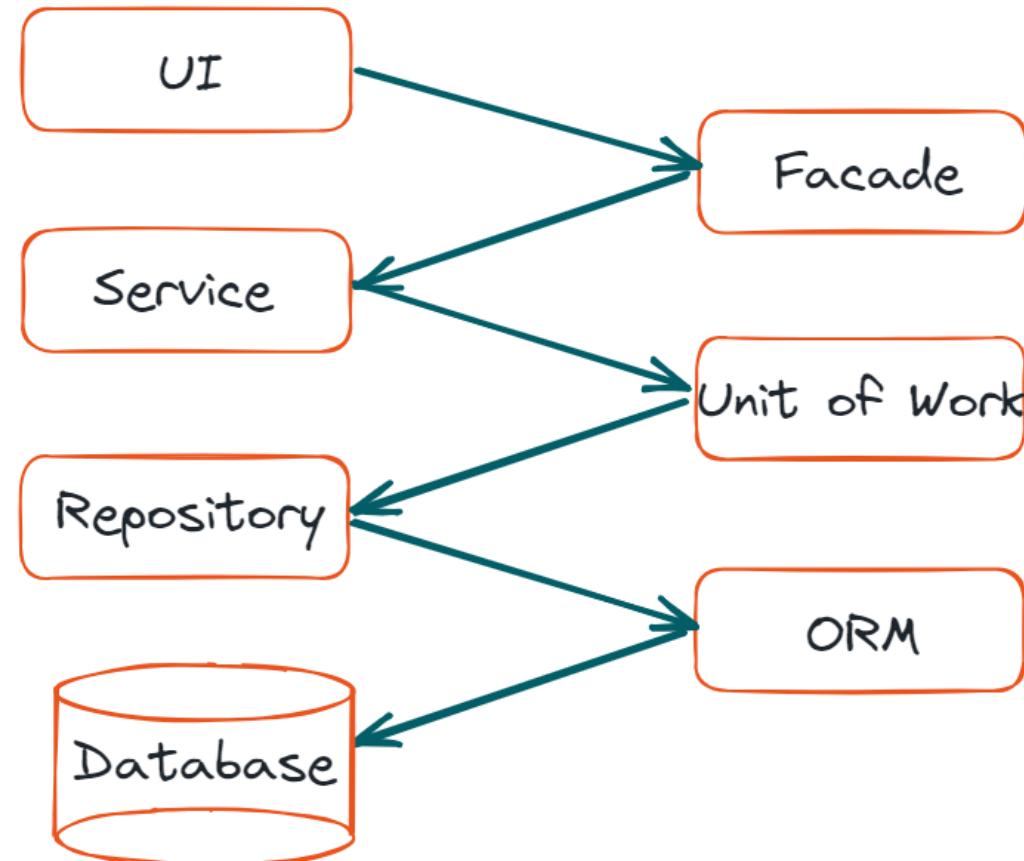
AXXES

## Step 2: Layers



AXXES

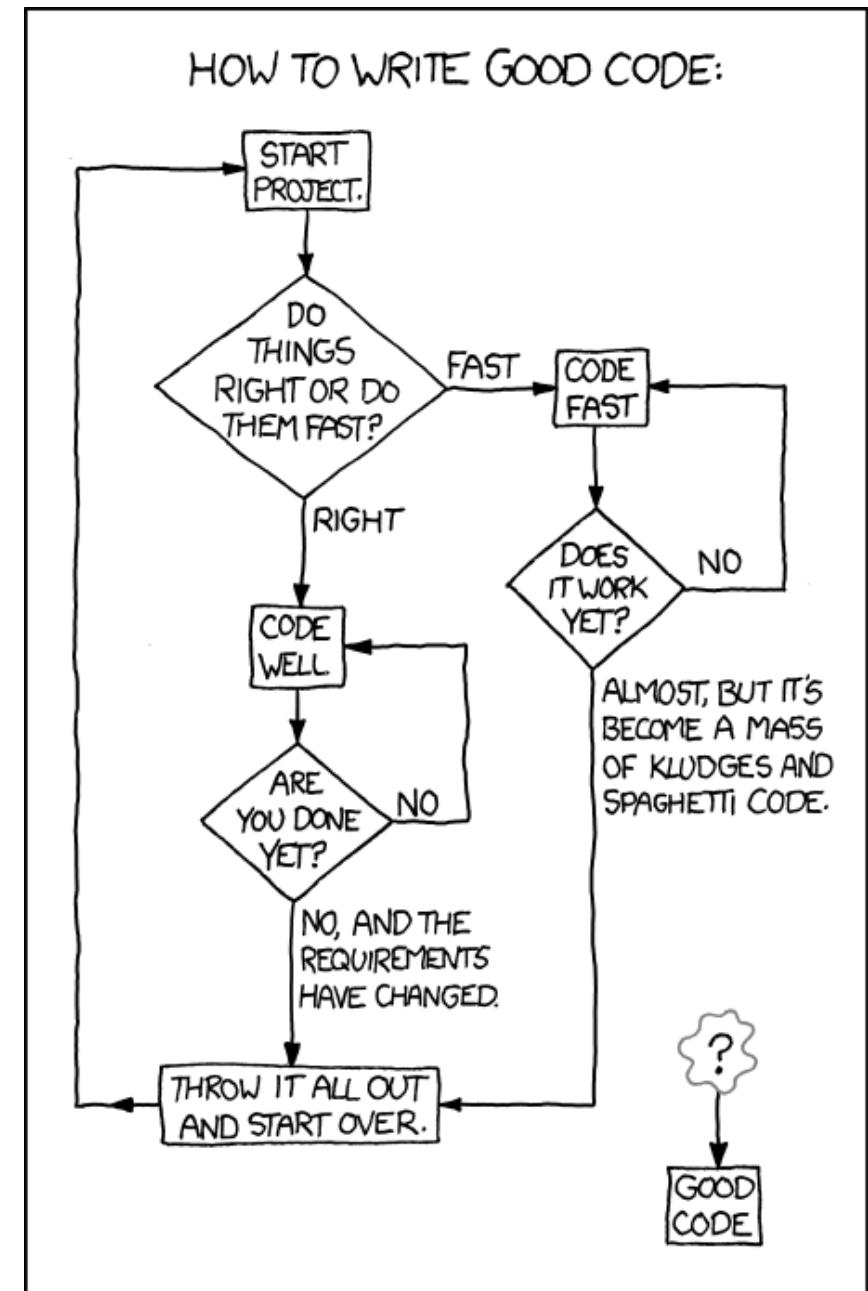
## Step 3: “SOLID”



# Step 4: SOLID

- Single Responsibility
- Open/closed
- Liskov Substitution
- Interface Segregation
- Dependency Inversion

Axes



# Step 5: Architecture

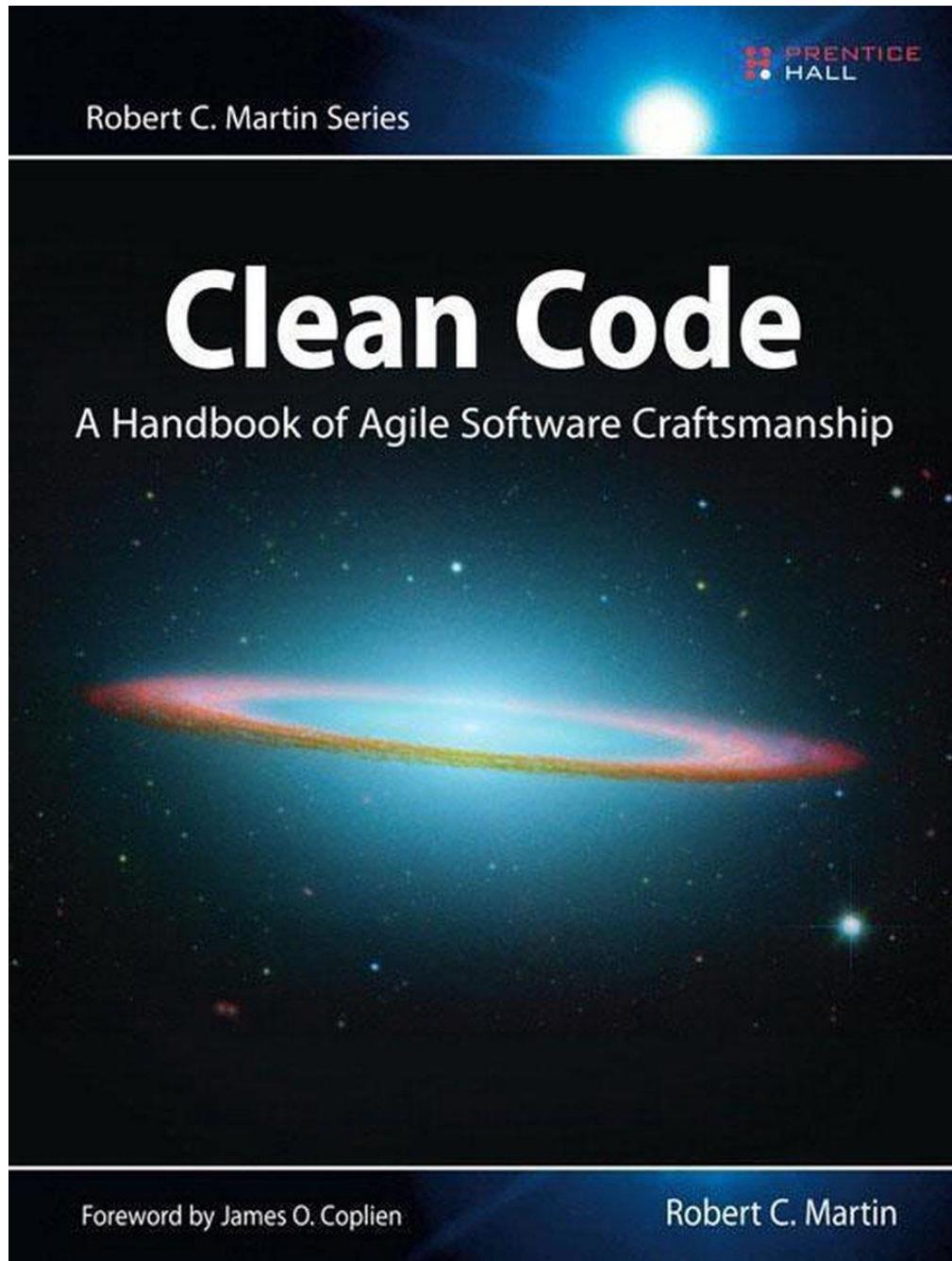
- Clean Architecture
- Ports & Adapters
- SOA
- Plugins
- Bus Systems
- Microservices
- ...



# Clean Code

---

How to structure code to be readable



# The book

- Naming things
- Structuring functions & flow
- Comments
- Formatting
- Objects & Data structures
- Unit tests
- Classes (incl. SOLID)
- ...

AXXES

# Reading vs. writing code

- More time spent
- Harder to do
  - Other developer's code
  - Your own code
- Risk when misunderstanding:
  - Regressions
  - Time lost
- Written once, read multiple times

→ Optimize for readability

AXXES\_





# Goals for code

- Easy to understand
- Reads like a story
- Expresses intent
- Comments only when needed
- **Naming things!**
- Easy to test
- Easy to refactor
- Easy to reuse

AXXES

# SOLID

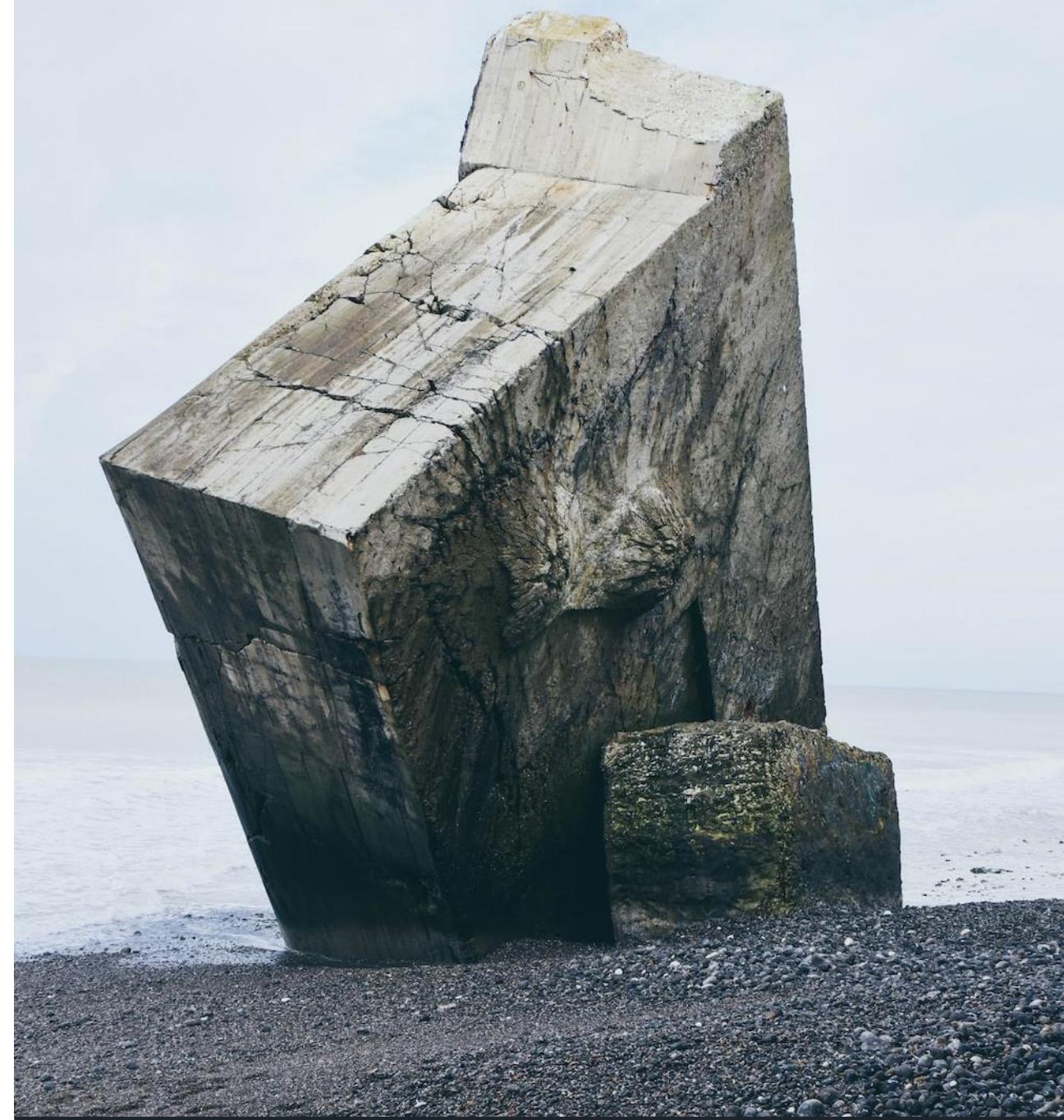
---

A model for OO development

# 5 principles

- Single Responsibility principle
- Open/closed principle
- Liskov Substitution principle
- Interface Segregation principle
- Dependency Inversion principle

AXXES\_





```
// Bad:  
class BeerPackage {  
    public int Capacity { get; set; }  
  
    public bool TryAddBottle(Beer bottle)  
    { ... }  
  
    public static BeerPackage GetFromDatabase(int id)  
    { ... }  
  
    public void SaveToDatabase()  
    { ... }  
}  
  
// Better:  
class BeerPackage {  
    public int Capacity { get; set; }  
}  
  
class BottleAdder {  
    public bool TryAddBottle(BeerPackage package, Beer bottle)  
    { ... }  
}  
  
class PackageRepository {  
    public BeerPackage GetFromDatabase(int id)  
    { ... }  
  
    public void SaveToDatabase(BeerPackage package)  
    { ... }  
}
```

# Single responsibility

A class has a single responsibility

- A class has 1 reason to change:
  - Database schema changed
  - A specific bit of logic has changed
  - Serialization changed
  - API contract changed
  - ...

# Open/closed

Software (entities, classes, functions) should be open for extension, but closed for modification

→ Use inheritance & composition



```
// Bad:  
class Beer {  
    public string Type { get; set; }  
  
    public void Package()  
    {  
        if(Type == "bottle"){  
            AddBubbleWrap();  
        } else if(Type == "can")  
            AddWrappingPaper();  
        } else {  
            throw new Exception("unknown type");  
        }  
    }  
  
// Better:  
abstract class Beer {  
    public abstract void Package();  
}  
  
class BeerBottle : Beer {  
    public override void Package(){  
        AddBubbleWrap();  
    }  
}  
  
class BeerCan : Beer {  
    public override void Package(){  
        AddWrappingPaper();  
    }  
}
```

# Liskov Substitution

Functions using a base class must be able to use objects of derived classes without knowing.

- Subclasses should not break baseclass behavior
- Mostly enforced by modern OO languages

Violations in C#:

- Throwing in implementations/overrides
- Hiding virtual members with new
- Returning values with limitations (e.g. immutable)
- Violate code policy in derived implementation (e.g. null)



```
abstract class Beer {  
    public abstract void Package();  
}  
  
class BeerBottle : Beer {  
    public override void Package(){  
        AddBubbleWrap();  
    }  
}  
  
// Bad  
class BeerCan : Beer {  
    public override void Package(){  
        throw new Exception("no packaging");  
    }  
}
```

# Interface Segregation

Clients should not be forced to depend on interfaces they don't use.

→ Group interface members by behavior, not implementation

→ It's better to have multiple smaller interfaces than a single larger one

TIP: interfaces are owned by the consumer

AXES\_



// Bad:

```
interface IBeerBottle {
    int GetContentInMilliliter();
    decimal GetAlcoholPercentage();
    bool HasScrewCap();
}
```

```
interface IBeerCan {
    int GetContentInMilliliter();
    decimal GetAlcoholPercentage();
    bool HasPullOffTab();
}
```

// Better:

```
interface IBeer {
    int GetContentInMilliliter();
    decimal GetAlcoholPercentage();
}
```

```
interface IHasCap {
    bool HasScrewCap();
}
```

```
interface IHasTab {
    bool HasPullOffTab();
}
```



```
// The behavior BeerAdder needs:  
interface IPackageRepository {  
    BeerPackage GetById(int id);  
    SavePackage(BeerPackage package);  
}  
  
class BeerAdder {  
    // Depends only on the abstraction:  
    private readonly IPackageRepository _repo;  
  
    // DI framework delivers implementation:  
    public BeerAdder(IPackageRepository repo) {  
        _repo = repo;  
    }  
  
    public void AddBottleToPackage(  
        int packageId, Beer bottle)  
    {  
        var package = _repo.GetById(packageId);  
        package.AddBottle(bottle);  
        _repo.SavePackage(package)  
    }  
  
    // BeerAdder doesn't know about implementation:  
    class PackageRepository: IPackageRepository {  
        public BeerPackage GetById(int id)  
        { ... }  
  
        public SavePackage(BeerPackage package)  
        { ... }  
    }
```

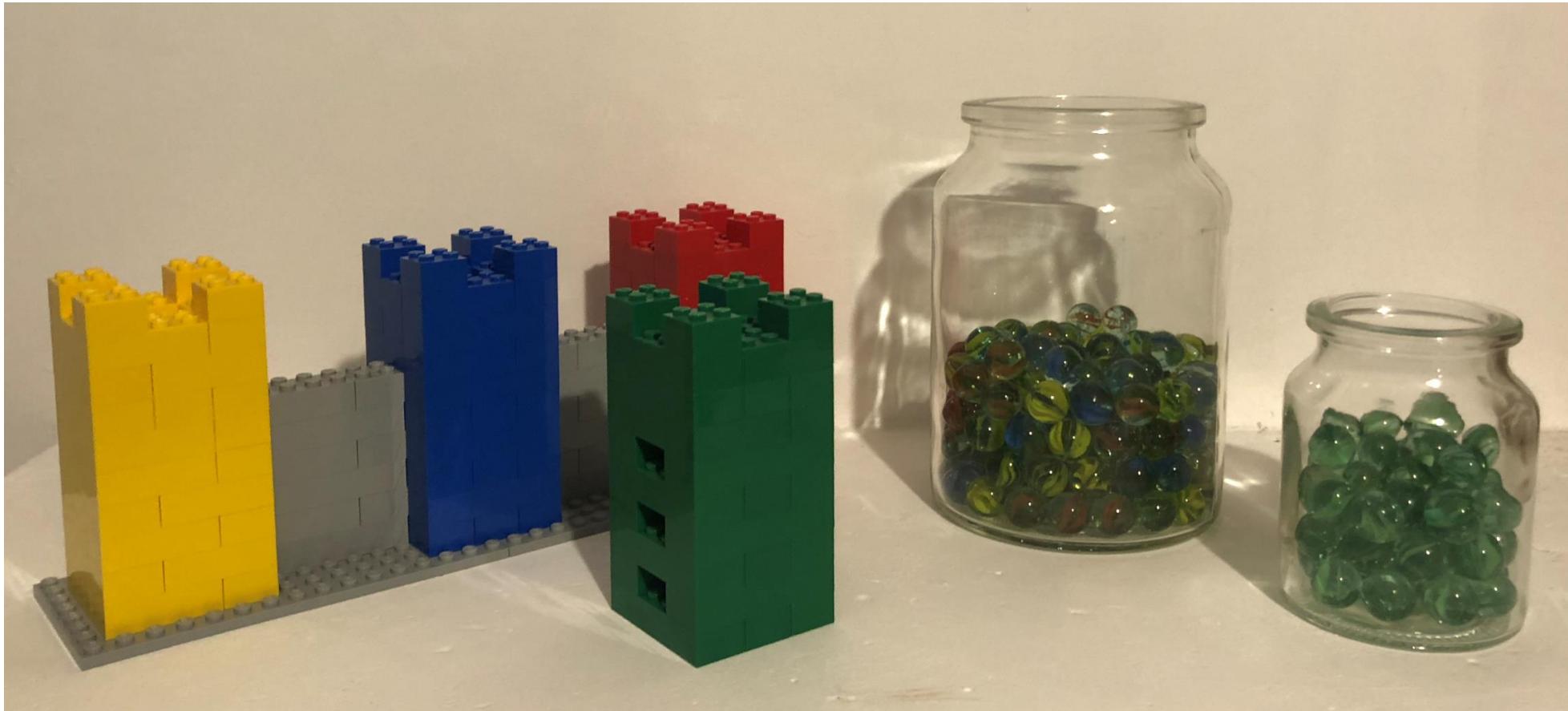
# Dependency Inversion

Abstractions should not depend upon details.  
Details should depend upon abstractions.

- Write code that consumes interfaces
- Define behavior, then implement
- Invert references (implementations reference abstractions)

AGAIN: interfaces are owned by the consumer

# Proper SOLID



AXXES

# Clean Architecture

---

SOLID – facilitating architecture

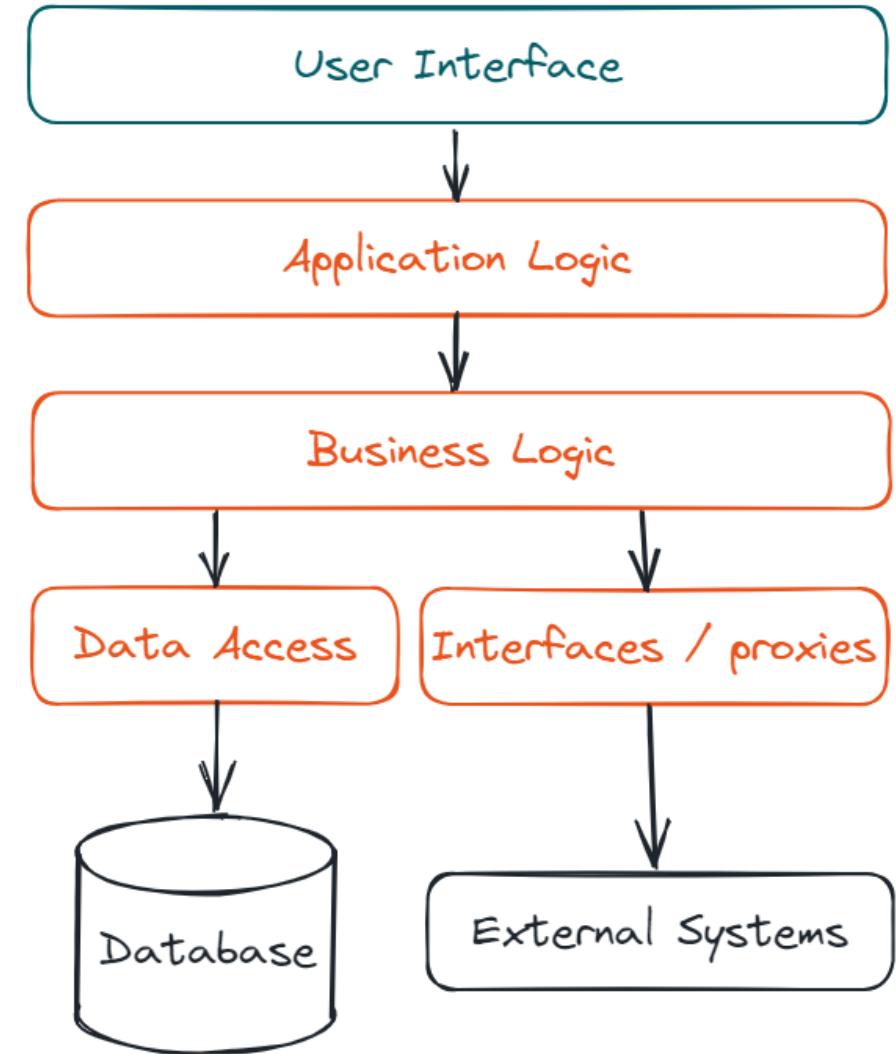
# What I learned in college

---

(and is still being taught)

# Layered Architecture

- Dependencies point 'downwards'
  - Replicates the path from UI to DB
  - Layers have 'knowledge' of what's 'below'
  - Separately deployed layers= tiers
- Matches up-front design approach



# A different way of thinking

---

The software business evolved

# Evolution

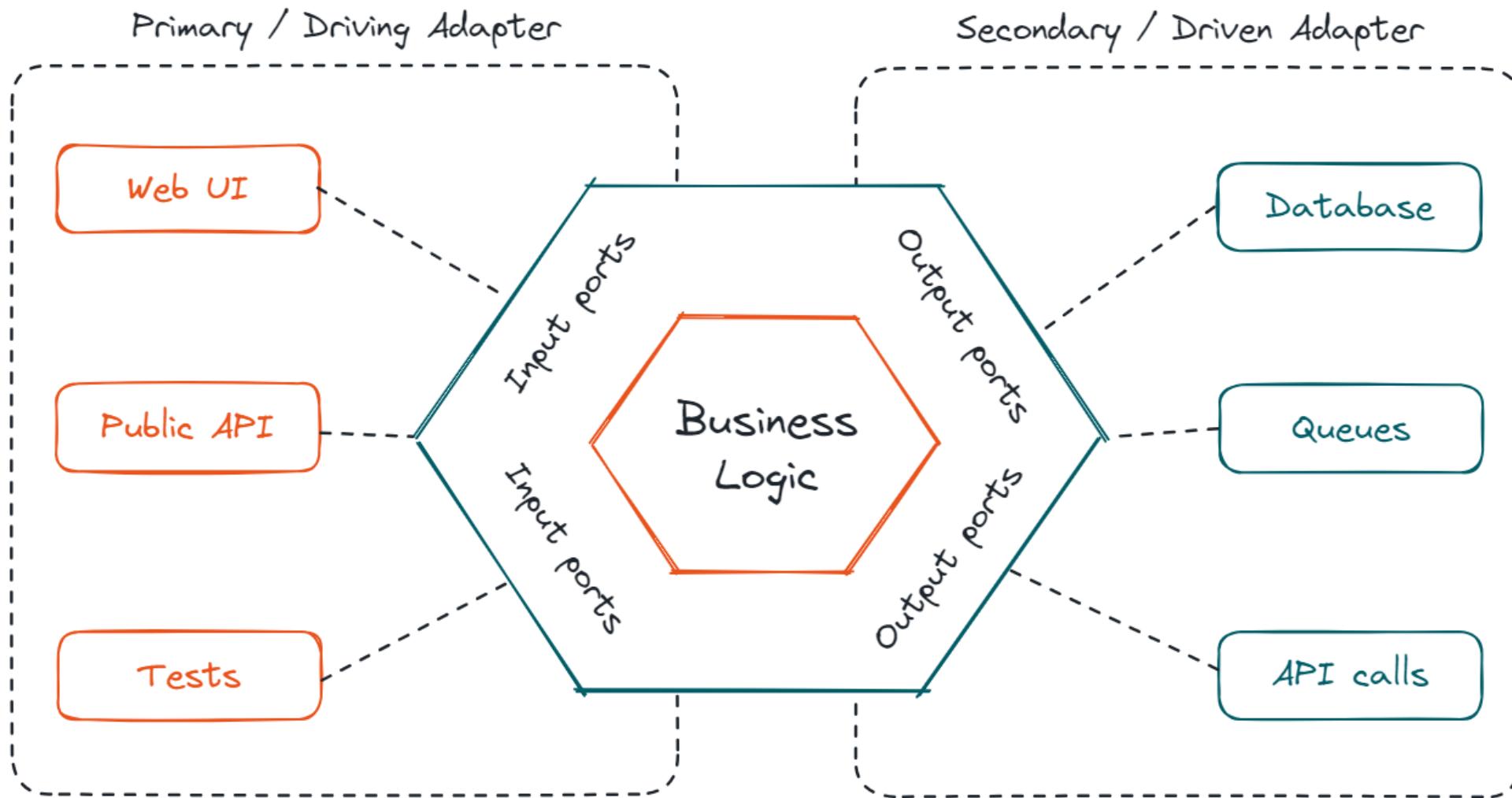
Long-lived software projects led to:

- Agile manifesto
- Continuous value delivery
- Changing requirements
- ...

→ Up-front architecture fails

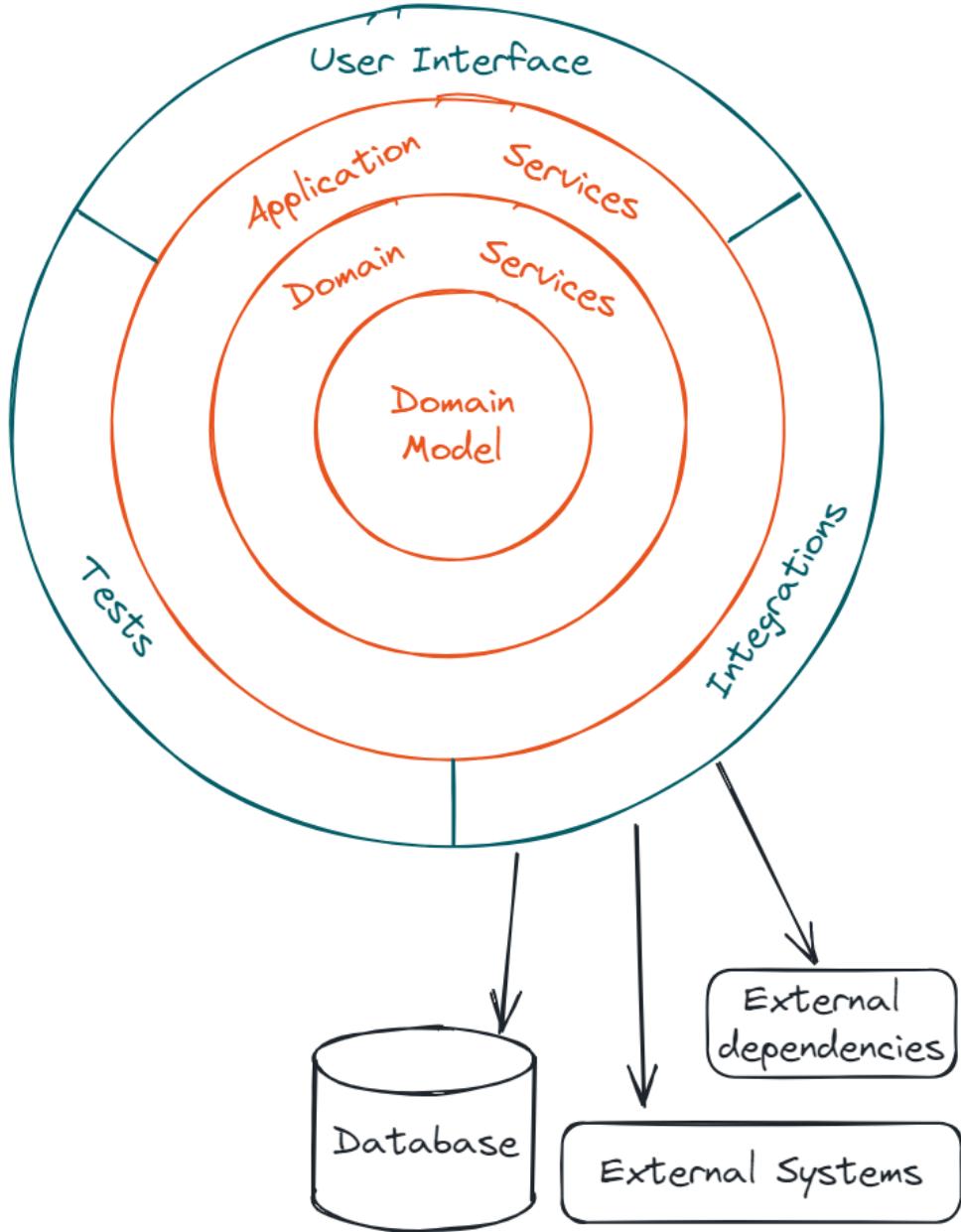


# Hexagon architecture / ports & adapters



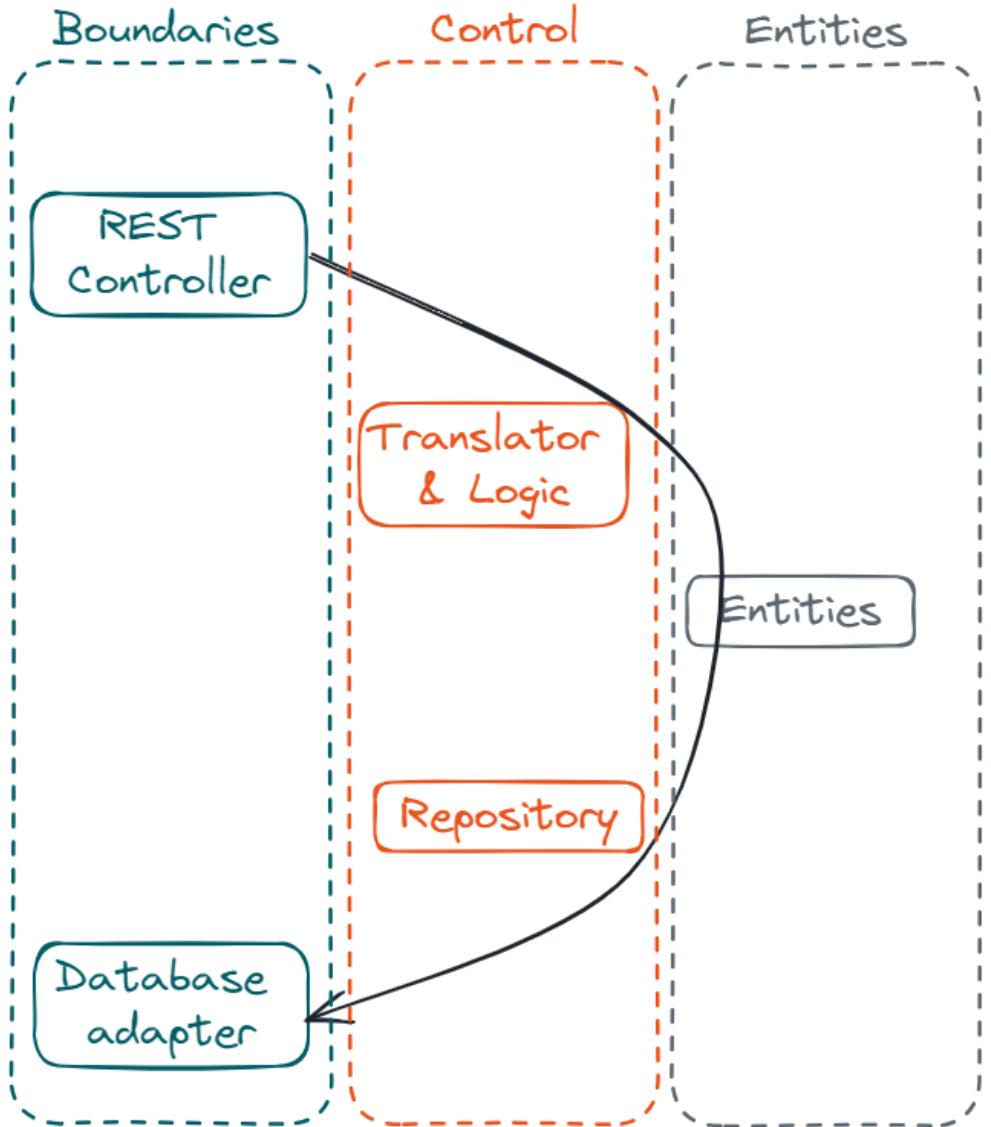
# Hexagon architecture / ports & adapters

- 2005
- Alistair Cockburn
- Avoids pitfalls in object-oriented software:
  - Undesired dependencies
  - Contaminations of UI with business logic
- Port: integration point for the core application
- Adapter: an implementation/consumer of the port



# Onion Architecture

- 2008
- Jeffrey Palermo
- Focus on limiting coupling

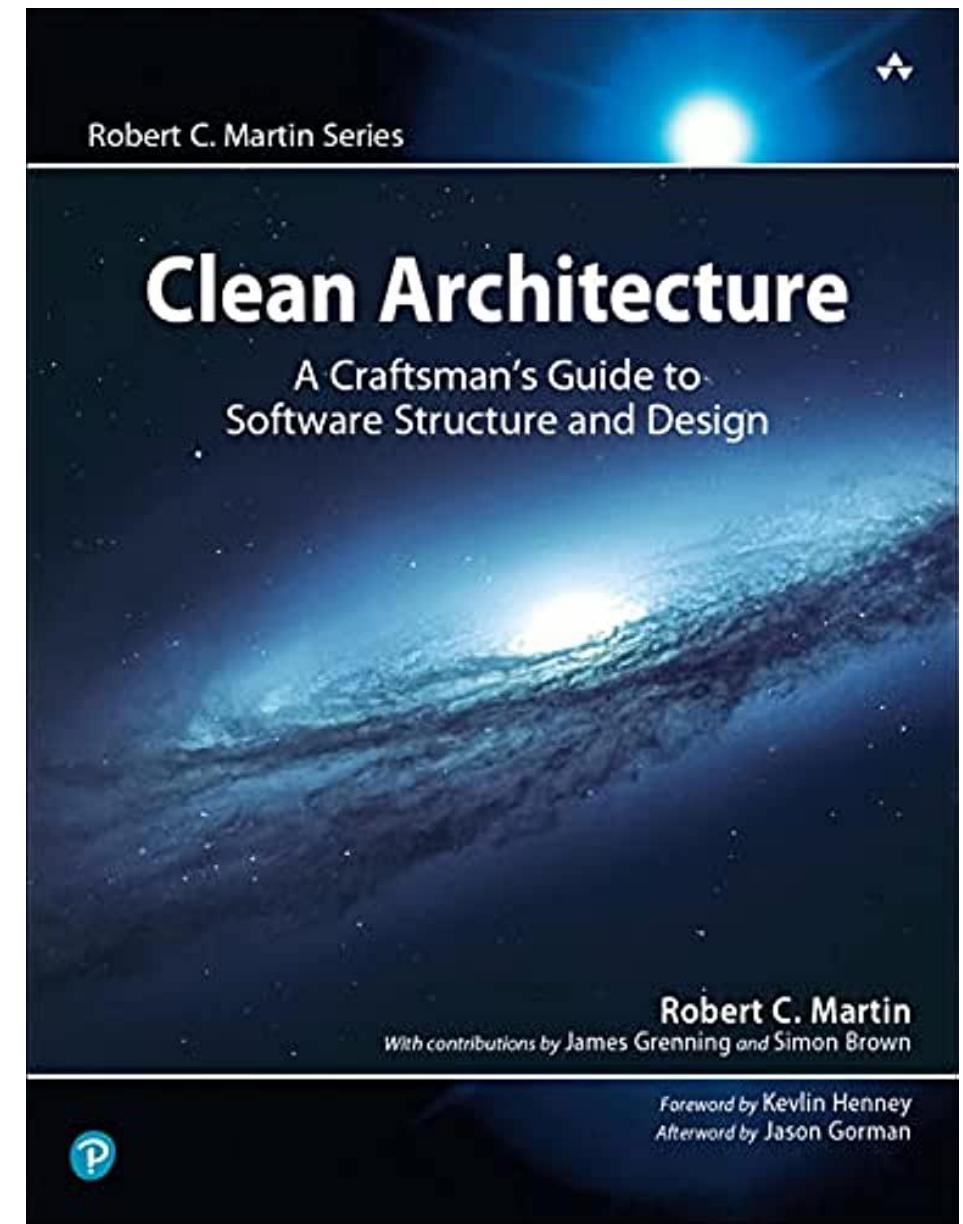


## Boundary – Control - Entity

- 1992
- Ivar Jacobson
- Organization of classes by responsibility

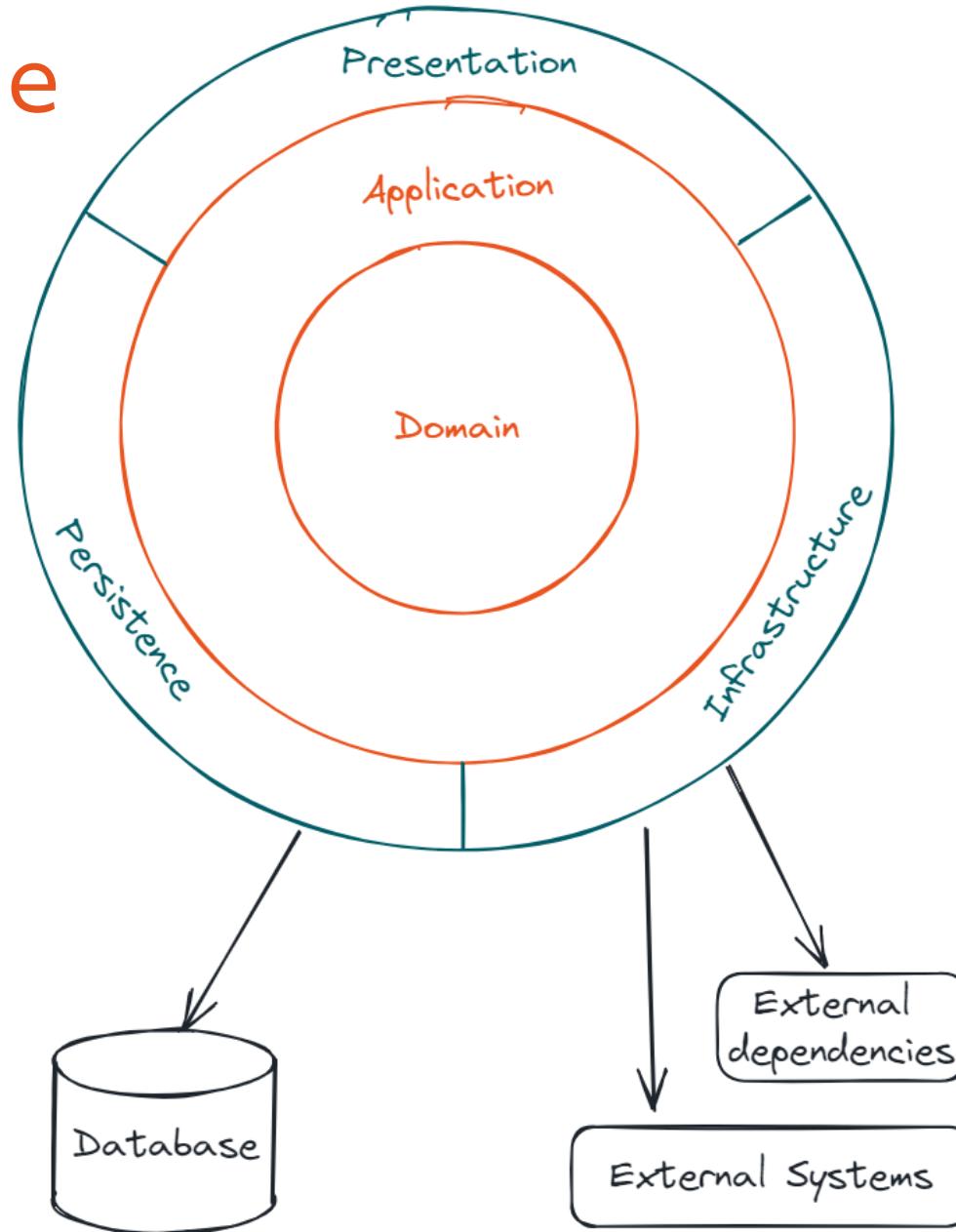
# Clean Architecture

- Uncle Bob
- 2017
- All the above

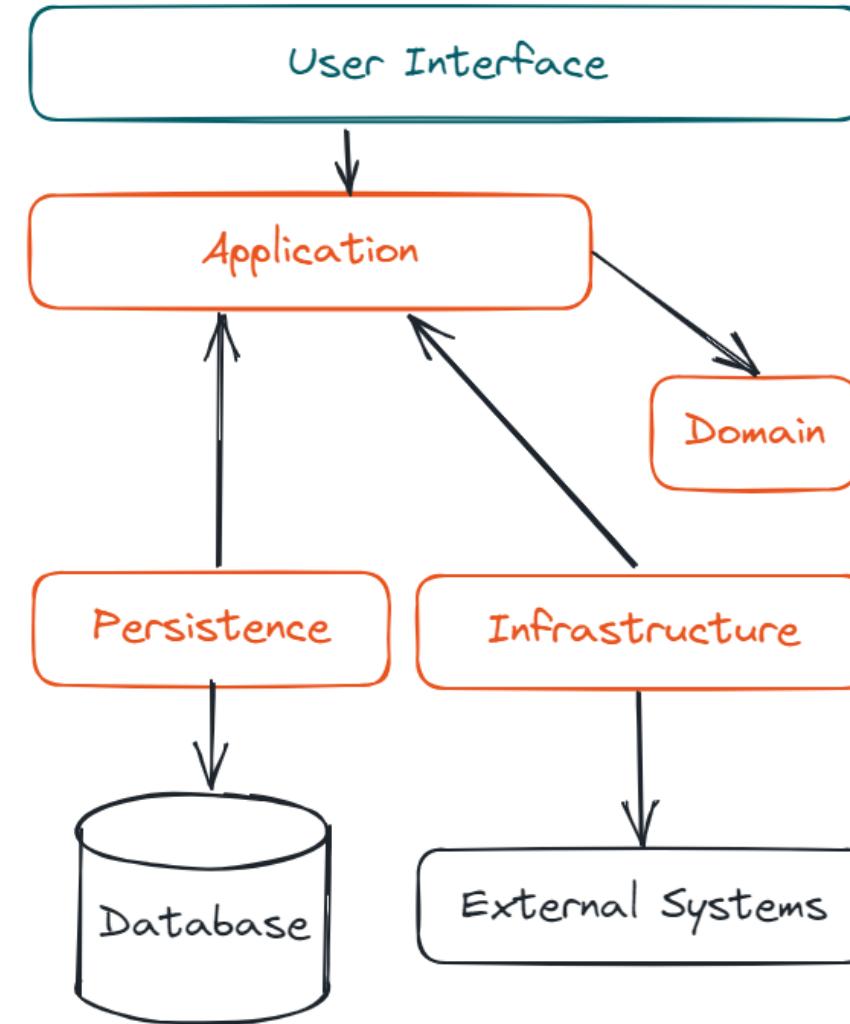
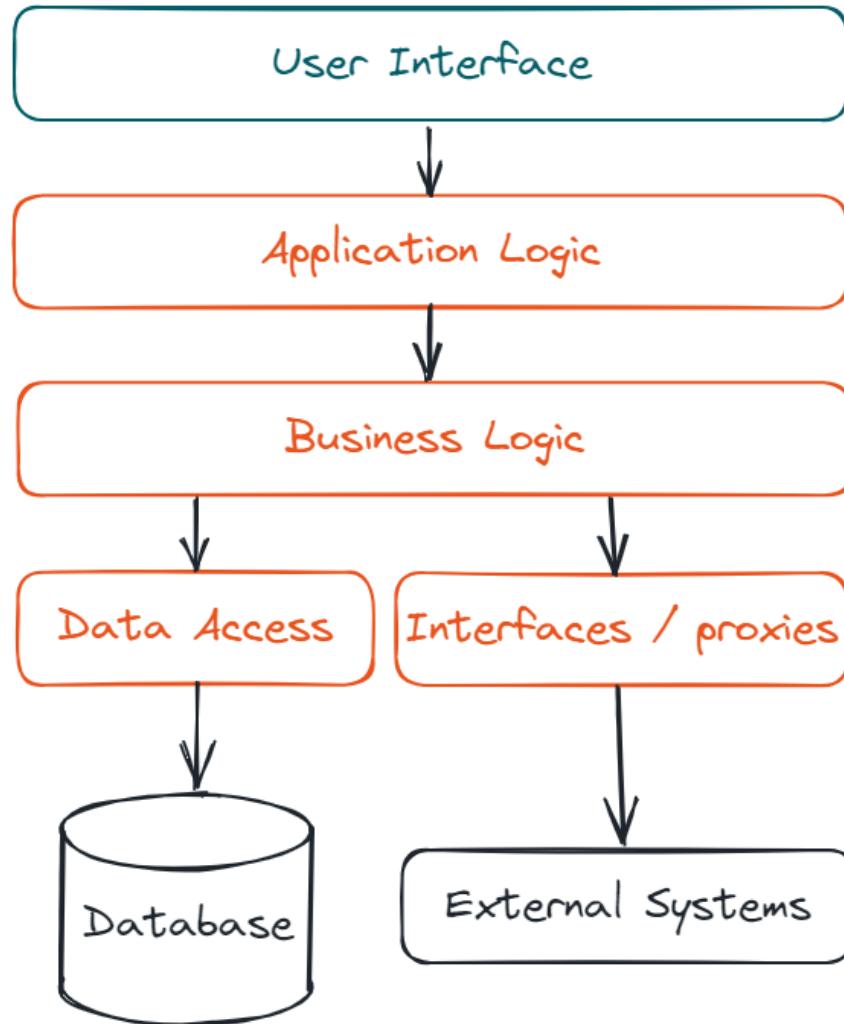


AXXES

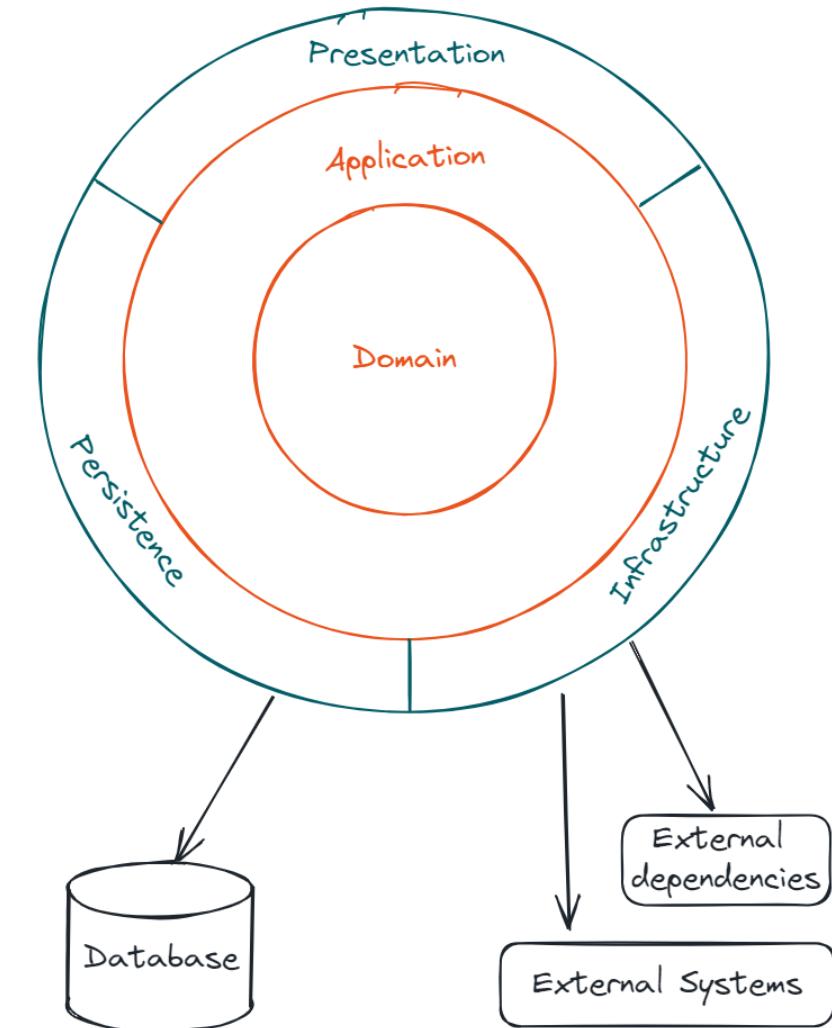
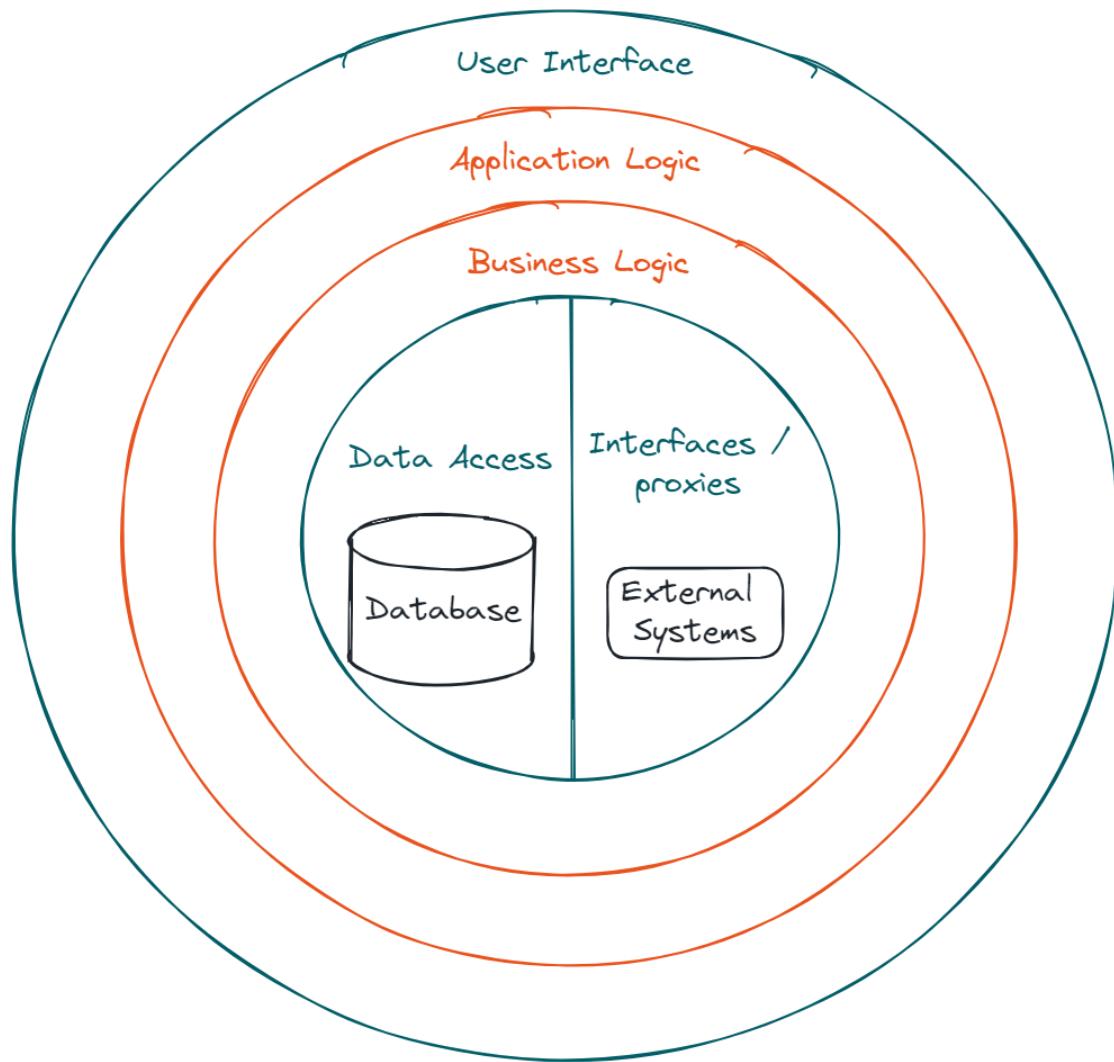
# Clean Architecture



# What has changed?



# What has changed?



AXXES

# Domain-centric architecture

- Domain doesn't have dependencies
- Application logic (use cases) depends only on domain
- Implementations & integrations are handled by DI

→ Focus on domain & core logic

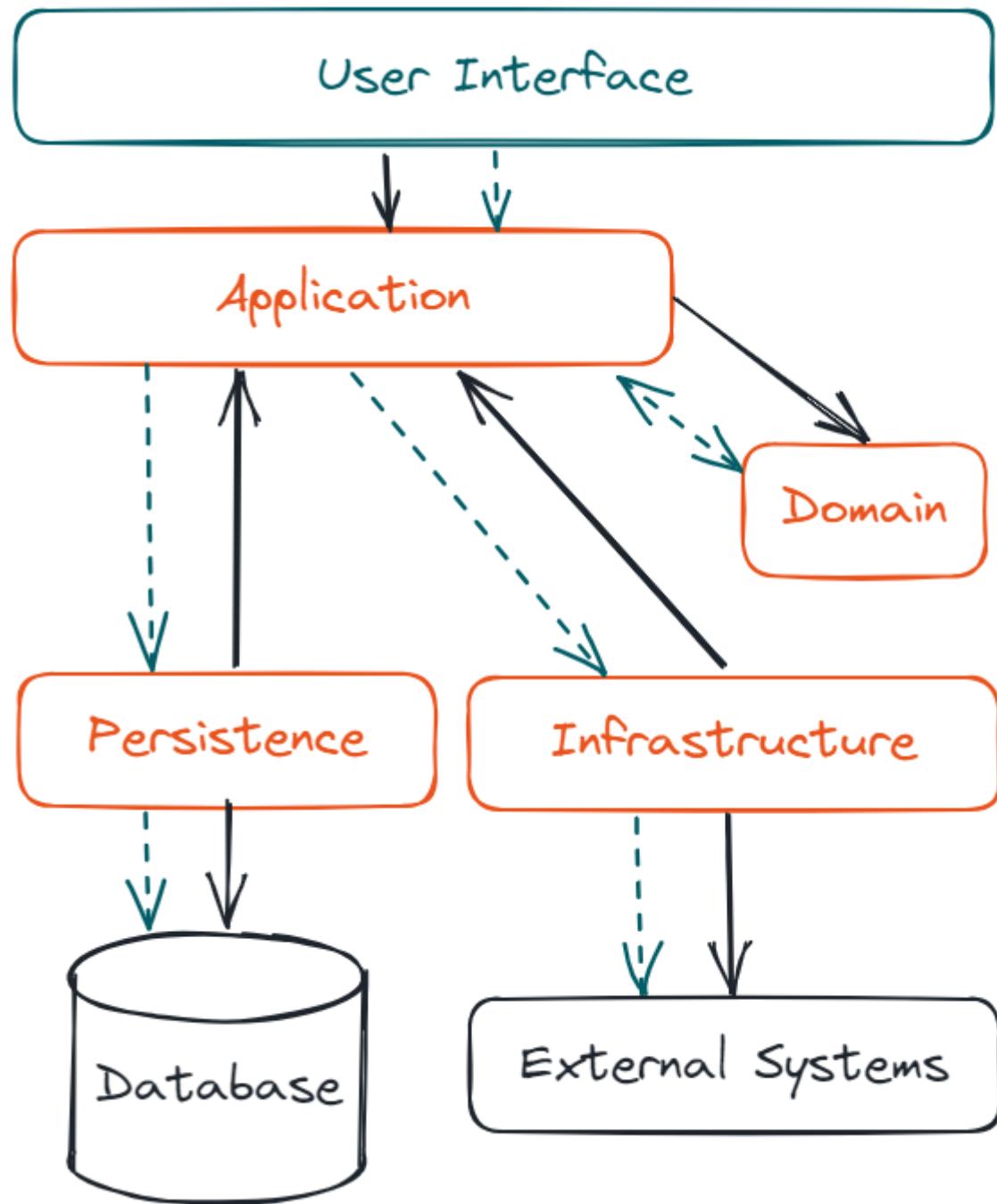
→ Cleaner dependencies

→ Allows for evolution with the system's needs

# Structured Dependencies

---

vs. Flow of Control



Control  
----->

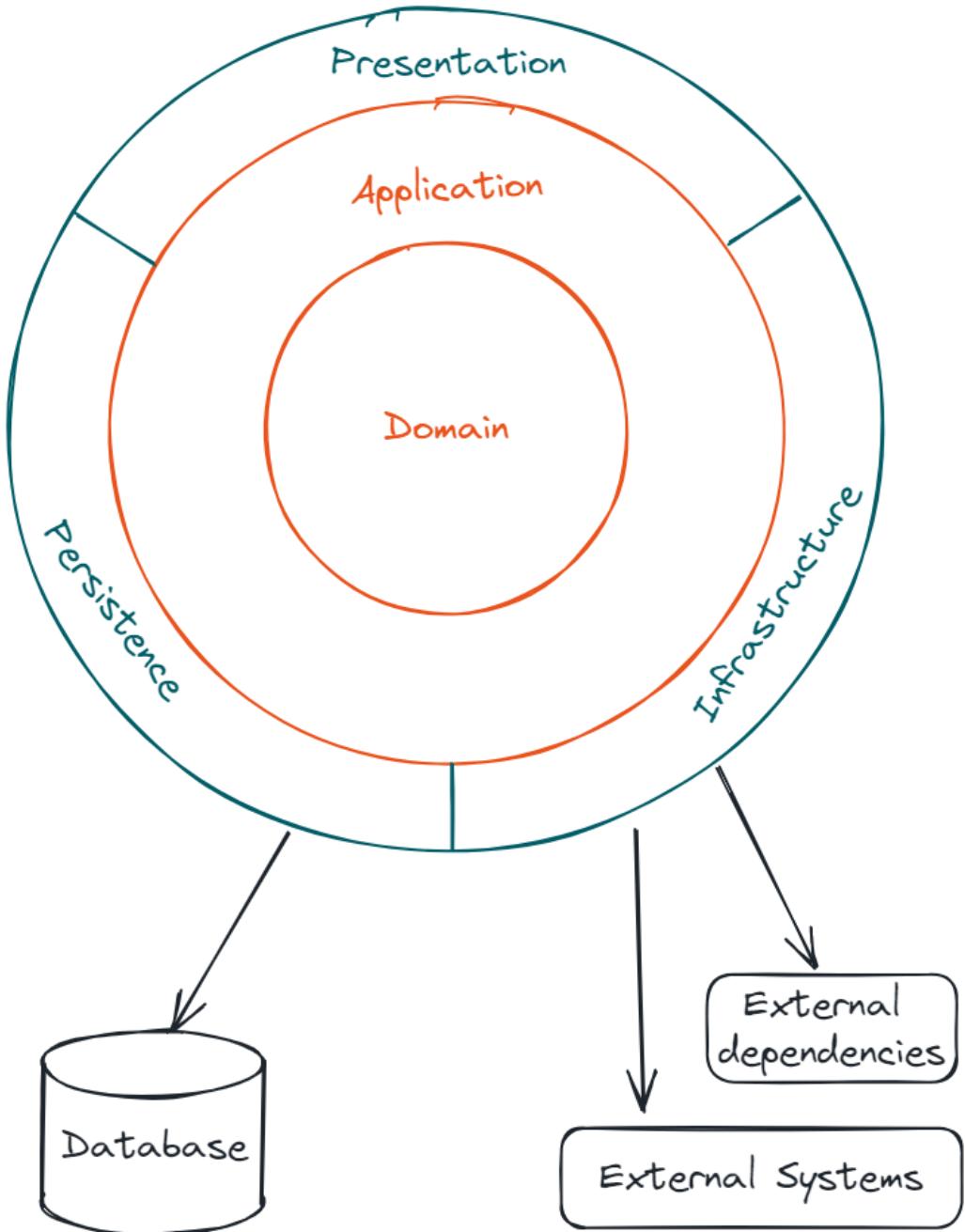
Dependency  
----->

# Structure the solution

---

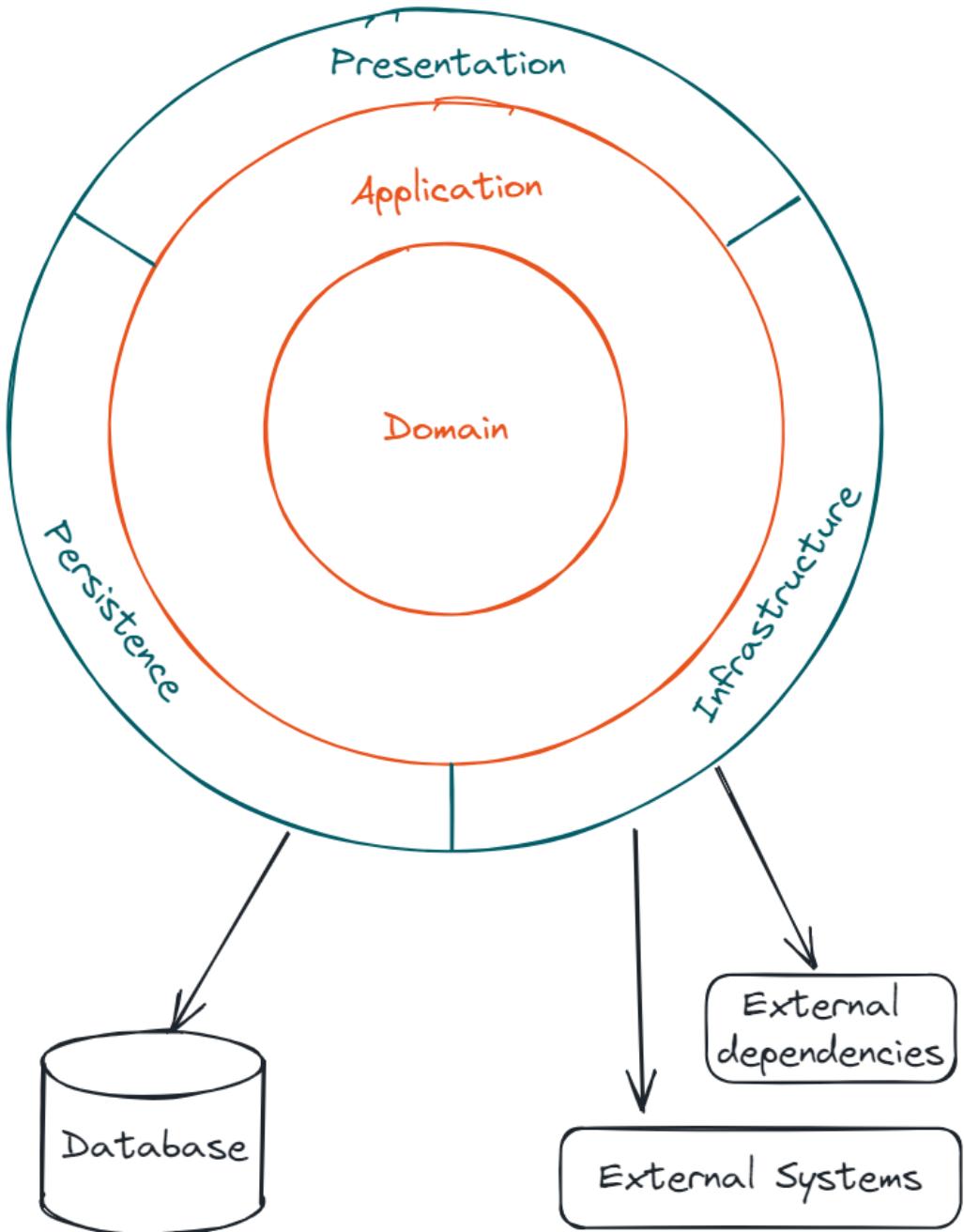
-

What parts go where?



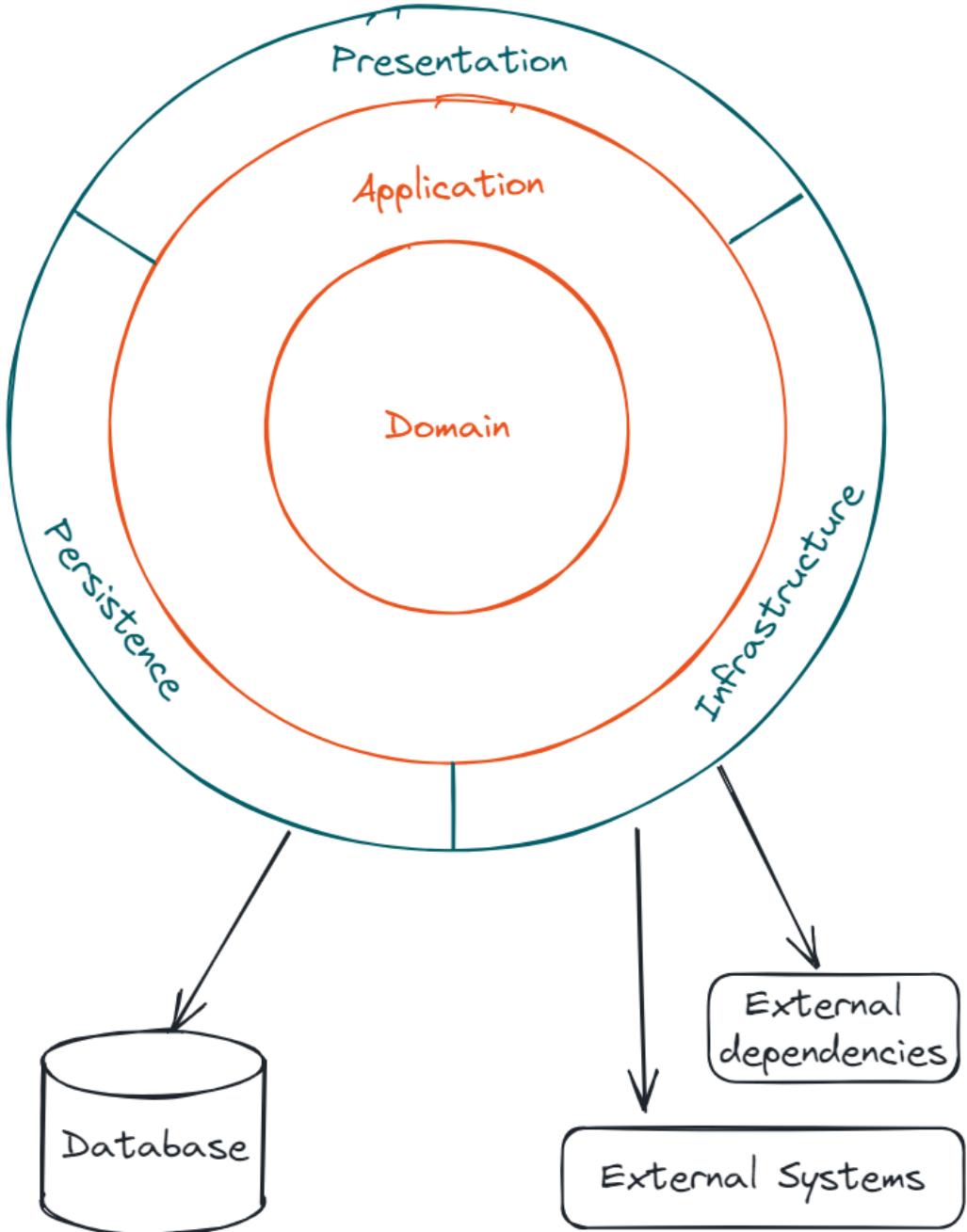
# Domain

- Entities
- (Domain Logic):
  - Aggregates
  - Value Types



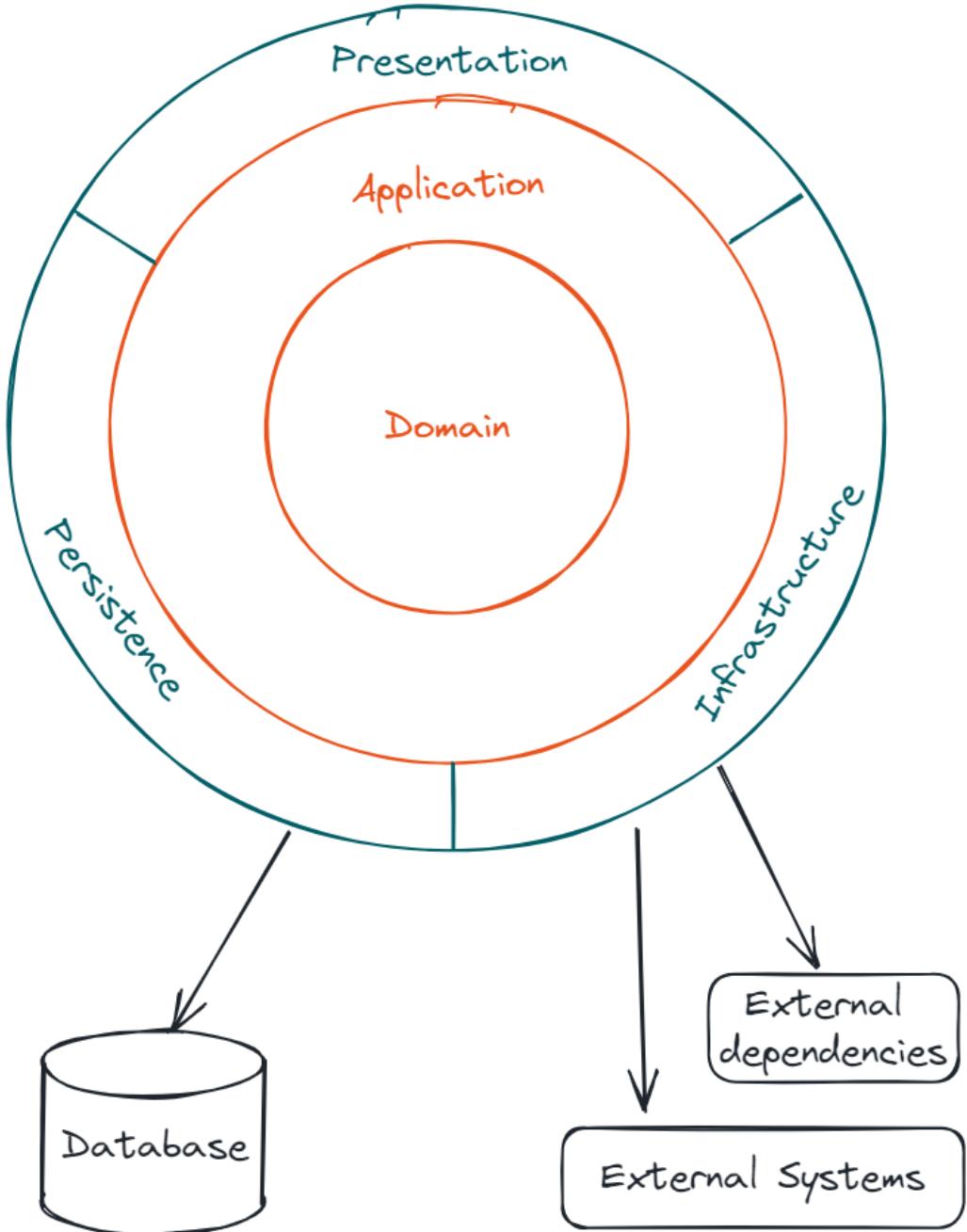
# Application

- Use cases (application logic)
- Contracts:
  - Application services
  - Persistence
  - Infrastructure (dependencies & systems)



# Infrastructure & Persistence

- Persistence implementations (repositories)
- Infrastructure implementations:
  - Service proxies
  - Bus/queue integrations
  - Dependency (package) abstractions
- (DI wiring code)



# Presentation

Translation between:

- Application
- User Interface
- API contract
- ...

# Architectural ideas

---

Things we will do throughout

# Screaming architecture

*"The architecture  
should scream  
the intent of the system"*

Axes



# Categorical

- Classes by “type” (e.g. controllers)
- Default for many frameworks:
  - Easy for the framework
  - Hard for the developer

# Functional

- Classes by “use case”
- Extra effort with many libraries
- Easier to reason about
- Code that works together sits together

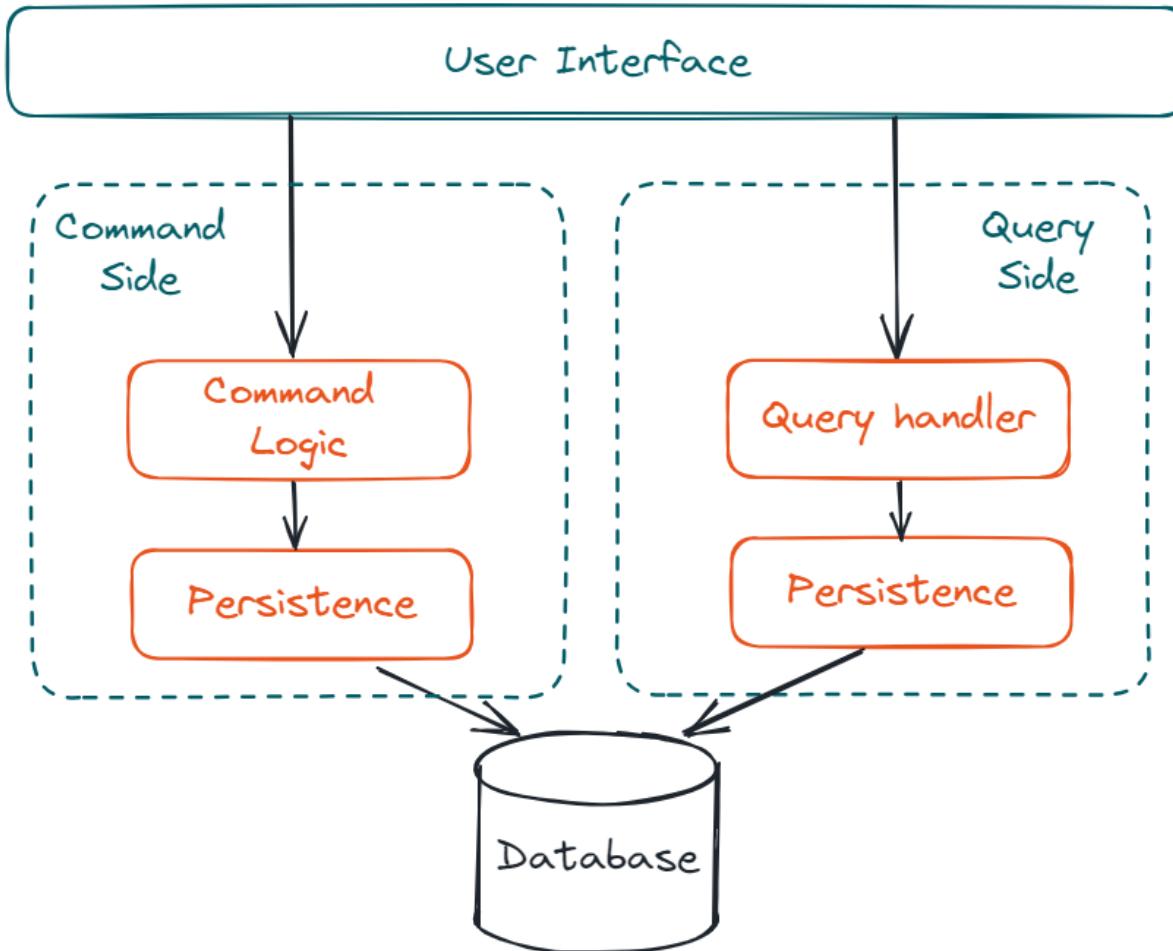
After a period of getting used to, it is awesome

# Commands

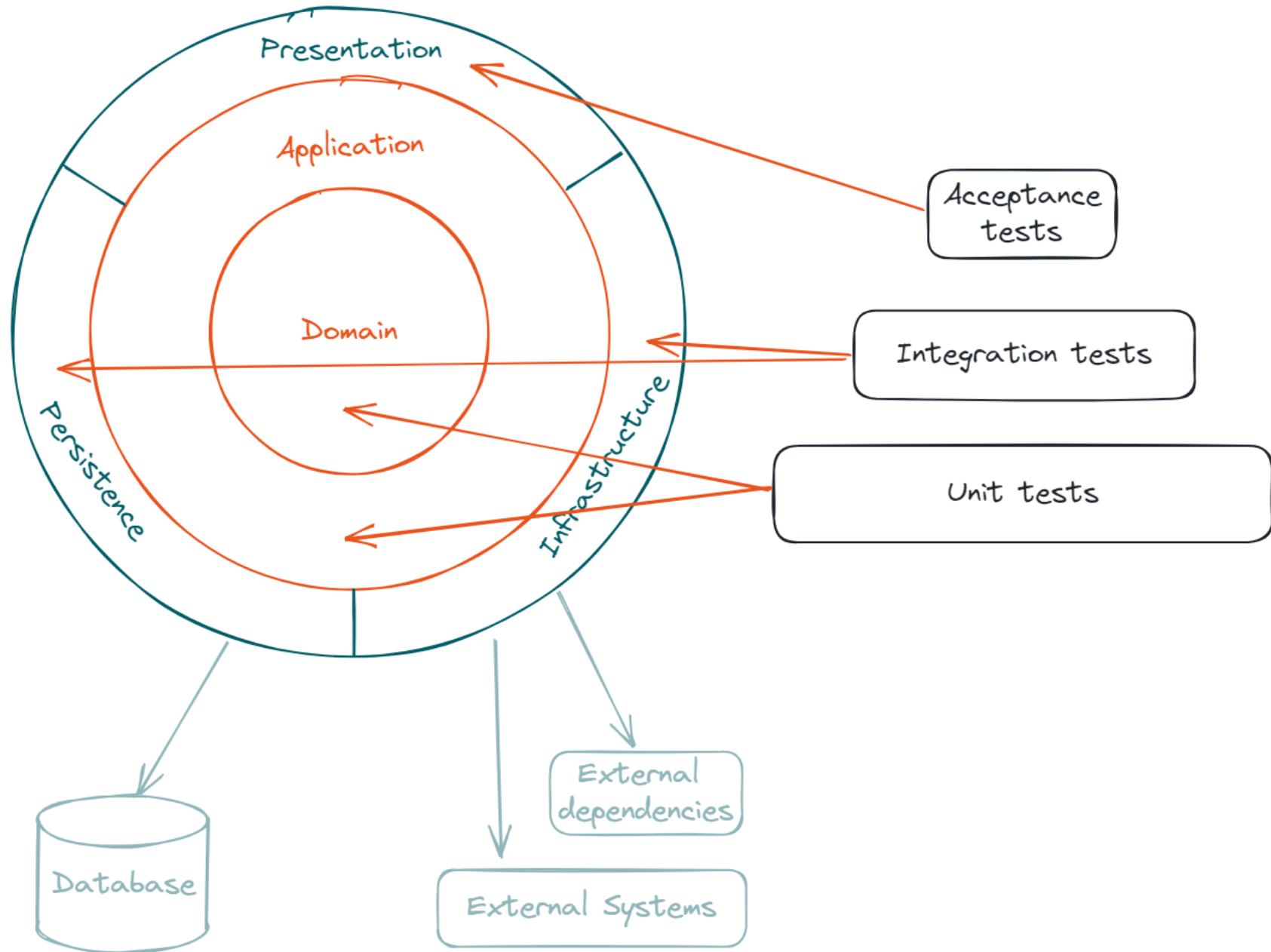
- Change the state of the system
- Don't return results

# Queries

- Return results
- Don't change the state



# Tests



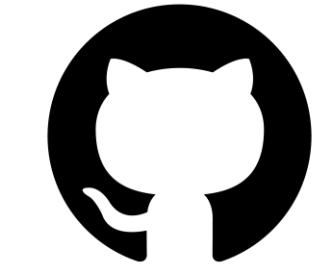
# Let's start building!



miro



[tinyurl.com/goto-ams-miro](https://tinyurl.com/goto-ams-miro)  
Password: xxx



GitHub



[tinyurl.com/goto-ams-git](https://tinyurl.com/goto-ams-git)

AXXES