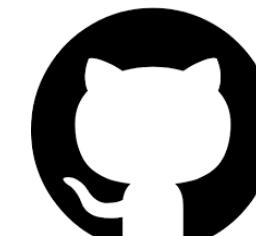


Introduction to Clean Architecture

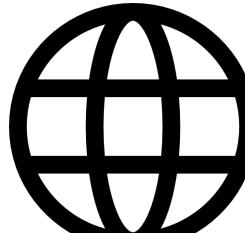
Ivan Paulovich

Stockholm
Software Architecture Meetup
June 29th, 2020

@ivanpaulovich



<https://paulovich.net>





#CleanArchitectureManga 🍄

Ivan Paulovich

ivanpaulovich

Sponsor

Tech Lead, Software Engineer, 20+ GitHub projects about Clean Architecture, SOLID, DDD and TDD. Speaker/Streamer.

Microsoft rMVP.

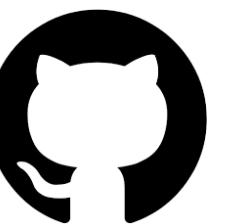
Nordax Bank

Stockholm, Sweden

Sign in to view email

<https://paulovich.net>

@ivanpaulovich



@ivanpaulovich

Paulovich.NET



[CleanArchitectureVSCodeSnippets](#)

Clean Architecture C# Snippets for Visual Studio Code

● TypeScript ⭐ 6

[FluentMediator](#)

FluentMediator is an unobtrusive library that allows developers to build custom pipelines for Commands, Queries and Events.

● C# ⭐ 83 ⚡ 11

[clean-architecture-manga](#)

Template

Clean Architecture with .NET Core 3.1, C# 8 and React+Redux. Use cases as central organizing structure, completely testable, decoupled from frameworks

● C# ⭐ 1.6k ⚡ 304

[todo](#)

Command-Line Task management with storage on your GitHub 🔥

● C# ⭐ 94 ⚡ 16

[dotnet-new-caju](#)

Template

Learn Clean Architecture with .NET Core 3.0 🔥

● C# ⭐ 202 ⚡ 30

[event-sourcing-jambo](#)

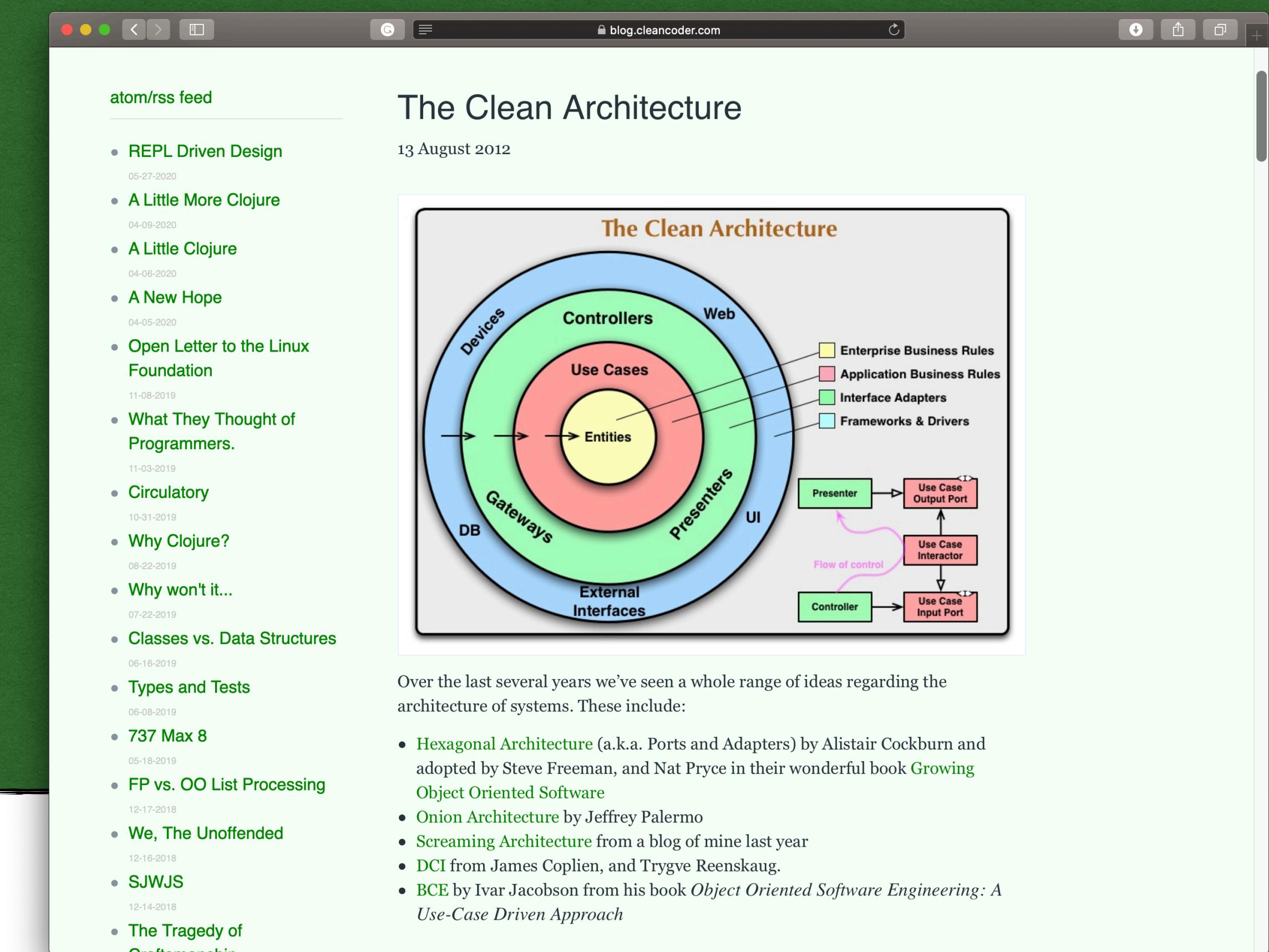
An Hexagonal Architecture with DDD + Aggregates + Event Sourcing using .NET Core, Kafka e MongoDB (Blog Engine)

● C# ⭐ 146 ⚡ 62

What is the Clean Architecture Style?

“ Each of these architectures produce systems that are:

- Independent of Frameworks.
- Testable.
- Independent of UI.
- Independent of Database.
- Independent of any external agency.





style

/stɪl/

See definitions in:

All

Fashion

Hairdressing

Biology

noun

1. a particular procedure by which something is done; a manner or way.
"different styles of management"

Similar:

manner

way

technique

method

methodology

approach



2. a distinctive appearance, typically determined by the principles according to which something is designed.
"the pillars are no exception to the general style"

verb

1. design or make in a particular form.

"the yacht is well proportioned and conservatively styled"

Similar:

design

fashion

tailor

make

produce

2. designate with a particular name, description, or title.

"the official is styled principal and vice chancellor of the university"

Similar:

call

name

title

entitle

dub

designate

term

address



Clean Architecture Style

Clean Architecture Style

Hexagonal Architecture Style

Clean Architecture Style

Hexagonal Architecture Style

Ports and Adapters
Pattern

Dependency Inversion
Principle

Test-Driven Development

Clean Architecture Style

Object-Oriented Design
Principles

Use Cases as Central
Organizing Structure

Pluggable User Interface

Hexagonal Architecture Style

Ports and Adapters
Pattern

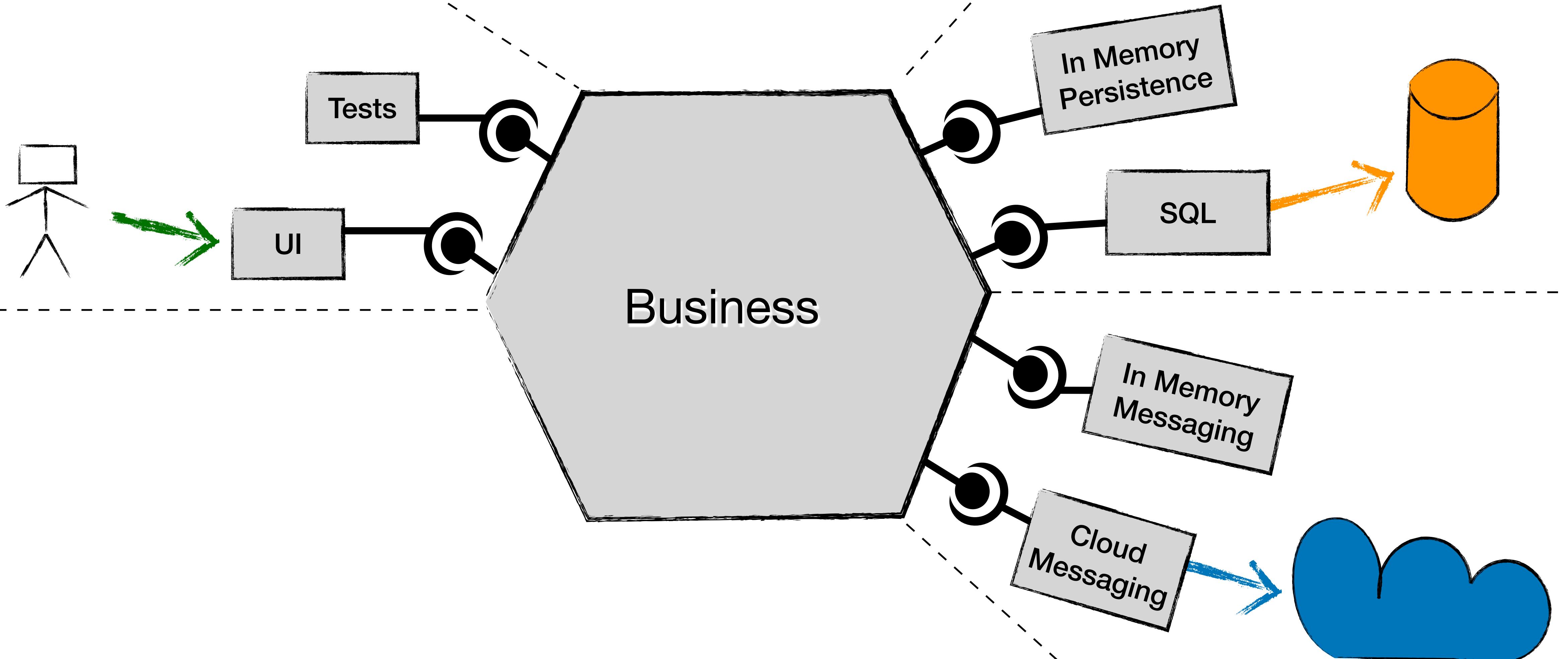
Dependency Inversion
Principle

Test-Driven Development

Hexagonal Architecture Style

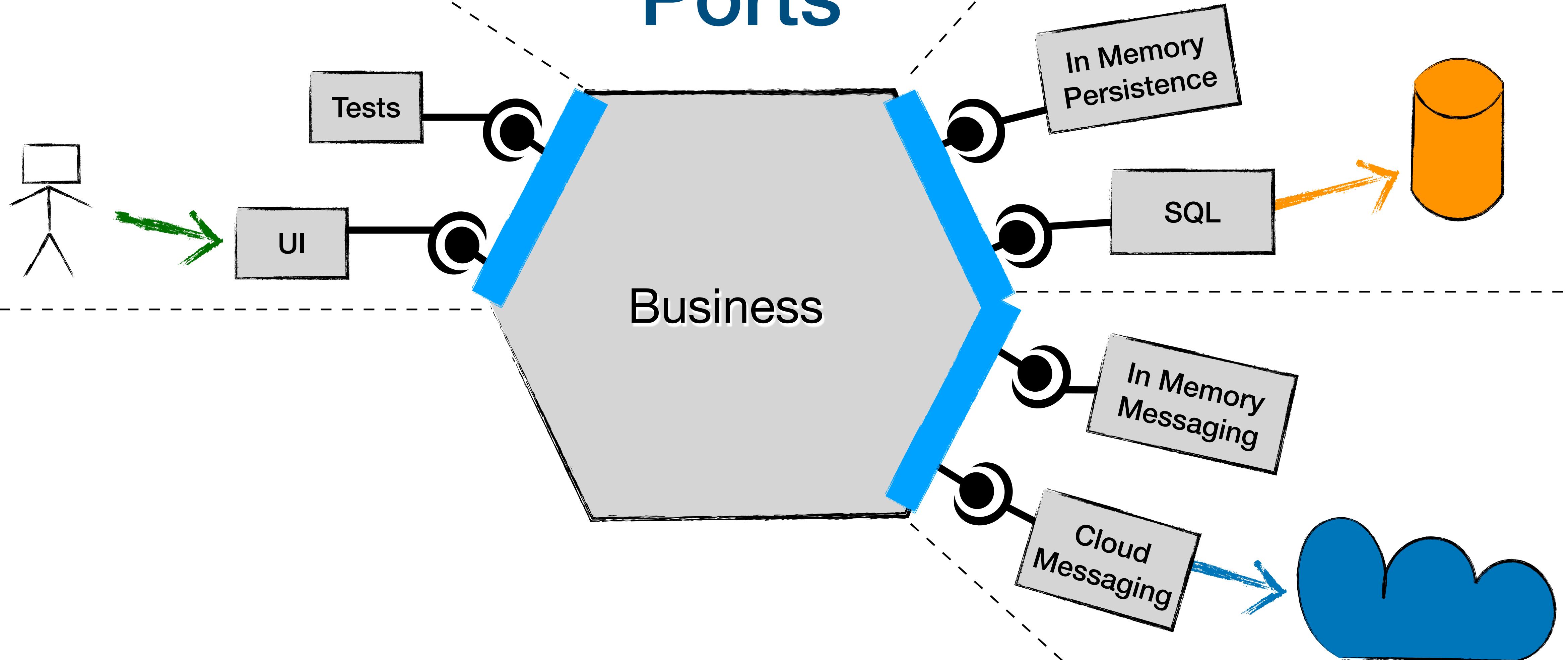
- Ports and Adapters pattern.
- The implementation is guided by tests.
- It is decoupled from technology details.

Ports and Adapters



Ports and Adapters

Ports



Ports

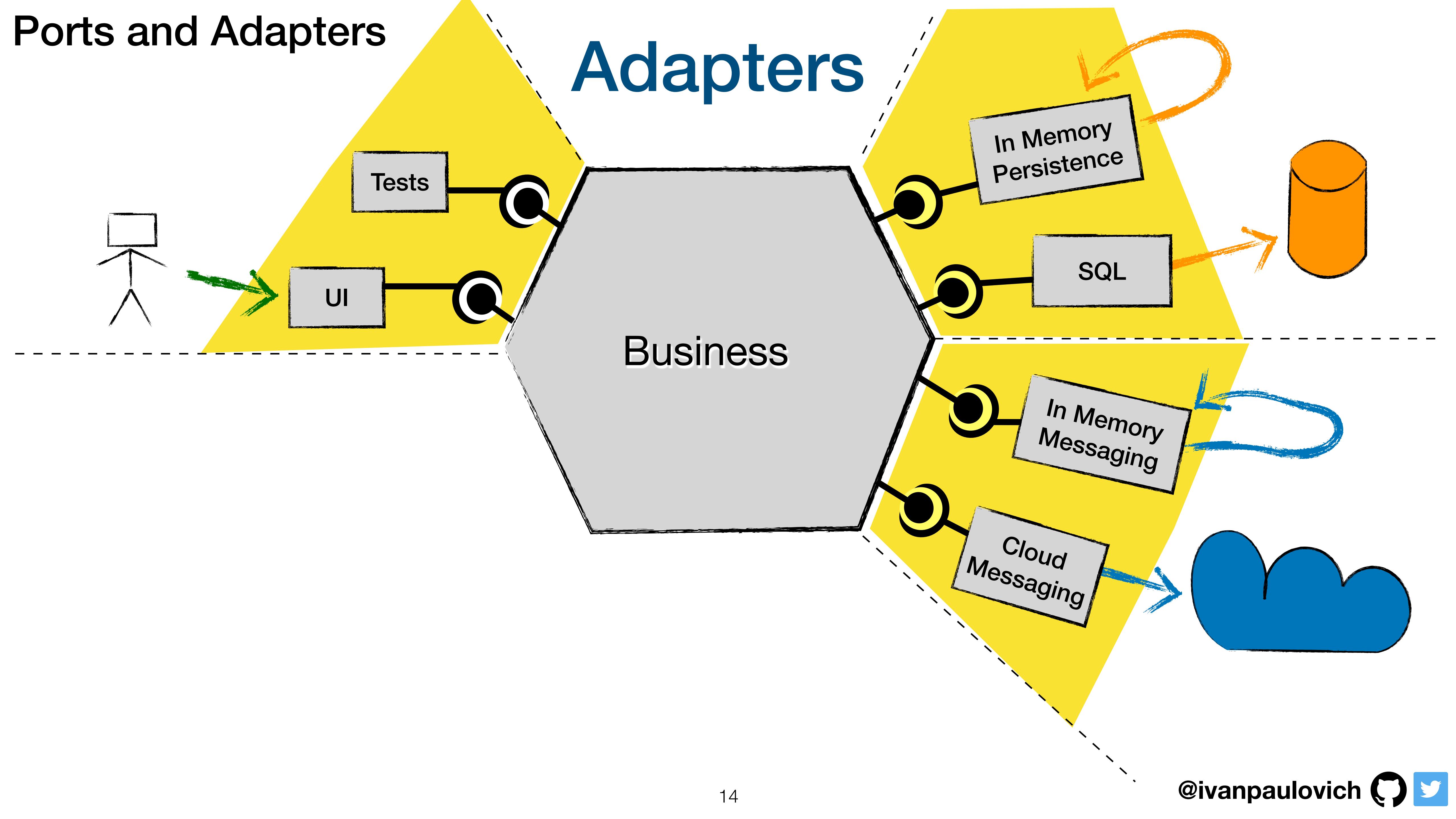
```
public interface IDepositUseCase
{
    Task Execute(DepositInput input);
}
```

```
public interface IDepositOutputPort
{
    void Standard(DepositOutput output);
    void WriteError(string message);
    void NotFound(string message);
}
```

```
public interface IAccountRepository
{
    Task<IAccount> GetAccount(AccountId accountId);
    Task<IList<IAccount>> GetBy(CustomerId customerId);
    Task Add(IAccount account, ICredit credit);
    Task Update(IAccount account, ICredit credit);
    Task Update(IAccount account, IDebit debit);
    Task Delete(IAccount account);
}
```

Ports and Adapters

Adapters



Adapters

```
✓ Infrastructure
  ✓ DataAccess
    > Configuration
    > Entities
    > Migrations
  ✓ Repositories
    C# AccountRepository.cs
    C# AccountRepositoryFake.cs
    C# CustomerRepository.cs
    C# CustomerRepositoryFake.cs
    C# UserRepository.cs
    C# UserRepositoryFake.cs
    C# ContextFactory.cs
    C# EntityFactory.cs
    C# MangaContext.cs
    C# MangaContextFake.cs
    C# SeedData.cs
    C# UnitOfWork.cs
    C# UnitOfWorkFake.cs
```

```
public sealed class AccountRepository : IAccountRepository
{
    private readonly MangaContext _context;

    /// <summary>
    /// </summary>
    /// <param name="context"></param>
    public AccountRepository(MangaContext context) => this._context = context ??
        throw new ArgumentNullException(
            nameof(context));

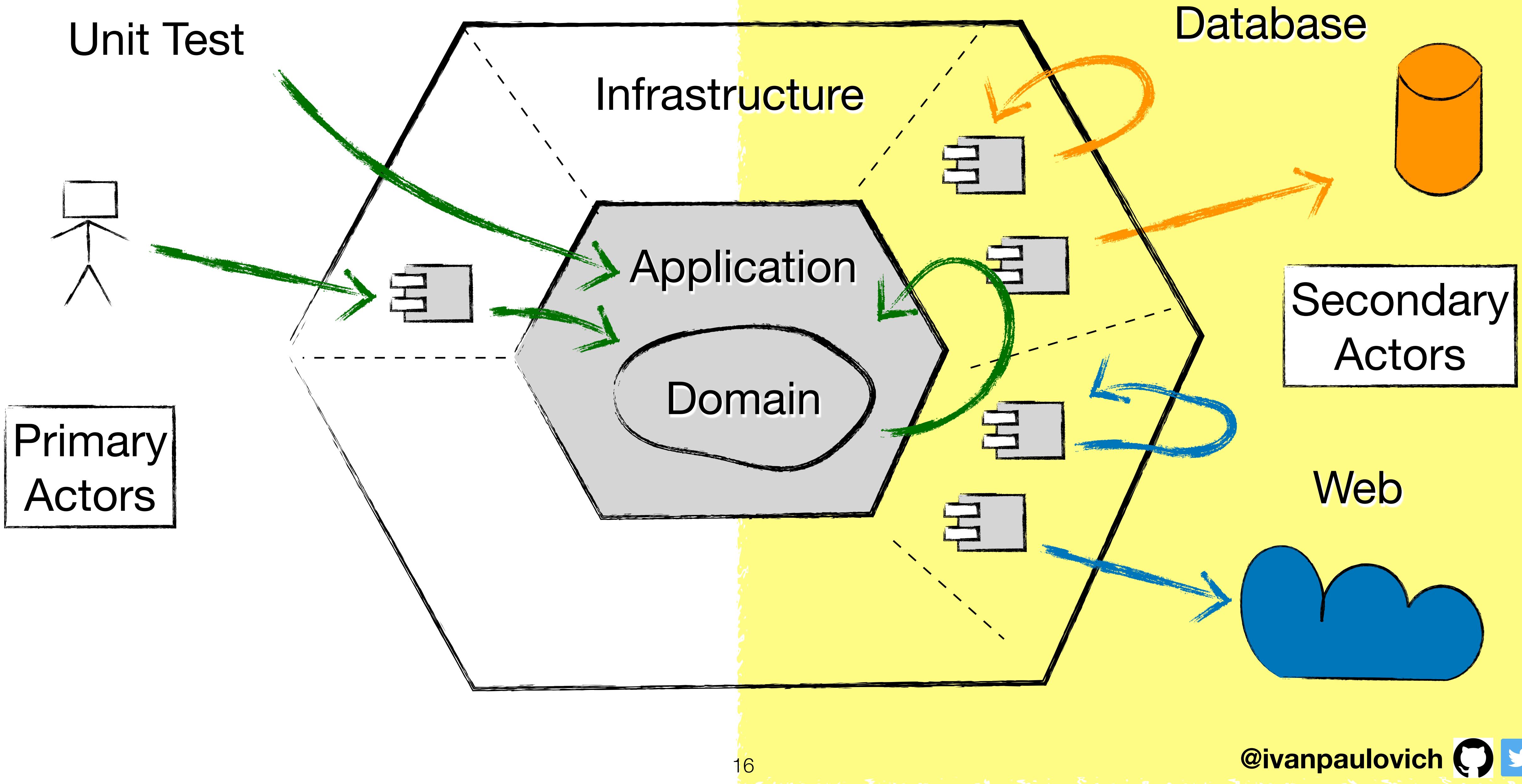
    /// <inheritdoc />
    public async Task<IList<IAccount>> GetBy(CustomerId customerId)
    {
        var accounts = this._context
            .Accounts
            .Where(e => e.CustomerId.Equals(customerId))
            .Select(e => (IAccount)e)
            .ToList();

        return await Task.FromResult(accounts)
            .ConfigureAwait(false);
    }

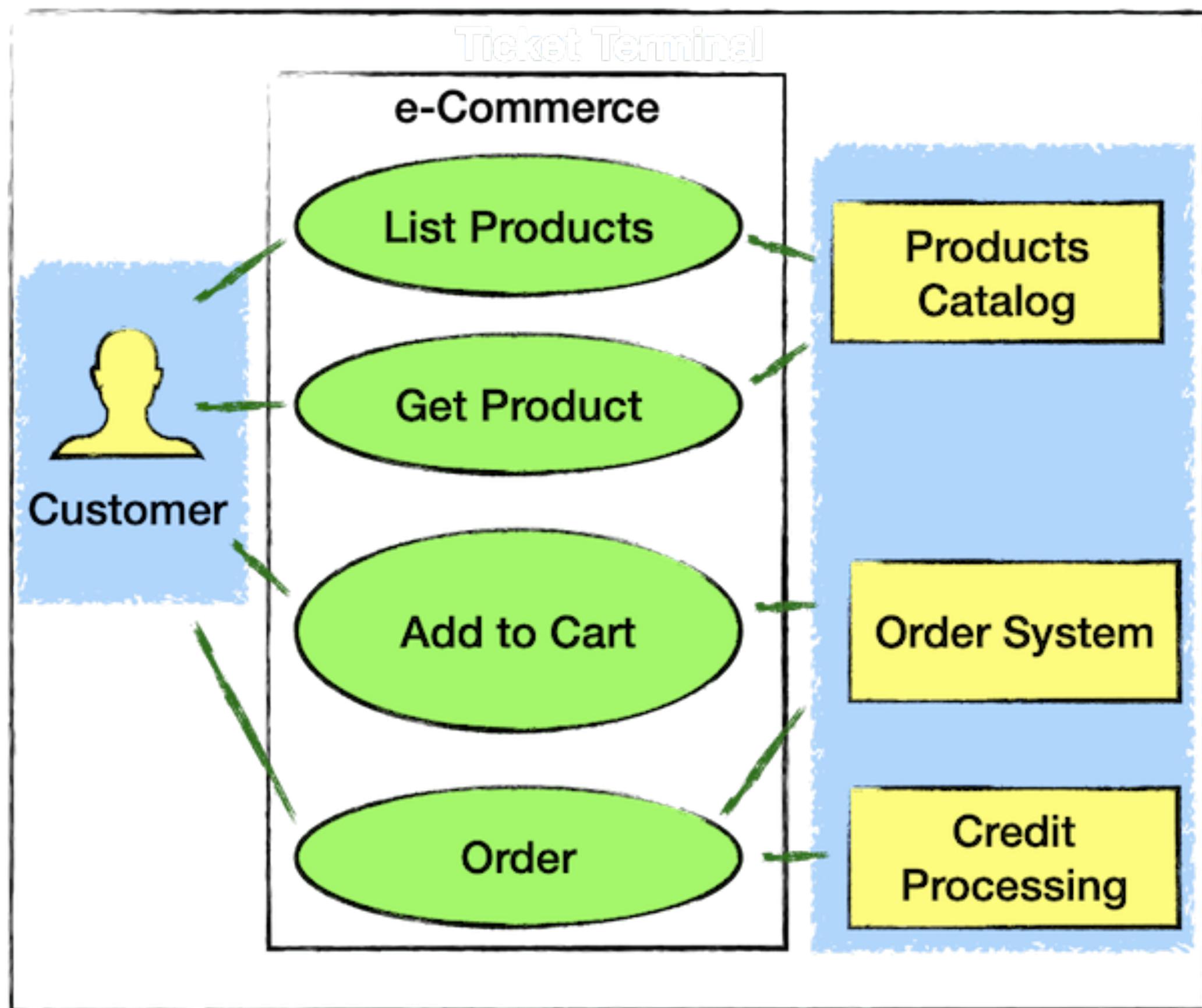
    /// <inheritdoc />
    public async Task Add(IAccount account, ICredit credit)
    {
        await this._context
            .Accounts
            .AddAsync((Account)account)
            .ConfigureAwait(false);

        await this._context
            .Credits
            .AddAsync((Credit)credit)
            .ConfigureAwait(false);
    }
}
```

Ports and Adapters



Use Cases



- Use Cases show the intent of a system.
- Use Cases are delivery independent.
- Use Cases are **algorithms** that interpret the input to generate the output data.
- Use Cases diagrams highlight the Primary and secondary actors.

Deposit Use Case

CLEAN-ARCHITECTURE-MANGA

- ✓ Application
 - > Boundaries
 - > Services
- ✓ UseCases
 - C# CloseAccountUseCase.cs
 - C# DepositUseCase.cs
 - C# GetAccountsUseCase.cs
 - C# GetAccountUseCase.cs
 - C# GetCustomerUseCase.cs
 - C# RegisterUseCase.cs
 - C# TransferUseCase.cs
 - C# WithdrawUseCase.cs

```
public async Task Execute(DepositInput input)
{
    if (input is null)
    {
        this._depositGetAccountsOutputPort
            .WriteError(Messages.InputIsNull);
        return;
    }

    IAccount account = await this._accountRepository
        .GetAccount(input.AccountId)
        .ConfigureAwait(false);

    if (account is null)
    {
        this._depositGetAccountsOutputPort
            .NotFound(Messages.AccountDoesNotExist);
        return;
    }

    ICredit credit = await this._accountService
        .Deposit(account, input.Amount)
        .ConfigureAwait(false);

    await this._unitOfWork
        .Save()
        .ConfigureAwait(false);

    this.BuildOutput(credit, account);
}
```

Deposit Use Case Input and Output Messages

```
public sealed class DepositInput
{
    /// <summary>
    ///     Initializes a new instance of the <see cref="DepositInput" /> class.
    /// </summary>
    /// <param name="accountId">AccountId.</param>
    /// <param name="amount">Positive amount to deposit.</param>
    5 references
    public DepositInput(
        Guid accountId,
        decimal amount)
    {
        this.AccountId = new AccountId(accountId);
        this.Amount = new PositiveMoney(amount);
    }

    /// <summary>
    ///     Gets AccountId.
    /// </summary>
    3 references
    public AccountId AccountId { get; }

    /// <summary>
    ///     Gets the Amount.
    /// </summary>
    2 references
    public PositiveMoney Amount { get; }
}
```

```
public sealed class DepositOutput
{
    /// <summary>
    ///     Initializes a new instance of the <see cref="DepositOutput" /> class.
    /// </summary>
    /// <param name="credit">Credit object.</param>
    /// <param name="updatedBalance">The updated balance.</param>
    1 reference
    public DepositOutput(
        ICredit credit,
        Money updatedBalance)
    {
        this.Transaction = credit ?? throw new ArgumentNullException(nameof(credit));
        this.UpdatedBalance = updatedBalance;
    }

    /// <summary>
    ///     Gets the Transaction object.
    /// </summary>
    3 references
    public ICredit Transaction { get; }

    /// <summary>
    ///     Gets the updated balance.
    /// </summary>
    2 references
    public Money UpdatedBalance { get; }
}
```

Deposit Use Case

Request/Response

```
/// <summary>
///     The request to Deposit.
/// </summary>
1 reference | You, 3 months ago | 1 author (You)
public sealed class DepositRequest
{
    /// <summary>
    ///     Gets or sets the Account ID.
    /// </summary>
    [Required]
1 reference
    public Guid AccountId { get; set; } = Guid.Empty;

    /// <summary>
    ///     Gets or sets the amount to Deposit.
    /// </summary>
    [Required]
1 reference
    public decimal Amount { get; set; } = .0M;
}

public sealed class DepositResponse
{
    /// <summary>
    ///     The Deposit response constructor.
    /// </summary>
    I reference
    public DepositResponse(
        decimal amount,
        string description,
        DateTime transactionDate,
        decimal updatedBalance)
    {
        this.Amount = amount;
        this.Description = description;
        this.TransactionDate = transactionDate;
        this.UpdateBalance = updatedBalance;
    }

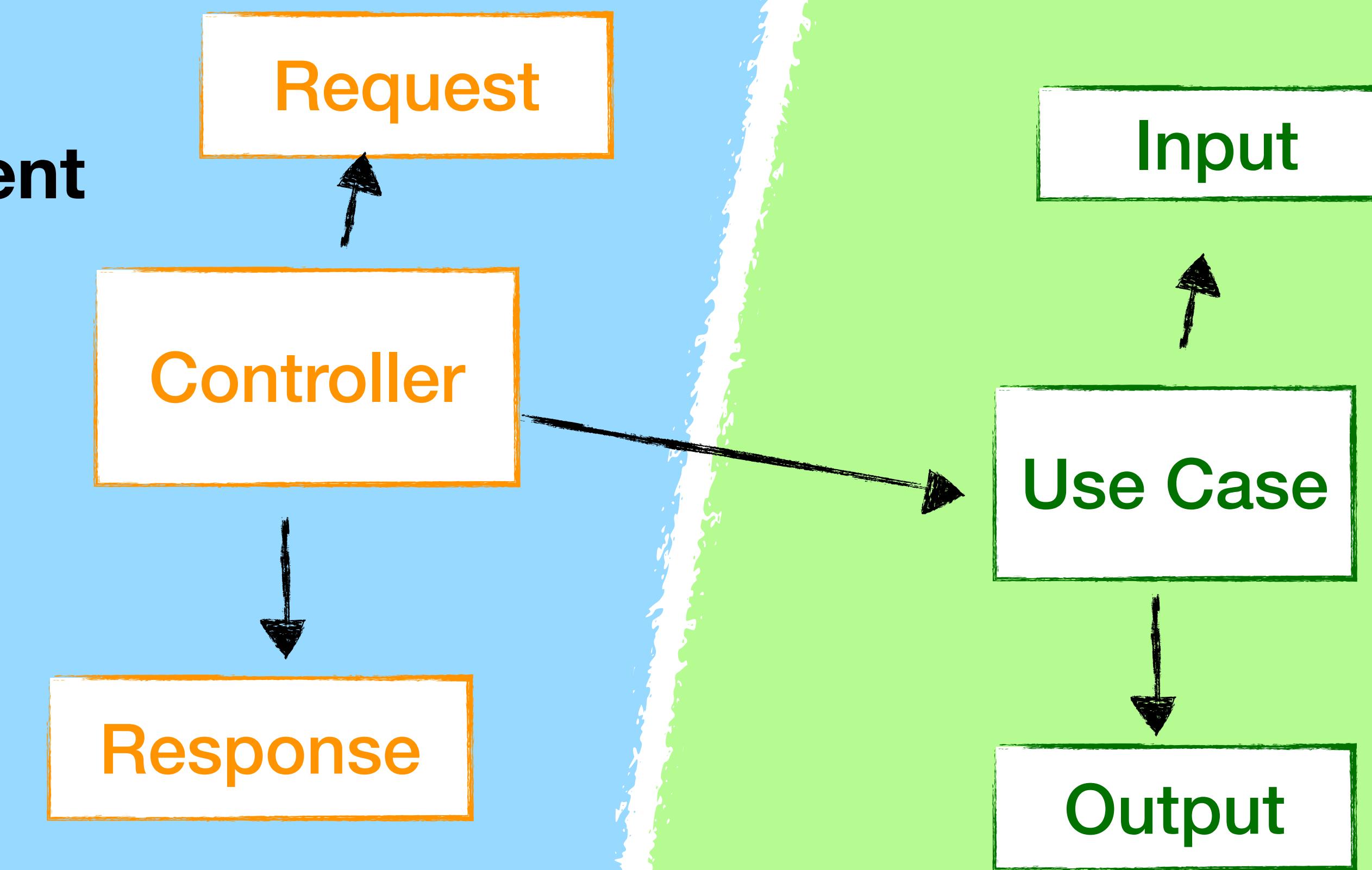
    /// <summary>
    ///     Gets amount Deposited.
    /// </summary>
    [Required]
1 reference
    public decimal Amount { get; }

    /// <summary>
    ///     Gets description.
    /// </summary>
    [Required]
1 reference
    public string Description { get; }

    /// <summary>
    ///     Gets transaction Date.
    /// </summary>
    [Required]
1 reference
    public DateTime TransactionDate { get; }

    /// <summary>
    ///     Gets updated Balance.
    /// </summary>
    [Required]
1 reference
    public decimal UpdateBalance { get; }
}
```

- Concrete
- Specific
- Unstable
- Inconsistent



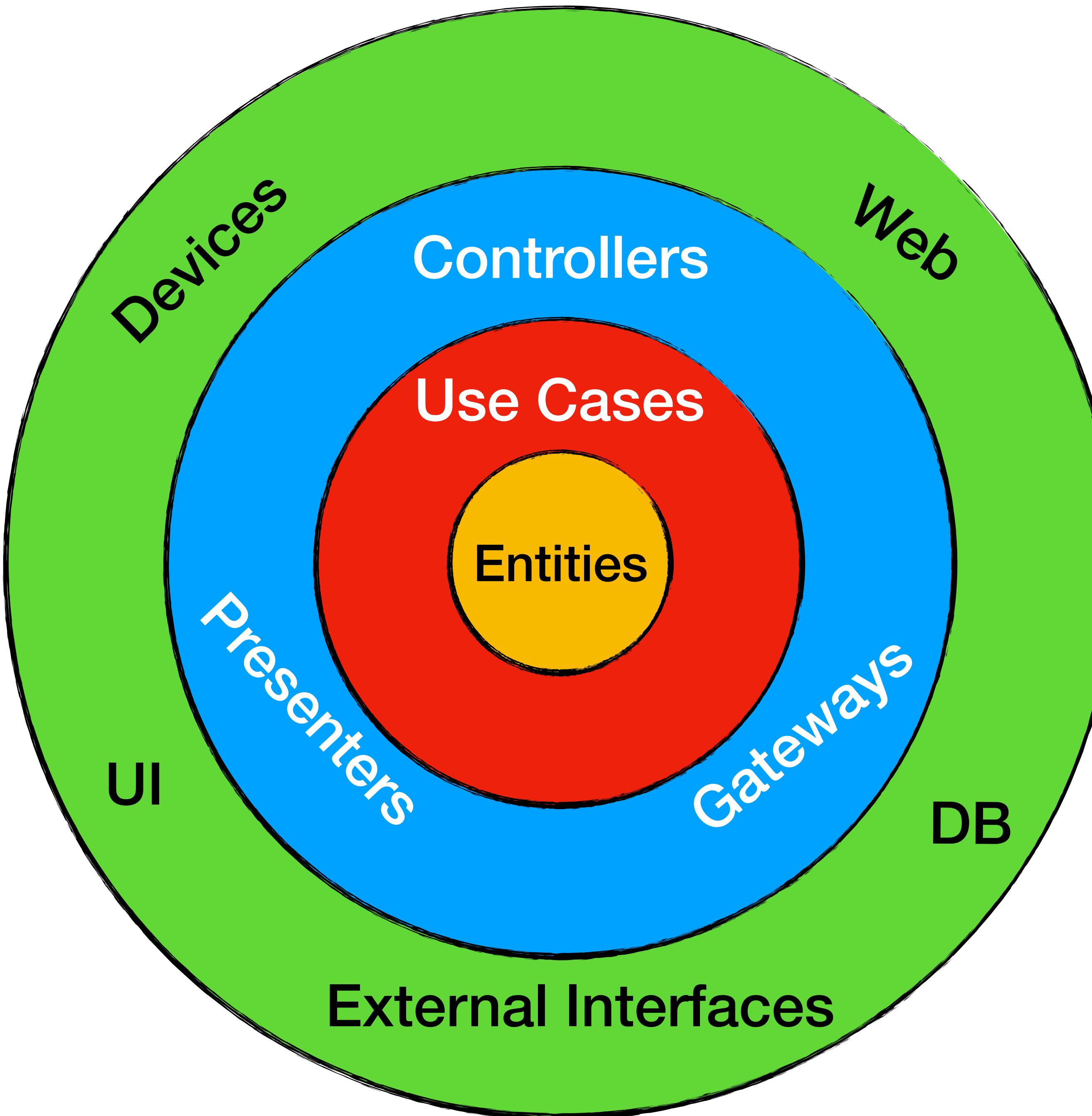
- Abstract
- General
- Stable
- Consistent

User Interface

Core

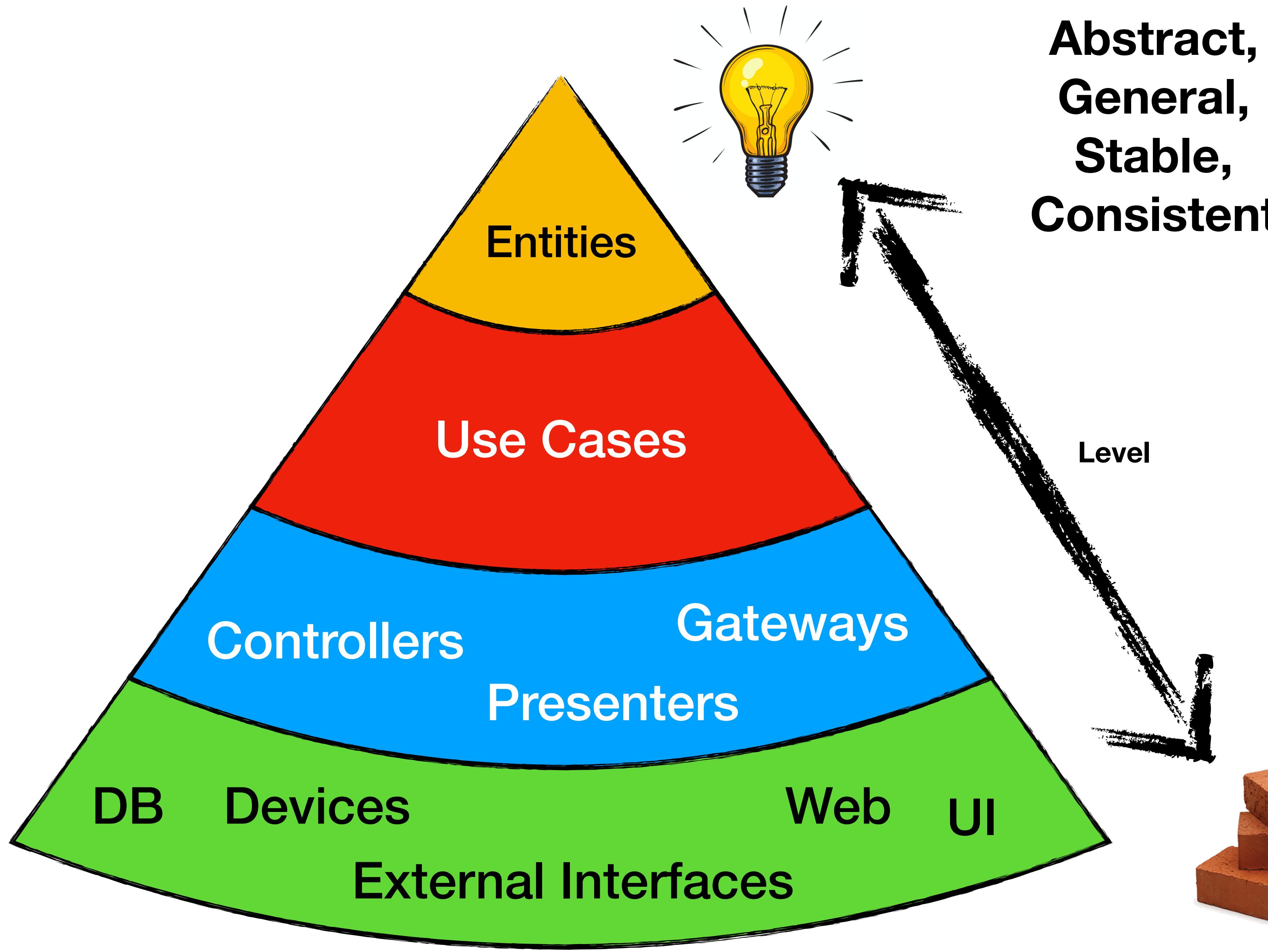
Object-Oriented Design Principles

Clean Architecture

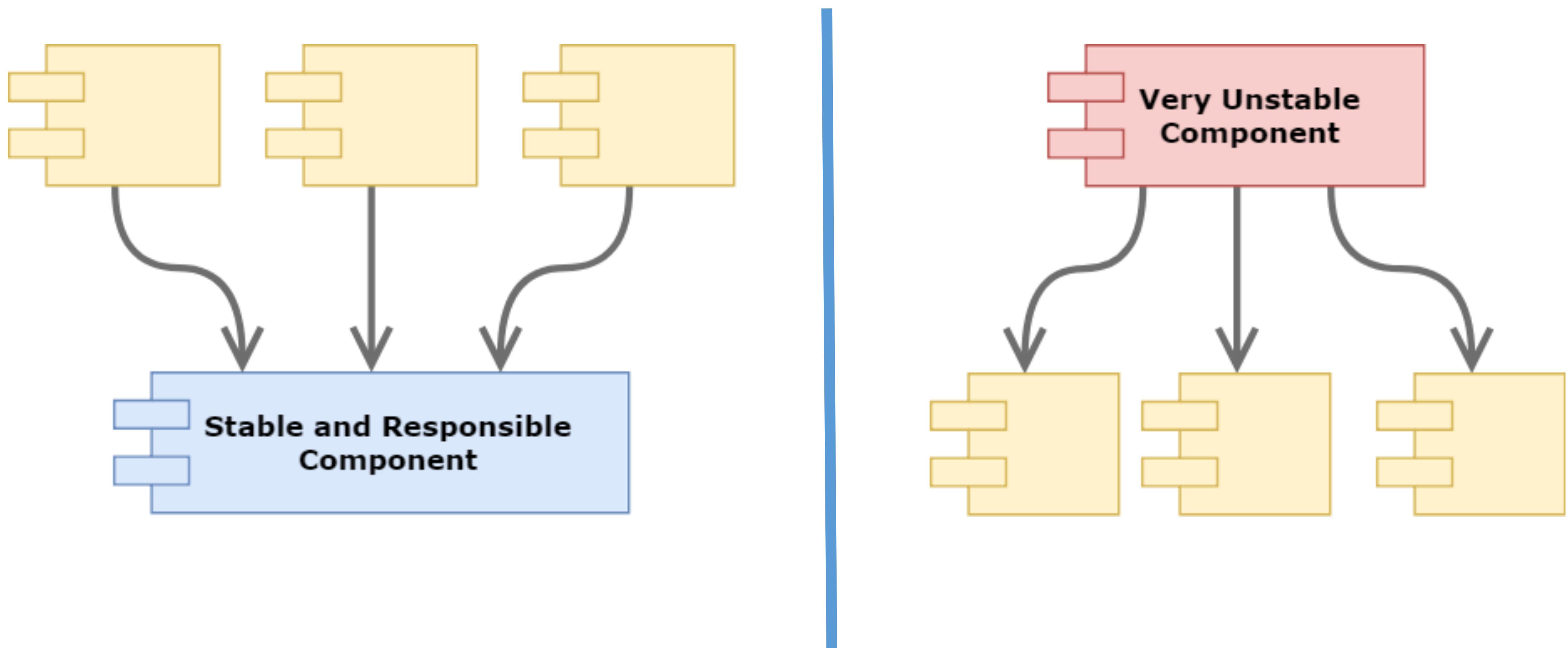


- Abstractness increases with stability.
- Modules depend in the direction of stability.
- Classes that change together are packaged together.

Clean Architecture

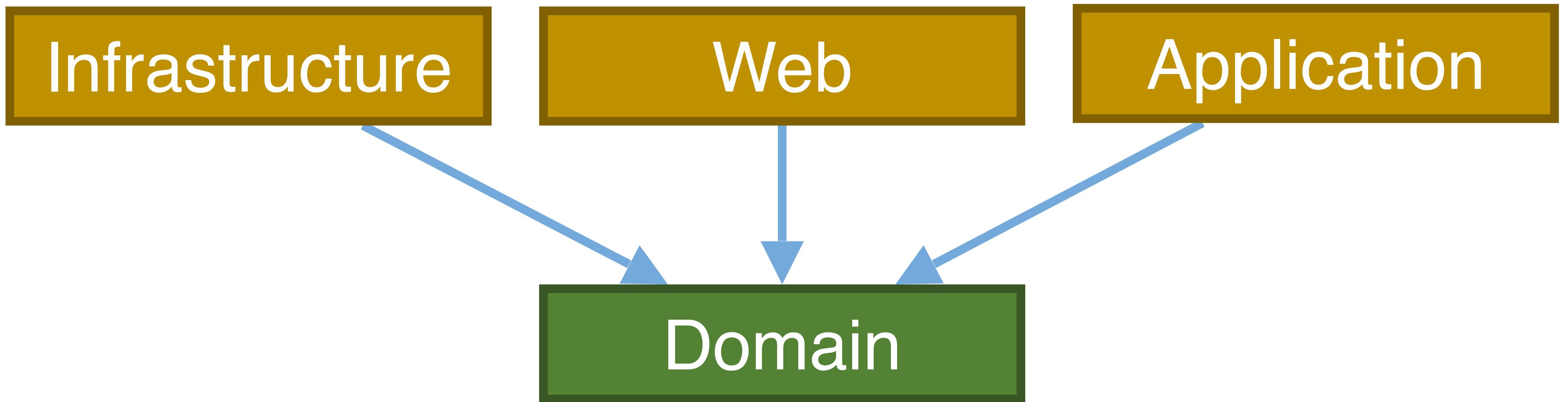


The Stable Dependencies Principle¹

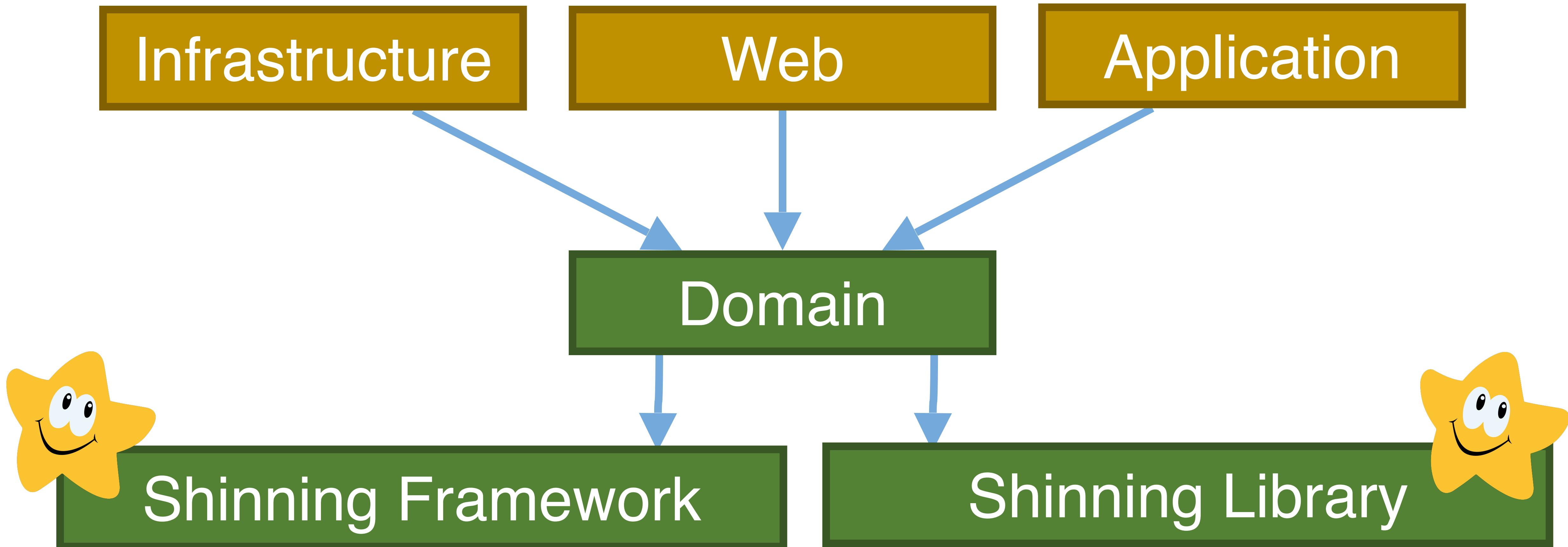


¹Clean Architecture, Robert C. Martin, 2017

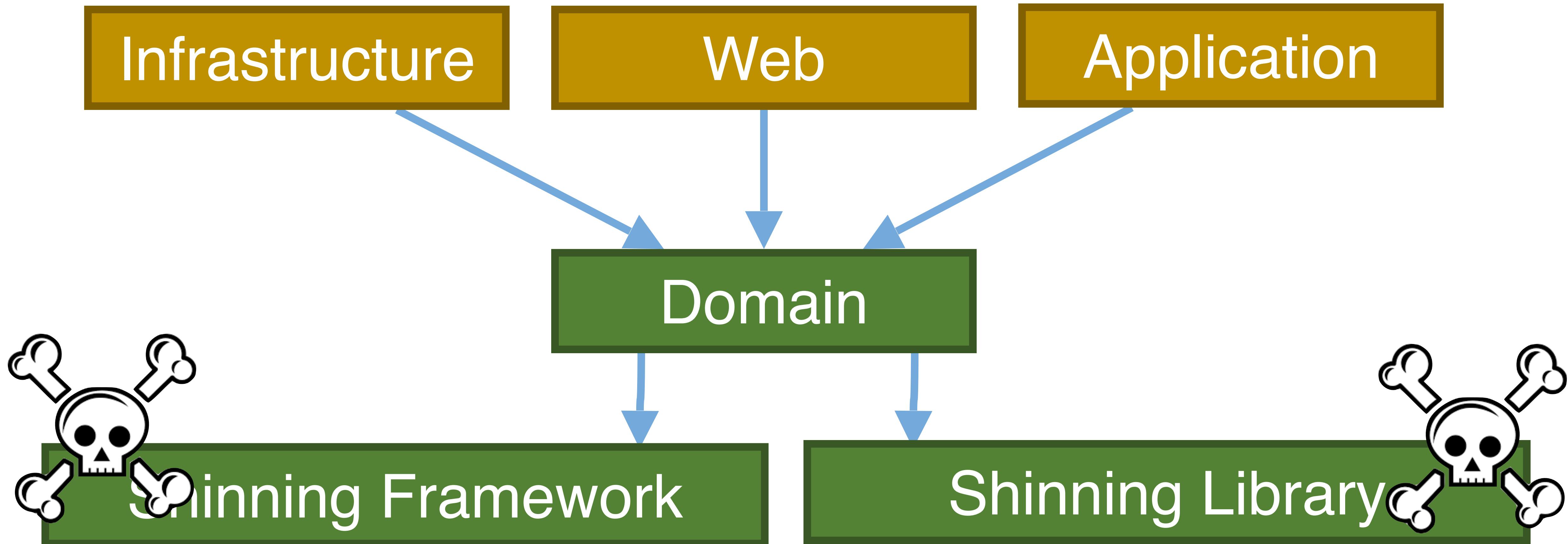
The Stable Dependencies Principle



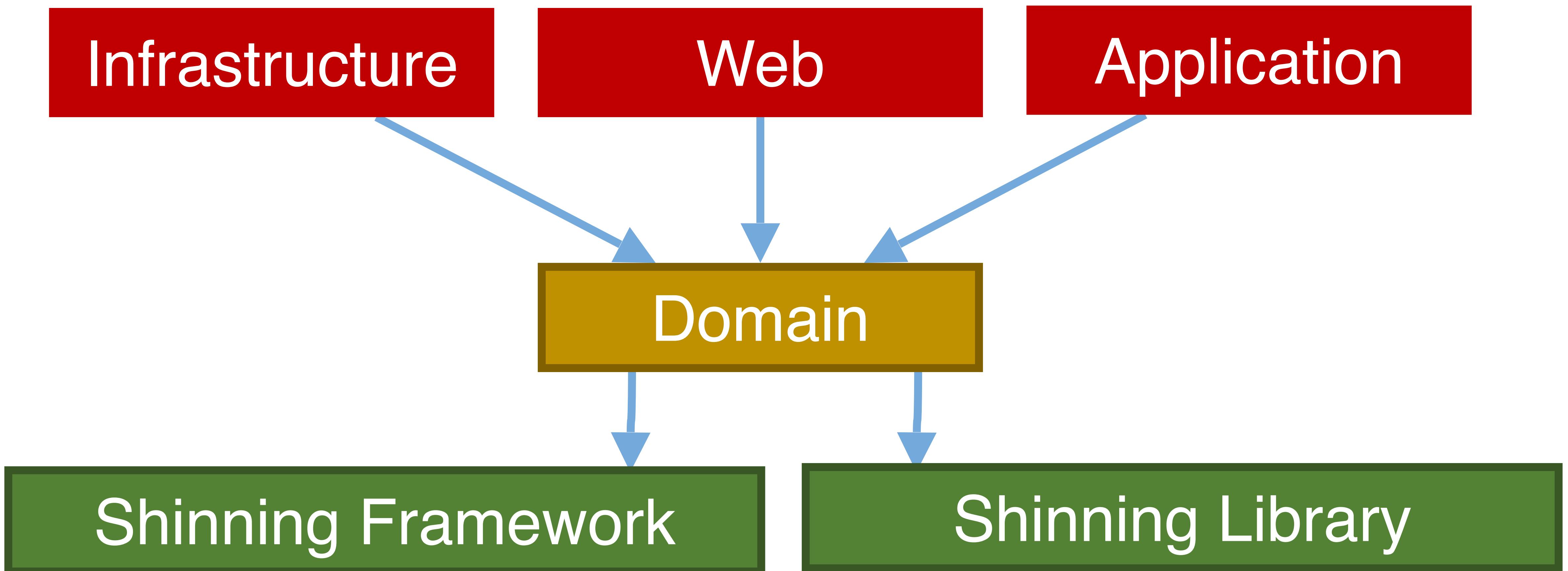
The Stable Dependencies Principle



The Stable Dependencies Principle



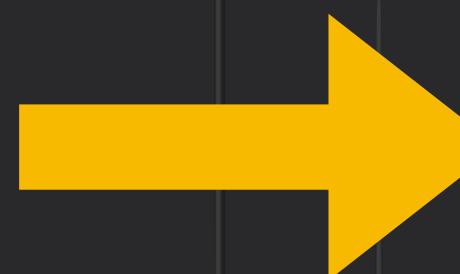
The Stable Dependencies Principle



Walkthrough an Use Case

```
public sealed class AccountsController : ControllerBase
{
    /// <summary>
    ///     Deposit on an account.
    /// </summary>
    /// <response code="200">The updated balance.</response>
    /// <response code="400">Bad request.</response>
    /// <response code="500">Error.</response>
    /// <param name="mediator"></param>
    /// <param name="presenter"></param>
    /// <param name="request">The request to deposit.</param>
    /// <returns>The updated balance.</returns>
    [Authorize]
    [HttpPatch("Deposit")]
    [ProducesResponseType(StatusCodes.Status200OK, Type = typeof(DepositResponse))]
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
    [ProducesResponseType(StatusCodes.Status500InternalServerError)]
    0 references

    public async Task<IActionResult> Deposit(
        [FromServices] IMediator mediator, [FromServices] DepositPresenter presenter,
        [FromForm] [Required] DepositRequest request)
    {
        var input = new DepositInput(request.AccountId, request.Amount);
        await mediator.PublishAsync(input)
            .ConfigureAwait(false);
        return presenter.ViewModel;
    }
}
```



Controller

Walkthrough an Use Case

Use Case

```
public async Task Execute(DepositInput input)
{
    if (input is null)
    {
        this._depositOutputPort
            .WriteError(Messages.InputIsNull);
        return;
    }

    IAccount account = await this._accountRepository
        .GetAccount(input.AccountId)
        .ConfigureAwait(false);

    if (account is null)
    {
        this._depositOutputPort
            .NotFound(Messages.AccountDoesNotExist);
        return;
    }

    ICredit credit = await this._accountService
        .Deposit(account, input.Amount)
        .ConfigureAwait(false);

    await this._unitOfWork
        .Save()
        .ConfigureAwait(false);

    this.BuildOutput(credit, account);
}

1 reference
private void BuildOutput(ICredit credit, IAccount account)
{
    var output = new DepositOutput(
        credit,
        account.GetCurrentBalance());

    this._depositOutputPort.Standard(output);
}
```

Walkthrough an Use Case

```
public async Task<IAccount> GetAccount(AccountId id)
{
    Account account = await this._context
        .Accounts
        .Where(a => a.Id.Equals(id))
        .SingleOrDefaultAsync()
        .ConfigureAwait(false);

    if (account is null)
    {
        return null!;
    }

    var credits = this._context
        .Credits
        .Where(e => e.AccountId.Equals(id))
        .ToList();

    var debits = this._context
        .Debits
        .Where(e => e.AccountId.Equals(id))
        .ToList();

    account.Credits
        .AddRange(credits);
    account.Debits
        .AddRange(debits);

    return account;
}
```

Repository

Walkthrough an Use Case

Use Case

```
public async Task Execute(DepositInput input)
{
    if (input is null)
    {
        this._depositOutputPort
            .WriteError(Messages.InputIsNull);
        return;
    }

    IAccount account = await this._accountRepository
        .GetAccount(input.AccountId)
        .ConfigureAwait(false);

    if (account is null)
    {
        this._depositOutputPort
            .NotFound(Messages.AccountDoesNotExist);
        return;
    }

    ICredit credit = await this._accountService
        .Deposit(account, input.Amount)
        .ConfigureAwait(false);

    await this._unitOfWork
        .Save()
        .ConfigureAwait(false);

    this.BuildOutput(credit, account);
}

1 reference
private void BuildOutput(ICredit credit, IAccount account)
{
    var output = new DepositOutput(
        credit,
        account.GetCurrentBalance());

    this._depositOutputPort.Standard(output);
}
```



Use Case

Walkthrough an Use Case

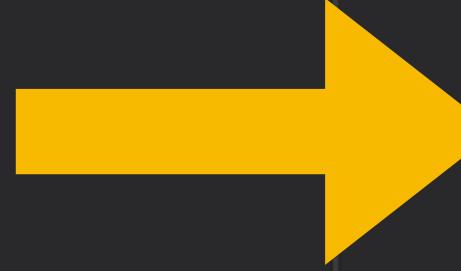
Case

Domain Service

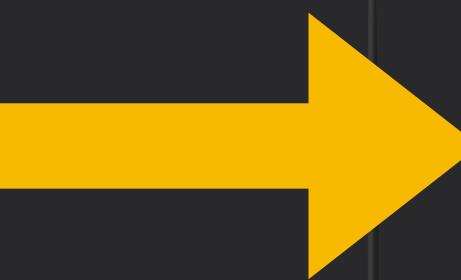
```
public async Task<ICredit> Deposit(IAccount account, PositiveMoney amount)
{
    if (account is null)
    {
        throw new ArgumentNullException(nameof(account));
    }

    ICredit credit = account.Deposit(this._accountFactory, amount);
    await this._accountRepository.Update(account, credit)
        .ConfigureAwait(false);

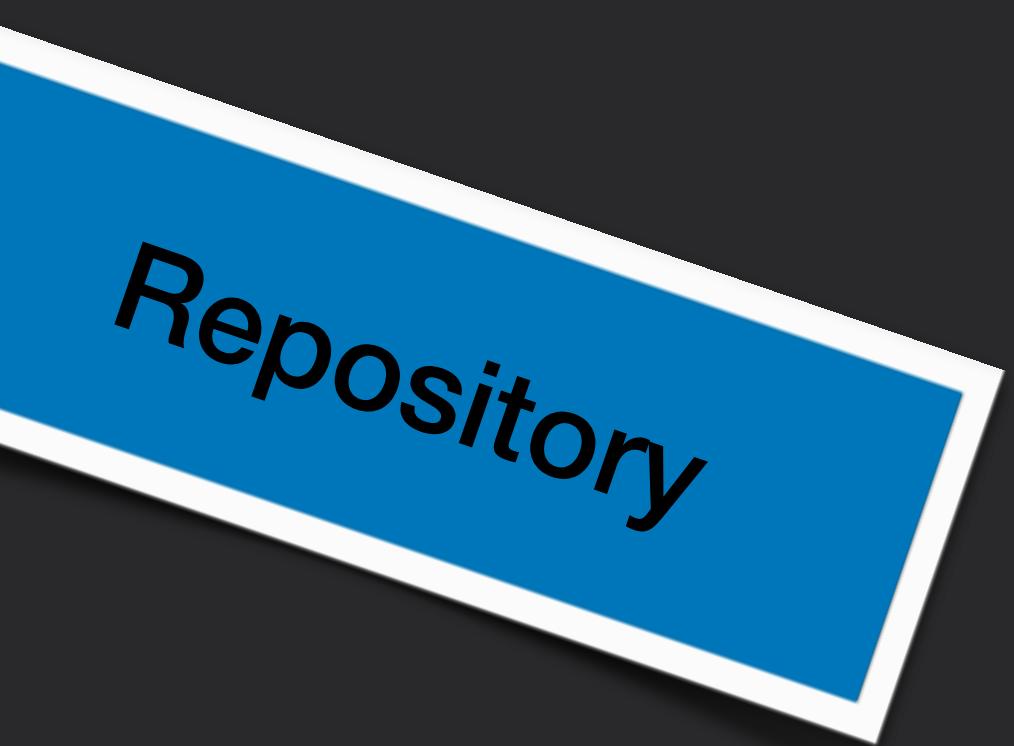
    return credit;
}
```



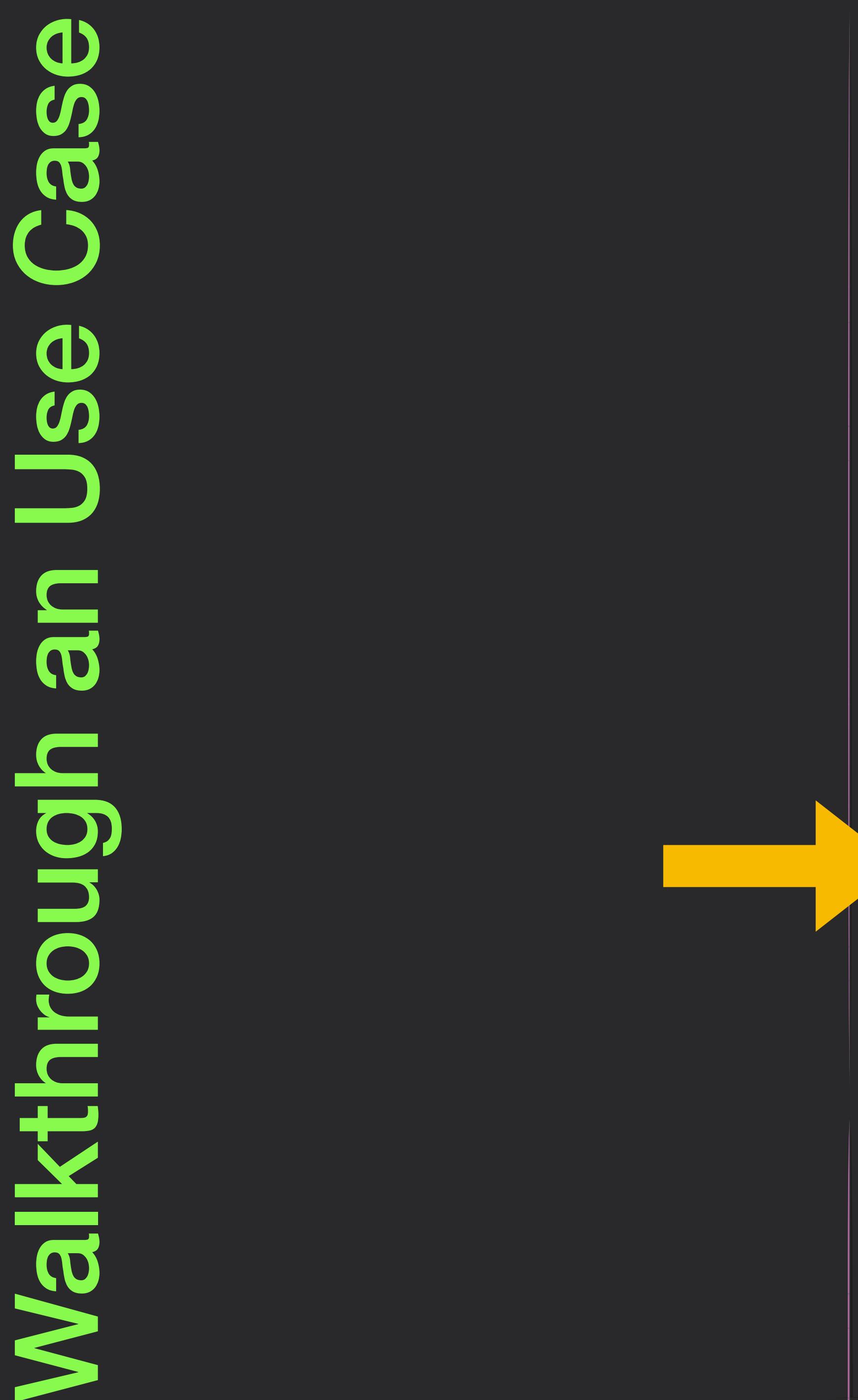
Walkthrough an Use Case



```
public async Task Update(IAccount account, ICredit credit) => await this._context  
    .Credits  
    .AddAsync((Credit)credit)  
    .ConfigureAwait(false);
```



Walkthrough an Use Case



```
public async Task Execute(DepositInput input)
{
    if (input is null)
    {
        this._depositOutputPort
            .WriteError(Messages.InputIsNull);
        return;
    }

    IAccount account = await this._accountRepository
        .GetAccount(input.AccountId)
        .ConfigureAwait(false);

    if (account is null)
    {
        this._depositOutputPort
            .NotFound(Messages.AccountDoesNotExist);
        return;
    }

    ICredit credit = await this._accountService
        .Deposit(account, input.Amount)
        .ConfigureAwait(false);

    await this._unitOfWork
        .Save()
        .ConfigureAwait(false);

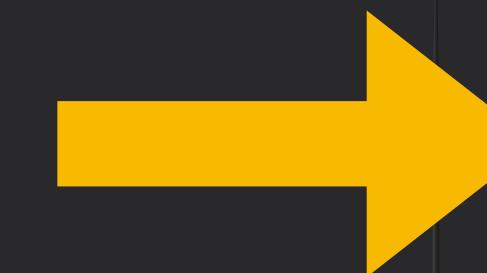
    this.BuildOutput(credit, account);
}

1 reference
private void BuildOutput(ICredit credit, IAccount account)
{
    var output = new DepositOutput(
        credit,
        account.GetCurrentBalance());

    this._depositOutputPort.Standard(output);
}
```



Walkthrough an Use Case



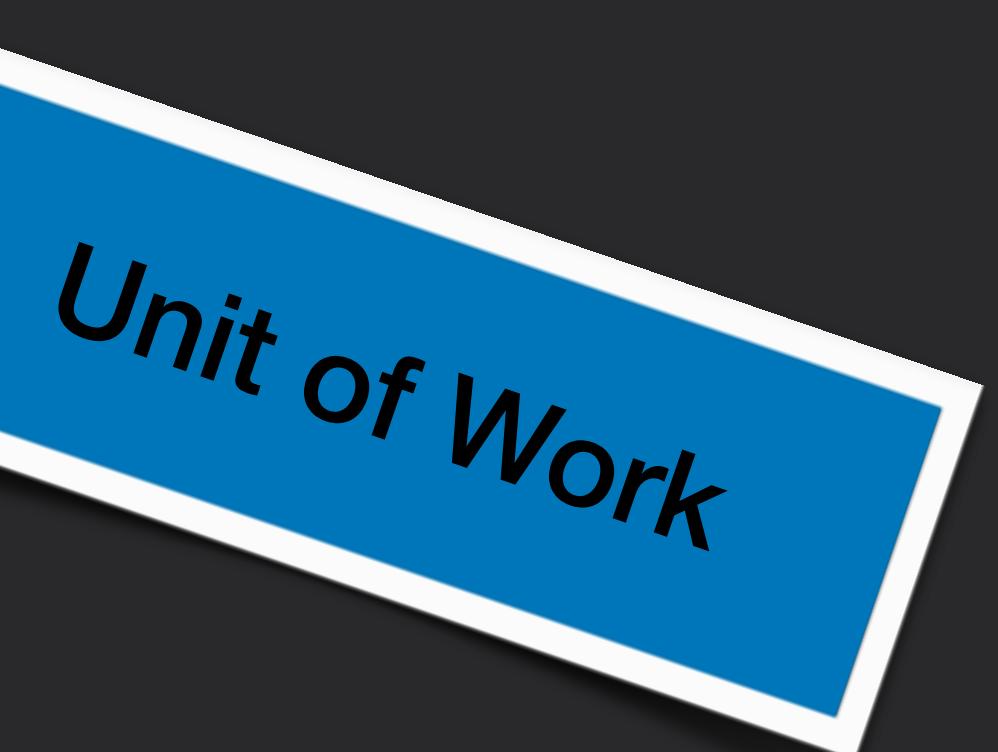
```
public sealed class UnitOfWork : IUnitOfWork, IDisposable
{
    3 references
    private readonly MangaContext _context;
    2 references
    private bool _disposed;

    0 references
    public UnitOfWork(MangaContext context) => this._context = context;

    /// <inheritdoc />
    public void Dispose()
    {
        this.Dispose(true);
        GC.SuppressFinalize(this);
    }

    /// <inheritdoc />
    4 references
    public async Task<int> Save()
    {
        int affectedRows = await this._context
            .SaveChangesAsync()
            .ConfigureAwait(false);
        return affectedRows;
    }

    private void Dispose(bool disposing)
    {
        if (!this._disposed)
        {
            if (disposing)
            {
                this._context.Dispose();
            }
        }
        this._disposed = true;
    }
}
```



Walkthrough an Use Case

Use Case

```
public async Task Execute(DepositInput input)
{
    if (input is null)
    {
        this._depositOutputPort
            .WriteError(Messages.InputIsNull);
        return;
    }

    IAccount account = await this._accountRepository
        .GetAccount(input.AccountId)
        .ConfigureAwait(false);

    if (account is null)
    {
        this._depositOutputPort
            .NotFound(Messages.AccountDoesNotExist);
        return;
    }

    ICredit credit = await this._accountService
        .Deposit(account, input.Amount)
        .ConfigureAwait(false);

    await this._unitOfWork
        .Save()
        .ConfigureAwait(false);

    this.BuildOutput(credit, account);
}

1 reference
private void BuildOutput(ICredit credit, IAccount account)
{
    var output = new DepositOutput(
        credit,
        account.GetCurrentBalance());
    this._depositOutputPort.Standard(output);
}
```



Walkthrough an Use Case

```
public sealed class DepositPresenter : IDepositOutputPort
{
    /// <summary>
    /// </summary>
    /// <returns></returns>
    4 references
    public IActionResult ViewModel { get; private set; } = new NoContentResult();

    /// <summary>
    /// </summary>
    /// <param name="message"></param>
    8 references
    public void NotFound(string message) => this.ViewModel = new NotFoundObjectResult(message);

    /// <summary>
    /// </summary>
    /// <param name="output"></param>
    8 references
    public void Standard([DepositOutput output]) You, 7 days ago • Fix Naming (#201)
    {
        var depositEntity = (Credit)output.Transaction;
        var depositResponse = new DepositResponse(
            depositEntity.Amount.ToDecimal(),
            Credit.Description,
            depositEntity.TransactionDate,
            output.UpdatedBalance.ToDecimal());
        this.ViewModel = new ObjectResult(depositResponse);
    }

    /// <summary>
    /// </summary>
    /// <param name="message"></param>
    8 references
    public void WriteError(string message) => this.ViewModel = new BadRequestObjectResult(message);
}
```

Walkthrough an Use Case

```
public sealed class AccountsController : ControllerBase
{
    /// <summary>
    ///     Deposit on an account.
    /// </summary>
    /// <response code="200">The updated balance.</response>
    /// <response code="400">Bad request.</response>
    /// <response code="500">Error.</response>
    /// <param name="mediator"></param>
    /// <param name="presenter"></param>
    /// <param name="request">The request to deposit.</param>
    /// <returns>The updated balance.</returns>
    [Authorize]
    [HttpPatch("Deposit")]
    [ProducesResponseType(StatusCodes.Status200OK, Type = typeof(DepositResponse))]
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
    [ProducesResponseType(StatusCodes.Status500InternalServerError)]
    0 references

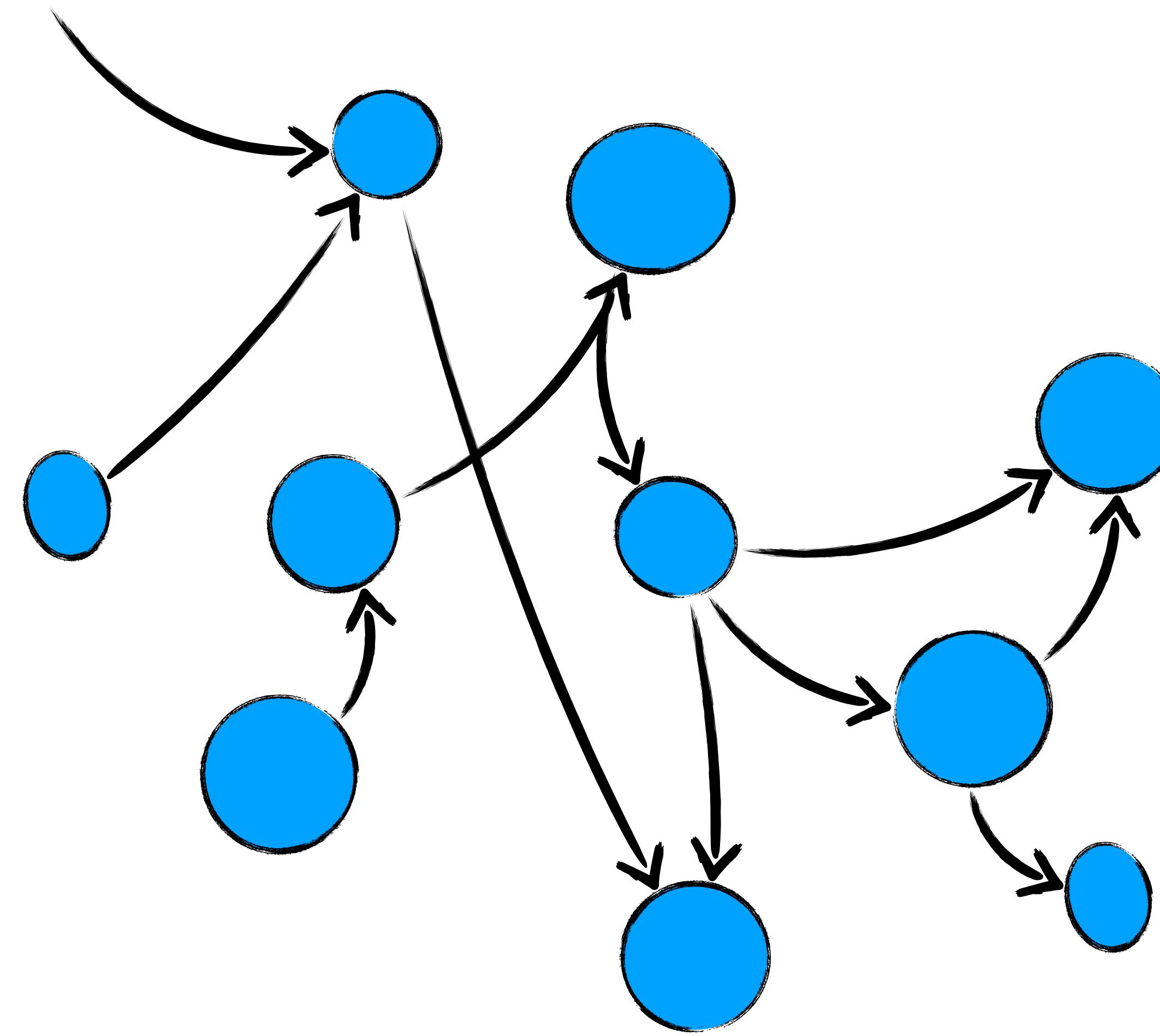
    public async Task<IActionResult> Deposit(
        [FromServices] IMediator mediator, [FromServices] DepositPresenter presenter,
        [FromForm] [Required] DepositRequest request)
    {
        var input = new DepositInput(request.AccountId, request.Amount);
        await mediator.PublishAsync(input)
            .ConfigureAwait(false);
        return presenter.ViewModel;
    }
}
```

Controller

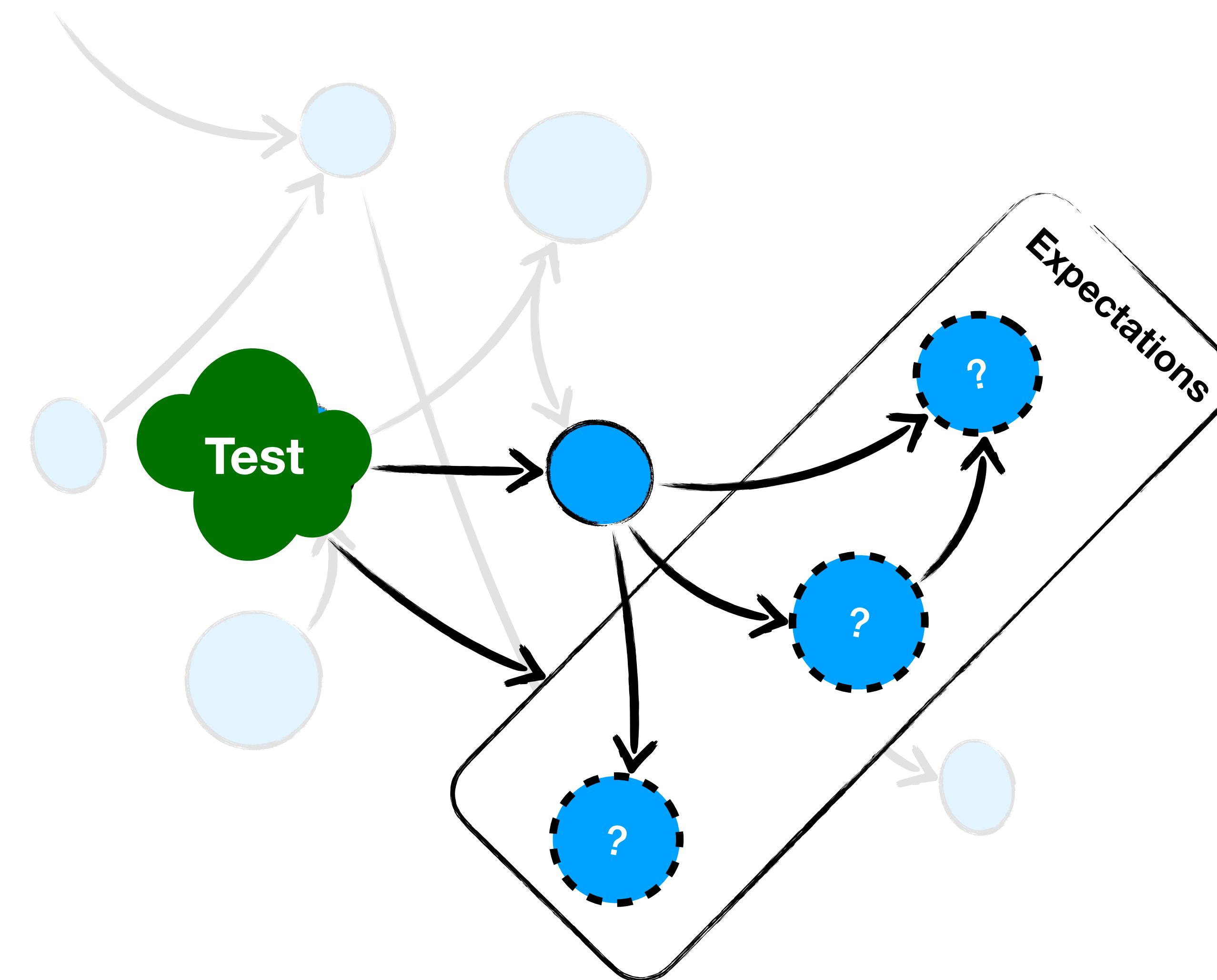


Testing Strategy

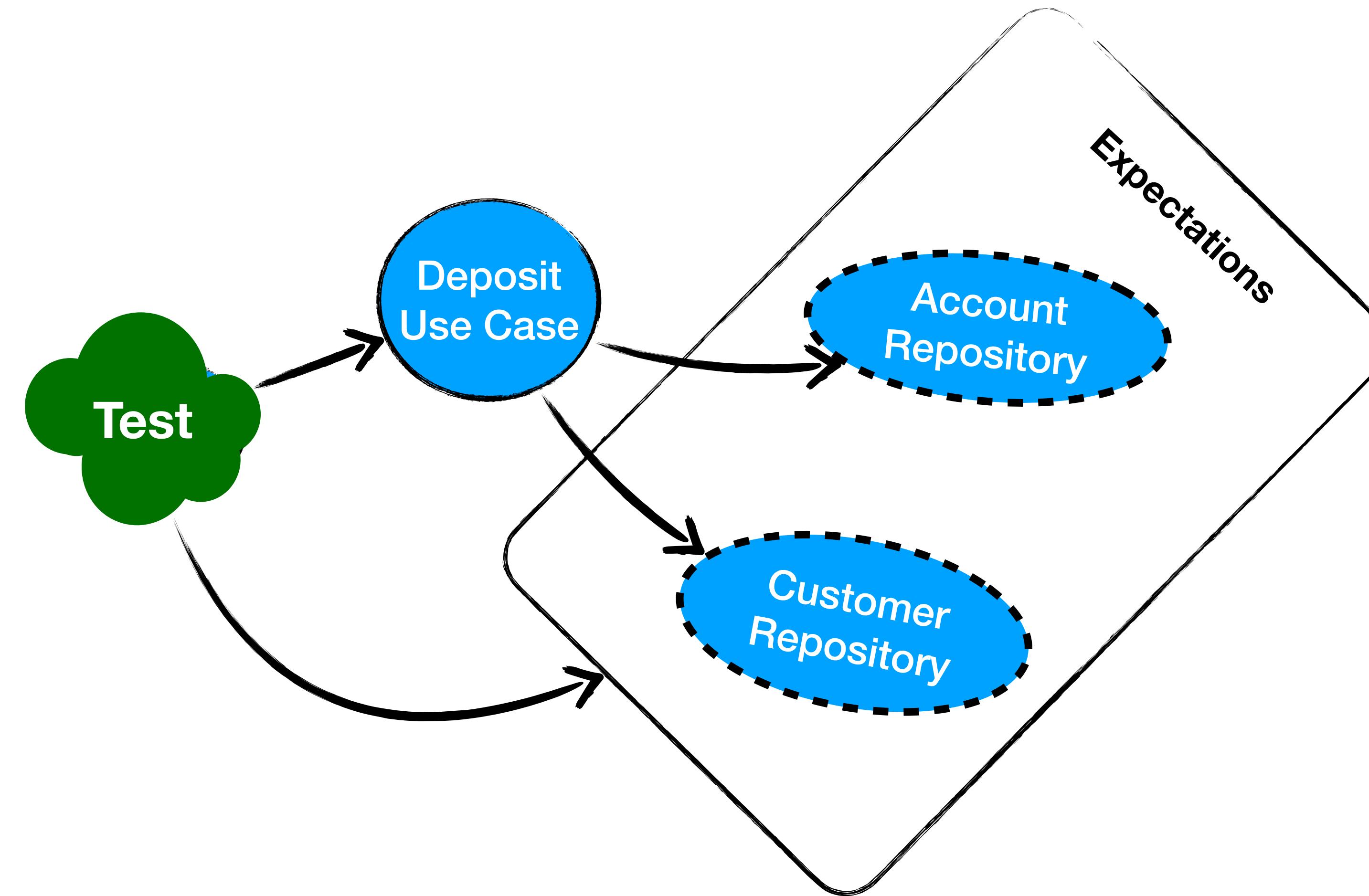
A Web of Objects



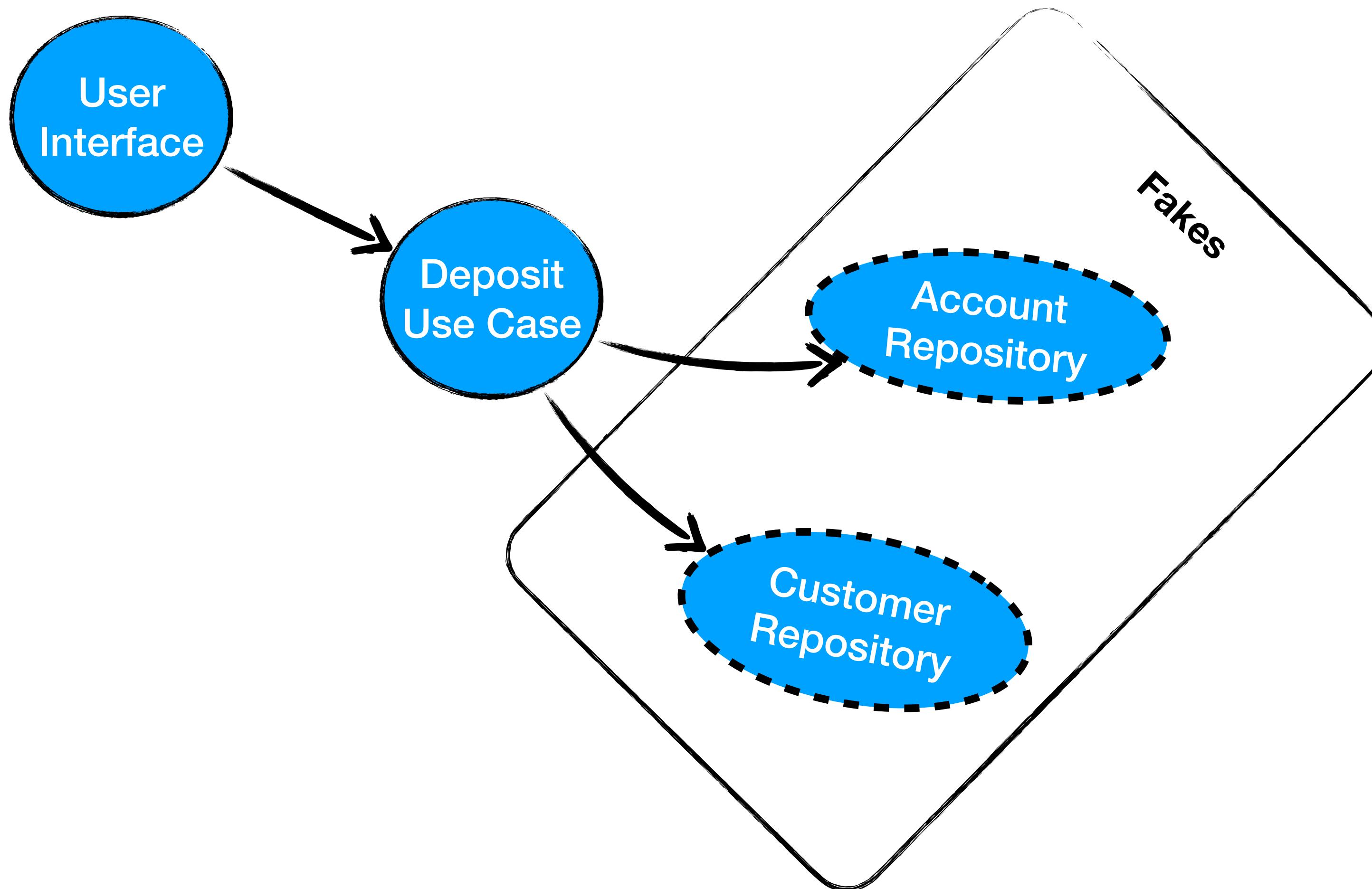
Testing



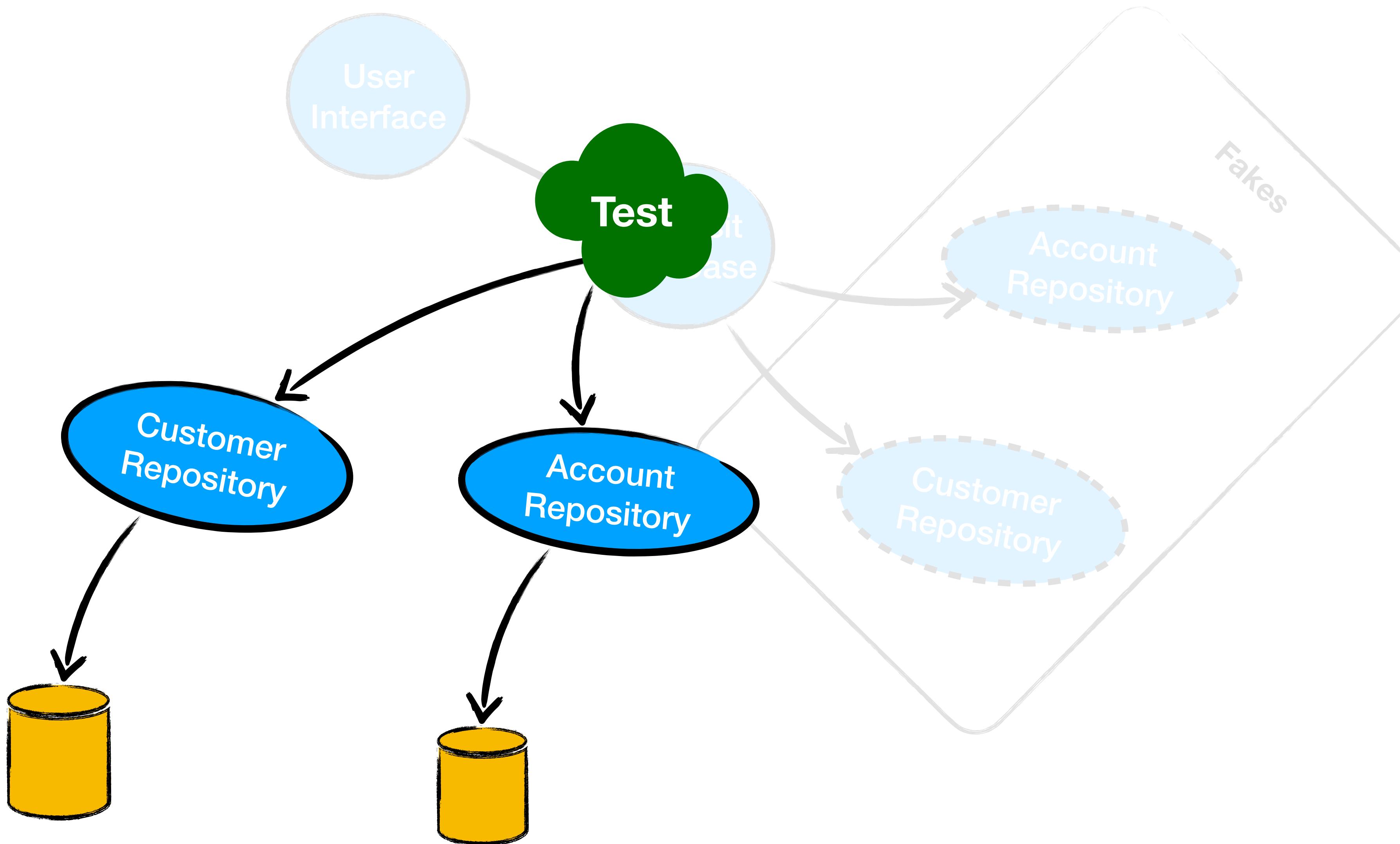
The First Test



The Walking Skeleton



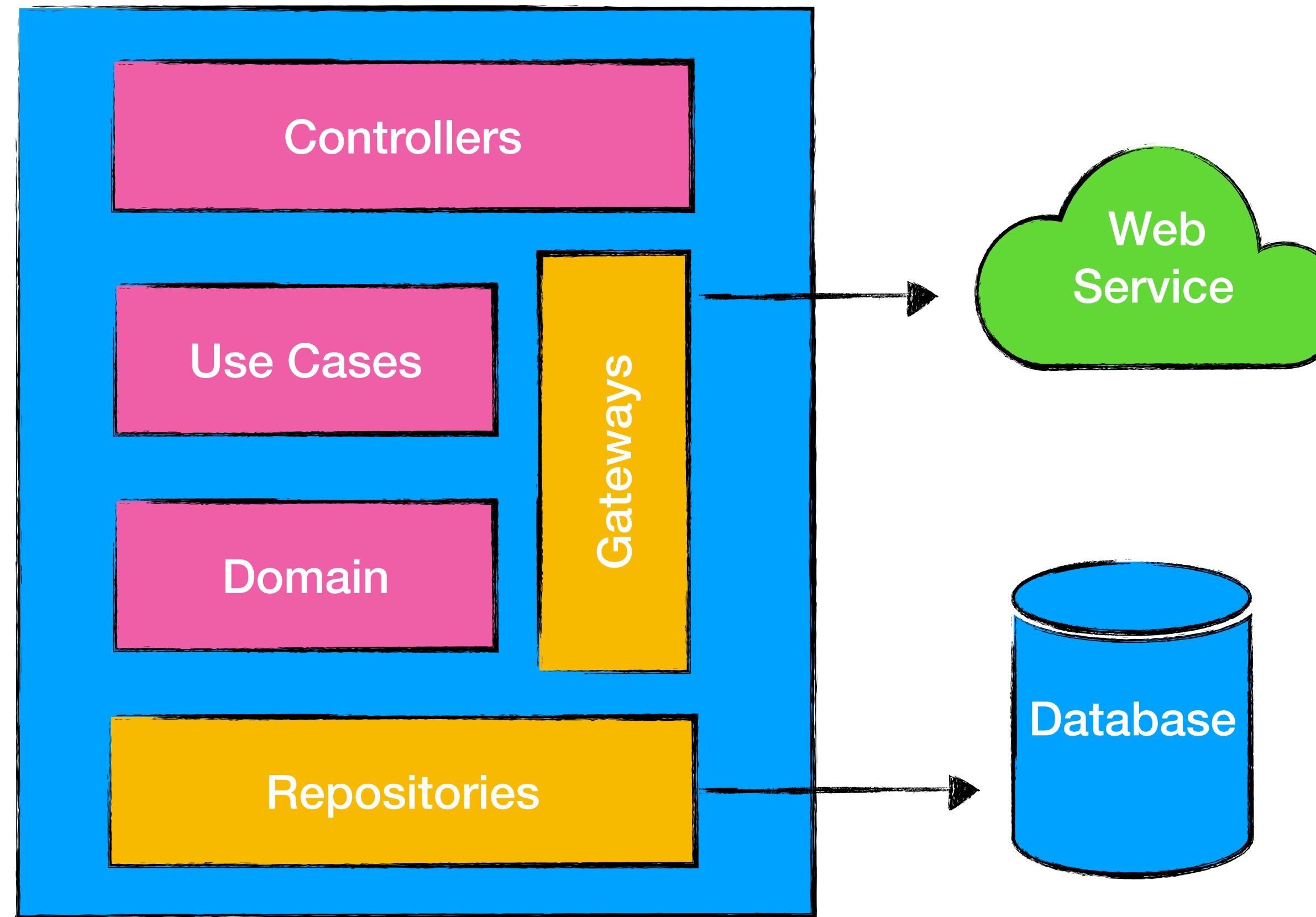
Integration Tests



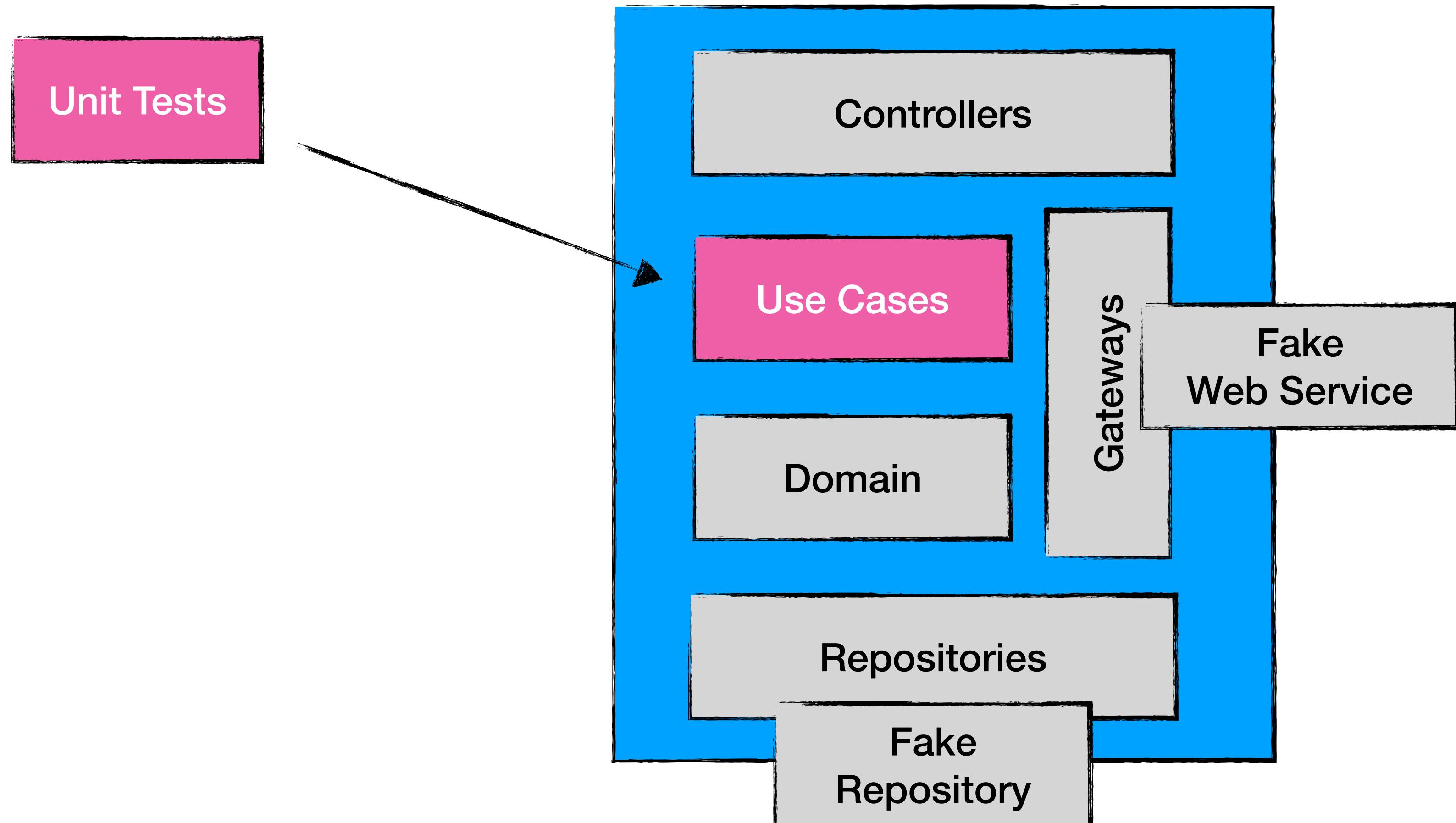
Tests Overview

	In Process	Out of Process
Memory	Within the application memory	Reads I/O. Secondary memory.
Speed	Fast	Slow
Setup Effort	Easy to design a variety of cases	Requires extensive setup
Samples	Use Cases, Domain	Repositories, Services

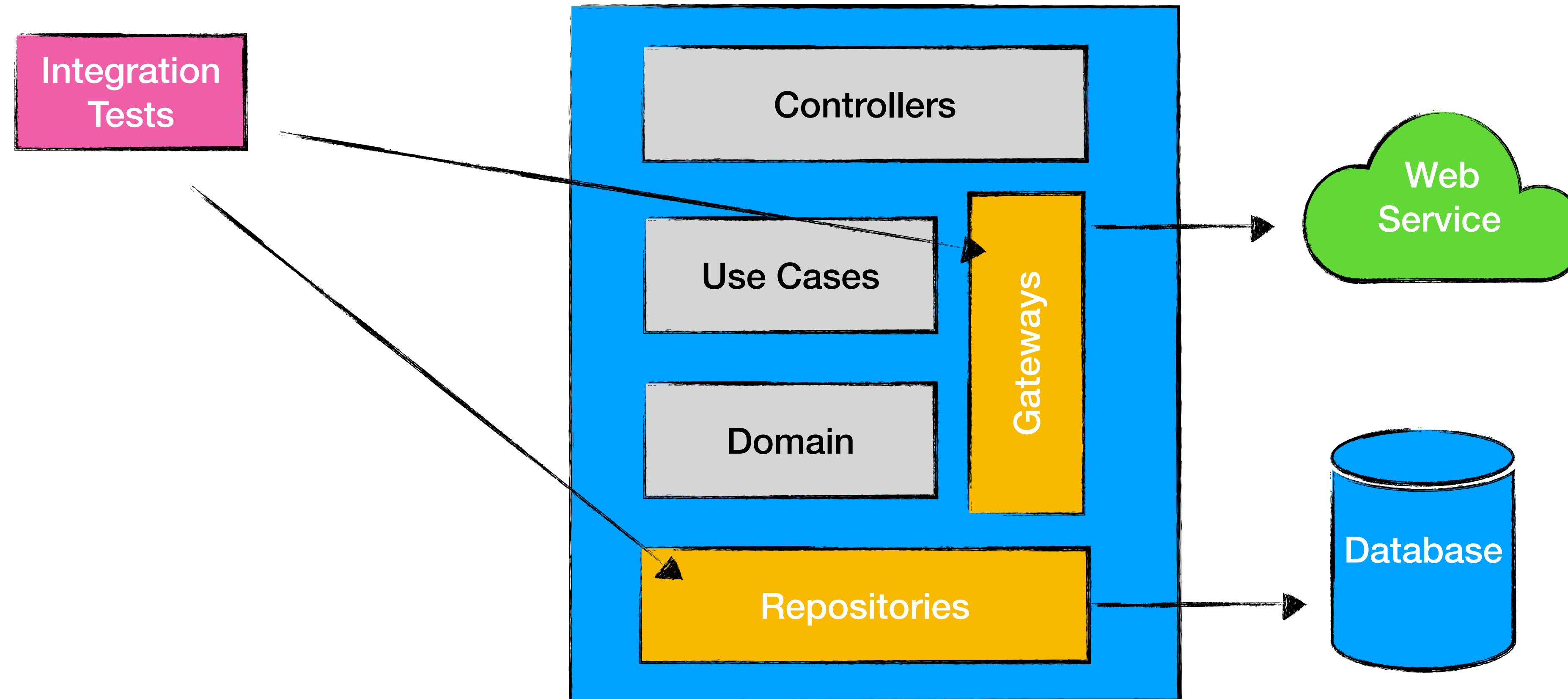
Software Skeleton



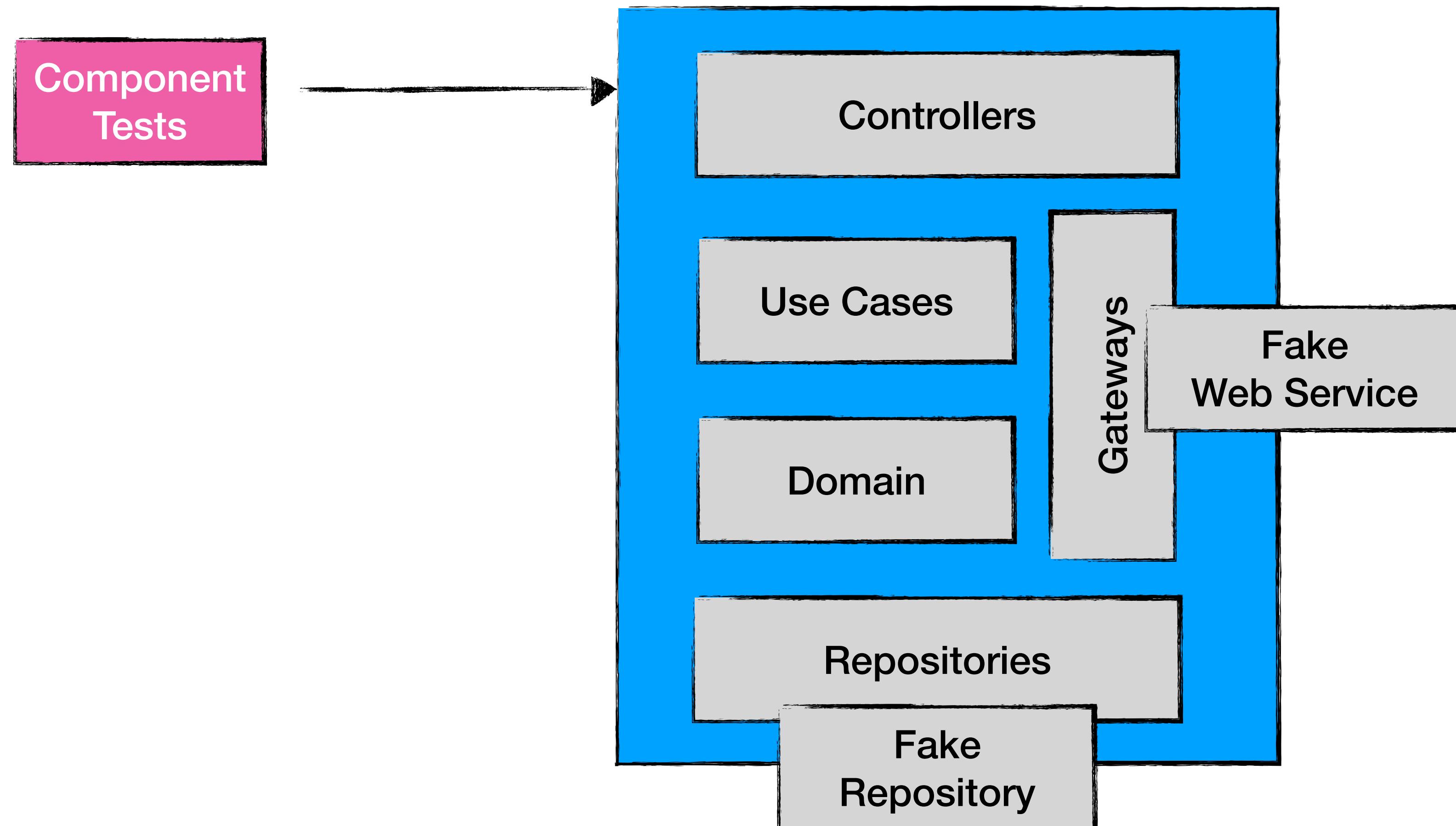
Unit Tests



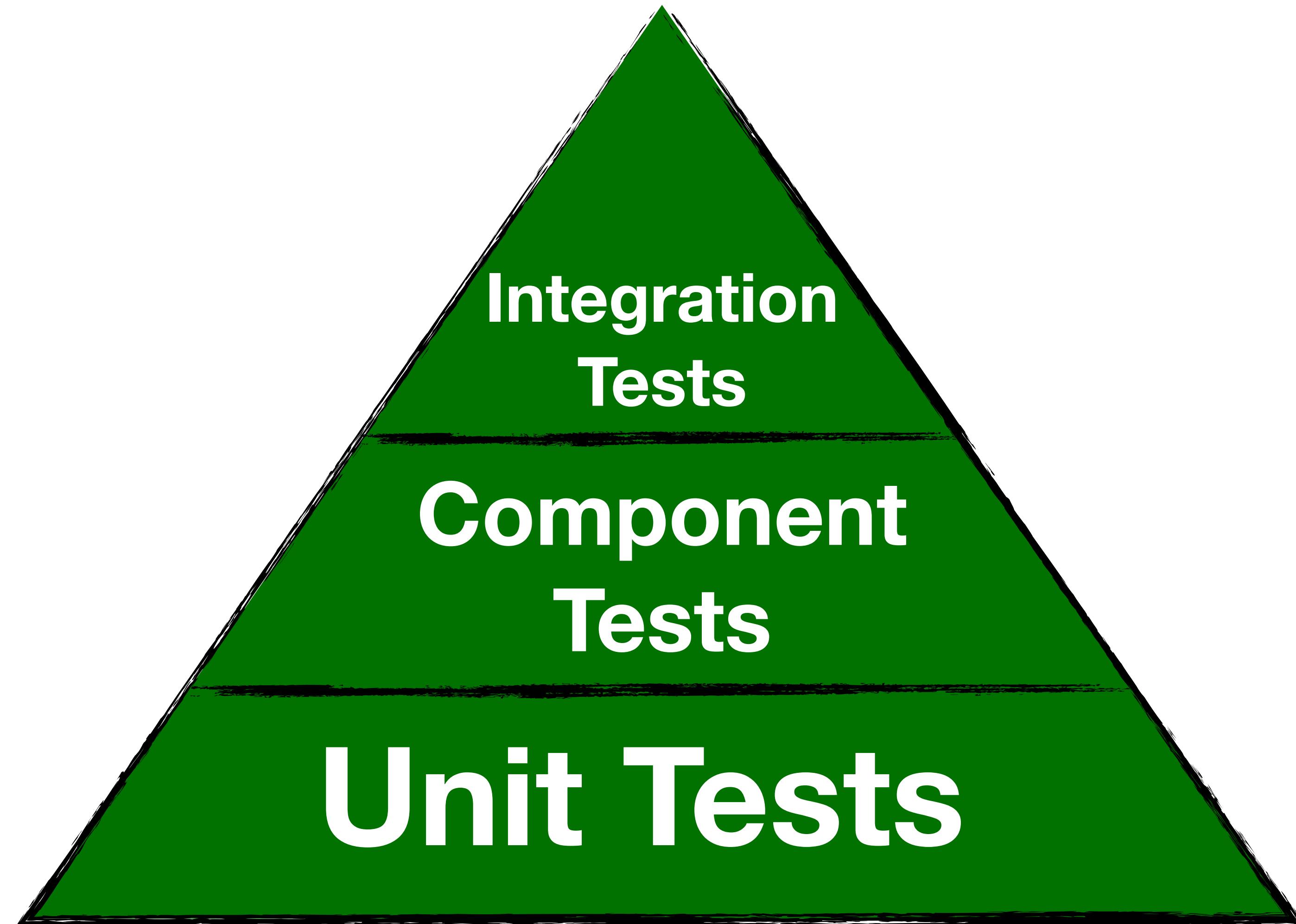
Integration Tests



Component Tests



Test Pyramid



Wrapping up Clean Architecture

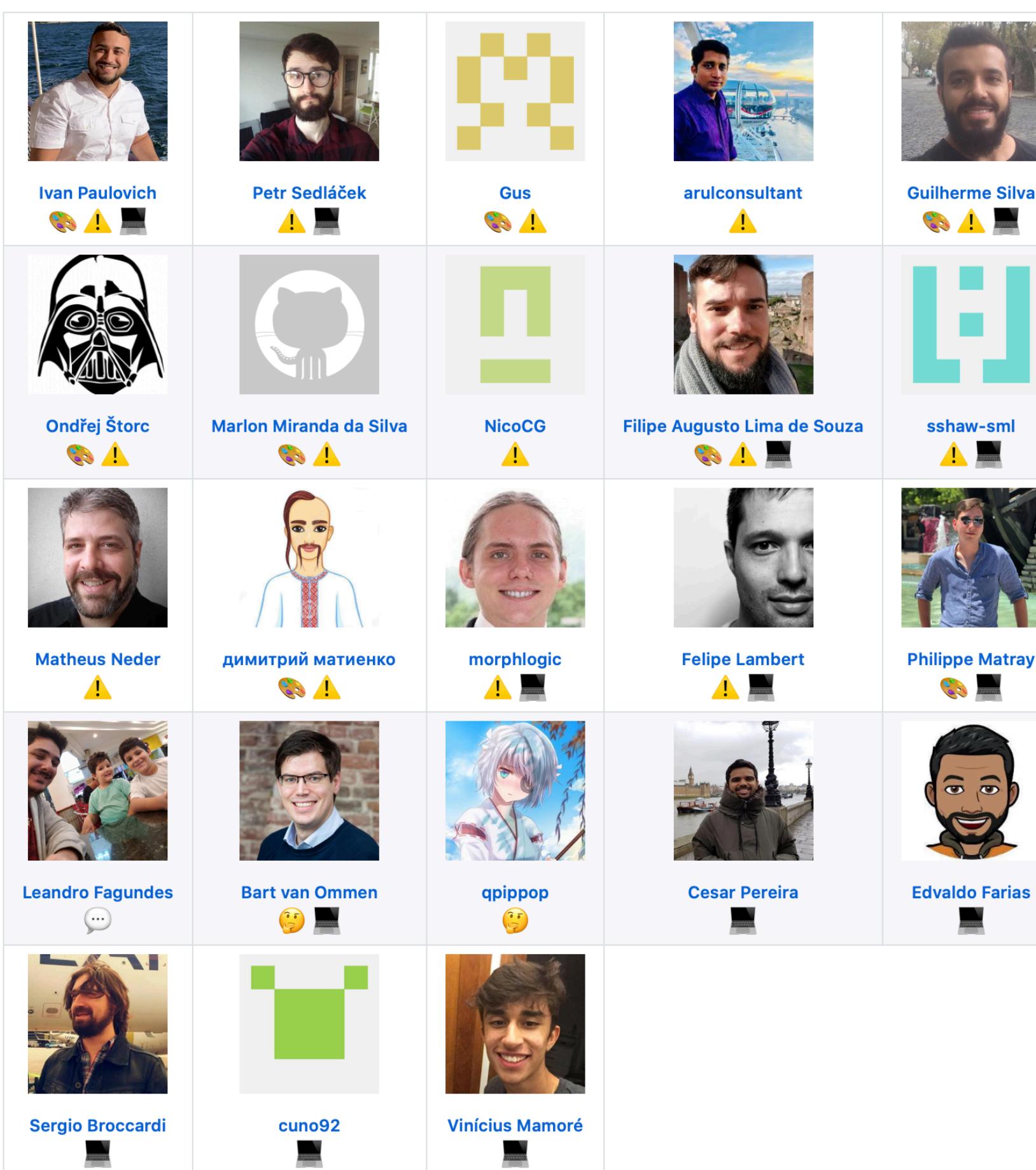
- Clean Architecture is about usage and the use cases are the central organizing structure.
- Use cases implementation are guided by tests.
- The User Interface and Persistence are designed to fulfil the core needs (not the opposite!).
- Defer decisions by implementing the **simplest component first**.

References

<https://github.com/ivanpaulovich/clean-architecture-manga>

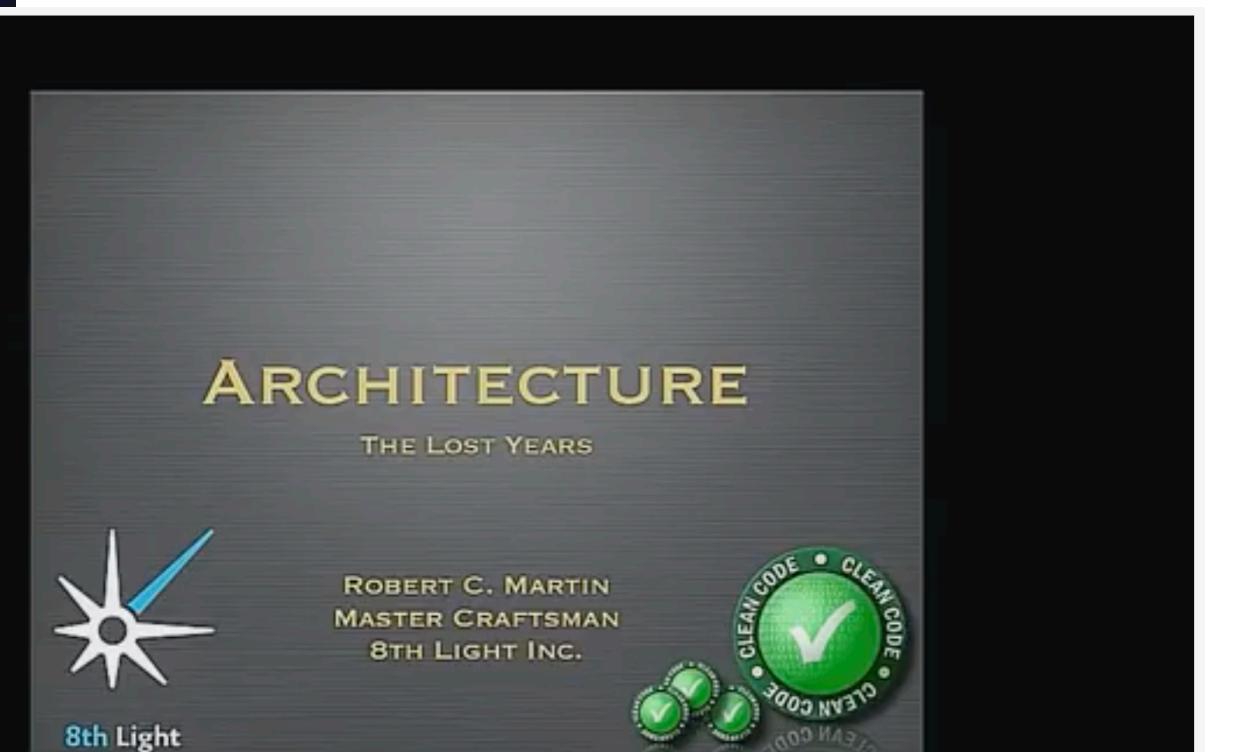
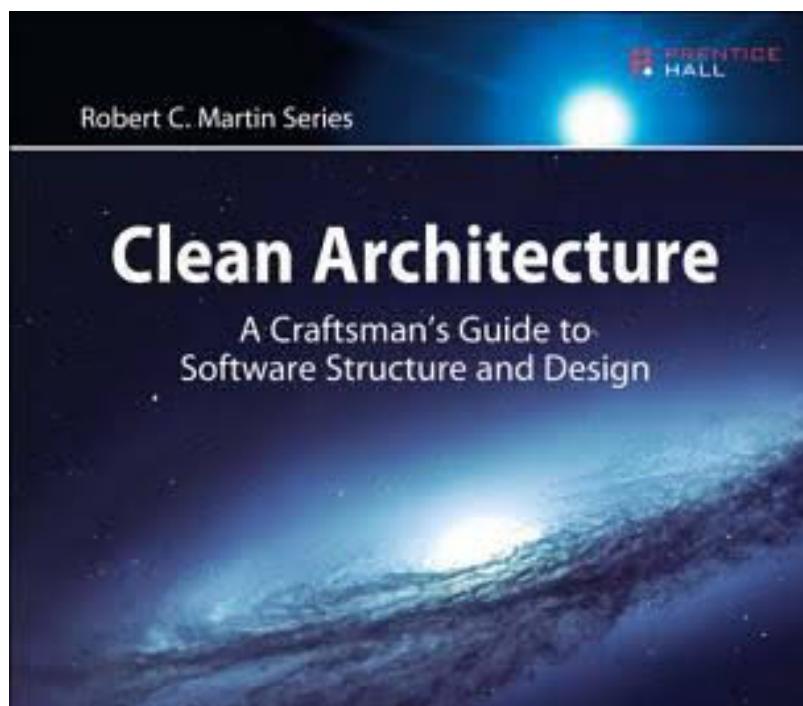
Contributors ✨

Thanks goes to these wonderful people (emoji key):

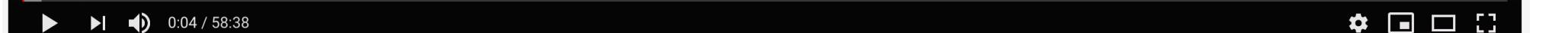


<https://cleancoders.com>

Clean Code: Component Design
Clean Code: SOLID Principles
Clean Code: Fundamentals



Robert C Martin - Clean Architecture and Design





#CleanArchitectureManga 🍄

Ivan Paulovich

ivanpaulovich

Sponsor

Tech Lead, Software Engineer, 20+ GitHub projects about Clean Architecture, SOLID, DDD and TDD. Speaker/Streamer.

Microsoft rMVP.

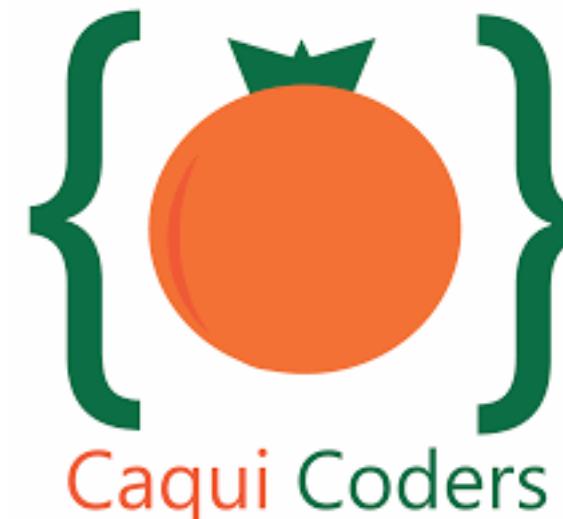
Nordax Bank

Stockholm, Sweden

Sign in to view email

<https://paulovich.net>

@ivanpaulovich



Ask me
2 questions!

Fork me on GitHub

[CleanArchitectureVSCodeSnippets](#)

Clean Architecture C# Snippets for Visual Studio Code

● TypeScript ⭐ 6

[FluentMediator](#)

FluentMediator is an unobtrusive library that allows developers to build custom pipelines for Commands, Queries and Events.

● C# ⭐ 83 ⚡ 11

[clean-architecture-manga](#)

Template

Clean Architecture with .NET Core 3.1, C# 8 and React+Redux. Use cases as central organizing structure, completely testable, decoupled from frameworks

● C# ⭐ 1.6k ⚡ 304

[todo](#)

Command-Line Task management with storage on your GitHub 🔥

● C# ⭐ 94 ⚡ 16

[dotnet-new-caju](#)

Template

Learn Clean Architecture with .NET Core 3.0 🔥

● C# ⭐ 202 ⚡ 30

[event-sourcing-jambo](#)

An Hexagonal Architecture with DDD + Aggregates + Event Sourcing using .NET Core, Kafka e MongoDB (Blog Engine)

● C# ⭐ 146 ⚡ 62