Clean Architecture
Patterns, Practices, and Principles

@MatthewRenze
#CodeMash
Overview

1. Clean Architecture
Overview

1. Clean Architecture
2. Domain-Centric Architecture
Overview

1. Clean Architecture
2. Domain-Centric Architecture
3. Application Layer
Overview

1. Clean Architecture
2. Domain-Centric Architecture
3. Application Layer
4. Commands and Queries
Overview

1. Clean Architecture
2. Domain-Centric Architecture
3. Application Layer
4. Commands and Queries
5. Functional Organization
Overview

1. Clean Architecture
2. Domain-Centric Architecture
3. Application Layer
4. Commands and Queries
5. Functional Organization
6. Microservices
Focus

Enterprise Architecture
Focus

Enterprise Architecture
Modern equivalent of 3-Layer
Focus

Enterprise Architecture
Modern equivalent of 3-Layer
Generally applicable
Focus

Enterprise Architecture
Modern equivalent of 3-Layer
Generally applicable
6 Key Points
Focus

Enterprise Architecture
Modern equivalent of 3-Layer
Generally applicable
6 Key Points
Q & A
What is Software Architecture?

What is Software Architecture?

High-level

What is Software Architecture?

High-level Structure

What is Software Architecture?

High-level Structure
Layers

What is Software Architecture?

High-level Structure
Layers
Components

What is Software Architecture?

High-level Structure
Layers
Components
Relationships

Levels of Architectural Abstraction

- System
- Sub-systems
- Layers
- Components
- Classes
- Data and Methods
Levels of Architectural Abstraction

- System
- Sub-systems
- Layers
- Components
- Classes
- Data and Methods
Messy vs Clean Architecture
Messy vs Clean Architecture
Messy vs Clean Architecture
What Is Bad Architecture?

Complex
Inconsistent
Incoherent
Rigid
Brittle
Untestable
Unmaintainable
What Is Clean Architecture?

Simple
Understandable
Flexible
Emergent
Testable
Maintainable
What Is Clean Architecture?

Architecture that is designed for the inhabitants of the architecture... not for the architect... or the machine.
What Is Clean Architecture?

Architecture that is designed for the inhabitants of the architecture... not for the architect... or the machine
What Is Clean Architecture?

Architecture that is designed for the inhabitants of the architecture... not for the architect... or the machine
What Is Clean Architecture?

Architecture that is designed for the inhabitants of the architecture... not for the architect... or the machine.
What Is Clean Architecture?

Architecture that is designed for the inhabitants of the architecture... not for the architect... or the machine
Why Is Clean Architecture Important?

Cost/benefit?
Why Is Clean Architecture Important?

Cost/benefit
Minimize cost to maintain
Why Is Clean Architecture Important?

Cost/benefit
Minimize cost to maintain
Maximize business value
Why Is Clean Architecture Important?

Cost/benefit
Minimize cost to maintain
Maximize business value
Maximize total ROI
Decisions, Decisions, Decisions...

Context is king
Decisions, Decisions, Decisions...

Context is king
All decisions are a tradeoff
Decisions, Decisions, Decisions...

Context is king
All decisions are a tradeoff
Use your best judgement
Domain-Centric Architecture
Classic 3-layer Database-centric Architecture
Database- vs. Domain-centric Architecture
“The first concern of the architect is to make sure that the house is usable, it is not to ensure that the house is made of brick.”

– Uncle Bob
Essential vs. Detail
Essential vs. Detail

Space is essential
Essential vs. Detail

Space is essential
Usability is essential
Essential vs. Detail

Building material is a detail
Essential vs. Detail

Building material is a detail
Ornamentation is a detail
Essential vs. Detail
Essential vs. Detail

Domain is essential
Essential vs. Detail

Domain is essential
Use cases are essential
Essential vs. Detail

Presentation is a detail
Essential vs. Detail

Presentation is a detail
Persistence is a detail
Database- vs. Domain-centric Architecture
Database- vs. Domain-centric Architecture
Database- vs. Domain-centric Architecture
Hexagonal Architecture

Original source: http://alistair.cockburn.us/Hexagonal+architecture
Onion Architecture

Original source: http://jeffreypalermo.com/blog/the-onion-architecture-part-2/
Clean Architecture
It’s All the Same Thing

Original Source: http://blog.ploeh.dk/2013/12/03/layers-onions-ports-adapters-its-all-the-same/
Why Use Domain-Centric Architecture?

Pros
Focus on essential
Why Use Domain-Centric Architecture?

Pros
Focus on essential
Less coupling to details
Why Use Domain-Centric Architecture?

**Pros**
Focus on essential
Less coupling to details
Necessary for DDD
Why Use Domain-Centric Architecture?

**Pros**
- Focus on essential
- Less coupling to details
- Necessary for DDD

**Cons**
- Change is difficult
Why Use Domain-Centric Architecture?

Pros
Focus on essential
Less coupling to details
Necessary for DDD

Cons
Change is difficult
Requires extra thought
Why Use Domain-Centric Architecture?

Pros
Focus on essential
Less coupling to details
Necessary for DDD

Cons
Change is difficult
Requires extra thought
Initial higher cost
Application Layer
What Are Layers?

Levels of abstraction
Single-Responsibility Principle
Developer roles / skills
Multiple implementations
Varying rates of change
Classic 3-Layer Architecture

- Users
  - UI
  - Business Logic
  - Data Access
  - Database

Dependency
Modern 4-Layer Architecture

- Users
  - Presentation
    - Application
      - Domain
        - Persistence
          - Database
        - Infrastructure
          - OS

Cross-Cutting Concerns

Dependency
Application Layer

Implements use cases
Application Layer

- Implements use cases
- High-level application logic
Application Layer

Implements use cases
High-level application logic
Knows about domain
Application Layer

Implements use cases
High-level application logic
Knows about domain
No knowledge of other layers
Application Layer

- Implements use cases
- High-level application logic
- Knows about domain
- No knowledge of other layers
- Contains interfaces for details
Layer Dependencies
Layer Dependencies

Dependency inversion
Layer Dependencies

Dependency inversion
Inversion of control
Layer Dependencies

Dependency inversion
Inversion of control
Independent deployability
Layer Dependencies

Dependency inversion
Inversion of control
Independent deployability
Flexibility and maintainability
Users

Presentation

SalesController

Application

ICreateSaleCommand

CreateSaleCommand

IDatabaseContext

InventoryClient

Domain

Sale

Persistence

DatabaseContext

Cross-Cutting Concerns

IInventoryClient

DateService

Composition

Implements

Infrastructure
Why Use an Application Layer?

Pros
Focus is on use cases
Why Use an Application Layer?

Pros
Focus is on use cases
Easy to understand
Why Use an Application Layer?

Pros
Focus is on use cases
Easy to understand
Follows DIP
Why Use an Application Layer?

Pros
Focus is on use cases
Easy to understand
Follows DIP

Cons
Additional cost
Why Use an Application Layer?

**Pros**
Focus is on use cases
Easy to understand
Follows DIP

**Cons**
Additional cost
Requires extra thought
Why Use an Application Layer?

Pros
- Focus is on use cases
- Easy to understand
- Follows DIP

Cons
- Additional cost
- Requires extra thought
- IoC is counter-intuitive
Commands and Queries
Command-Query Separation

Command
Does something
Should modify state
Should not return a value
Command-Query Separation

**Command**
- Does something
- Should modify state
- Should not return a value

**Query**
- Answers a question
- Should not modify state
- Always returns a value
Command-Query Separation

Command
Does something
Should modify state
Should not return a value
(ideally)

Query
Answers a question
Should not modify state
Always returns a value

Avoid mixing the two!
CQRS Architectures

Diagram:
- Presentation
  - Queries
  - Data Access
  - Database
- Persistence
  - Domain
  - Commands

Data Flow: 
- From Users to Presentation
- From Presentation to Queries and Commands
- From Queries and Commands to Data Access and Persistence
- From Persistence to Database

CQRS Architectures
CQRS Architectures
CQRS Architectures

Data Flow
CQRS Type 1 – Single Database
CQRS Type 1 – Single Database

Data Flow

Users → Presentation

Queries → Data Access

Commands → Persistence

Domain

Database
CQRS Type 1 – Single Database
CQRS Type 2 – Read/Write Databases

[Diagram showing data flow from Users to Presentation, then to Queries and Commands, and further to Data Access (Read Database) and Persistence (Write Database).]
CQRS Type 2 – Read/Write Databases
CQRS Type 2 – Read/Write Databases
CQRS Type 2 – Read/Write Databases
CQRS Type 2 – Read/Write Databases
CQRS Type 3 – Event Sourcing

Diagram:
- Users
  - Presentation
    - Queries
    - Commands
      - Domain
      - Persistence
    - Data Access
      - Read Database
      - Event Store

Data Flow
CQRS Type 3 – Event Sourcing

- Presentation
  - Users
    - Queries
    - Commands
    - Domain
    - Persistence
    - Data Access
      - Read Database
    - Event Store
- Events
  - Sale Created
  - Item 1 Added
  - Item 2 Added
  - Payment Made
  - Sale Completed

Data Flow
CQRS Type 3 – Event Sourcing

Data Flow:
- Users
  - Presentation
    - Queries
    - Commands
    - Domain
    - Persistence
    - Data Access
      - Read Database
        - Event Store
  - Event Sourcing

Events:
- Sale Created
- Item 1 Added
- Item 2 Added
- Payment Made
- Sale Completed
CQRS Type 3 – Event Sourcing

Data Flow

Users

Presentation

Queries

Commands

Domain

Persistence

Data Access

Read Database

Event Store
CQRS Type 3 – Event Sourcing
CQRS Type 3 – Event Sourcing

Complete audit trail
CQRS Type 3 – Event Sourcing

Complete audit trail
Point-in-time reconstruction
CQRS Type 3 – Event Sourcing

Complete audit trail
Point-in-time reconstruction
Replay events
CQRS Type 3 – Event Sourcing

Complete audit trail
Point-in-time reconstruction
Replay events
Rebuild production database
Why Use CQRS?

Pros
More efficient design
Why Use CQRS?

Pros
More efficient design
Simpler within each stack
Why Use CQRS?

Pros
More efficient design
Simpler within each stack
Optimized performance
Why Use CQRS?

**Pros**
- More efficient design
- Simpler within each stack
- Optimized performance

**Cons**
- Inconsistent across stacks
Why Use CQRS?

**Pros**
- More efficient design
- Simpler within each stack
- Optimized performance

**Cons**
- Inconsistent across stacks
- Type 2 is more complex
Why Use CQRS?

Pros
- More efficient design
- Simpler within each stack
- Optimized performance

Cons
- Inconsistent across stacks
- Type 2 is more complex
- Type 3 might be overkill
Functional Organization
“The architecture should scream the intent of the system!”

– Uncle Bob
Bedroom
Dining Room
Kitchen
Entry
Utility
Bath
<table>
<thead>
<tr>
<th>Material</th>
<th>Quantity</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appliances</td>
<td>5</td>
<td>$5,000</td>
</tr>
<tr>
<td>Cabinets</td>
<td>10</td>
<td>$2,500</td>
</tr>
<tr>
<td>Doors</td>
<td>15</td>
<td>$750</td>
</tr>
<tr>
<td>Fixtures</td>
<td>12</td>
<td>$2,400</td>
</tr>
<tr>
<td>Floors</td>
<td>9</td>
<td>$4,000</td>
</tr>
<tr>
<td>Walls</td>
<td>20</td>
<td>$10,000</td>
</tr>
<tr>
<td>Windows</td>
<td>8</td>
<td>$2,500</td>
</tr>
</tbody>
</table>
Models

Views

Controllers
Controllers vs Vendors

- Models
- Views
- Controllers

VS

- Customers
- Products
- Vendors
Content
Controllers
Models
Scripts
Views
So what?
Why Use Functional Organization

Pros
Spatial locality
Why Use Functional Organization

Pros
Spatial locality
Easy to navigate
Why Use Functional Organization

Pros
Spatial locality
Easy to navigate
Avoid vendor lock-in
Why Use Functional Organization

Pros
Spatial locality
Easy to navigate
Avoid vendor lock-in

Cons
Lose framework conventions
Why Use Functional Organization

Pros
Spatial locality
Easy to navigate
Avoid vendor lock-in

Cons
Lose framework conventions
Lose automatic scaffolding
Why Use Functional Organization

**Pros**
- Spatial locality
- Easy to navigate
- Avoid vendor lock-in

**Cons**
- Lose framework conventions
- Lose automatic scaffolding
- Categorical is easier at first
Microservices
Components

UI
- Sales
- Support
- Inventory

Business
- Sales
- Support
- Inventory

Data Access
- Sales
- Support
- Inventory

Database
Components
Problem Domain

**Sales**
- Sales Opportunity
- Contact
- Sales Person
- Product
- Sales Territory

**Support**
- Support Ticket
- Customer
- Support Person
- Product
- Resolution
Single Domain Model
Single Domain Model
Single Domain Model

- Customer
  - Sales Territory
  - Sales Opportunity
  - Product
  - Support Ticket
  - Employee
  - Resolution
Single Domain Model
Overlapping Contexts

Sales

- Sales Territory
- Sales Opportunity

Support

- Customer
- Product
- Support Ticket
- Resolution

Employee
Bounded Contexts

Sales
- Sales Territory
- Sales Opportunity
- Product
- Sales Person

Support
- Customer
- Product
- Support Ticket
- Resolution
- Support Person
Microservice Architectures
Microservice Architectures

Subdivide system
Microservice Architectures

Subdivide system
Light-weight APIs
Microservice Architectures

Subdivide system
Light-weight APIs
Small teams
Microservice Architectures

Independent
Microservice Architectures

Independent
Similar to SOA
Microservice Architectures

Independent
Similar to SOA
Size matters
Why Use Microservices?

Pros
Less cost for large domains
Why Use Microservices?

Pros
Less cost for large domains
Smaller teams
Why Use Microservices?

Pros
Less cost for large domains
Smaller teams
Independence
Why Use Microservices?

Pros
Less cost for large domains
Smaller teams
Independence

Cons
Only for large domains
Why Use Microservices?

Pros
Less cost for large domains
Smaller teams
Independence

Cons
Only for large domains
Higher up-front cost
Why Use Microservices?

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less cost for large domains</td>
<td>Only for large domains</td>
</tr>
<tr>
<td>Smaller teams</td>
<td>Higher up-front cost</td>
</tr>
<tr>
<td>Independence</td>
<td>Distributed system costs</td>
</tr>
</tbody>
</table>
Code Demo
Where to Go Next?
Where to Go Next?

Martin Fowler
Where to Go Next?

http://cleancoders.com/

Robert C. Martin
Where to Go Next?

Domain-Driven Design
Tackling Complexity in the Heart of Software

Eric Evans
Foreword by Martin Fowler

Eric Evans
Where to Go Next?

Greg Young

Udi Dahan
Where to Go Next?

Articles
Courses
Presentations
Source Code
Videos

www.matthewrenze.com
Clean Architecture: Patterns, Practices, and Principles

INTRODUCTION

Matthew Renze
SOFTWARE CONSULTANT
@matthewrenze  www.matthewrenze.com
Conclusion
Summary

Focus on the inhabitants
Summary

Focus on the inhabitants
Domain-centric Architecture
Summary

Focus on the inhabitants
Domain-centric Architecture
Application Layer
Summary

Focus on the inhabitants
Domain-centric Architecture
Application Layer
Commands and Queries
Summary

Focus on the inhabitants
Domain-centric Architecture
Application Layer
Commands and Queries
Functional Cohesion
Summary

Focus on the inhabitants
Domain-centric Architecture
Application Layer
Commands and Queries
Functional Cohesion
Bounded Contexts
Feedback

Very important to me!
One thing you liked?
One thing I could improve?
Contact Info

Matthew Renze
Data Science Consultant
Renze Consulting

Twitter: @matthewrenze
Email: info@matthewrenze.com
Website: www.matthewrenze.com

Thank You! : )