Clean Architecture
Patterns, Practices, and Principles

@matthewrenze
#DevSum17
About Me

Independent consultant

Education
  B.S. in Computer Science (ISU)
  B.A. in Philosophy (ISU)

Community
  Public Speaker
  Pluralsight Author
  Microsoft MVP
  ASPInsider
  Open-Source Software
Overview

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

Enterprise Architecture
Line-of-Business Applications
Modern equivalent of 3-Layer
Focus

Generally applicable
6 Key Points
Q & A
What is Software Architecture?

High-level Structure

Layers

Components

Relationships

Levels of Architectural Abstraction

- Data and Methods
- Classes
- Components
- Layers
- Sub-systems
- System
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
Minimize cost to maintain
Maximize business value
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
Earth
Mercury
Venus
Sun
Mars
Venus
Mercury
Jupiter
Saturn
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

Space is essential
Usability is essential
Essential vs. Detail

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

Domain is essential
Use cases are essential
Essential vs. Detail

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

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

User Interface
Application Services
Domain Services
Domain Model
Application Core

Tests
Infrastructure

DB
File

web service

Original source: http://jeffreypalermo.com/blog/the-onion-architecture-part-2/
Clean Architecture

Original source: http://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html
It’s All the Same Thing

Hexagonal

Onion

Clean

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
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
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
Application Layer

Implements use cases
High-level application logic
Application Layer

Knows about domain
No knowledge of other layers
Contains interfaces for details
Layer Dependencies

Dependency inversion
Inversion of control
Layer Dependencies

Dependency inversion
Inversion of control
Independent deployability
Flexibility and maintainability
Users

Presentation

SalesController

Application

ICreateSaleCommand

CreateSaleCommand

IDatabaseContext

InventoryClient

Domain

Sale

Persistence

DatabaseContext

Interface

IDatabaseContext

ICreateSaleCommand

CreateSaleCommand

IDateService

Cross-Cutting Concerns

DateService

Composition

->

Implements

<->
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
- 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

<table>
<thead>
<tr>
<th>Command</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does something</td>
<td>Answers a question</td>
</tr>
<tr>
<td>Should modify state</td>
<td>Should not modify state</td>
</tr>
<tr>
<td>Should not return a value</td>
<td>Always returns a value</td>
</tr>
</tbody>
</table>

*(ideally)*

Avoid mixing the two!
CQRS Architectures

Presentation

Queries

Data Access

Database

Users

Commands

Domain

Persistence

Data Flow
CQRS Architectures

- Presentation
  - Queries
  - Data Access
  - Database
- Users
- Commands
- Domain
- Persistence

Data Flow
CQRS Architectures

Presentation

Users

Queries

Commands

Domain

Data Access

Persistence

Database

Data Flow
CQRS Type 1 – Single Database

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

- Users
- Presentation
  - Queries
  - Data Access
  - Database
- Commands
  - Domain
  - Persistence

→ Data Flow
CQRS Type 2 – Read/Write Databases
CQRS Type 2 – Read/Write Databases

Presentation

Queries

Data Access

Read Database

Commands

Domain

Persistence

Write Database

Data Flow
CQRS Type 2 – Read/Write Databases

Data Flow
CQRS Type 2 – Read/Write Databases
CQRS Type 2 – Read/Write Databases
CQRS Type 3 – Event Sourcing

Users

Presentation

Queries

Commands

Domain

Persistence

Data Access

Read Database

Event Store

Data Flow
CQRS Type 3 – Event Sourcing

Data Flow

Users

Presentation

Queries

Commands

Domain

Persistence

Event Store

Read Database

Events

Sale Created

Item 1 Added

Item 2 Added

Payment Made

Sale Completed

→ Data Flow
CQRS Type 3 – Event Sourcing

Data Flow:

- Sale Created
- Item 1 Added
- Item 2 Added
- Payment Made
- Sale Completed

Events:

- Sale Created
- Item 1 Added
- Item 2 Added
- Payment Made
- Sale Completed

Data Access:

- Read Database

Persistence:

- Event Store

Domain:

- Commands

Presentation:

- Queries

Users:

- Presentation

→ Data Flow
CQRS Type 3 – Event Sourcing
CQRS Type 3 – Event Sourcing

Complete audit trail
Point-in-time reconstruction
Replay events
Rebuild production database
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
Type 2 is more complex
Type 3 might be overkill
Functional Organization
“The architecture should scream the intent of the system!”

– Uncle Bob
<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

Views

Models

Controllers

_views

Models

Controllers

Views

Models

Controllers
Content
Controllers
Models
Scripts
Views
So what?
vs
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
- Lose automatic scaffolding
- Categorical is easier at first
Microservices
Components

UI
Sales | Support | Inventory

Business
Sales | Support | Inventory

Data Access
Sales | Support | Inventory

Database
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

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

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

Sales Territory — Sales Opportunity — Product — Support Ticket — Resolution

Customer — Employee
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

- Sales
- Support
- Inventory
- Marketing
- HR
Microservice Architectures

Independent
Similar to SOA
Microservice Architectures

Independent
Similar to SOA
Size matters
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
- Higher up-front cost
- Distributed system costs
Code Demo
Where to Go Next?

http://cleancoders.com/
Where to Go Next?

Eric Evans

Domain-Driven Design
Tackling Complexity in the Heart of Software

Eric Evans
Foreword by Martin Fowler
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: matthew@matthewrenze.com
Website: www.matthewrenze.com

Thank You! : )