

# Domain Driven Design

---

And when not to use it

Cincinnati SC 2019-11-06

[mark@agiledna.com](mailto:mark@agiledna.com)

<https://github.com/mwindholtz/presentations/DDD-SC-Cinci>



# About me: mark@agiledna.com

---

- Built ...
  - eCommerce site for ~12 years
  - Memberships site for ~8 years
- Elixir: CC Processing, Medical Research, Energy Grid
- DDD Since 2006
- TDD since 1999 — sUnit
  - Learned from Kent Beck, Bob Martin, Eric Evans

# About YOU

---

- What Drives Your Product Code Design?

# Some Observed Drivers ...

---

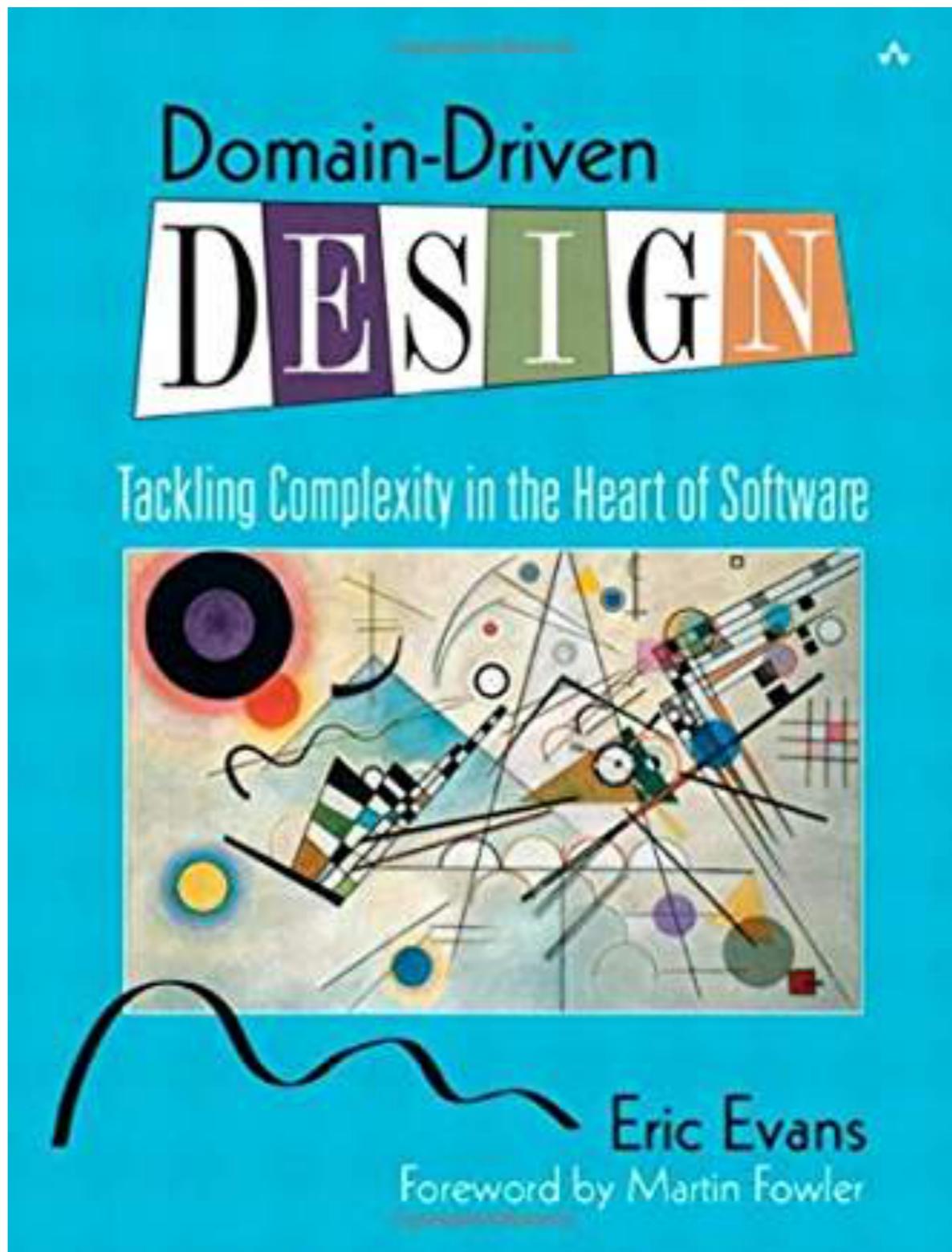
- UI Screens / API
- Database Model
- Doing Risk Items First, or Last
- Architect's Personal Issues - opposite of last project
- Features, Features, more Features



# Domain Driven Design, 2003

---

- Tackling Complexity in the Heart of S/W
- By Eric Evans
- “The Big Blue Book”



# Domain-Driven Design (DDD)

---

- For ...
  - complex needs
  - connects implementation to evolving model
  - **DOMAIN** is the problem space
  - **MODEL** is an abstracted potential solution
  - Code is code

# DDD: From Debt to Dividend

---

- **Tech Debt**
  - Paying interest plus principle to temporarily move faster
- **Tech Dividend**
  - Getting extra value on deep design to unlock extra features

# DDD is a Set of Principles

---

- **Core Domain** - focus here
- **Explore Models** in a Collaboration of
  - Domain experts and the software team
  - Separate Business from Technical
- Speak in a **Ubiquitous Language**
  - within a Bounded Context

# When to Apply Domain Design

---

- **For Simple systems**

- No worries. It fits inside a person's head.

- **For Medium systems**

- No worries. Hire smart people so that ..
  - It fits inside a person's head. Oh and write loads of *Documentation!* \*\*

- **For Complex systems**

\*\* has an unknown expiration date.

\*\* may also be wrong.

\*\*Other restrictions may apply.

- Starting is ok. It initially still fits inside a person's head.
  - Then Documents help a while
  - But As It Grows ...

# Warning !

---

- **Simple** Systems ...
  - Become **Medium** System ...
    - Which Become **Complex** systems

# Why “Agile” Software becomes a Mess

---

- Feature story
- Design, design, design 
- Feature story, Feature story
- Design. 
- Feature story, Feature story, Feature story, Feature story,  
Feature story 
- Feature story, Feature story, Feature story, Feature story,  
Feature story 

# Code Structure: **Big Ball Of Mud**

---

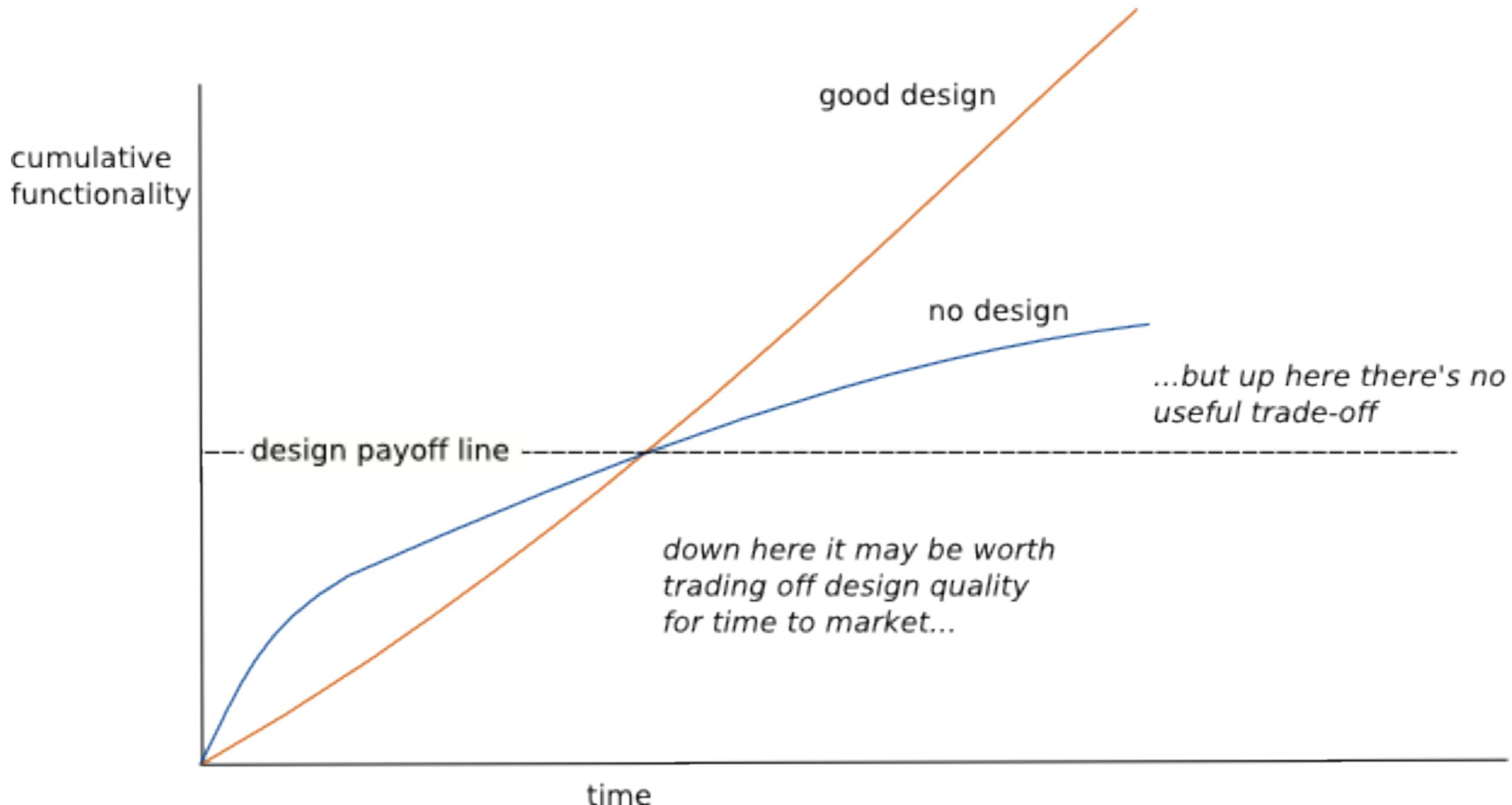
Process Diagnosis:  
**Featureitis**

Design:  
Preventing ripple effects



# Design Stamina Hypothesis

- Martin Fowler



# Software Craftsmanship – IS NOT ENOUGH –

---

- Refactoring
- Better names
- Test Driven Design
- Continuous Single Integration
- Something is still missing





**DDD Europe** @ddd\_eu

5d

"No refactoring without  
remodelling. Clean Code by itself  
cannot save a rotten model."

From "Technical debt isn't  
technical" by Einar Høst

@einarwh at #DDDEU 2019  
[buff.ly/2WGYyss](https://buff.ly/2WGYyss)

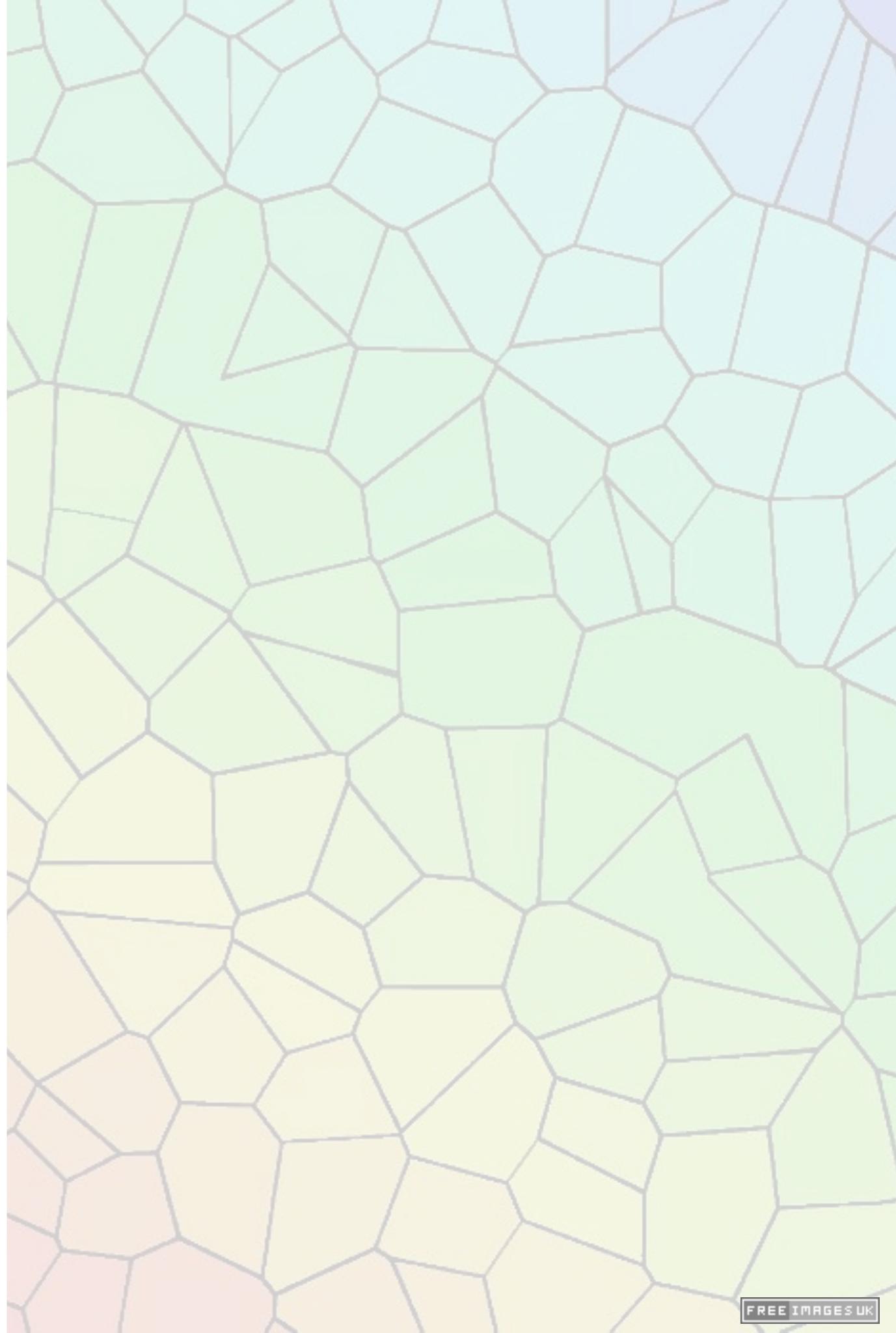
💬 17 ❤️ 35 ...

Technical debt isn't  
technical - Einar Høst -  
DDD Europe 2019

# DDD Prerequisites

---

- **Close Collaboration of Business**
- **Development in Ubiquitous Language**
- **Iterative Development**



# DDD is Difficult to Explain. Why?

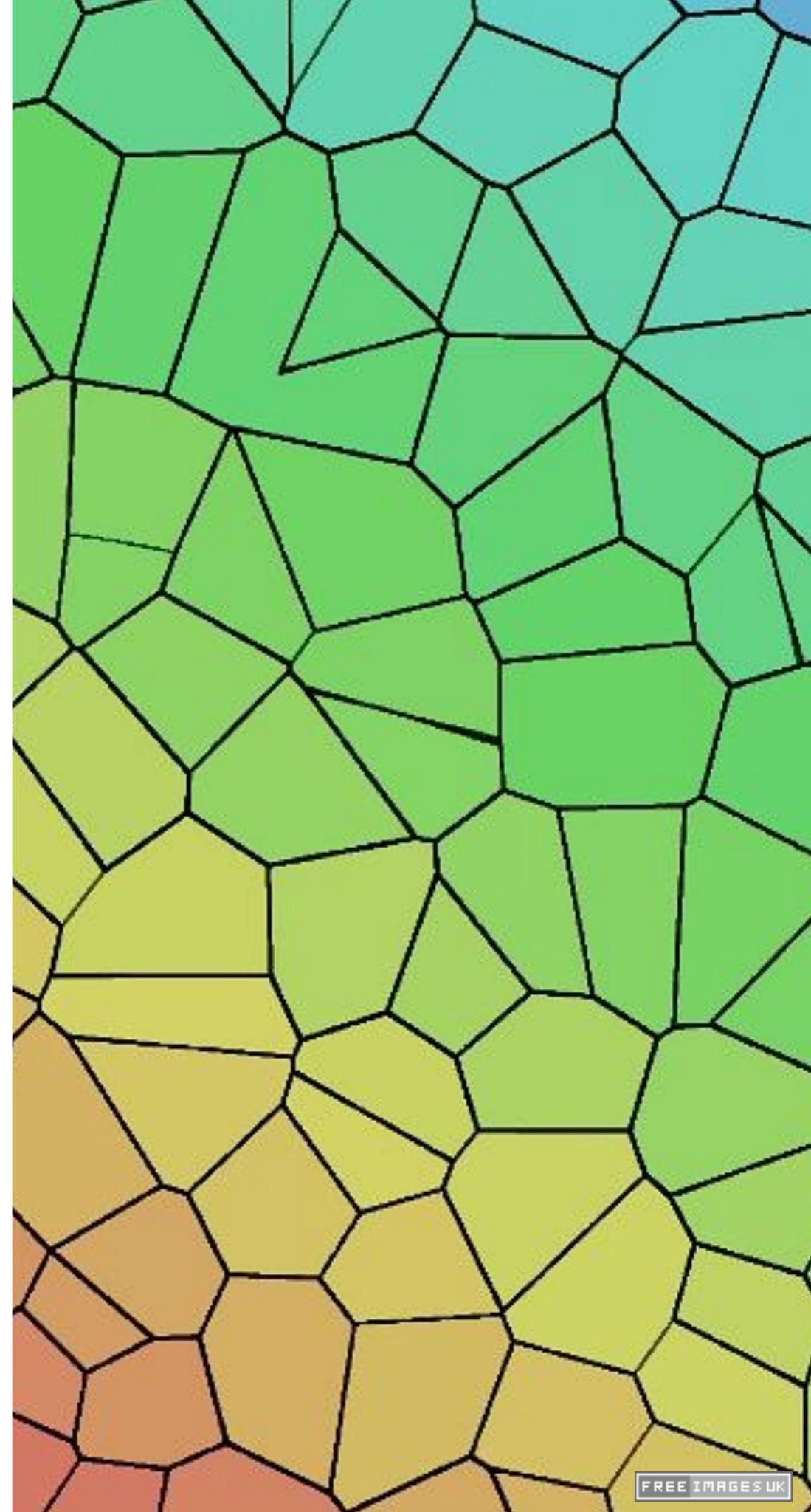
---

- Since the ***Problem*** is Complex and Subtle
- So is the ***Solution***
- Difficult to scale down into examples
- **Large Vocabulary of Interrelated Patterns**
  - Pattern Languages

# Doing DDD - mark's version

---

- ***Core Domain***, and SubDomains
- ***Bounded Contexts***
- ***Ubiquitous Language***
- ***Building Blocks*** are *Tactical Patterns*
- Repeat ... and Revisit



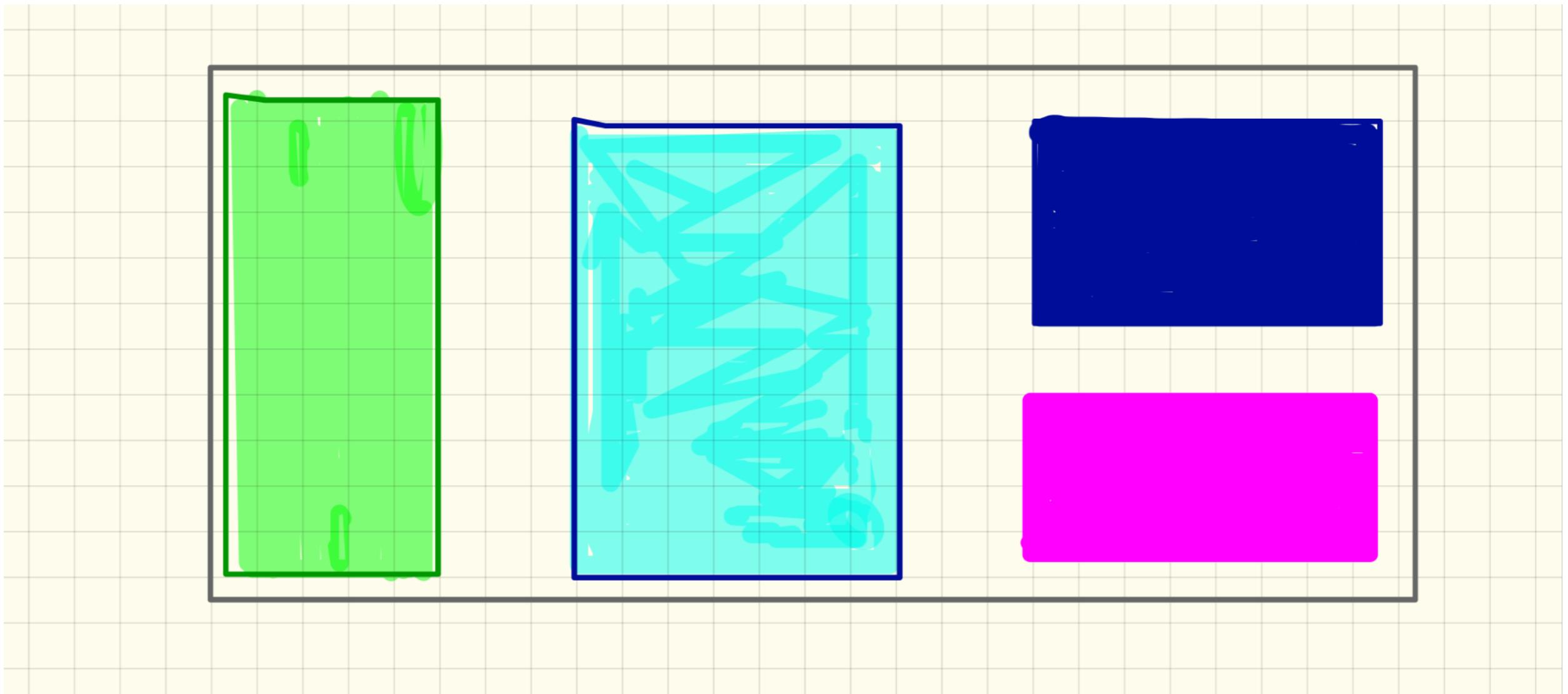
# CORE DOMAIN

---

- A System Hard to understand is hard to change
- Boil down the MODEL
- Skilled developers are drawn to new tech
- Get top talent into the CORE DOMAIN

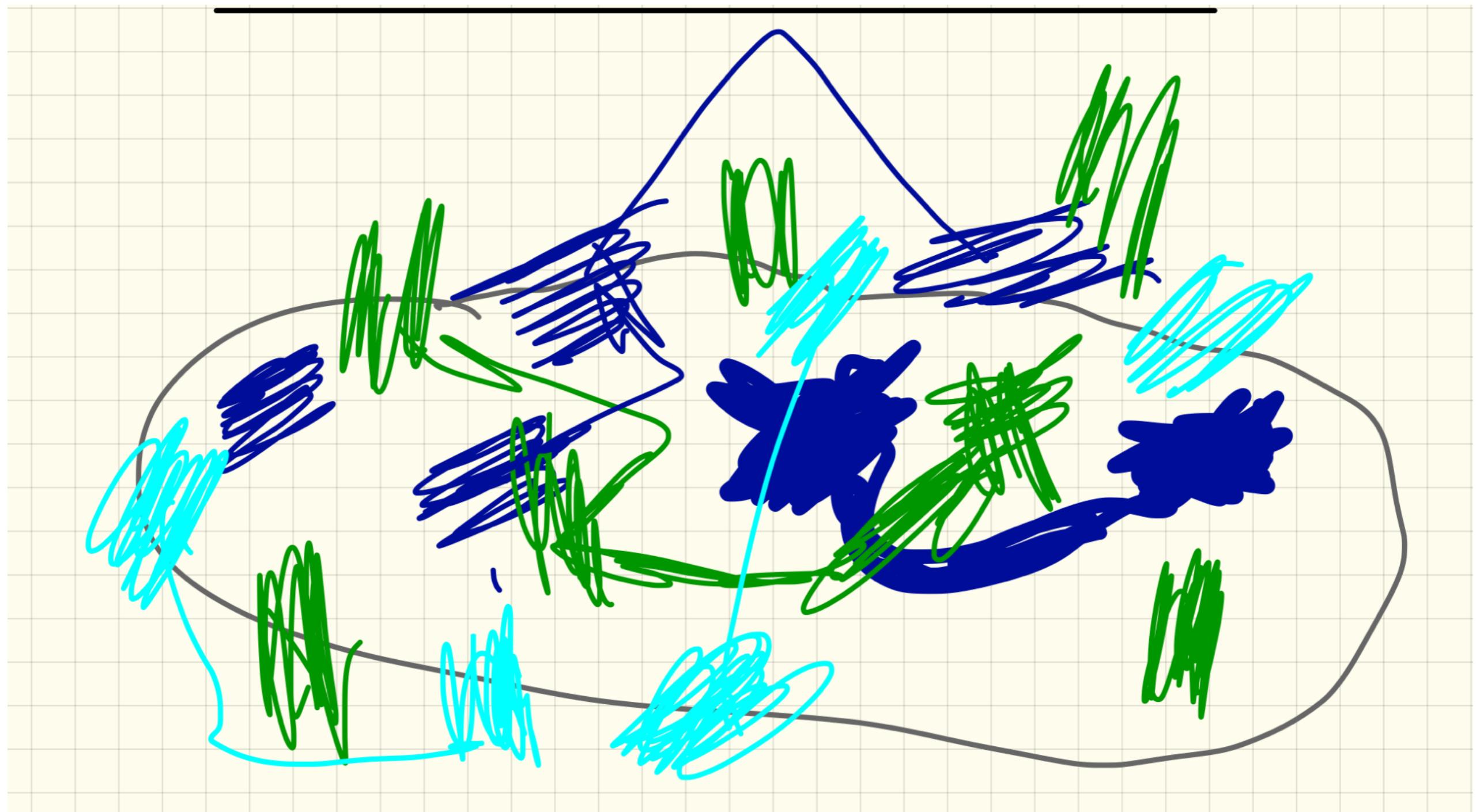
# Application, as we Imagine it could be

---

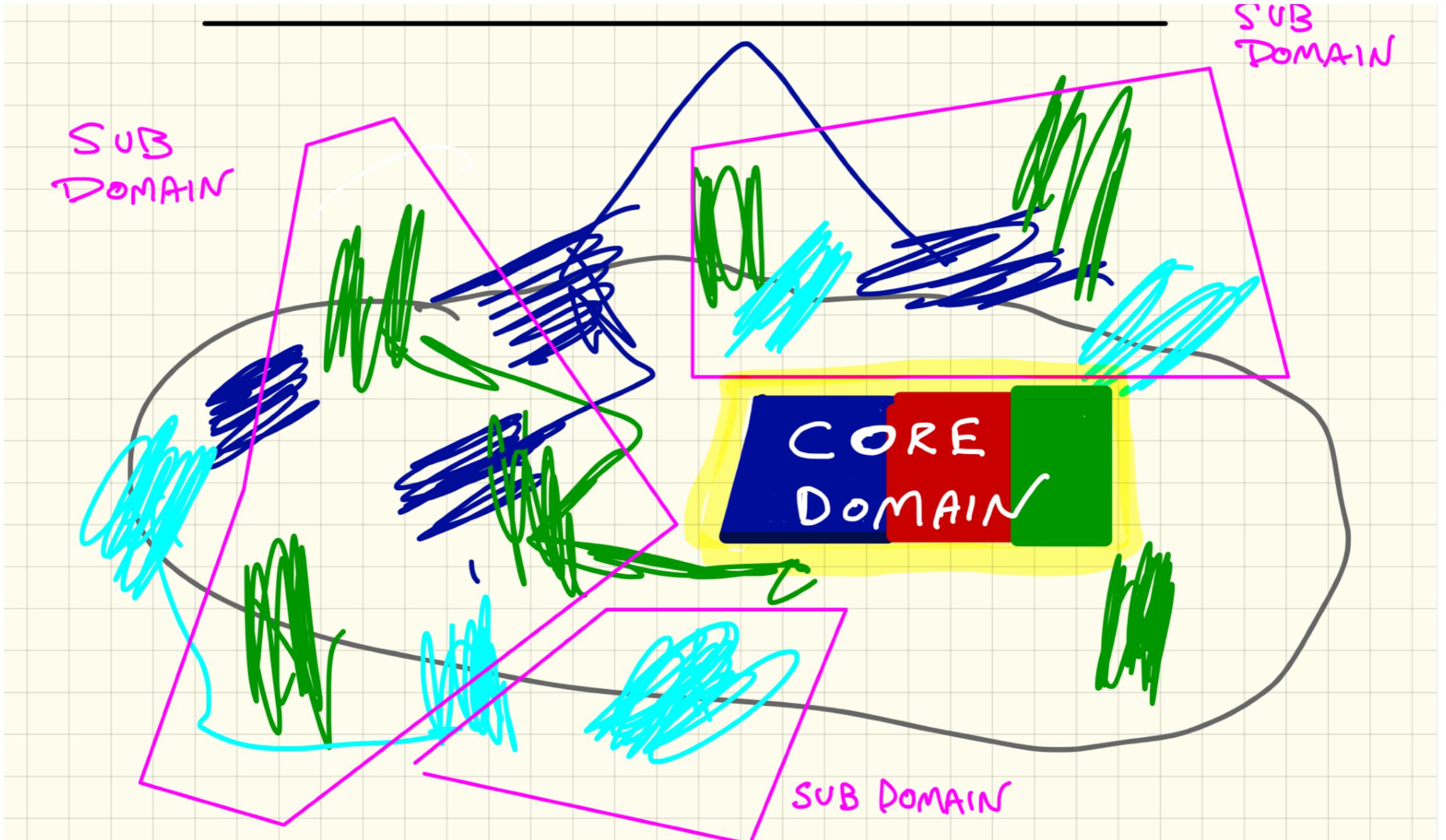


# Application, in Reality

---



# Apply strategic patterns to find the Core Domain



# DDD Topic Areas

---

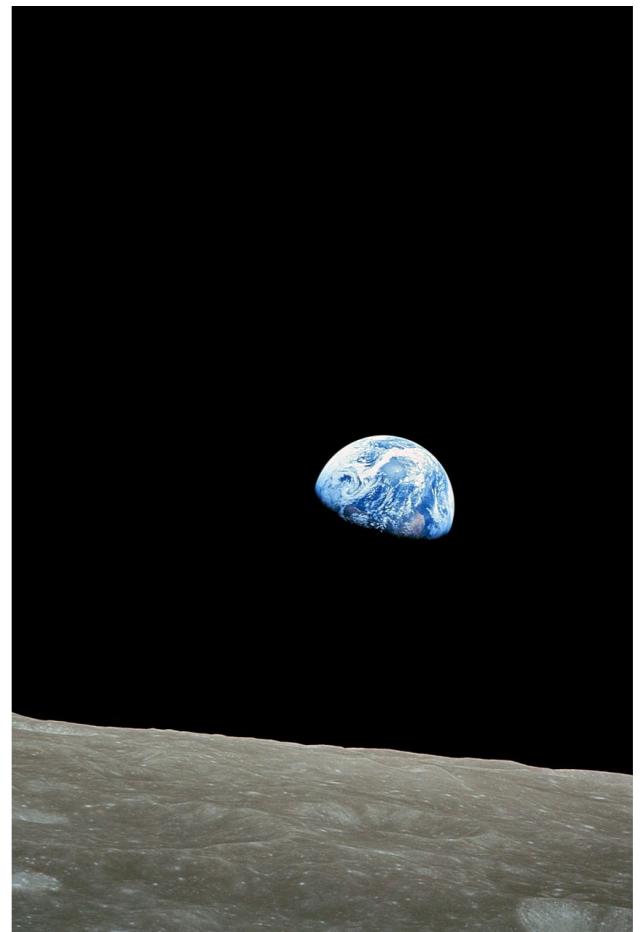
- ***Strategic Patterns***
- Tactical Patterns
- Communication Tips

# Strategic Patterns

---

Finding & Separating

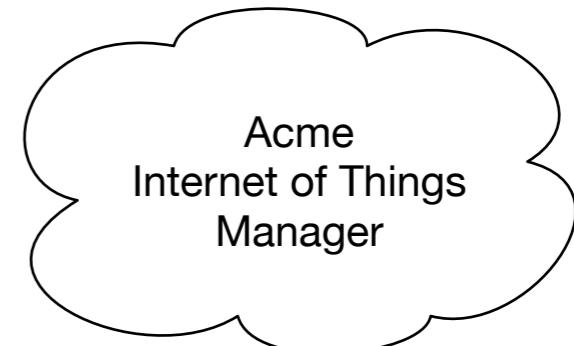
The Core Domain



# Bounded Context

---

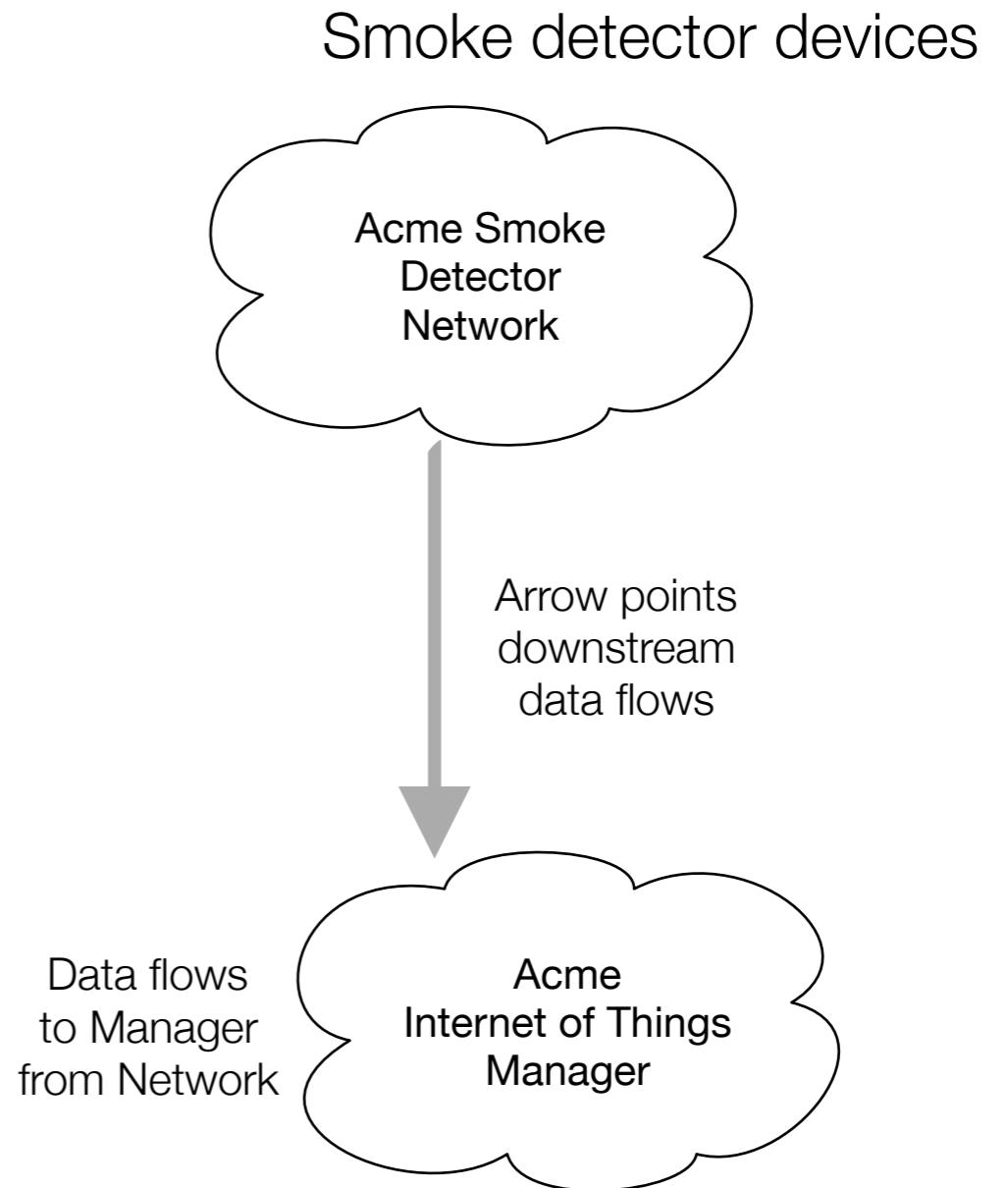
- Large projects have multiple Models
- Combining Models causes bugs
- Model only has meaning in a context
- **Therefore ...**
- Define the context of the Model
- Set boundaries in terms of
  - parts of the application
  - team organization
  - code bases, git-repositories



# ContextMap

---

- To develop a strategy, we need a large-scale view across our project and others we integrate with.
- **Therefore ...**
- Describe the points of contact between Models.
  - Explicit translations
  - Sharing
  - Isolation mechanisms
  - Levels of influence



# DDD Topic Areas

---

- Strategic Patterns
- *Tactical Patterns*
- Communication Tips

# Tactical Patterns

## - Building Blocks

---

- **Layered Architecture**
- **Aggregates**
- Value Objects
- Entities
- Services
- Domain Events

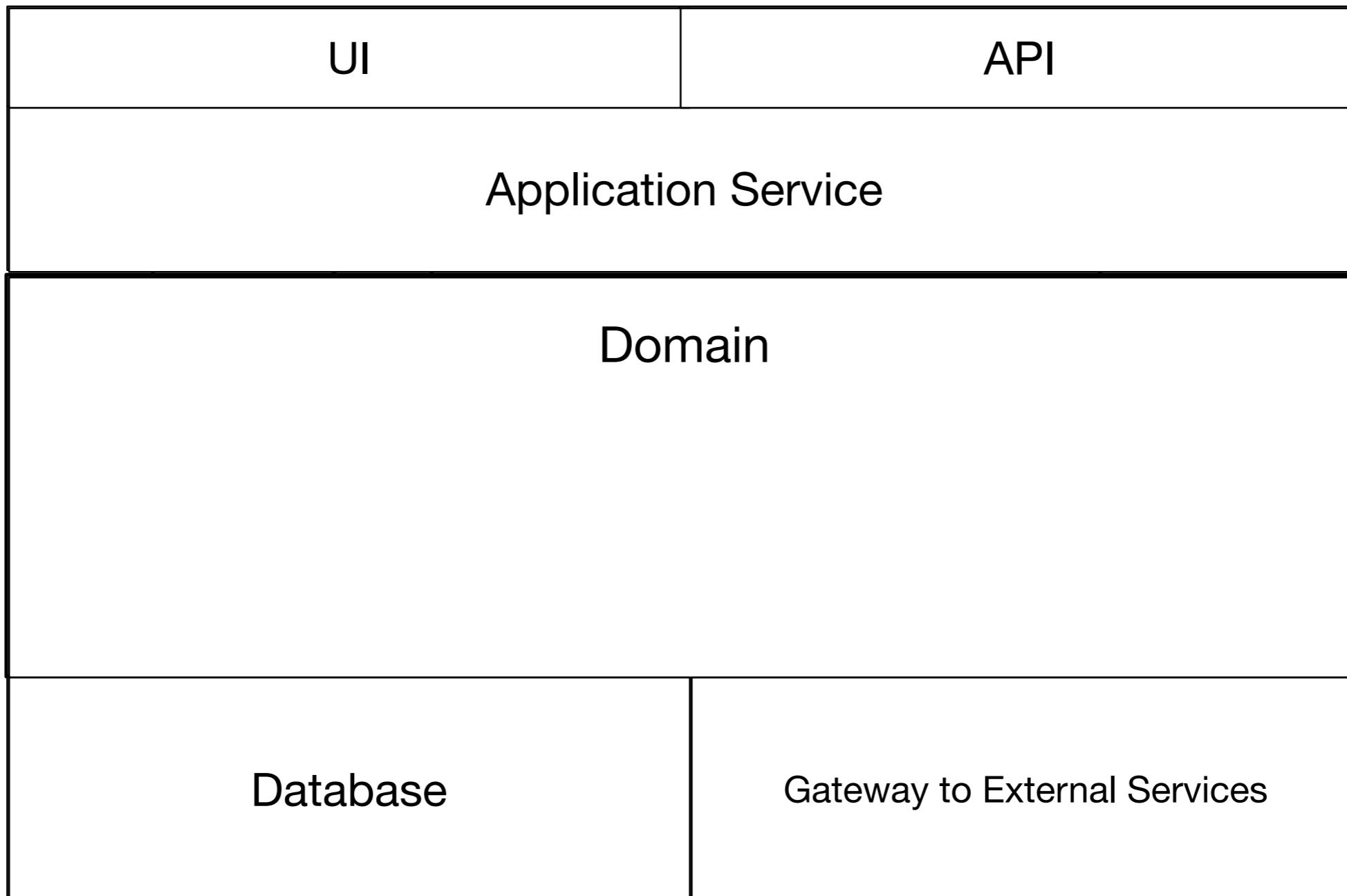


# Layered Architecture

| Name           | Description  | Example                |
|----------------|--|------------------------|
| UI             | User Interface or API  | Web form<br>Json API   |
| Application    | Thin. No Biz Rules. Stateless.<br>Use Cases. Coordinator.            | Funds Transfer Service |
| Domain         | Business Concepts, Biz Rules.<br>The heart of the business software. | DDD                    |
| Infrastructure | Support for the other layers   | Pub-Sub<br>DB          |

# Aggregates in Layers - 1

---



# A few other Building Blocks

---

## Value Object

- Value only

{ :usd, 1000.00 }

{ :watt\_hours, 900 }

## Entity

- Life Cycle

- Has Identity field

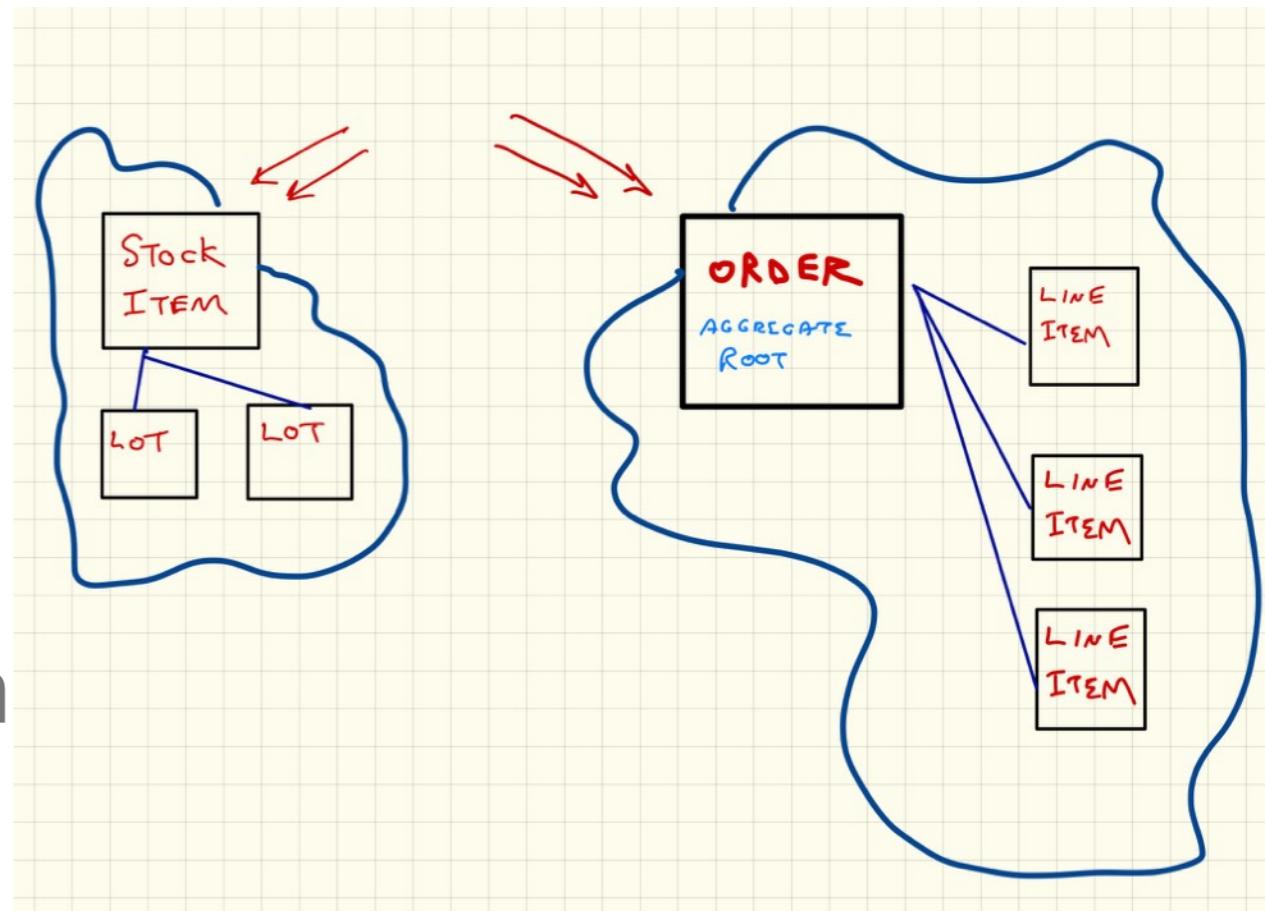
%Order{ id: 2341334 }

%User{ id: 3421424 }

# Aggregate

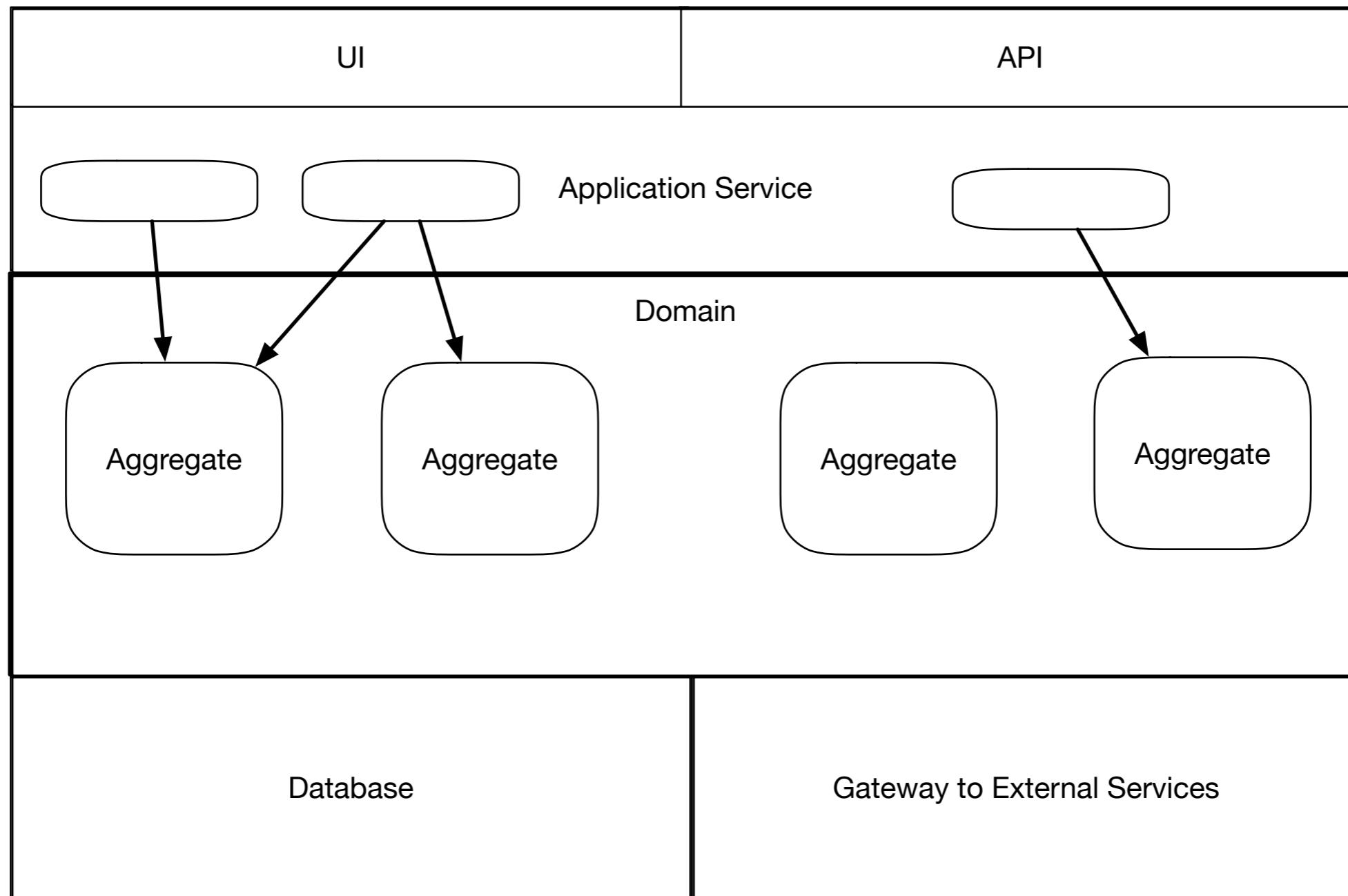
---

- Keep them Small
  - Protect Business Invariants inside Aggregate
  - Reference Other Aggregates by Identity only
  - Update other Aggregates with Eventual Consistency
- 



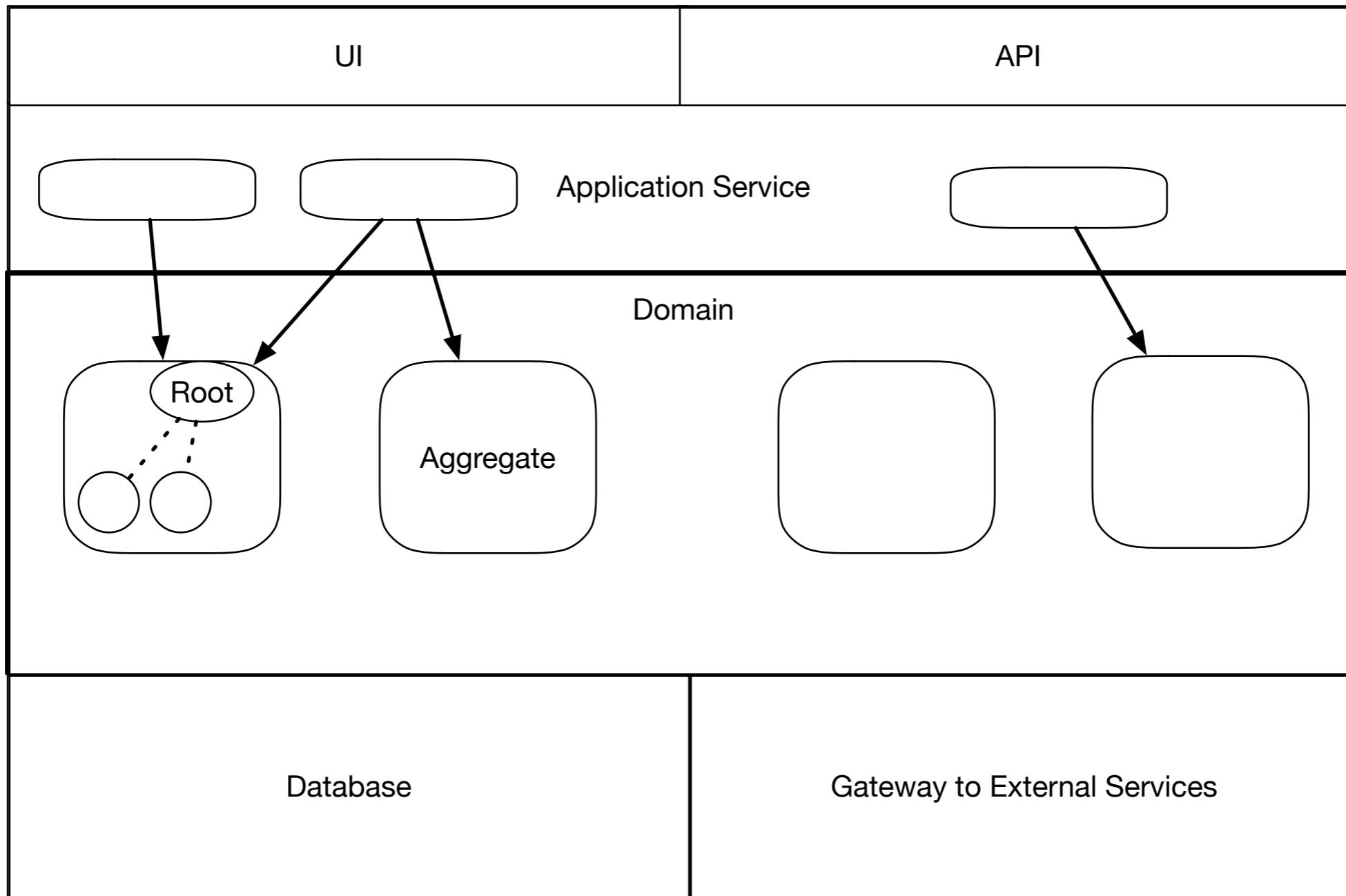
# Aggregates in Layers - 2

---



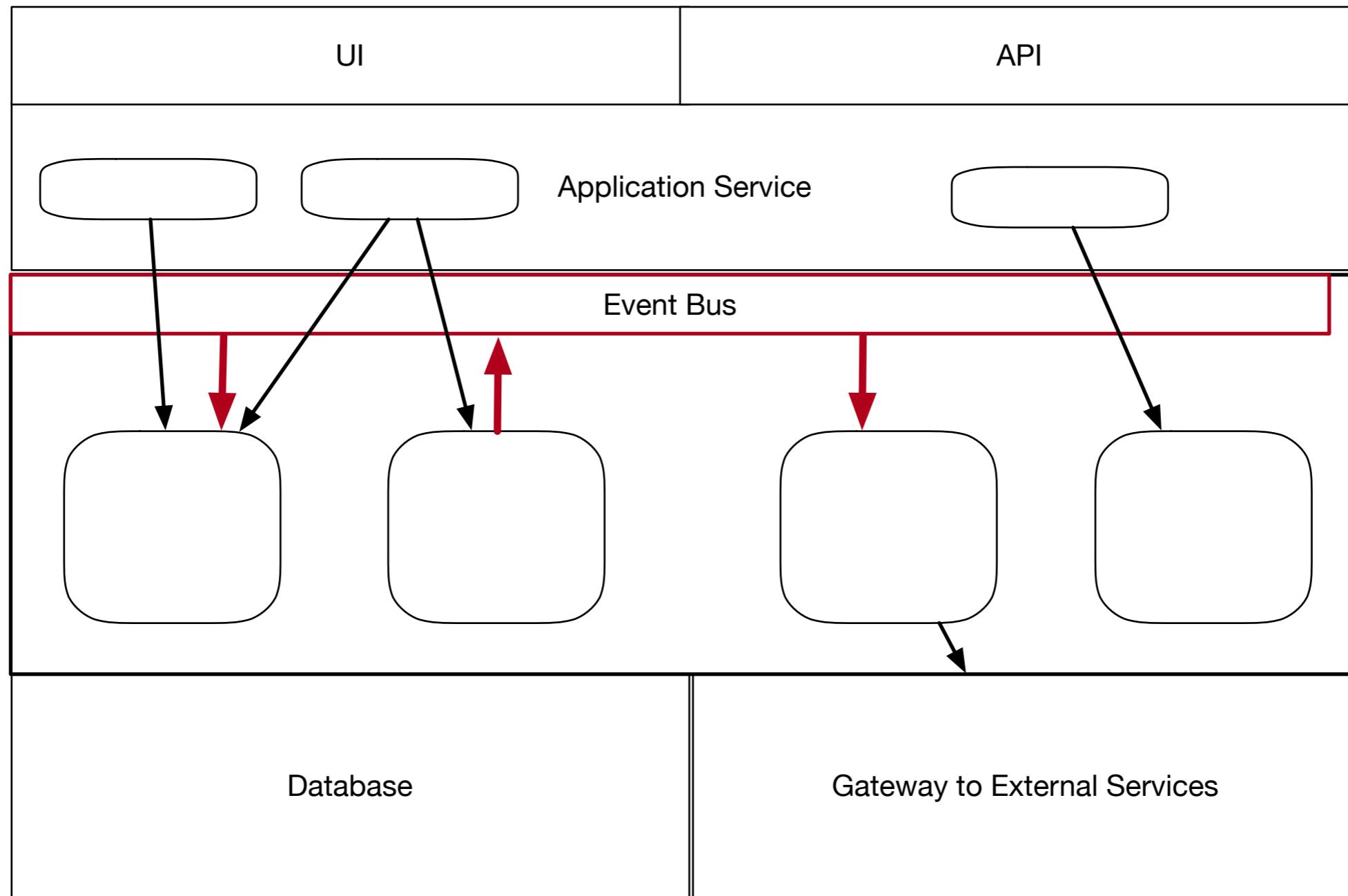
# Aggregates in Layers - 3

---



# Aggregates in Layers - 4

---



# DDD Topic Areas

---

- Strategic Patterns
- Tactical Patterns
- *Communication Tips*

# Communication Tips

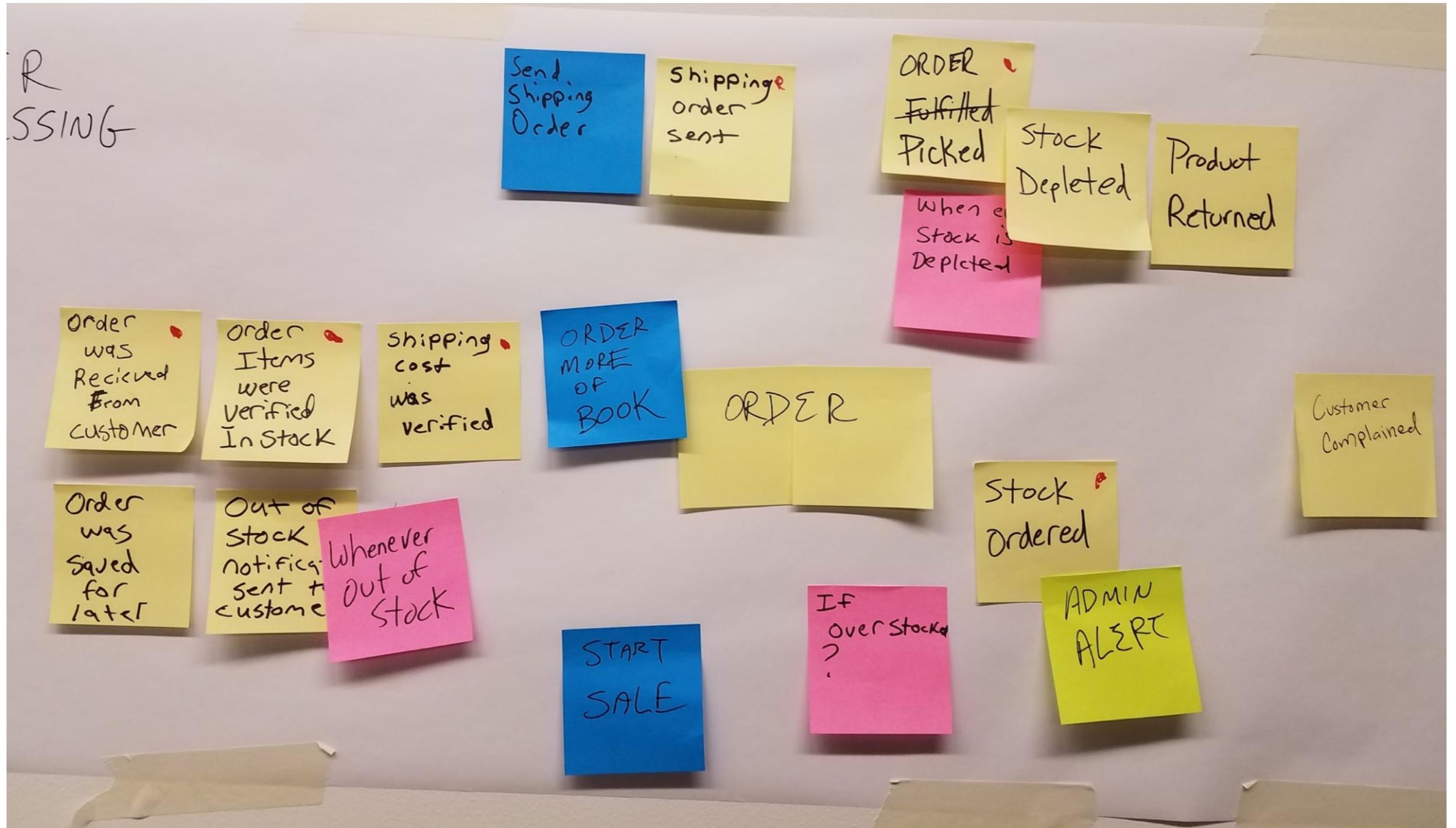
---

- Investigate, question Constraints
- Listening for missing Words and Concepts
- Ask “Why?”
- Event Storming
- Whirlpool

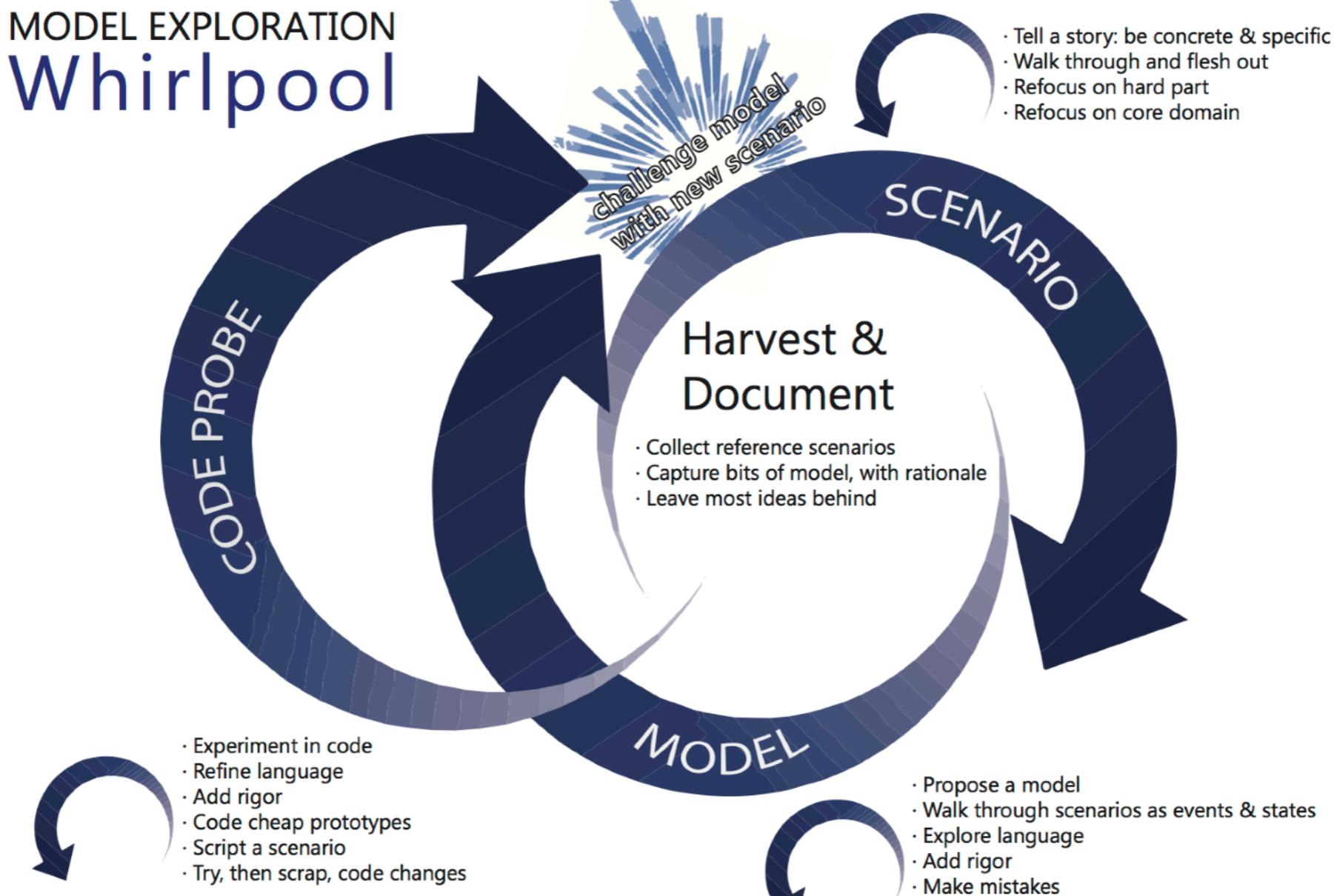
# Event Storming



# Event Storming



# MODEL Exploration



# When to use Full-DDD

---

- Strategic, Tactical, Communication
- Increasing complexity is slowing down progress
- Defects about fundamental Model are increasing
- Edge Cases are increasing

# When to use Partial-DDD - Strategic

---

- **As an Architect**
- Use Bounded Context to reduce confusion
- Context Mapping to see flow of the MODEL
- When: more then two main modules

# When to use Partial-DDD - Tactical

---

- **As a Developer:** Use Building Blocks
- Layered Architecture, Aggregate
- TDD
- When: Increasing conditional statements
- When: DB accessed everywhere, big test setup
- When: DB Locking issues

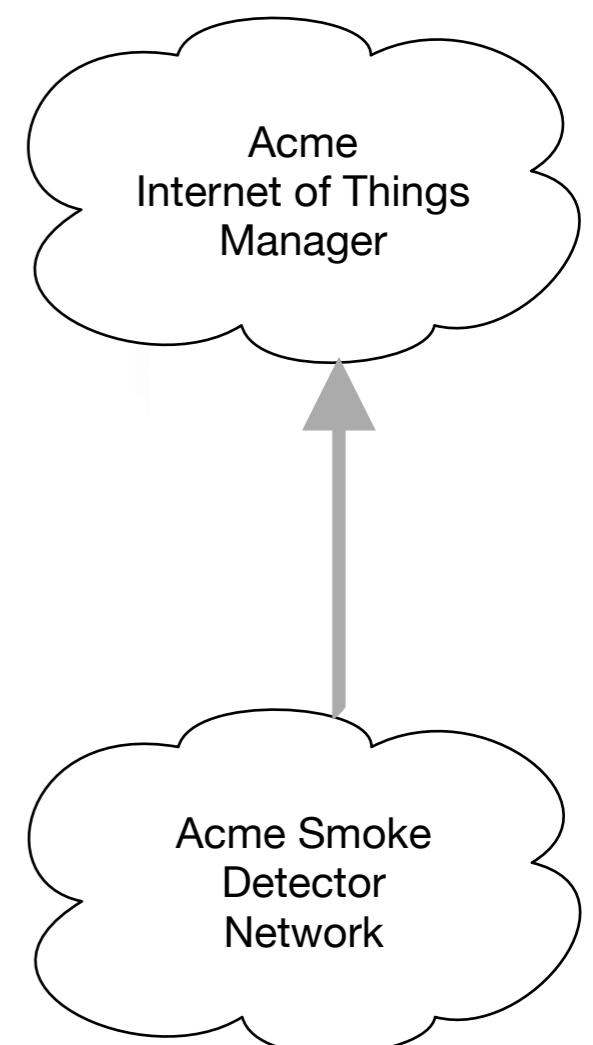
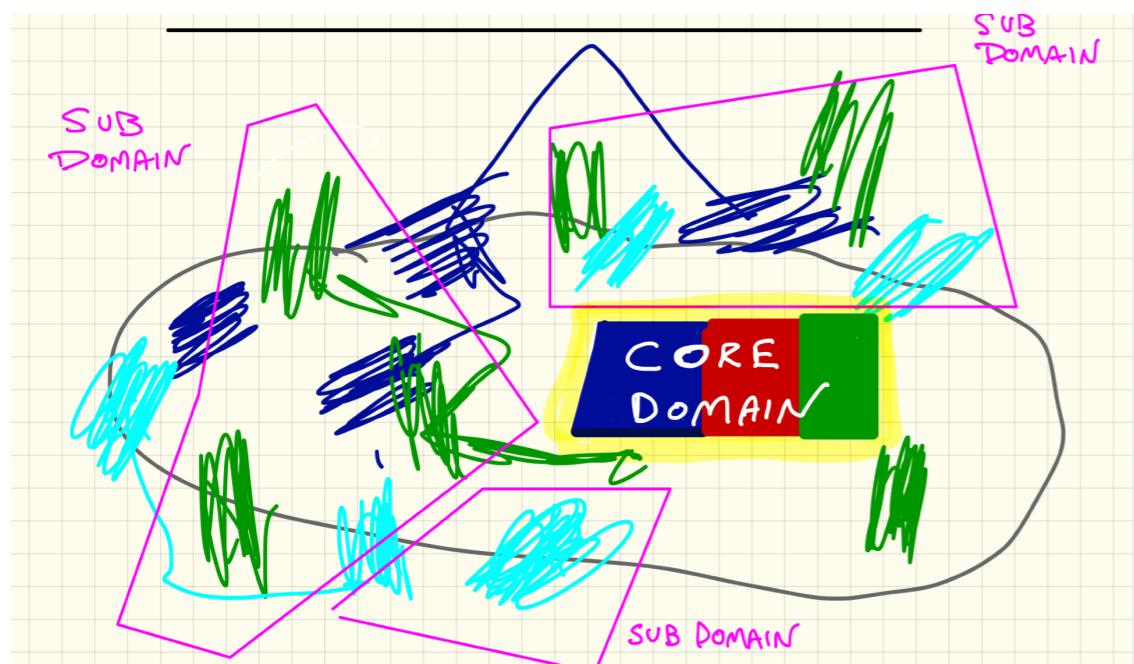
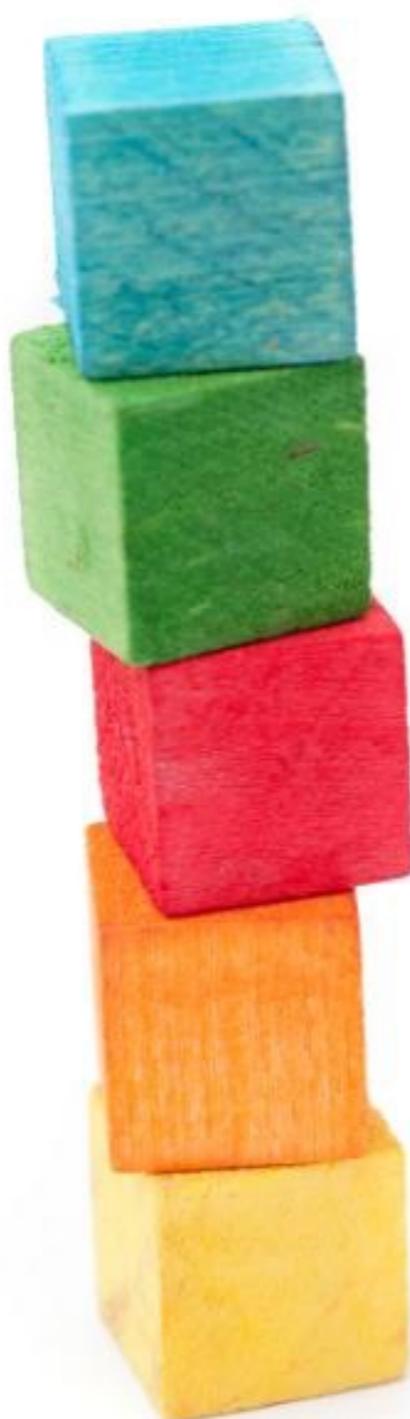
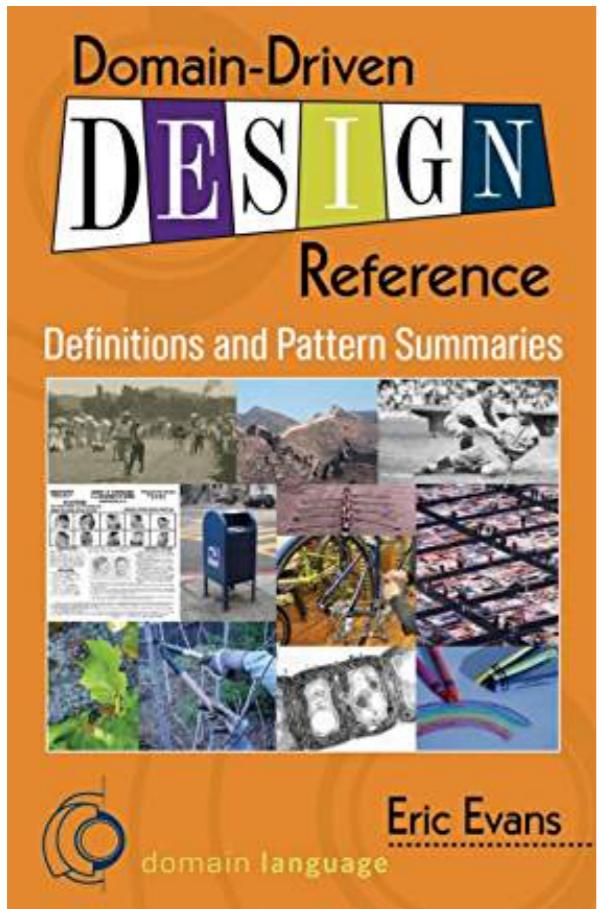
# When not to use DDD

---

- Simple CRUD / REST application needs
- Not using an iterative process
- Poor access to Domain Experts
- Build and forget apps

CRUD => Active Record => Domain Model => DDD

- Davin Tryon, via stack overflow



Just the beginning ...