Domain Driven Design

Building Blocks

DDD-Meetup Cincinnati
2019-07-23

https://github.com/mwindholtz/presentations/tree/master/DDD

Domain-driven design (DDD)

An approach to software development

- for complex needs
- by connecting the implementation
- to an evolving model.

The term was coined by Eric Evans

A Short Review or Overview …

Meeting Topic Areas

1. **Strategic Patterns**
    - Context Maps, Sub Domains, etc
2. **Tactical Patterns**
    - DomainEvents, Aggregates, etc
3. **Communication Tips**
    - Whirlpool, Knowledge Crunching, Event Storming
4. **Code Examples**
    - Build In Your Own Language: The Cargo Shipping Example

When to Apply Domain Design

- **For Simple systems**
  - No worries. It fits inside a person's head. REST
- **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**
  - Starting is ok. It initially still fits inside a person's head.
  - Then Documents help a while
  - But As It Grows …

** Documentation has an unknown expiration date.
And may be wrong to begin with.
Other restrictions may apply.

Typical “Agile” project progression

- 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, Feature story, Feature story *:-o*

Code Structure:
Big Ball Of Mud

---

http://www.laputan.org/mud/

Process Diagnosis:
Featureatitis

Software Craftsmanship — IS NOT ENOUGH —

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

Kent Beck @KentBeck May 28

Software development is a leaky rowboat. Behavior changes are rowing--making progress toward a goal, however dimly glimpsed. Structure changes are bailing--not progress in a measurable sense but absolutely necessary for progress.

DDD Europe @ddd_eu 5d

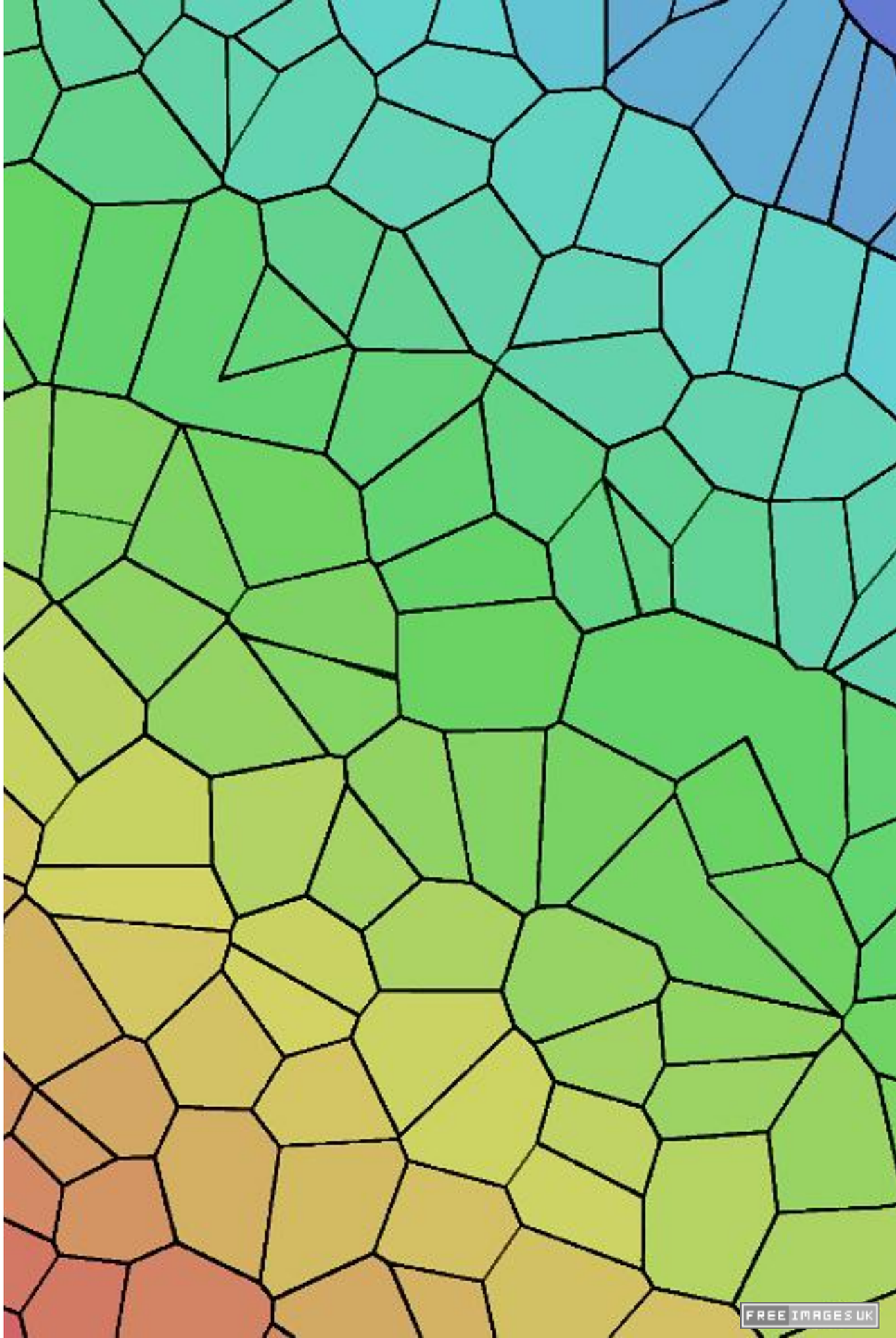
"No refactoring without remodelling. Clean Code by itself cannot save a rotten model." From "Technical debt isn't technical" by Einar Høst @einarwh at #DDDEU 2019 buff.ly/2WGYyss

17 35

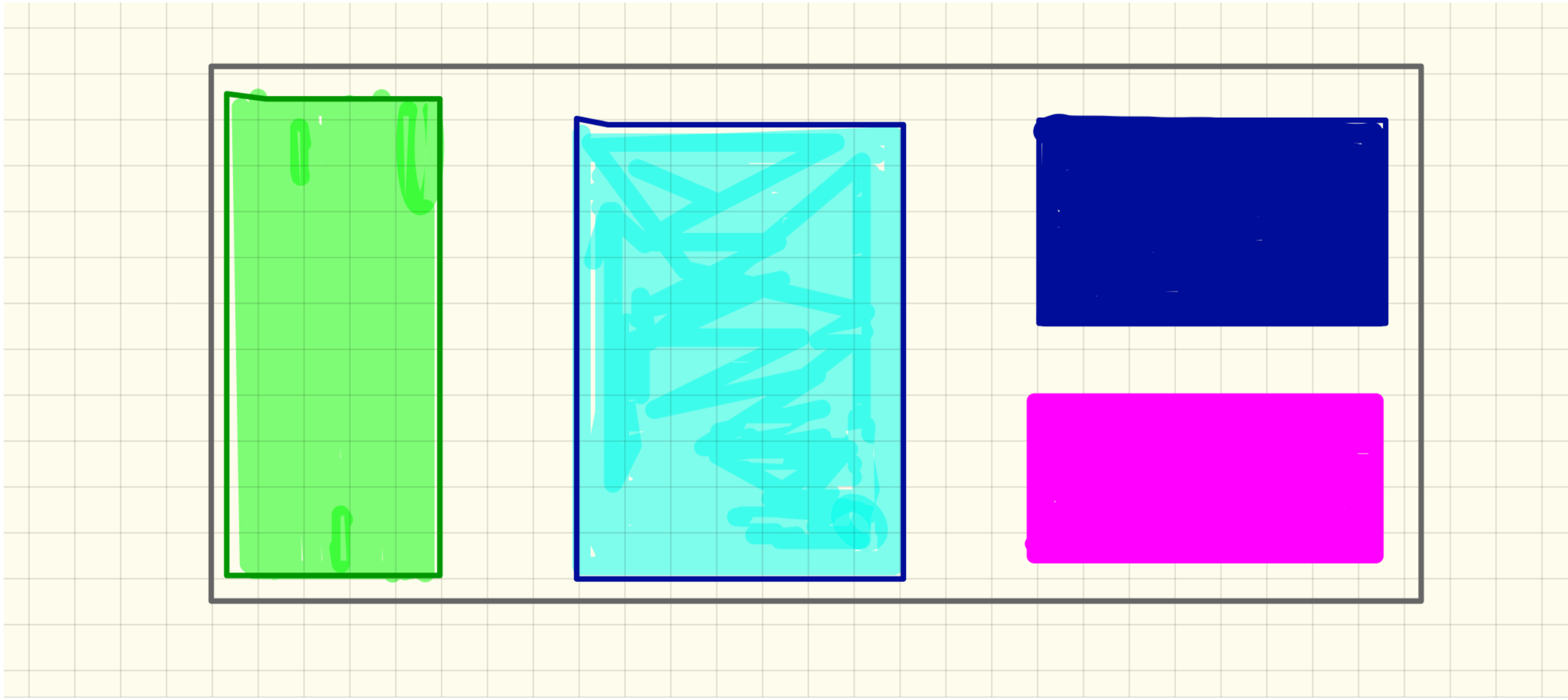
DDD Europe Videos 1999

“Doing” DDD

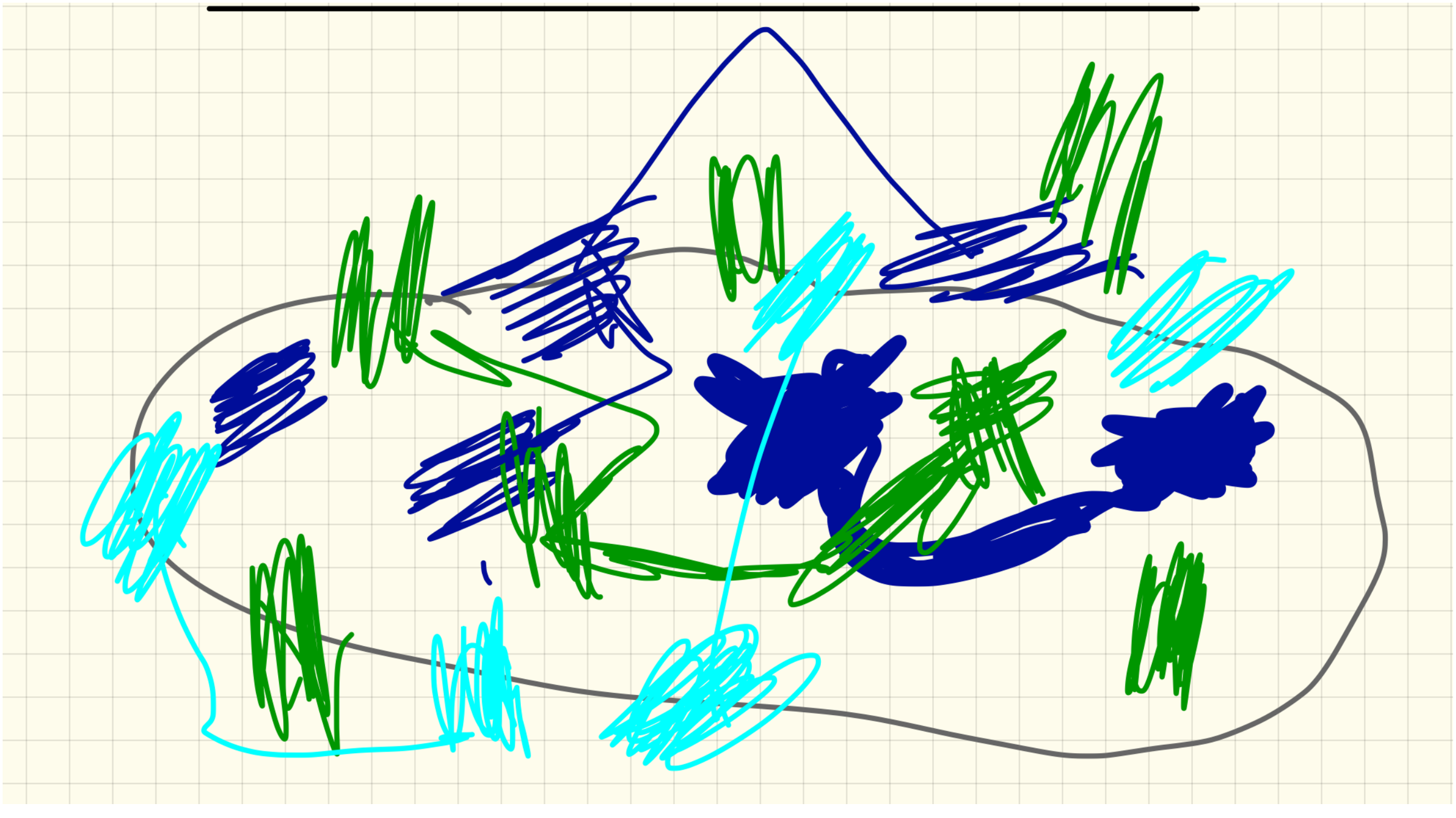
- Find ***Bounded Contexts***
- Build ***Context Map***
- Focus on ***Core Domain***
- Apply ***Building Blocks***
- Engage Domain Experts to build ***Ubiquitous Language***
- Repeat … (in parallel ?)



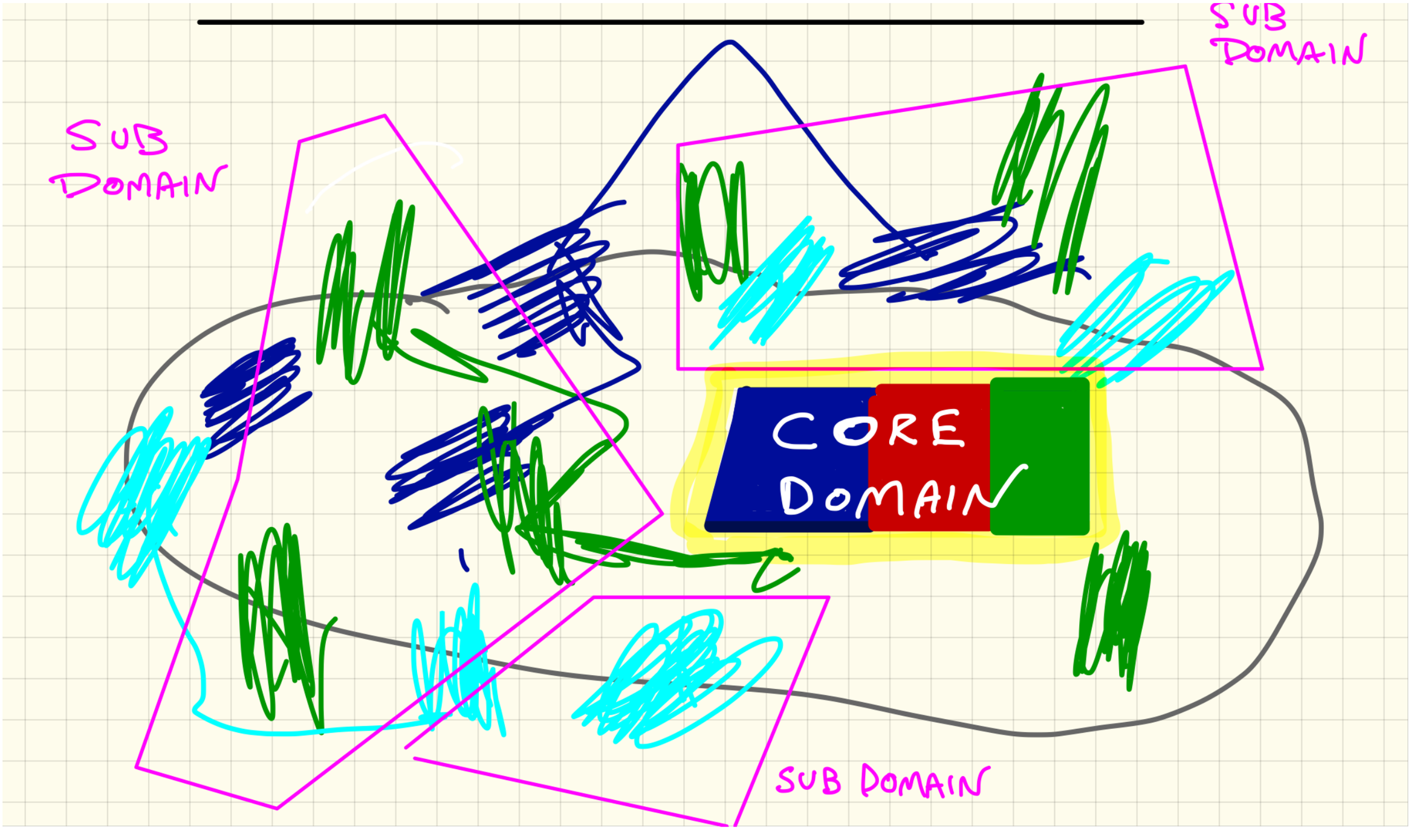
Application, in as we Imagine it



Application, as it Really is



Application with DDD added



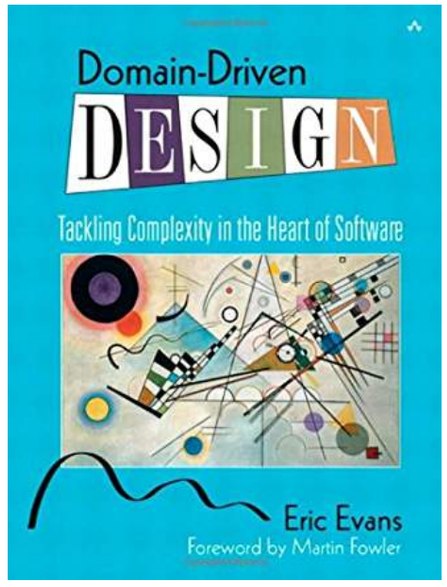
Domain

- **A sphere of knowledge, influence, or activity.**

- The subject area to which the user applies a program is the domain of the software.

Why Now ?

- **Domain Driven Design, “Big Blue Book”, 2003**
  - Tactical Patterns get most of the attention



- **Micro-Services, 2012**
  - Bounded Context and Context Mapping for organization
  - Content of each Micro-Service
  - Relationships among Micro-Services

Why is DDD Difficult to Explain?

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

Building Blocks

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

Layered Architecture

Name	Description	Example
UI	Shows information to the user. API to another system.	Web form Json API
Application	Thin. No Biz Rules. Stateless except for workflow progress. Use Cases. Coordinator.	Funds Transfer Service
Domain	Concepts of the Business, Biz Rules. The heart of the business software.	*more later
Infrastructure	Support for the other layers. Message sending, persistence	Pub-Sub DB

Building Blocks (part 1)

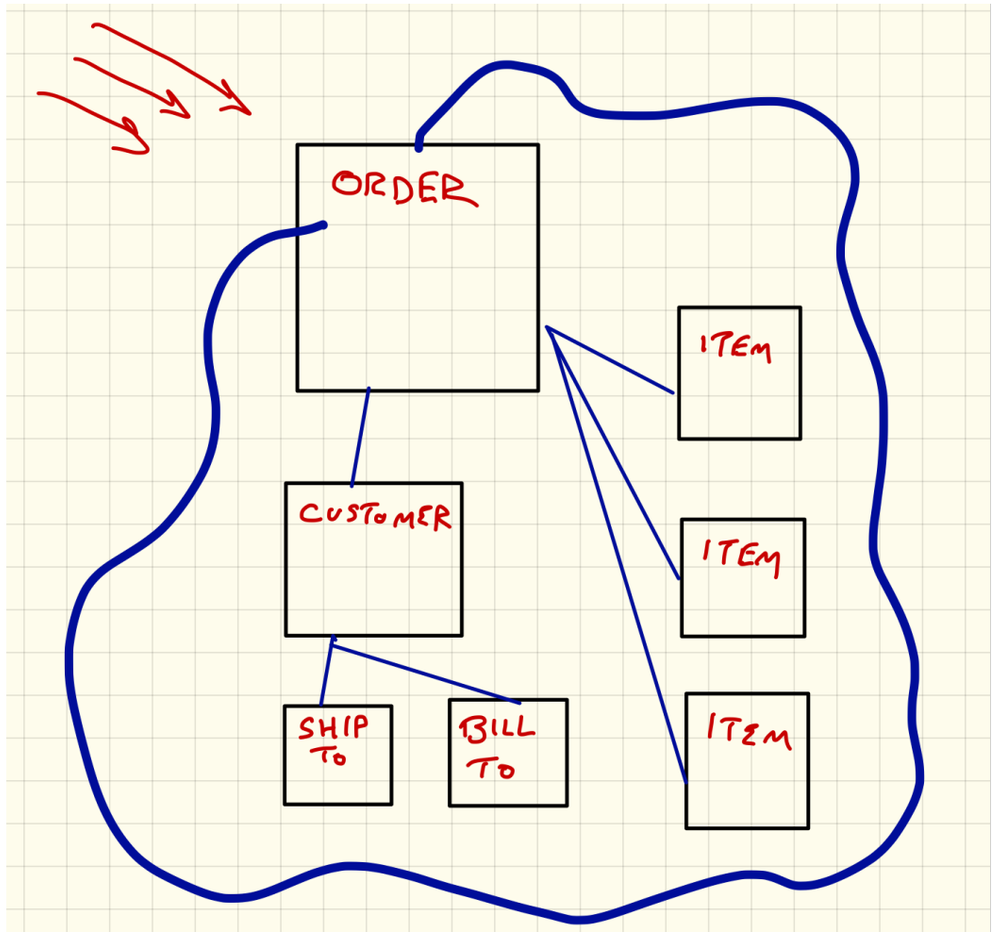
Name	Description	Example
Value Objects	Defined by the attributes, immutable	{2001,07,04} {:cents_us, 299} { :grams, 300}
Entities	Thread of continuity and identity	Customer IoT device Policy
Factories	Atomic creation, enforcing invariants	
Repositories	Saving and Reconstituting aggregates	

Building Blocks (part 2)

Name	Description	Example
Aggregates	Cluster of assoc objs treated as a unit. The Root is an Entity. Hides the entities inside	Policy IoT Site
Services	Stateless, UseCase	
Domain Events	Verb in past tense that the Domain Expert cares about	- Claim Rejected - Purchase Completed

Aggregate

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



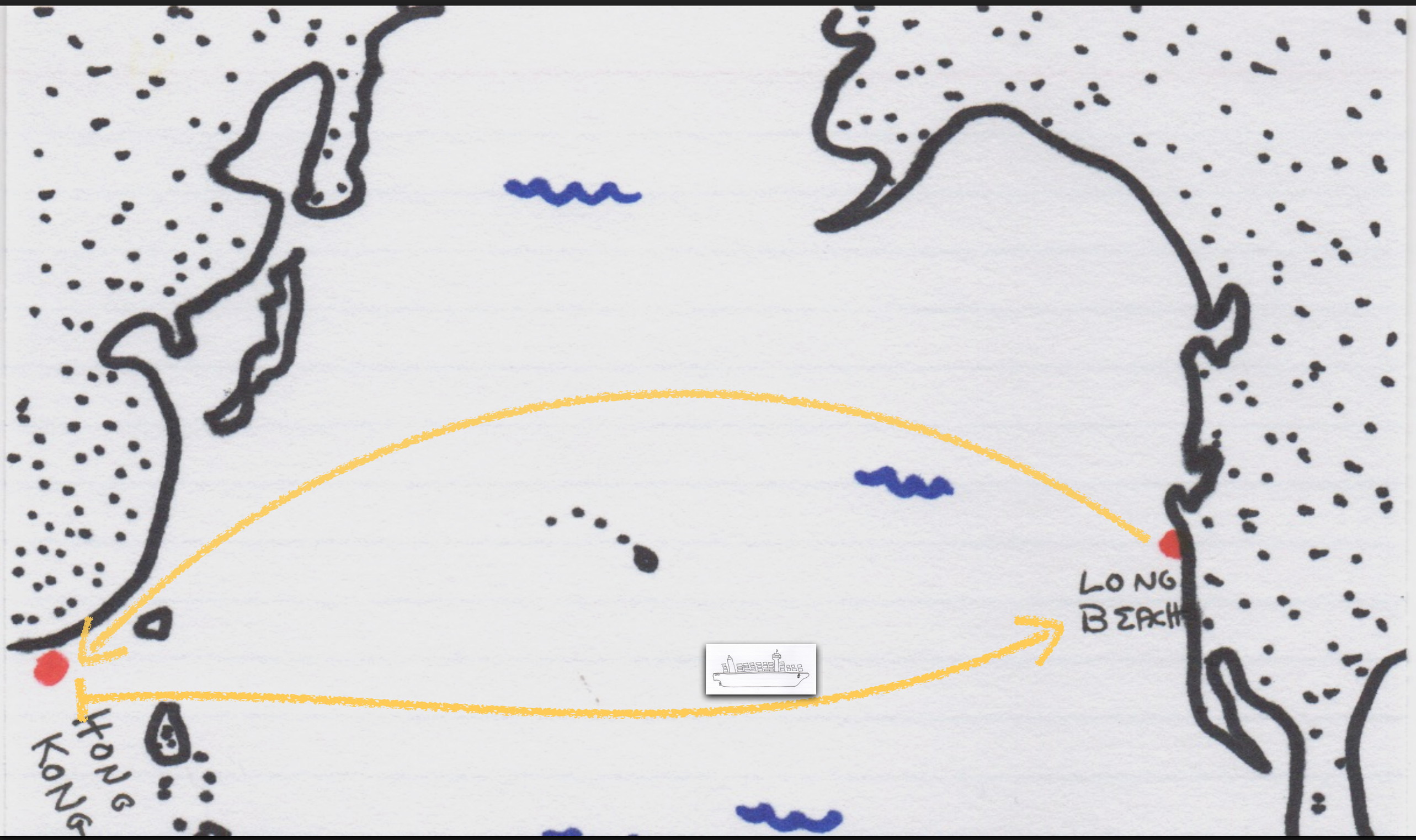
Services

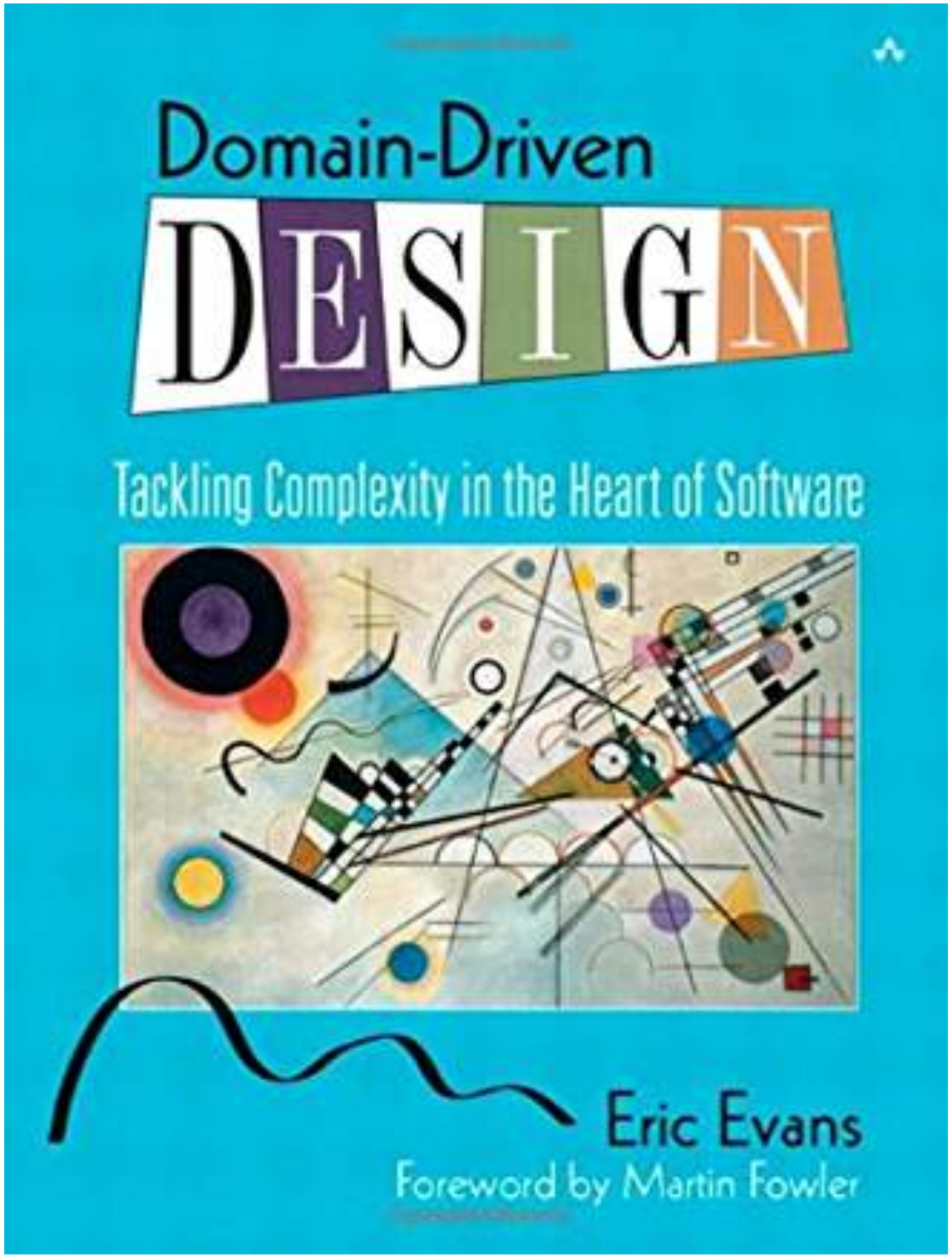
- An Operation that relates to a domain concept that is not a natural part of an ENTITY or a VALUE OBJECT
- The interface is defined in terms of other elements of the domain model
- The Operation is stateless
- Services can be in Domain Layer or with a different focus also in the Application Layer, or Infrastructure Layer

Domain Events

- Verb in past tense that the Domain Expert cares about
- Immutable
- Timestamped
- Primary or Derived from other Events
- Useful for distributed processes and eventual consistency
- Useful to find out ***why*** and ***when*** and entity changed.

CARGO SHIPPING




Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

The End