Domain Driven Design

Big Picture

DDD-Meetup Cincinnati
2019-08-20

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 <u>Simple</u> systems**
  - No worries. It fits inside a person’s head.
- **For <u>Medium</u> systems**
  - No worries. Hire smart people so that ..
  - It fits inside a person’s head. Oh and write loads of *Documentation! ***
- **For <u>Complex</u> 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 :-o

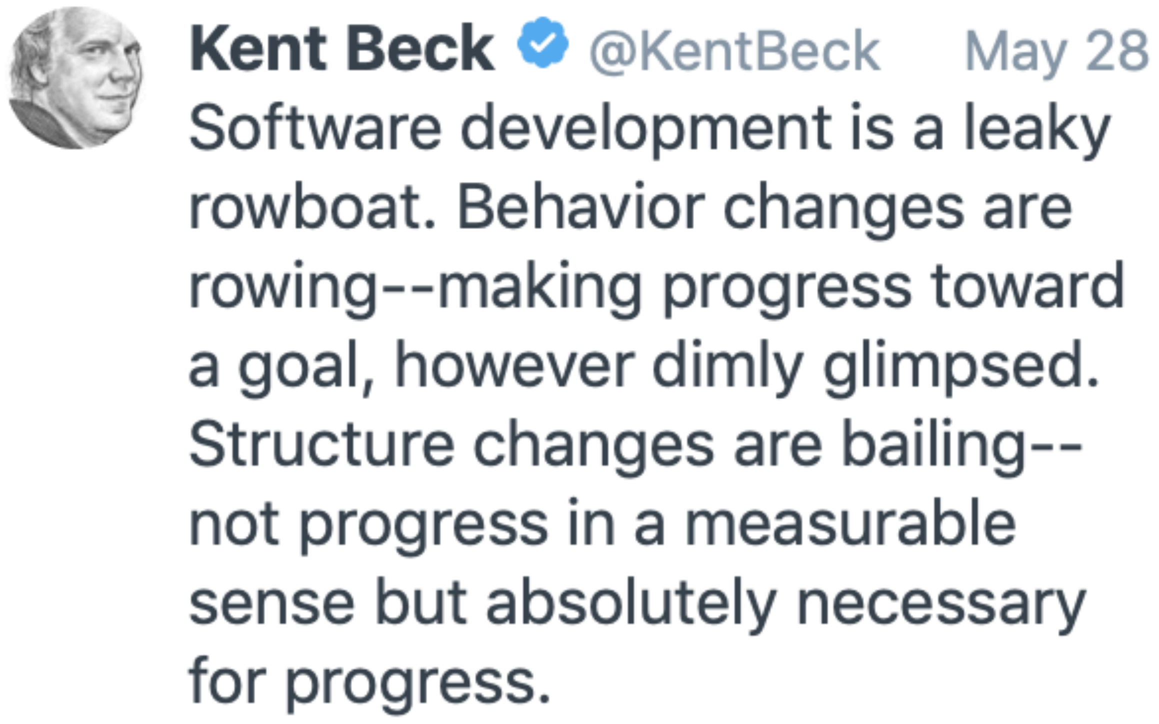
Code Structure:
Big Ball Of Mud

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

Process Diagnosis:
Featureatitis

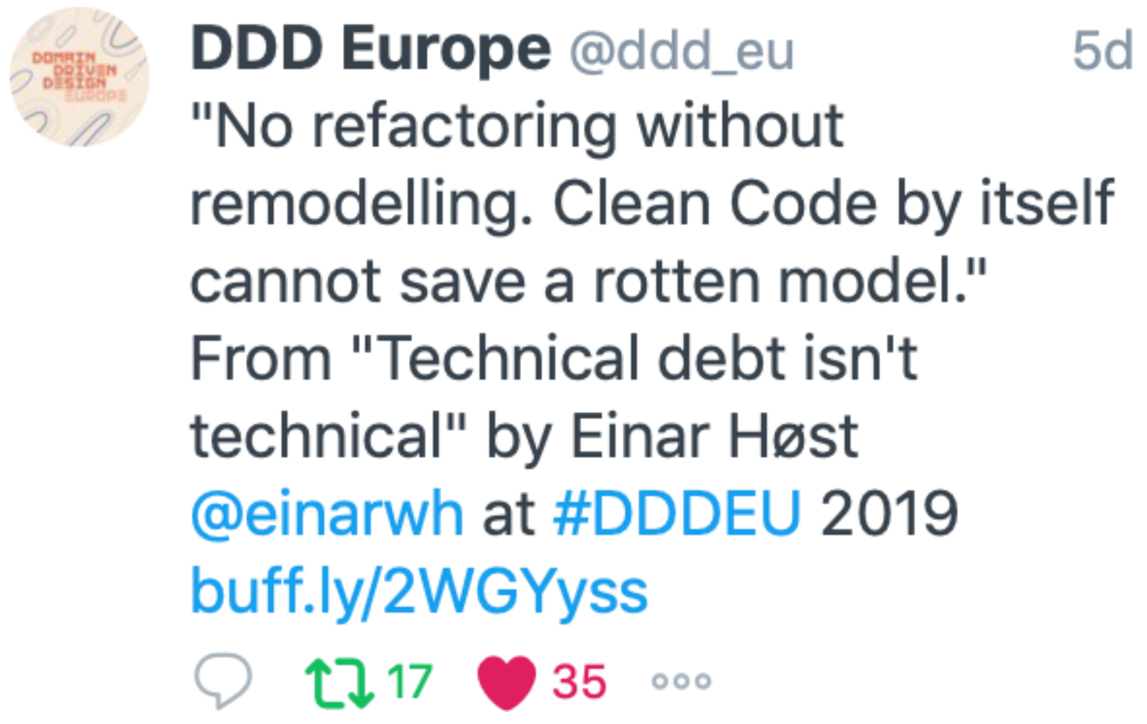
Software Craftsmanship — IS NOT ENOUGH —

- 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

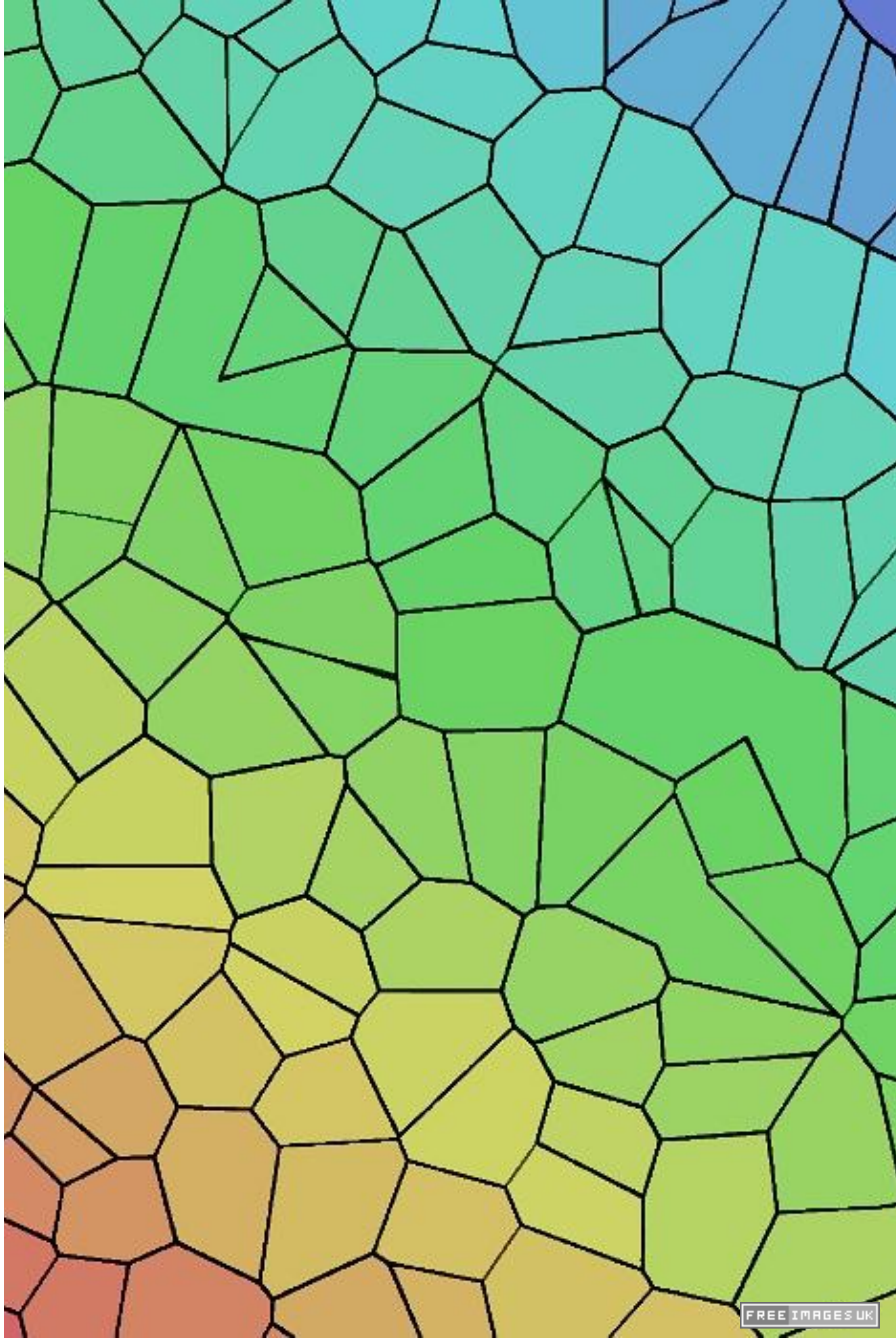
"No refactoring without remodelling. Clean Code by itself cannot save a rotten model." From "Technical debt isn't technical" by Einar Høst @einarwh at #DDDEU 2019 buff.ly/2WGYyss

17 35

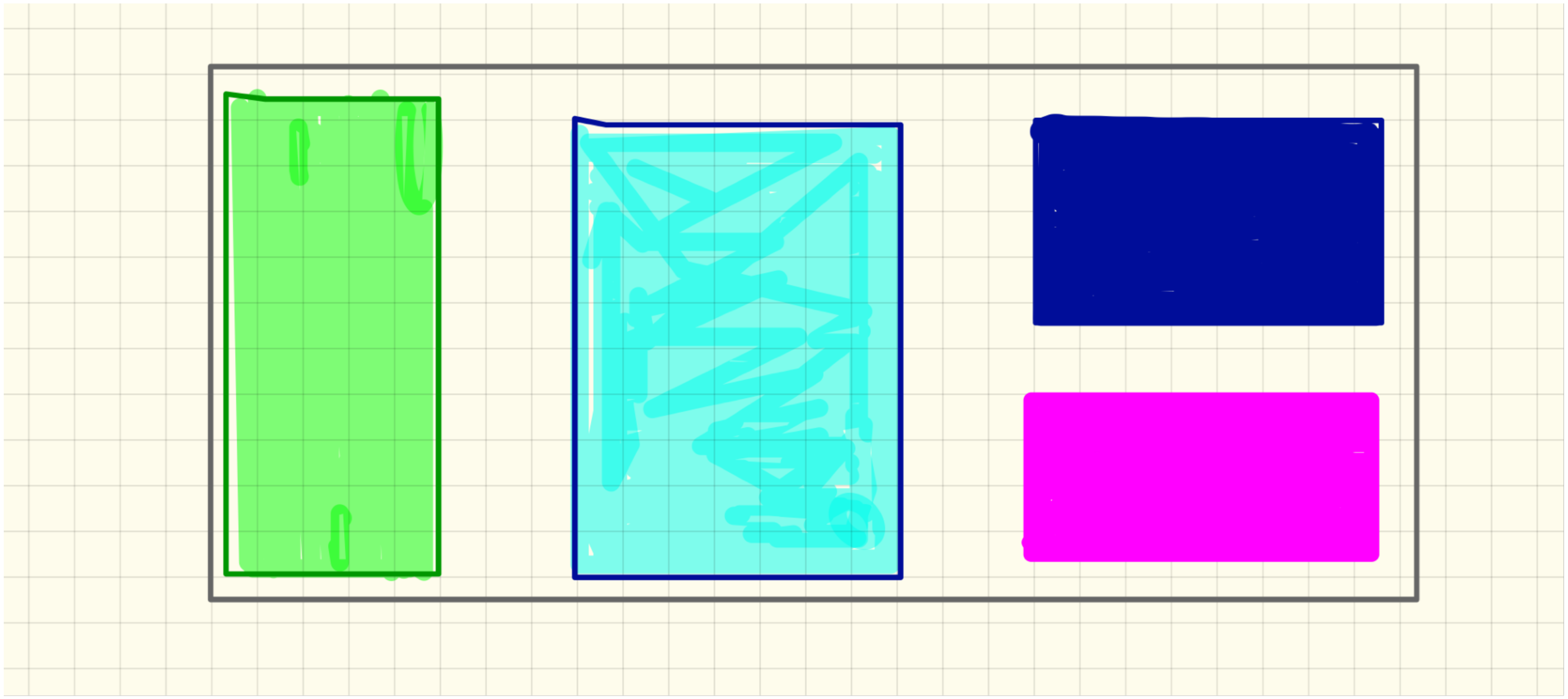
DDD Europe Videos 1999

“Doing” DDD

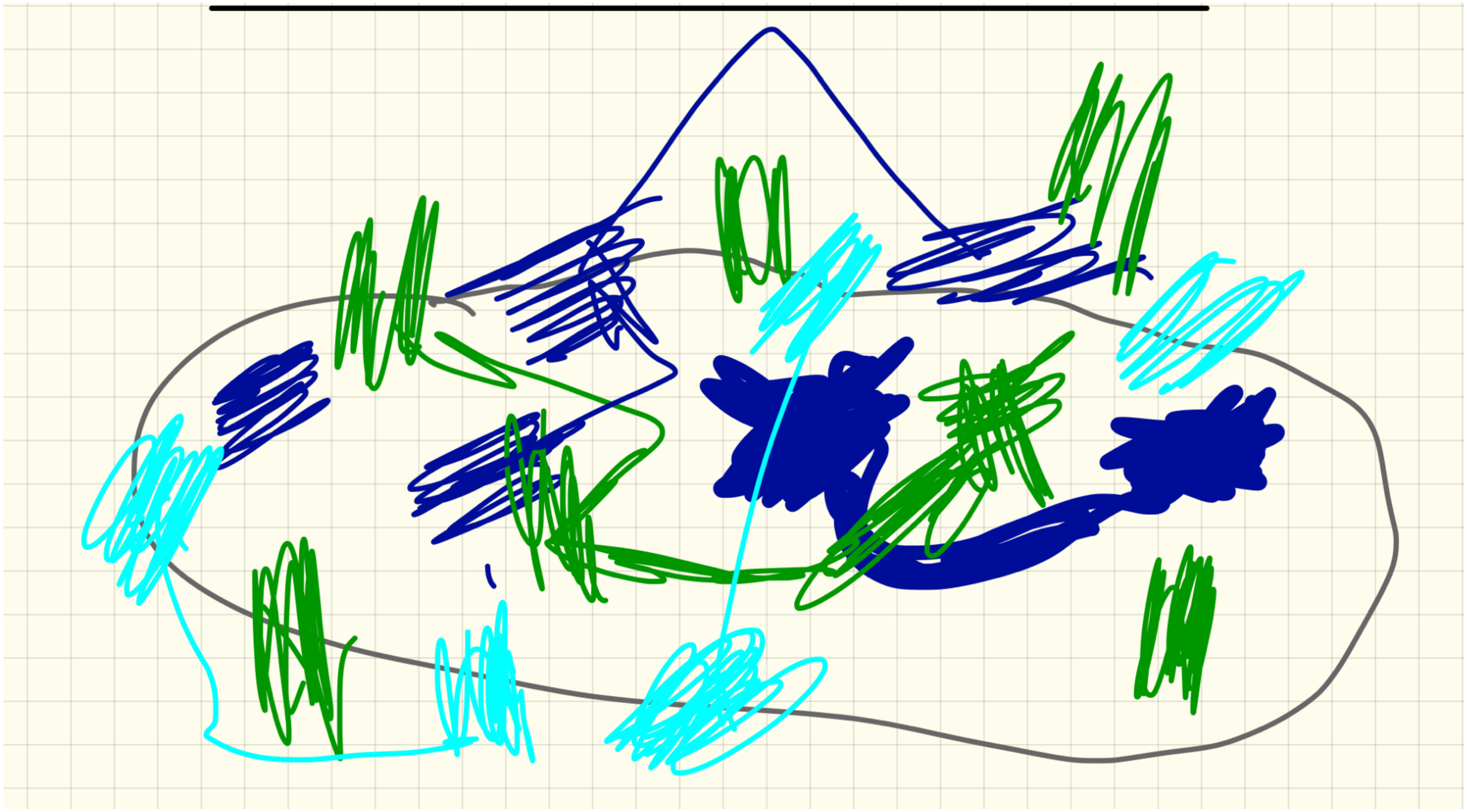
- Find ***Bounded Contexts***
- Build ***Context Map***
- Focus on ***Core Domain***
- Apply ***Building Blocks***
- Engage Domain Experts to build ***Ubiquitous Language***
- Repeat … and Revisit :||



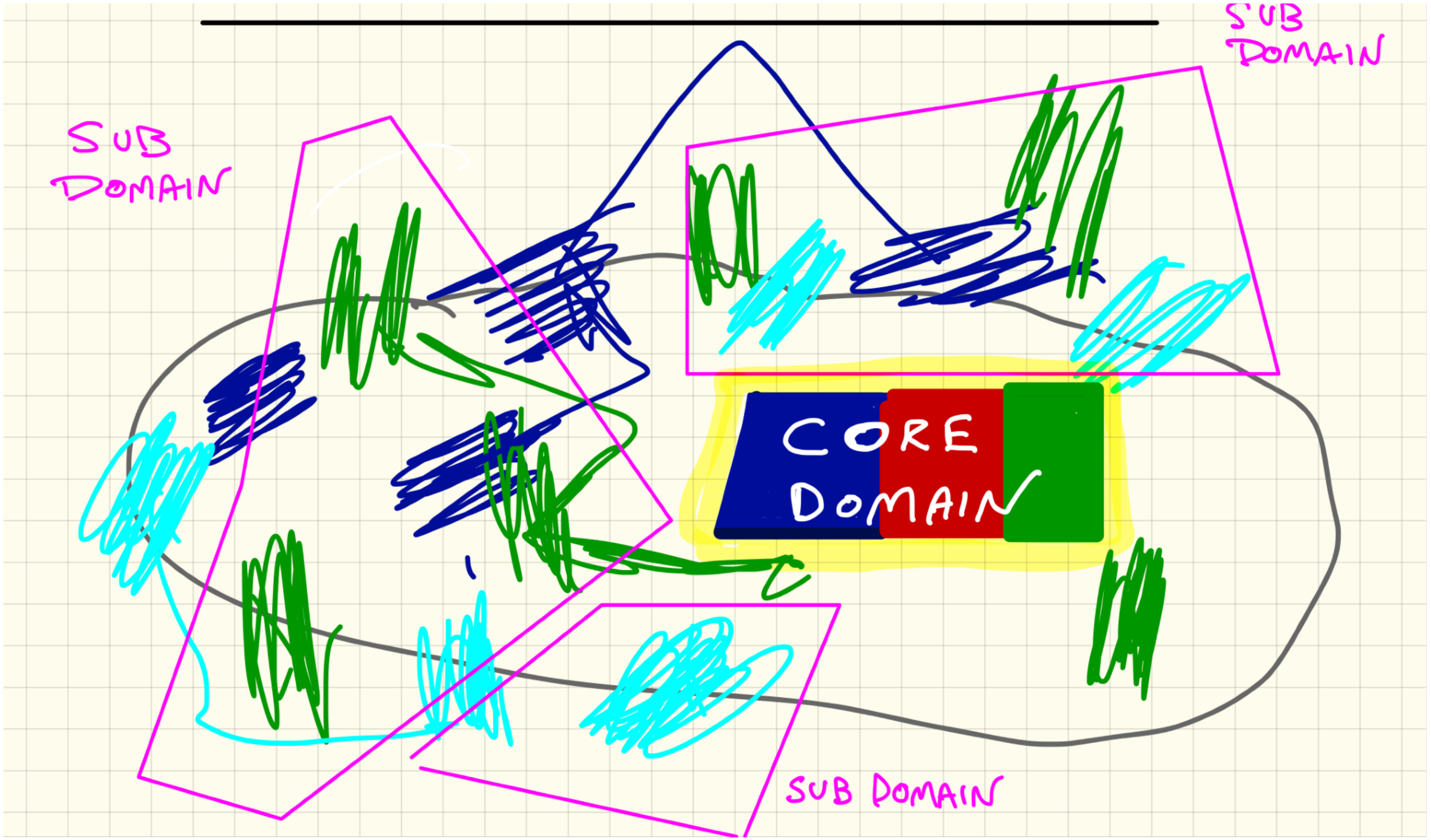
Application, as we Imagine it



Application, in Reality



Application with DDD added

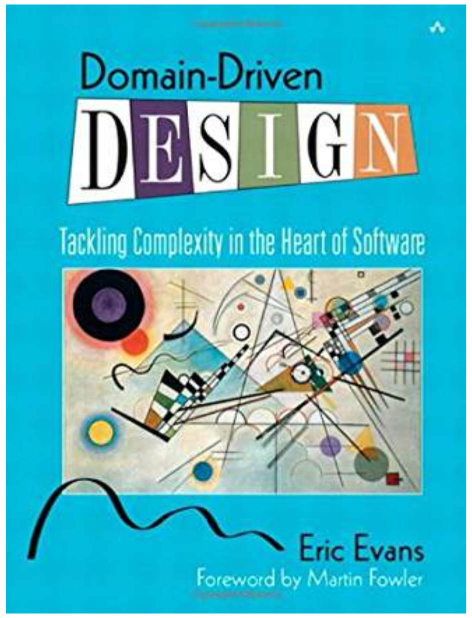


Domain

- **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


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

- **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

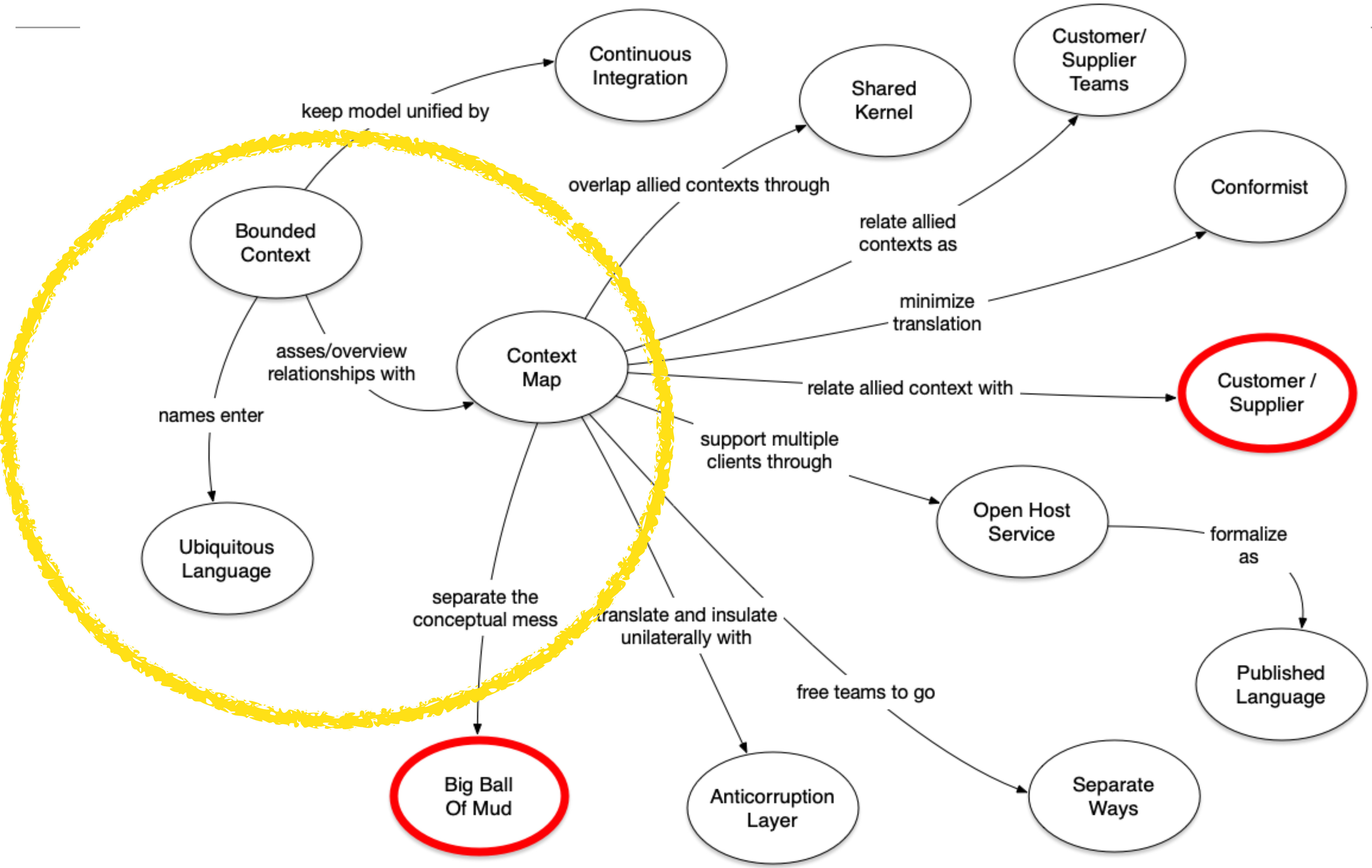
Building Blocks

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

Strategic Patterns

Large projects involving multiple teams.

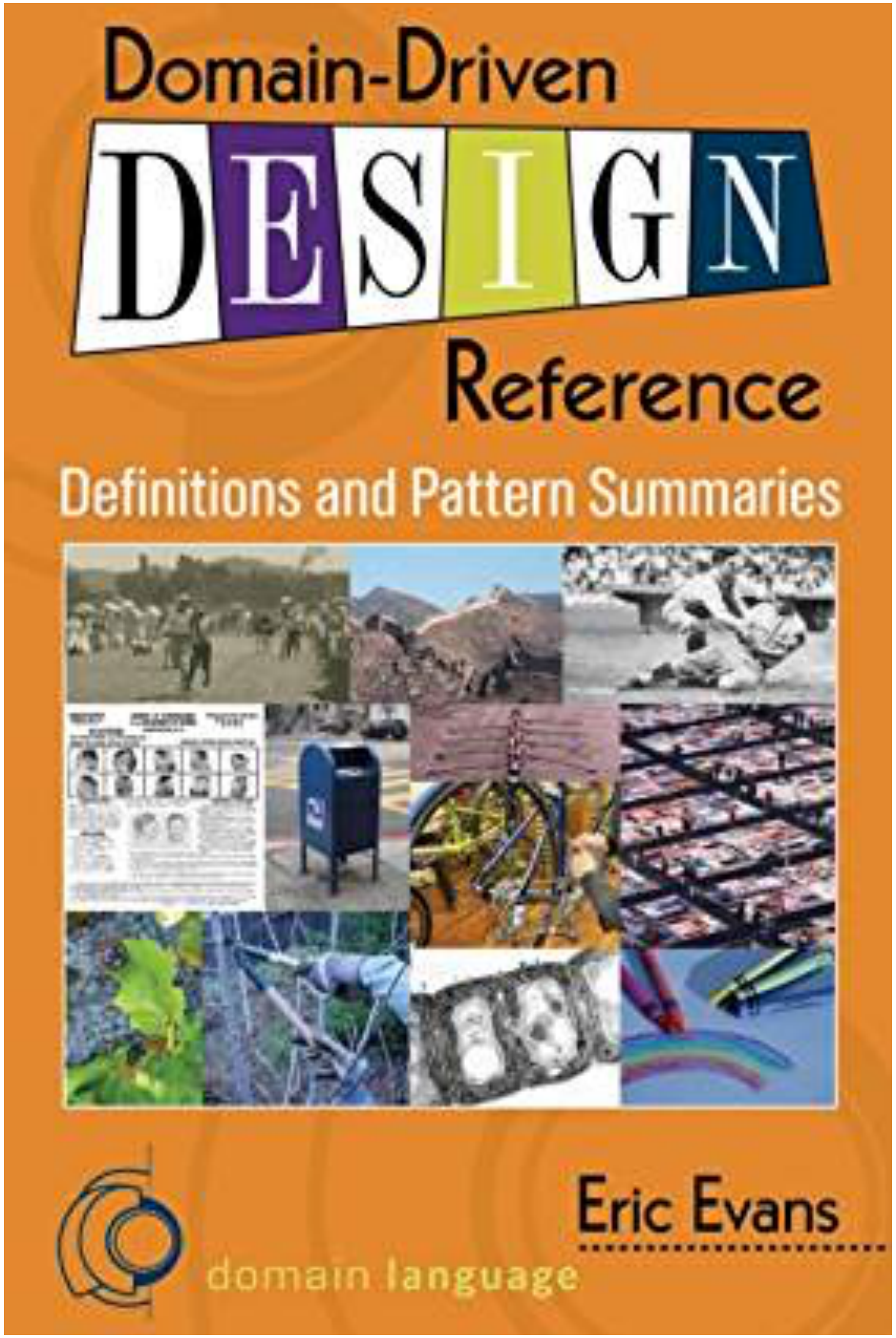
Strategic Patterns



DDD Reference

- Domain-Driven Design Reference: Definitions and Pattern Summaries

-



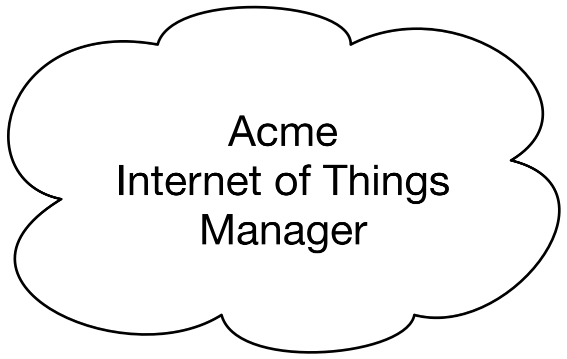
Model

- A system of abstractions that describes selected aspects of a domain
- Can be used to solve problems related to the domain

<< Some UML class diagram >>

Bounded Context

- Large projects have multiple Models
- Combining Models causes bugs
- Model expressions only have meaning in a context
- **Therefore …**
- Define the context of the Model
- Set boundaries in terms of team organization, parts of the application, code bases, and DB schemas



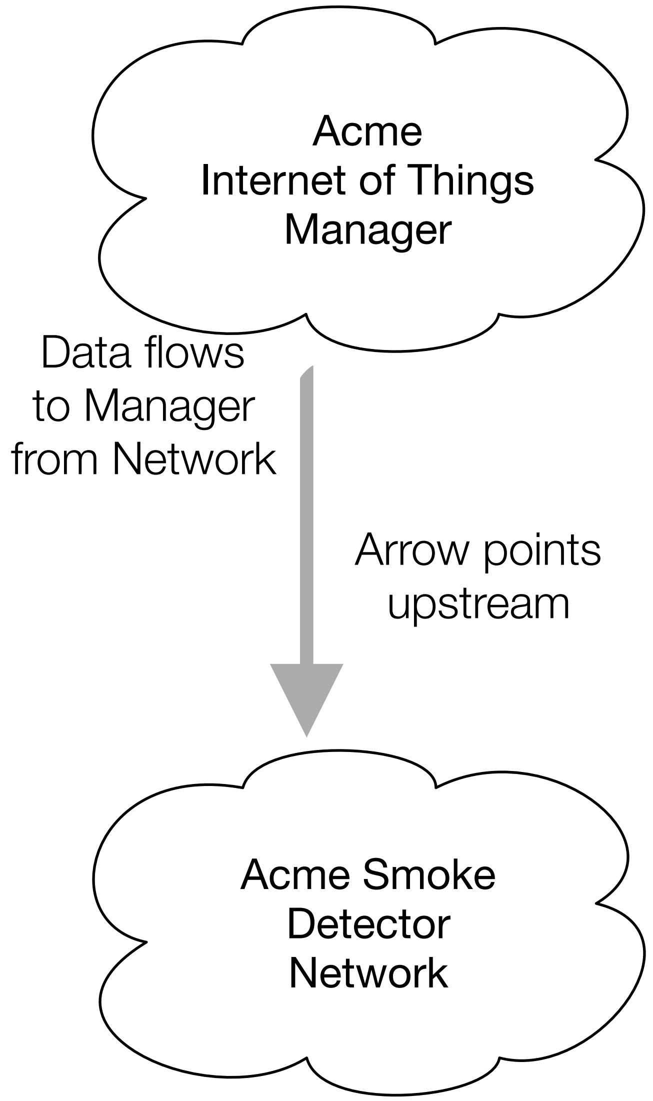
Ubiquitous Language

- Translating between User speak and software developer speak is losses fidelity and is error prone.
- **Therefore …**
- Define a set of names for interrelated concepts from the domain
- Use these names in the implementation code

- User: Fuel onload price at a given port varies based on supplier and region
- Dev: Price has a foreign key to the SupplierRegion table.

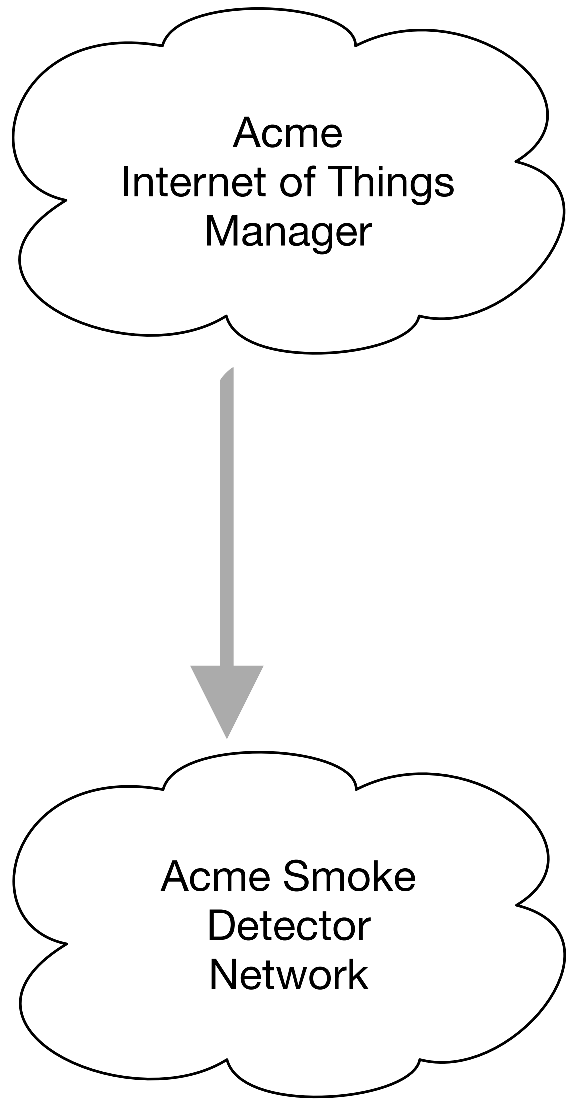
ContextMap

- To develop a strategy, we need a large-scale view across our project and others we integrate with.
- **Therefore …**
- Define each Model and it's BoundedContext
- Describe the points of contact between Models. Identify …
- Explicit translations
- Sharing
- Isolation mechanisms
- Levels of influence


Acme Internet of Things Manager
Data flows to Manager from Network
Arrow points upstream
Acme Smoke Detector Network

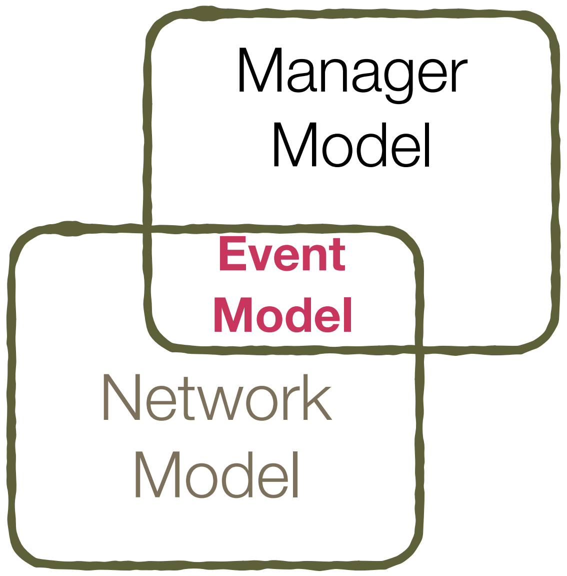
Context Relationship Types

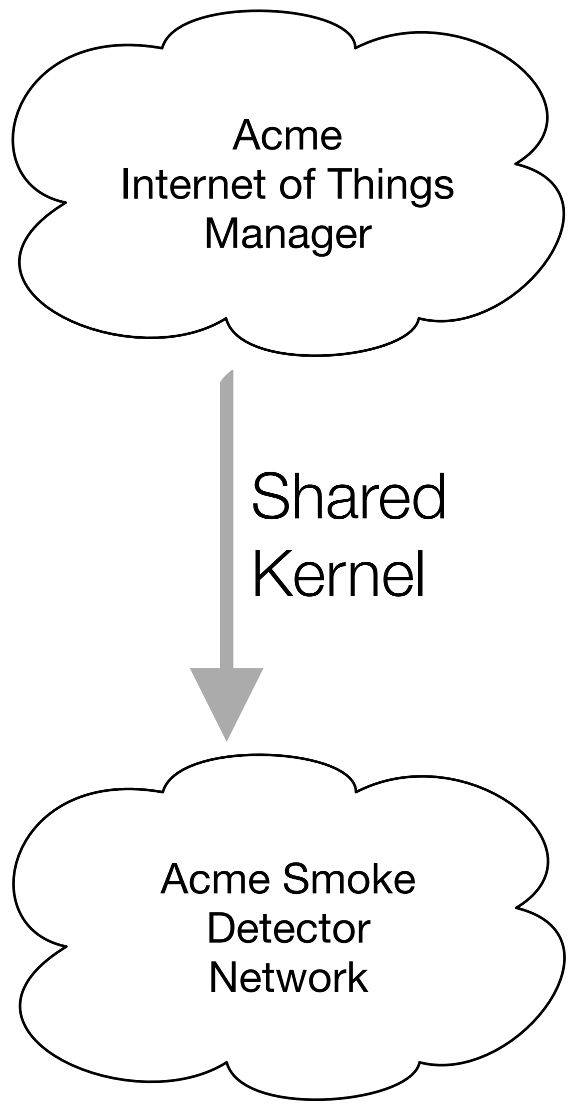
- Shared Kernel
- Customer/Supplier Teams
- Conformist
- Open Host Service
- Separate Ways
- Big Ball Of Mud



Shared Kernel

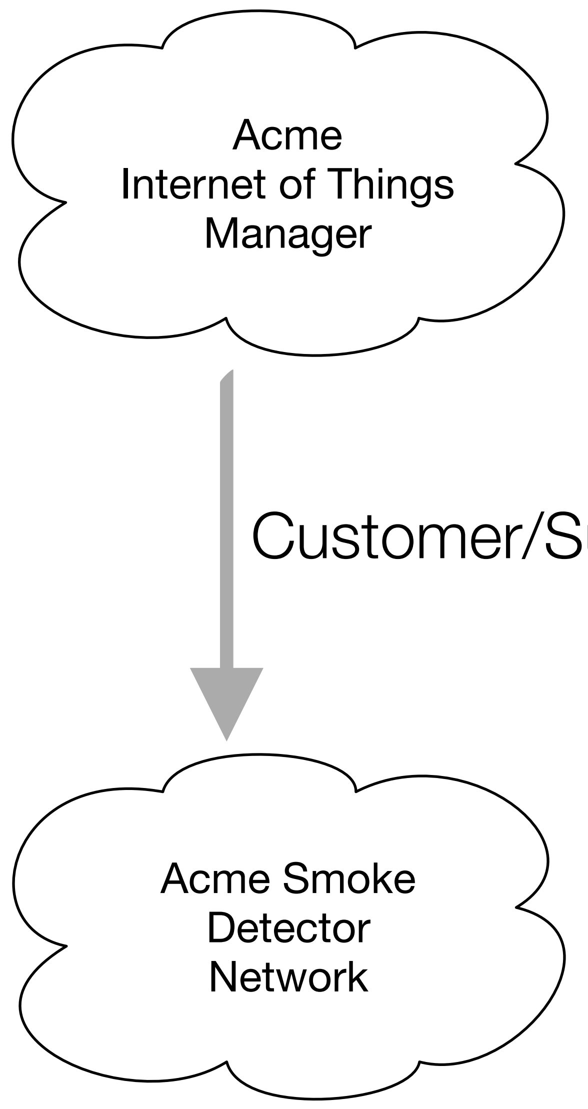
- Designate some subset of the Domain Model that the two teams agree to share.





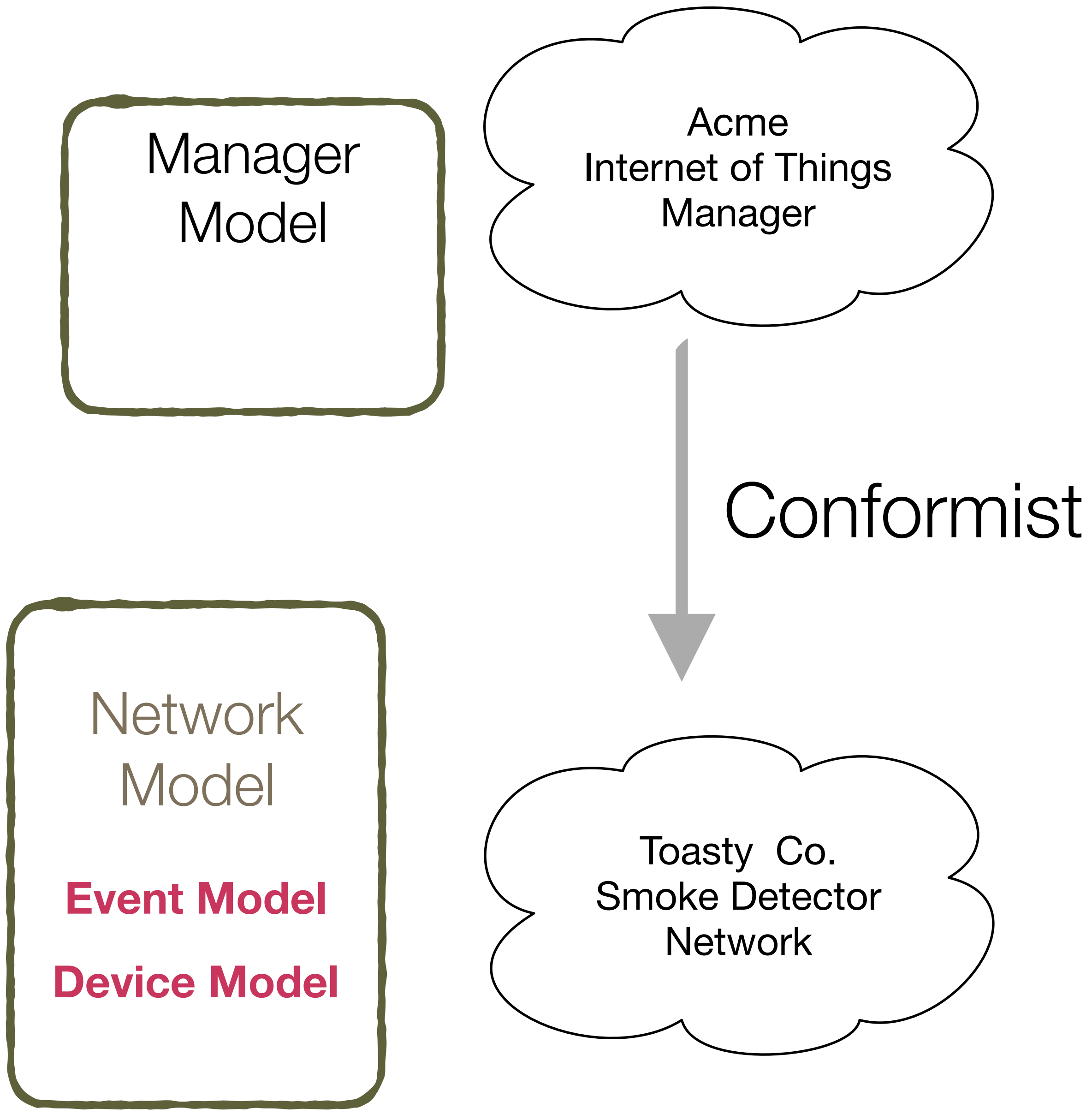
Customer/Supplier Teams

- Establish Clear Customer/Supplier relationship
- The Manager Team is a Customer to the Network Team
- In Agile process Manager Team can add stories to Network Teams Backlog.
- Or Tests to Network Teams test Suite.



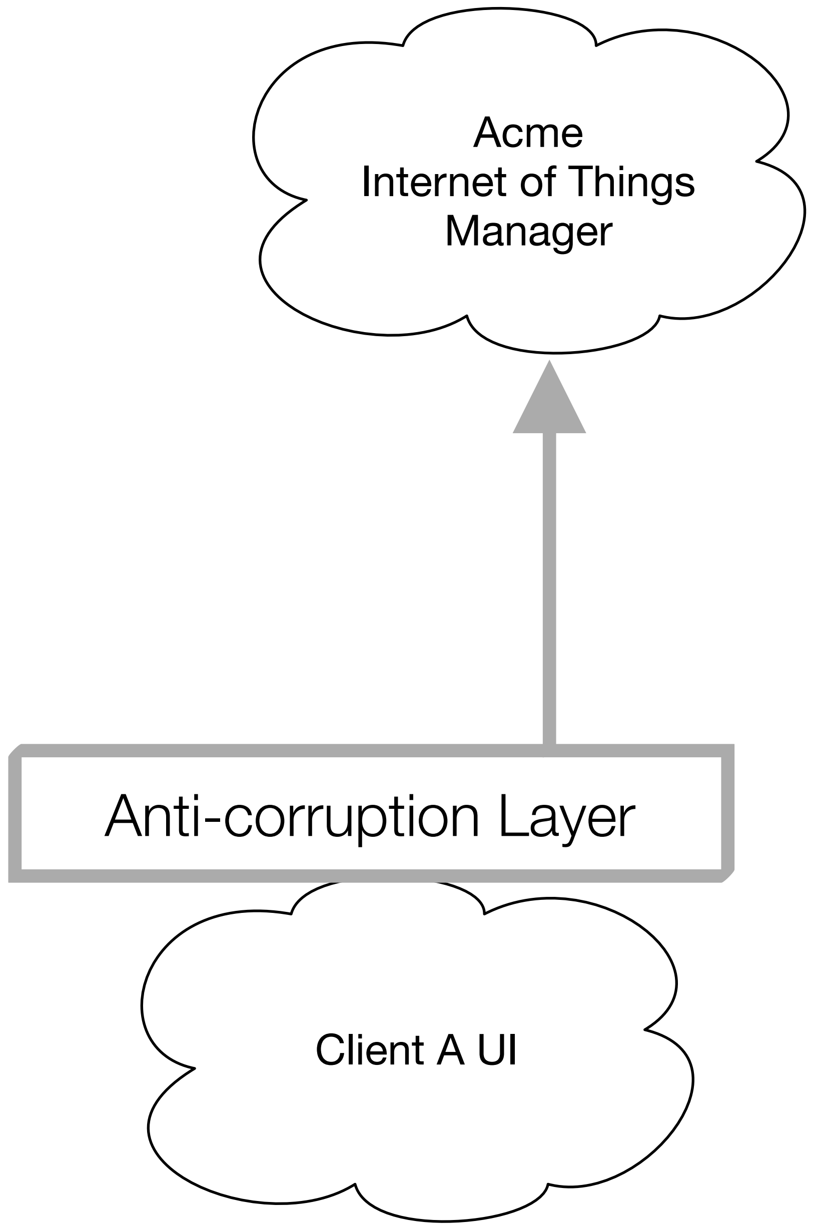
Conformist

- Upstream Team (ToastyCo.) has no interest in adapting to Acme's needs
- Eliminate the Complexity of translating the Event and Device Models, by using the them as-is from the Toasty Co. API
- Also share the Toasty Co. Ubiquitous Language
- Conformity simplifies Integration


Manager Model
Acme Internet of Things Manager
Conformist
Network Model
Event Model
Device Model
Toasty Co. Smoke Detector Network

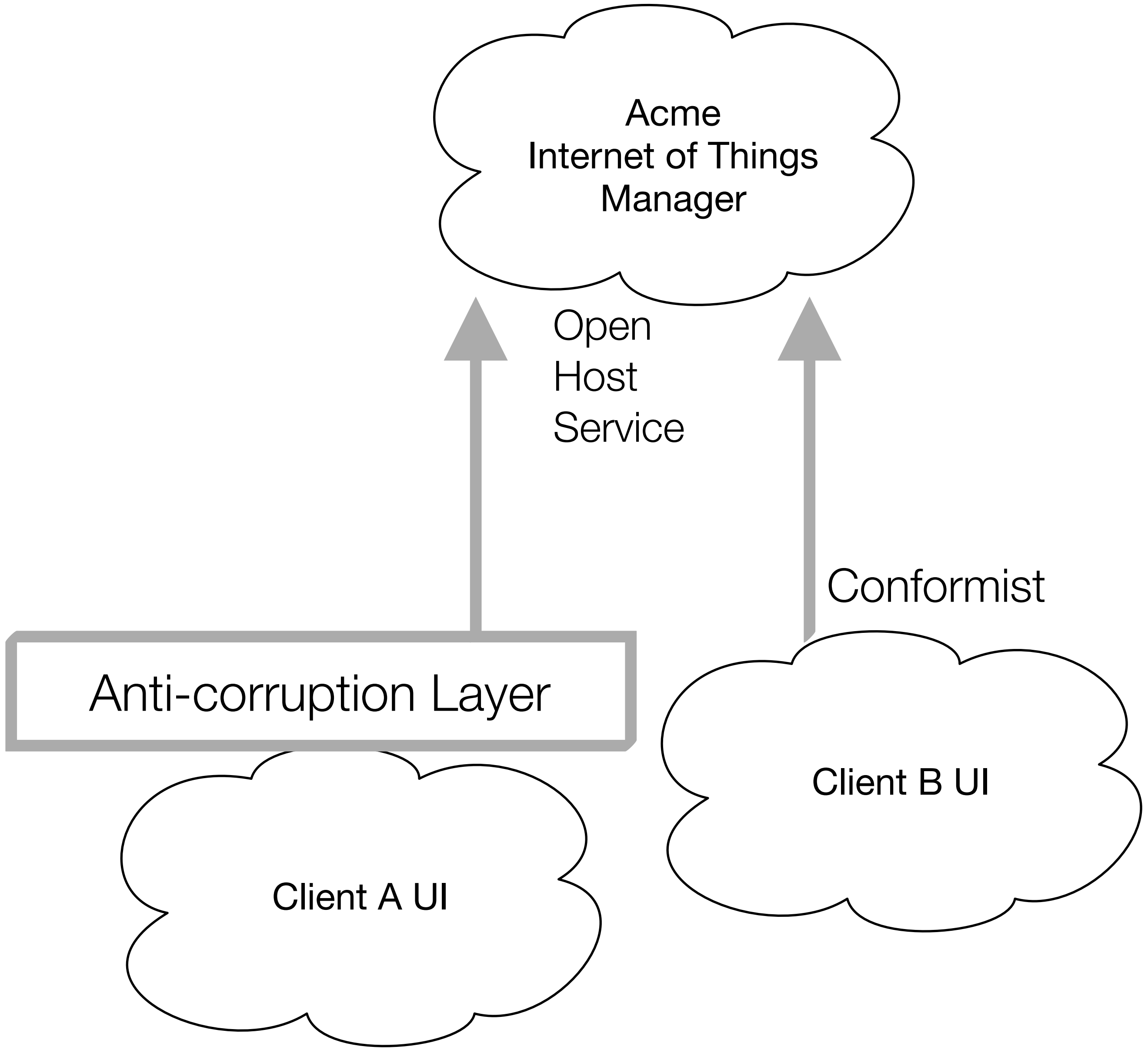
Anti-corruption Layer

- As a downstream Client create an isolating layer to provide the functionality of the upstream (Acme Manager)
- but translated into the (Client A) Model.
- The layer translates in one or both directions



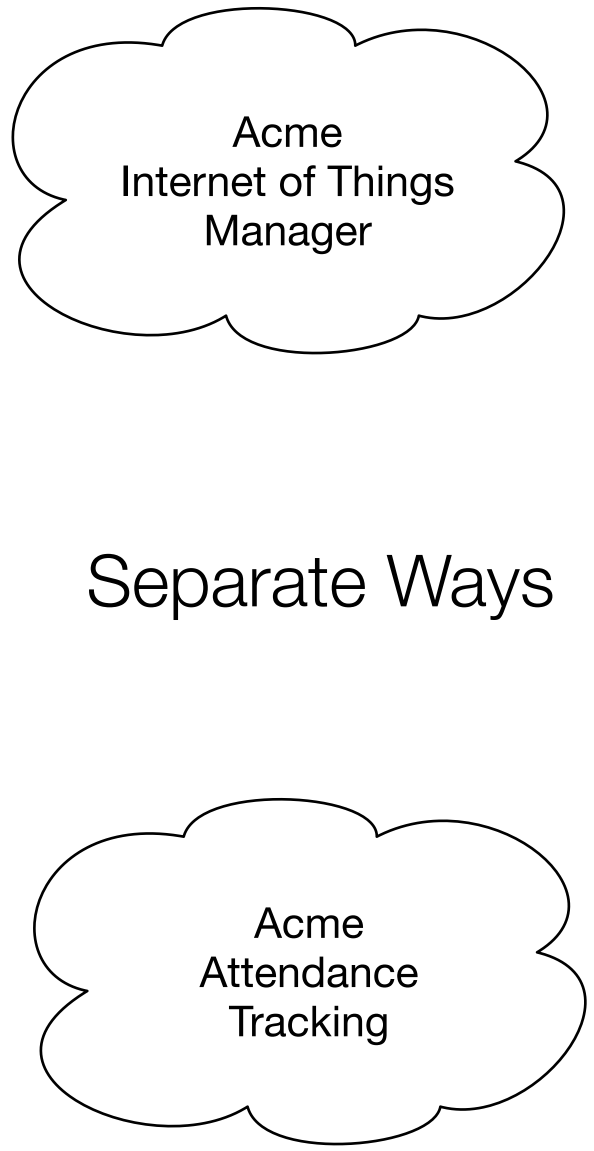
Open Host Service

- When a subsystem needs to be integrated with many others as a set of services.
- Define an open protocol or API for all those that need to use it.



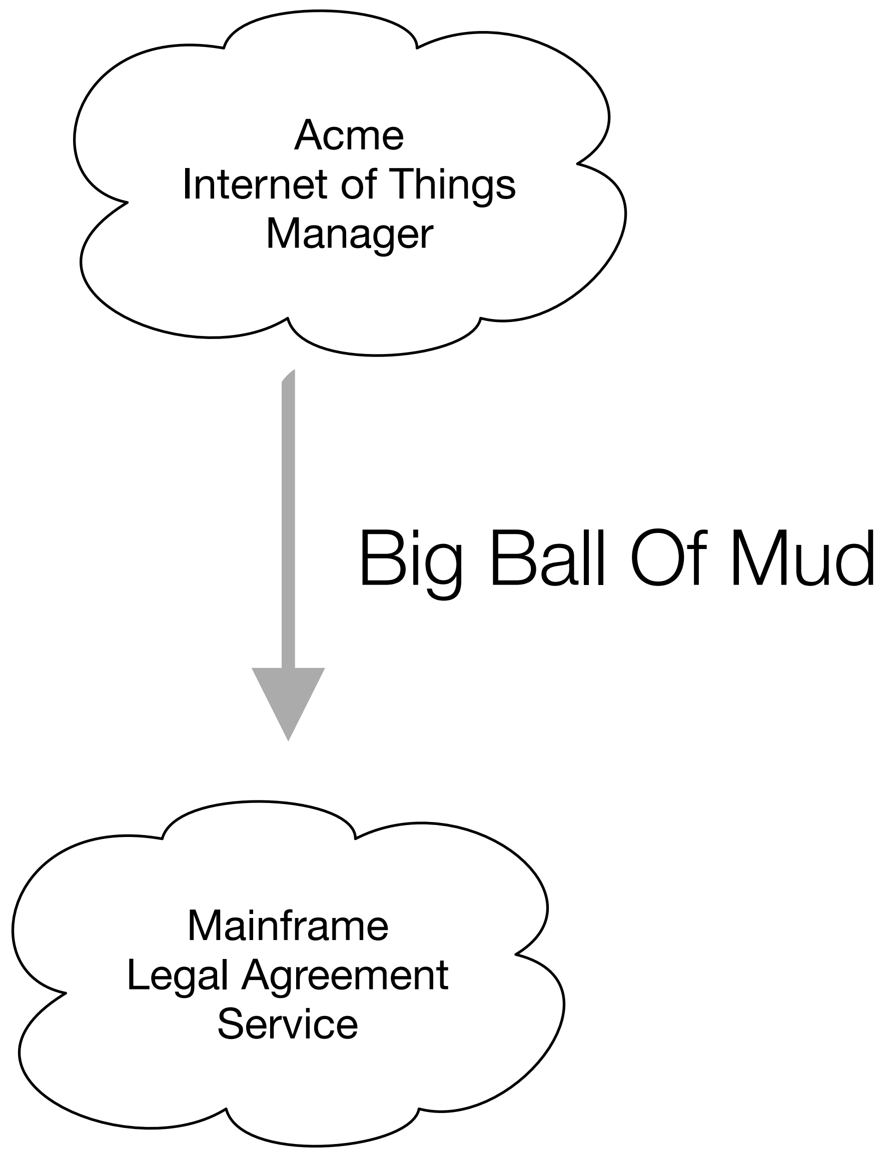
Separate Ways

- If two sets of functionality have no significant relationships.
- Cut them loose from each other.
- So they can each develop simpler Models for their own needs.

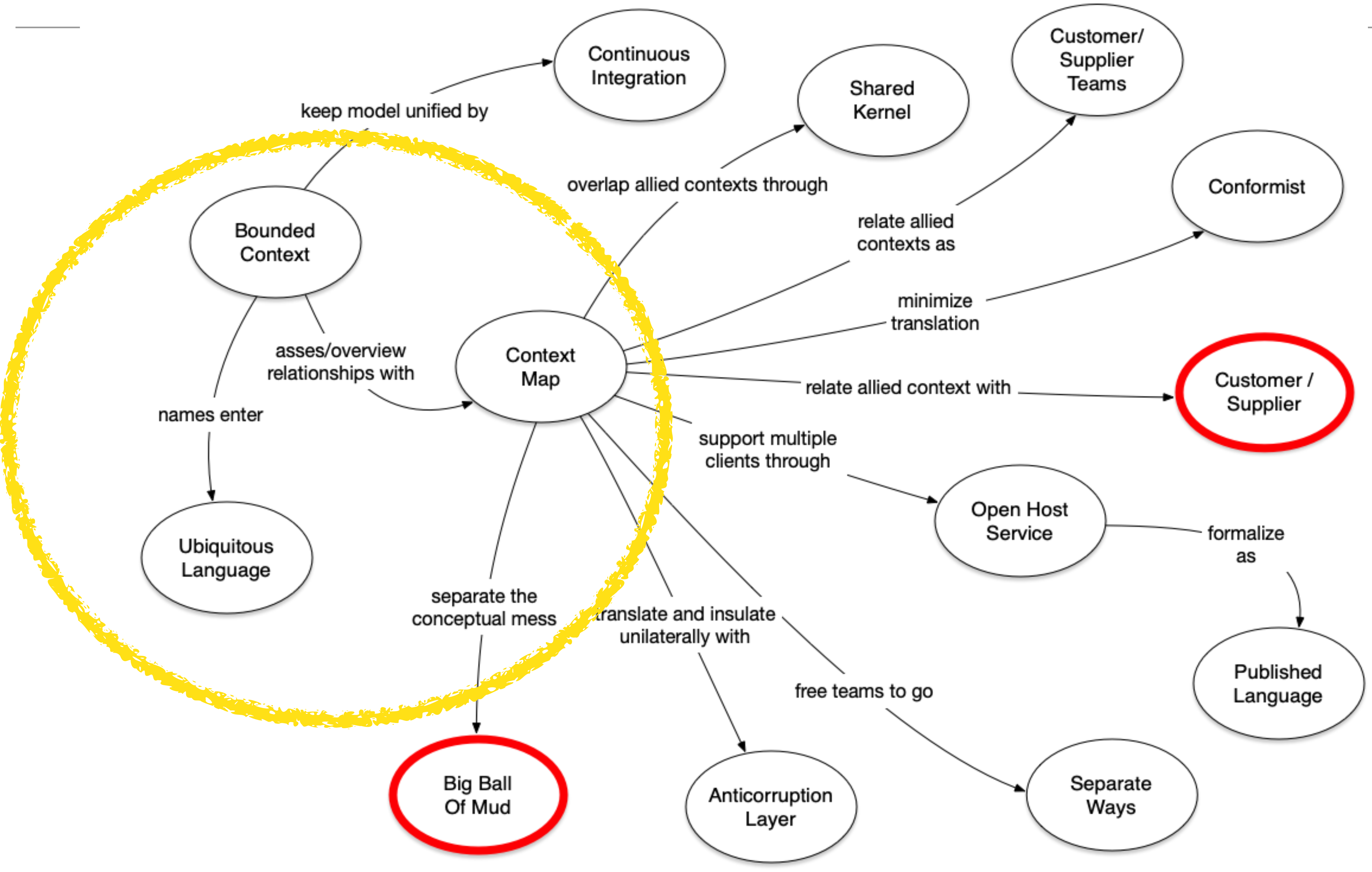


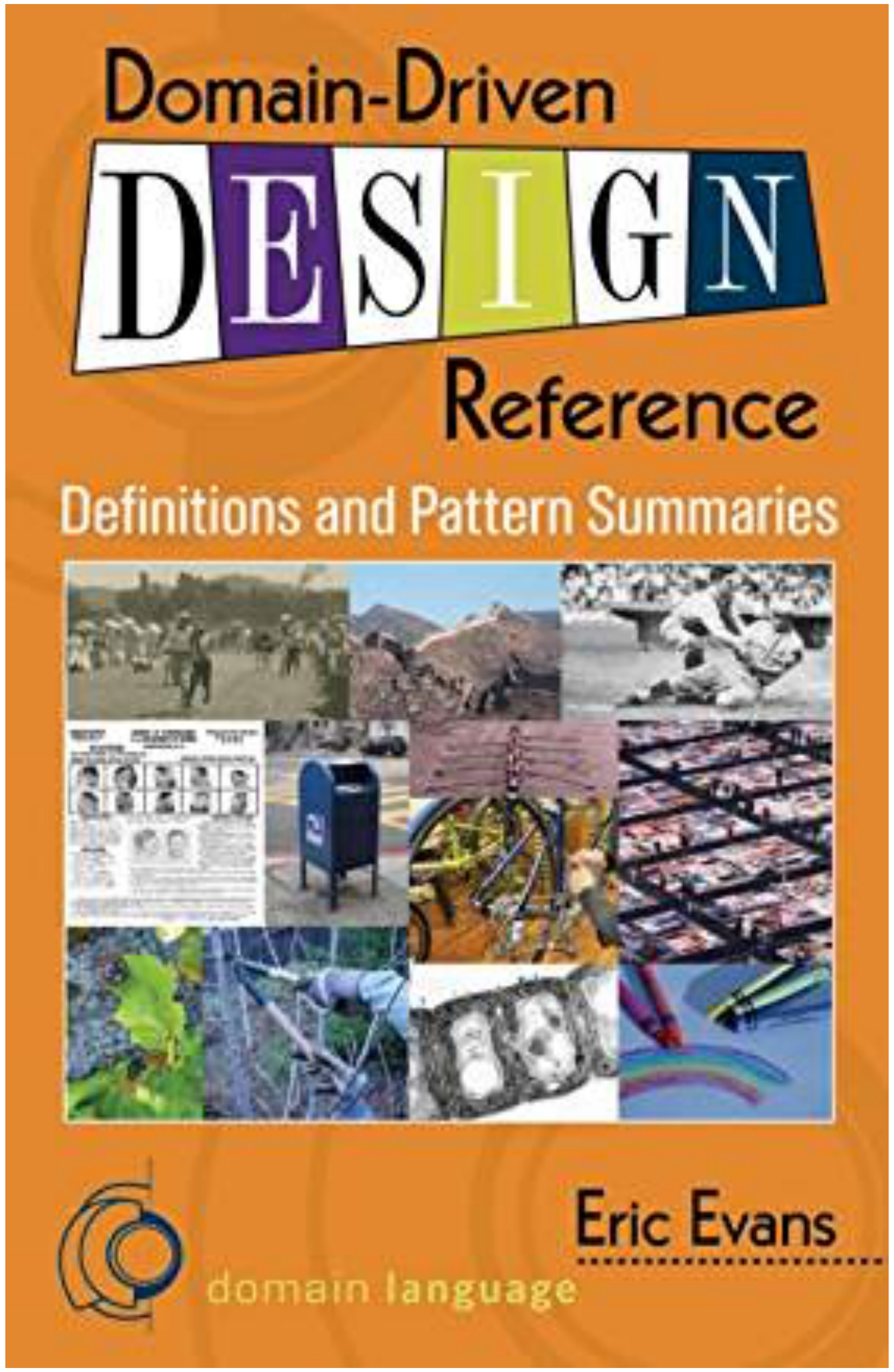
Big Ball Of Mud

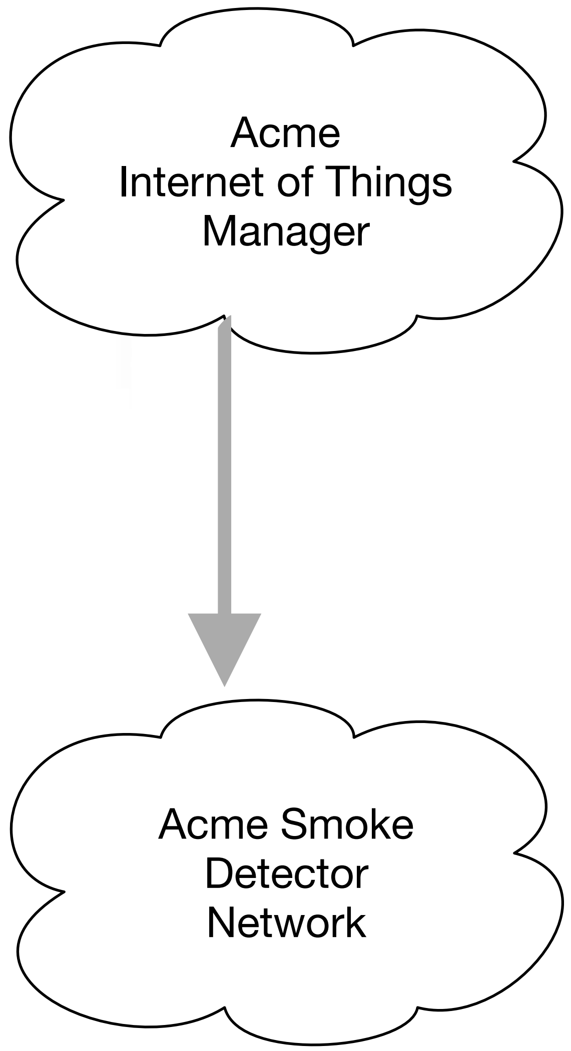
- Sometimes the other system grown over many years has no defined Model. Or multiple Models.
- No well defined Context Boundaries.
- Draw a boundary around the entire mess. Call it a bill of Mud.
- Do not try to DDD inside of that boundary.
- Watch out: The Ball of Mud will try to spread outside of its area.



Strategic Patterns




Domain-Driven
DESIGN
Reference
Definitions and Pattern Summaries
Eric Evans
domain language


Acme
Internet of Things
Manager
Acme Smoke
Detector
Network

The End