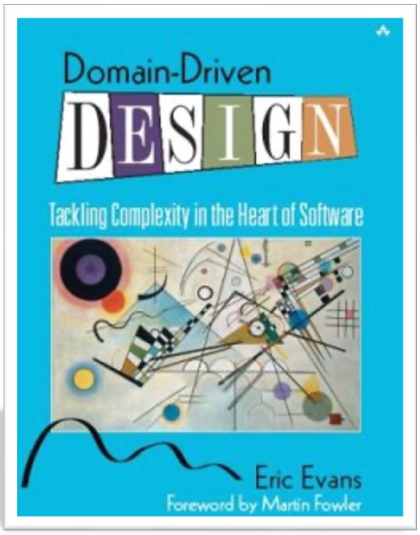
Domain Driven Design

Dominic Duggan
Stevens Institute of Technology

Based in part on material
by Eric Evans and Nathan Farrington

1

1



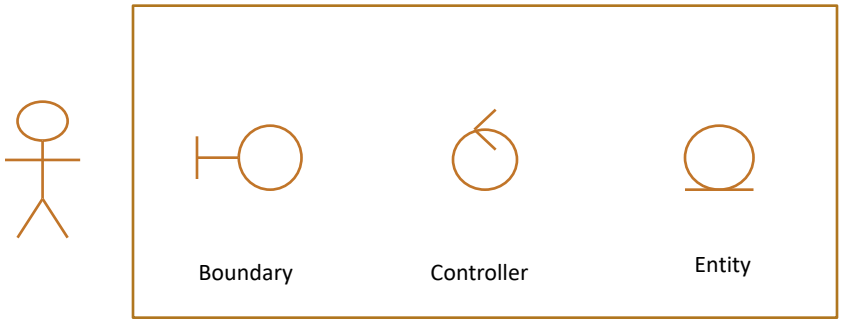
Evans, Eric. *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Addison-Wesley Professional, 2004.

2

Avram, Abel and Marinescu, Floyd. *Domain-Driven Design Quickly.* C4Media, Inc. 2006. Available at: http://www.infoq.com/minibooks/domain-driven-design-quickly.

3

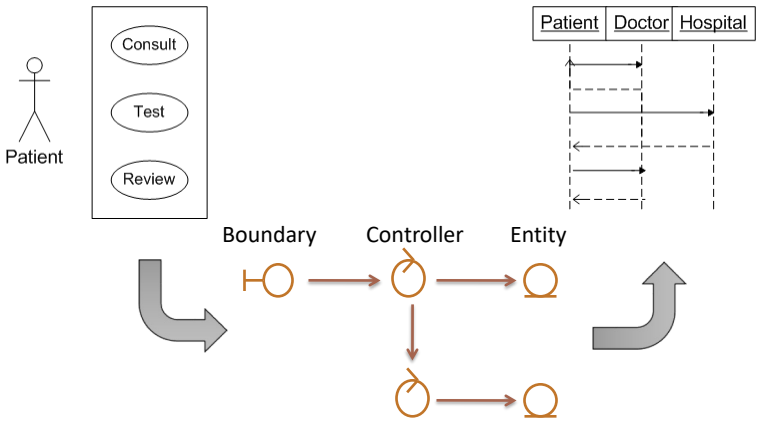
Robustness Diagram



4

4

From Requirements to Design



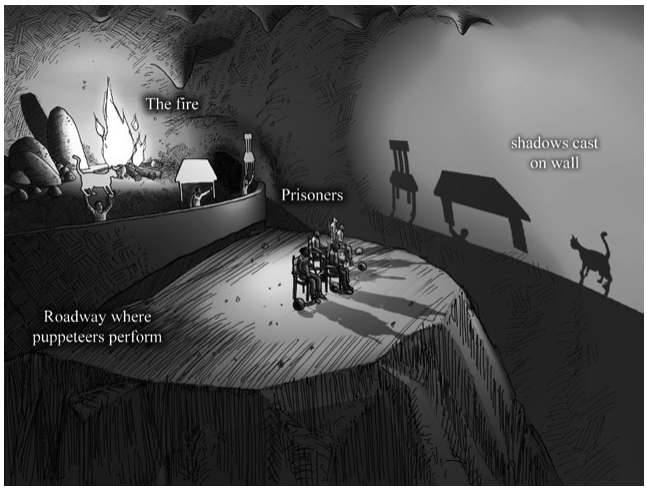
5

From Requirements to Design

- Requirements
  - Identify domain model
  - Provide use cases
  - Interviews with domain experts
- Design
  - Domain model to software design
  - Business logic for use cases
  - Developers

6

Developer = Prisoner in Plato's Cave



7

7

Domain-Driven Design

- Model-driven design
- Involve developers in domain analysis
- Developers translate analysis into software abstractions
  - Domain concepts as OO abstractions
  - Resulting software design is the domain model
  - Changes to software require changes to model
  - Avoid gap between domain model and design
  - Emphasis on model rather than executable (agile)

8

8

Domain-Driven Development

```
while (projectFunded)
{
   showLatestSoftwareToDomainExperts();
   updateDomainModelTogether();
   updateCodeToReflectDomainModel();
}
```

9

Model-Driven Design

- Model: A system of abstractions that describes selected aspects of a domain and can be used to solve problems related to that domain.
- Serves a particular use
  - Not “as realistic as possible”
  - Useful relative to specific domain scenarios

10

10

Ubiquitous Language

- Software developers & domain experts
- Undefined terminology and acronyms
- *Ubiquitous Language: "A language structured around the domain model and used by all team members to connect all the activities of the team with the software."*

11

Context

- *Context:* The setting in which a word or statement appears that determines its meaning
- *Bounded Context:* Defines the boundaries of submodels of the domain
  - *Maximal cohesion and minimal coupling*
- *Context map:* Defines relationships between these submodels

12

12

It was six men of Indostan
To learning much inclined,
Who went to see the Elephant
(Though all of them were blind),
That each by observation
Might satisfy his mind.

The *First* approached the Elephant,
And happening to fall
Against his broad and sturdy side,
At once began to bawl:
"God bless me! but the Elephant
Is very like a wall!"

. . .

The *Third* approached the animal,
And happening to take
The squirming trunk within his hands,
Thus boldly up and spake:
"I see," quoth he, "the Elephant
Is very like a snake."

The *Fourth* reached out his eager hand,
And felt about the knee.
"What most this wondrous beast is like
Is mighty plain," quoth he;
"'Tis clear enough the Elephant
Is very like a tree!"

. . .

The *Sixth* no sooner had begun
About the beast to grope,
Than, seizing on the swinging tail
That fell within his scope,
"I see," quoth he, "the Elephant
Is very like a rope!"

And so these men of Indostan
Disputed loud and long,
Each in his own opinion
Exceeding stiff and strong,
Though each was partly in the right,
And all were in the wrong!

. . .

—From "The Blind Men and the Elephant," by John Godfrey Saxe (1816-1887), based on a story in the Udana*, a Hindu text*

13

CORE PATTERNS

14

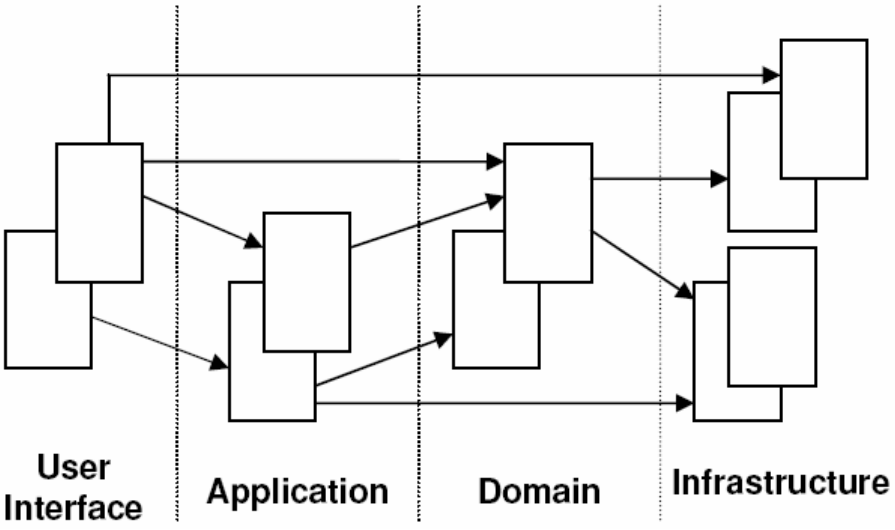
Layered Architecture

- Structure for domain model
- Distinguish “big picture” from details

15

15

Layered Architecture



16

Layered Architecture

- User Interface:
  - Presents information
  - Interprets commands
- Application Layer:
  - Orchestrates domain objects
  - *State reflecting the progress of a service call*
- Domain Layer:
  - Represents business concepts, situation, rules.
  - *State reflecting the business situation*
- Infrastructure Layer: Generic capabilities

17

Three Kinds of Objects

- *Entities:*
  - Object **identity**
- *Value Objects:*
  - **Immutable** objects
  - No conceptual identity
  - Describe some attribute of another object
- *Services:*
  - No state
  - Represent an **operation**

18

Entities

- Entity identity
  - Based on domain-specific attributes
- How entities act upon each other
  - Distinguishes it from data modeling
  - Distinguishes it from SOA
- Domain-specific constraint-checking

19

19

Entities & Constraint Checking

```
class Patient {
  Age age;
  Weight weight;
  FlowSheet currentFlowSheet () { ... }
  boolean dangerSigns (BloodPressure bp, Temperature temp) {
    return ...;
  }
}
class FlowSheet {
  Patient patient;
  Station station;
  void recordObs (Time time, BloodPressure bp, Temperature temp) {
    ... record the observations ...
    if (patient.dangerSigns(bp,temp))
      station.alert(time,patient);
  }
}
```

20

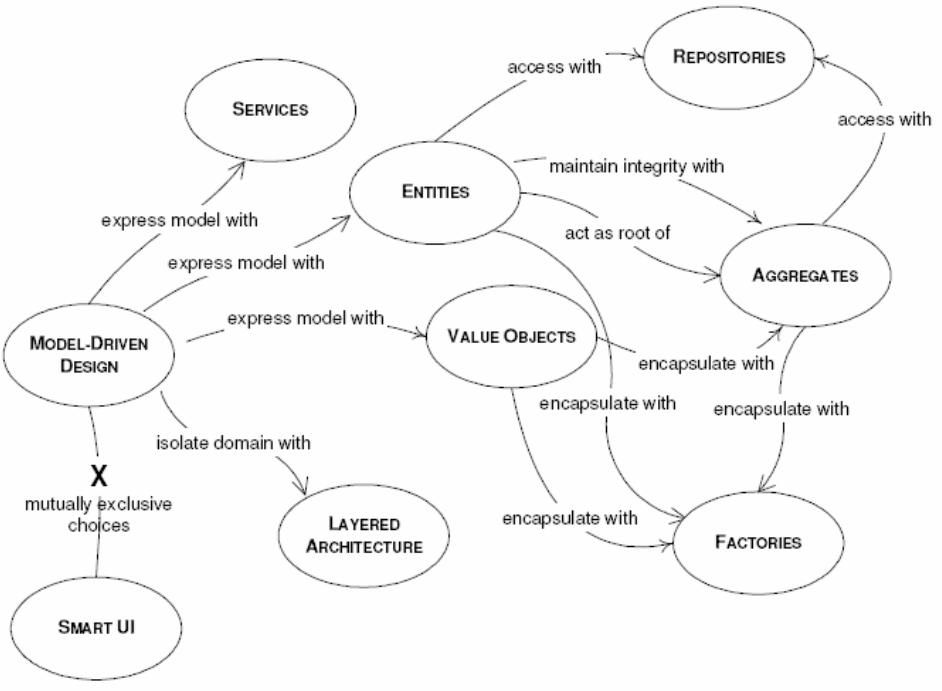
20

Value Objects

- Not all domain values are entities
  - E.g. consent form
  - Immutable, may be copied and distributed
- Different from Data Transfer Objects (DTOs)
  - DTOs are just containers

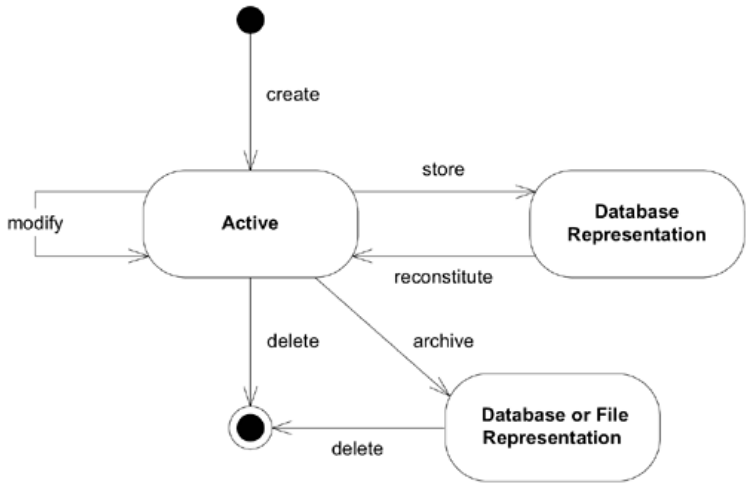
21

21



22

Domain Object Life Cycle



23

Aggregate

- Organize entities into hierarchies for access
  - Transactional update
  - Lazy loading
  - Logging accesses
- *Aggregate: A cluster of associated objects that are treated as a unit with regard to data changes.*
  - All external access through root Entity
  - Enforces invariants with other objects

24

24

Aggregate

- Instead of:

```
patient.currentFlowSheet().recordObs(...);
```

- do

```
class Patient {
  void recordObs (Time time,
                  BloodPressure bp,
                  Temperature temp) {
    this.currentFlowSheet()
        .recordObs(time,bp,temp);
  }
}
patient.recordObs(...);
```

25

25

Factory

- Creation code is different from run-time behavior code
- *Factory: Encapsulates the knowledge necessary for complex object creation.*
- Relevant patterns:
  - Factory Method
  - Abstract Factory
  - Builder

26

Repository

- How do you find an Aggregate?
  - Navigate a network of object references.
    - Tight coupling
    - Reference pollution
  - Pull data from the database
    - SQL inside domain objects?
    - Data hiding principles

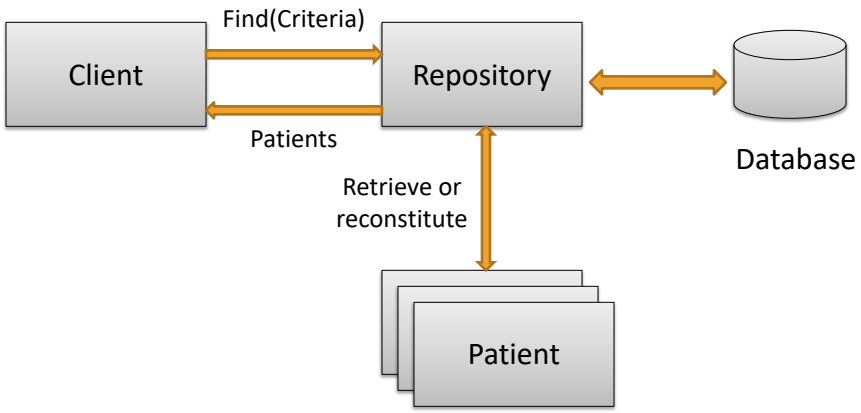
27

Repository

- *Repository: Encapsulates database storage, retrieval, and search behavior which emulates a collection of objects.*
  - Looks like java.util.Collection
  - Acts like a relational database

28

Repository



29

29

ADVANCED PATTERNS: PRESERVING MODEL INTEGRITY

30

Overall Problem

- Teams must work in parallel
- $O(n^2)$ communication channels
- Eventually the domain model becomes inconsistent
- *Solution: Several small models with well-defined relationships*

31

Preserving Model Integrity

- Bounded Context
- Continuous Integration
- Context Map
- Shared Kernel
- Customer-Supplier
- Conformist
- Anticorruption Layer
- Separate Ways
- Open Host Service
- Distillation

32

Bounded Context

- Where does one domain model end and the next one begin?
- *Bounded Context: The delimited applicability of a particular model*
  - Shared understanding of what has to be consistent
  - What can be developed independently

33

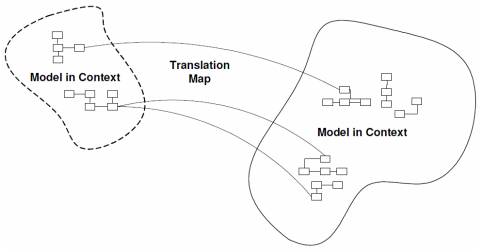
Continuous Integration

- Within a single Bounded Context, we still have $O(n^2)$ communication channels
- All updates to the domain model must be reflected in the code, and vice versa
- *Continuous Integration*
  - Process to harmoniously update the model
  - Procedure to merge model into the code
  - Procedure to merge code into the model
  - Automated builds
  - Automated tests

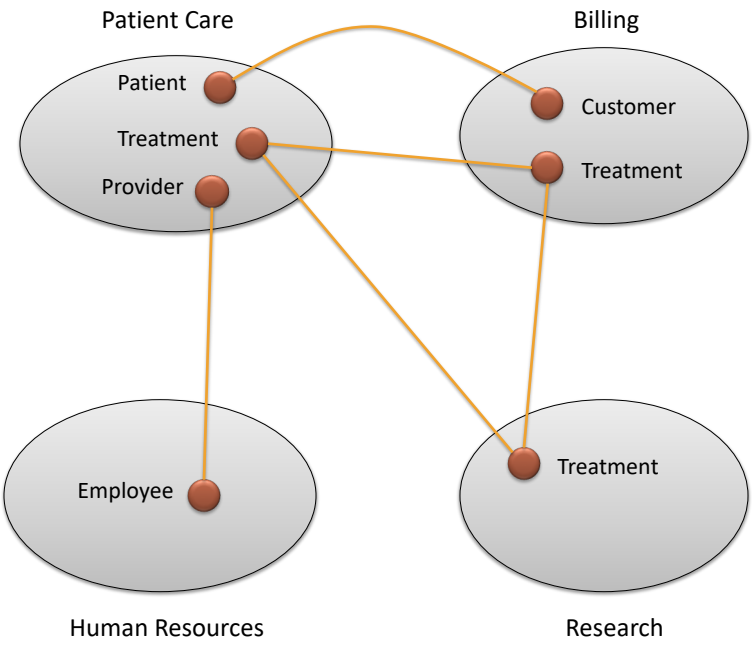
34

Context Map

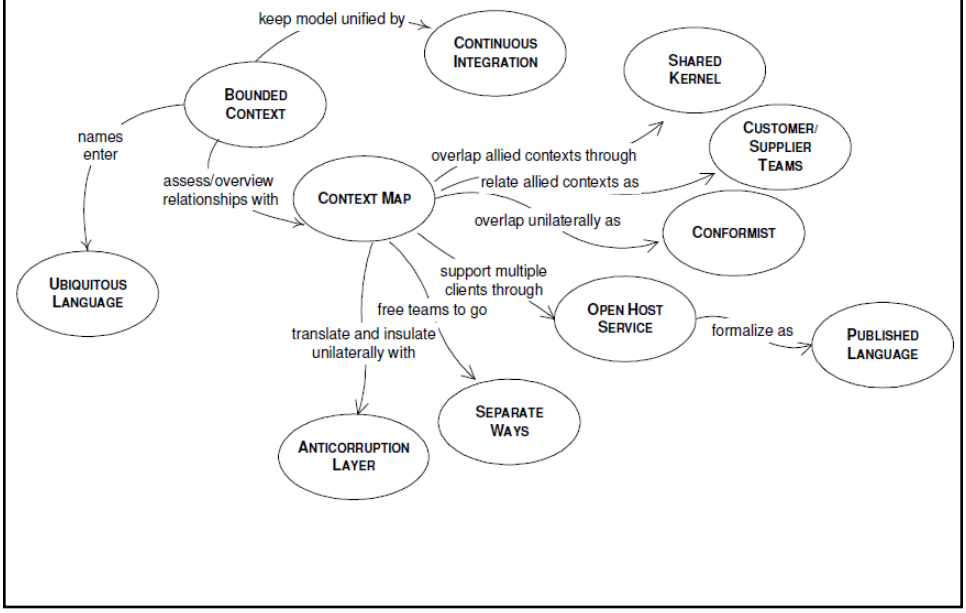
- Model for our models
  - shared understanding
- *Context Map: A document or diagram which outlines the different Bounded Contexts and the relationships between them*



35



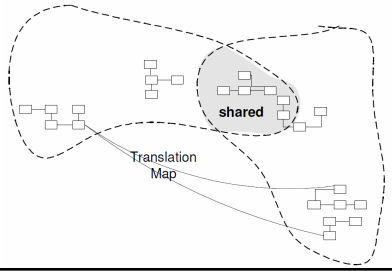
36



37

Shared Kernel

- Uncoordinated teams
- Subsystems closely related
  - Danger: Lack of interoperation
  - Danger: Code duplication
- *Shared Kernel: Designate some subset of the domain model that two teams agree to share*



38

Customer-Supplier

- One subsystem depends on another
  - Shared Kernel is not applicable
  - Both teams are under same management
- *Customer-Supplier:*
  - Customer gives requirements to supplier
  - Supplier implements strict interface for customer
  - Automated tests
    - supplier has not broken the interface

39

Conformist

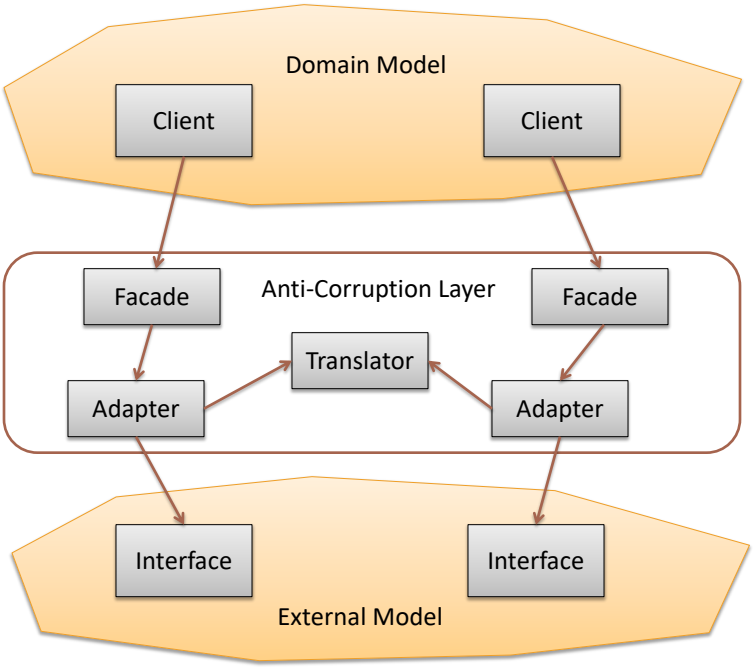
- One subsystem depends on another
  - Shared Kernel is not applicable
  - Supplier does not want to help the customer
  - Customer still wants supplier's model
- *Conformist*
  - Customer uses supplier's model
  - Could also create an adapter

40

Anticorruption Layer

- Must connect to external legacy software.
  - No value from the external model
  - Isolate internal model from external model
- *Anticorruption Layer: Works as a two way translator between two domains and languages*

41



42

Separate Ways

- Connecting two subsystems is more trouble than it's worth
  - Domain models are naturally independent
- *Separate Ways: Create separate Bounded Contexts and do the domain modeling independently*

43

Open Host Service

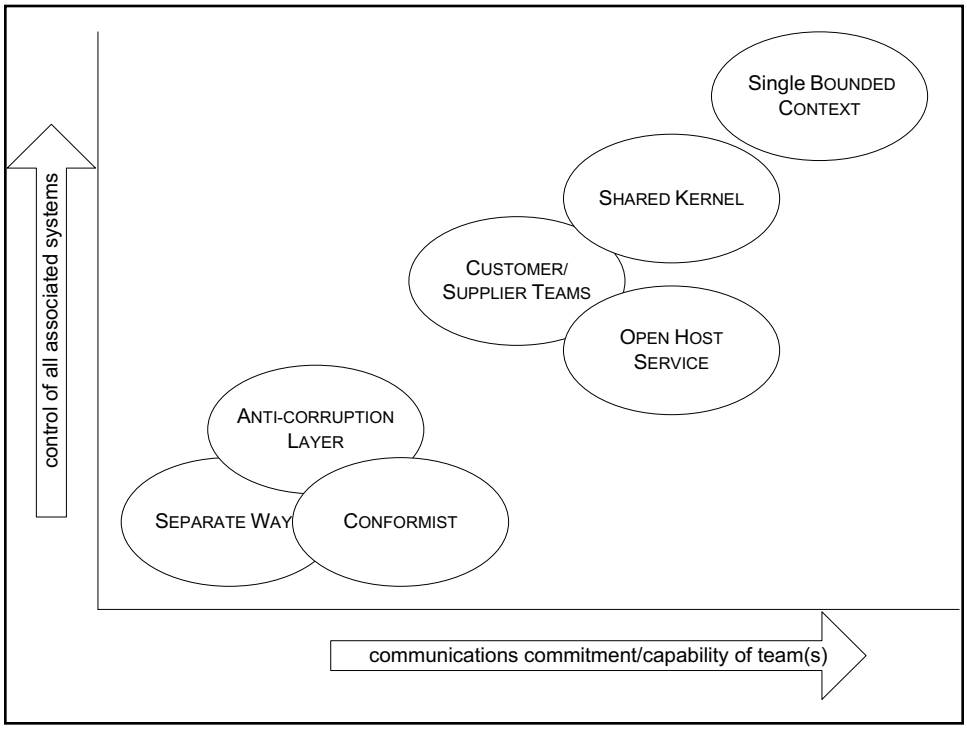
- We need to create $O(n^2)$ translation layers, one for every pair of subsystems.
  - Significant code duplication
- *Open Host Service:*
  - External subsystem publishes API for a set of Services
  - API is extended for new integration requirements
  - No translators are required

44

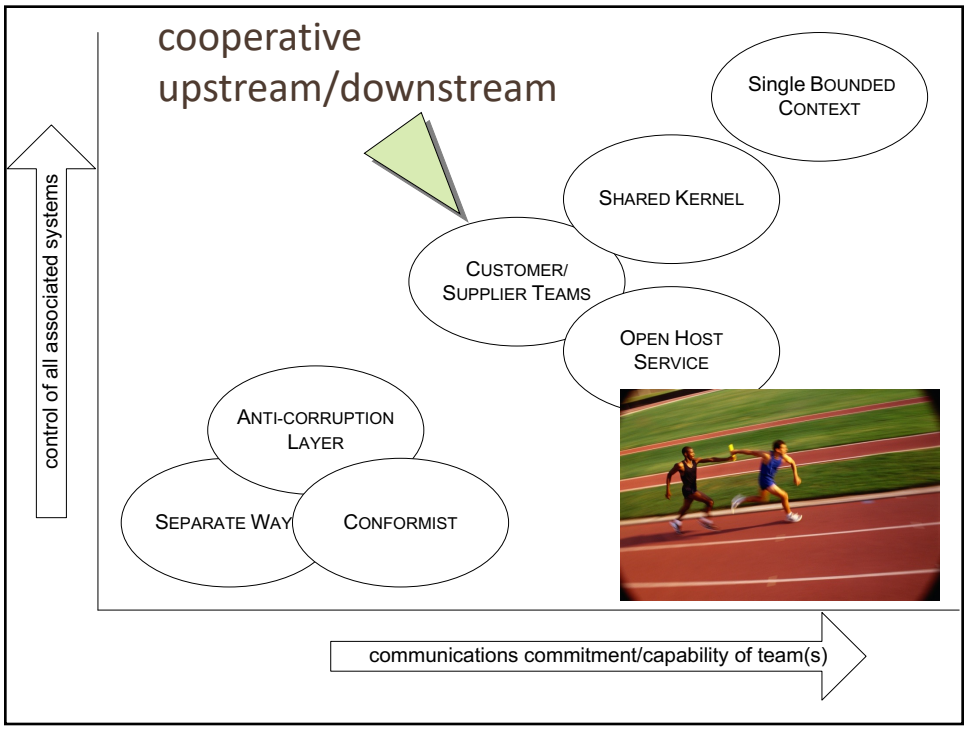
Distillation

- Big domains have big domain models
  - The essence is obscured
- *Distillation*
  - Define a *Core Domain* for the essence
  - Byproducts of Distillation: *Generic Subdomains*

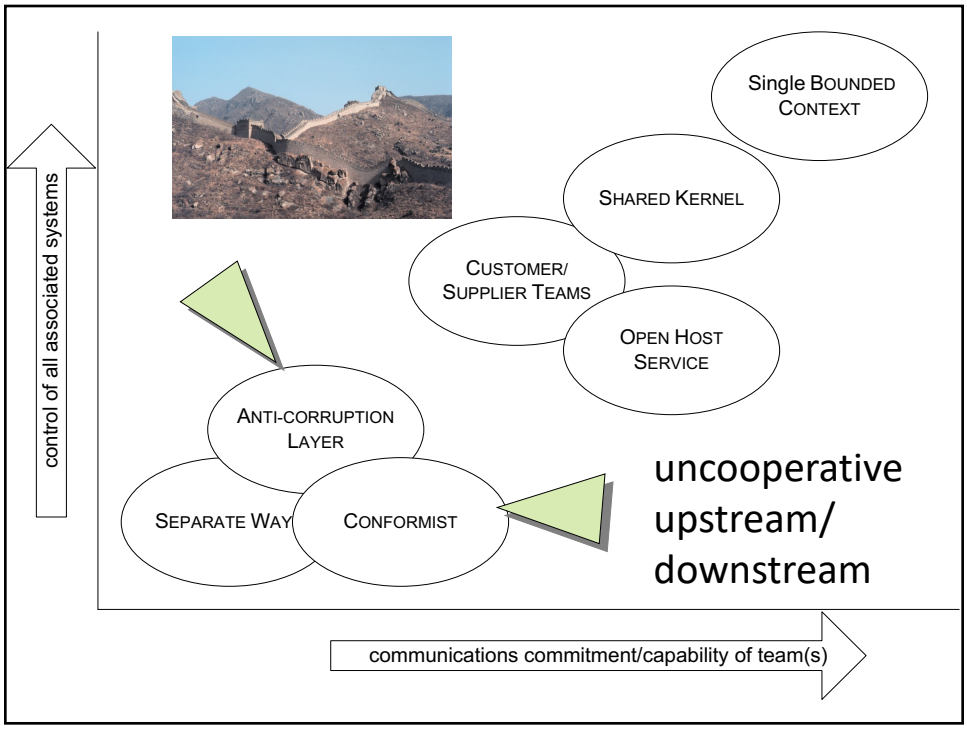
45



46


cooperative
upstream/downstream
Single Bounded Context
Shared Kernel
Customer/ Supplier Teams
Open Host Service
Anti-corruption Layer
Separate Way
Conformist
control of all associated systems
communications commitment/capability of team(s)

47


Single Bounded Context
Shared Kernel
Customer/ Supplier Teams
Open Host Service
Anti-corruption Layer
Separate Way
Conformist
uncooperative
upstream/
downstream
control of all associated systems
communications commitment/capability of team(s)

48