Challenges, Pains and Points of Software Development Today

The world is being rebuilt in code and talent is lacking.

Dino Esposito

The world is being rebuilt in code

Meaning that ...

- Every company is a tech company
- Every company needs strong talents
- Talent is lacking
- Every company fights to acqui-hire “fish”
- No companies plan to teach “how to fish”
- Huge barrier between theory and practice
- Just shortage of senior people

What it means to be a software writer?

Challenges

- **Domain analysis**
  - Modeling vs. mirroring the business domain
  - DDD and tools for domain analysis
- **Implementation**
  - Focus on tasks and simplicity
  - UX-first methodology
- **Technology**
  - Mere infrastructure
  - Choose and handle with care

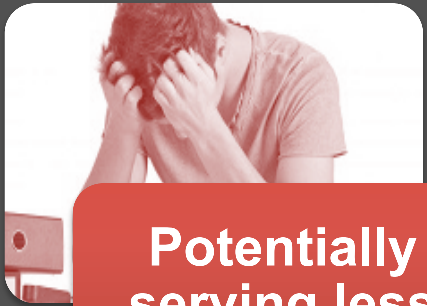
Domain Analysis

Modeling

- **The God Anti-pattern**

  Developers to design an ideal model of the domain rather than just mirroring what they see

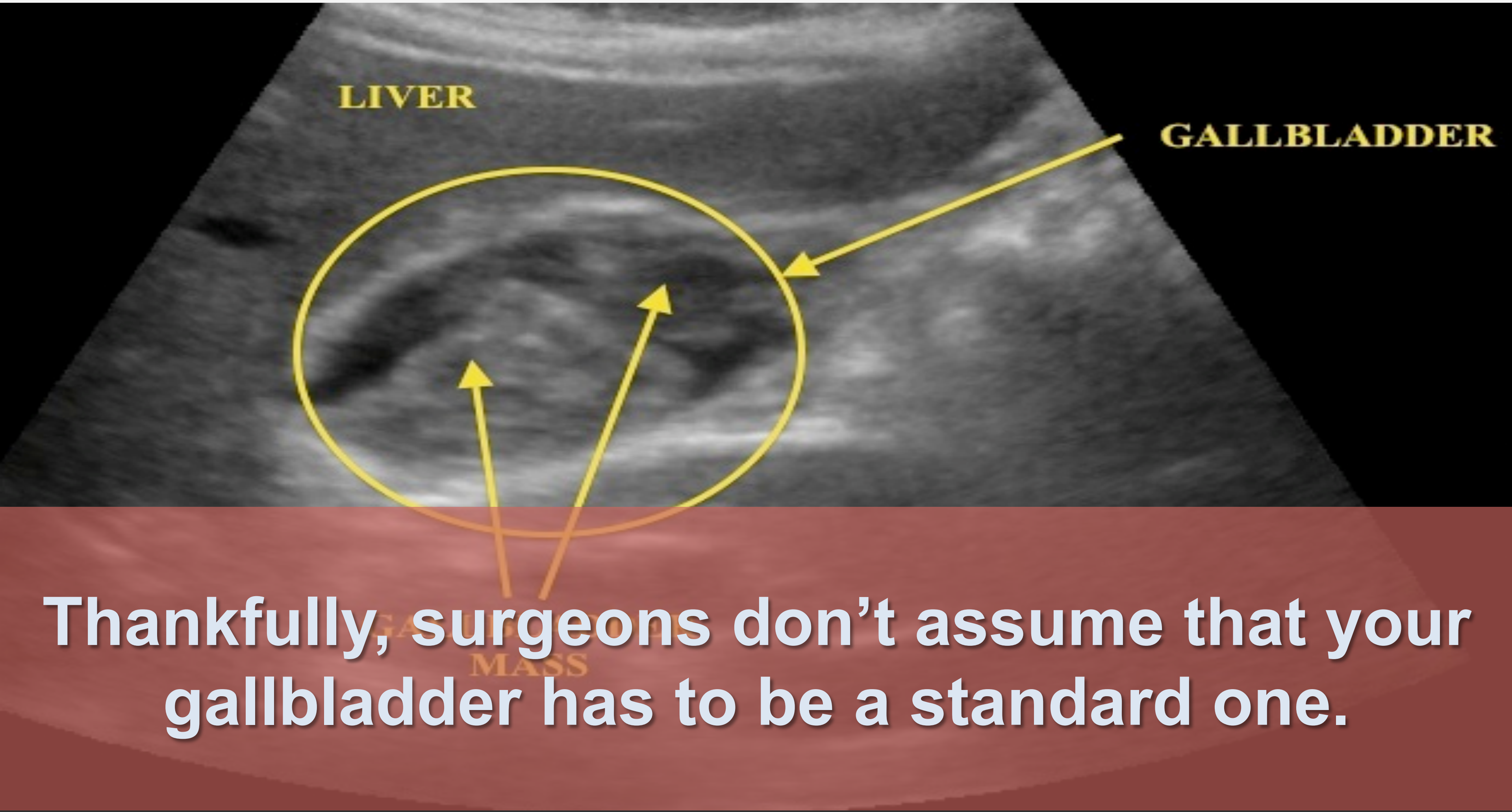
Virtual reality



Potentially serving less than optimal experience

Potentially missing nonfunctional requirements

If Surgeons Were Software Architects



Thankfully, surgeons don't assume that your gallbladder has to be a standard one.

Oh yes, good modeling…

- **YAGNI, KISS, DRY**

  Pure tautology at architecture level.

- **Tell-don’t-Ask**

  Much better help to stay focused on what’s really required in the domain.

- **Think-ahead?**

  Comes naturally once you’ve managed to understand the mechanics of the business domain.

Mirroring

- **Discovering the top-level architecture**

  Top-level architecture mirrors the real world. The actual implementation models the real-world with any due and inevitable approximation that software may face.

The world is being rebuilt in code.

Software must implement and streamline real business processes. Software not to redesign business processes. That’s a different job.

Behavior

- **Mirroring is just better analysis of the domain**

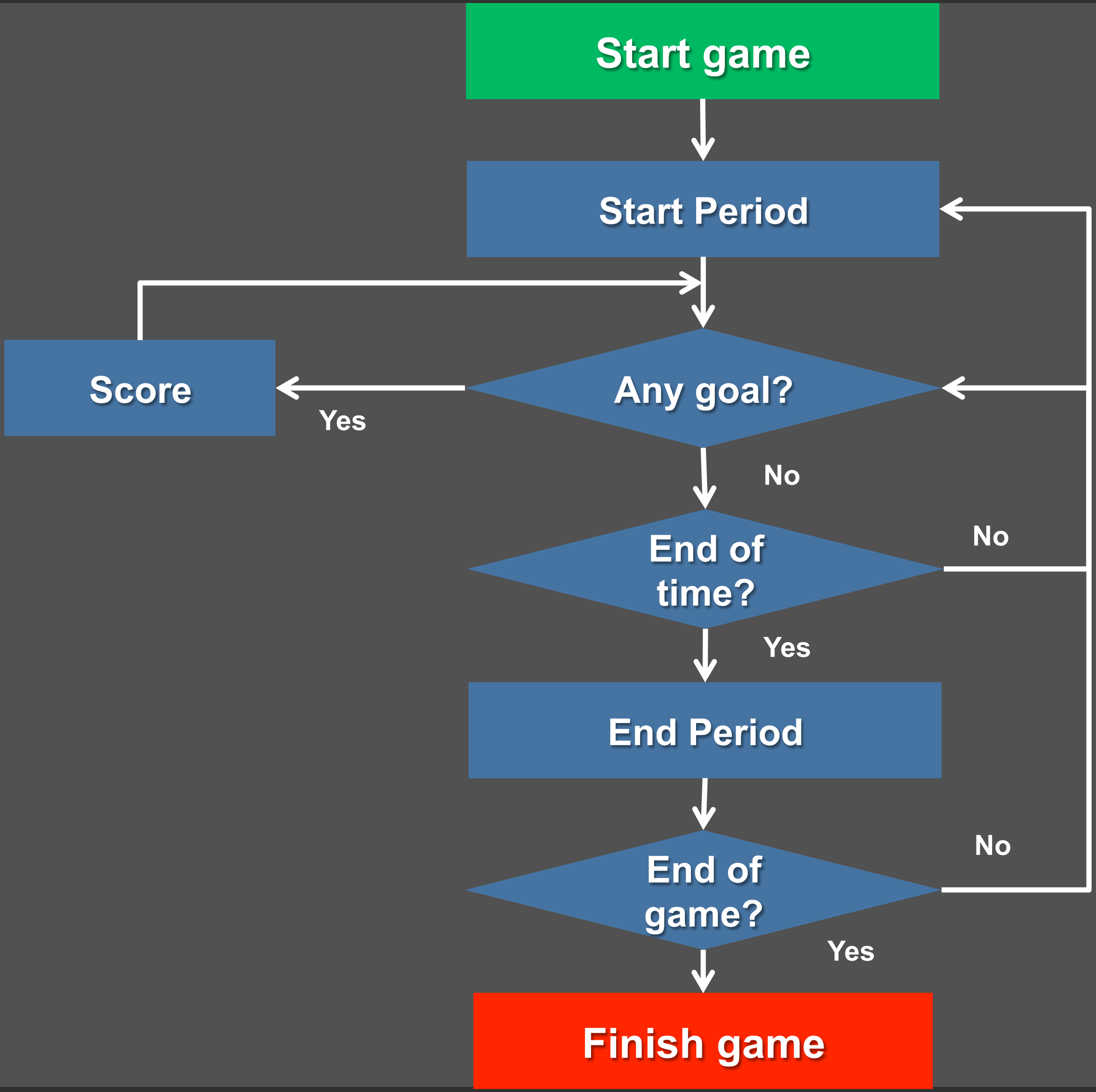
  Domain made of entities, actions on entities, processes and relationships. All together domain expresses behavior finalized to implementing processes.

- **Domain is described in business lingo**

  That must fully reflected in software

Want to see an example?

How would you describe a sport game?


Start game
Start Period
Score
Yes
Any goal?
No
End of time?
No
Yes
End Period
End of game?
No
Yes
Finish game

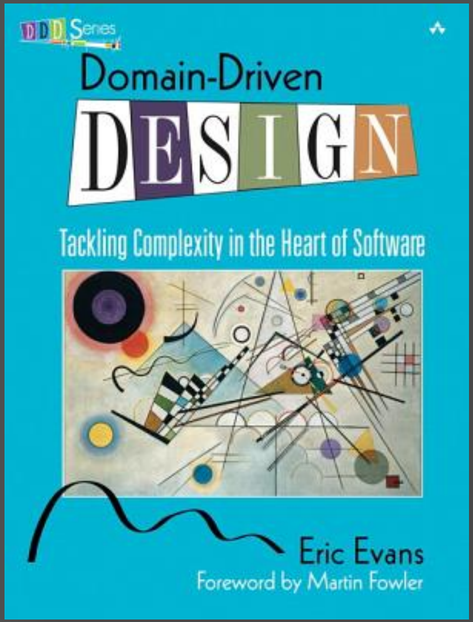
```
public class Match
{
   public string Team1 { get; set; }
   public string Team2 { get; set; }
   public bool IsBallInPlay { get; set; }
   public int TotalGoals1 { get; set; }
   public int TotalGoals2 { get; set; }
   public int MatchState { get; set; }
   public int CurrentPeriod { get; set; }
   public TimeSpan Matchtime { get; set; }
}
```

```
[TestMethod]
public void test_if_score_is_correct()
{
    var match = new Match("12345", "Home", "Visitors");
    match.Start()
         .StartPeriod()
         .Goal(TeamId.Home)
         .Goal(TeamId.Home)
         .EndPeriod()
         .StartPeriod()
         .Goal(TeamId.Visitors)
         .EndPeriod()
         .StartPeriod()
         .EndPeriod()
         .StartPeriod()
         .Goal(TeamId.Home)
         .EndPeriod()
         .Finish();
    Assert.AreEqual(new Score(3, 1), match.CurrentScore);
}
```

Behavior-centric design

DDD – design driven by the domain

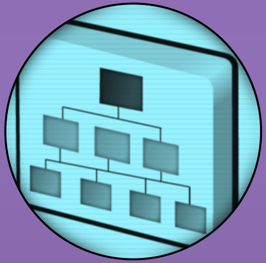
An approach to software design and development introduced 10+ years ago with the intent of …


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

Tackling complexity in the heart of software

- Innovative guidelines
- Design software driven by the domain
- Perceived as *all-or-nothing* approach
- Set of prescriptions

Common Summary of DDD



Build an object model for the business domain

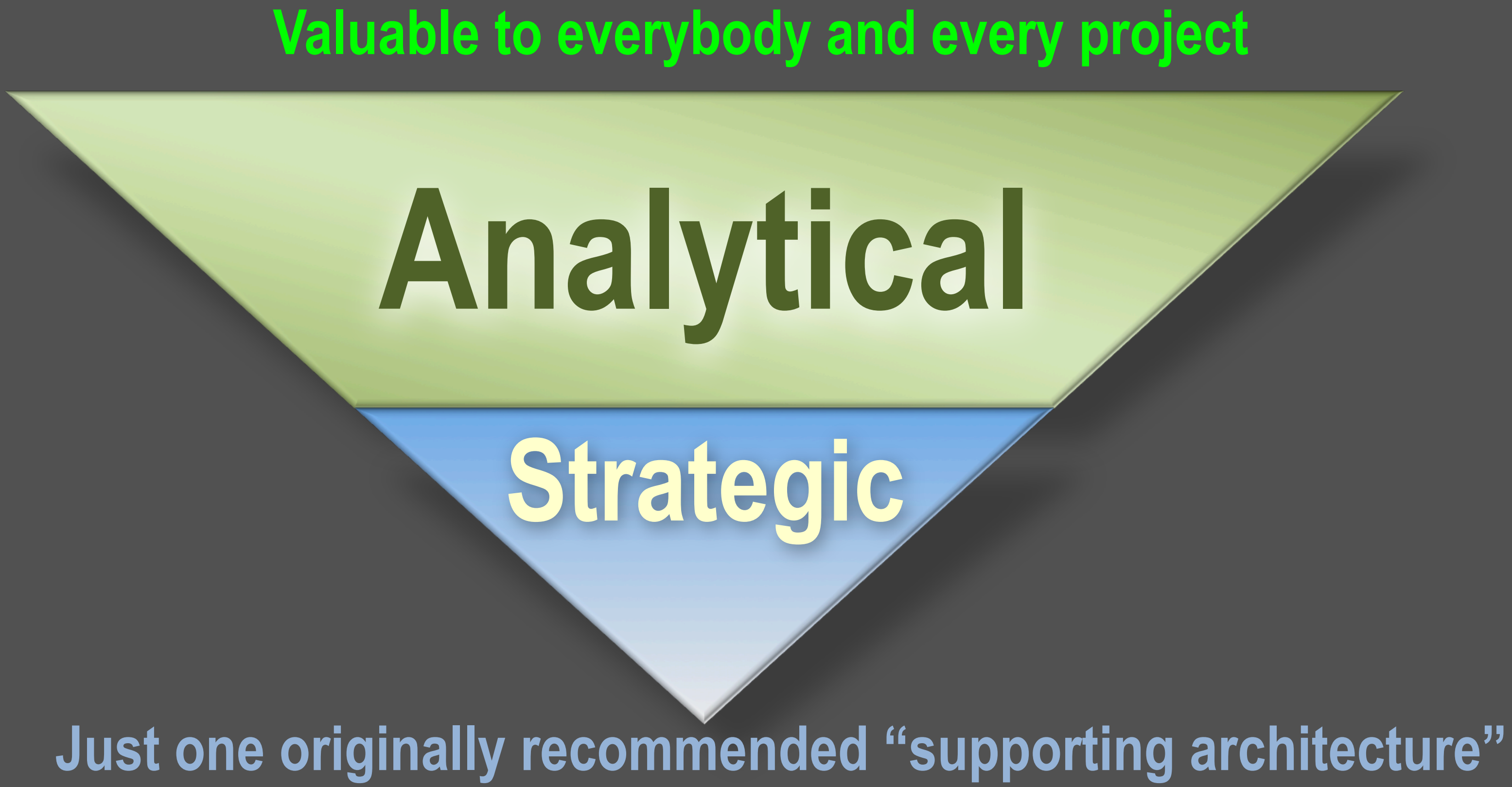
- Call it a "domain model"

Consume the model in a layered architecture

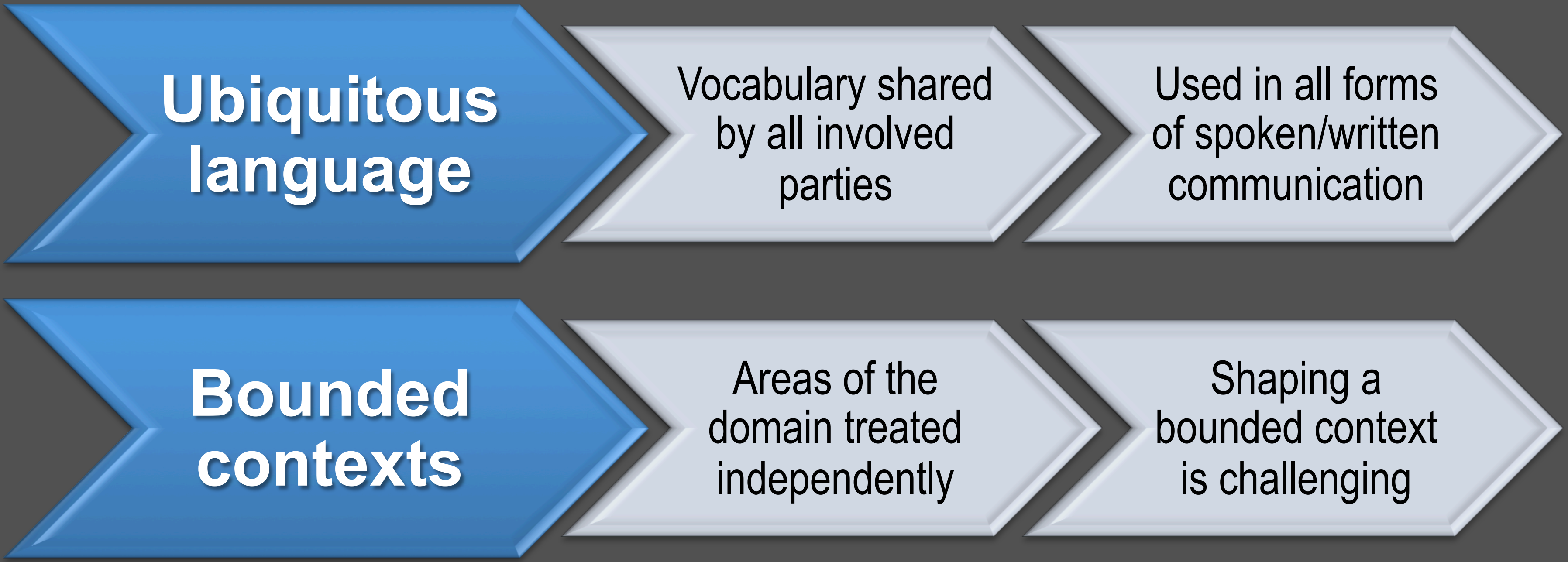
- Four layers
- Business logic split and renamed
- Application layer and Domain layer

Not really hard **to do right; just** easier **to do wrong.**

DDD has **two** distinct parts. You always **need one** and can sometimes happily **ignore** the other.


Valuable to everybody and every project
Analytical
Strategic
Just one originally recommended "supporting architecture"

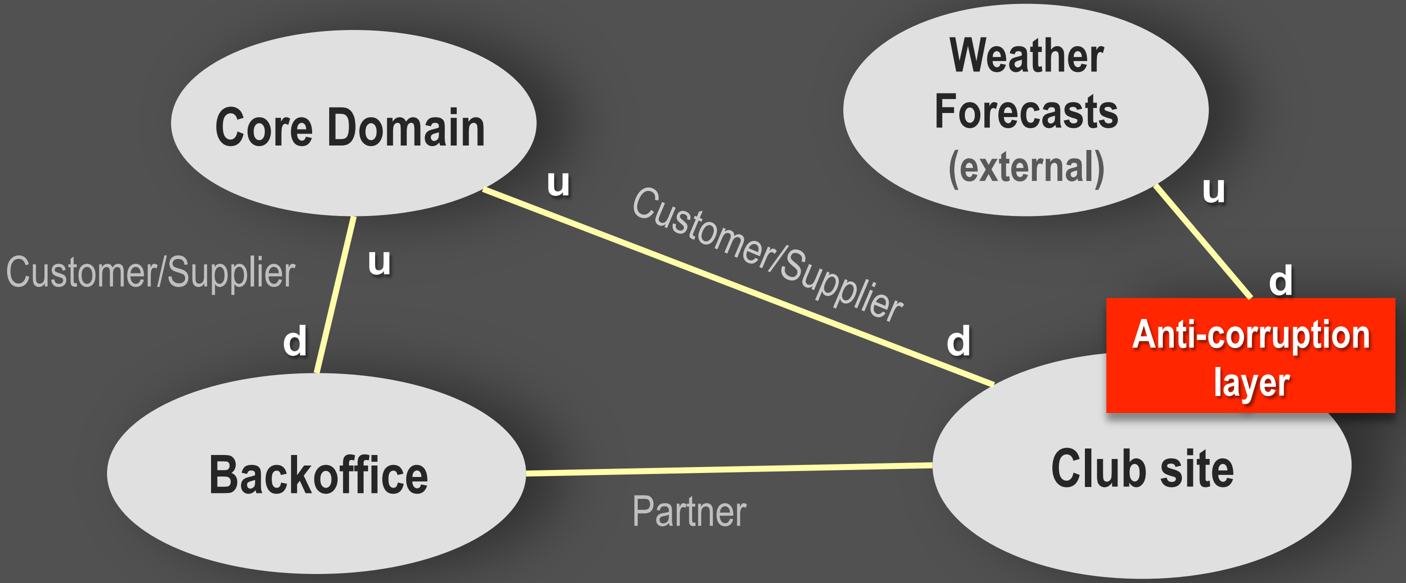
Conducting Analysis Using DDD


Ubiquitous language
Vocabulary shared by all involved parties
Used in all forms of spoken/written communication
Bounded contexts
Areas of the domain treated independently
Shaping a bounded context is challenging

Context Mapping

Bounded contexts often relate to each other

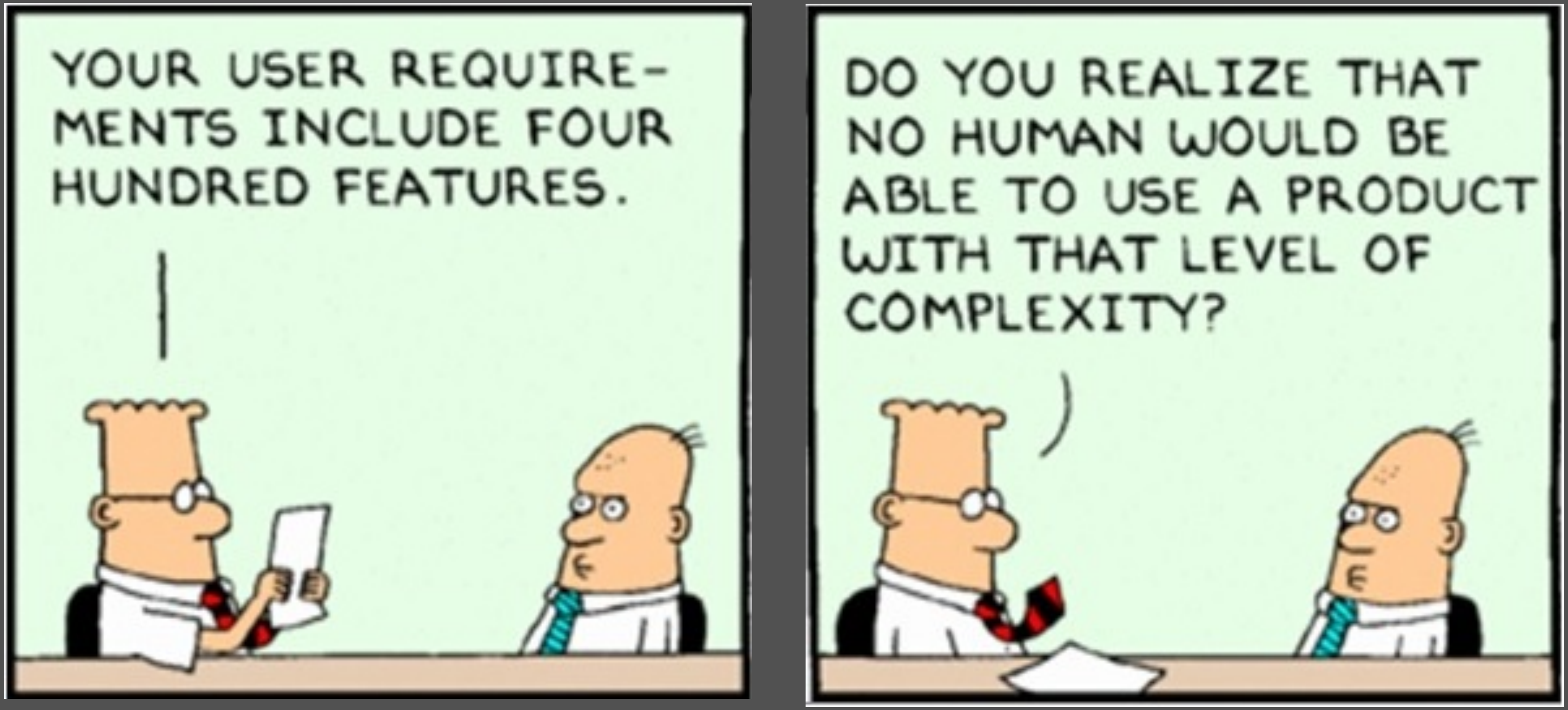
- *Context map* is the diagram that provides a comprehensive view of the system being designed



Supporting Architectures

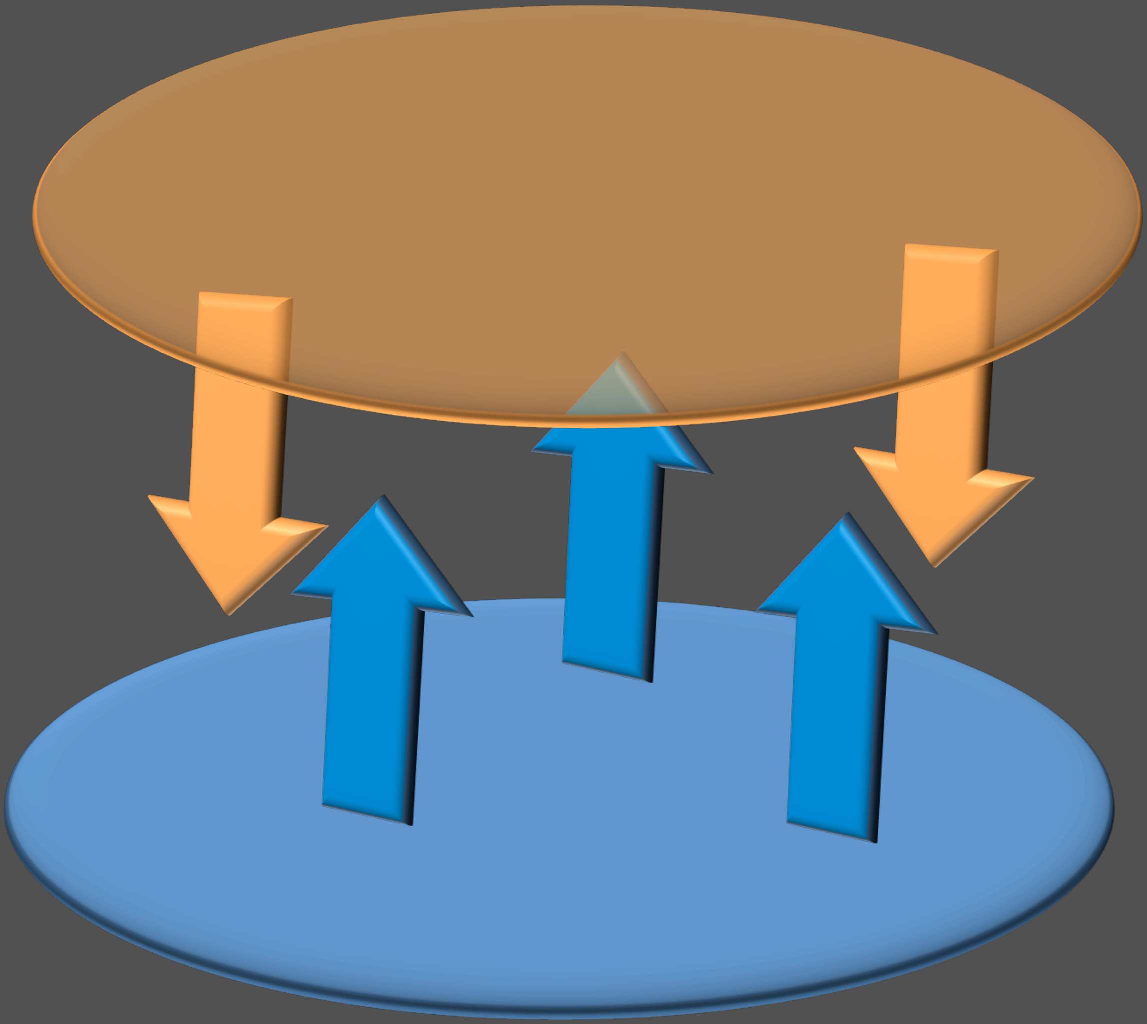
- Multi-layered (-tiered)
- Client/server (2-layer/tier)
- **Domain Model**
  - Object-oriented
  - Functional
- **CQRS**
  - Plain and simple
  - Event bus (+Event Sourcing)

Implementation


YOUR USER REQUIREMENTS INCLUDE FOUR HUNDRED FEATURES.
DO YOU REALIZE THAT NO HUMAN WOULD BE ABLE TO USE A PRODUCT WITH THAT LEVEL OF COMPLEXITY?

www.**dilbert**.com

www.dilbert.com
GOOD POINT. I'D BETTER ADD "EASY TO USE" TO THE LIST.



UX-First

Two distinct architect roles acting together

Software Architect

UX Architect

Interviews to collect requirements and business information to build the domain layer

Interviews to collect usability requirements data and build the ideal UX for the presentation layer

Responsibilities of a UX Expert

- **UI is not UX**
  - Information architecture
  - Interaction and visual design
- **Usability reviews**
  - Observing users live in action (even filming users)
  - Listening to their feedback
- **Catch design/process bottlenecks soon**
  - Focus is data flow, NOT graphics

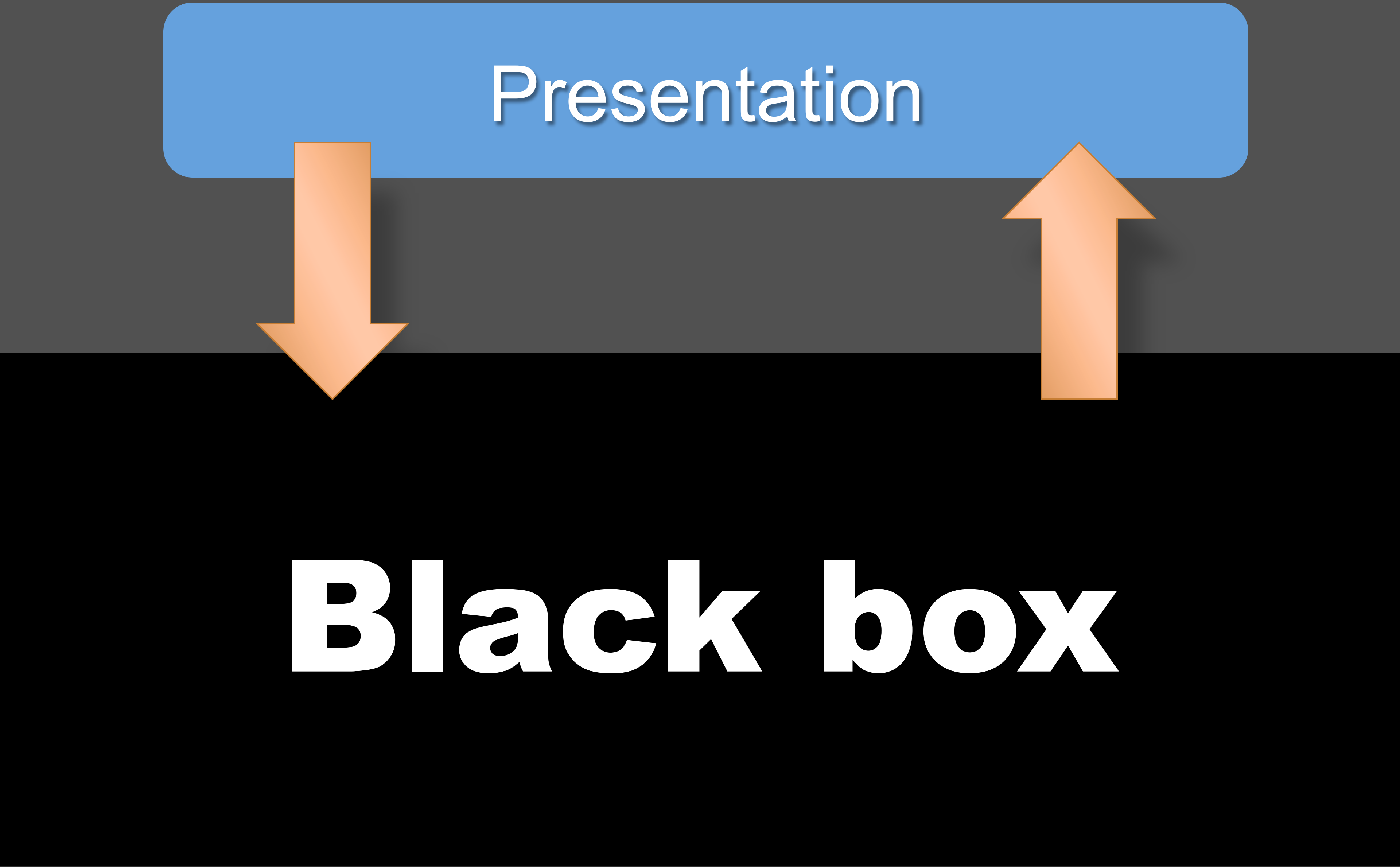
UX-first Design

For each screen have a basic flowchart

1) Determine what comes in and out and create view model classes

2) Make application layer endpoints receive/ return such DTO classes

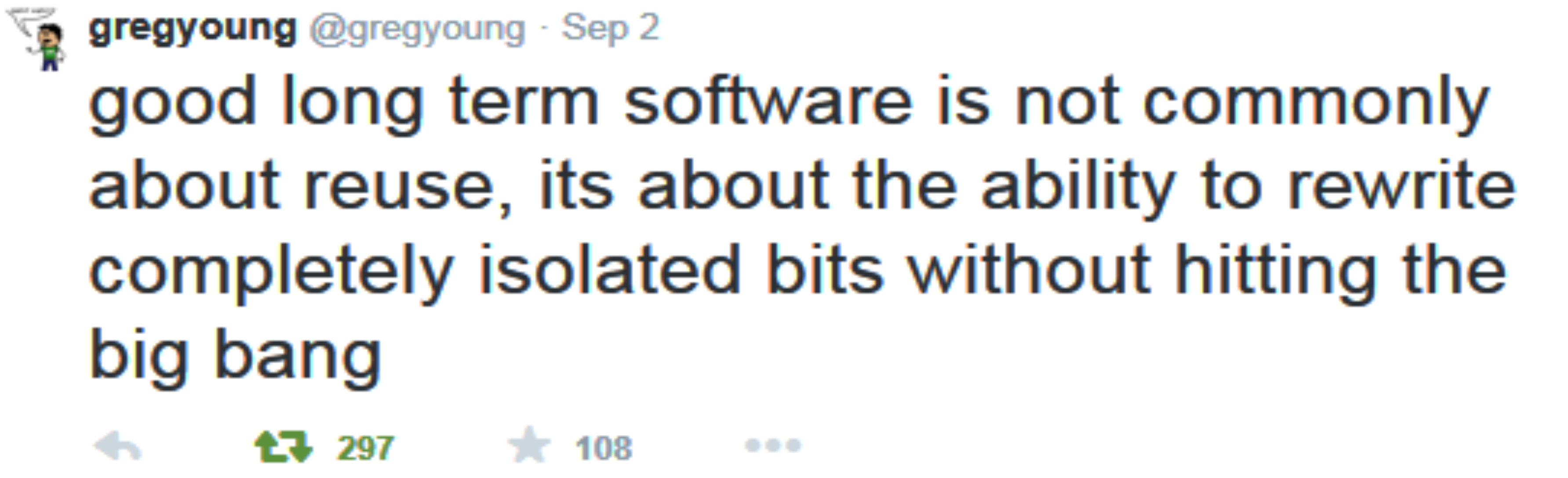
3) Make application layer orchestrate tasks on layers down the stack


Presentation
Black box

Attributes of software

Maintainability

- Is about a wonderful model of tightly chained objects?
- Is about easy-to-replace components?



gregyoung @gregyoung · Sep 2

good long term software is not commonly about reuse, its about the ability to rewrite completely isolated bits without hitting the big bang

297 108

Technology

Gone (forever) are the days in which an upgrade to the next version solved the problem.

Technology today is infrastructure

- Required and critical
- But serving a superior purpose

Serving a superior purpose

Technology is like a medicine

- Can’t be wrong
- But must serve a purpose
- Doc is required
- You’re the doc!

Follow **@despos**

Like **facebook.com/**naa4e