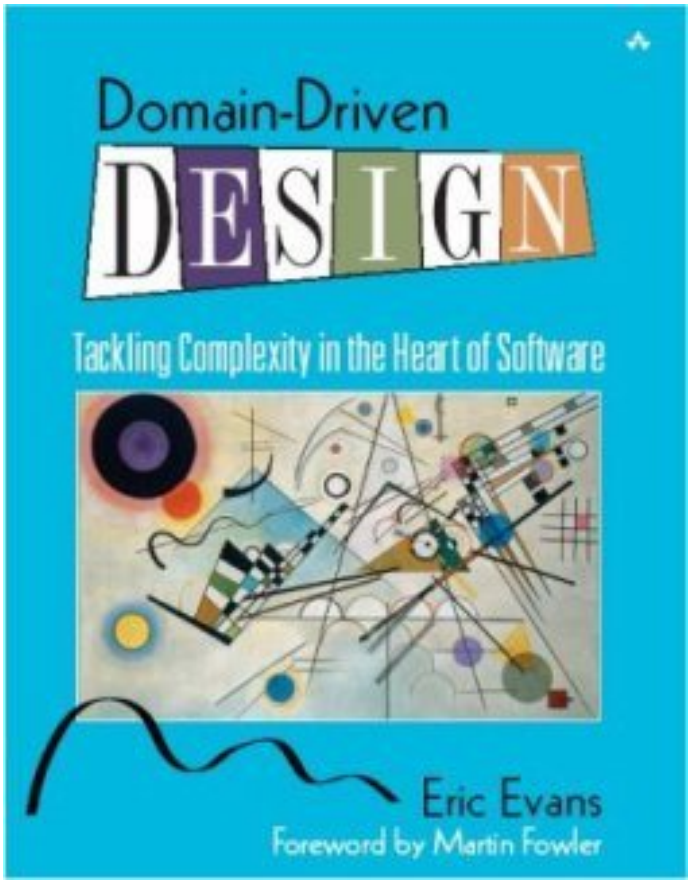
Domain-Driven Design Fundamentals

Introducing DDD

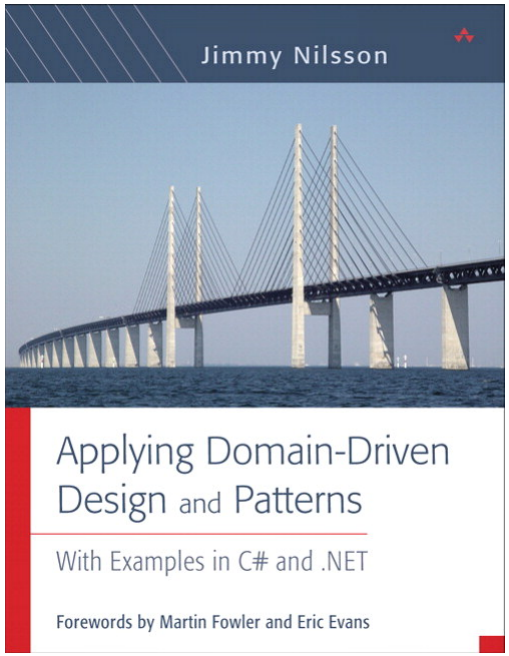
Julie Lerman
TheDataFarm.com
@julielerman

Steve Smith
Ardalis.com
@ardalis

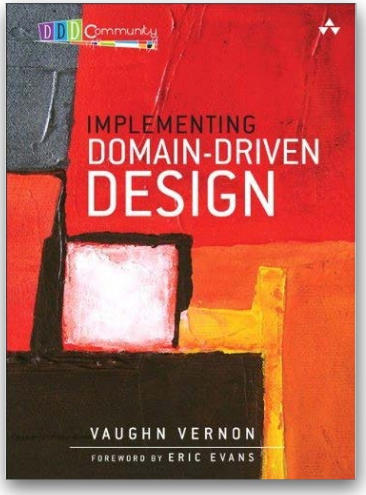
pluralsight
hardcore developer training



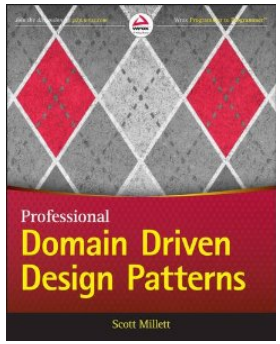
2003



2006



2013



2014

Do Not Place Anything in This Space
(Add watermark during editing)
Note: Warning will not

In This Course

- Why Should You Care About DDD?
- What does a DDD solution look like?
- Modeling Problems in Software
- Technical Components of DDD
- Managing Complexity using DDD
- Extending our Example *(based on a client request)*

“

The more you know,
the more you realize you know nothing

”

— Socrates

Why You Should Care about DDD

Principles & patterns to **solve difficult problems**

History of **success** with complex projects

Aligns with practices from our own **experience**

Clear, testable code that represents the domain

Solve Problems

Understand Client Needs

Write Code

Note: Warning will not

Engineering

Purchase Materials

Accounting

Sales

Manage Employees

Marketing

- Note: Warning will not

Benefits of Domain Driven Design

- Flexible
- Customer's vision/perspective of the problem
- Path through a very complex problem
- Well-organized and easily tested code
- Business logic lives in one place.
- Many great patterns to leverage

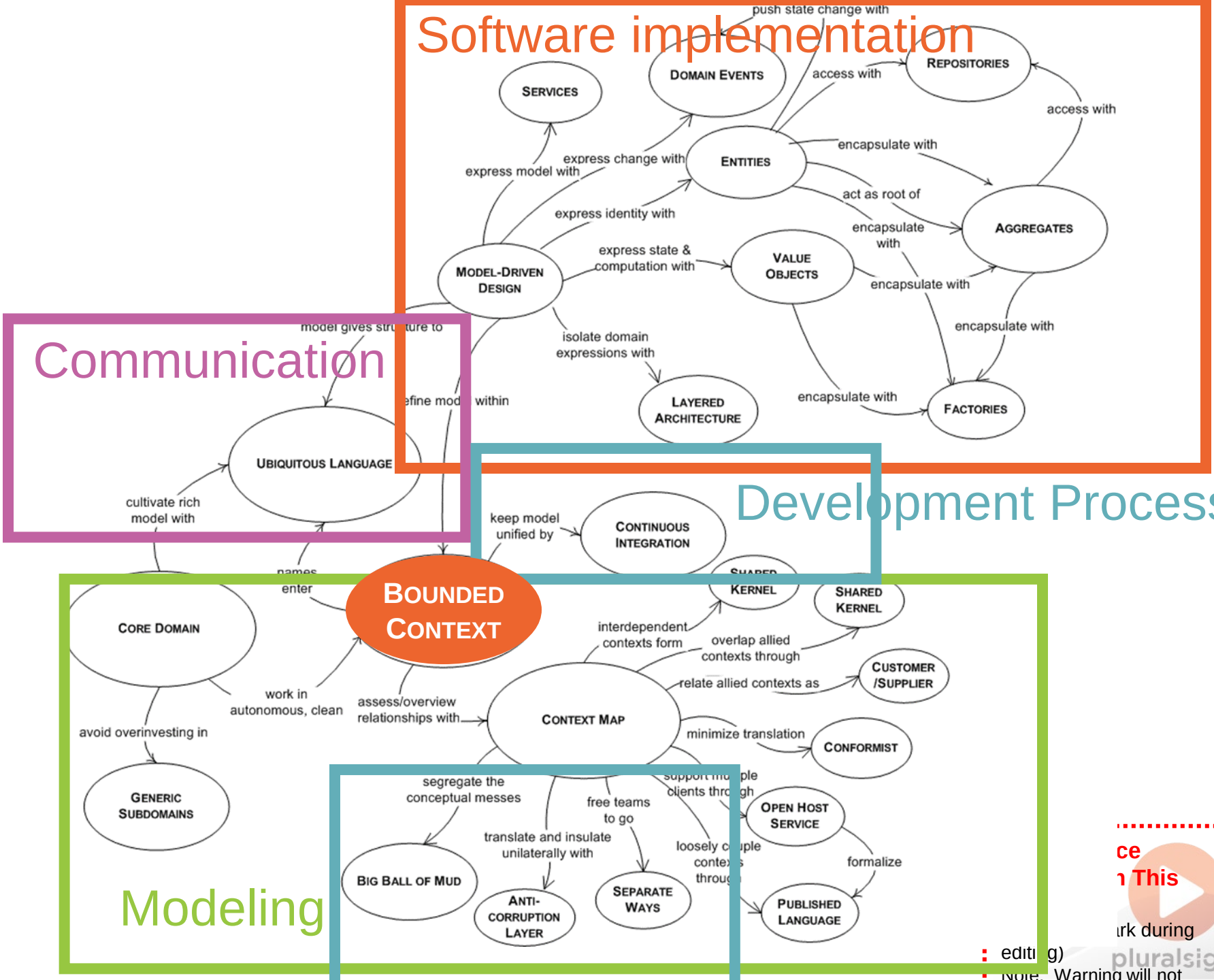
> While Domain-Driven Design provides many technical benefits, such as maintainability, it should be applied only to complex domains where the model and the linguistic processes provide clear benefits in the communication of complex information, and in the formulation of a common understanding of the domain.

— Eric Evans

Domain-Driven Design

Drawbacks of DDD

- **Time and Effort**
  - Discuss & model the problem with domain experts
  - Isolate domain logic from other parts of application
- **Learning curve (why you're watching this course)**
  - New principles
  - New patterns
  - New processes
- **Only makes sense when there is complexity in the problem**
  - Not just CRUD or data-driven applications
  - Not just technical complexity without business domain complexity
- **Team or Company Buy-In to DDD**


Software implementation
push state change with
SERVICES
DOMAIN EVENTS
access with
REPOSITORIES
access with
express model with
express change with
ENTITIES
encapsulate with
express identity with
act as root of
AGGREGATES
encapsulate with
express state & computation with
MODEL-DRIVEN DESIGN
VALUE OBJECTS
encapsulate with
encapsulate with
isolate domain expressions with
LAYERED ARCHITECTURE
encapsulate with
FACTORIES
Communication
model gives structure to
UBIQUITOUS LANGUAGE
cultivate rich model with
Development Process
keep model unified by
CONTINUOUS INTEGRATION
BOUNDED CONTEXT
CORE DOMAIN
work in autonomous, clean
assess/overview relationships with
CONTEXT MAP
interdependent contexts form
SHARED KERNEL
overlap allied contexts through
SHARED KERNEL
relate allied contexts as
CUSTOMER /SUPPLIER
minimize translation
CONFORMIST
avoid overinvesting in
GENERIC SUBDOMAINS
segregate the conceptual messes
free teams to go
translate and insulate unilaterally with
BIG BALL OF MUD
ANTI-CORRUPTION LAYER
SEPARATE WAYS
OPEN HOST SERVICE
formalize
PUBLISHED LANGUAGE
Modeling

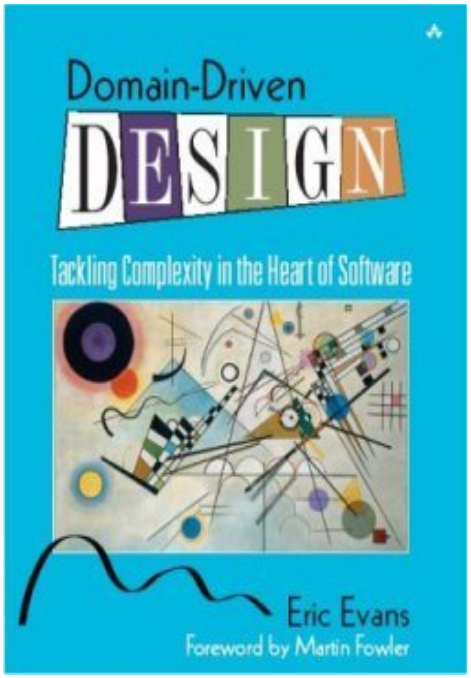
Demo

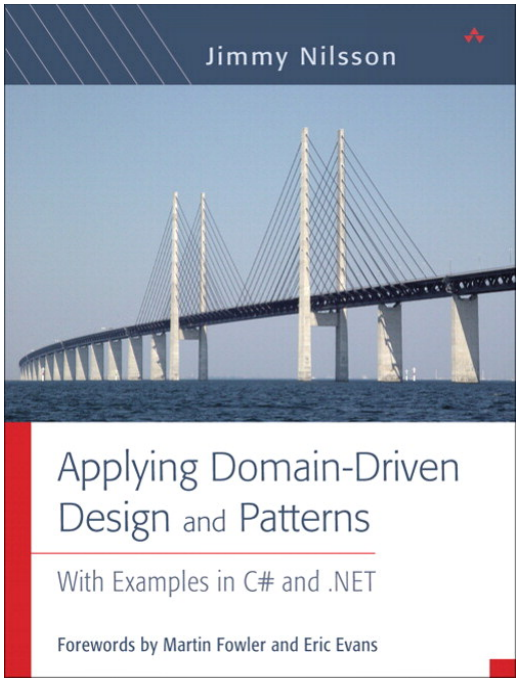
A small DDD Solution

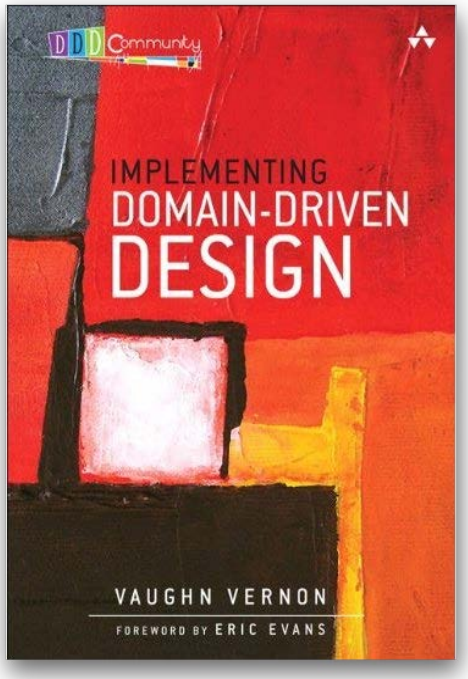
Solve Problems

Understand client needs

References







- **DDDCommunity.org**

On Pluralsight:

- **Creating N-Tier Applications in C#, Part 1 bit.ly/ntier-smith**

Thanks!

Julie Lerman

TheDataFarm.com

Twitter: @julielerman

Steve Smith

Ardalis.com

Twitter: @ardalis

To Teach Is To Learn Twice