_

Domain-Driven Design (DDD)

Agenda

1. INTRODUCING DDD
2. MODELING PROBLEMS IN SOFTWARE
3. ELEMENTS OF A DOMAIN MODEL
4. UNDERSTANDING VALUE OBJECTS & SERVICES IN THE MODEL
5. TALKING COMPLEXITY WITH AGGREGATES
6. WORKING WITH REPOSITORIES
7. DOMAIN EVENTS AND ANTI-CORRUPTION LAYERS
8. RESOURCES

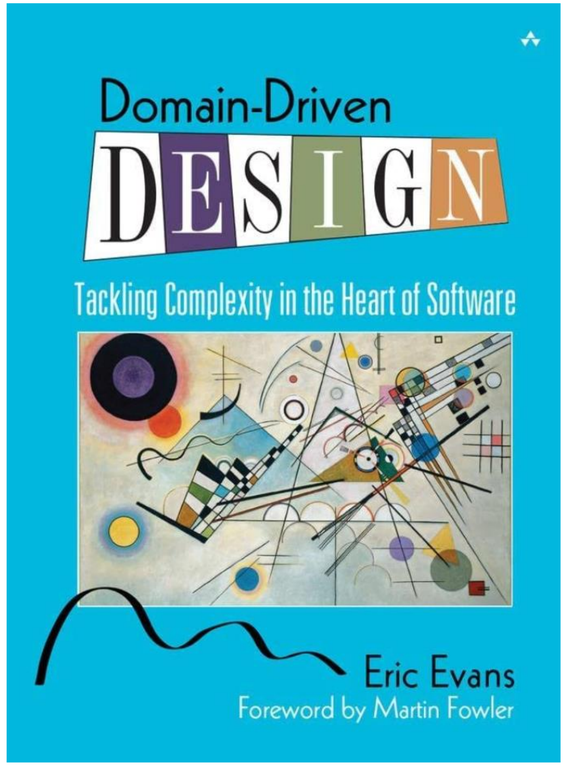
1

Introducing DDD

DDD History

Beginnings

Eric Evans 2003



Now

Many new DDD books

- **Applying Domain-Driven Design and Patterns, Jimmy Nilsson (2006)**
- **Implementing Domain-Driven Design,** Vaughn Vernon
- **Professional Domain-Driven Design Patterns,** Scott Millett, Nick Tune
- **Domain-Driven Design Distilled,** Vaughn Vernon
- **Domain Modeling Made Functional,** Scott Wlaschin
- **Hands-on Domain-Driven Design with .NET Core,** Alexey Zimarev

Many conferences

- **Domain Driven Design Europe (Amsterdam)**
- **KanDDDinsky (Berlin)**
- **Explore DDD Conference (Colorado)**
- **DDD Exchange from Skills Matter (London)**
- **Virtual DDD Meetup (on the web)**

What is DDD?

Domain-Driven Design is an approach to software development that centers the development on programming a domain model that has rich understanding of the processes and rules of a domain.

DDD Core Themes

1. Interaction with domain experts
2. Model a single subdomain at a time
3. Implementation of subdomains

DDD is Complex

DDD is a really complex topic, this is just an introduction.

Also, DDD is for solving COMPLEX problems.

DDD isn't suitable for problems when there is little business domain complexity.

DDD Mind Map


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

DDD Prerequisites

1. Development is iterative. (the agile development methods are good fit.)
2. Developers and domain experts have a close relationship. (will be explained later)

2

Modeling Problems in Software

Domain Expert

These are the people who live and breed the business or process or whatever you are targeting with the software you're planning to write.

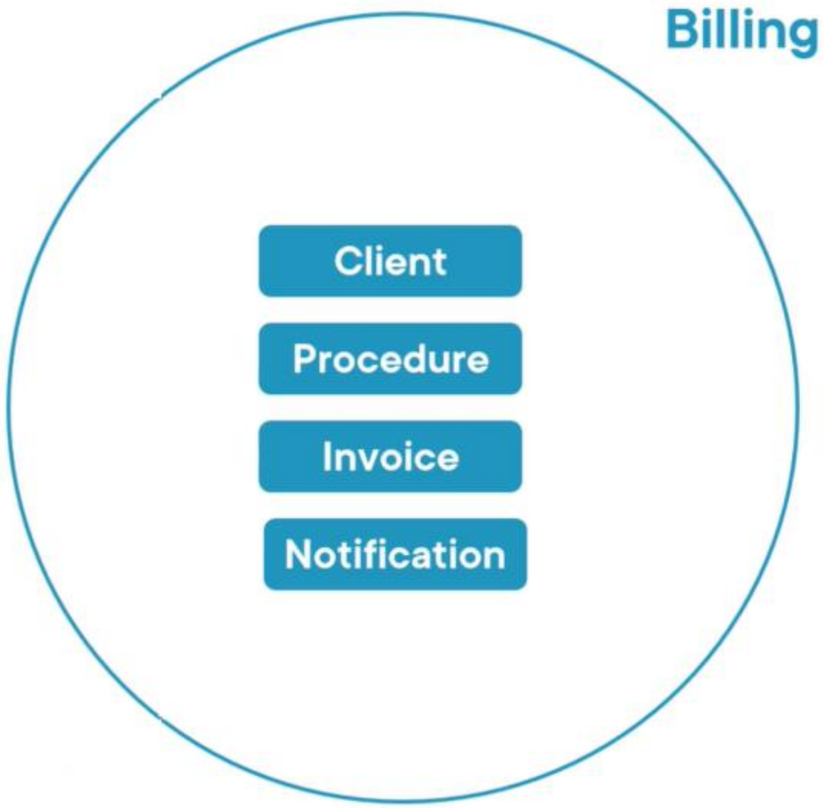
Invest in your relationship with the domain expert.

Important to get on the "same page" with the domain expert.

Learn and communicate in the language of domain, not the language of technology.

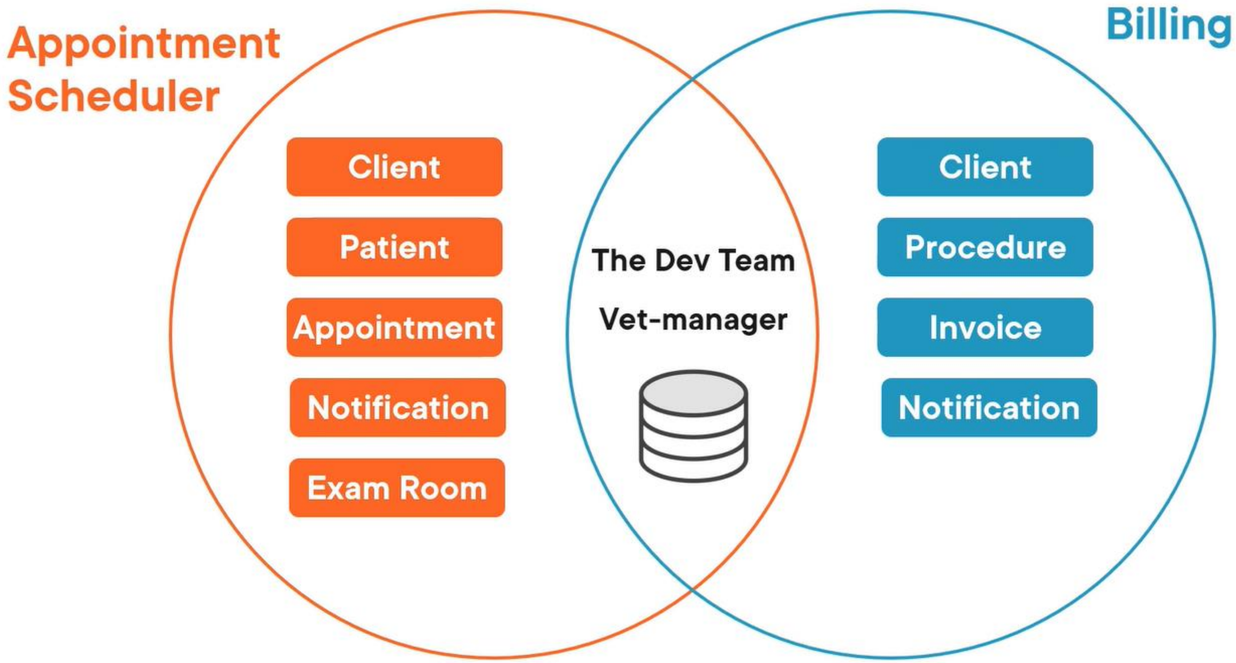
Bounded Context

A bounded context is simply the boundary within a domain where a particular domain model applies.



Context Map

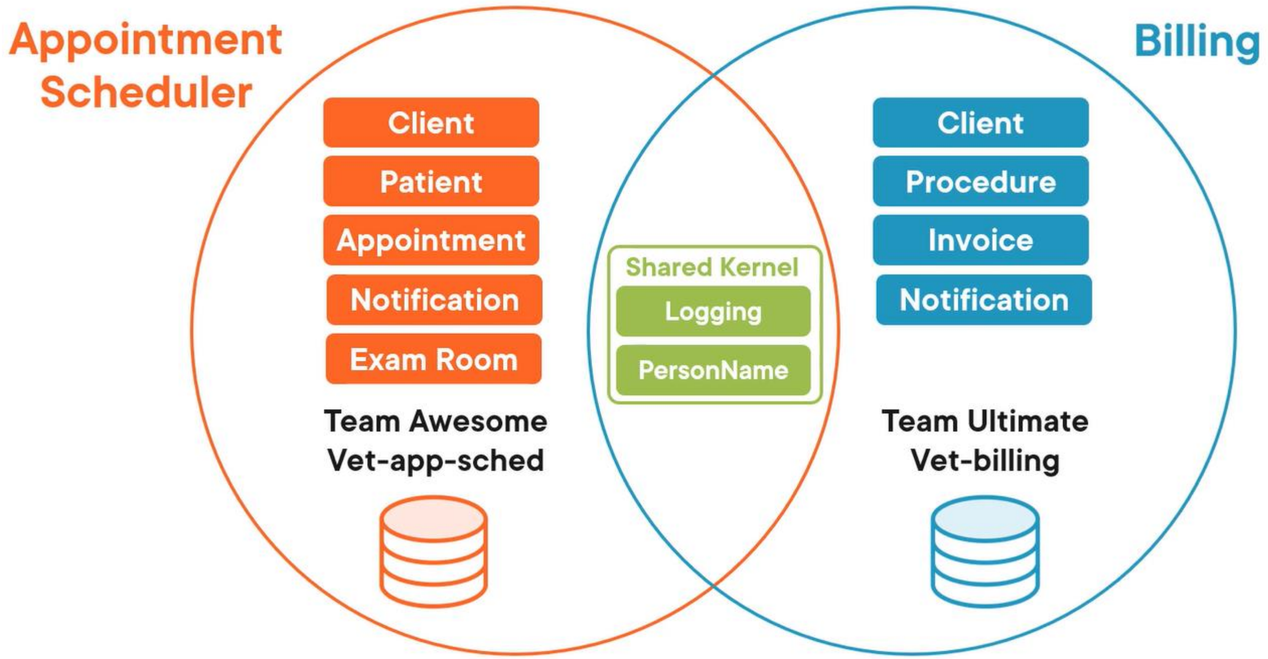
Demonstrates how bounded contexts connect to one another while supporting communication between teams.



Shared Kernel

Bounded contexts have shared concepts or resources - this is called shared kernel.

The goal is to reduce duplication.



Ubiquitous Language

A simple definition of a ubiquitous language is to come up with terms that'll be commonly used when discussing a particular subdomain.

And they will most likely be terms that come from the PROBLEM SPACE, NOT THE SOFTWARE WORLD!

3

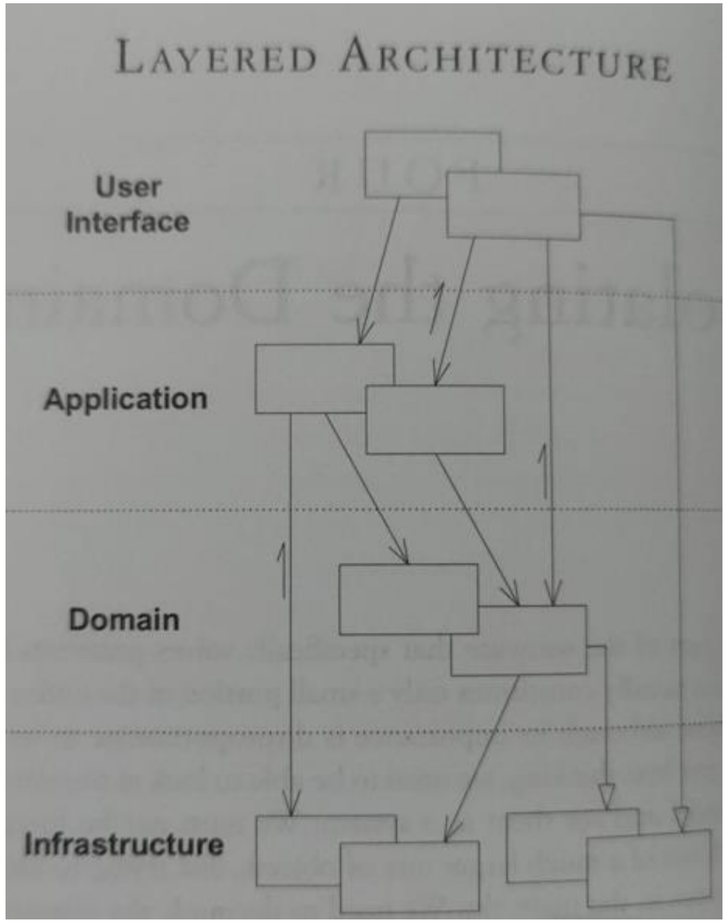
Elements of a Domain Model

Domain Layer

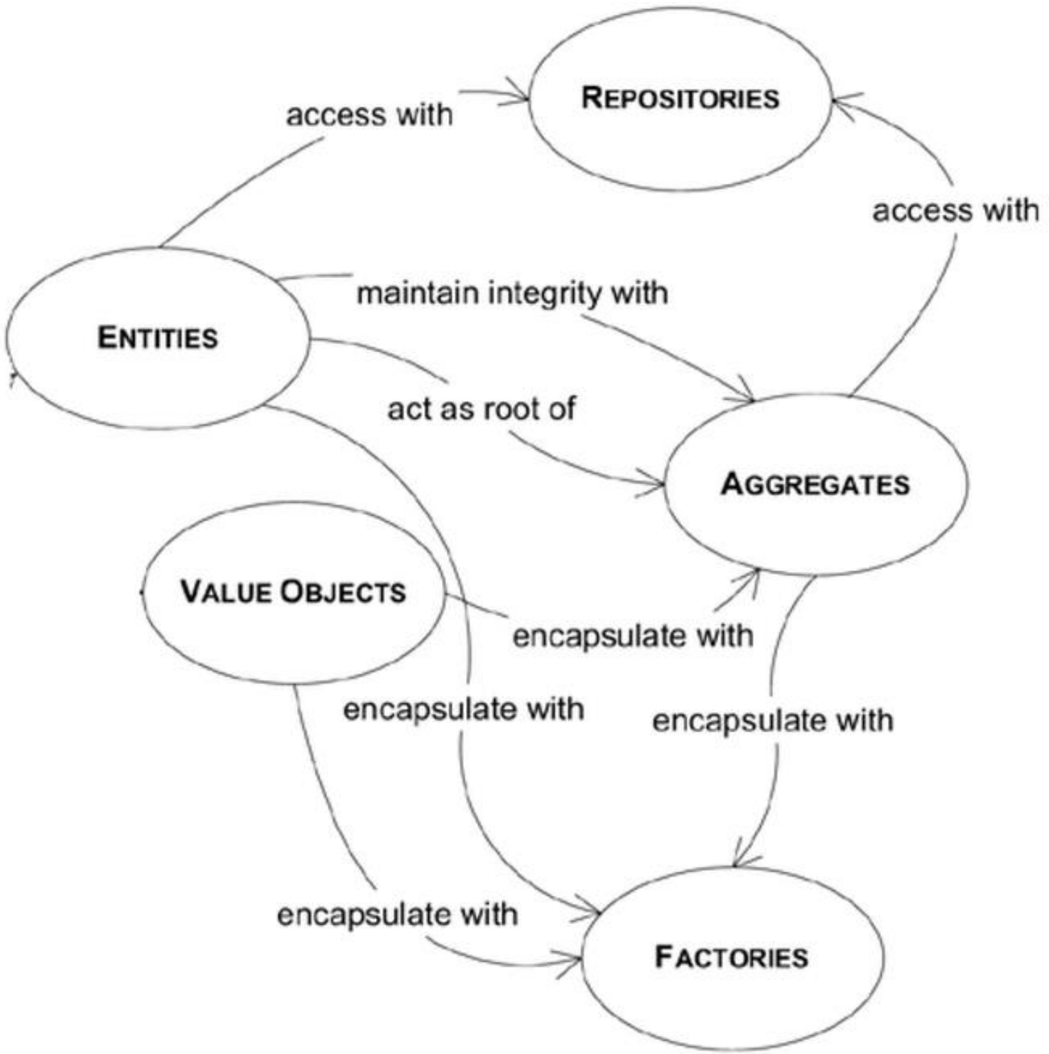
The domain is the hart of business software.

The Domain Layer is responsible for representing concepts of the business, information about the business situation, and business rules.

We're isolating the domain using layered architecture.



Elements of Domain model



Entity

An entity is something we need to be able to track, locate, retrieve, and store, and we do that with an identity key.

Entity is a mutable class with an identity.

DDD Terms

Some of the issues are that the DDD terms overlap with technology terms. Most of the terms are really different.

	Entity Framework Core	Domain-Driven Design
Entity	A data model class with a key that is mapped to a table in a database	A domain class with an identity for tracking
Context	A DbContext class provides access to entities and defines how entities map to the database	A Bounded Context defines the scope and boundaries of a subset of a domain

4

Understanding Value Objects & Services in the Model

Value Object

An object that represents a descriptive aspect of the domain with no conceptual identity is called a VALUE OBJECT.

Value objects:
- Measures, quantifies, or describes a thing in the domain
- Identity is based on composition of values
- Immutable
- Compared using all values
- No side effect

Examples: string, datetime, money

Domain Service

Provide a place in the model to hold behavior that doesn't belong elsewhere in that domain.

A service is often appropriate for operation that doesn't belong in entity or value object.

4

Talking Complexity with Aggregates

Aggregates

An aggregate is a cluster of associated objects (Entities and Value Objects) that we treat as a unit for the purpose of data changes.

Any object internal to an aggregate is prohibited from access except by traversal from the root.

Aggregate Root

Every aggregate has a root, the root should be an entity.

Choose one entity to be the root of each aggregate, and control all access to the objects inside the boundary through the root.

4

Working with Repositories

Repository

An abstraction the domain model uses to define what persistence needs it has.

General repository tips
1. Use repositories for aggregate roots only.
2. Keep the clients focused on model, while delegating all of the object storage and access concerns to the repositories.

Specification

Specification is a method of encapsulating a business rule so that it can be passed to other methods which are responsible for applying it.

4

Domain Events and Anti-corruption Layers

Domain Events

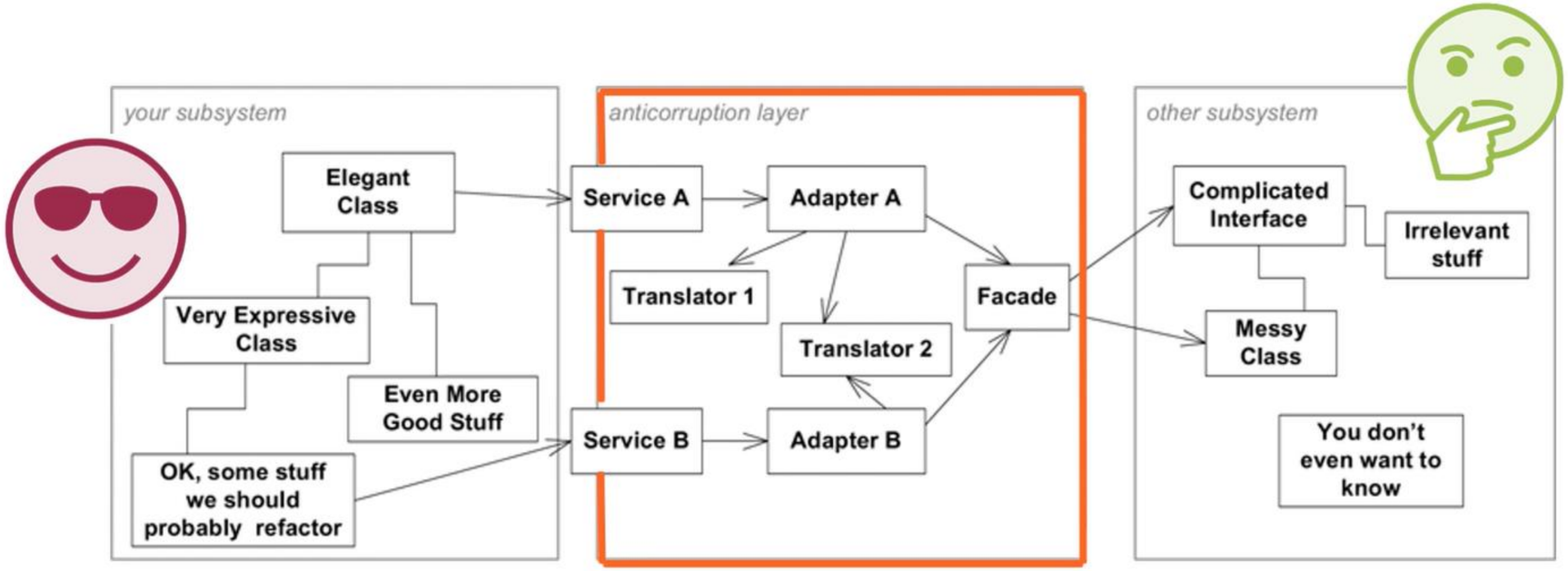
Domain events alert that some activity occurred, or some state changed in the context and their domains can subscribe to the "news".

Domain events features:
- Can communicate outside of the domain
- Encapsulated as objects
- Each event is full-fledged class

Anti-corruption layer

Anti-corruption layers helps to prevent corruption in the domain model.

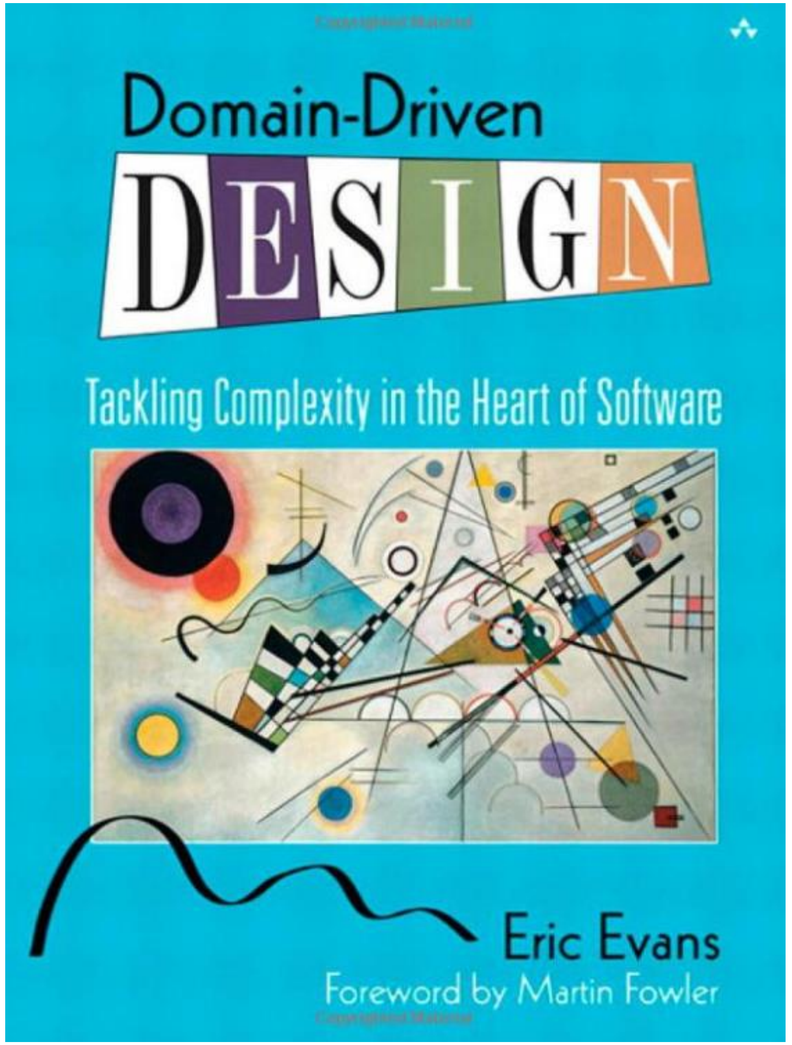
This layer translates the communication between layers, with this we're stopping corruption in our models.



6

Resources

Domain-Driven Design: Tackling Complexity in the Heart of Software



Other resources

As mentioned in the beginning there are many newer books about DDD.

Courses:
Domain-Driven Design Fundamentals (new version, 2021)
https://app.pluralsight.com/library/courses/fundamentals-domain-driven-design/table-of-contents
Domain-Driven Design Fundamentals (old version, from the same authors, 2014)
https://app.pluralsight.com/library/courses/domain-driven-design-fundamentals/table-of-contents
Domain-Driven Design in Practice
https://app.pluralsight.com/library/courses/domain-driven-design-in-practice/table-of-contents
Domain-Driven Design: Working with Legacy Projects
https://app.pluralsight.com/library/courses/domain-driven-design-legacy-projects/table-of-contents
Domain-Driven Design Path
https://app.pluralsight.com/paths/skill/domain-driven-design

Code repo:
https://github.com/ardalis/pluralsight-ddd-fundamentals

Links:
https://www.dddcommunity.org
https://www.domainlanguage.com


endava