pattern

DDD with Ruby

Sandesh Bodke (Software Engineer @Pattern)

Agenda

01.

What and Why DDD?

02.

Deep dive into DDD

03.

DDD <> Hexagonal architecture

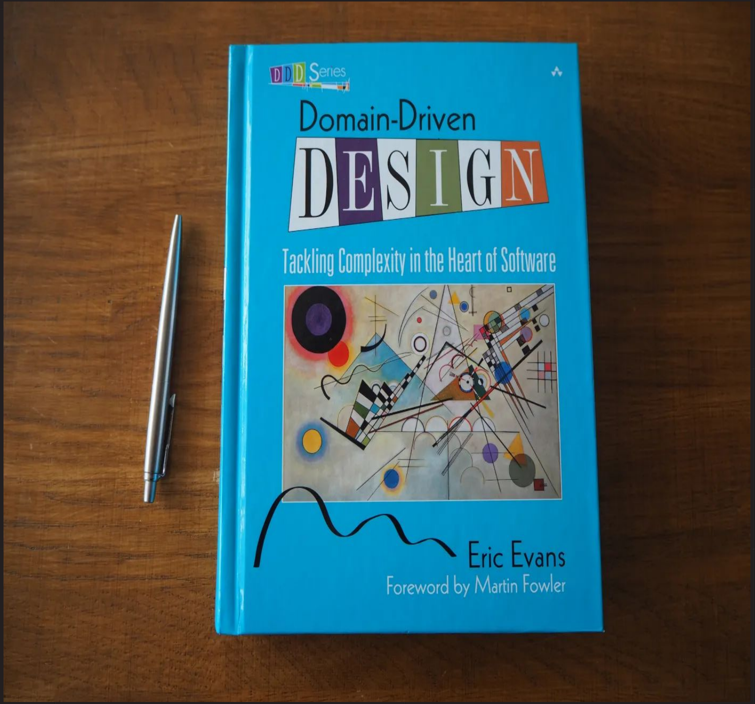
04.

Code walkthrough

What is DDD?

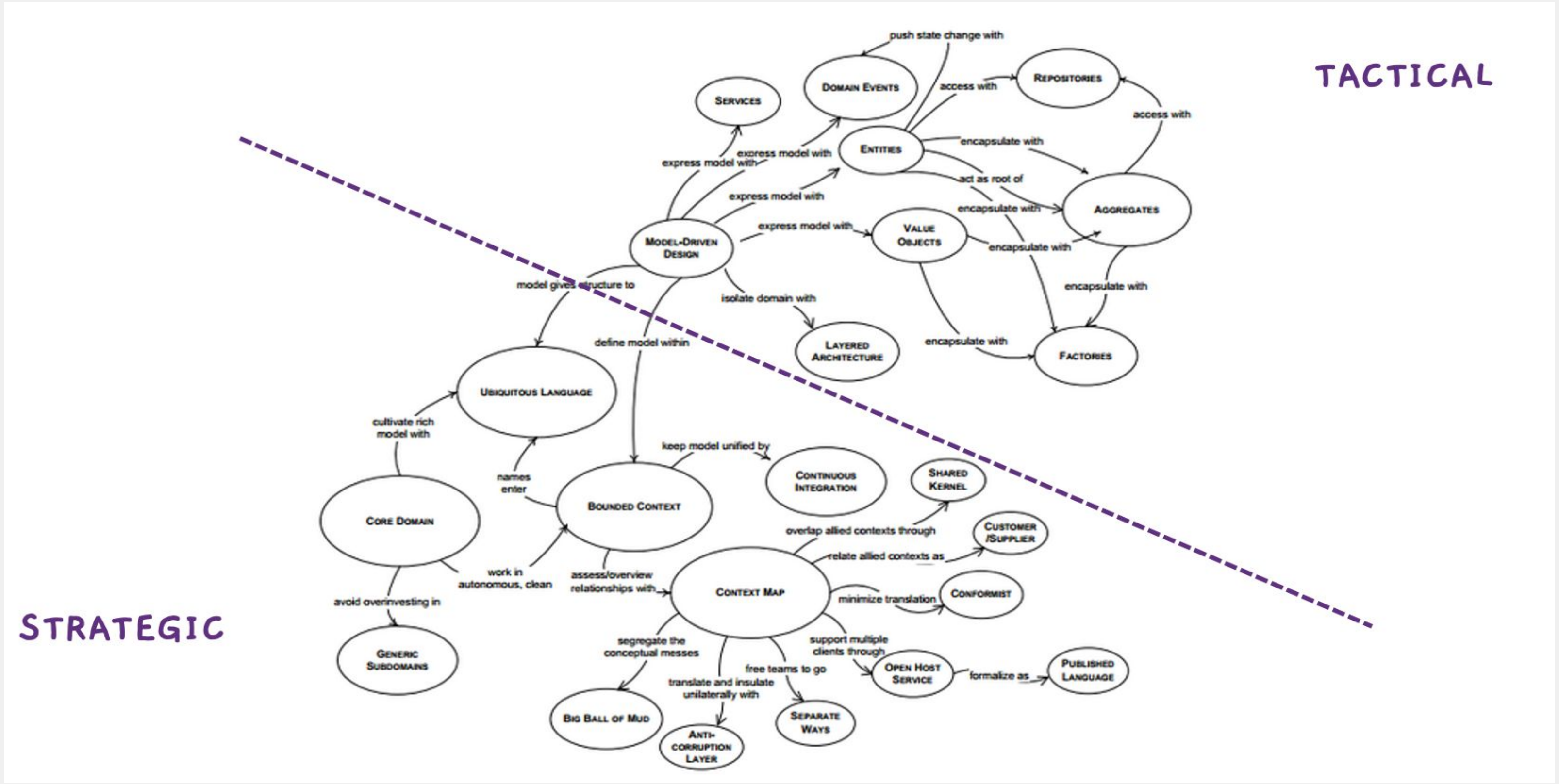
A software design is driven by Domain.

Domain is a subject area around which the application that is being developed is centered.


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

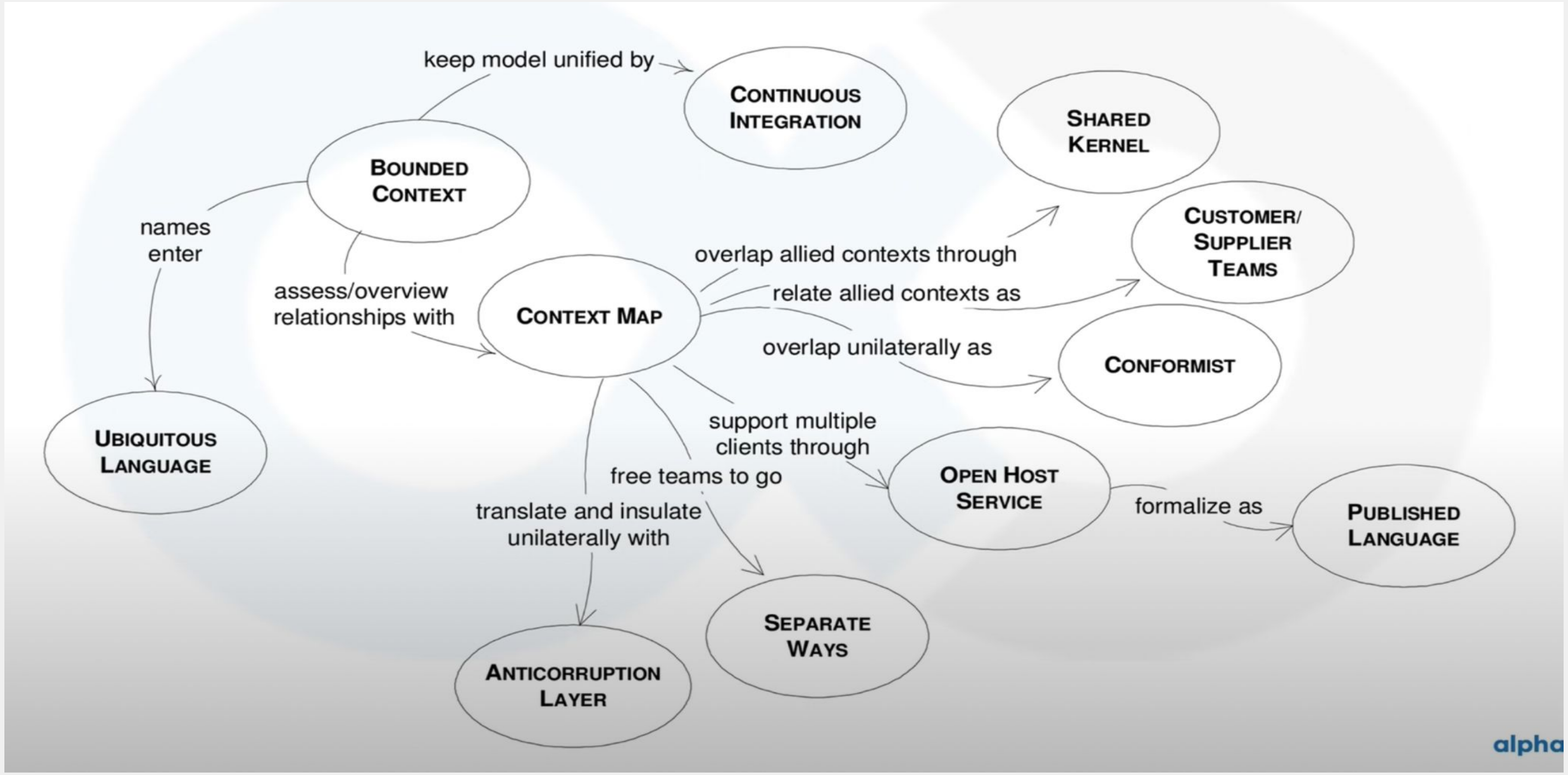
pattern

What DDD gives us?



pattern

Strategic Design



Strategic Design

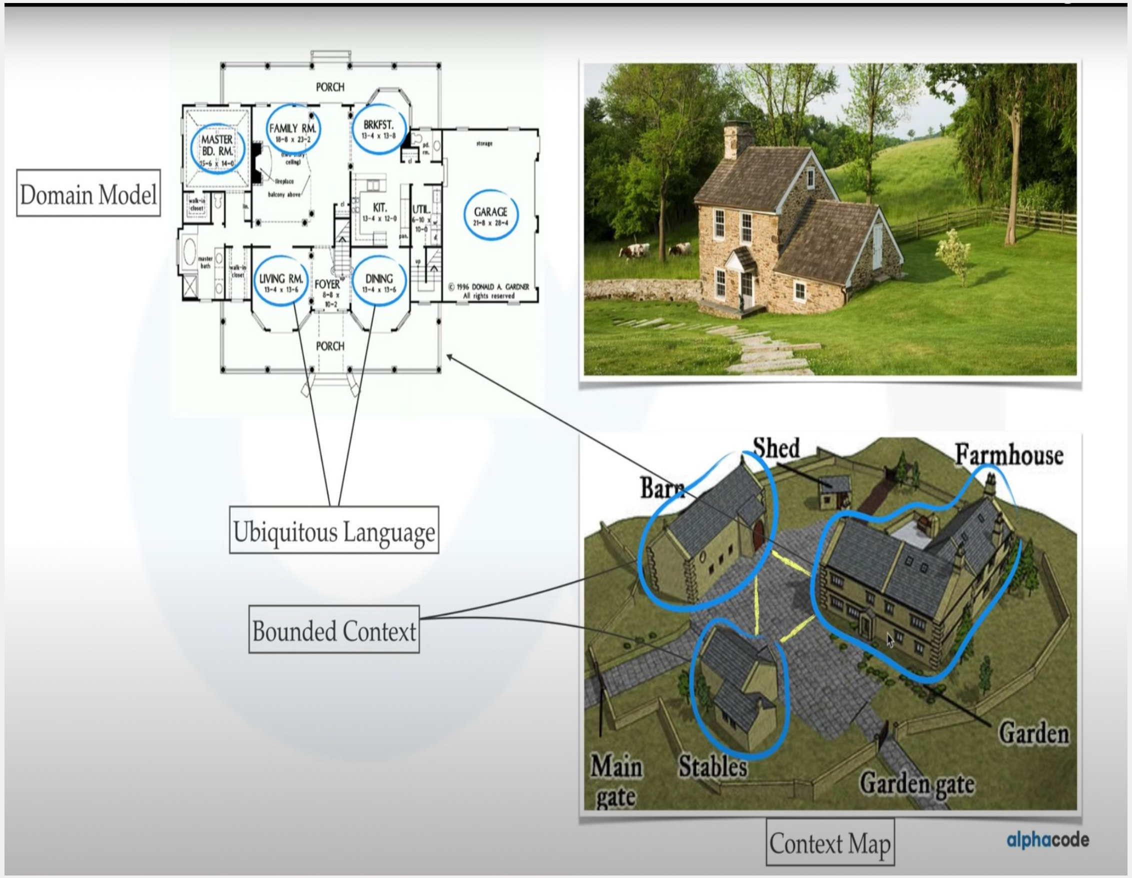
Build a house

What kind of house?

You'll talk to domain expert

You'll try to find out core values

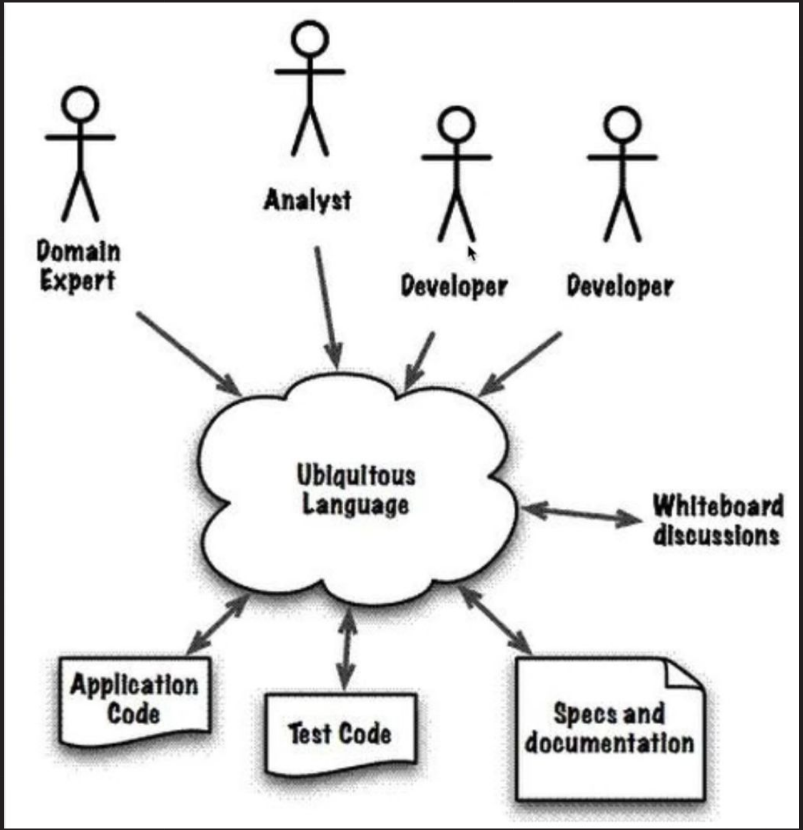
You'll see what other have done



Ubiquitous Language

Language shared by domain experts and development team

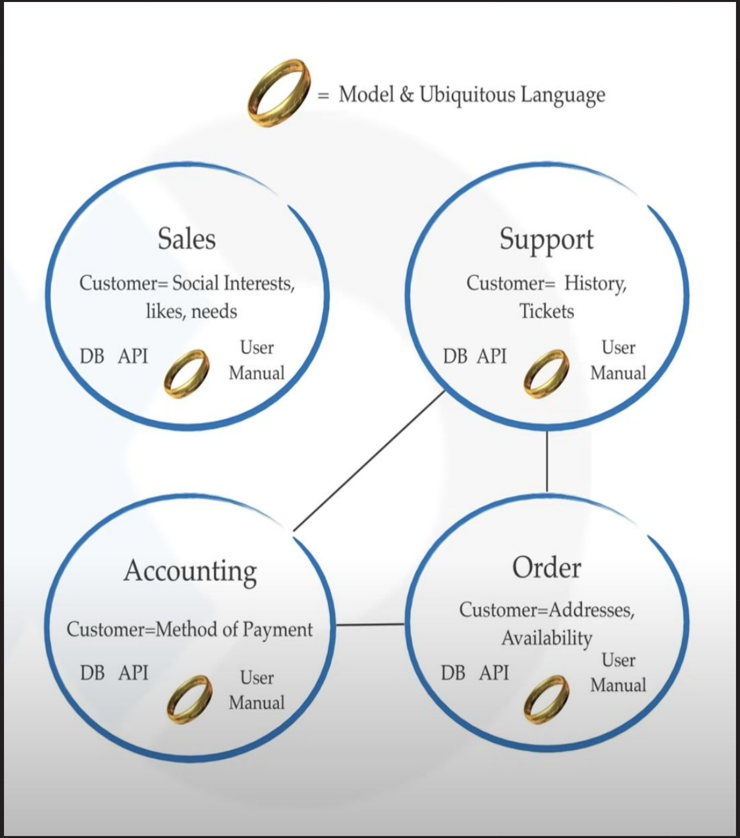
Language use in the discussion, in the domain model, in the code of application, in the classes, in the methods



Bounded Context

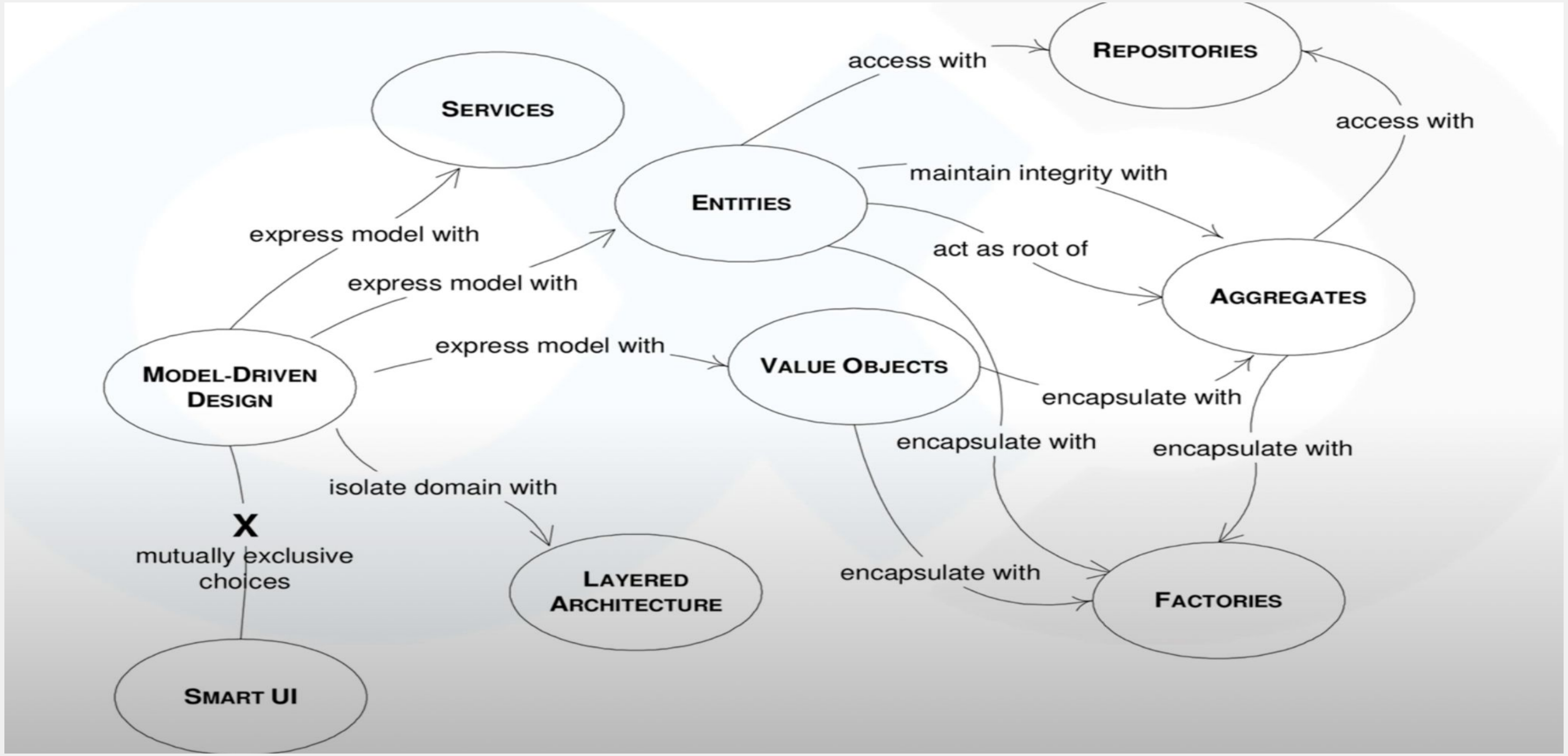
To deal with large domains, you can divide the model , you have into different zones called Bounded Context.

Each zone operates within its own context, and have own domain expert.


= Model & Ubiquitous Language
Sales
Customer= Social Interests, likes, needs
DB API
User Manual
Support
Customer= History, Tickets
DB API
User Manual
Accounting
Customer=Method of Payment
DB API
User Manual
Order
Customer=Addresses, Availability
DB API
User Manual

pattern

Tactical Design



Layered Architecture

User Interface

Responsible for showing information to the user and processing the user's input.

Application Layer

This layer is supposed to be thin and it should not contain any domain logic. It can have functionality that is valuable for the business but is not "domain" specific. This includes generating reports, sending email notifications etc.

Domain Layer

Responsible for describing business processes. Abstract domain concepts (including entities, business rules) must be contained in this layer. In contrast, persistence, message sending do not belong here.

Infrastructure Layer

Responsible for persistence, messaging, email delivery etc.

Entity and Value object

Value Object

Don’t care about uniqueness

Always immutable

Rich domain logic

Auto-validating

Entity

Can be uniquely identified using ID

Consists of value objects

Generally persisted as a row in a DB

Typically mutable

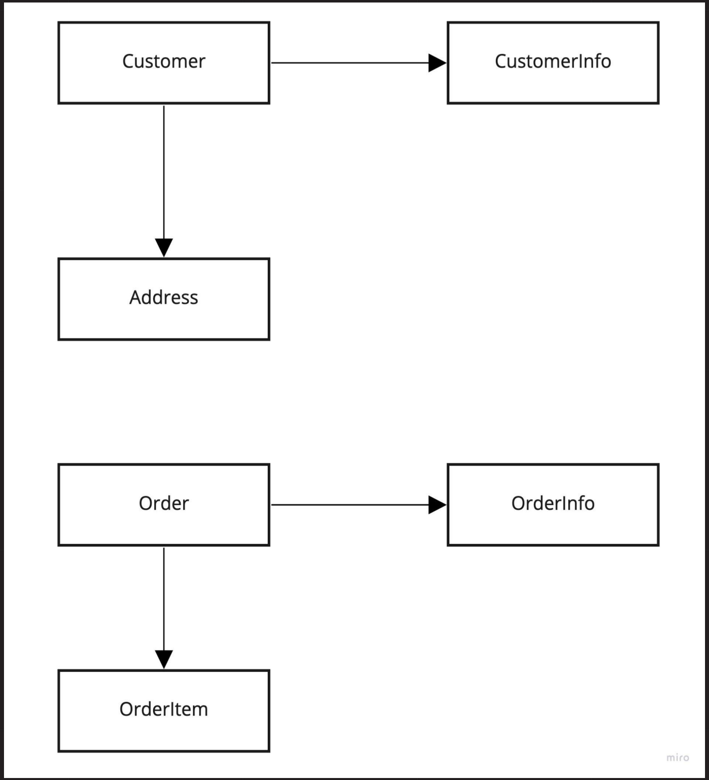
Generally Implements Some business logic

Aggregates

An Aggregate is collection of entities and value which comes under single transaction boundary

An aggregate always has a root entity

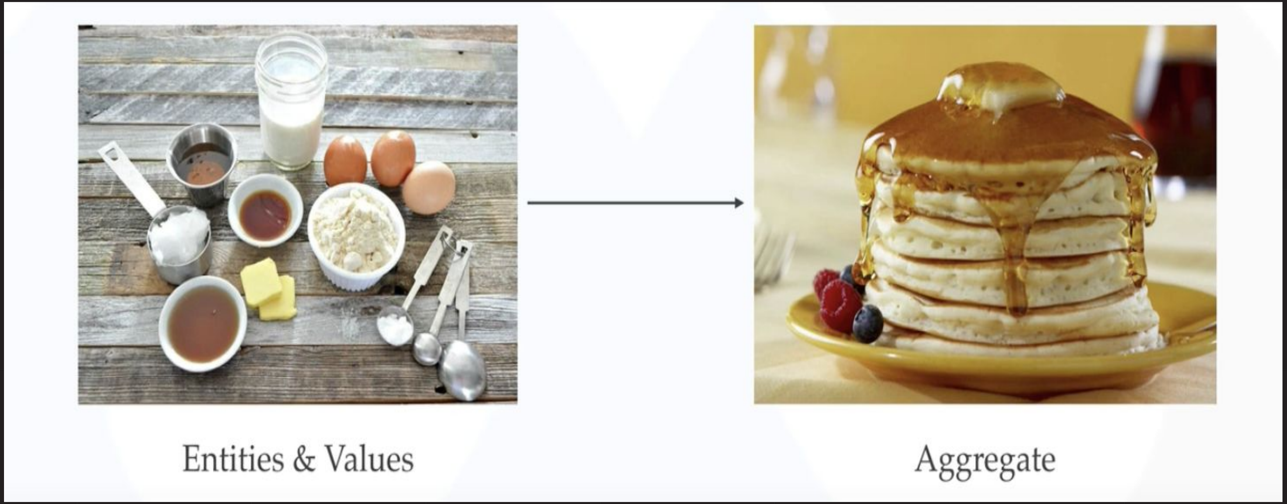
Root entity governs the lifetime of other entities



Factories And Repositories

Factories helps to create new aggregates

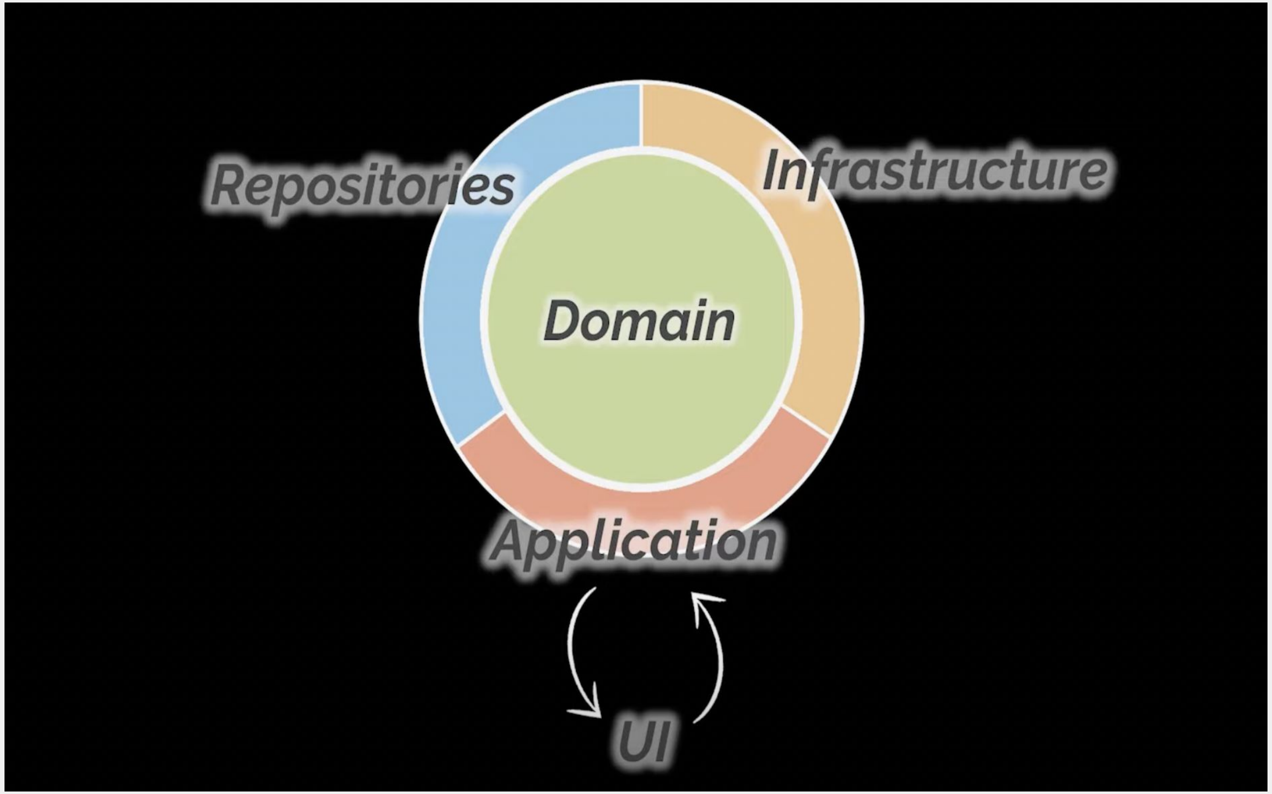
Repositories helps to get persisted aggregates



Hexagonal Architecture

The hexagonal architecture, or ports and adapters architecture, is an architectural pattern used in software design.

It aims at creating loosely coupled application components that can be easily connected to their software environment by means of ports and adapters.


Repositories
Infrastructure
Domain
Application
UI

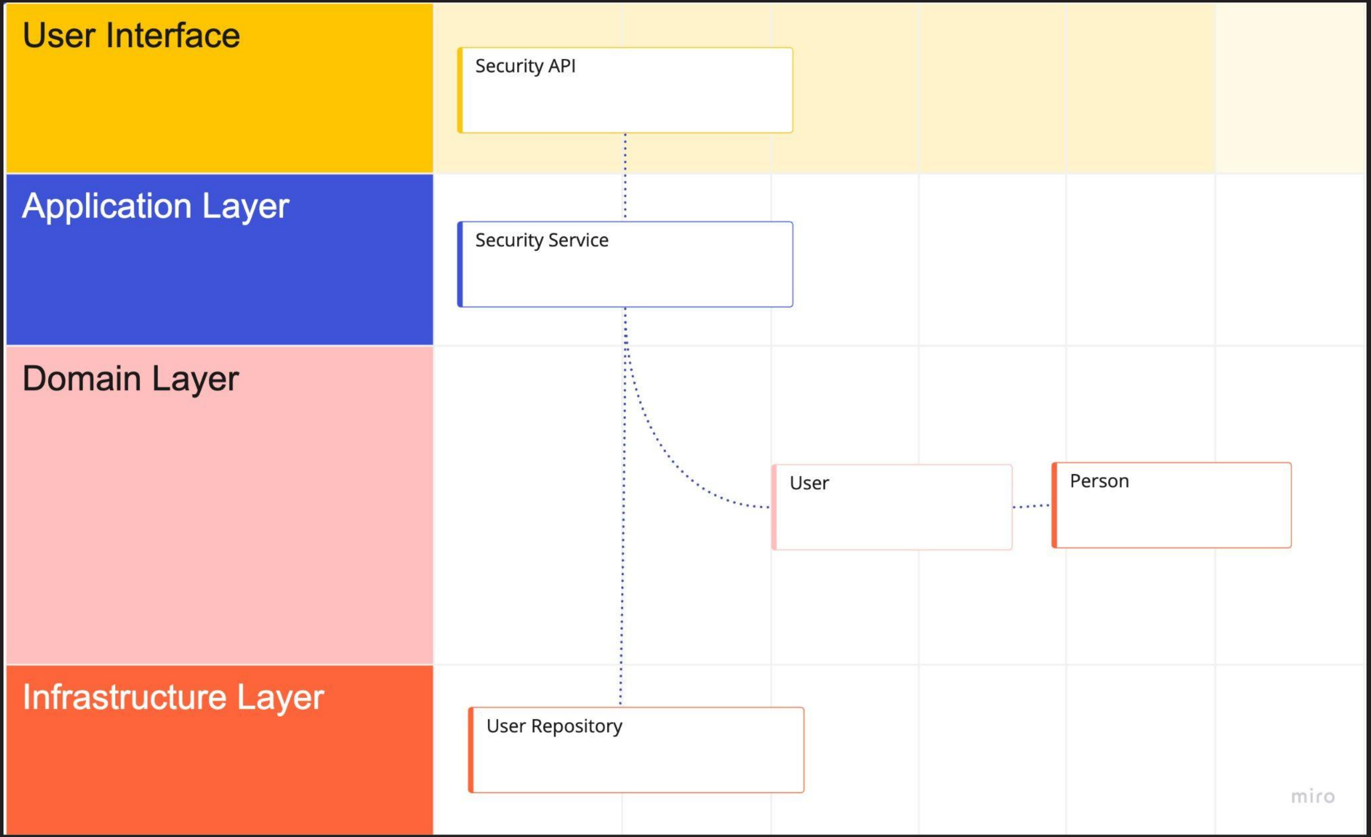
CCD (CORE-CONNECTOR-DELIVERY)

Inside core all business logic related to your domain,

Inside Connector infrastructure code like creating database connection, making s3 connection, SES connection etc

Delivery is nothing but API handler/Event Handler, delivery layer only deals with input/output operation

Let’s jump to the implementation…


User Interface
Application Layer
Domain Layer
Infrastructure Layer
Security API
Security Service
User
Person
User Repository
miro

References

https://www.amazon.in/Domain-Driven-Design-Tackling-Complexity-Software/dp/0321125215

https://en.wikipedia.org/wiki/Hexagonal_architecture_(software)

https://www.sitepoint.com/ddd-for-rails-developers-part-1-layered-architecture/

Connect me

pattern

pattern

Thank You