10. DDD

10.1 What is DDD?

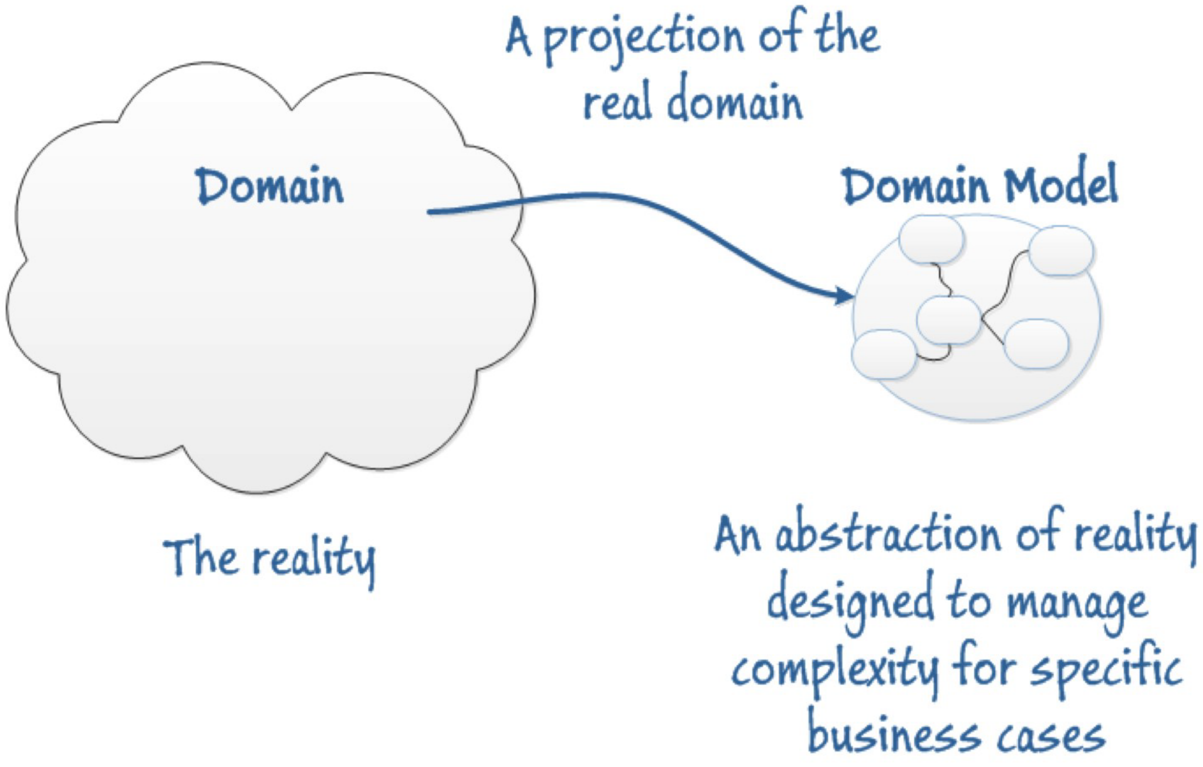
It is a way of thinking (design approach) and a set of priorities, for accellerating sw projects with complicated domains.

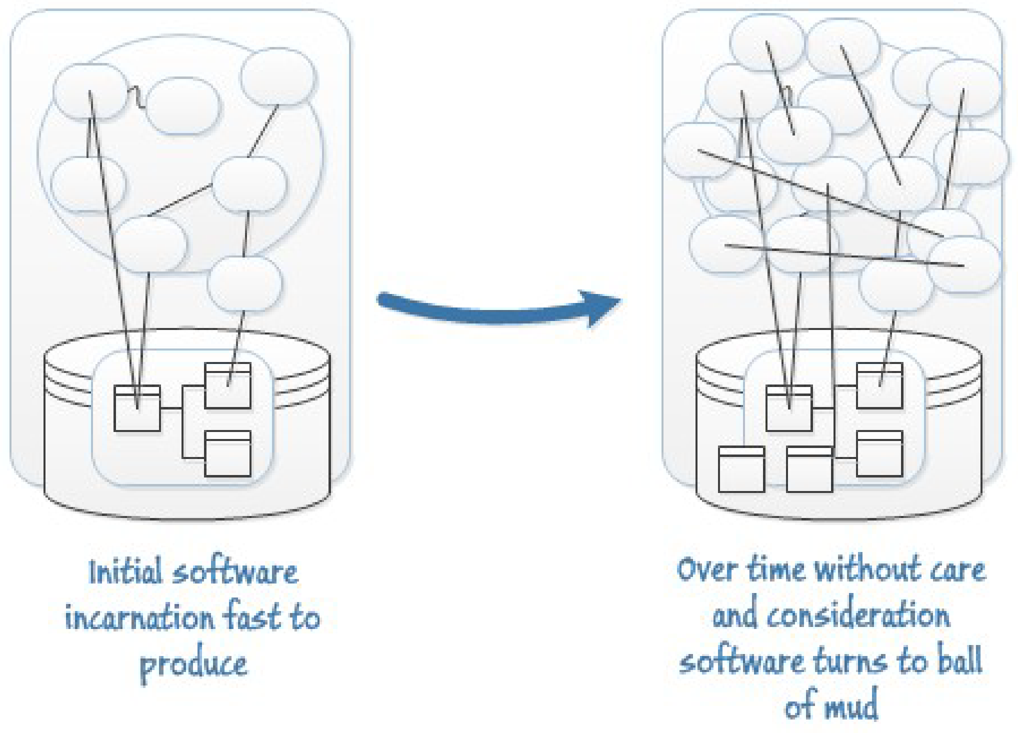
Used to tackle the complexity of sw projects.

help to obtain elegant systems.

Domain: sphere of knowledge or activity

Model: a system of abstractions that describes selected aspects of a domain and ignores extraneous detail. A distilled form of domain knowledge, assumptions, rules, and choices.


A projection of the real domain
Domain
Domain Model
The reality
An abstraction of reality designed to manage complexity for specific business cases


Initial software incarnation fast to produce
Over time without care and consideration software turns to ball of mud

10.2 Principles

Ubiquitous language

a language that structured around the domain model and used by all team members to connect all the activities of the team with the sw.

Change in the language < = > Change model and domain

Model

Helps us solve specific problems, form the basis of the language, should remain current.

Express the model

The model can be expressed through class diagram.

The design document is not the model; its purpose is to help communicate and explain the model.

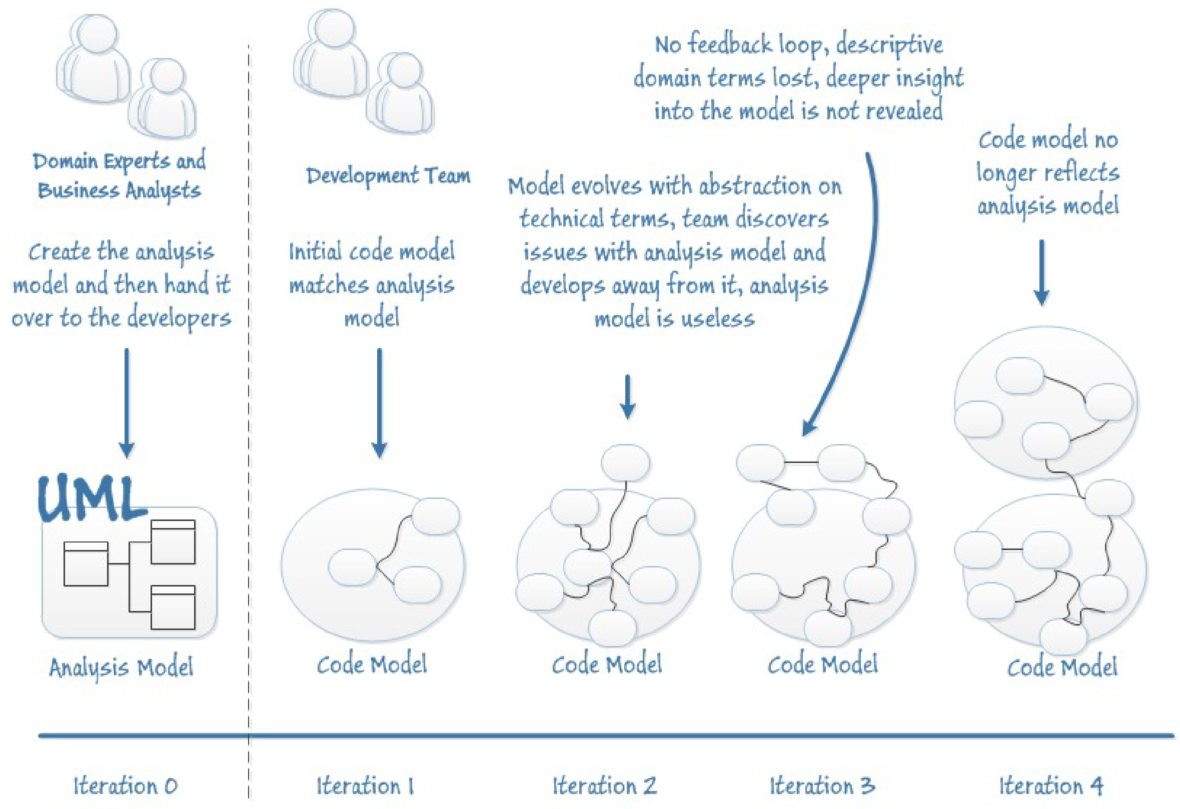
The model is expressed in the code.

10.3 DDD is Agile and iterative

The problem with Big Design Up Front:

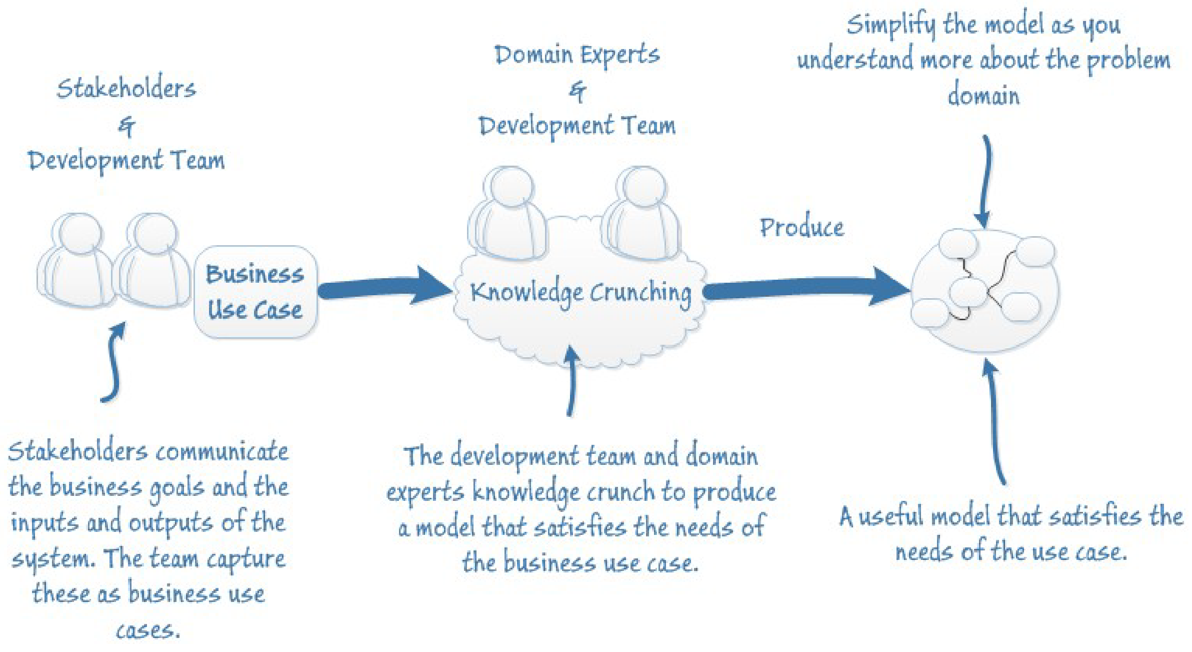
- Models are distilled knowledge.
- At the beginning of a project, the team is as ignorant as it will ever be.

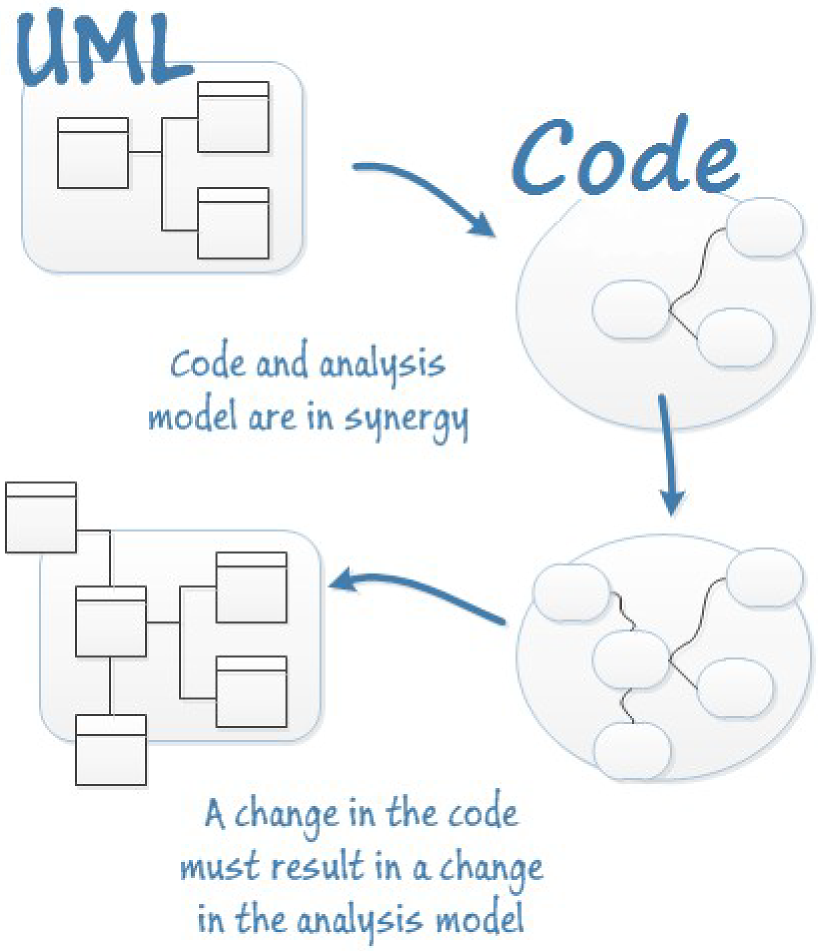
Up front analysis Lock in Ignorance.


Domain Experts and Business Analysts
Create the analysis model and then hand it over to the developers
UML
Analysis Model
Development Team
Initial code model matches analysis model
Code Model
Model evolves with abstraction on technical terms, team discovers issues with analysis model and develops away from it, analysis model is useless
Code Model
No feedback loop, descriptive domain terms lost, deeper insight into the model is not revealed
Code Model
Code model no longer reflects analysis model
Code Model
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4

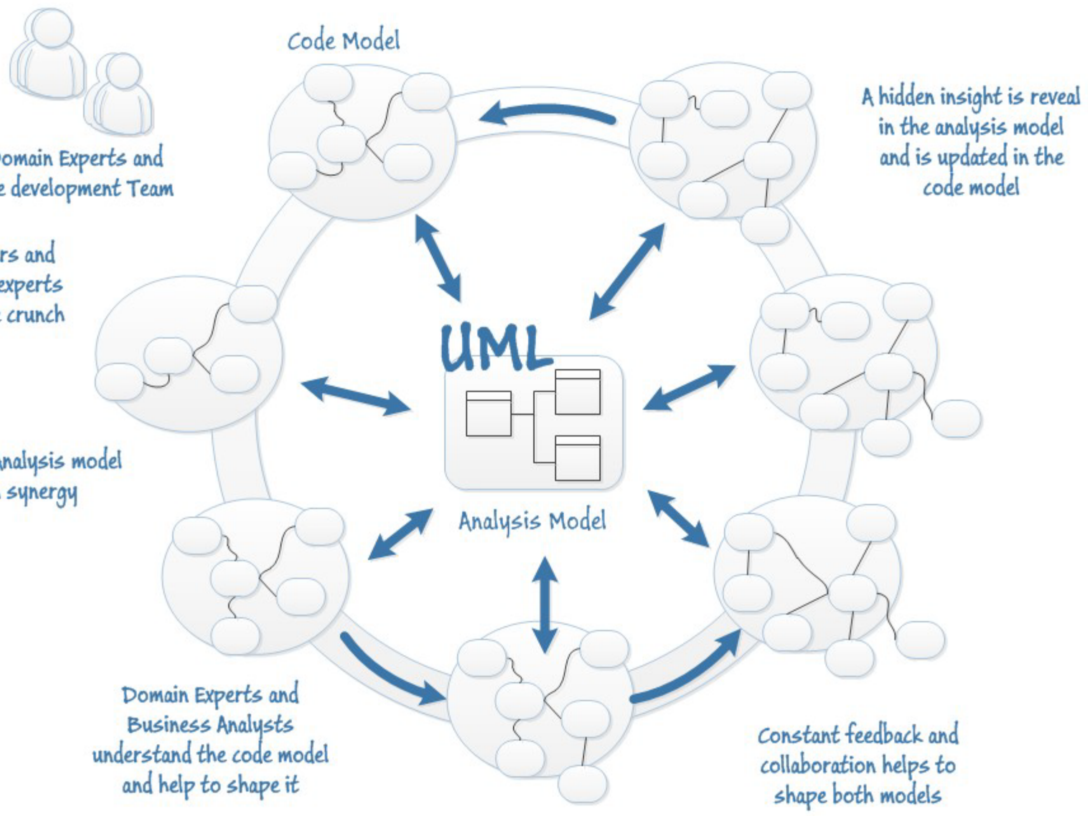
The idea is to continue analyze everything, also the model.

10.4 The DDD process

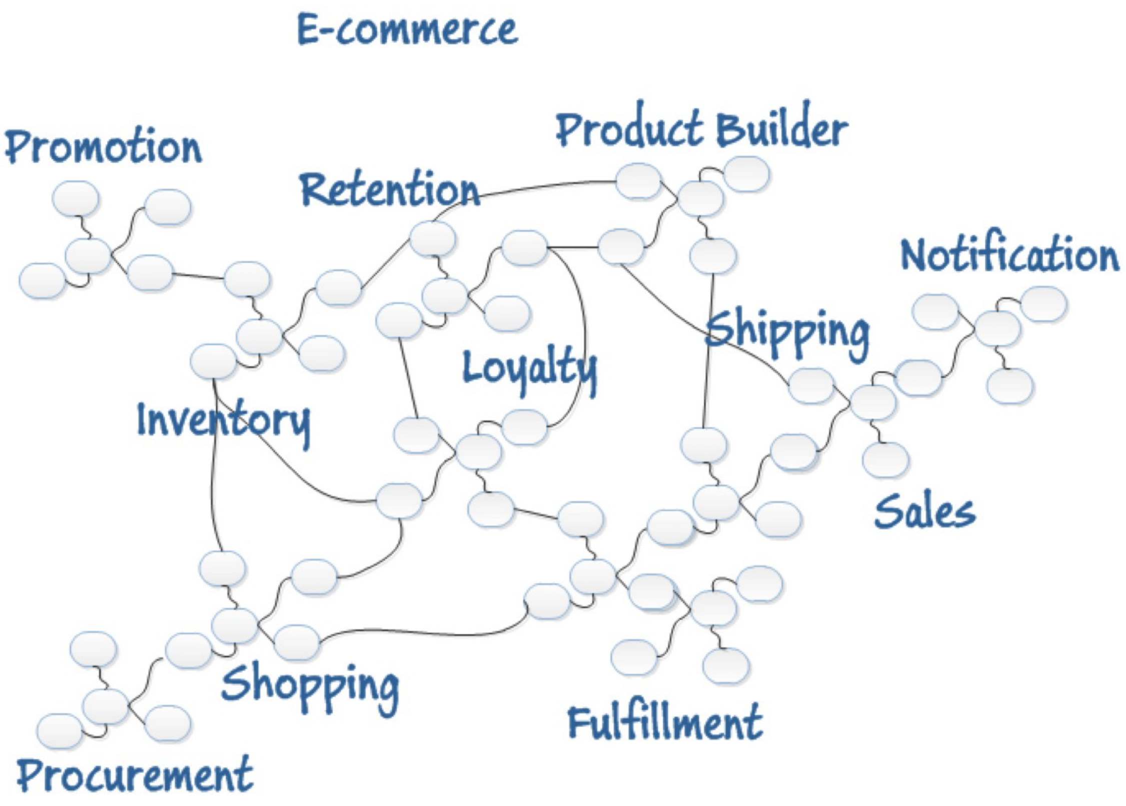

Stakeholders
&
Development Team
Business Use Case
Domain Experts
&
Development Team
Knowledge Crunching
Produce
Simplify the model as you understand more about the problem domain
Stakeholders communicate the business goals and the inputs and outputs of the system. The team capture these as business use cases.
The development team and domain experts knowledge crunch to produce a model that satisfies the needs of the business use case.
A useful model that satisfies the needs of the use case.


UML
Code
Code and analysis model are in synergy
A change in the code must result in a change in the analysis model

One Team, One Language, One Model


Code Model
A hidden insight is reveal in the analysis model and is updated in the code model
Domain Experts and the development Team
Developers and business experts knowledge crunch
UML
Analysis Model
Code and Analysis model are in synergy
Domain Experts and Business Analysts understand the code model and help to shape it
Constant feedback and collaboration helps to shape both models

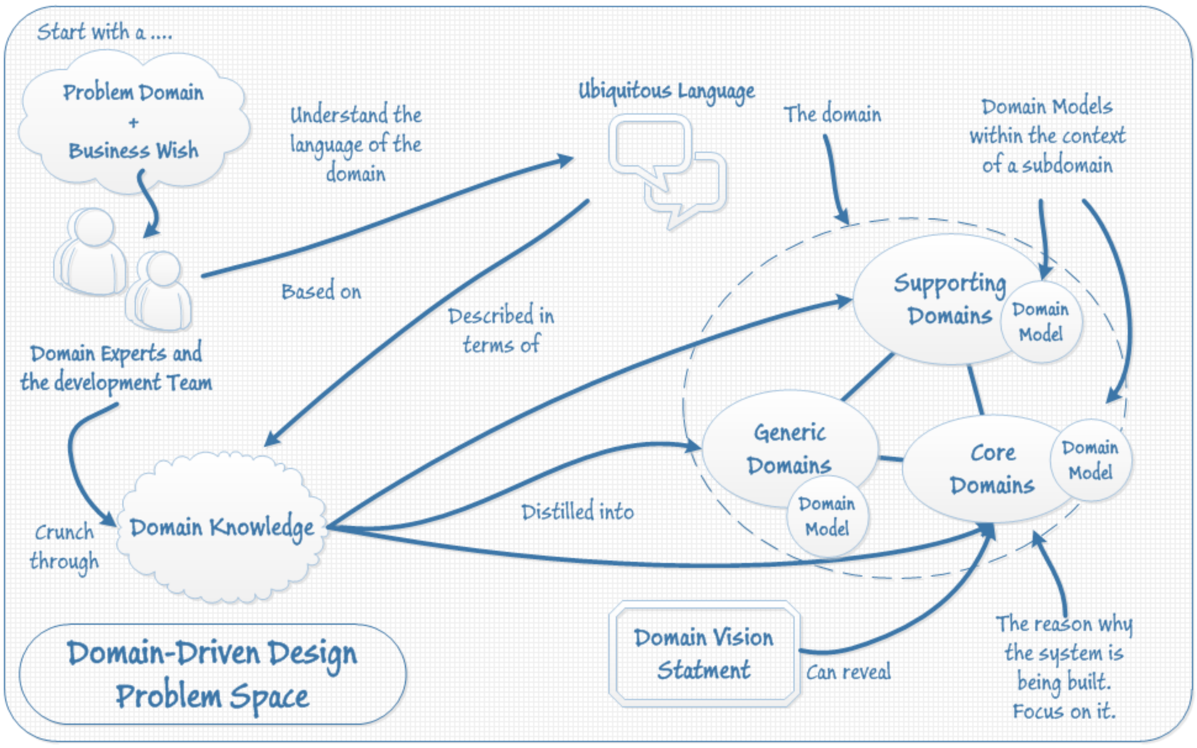
10.5 A Complex Domain



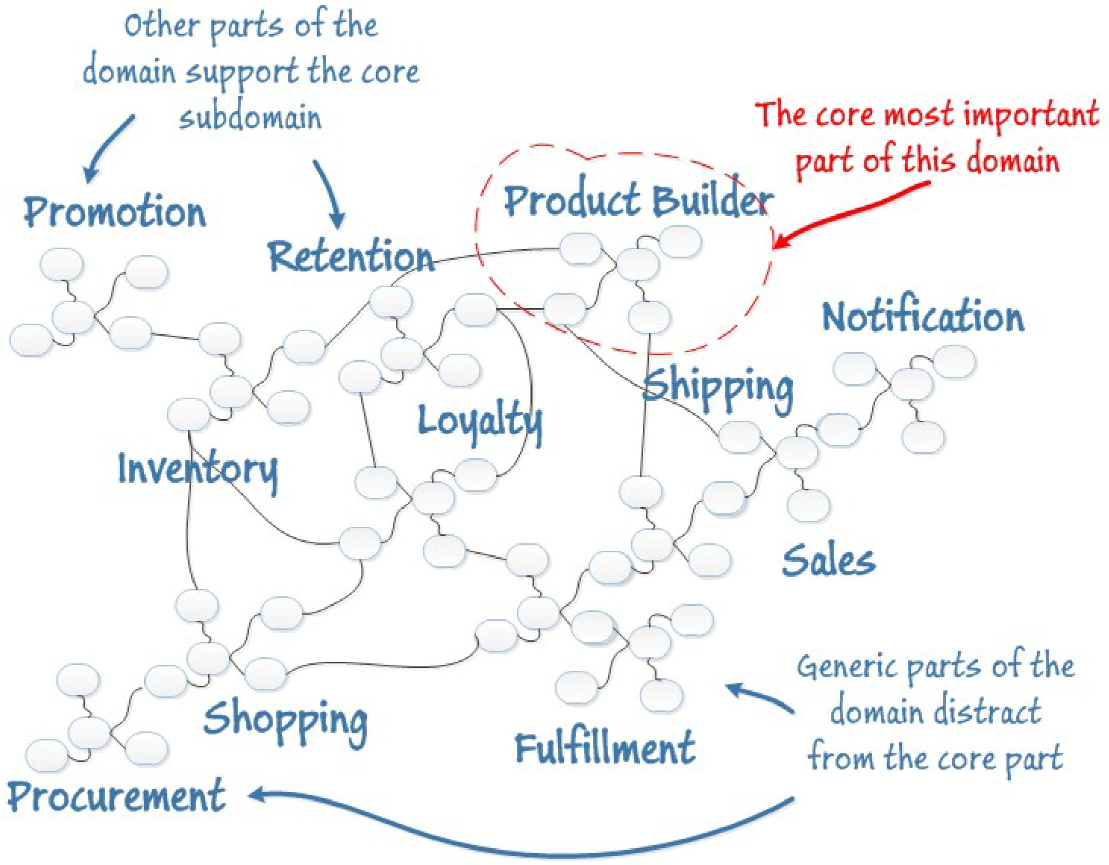
Breaking Down a Complex Domain

Bounded Context: An operational definition of where a particular model is well-defined and applicable.
Subdomain: Part of the domain, based on a particular conceptual decomposition of the domain.

10.6 The Problem Space

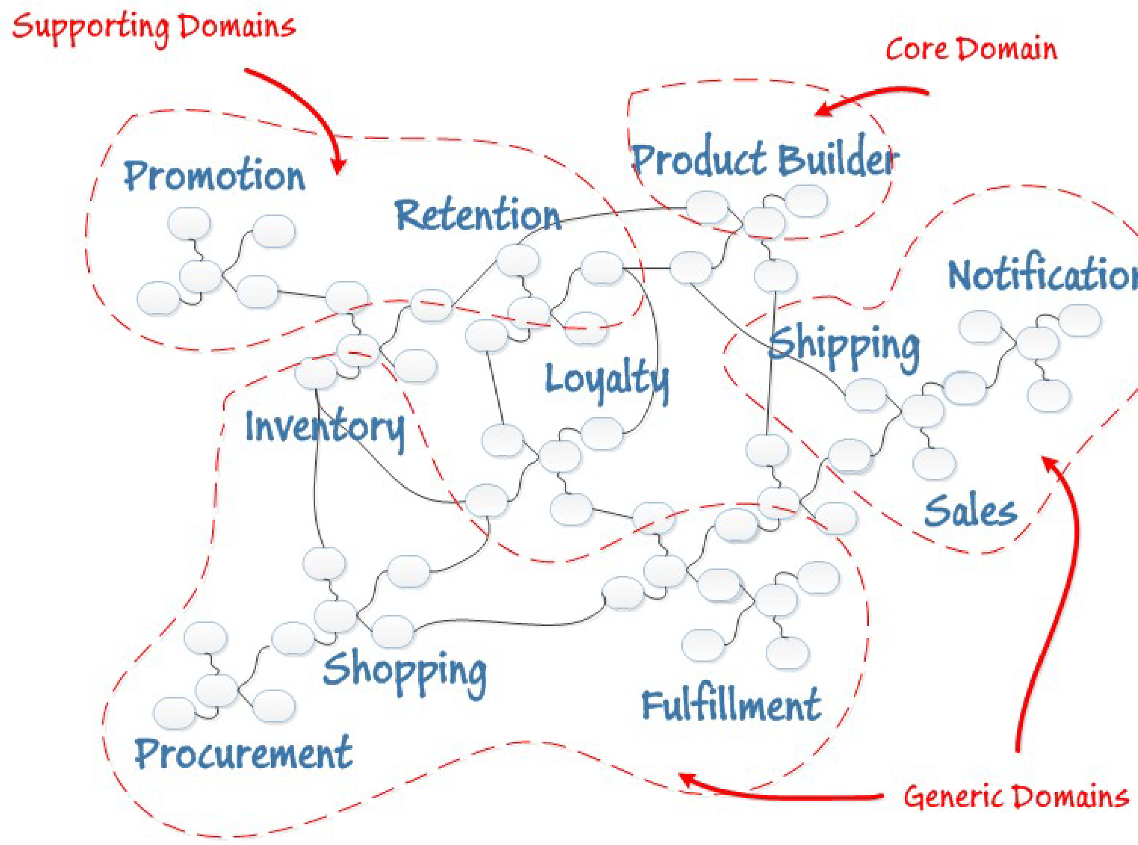


Start with a ....
Problem Domain + Business Wish
Understand the language of the domain
Ubiquitous Language
The domain
Domain Models within the context of a subdomain
Based on
Described in terms of
Supporting Domains
Domain Model
Domain Experts and the development Team
Generic Domains
Domain Model
Core Domains
Domain Model
Crunch through
Domain Knowledge
Distilled into
Domain-Driven Design Problem Space
Domain Vision Statment
Can reveal
The reason why the system is being built. Focus on it.

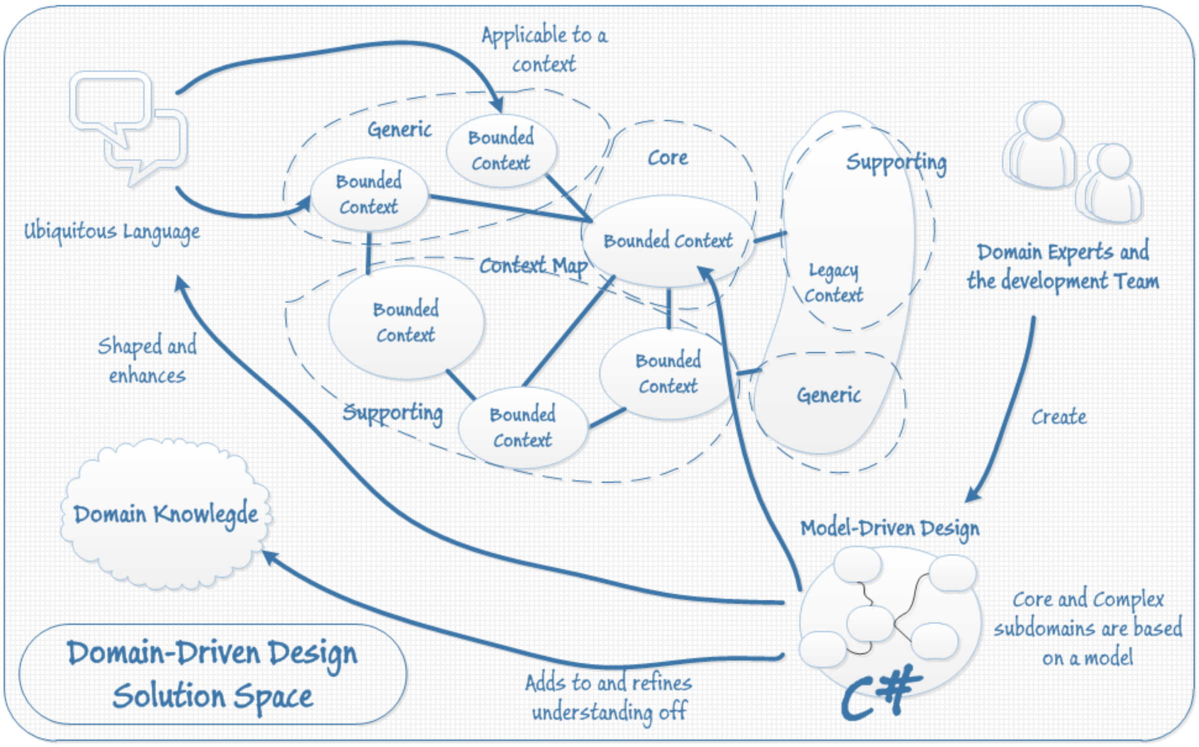


Other parts of the domain support the core subdomain
The core most important part of this domain
Promotion
Retention
Product Builder
Notification
Shipping
Loyalty
Inventory
Sales
Generic parts of the domain distract from the core part
Shopping
Fulfillment
Procurement

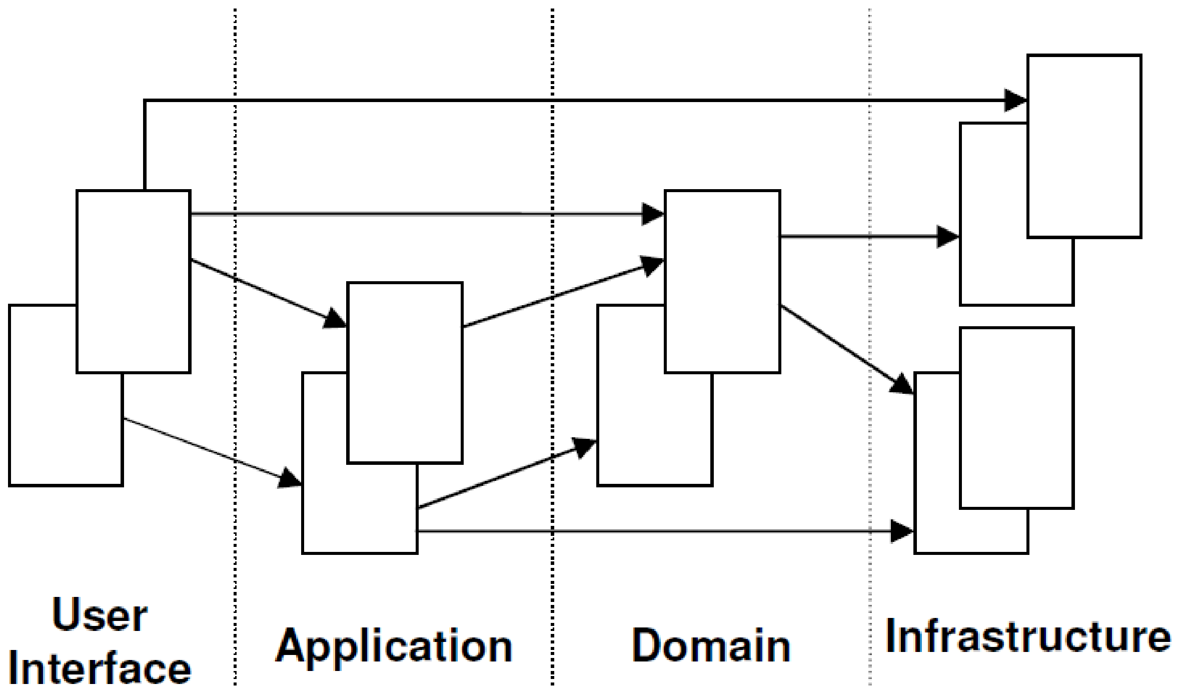
We need to understand how to divide areas:



The Solution Space



10.7 Layered Architecture



Typically when we do in DDD the architecture is layered.

Layers

- UI: responsible for presenting info to the user.
- APP: thin layer, coordinates app activity; not contains business logic.
- Domain: contains information about the domain (heart of the business sw) and the state of business; Persistence details delegated to the Infrastructure layer.
- Infrastructure: acts as a supporting library for all the other layers; provides communication between layers, contains supporting libraries for the UI.

10.8 Model Expressed in SW

Entities

objects with identity

must be distinguished from other similar with the same attributes; Attributes can change; Entities should have behaviour.

Value Objects

Thing within your model with no uniqueness; are equal to other value objects only if all their attributes match.

They are interchangeable, and immutable.

Aggregates

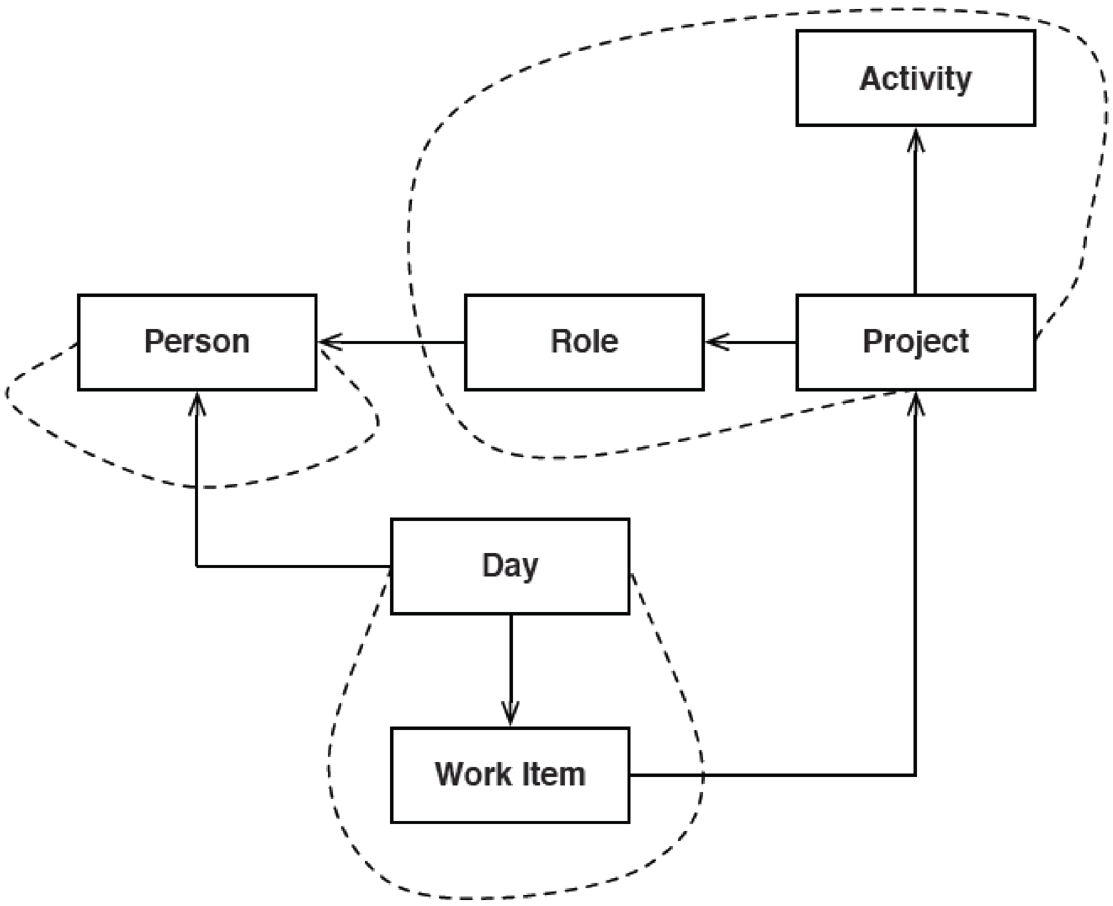
It is a cluster of Entities and Value objects.

Each aggregate is treated as a single unit and has one root entity know as the **Aggregate Root**.

The root identity is global, the identities of entities inside are local.

External objects may have references only to root.

Internal cannot be change from the outside.



Associations

Impose a traversal direction, add a qualifier, reduce multiplicity.

Services

They reside in Multiple Layers

- Application: all the activity and services are APPLICATION SERVICE
- Domain: everything that makes sense at domain level (debits and credits transfer).
- Infrastructure: the services that are composed by systems etc. are called Infrastructure Service.

Domain services: business logic in the domain that are not natural part of an entity or Value Object.

Services are stateless; A service has to be offered as an interface that is defined as part of the model.

Factories

Object whose responsibility is the creation of other objects

for creating aggregates and complex domains objects.

Repositories

encapsulate domain objects persistence and retrieval

clean separationand one-way dependency between the domain and data mapping layers

encapsulate different fetching strategies; One Repository per Aggregate!

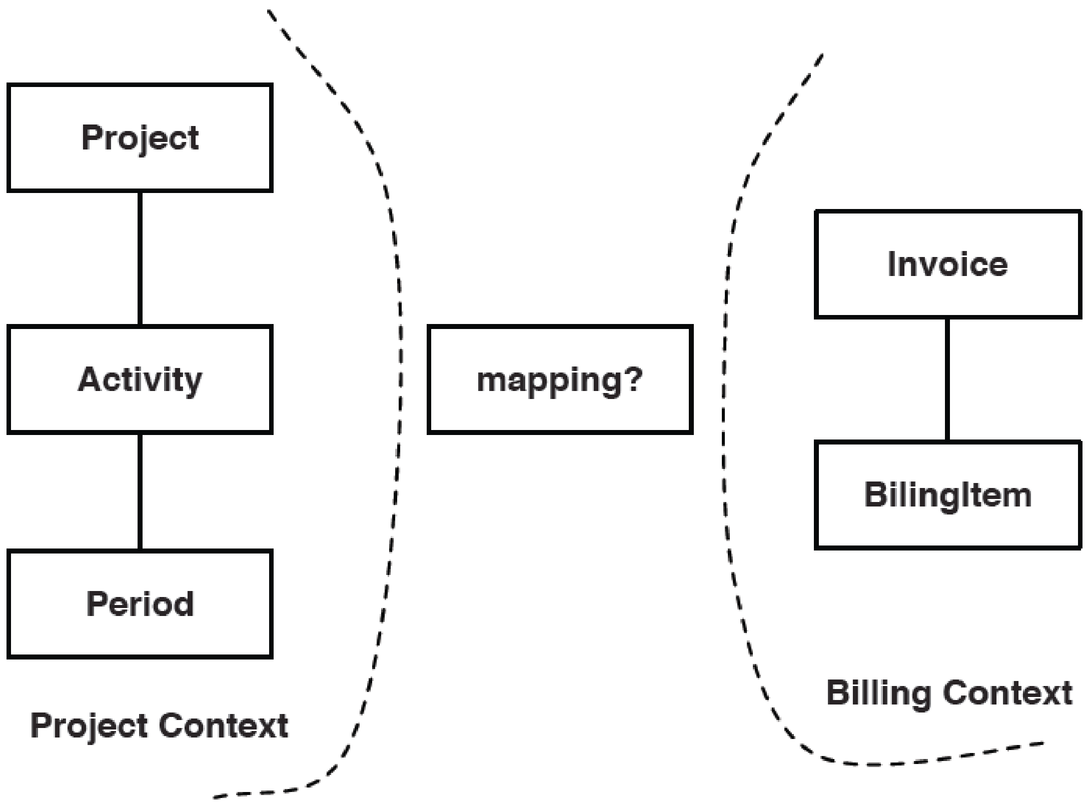
Modules

reduce complexity; hgh cohesion within module

part of the ubiquitous language, helps the decoupling

Context mapping

Mapping the contact points and translations between bounded contexts.



Persistence details delegated to the
Infrastructure layer.