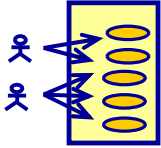
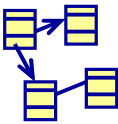
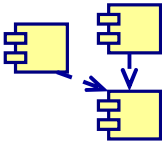
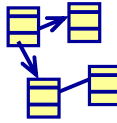
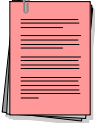
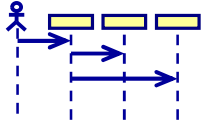
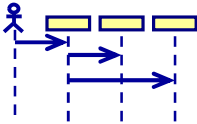
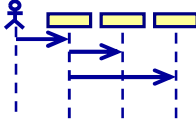
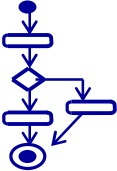
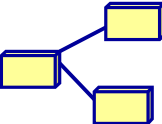
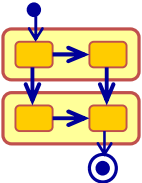


Lesson 7

Architecture

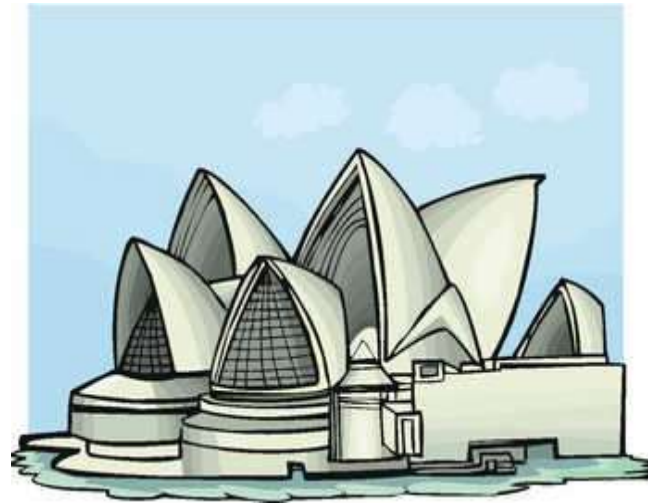
Requirements	Analysis	Architecture	Design
 use case diagram	 class diagram	 component diagram	 class diagram
 scenarios	 sequence diagram	 sequence diagram	 sequence diagram
 activity diagram		 deployment diagram	
 state diagram			

ARCHITECTURE/DESIGN PRINCIPLES

Design principles

- Keep it simple
- Keep it flexible
- Loose coupling
- Separation of concern
- Information hiding
- Principle of modularity

Keep it simple



Keep it flexible

- Everthing changes
 - Business
 - Technical
- More flexibility leads to more complexity



Loose coupling

- Different levels of coupling
 - Technology
 - Time
 - Location
 - Data structure
- You need coupling somewhere
 - Important is the level of coupling



Separation of concern

- Separate technology from business
- Separate stable things from changing things
- Separate things that need separate skills
- Separate business process from application logic
- Separate implementation from specification



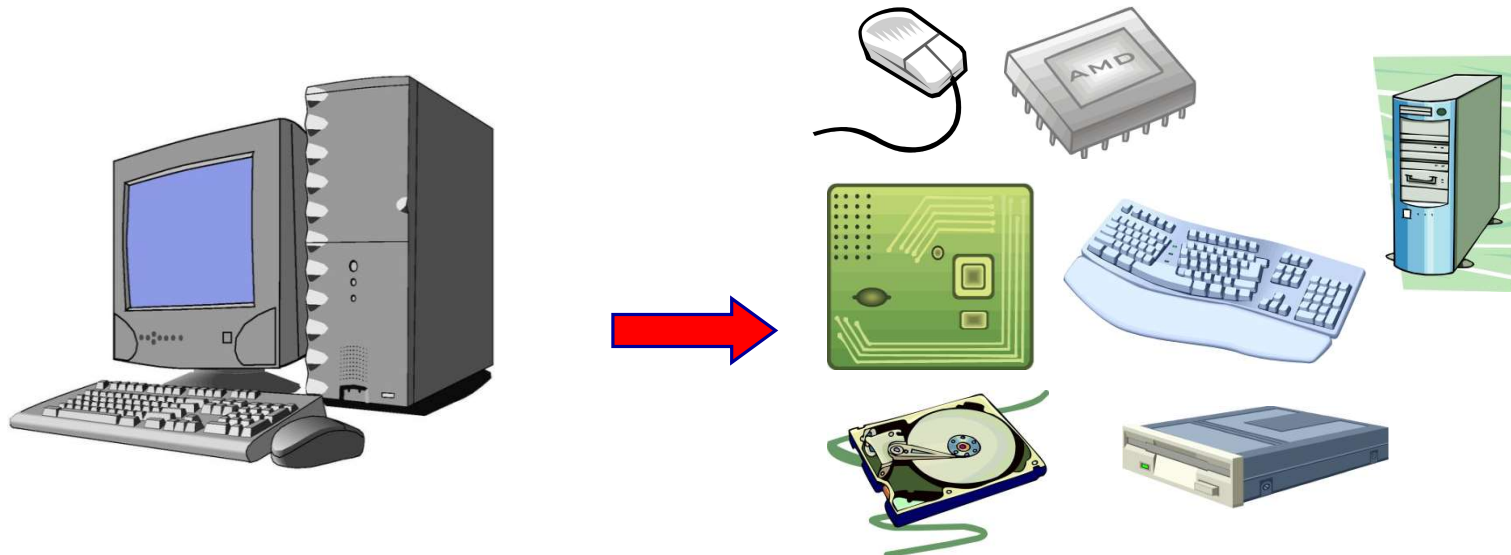
Information hiding

- Black box principle
- Hide implementation behind an interface
- Hide the data structure behind stored procedures
- Hide the data structure behind business logic

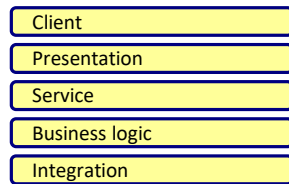


Principle of modularity

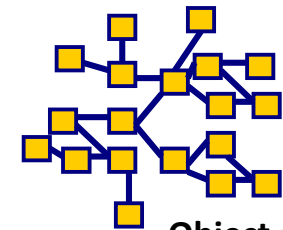
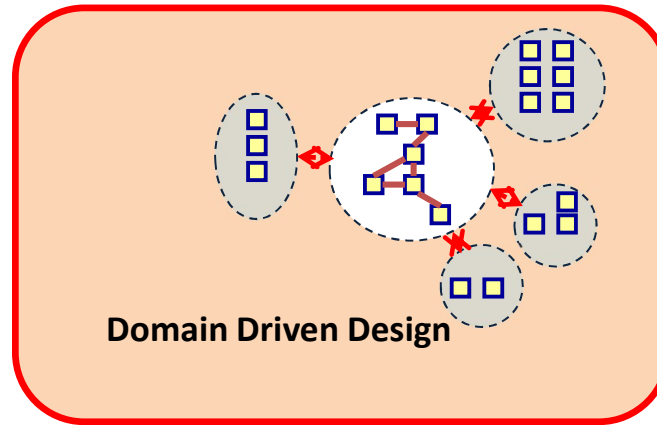
- Decomposition
- Devide a big complex problem is smaller parts
- Use components that are
 - Better understandable
 - Independent
 - Reusable
- Leads to more flexibility
- Makes finding and solvings bugs easier



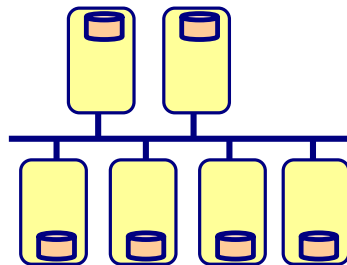
Architecture styles



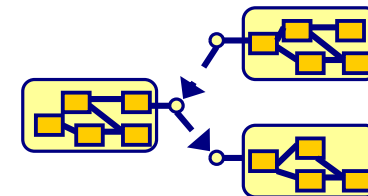
Layering



Object oriented



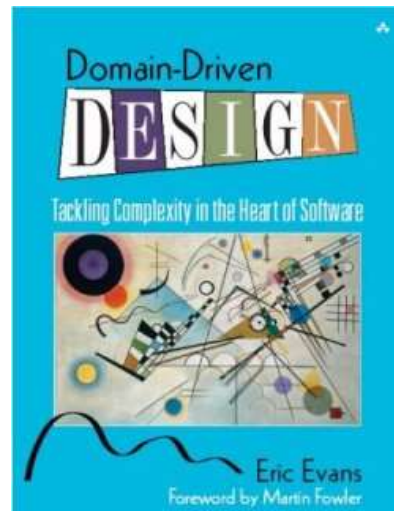
Microservices



Component based

What is Domain Driven Design?

- An approach to software development where the focus is on the core **Domain**.
 - We create a **domain model** to communicate the domain
 - Everything we do (discussions, design, coding, testing, documenting, etc.) is based on the domain model.



Principles of Domain Driven Design

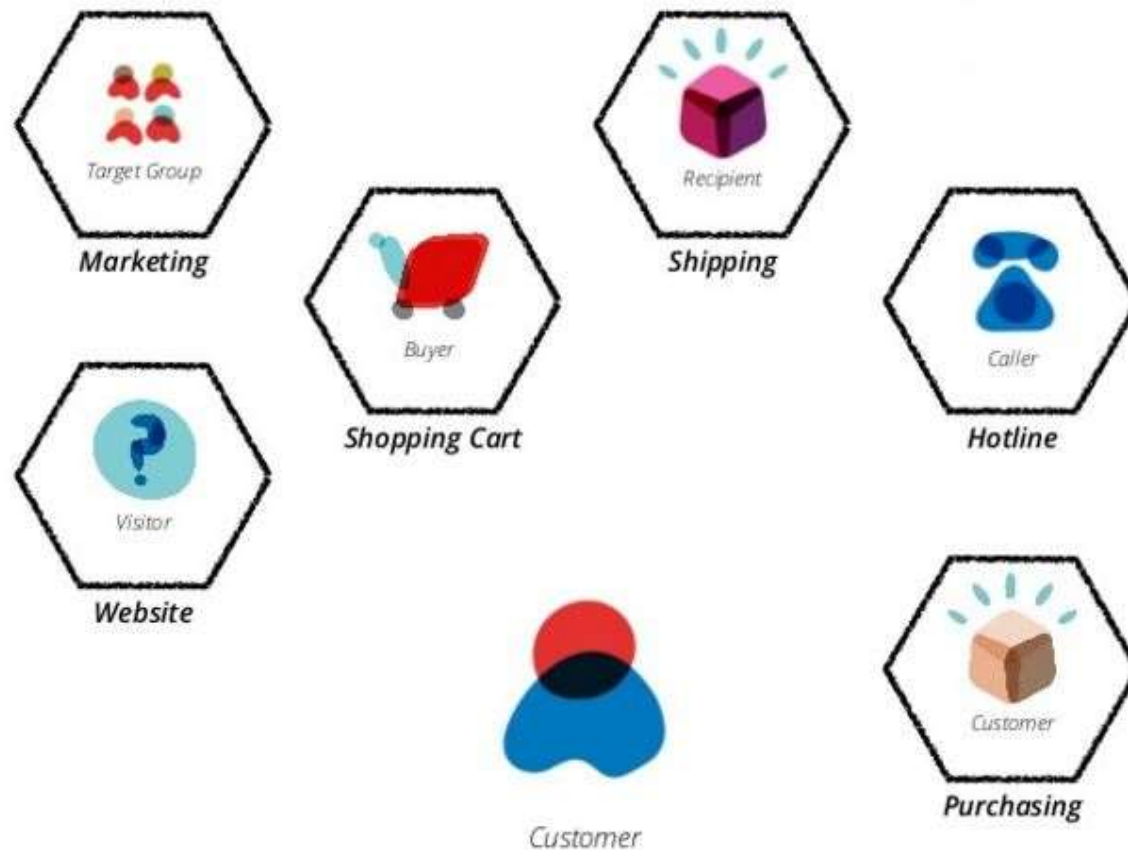
- Use one common language to describe the concepts of a domain
 - Ubiquitous language
- Create a domain model that shows the important concepts of the domain
 - Rich domain model
- Let the software be a reflection of the real world domain
- Create small contexts in which a domain model is valid
 - Bounded context

Principles of Domain Driven Design

- Use one common language to describe the concepts of a domain
 - Ubiquitous language
- Create a domain model that shows the important concepts of the domain
 - Rich domain model
- Let the software be a reflection of the real world domain
- Create small contexts in which a domain model is valid
 - Bounded context

Common language

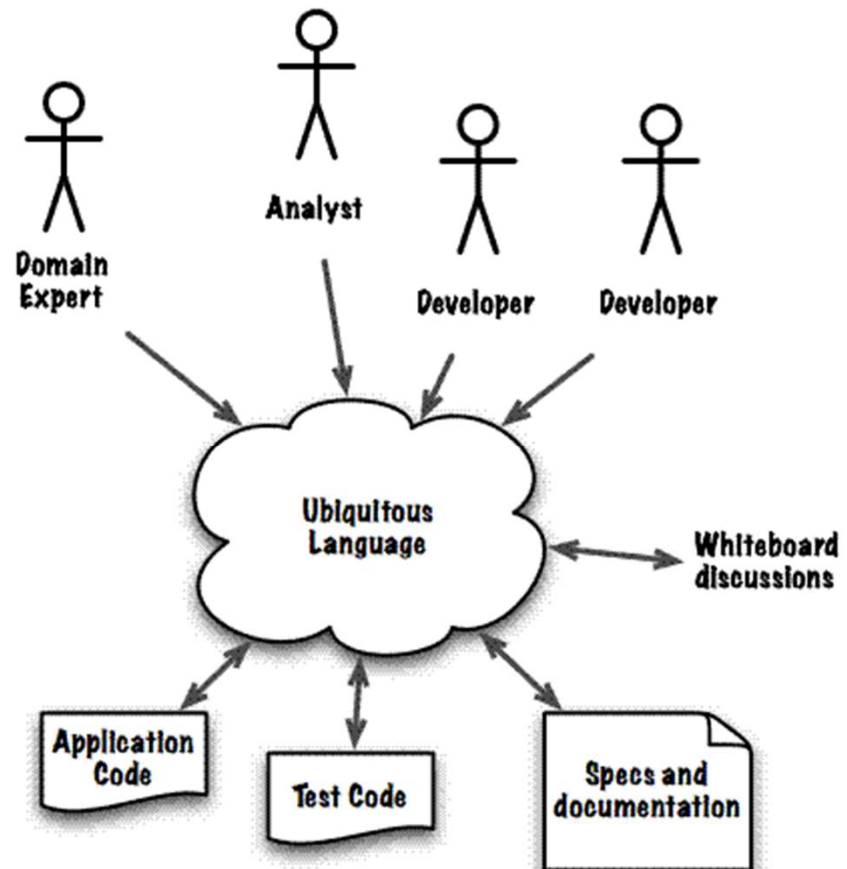
- Different people from the business use different names for the same thing.



We need a common language

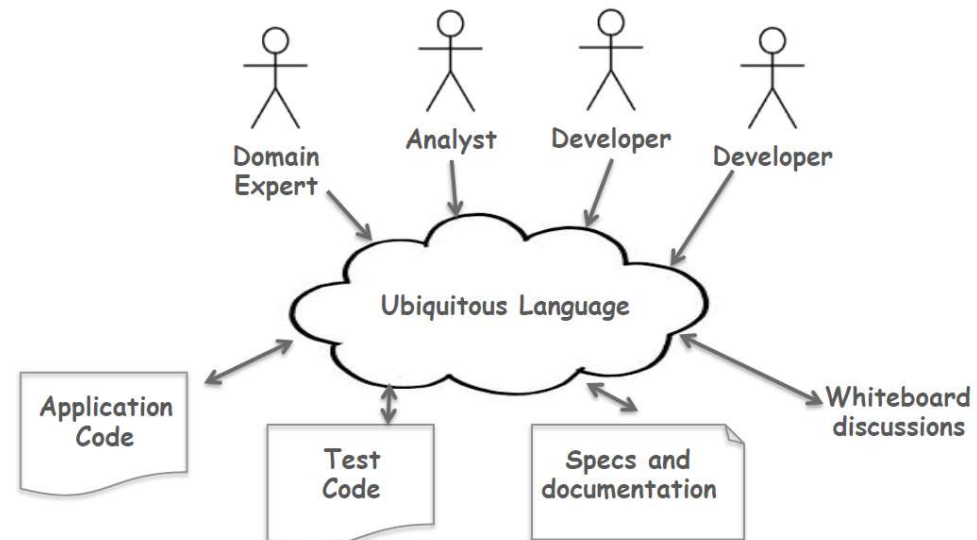
Ubiquitous Language

- Language used by the team to capture the concepts and terms of a specific core business domain.
 - Used by the people
 - Used in the code
 - Used everywhere



Ubiquitous Language

- Based on the domain model
- Takes time to create
- Can change in time
 - New names are discovered
- The whole team should use the common language



Principles of Domain Driven Design

- Use one common language to describe the concepts of a domain
 - Ubiquitous language
- Create a domain model that shows the important concepts of the domain
 - Rich domain model
- Let the software be a reflection of the real world domain
- Create small contexts in which a domain model is valid
 - Bounded context

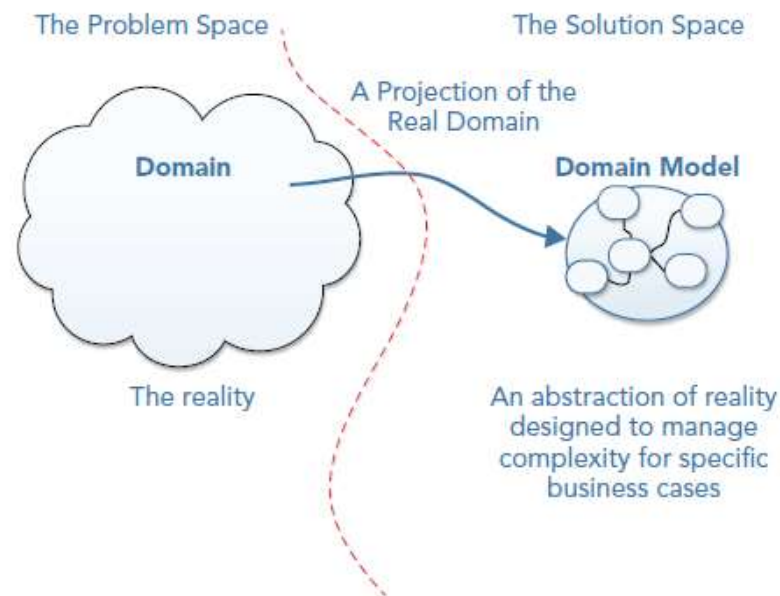
Model



- More complexity -> More modeling
 - Higher level of abstraction
 - Allows for visualization
 - Vehicle of communication

Domain model

- Extracts domain **essential** elements
 - **Relevant** to a specific use
- Layers of **abstractions** representing **selected** aspects of the domain
- Contains **concepts** of importance and their **relationships**



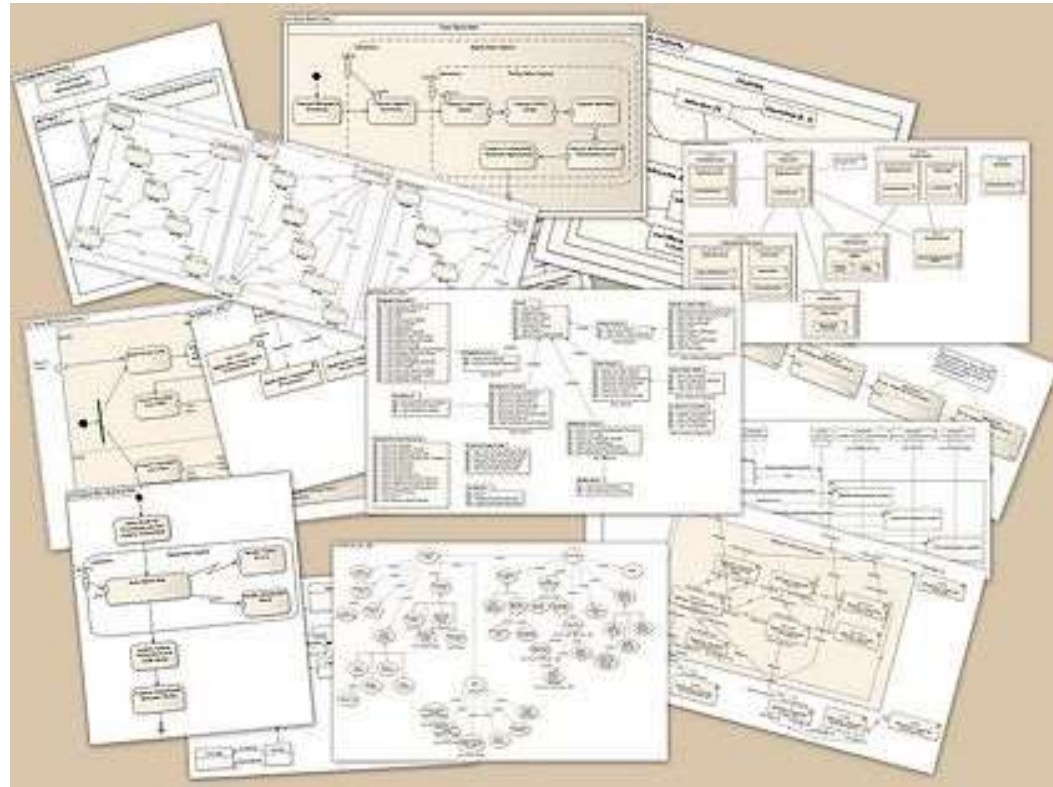
Domain model

- Simplification of reality
- Area of interest



Structure of the domain model

- A domain model is not a particular diagram
- Use the format that communicates the best
 - Diagram
 - Text
 - Code
 - Table
 - Formula



Effective communication

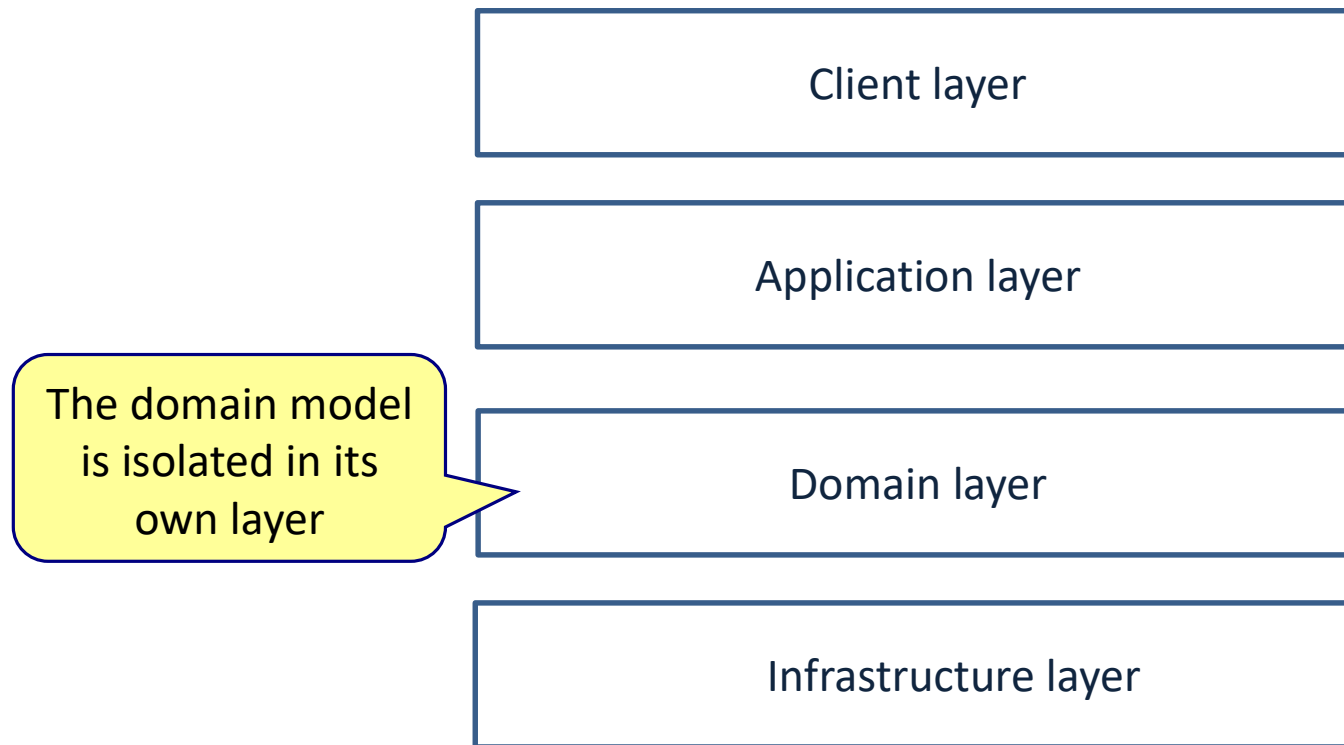




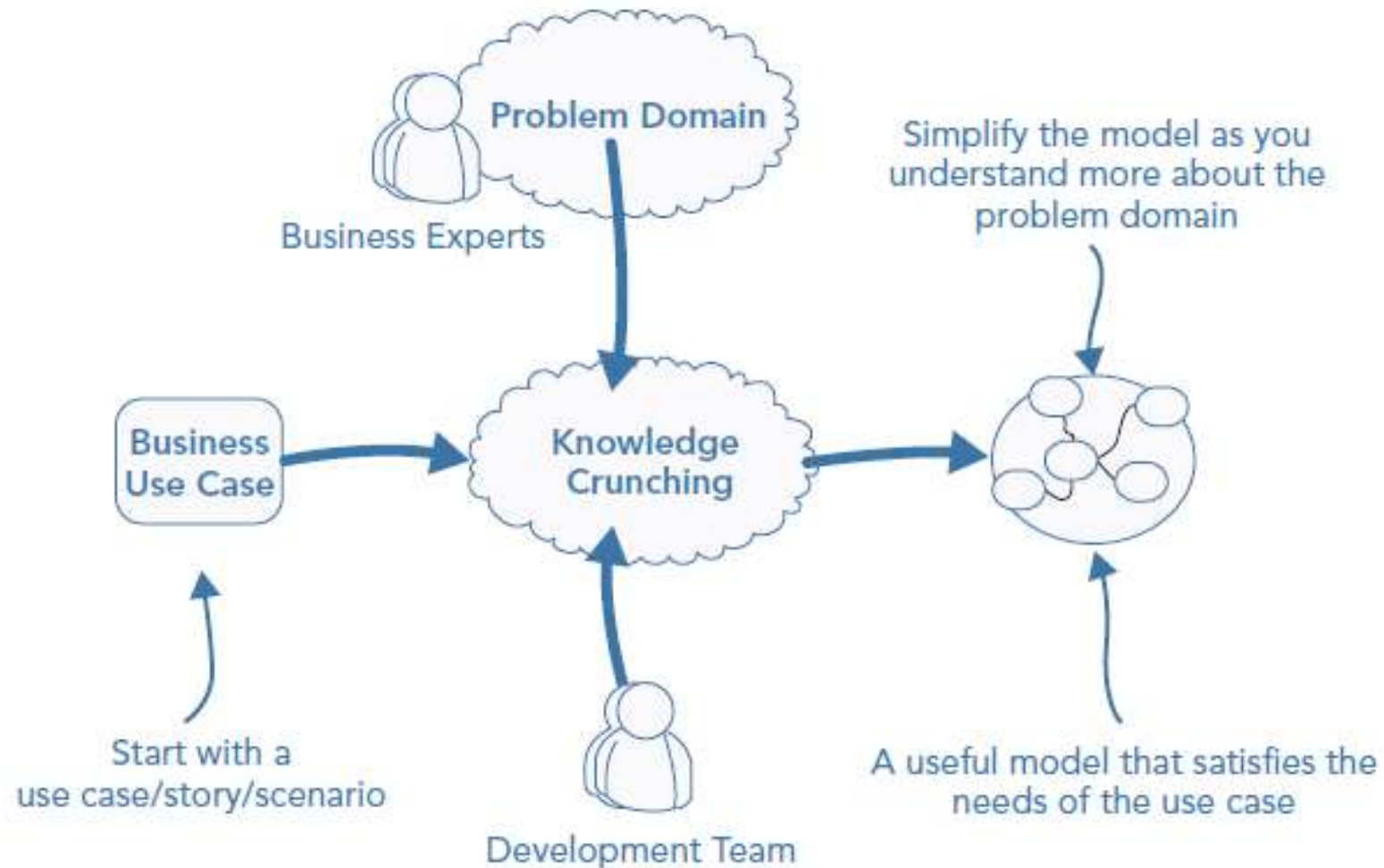
Advantages of a domain model

- Improves understanding
- Validates understanding
- Improves communication
- Shared glossary
- Improves discovery

Isolate the domain model



Knowledge crunching



Principles of Domain Driven Design

- Use one common language to describe the concepts of a domain
 - Ubiquitous language
- Create a domain model that shows the important concepts of the domain
 - Rich domain model
- Let the software be a reflection of the real world domain
- Create small contexts in which a domain model is valid
 - Bounded context

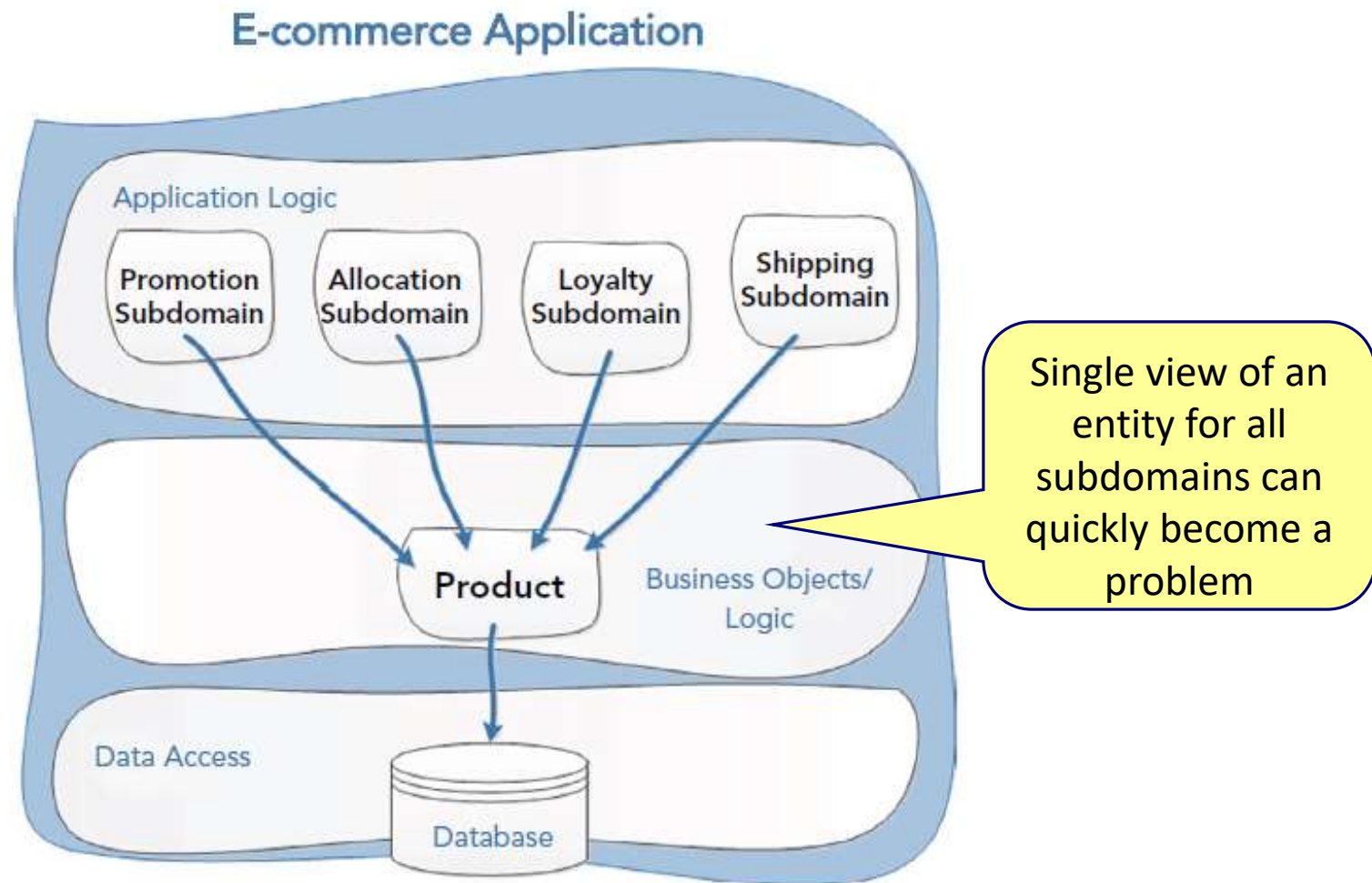
The software is a reflection of the real world

- It is easier to spot inconsistencies, errors, misconceptions.
- The software is easier to understand for
 - Existing developers
 - Testers
 - Business people (with guidance)
 - New developers and testers
- By looking at the code you can learn a lot of domain knowledge
- No translation necessary
- It is easier to write tests
- Easier to maintain the code

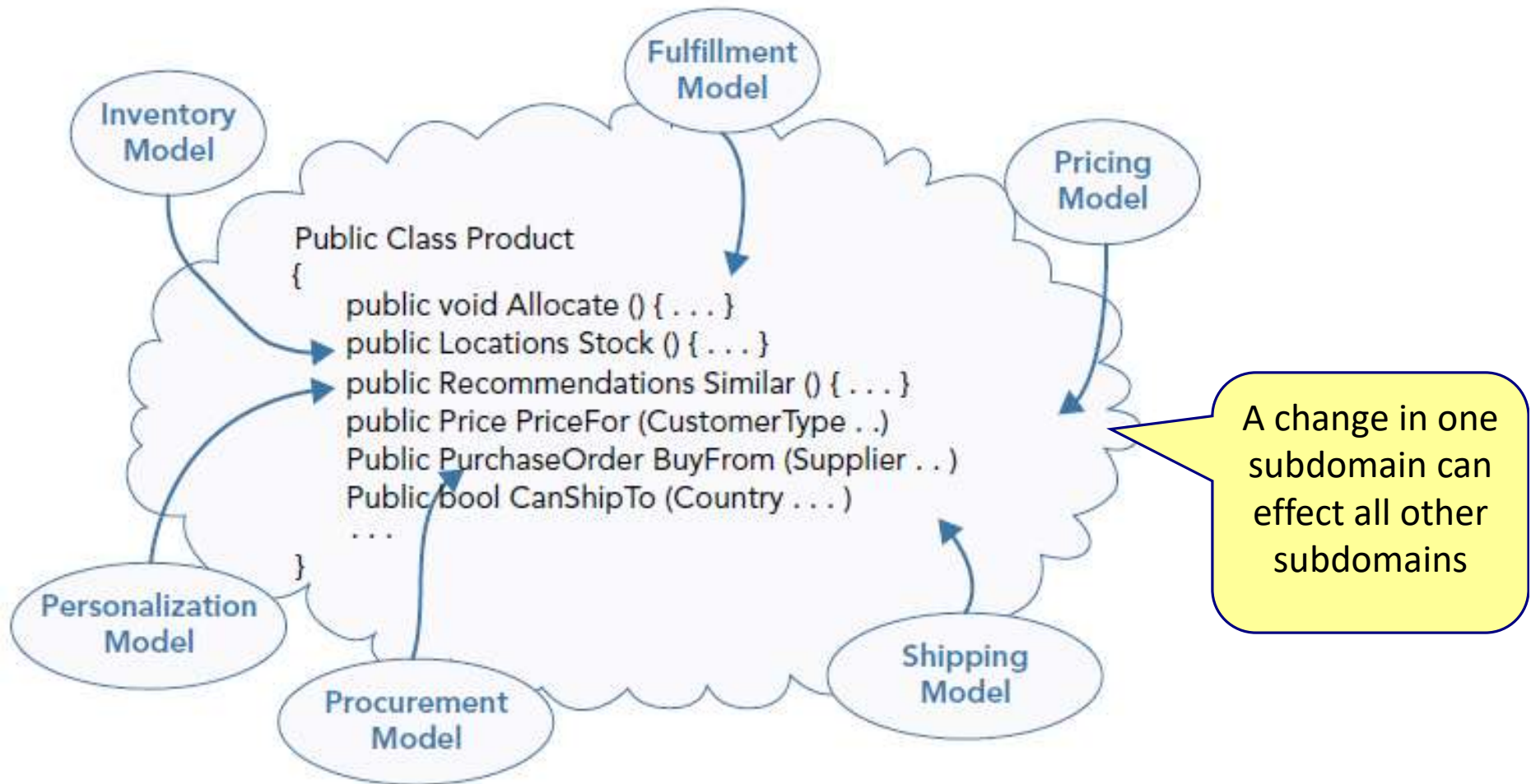
Principles of Domain Driven Design

- Use one common language to describe the concepts of a domain
 - Ubiquitous language
- Create a domain model that shows the important concepts of the domain
 - Rich domain model
- Let the software be a reflection of the real world domain
- Create small contexts in which a domain model is valid
 - Bounded context

Shared objects between subdomains

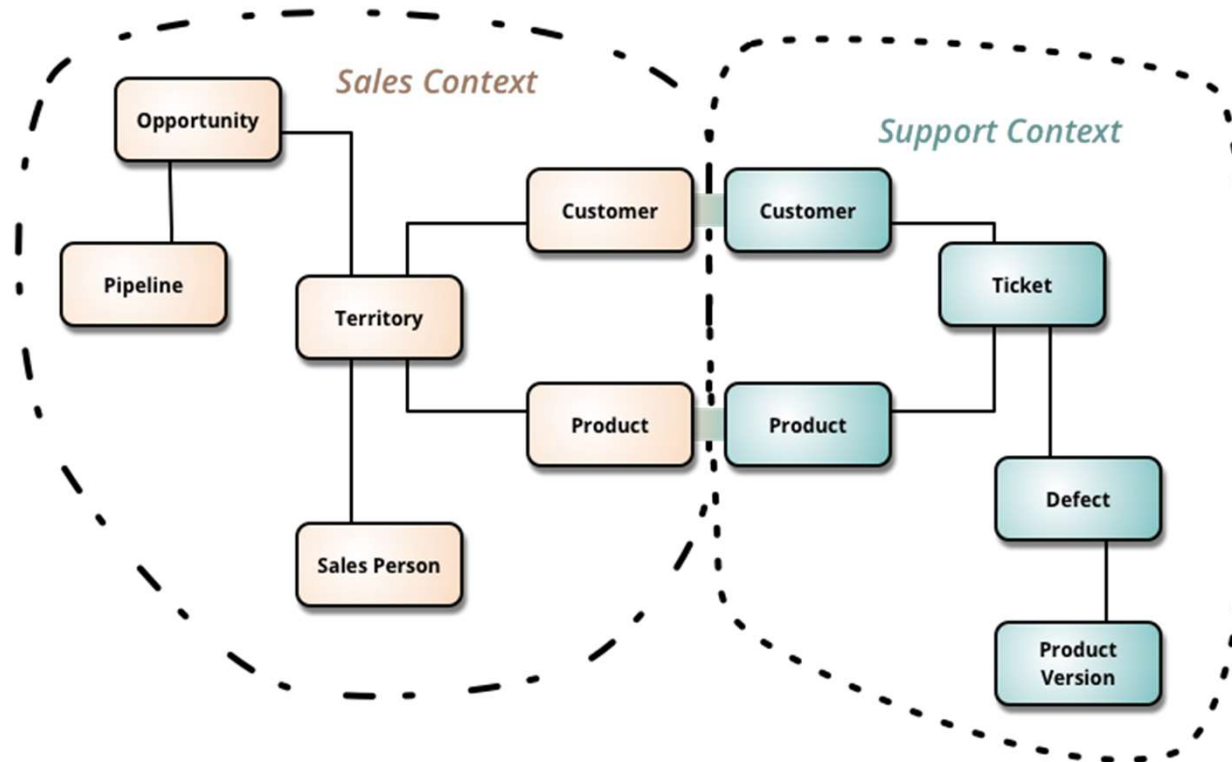


Big ball of mud

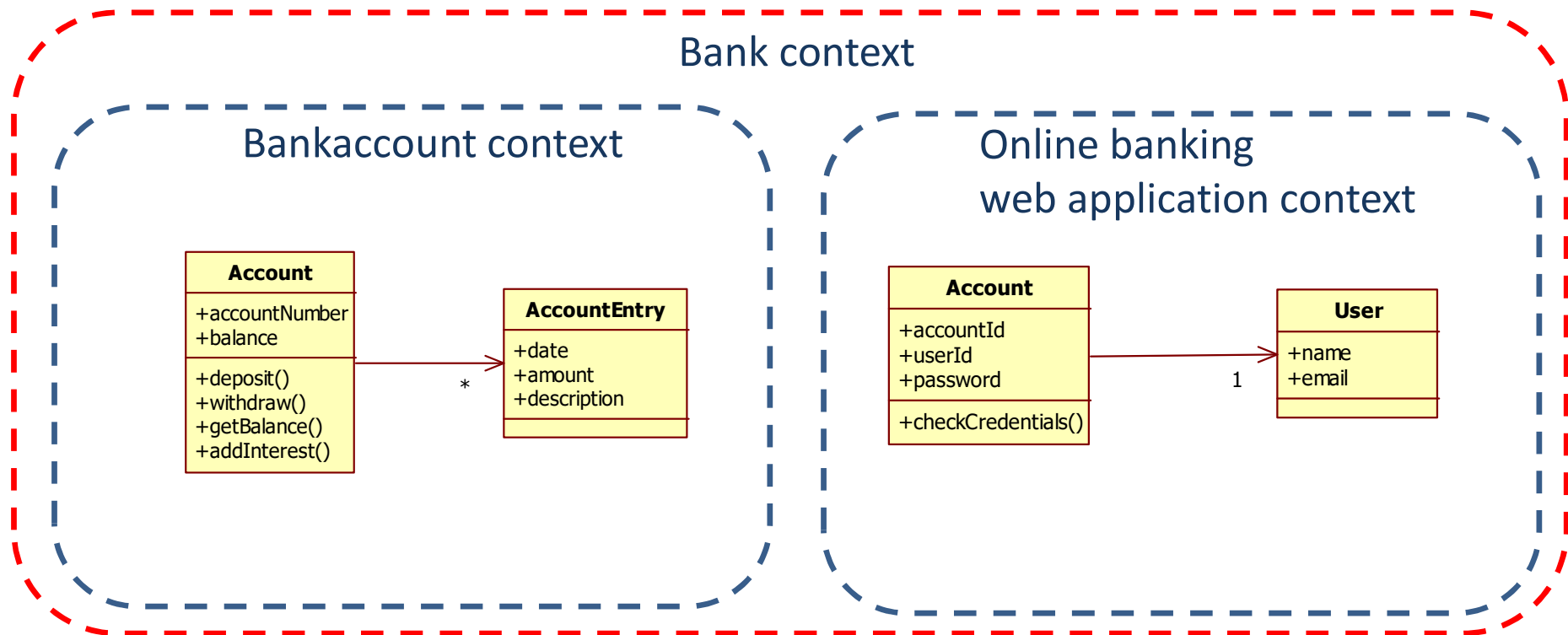


Context

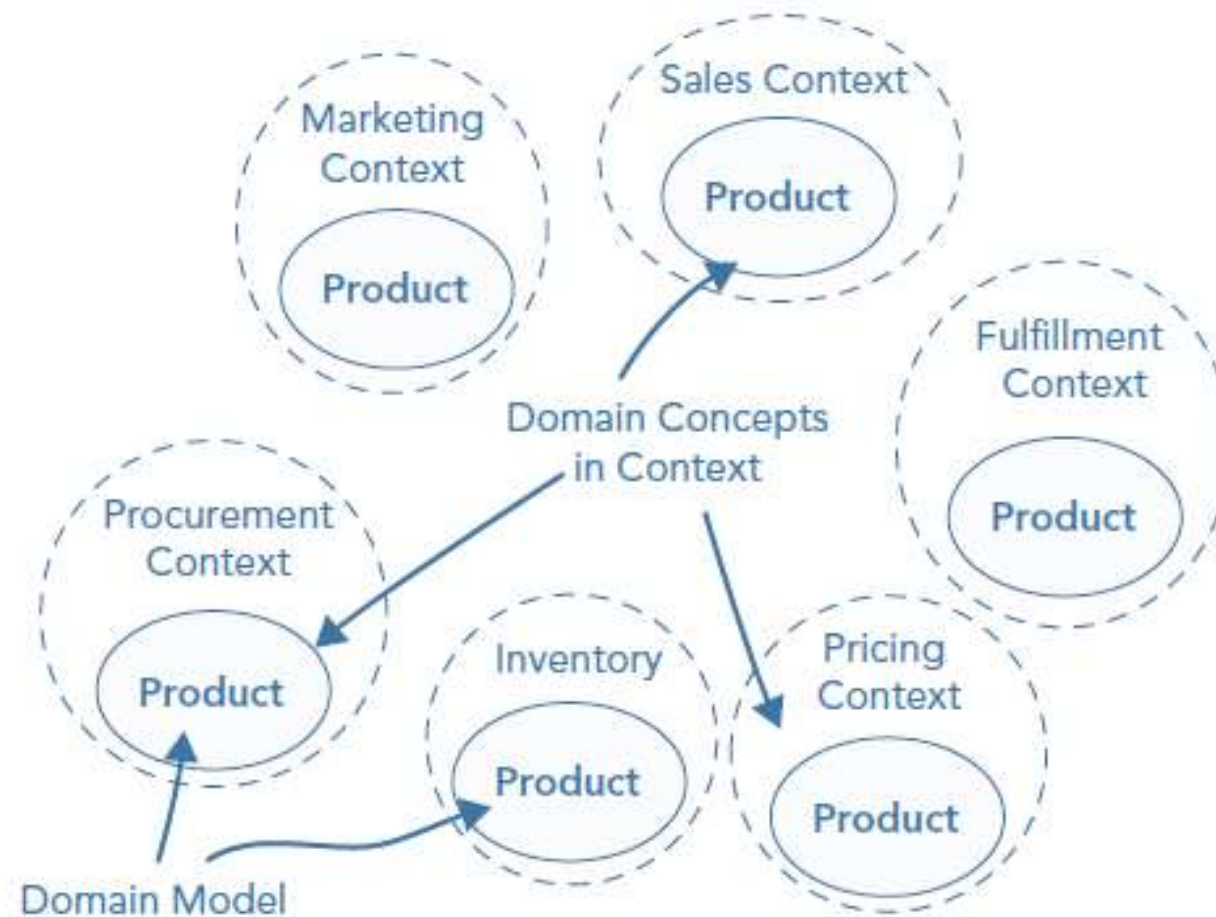
- A specific domain term may have a different definition in a different context



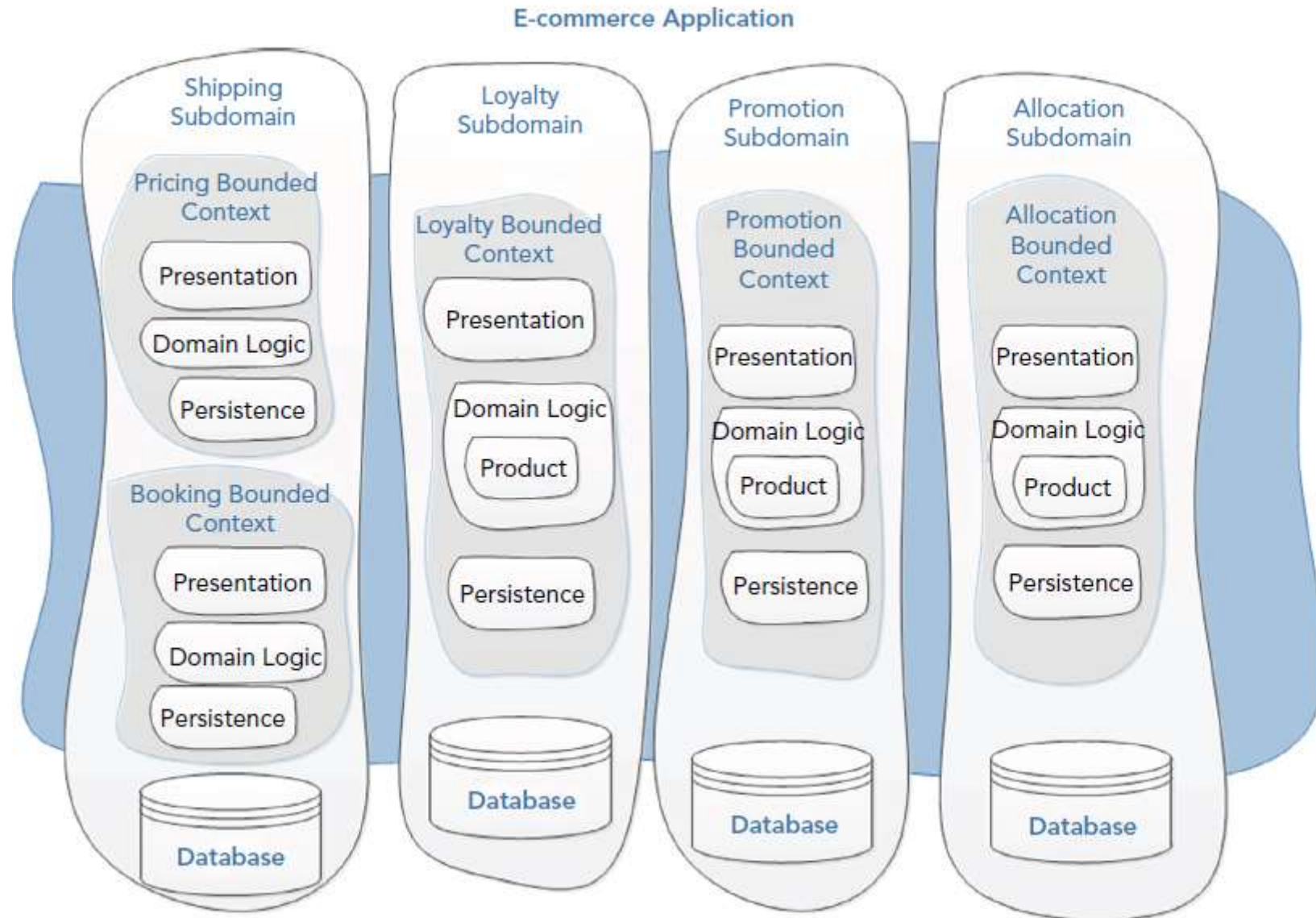
Bounded context



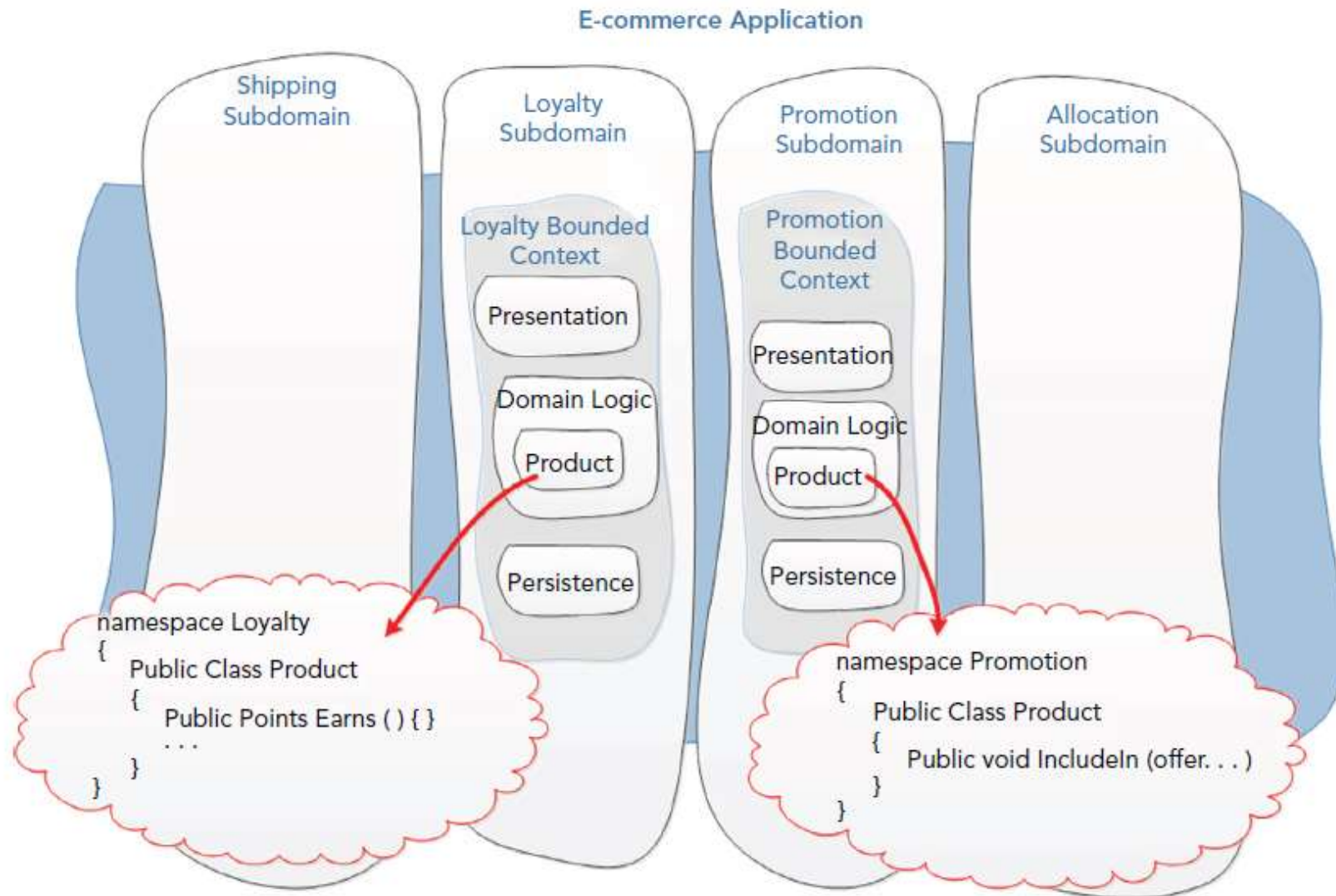
Each bounded context has its own model



Bounded context example



Bounded context example



Why DDD?

- Writing code is easy.
- Learning an new technology is fairly easy
- The hardest part of software development is to understand what the business wants and how the business works
 - Focus on those areas of the application that deliver the most value to the business
 - This greatly helps to
 - Understand the application
 - To evolve the application
 - To test the application