
agap2IT
TRAINING CENTER
Domain Driven Design
Module 1
Strategic Domain-Driven Design

Domain Driven Design

Module Overview

Getting Started with DDD

Domains, Subdomains and Bounded Context

Context Mapping

Getting Started with DDD

The Domain is the world of the business in which you are working – all of the knowledge of the company and how it operates

It contains the problems that you want to solve with code

It involves assumptions, business rules, processes and existing systems

It includes terminology that has specific meaning to the organization

It has metrics, goals and objectives that are unique to the business

Start with the Domain

The Domain Model is your solution to a problem you want to solve

Visual dictionnary of “real-world” entities, relationships and behaviors within a domain

Contains common vocabulary (ubiquitous language) for requirements and specification

Used as a communication tool for a software project

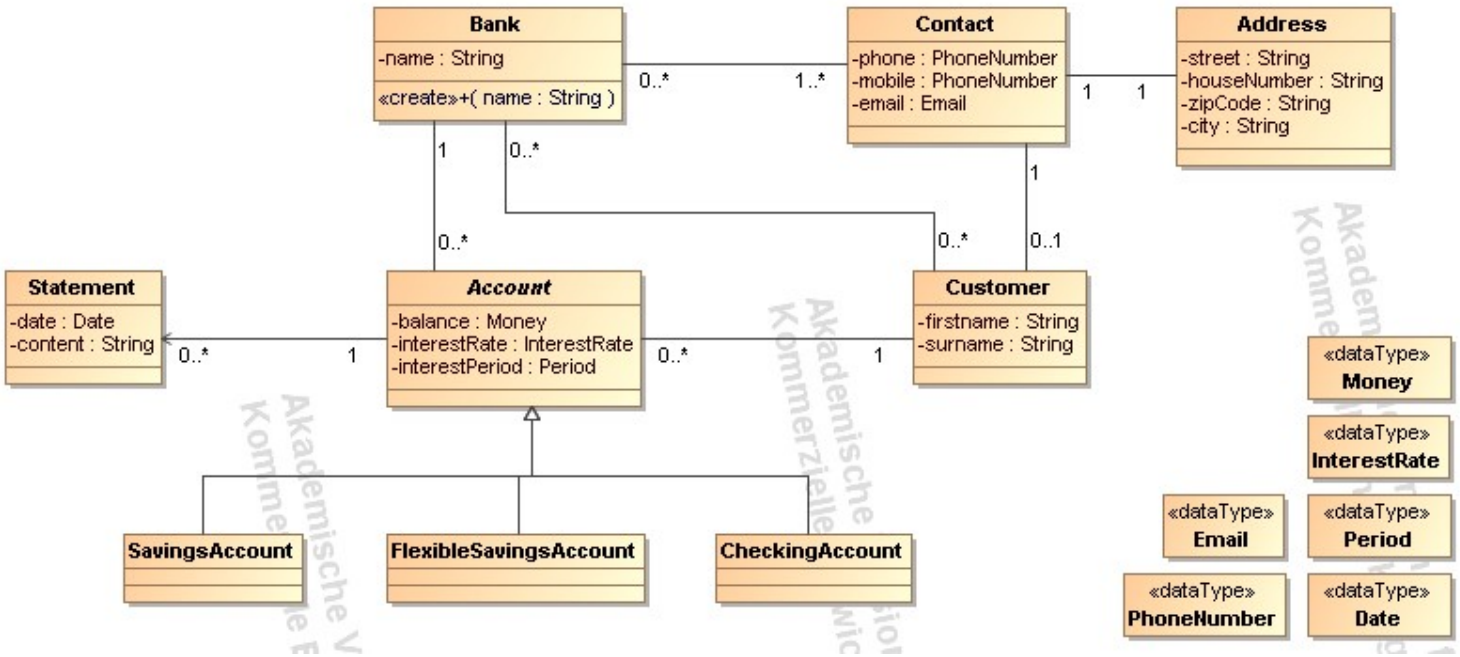
Living document - continuously distilled and refined

Key Takeaway: The Domain Model is a clear depiction of the problem to be solved along with the proposed solution

Flying Over a Domain Model

Conceptual model depicts …

- Entities (domain objects)
- Associations
- Multiplicity
- Attributes
- Inheritance
- Composition
- Aggregation



Flying Over a Domain Model

Eric Evans
DDD Author

… Which Leads to Domain Driven Design

- Set of patterns/principles for designing complex business systems
- Focus on domain and business concerns
- Initiates collaboration between technical and domain experts
- Iteratively refines a conceptual model that addresses the particular domain problems

DDD Terminology	
Entities	Key domain objects of interest
Bounded Context	Divide domain into logical subsets
Aggregates	Logical groupings of entities
Repositories	Data encapsulation

Domain Driven Design

Decomposition

You must *decompose* a large domain into smaller sub-domains

Collaborative effort involving domain experts, technologists and other stakeholders

Common strategy: Delineate by *business capabilities* (activities that generate value)

Each sub-domain should have its own data and lifecycle for change and features

Start with large domains, examine their use cases and tease out the functionality

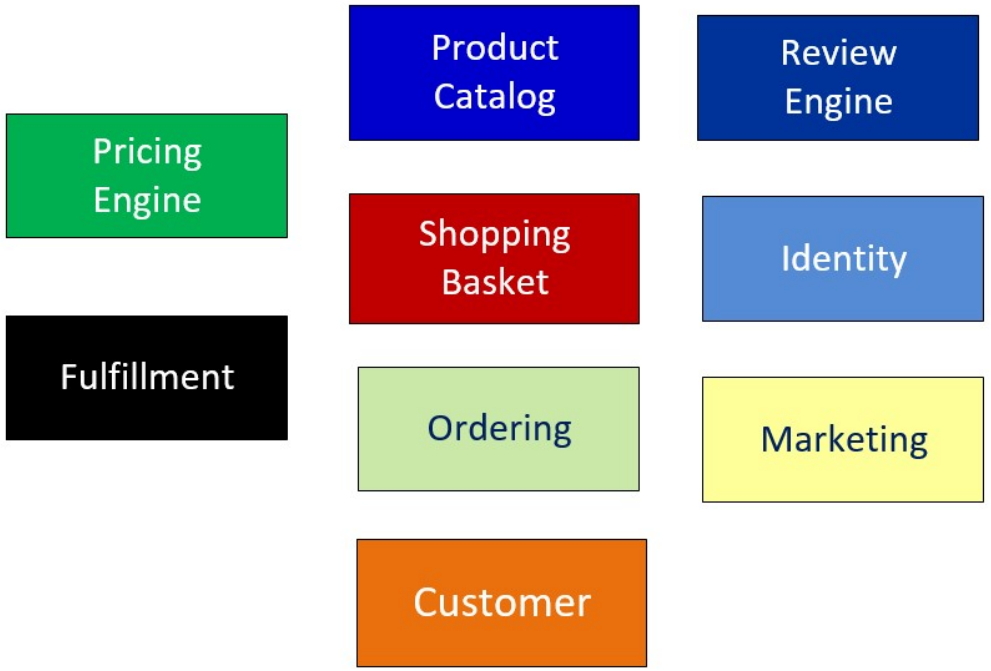
Seek out single transactional boundaries so that each service can own its data

- Other Decomposition Strategies…
  - Rate of functionality change
  - Technology used
  - Communication overhead or splitting of data
  - Error prone functionality (with express aim to improve quality and establish automated testing around these service)
  - Resource intensive and would benefit from the ability to be scaled independently

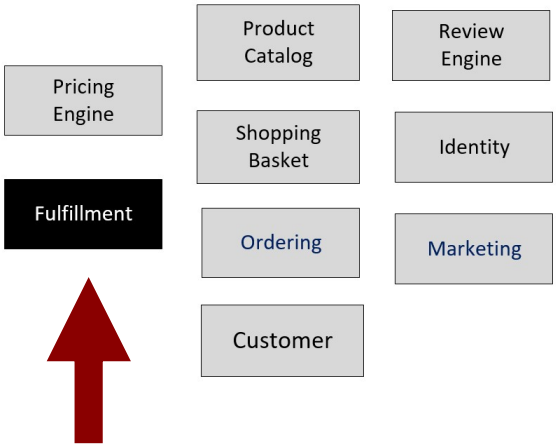
Decompose eCommerce Domain

How might we divide a large eCommerce domain into sub-domains ?

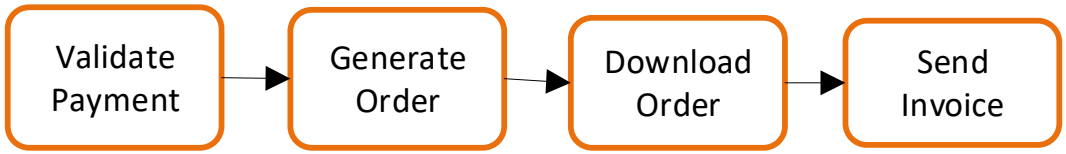
Scope out the business capabilities - activities that adds value to the business?



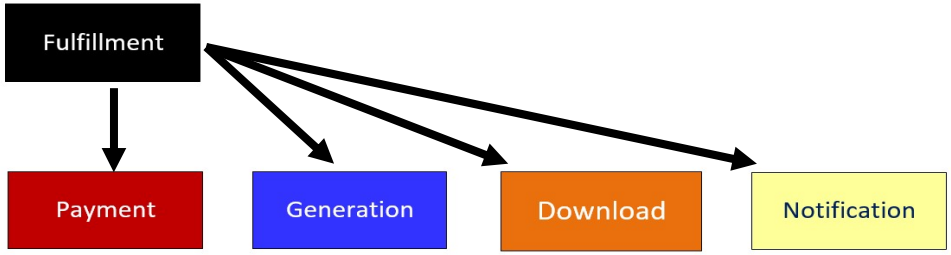
But, We're Not Done...



- Next, you'd probe, or rationalize each sub-domain
- Apply use-cases to validate the required level of granularity
- What functionality would the Fulfillment sub-domain contain?



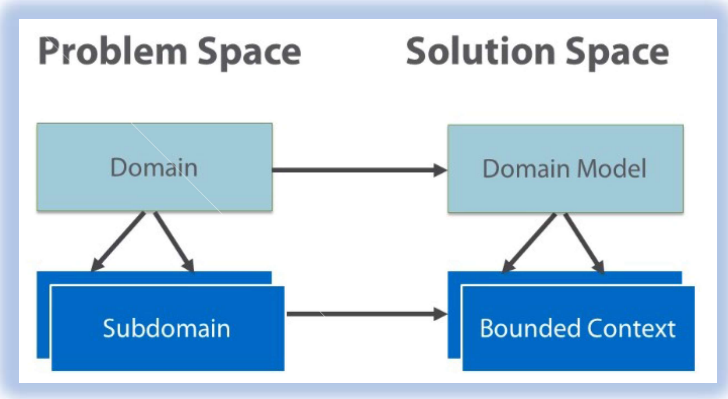
  - Does each step have its own data and lifecycle for change and features?
  - Is each a single transaction update?
- The probing process could very well result in further decomposition



Bounded Context

Eric Evans
DDD Author

- Each rationalized sub-domain becomes a Bounded Context
- Widely accepted pattern for dividing large domains
- Each Bounded Context specifies a boundary around a set of functionality that aligns with a given business capability
- Each is autonomous and encapsulates its own:
  - Sub Domain Model
  - Entities
  - Behavior
  - Ubiquitous language
  - Context, rules, code, data
  - Implementation



Single Responsibility

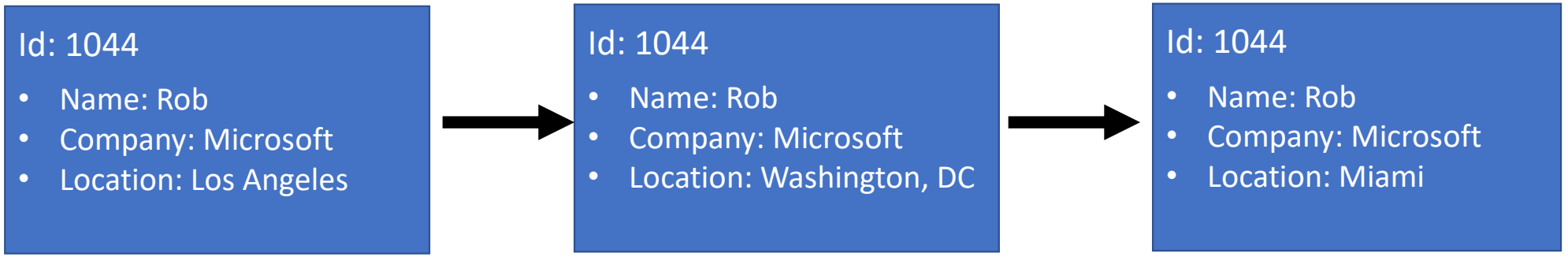
Eric Evans
DDD Author

- Each Bounded Context should do one and one thing only
- Enforces the Single Responsibility Principle (SRP)
- Each context implements one and only one responsibility, or business capability
- Each member in that context should focus on carrying out that single responsibility

Entities

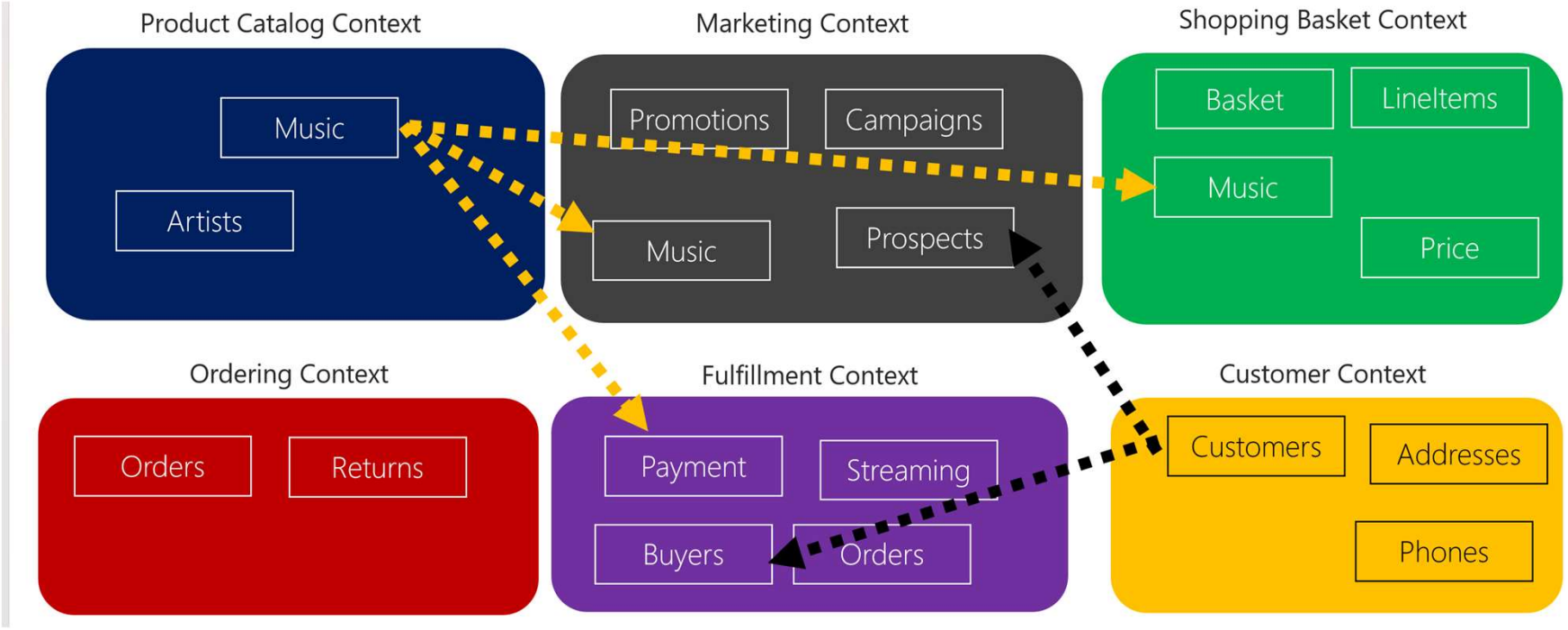
Eric Evans
DDD Author

- A bounded context contains entities…
- A key object in the domain
- Typically corresponds to a real-world business object, i.e., customer or order
- Defined by its identity
- Typically made up of state and behavior
- Attribute values can change, but Id value remains the same



Bounded Contexts

- Putting it together: Bounded contexts and entities
- Some entities will be required across multiple contexts

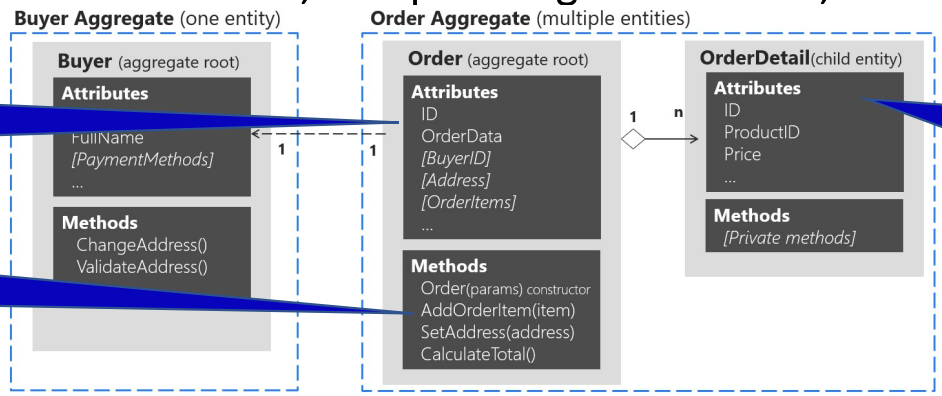


Domain Aggregates

- The Ordering Context also implements the Domain Aggregate pattern
- Groups together related entities as an aggregate, i.e., Orders and OrderDetails
- Each aggregate is self-contained, encapsulating related state, behavior and business rules

One entity is root
All others are children of root

Root exposes members that access both root and children



Cannot reference children directly
Must reference through the root

- The aggregate acts as a single unit and implements a functional boundary
- Guarantees consistency at the root level, forbidding external objects from holding references to its internal members

Who else needs some coffee ?

Domain Driven Design

Technical Training

Let’s do it