Microservice Architecture & Domain Driven Design

Amro Kudssi, MSc
AAAEA IL - President

Software Service

A service that provides a business or technical capability

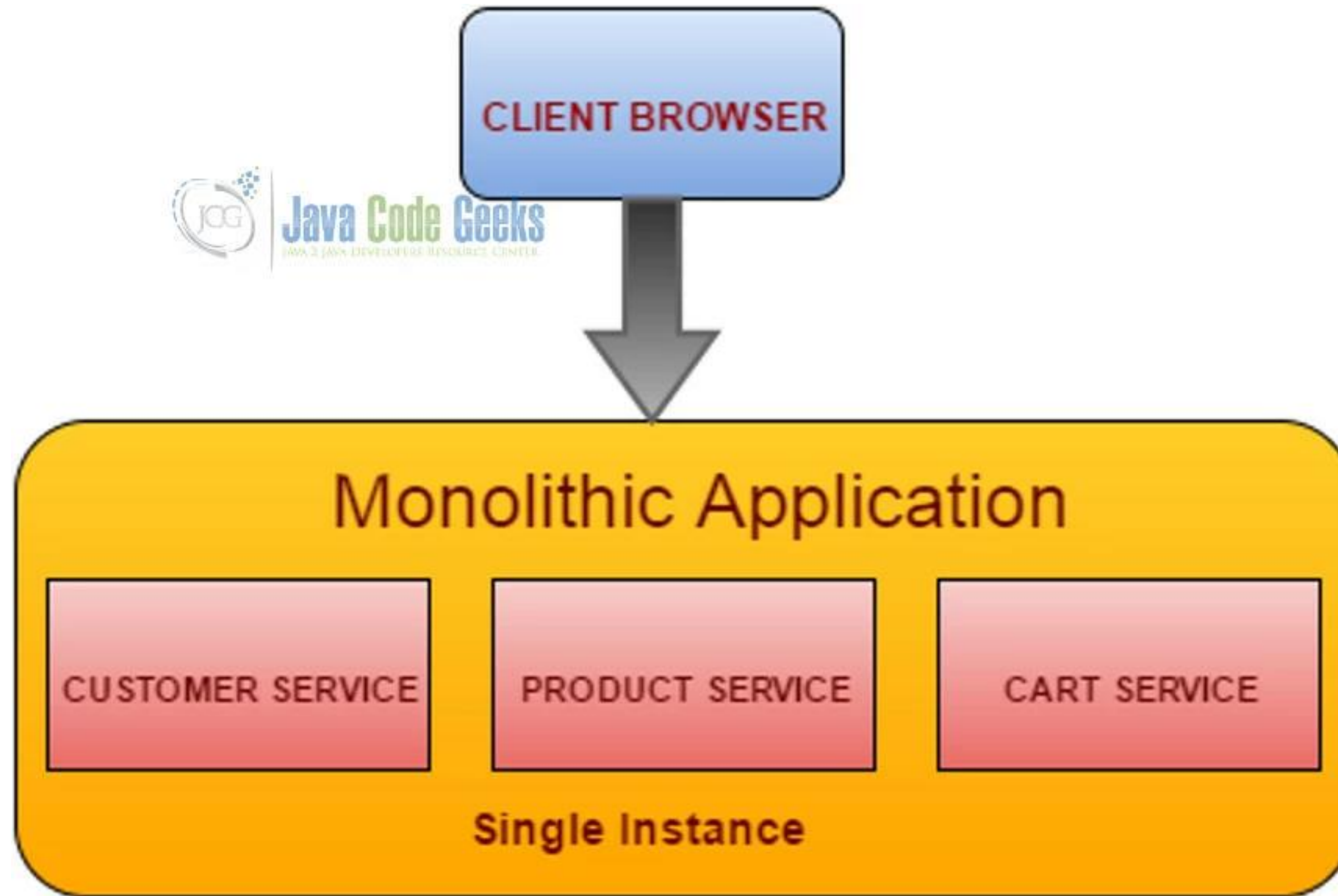
- Weather forecasting service
- Shipment tracking service
- Address searching service
- Account Transfer service
- Wire service
- Send Email
- Buy a stock
- Find a flight
- Reserve a flight
- …
- …

Software Application

An Application is a combination of software services

- Gmail
  - View list of emails
  - Read an email
  - Send email
  - Search email
  - ..
- Weather.com
  - Get current temperature
  - Get a forecast by zip
  - …

A Monolith Application



Some of Monolith Issues

Quality attributes

- Complexity
- Changeability
- Deployability
- Scalability
- Reliability
- Testability
- …

Source: www.crystalinks.com

A Microservice

- A relatively small software component that does one thing and one thing only
  - Simple to build.
  - Simple to test.
  - Simple to deploy.
  - Simple to scale.

Microservice Architecture

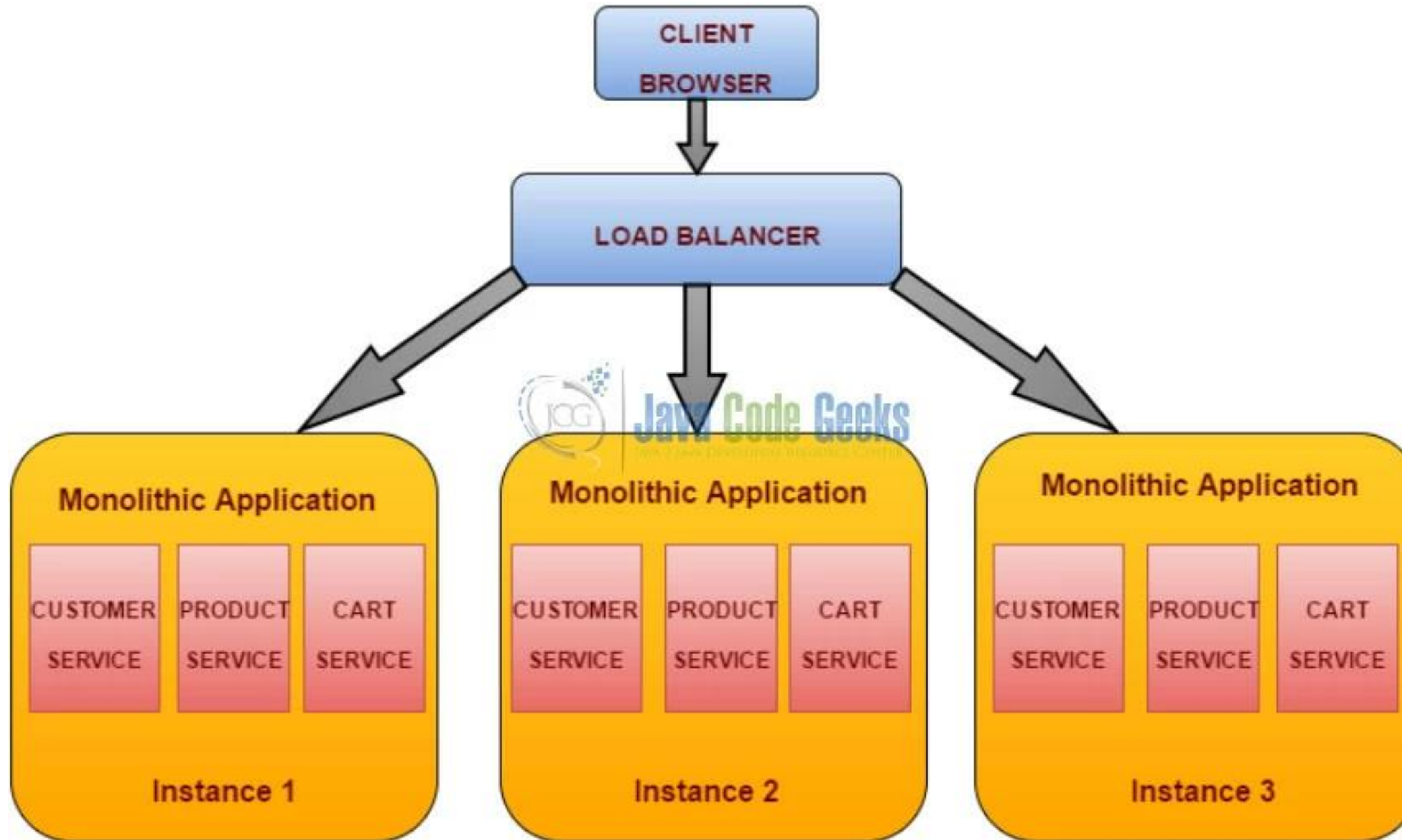
- Bunch of loosely coupled microservices that provide business capabilities
- The microservice architecture enables the continuous delivery/deployment of large, complex applications.
- It also enables an organization to evolve its technology stack.

In Microservice Architecture

Quality attributes

- Complexity
- Changeability
- Deployability
- Scalability
- Reliability
- Testability

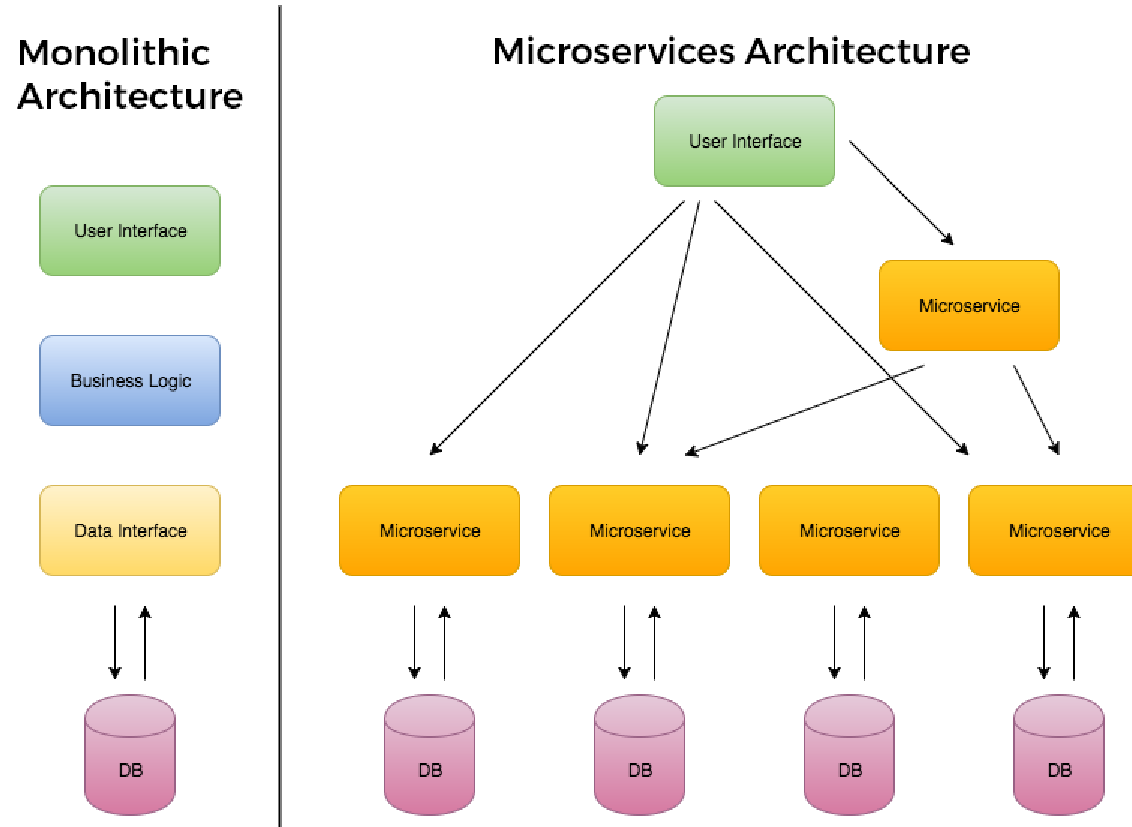
Microservice Architecture



Monolith vs Microservices

Source: www.knoldus.com

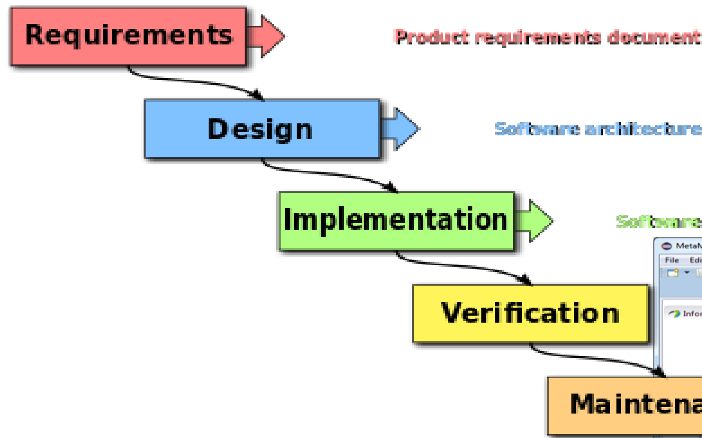
Monolith vs Microservices



Source: www.medium.com

CHICAGO TECHFEST
Hear It. See It. Do It.
October 23-24

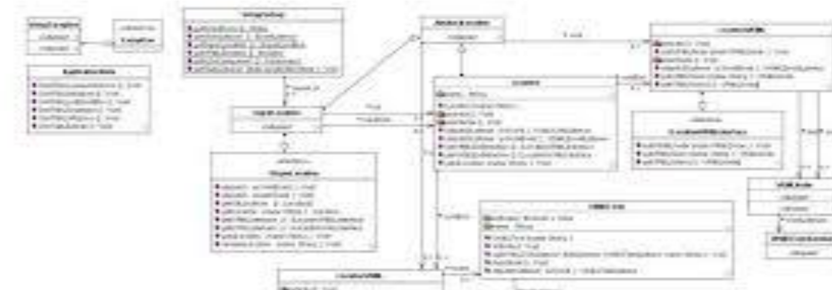
The Design Problem

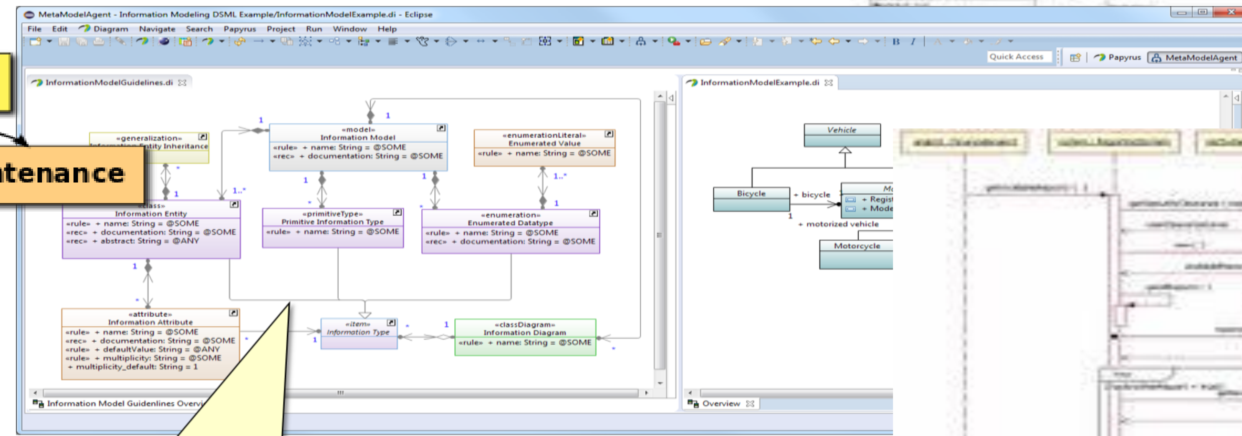


Product requirements document

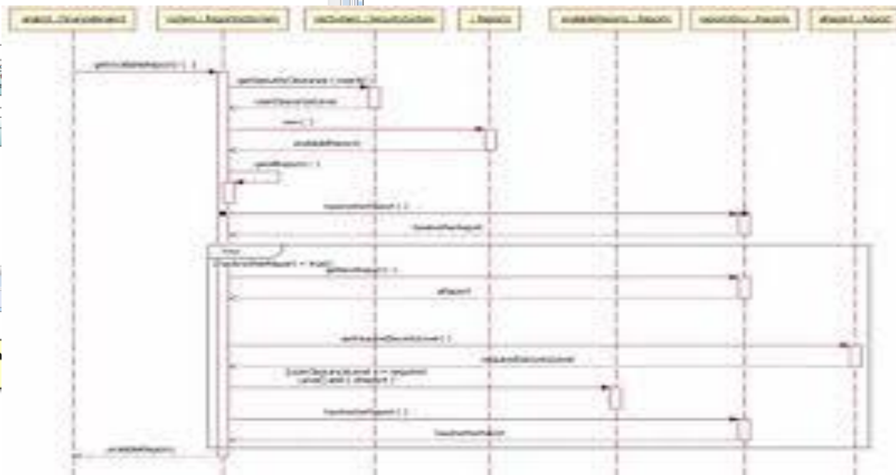
Software architecture

Software





Subset of a DSML-definition for Information Modeling in terms of a metamodel for MetaModelAgent



Example of an In metamodel and

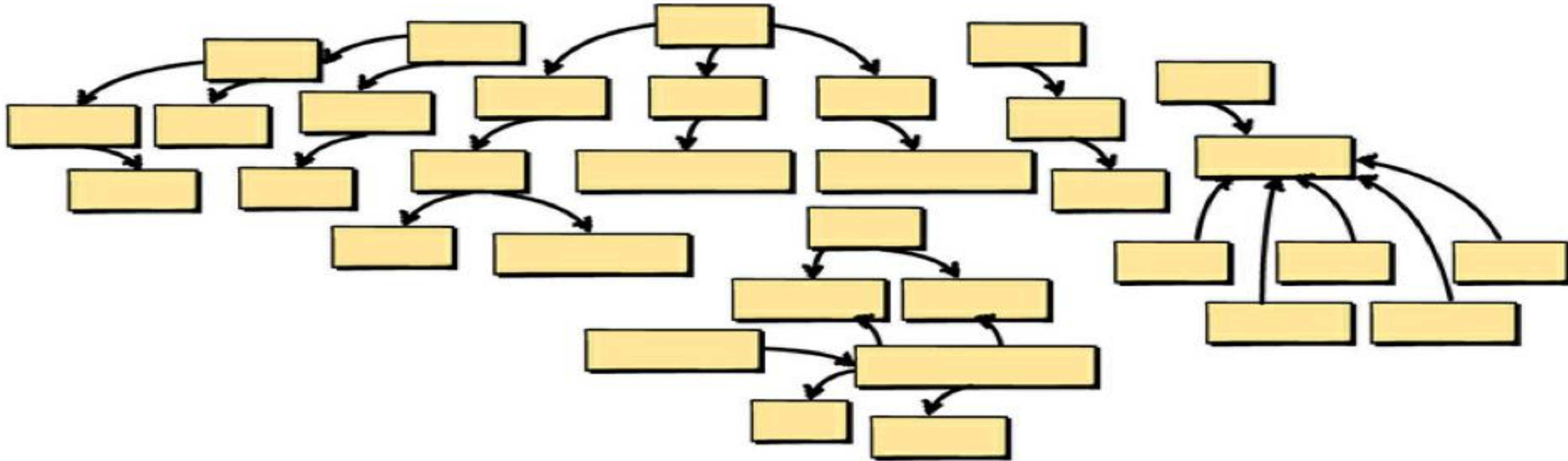
Bad Design - Business Reasons

- Software development is considered a cost center rather than a profit center.
- Producing Designs adds significant time and effort, resulting in the delay of software deliverables.
- Overly generalizing solutions rather than addressing actual concrete business needs.

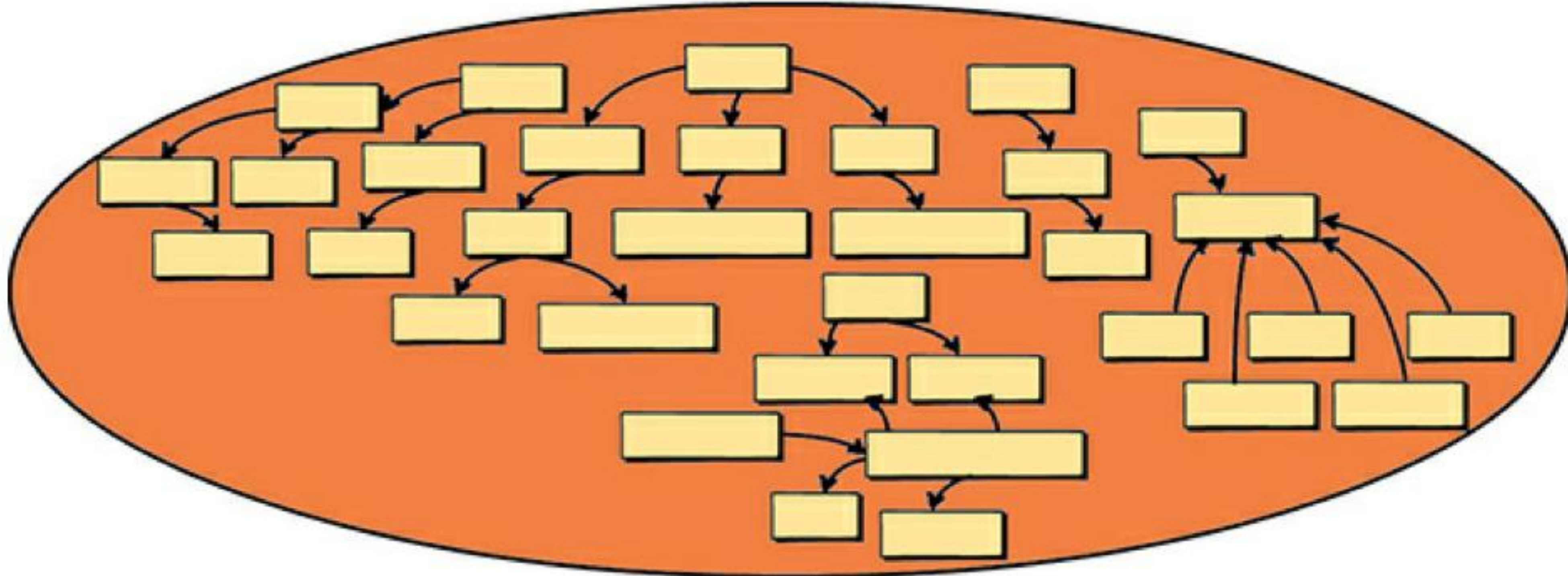
Bad Designs - Technical Reasons

- Developers are too concerned about technology.
- Developers attempt to address all current and imagined future needs
- No emphasis on naming objects and operations according to the business purpose that they fill
- The database is given too much priority.
  - Large, slow, and locking DB queries.

Traditional Design Approaches



Big Ball of Mud



Domain Driven Design (DDD)

DDD is about modeling a Ubiquitous Language in an explicitly Bounded Context.

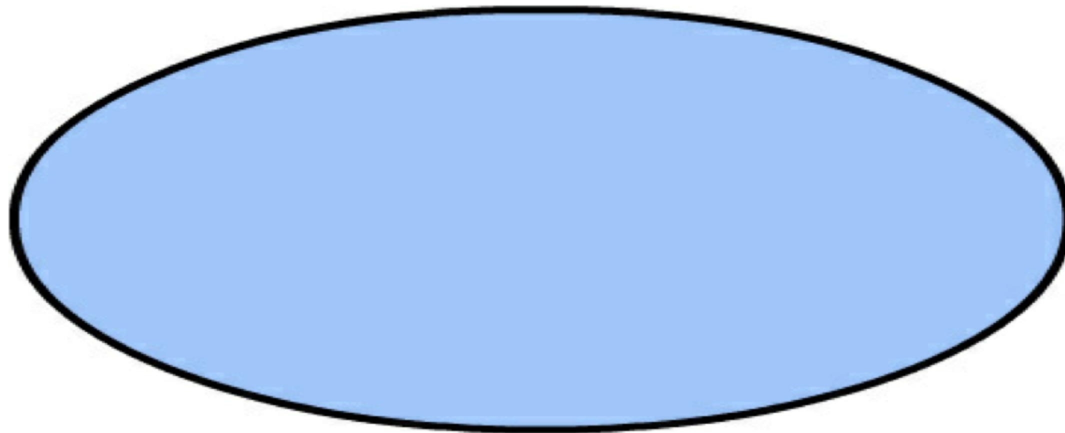
Ubiquitous Language

The language that is used between team members (developers and Domain Experts) is called the Ubiquitous Language because it is both spoken among the team members and implemented in the software model.

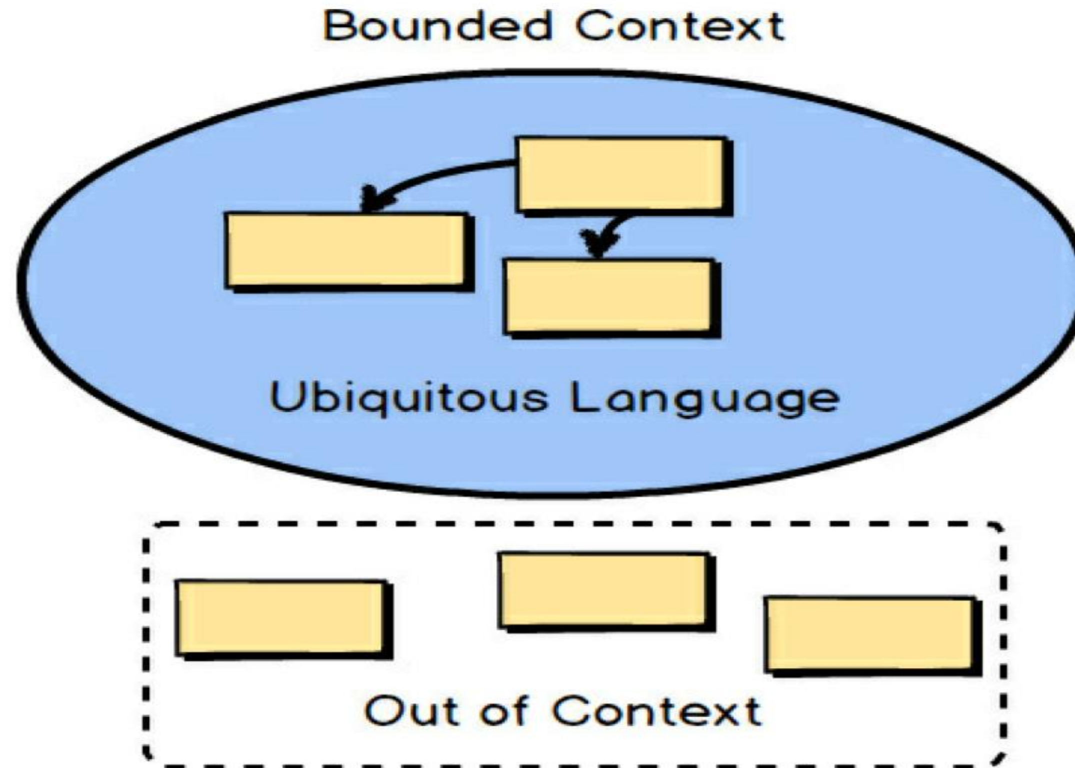
It is necessary to be **rigorous**, **strict**, **exact**, and **tight**.

Bounded Context

a Bounded Context is a semantic contextual boundary.

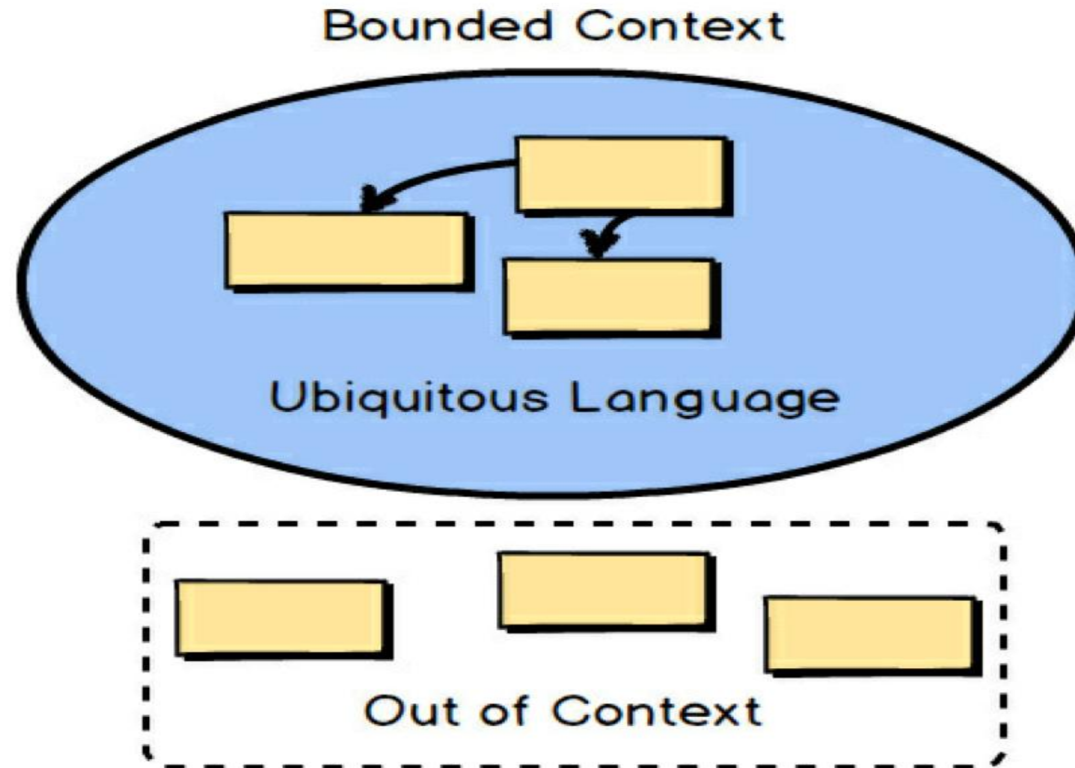


Bounded Context



Source: Vernon, Vaughn. Domain-Driven Design Distilled

Bounded Context



Source: Vernon, Vaughn. Domain-Driven Design Distilled

DDD Techniques

Event Storming

Strategic Designs Techniques

- Bounded Context and Ubiquitous Language
- Domains and Subdomains
- Context Mapping

Tactical Designs Techniques

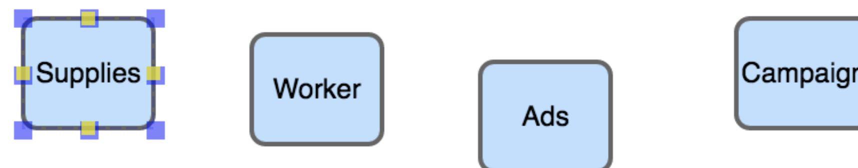
- Aggregates
- Entities and Value Types
- Domain Events & Commands

Relationship with our Agile Process

Package Shipping Business Domain

Invoice
Package
Tracking
Time
Customer
Location
Weight
Owner
Schedule
Checkout
Address
Status

Shipping Core Domain



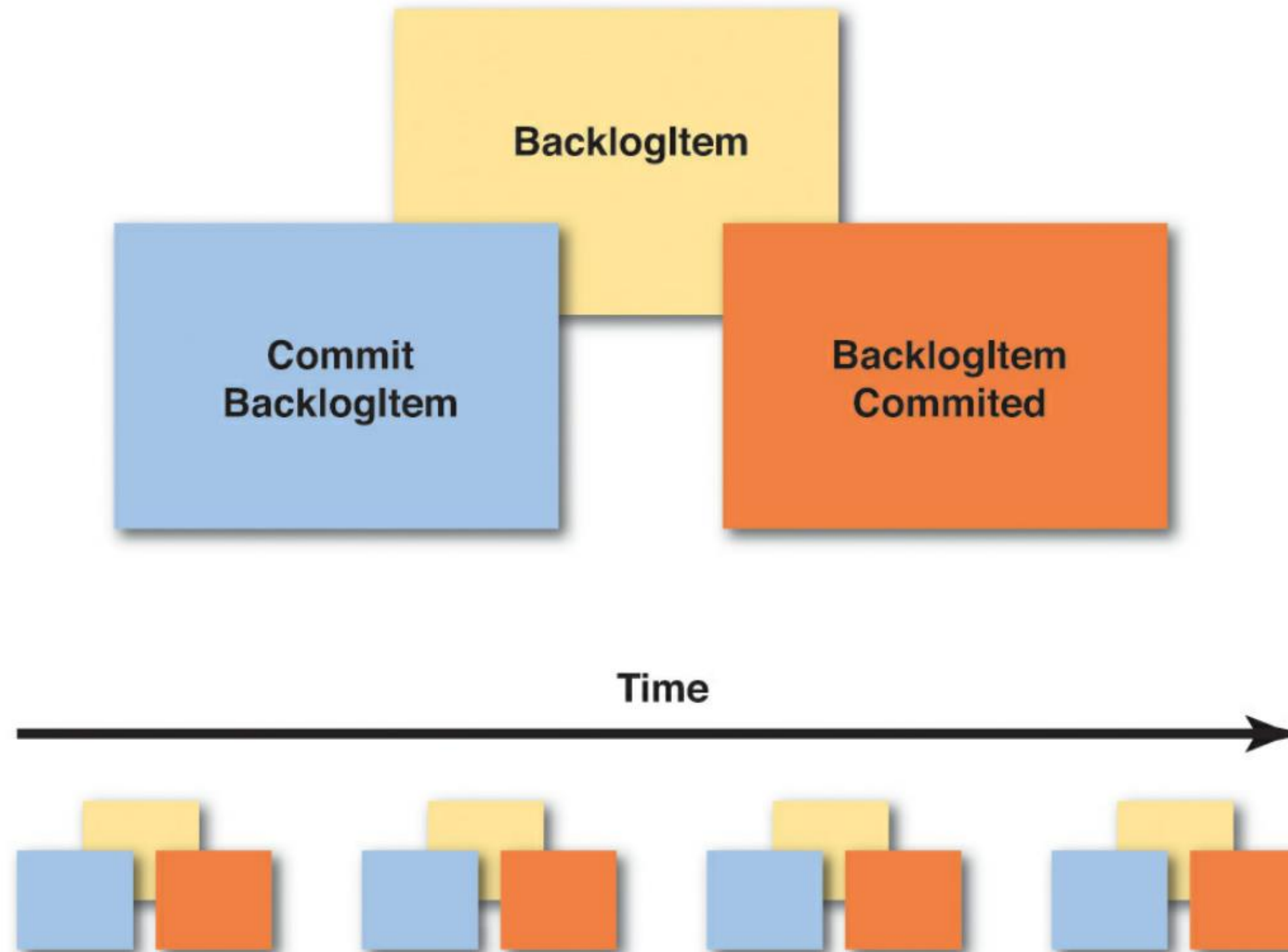
Domains and Sub-domains

- Enterprise Applications usually consists of multiple Domains
- The organization strategic initiative Bounded Context is called the ***Core Domain***.
- Other surrounding domains are called ***Sub-Domains***
- a Subdomain is a sub-part of the overall business domain.
- Subdomains can be used to logically break up the whole business domain to simplify problem space on a large, complex project.

Event Storming

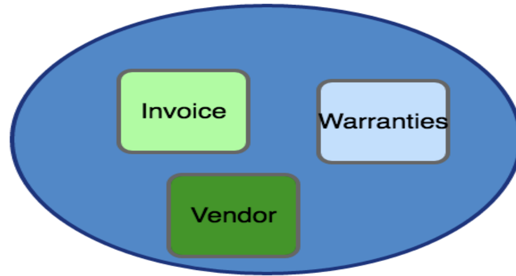
- Rapid design technique engages *Business* and *Tech* in rapid learning process to define business process(s)
- Using sticky notes & markers everyone focuses on creating Business Events. Wide wall or long role of paper
- Event Storming:
  - Storm out the following from left to right and in following order
    1. Business domain Events Past tense verb (Orange stickies)
    2. Before each event place Command that caused the event (Blue sticky)
    3. Any process that is caused by Event or a Command should be placed and connected with arrow (Lilac sticky)
    4. Identify any special roles needed to any command
    5. Start forming the Entities and Aggregates around your corresponding Commands (Yellow sticky)
    6. Draw boundaries around aggregates creating your core domain and other subdomains

Event Storming

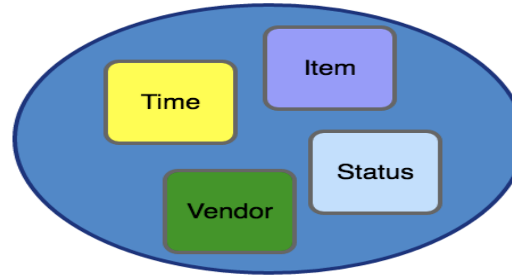


Domains and Sub-domains

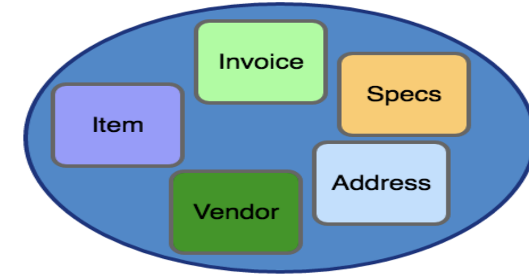
CHICAGO TECHFEST
Hear It. See It. Do It.
October 23-24



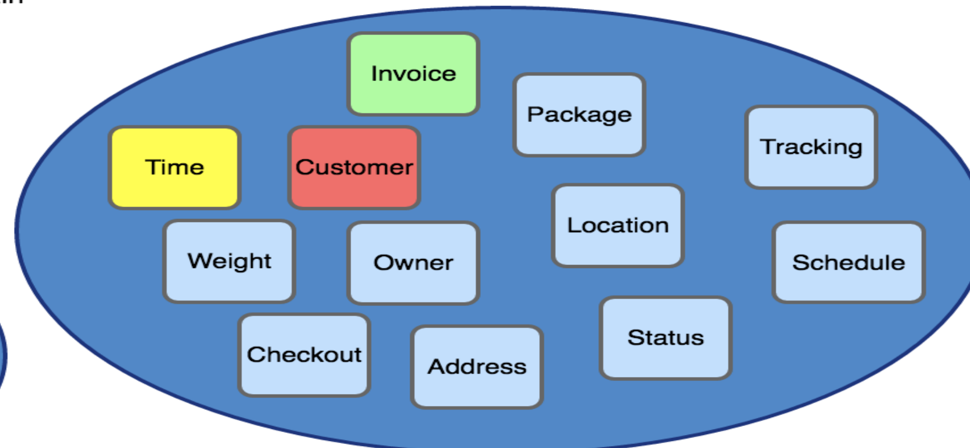
Fleet Maintanance Domain



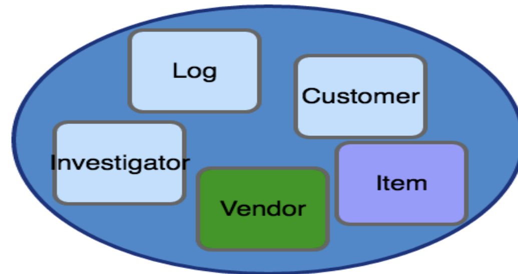
Audit Domain



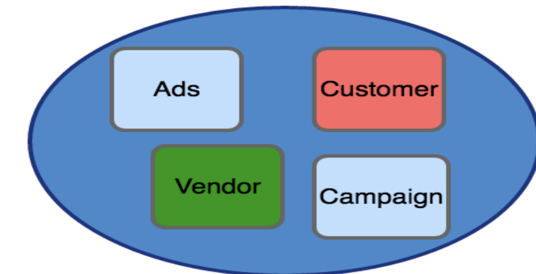
Suppiles Procurement Domain



Shipping Core Domain



Package Investigation Domain



Marketing Domain

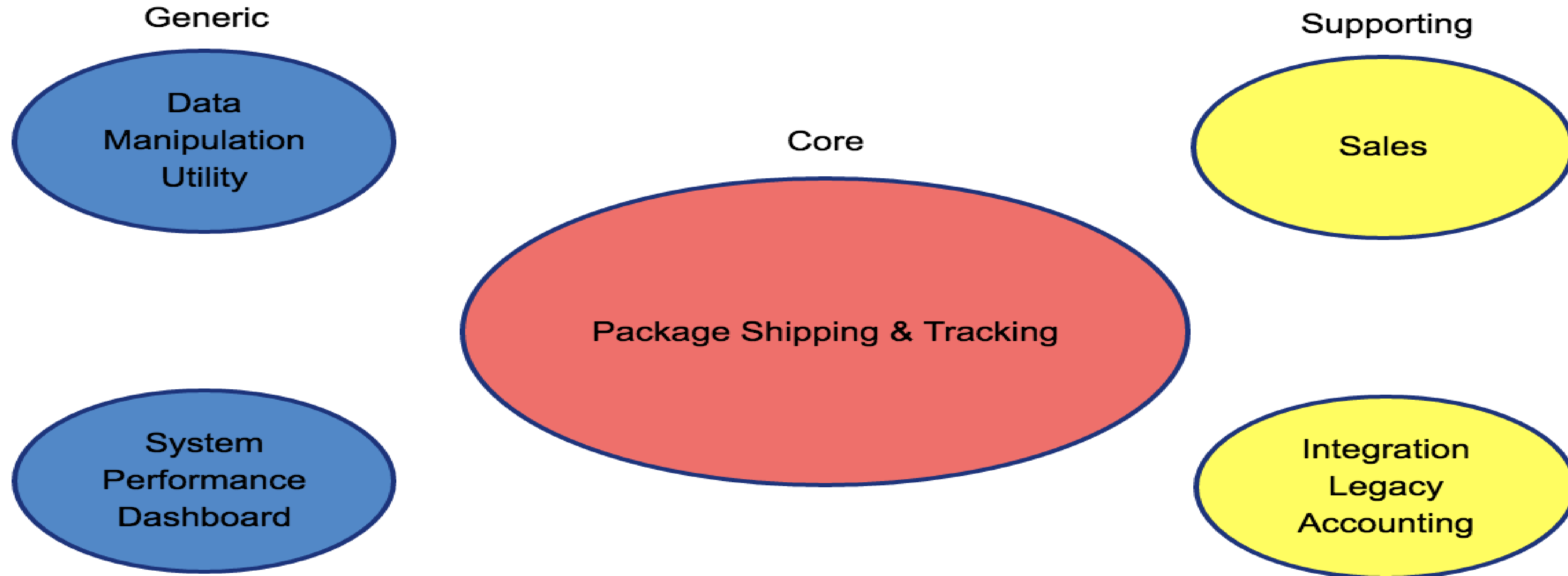
Subdomain Types

Core Domain: Strategic Investment, Core business , Elite developers, Competitive edge.

Supporting Subdomain: Supports Core Domain. Custom development because an off-the-shelf solution doesn't exist. Not the same investment of the Core Domain.

Generic Subdomain: Could be available for purchase off the shelf but may also be outsourced or even developed in house by a team that doesn't have elite developers

Subdomain Types



DDD Tactical Design Techniques

Entities vs Value objects

Aggregates

Domain Models

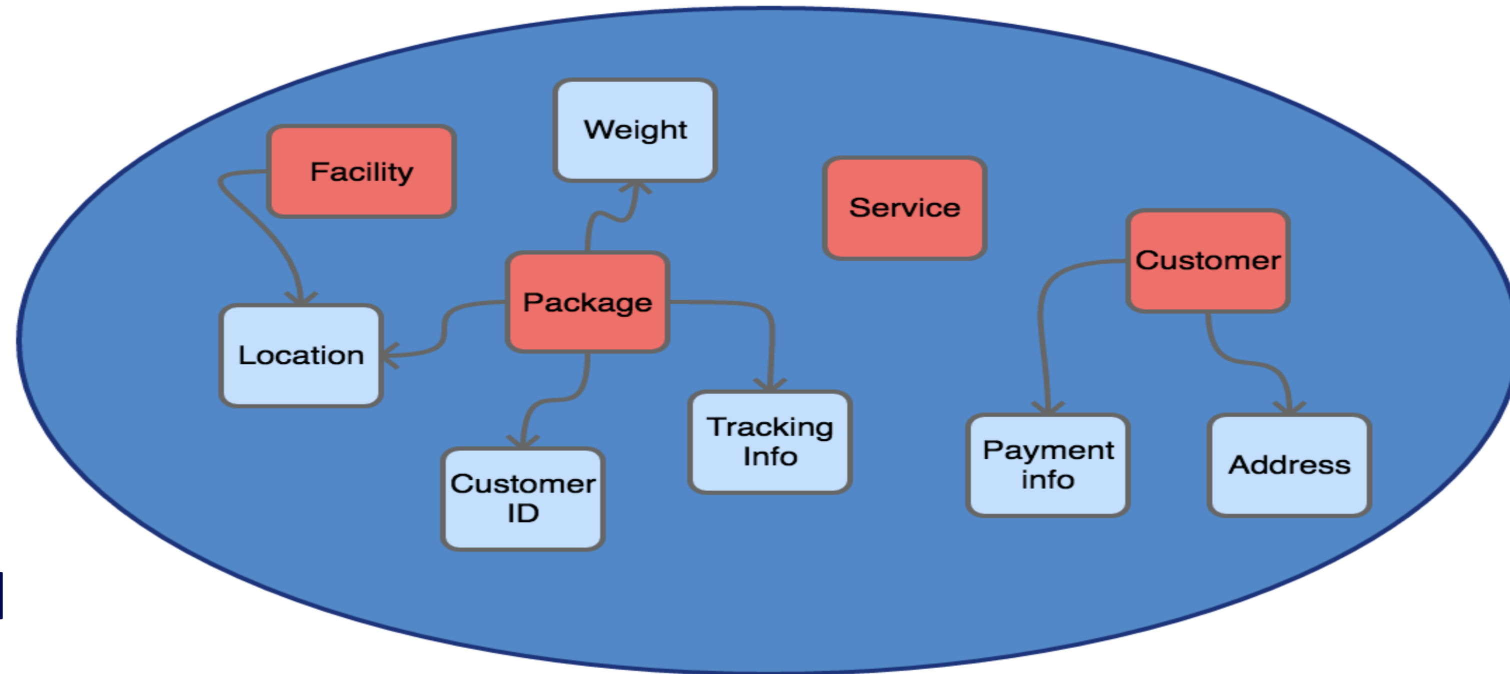
Domain Events

Entities and Value Objects

Entity is a unique individual thing in the domain that can be distinguished from other Entities that are of Same or Different types

Value Object is an immutable value that describes the Entity. It does not a thing and not unique

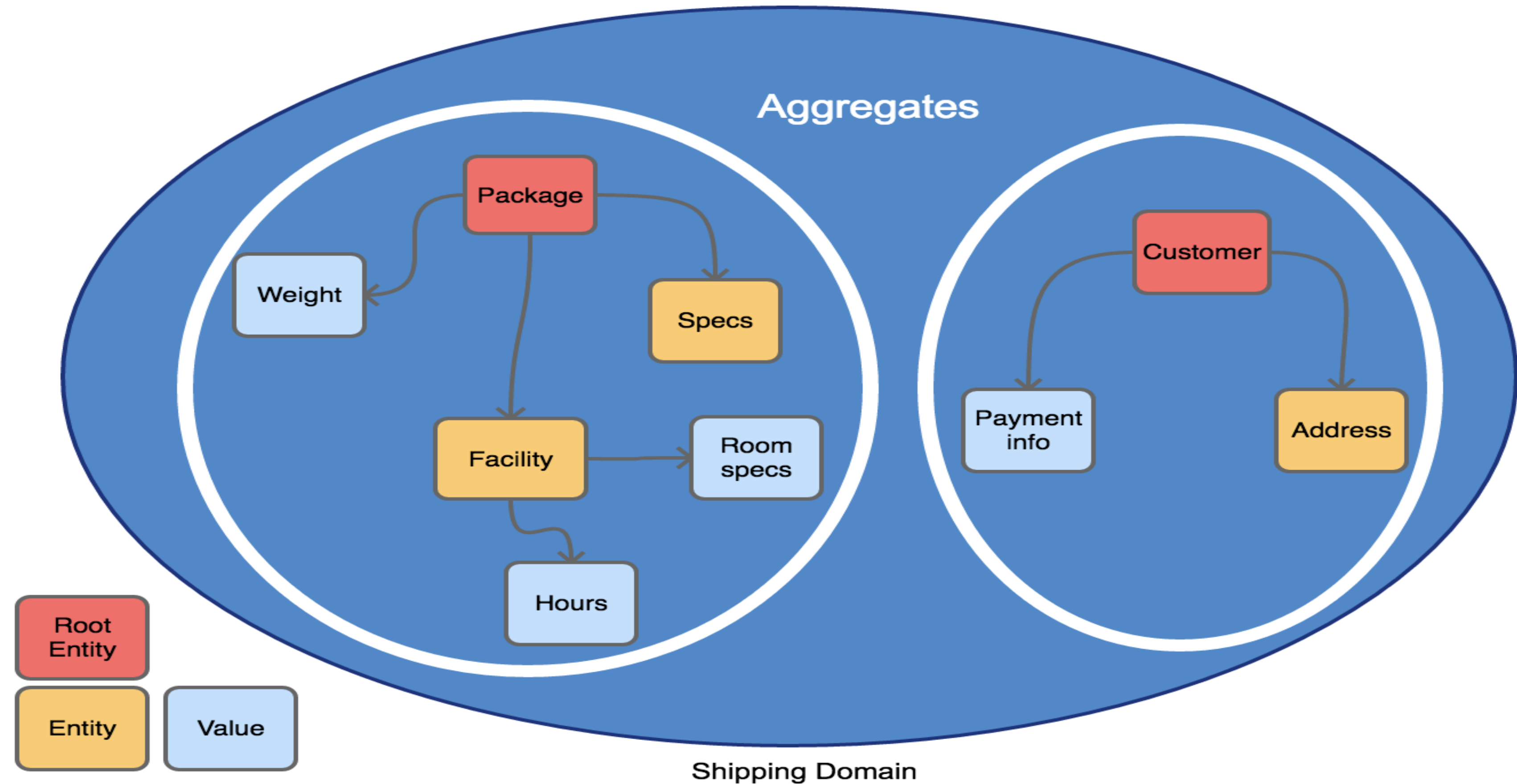
References to Entities in Other Aggregates



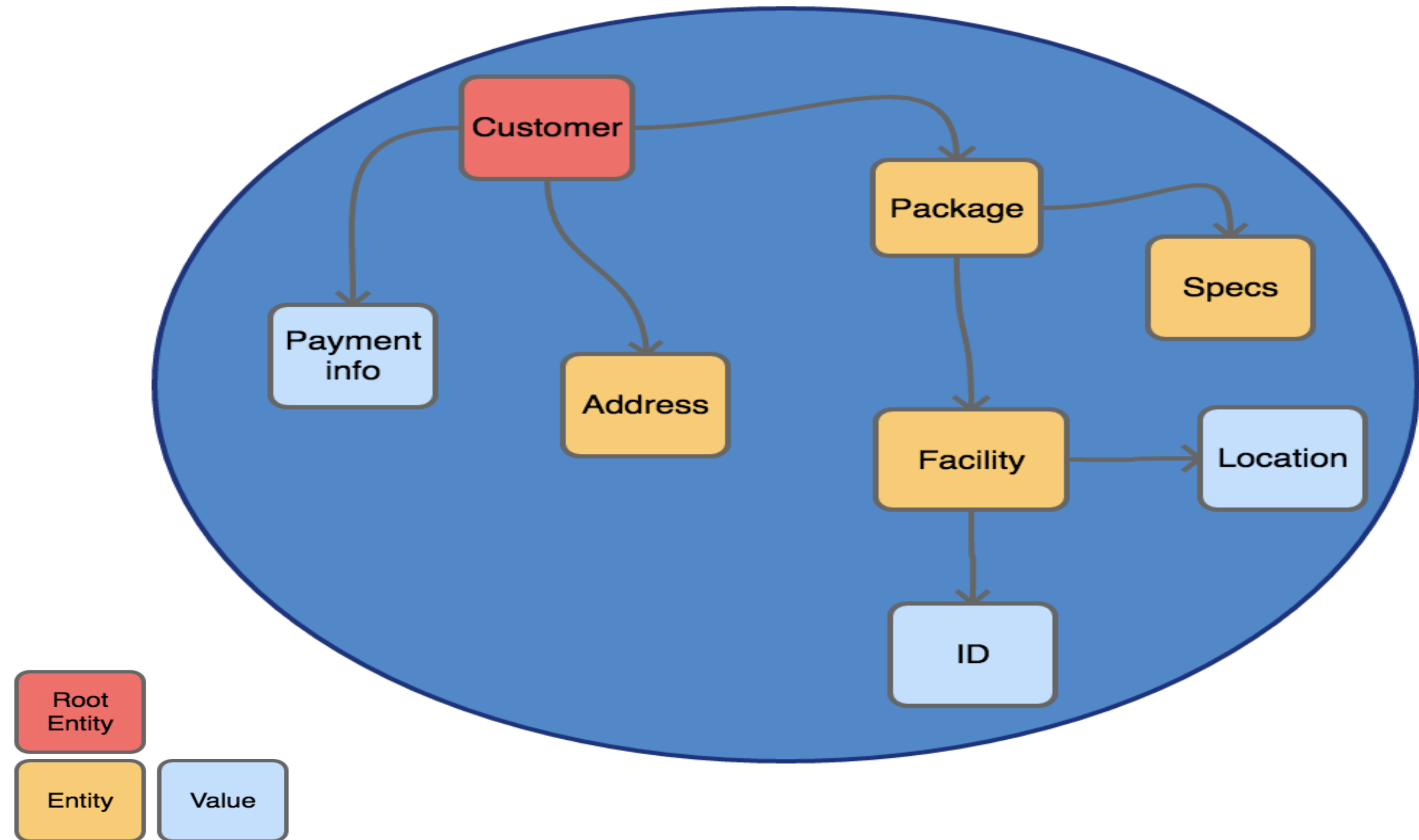
Package Shipping Domain

Aggregates

- Aggregates are clusters of objects
- Each Aggregate has Root Entity that owns all elements
- Aggregates forms Transactional Consistency boundaries
- Aggregates update in single business transaction



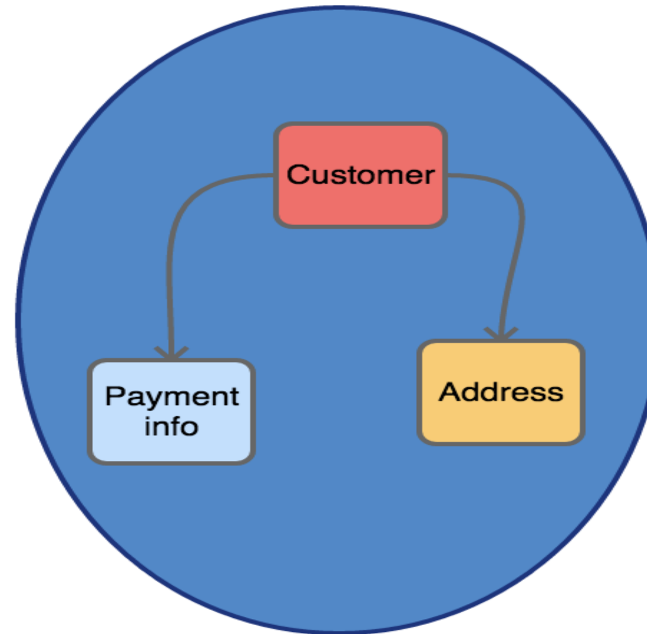
Aggregate Rules

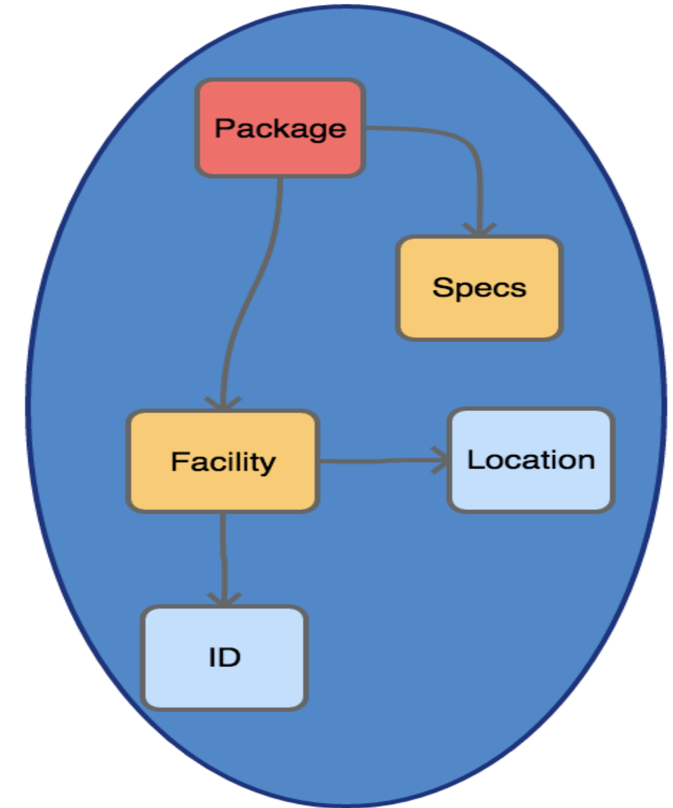


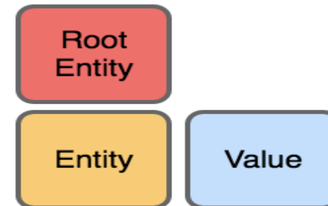
Aggregate Design Rules -1

Design Small Aggregates

-More transaction Success
-Single Responsibility Principle



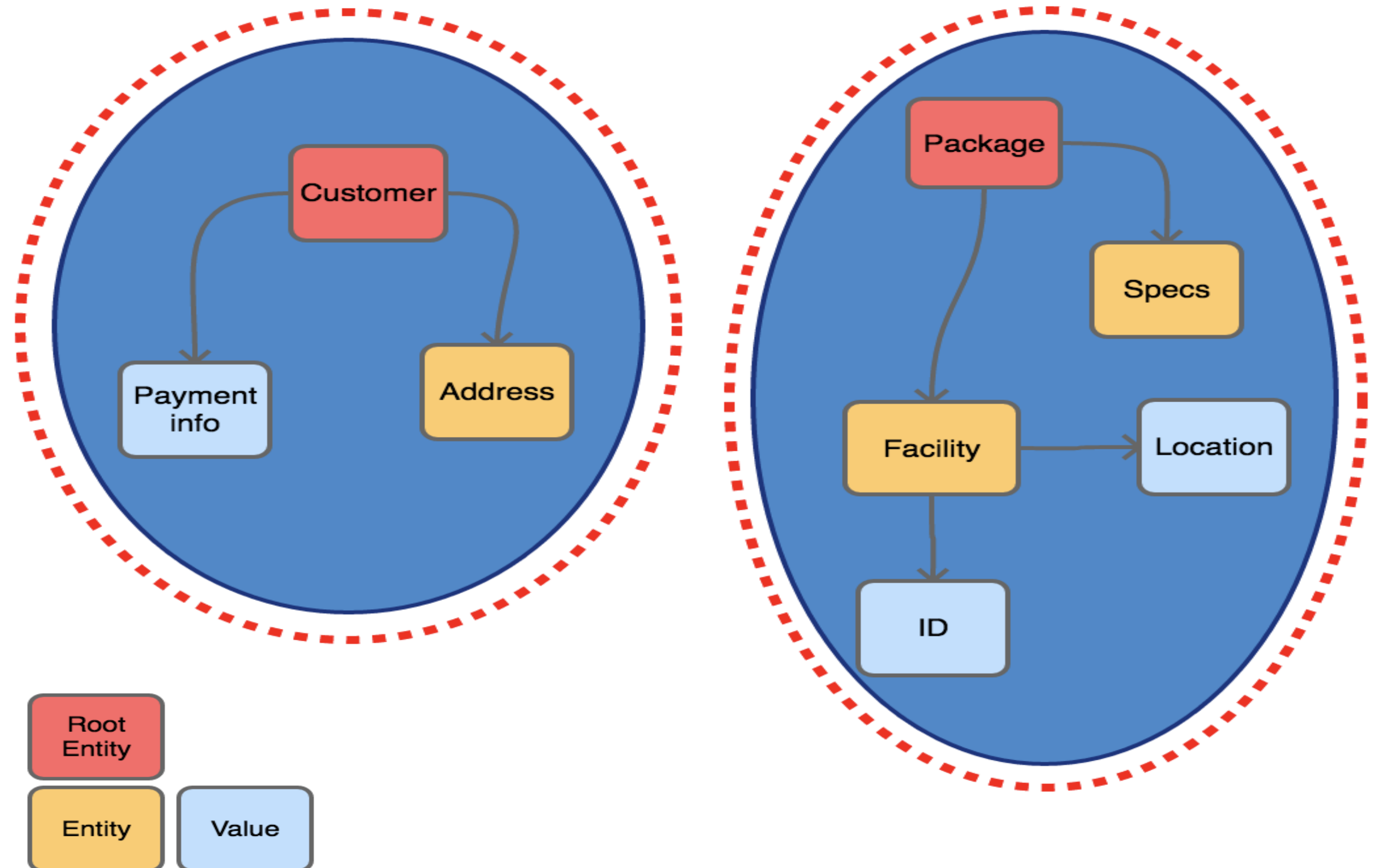




Aggregate Design Rules -2

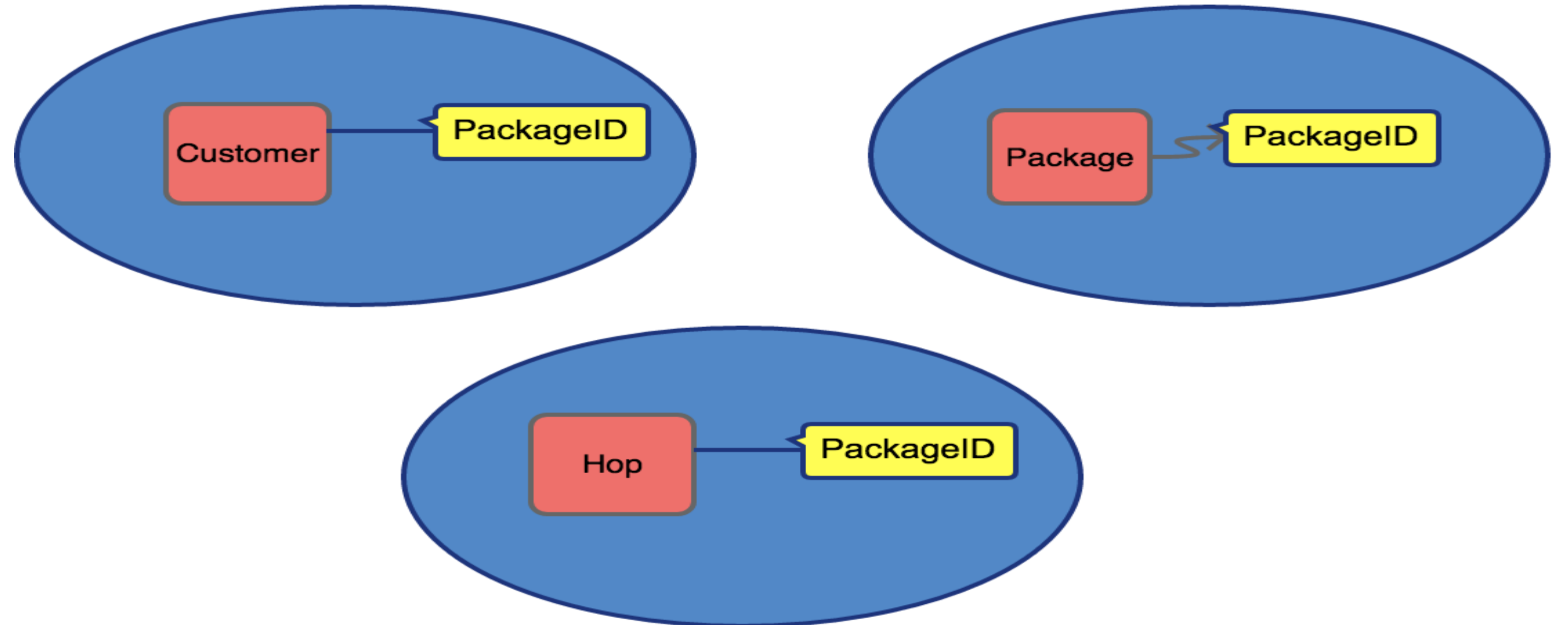
CHICAGO TECHFEST
Hear It. See It. Do It.
October 23-24

Protect Business Invariants inside Aggregate Boundaries



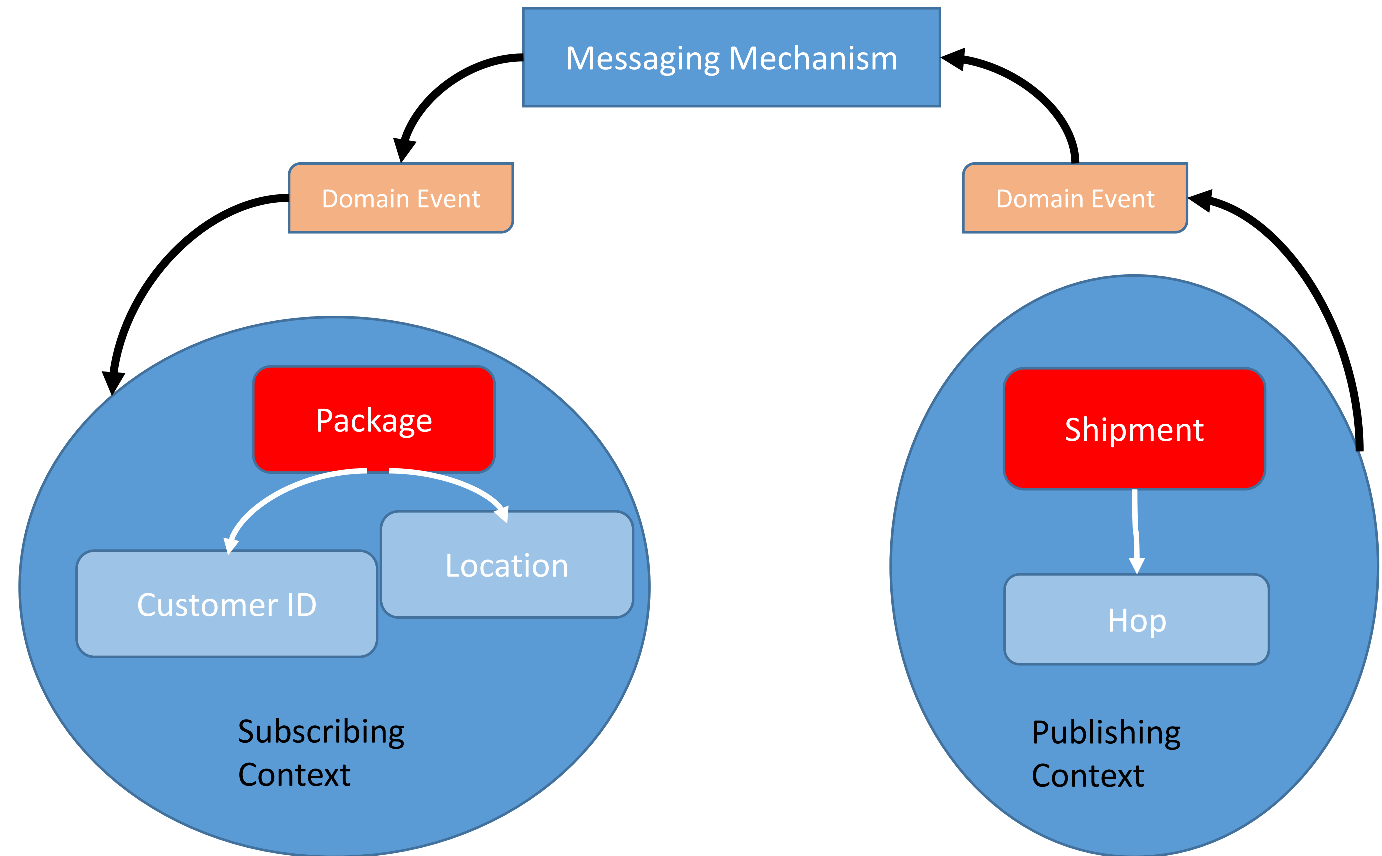
Aggregate Rules -3

Reference other Aggregates by identity (key) only.

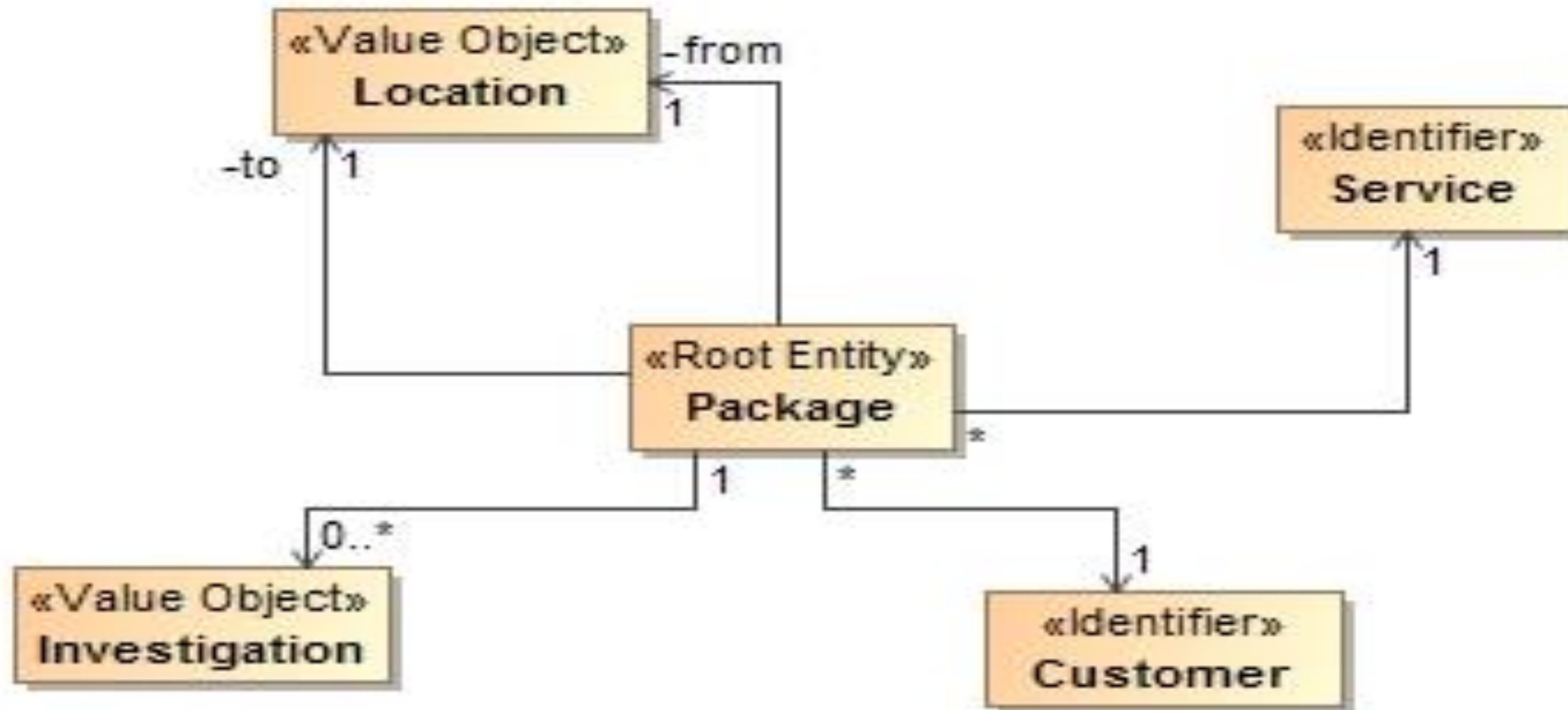


Aggregate Rules -4

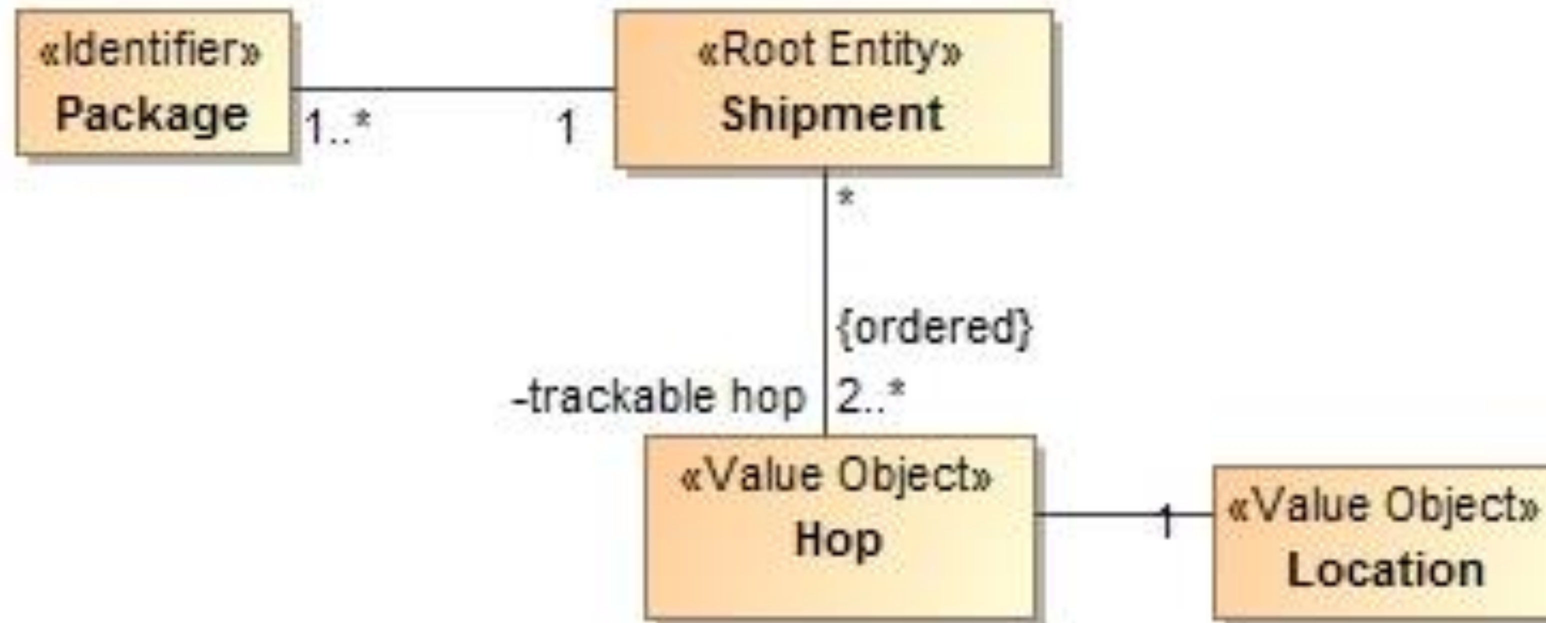
Update other Aggregates using Eventual Consistency



Domain Model



Domain Model



Messages for each Aggregate

Bounded Context: Sales

Customer: View Service Rates (in: Package, From Location, To Location out: Services)

Customer: Buy Service (in: Customer, Package, From Location, To Location, Service out: Package Id)

Customer: Report Missing Delivery (in: Package Id, out: Investigation Id)

Bounded Context: Delivery Tracking

Customer: Receive Delivery Status (in: Service Id out: trackable steps: List::Hop)

Bounded Context: Shipping Context

System: Create Shipment (in: Package Id, Collection Location, Delivery Location, Hops {ordered set} out: Shipment Id)

System: Register Package (in: Shipment Id, Package Id out: Shipment Id)

Courier: Deliver Package (in: Shipment Id, Package Id, Digital Signature)

Bounded Context: Service Ref Data

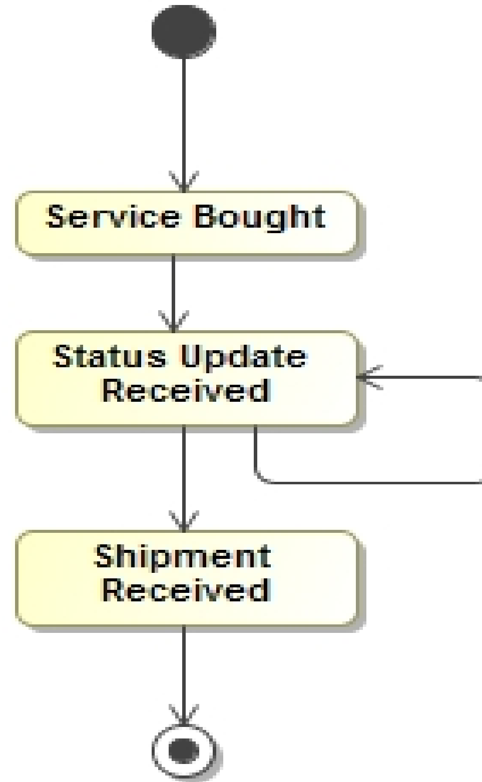
System Administrator: Maintain Hop Metadata (in: Location, Facilities out: Hop Id)

System Administrator: Maintain Service Type (in: Type, Price out: Service Id)

Define a State Machine for each Aggregate

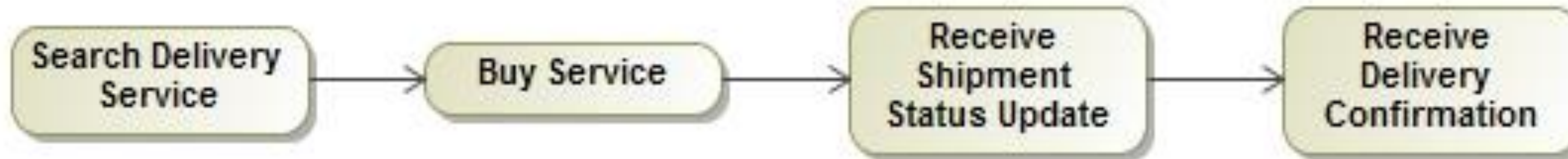
CHICAGO TECHFEST
Hear It. See It. Do It.
October 23-24

Define a State Machine for each Aggregate



Customer Journey

CHICAGO TECHFEST
Hear It. See It. Do It.
October 23-24



Domain Design Easy Traps

Anemic Domain Model

- Aggregates have technical rather than business focus
- Takes all the overhead of OOD/OOP without realizing the benefits

Leaking Business Logic into the service layer

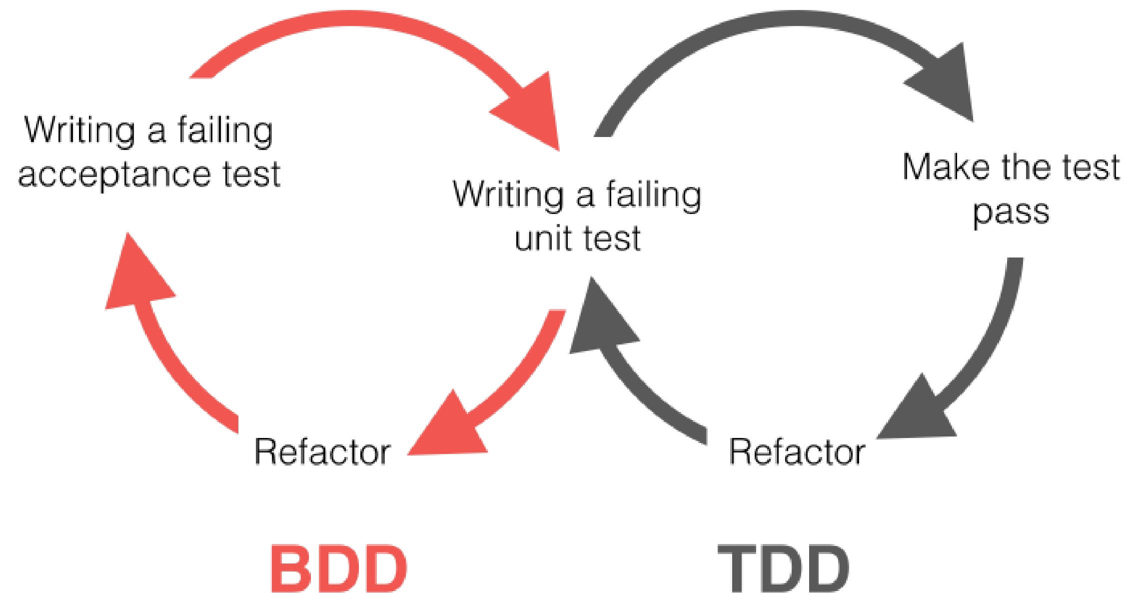
- Services suffer from identity crisis
- Business Logic must be embedded inside its domain model
- Bunch of public empty getters and setters

DDD from Design To Microservice Implementation

1. Start with Event Storming Session
2. Create use case from business requirements
3. Choose your Core, Support, Generic business domains
4. Create your aggregates into their own Bounded Contexts using their ubiquitous language
5. Create the Domain models
6. The use cases become messages (operations) between aggregates
7. Create State Machine diagrams to represents various aggregate states
8. System documentation is actually your functional test using Behavioral Driven Design (BDD)

TDD , BDD

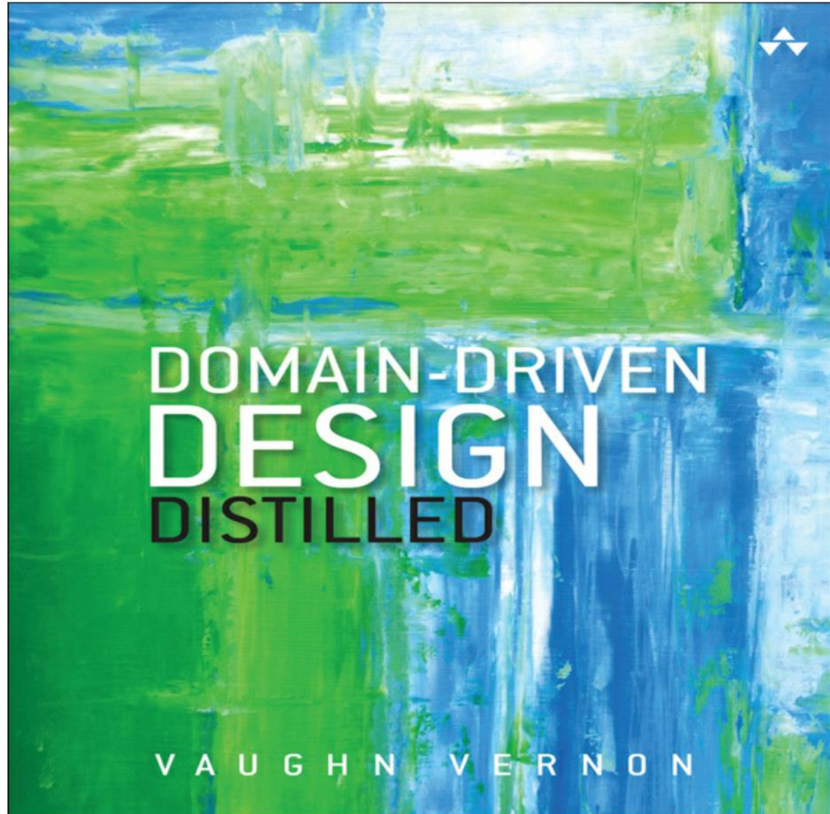
1. Identify business **feature**.
2. Identify **scenarios** under the selected feature.
3. Define **steps** for each scenario.
4. **Run** feature and **fail**.
5. Write code to **make steps pass**.
6. **Refactor** code, Create **reusable automation library**.
7. Run feature and **pass**.
8. Generate **test reports**.

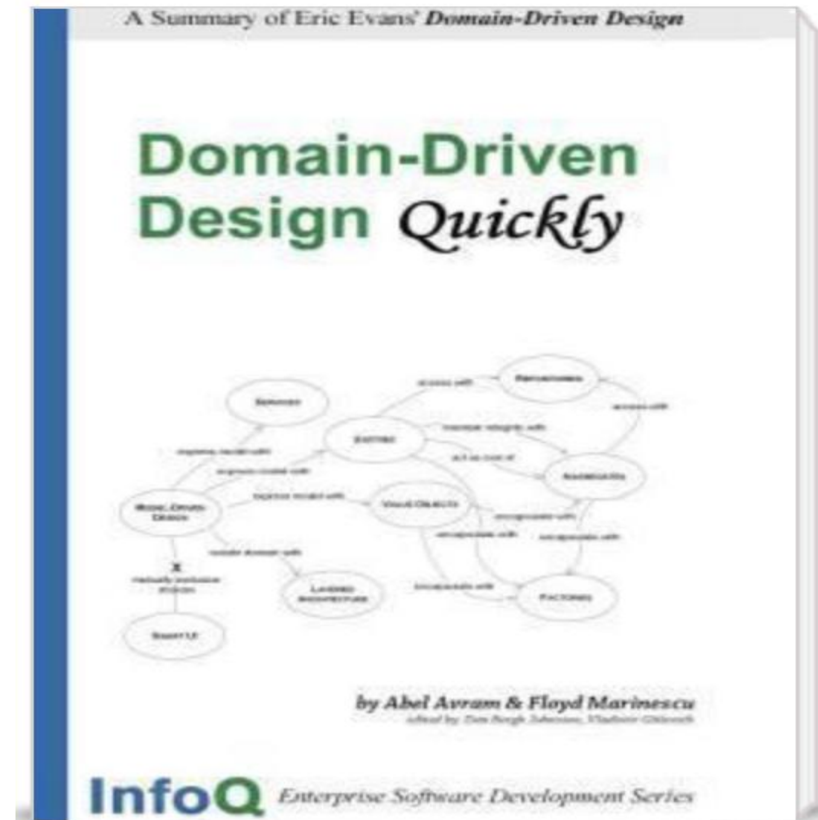


Other DDD Tools

- SWOT (Strength, Weakness, Opportunity and Threat
- Timebox modeling
- Functional Tests and BDD/ATDD System specifications need to be documented as features and scenarios (Given/when/then)

Resources and References





THANK YOU!!!

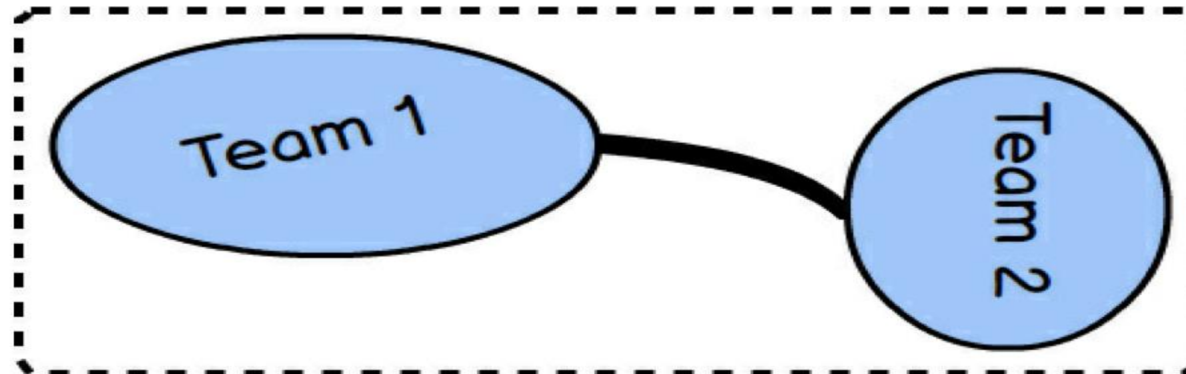
What’s Next on DDD

Full day workshop on DDD and Microservice Design Concepts is in all CCB TechHubs in October, November and December

Appendix

Context Mapping

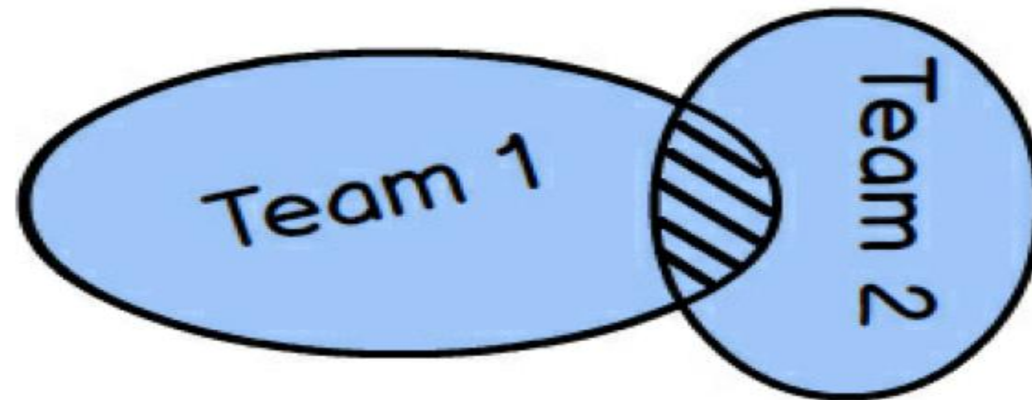
Represents the relation ship between the contexts. Both Team and Technology



Context Mapping - Partnership

Closely aligns two teams with dependent set of goals.

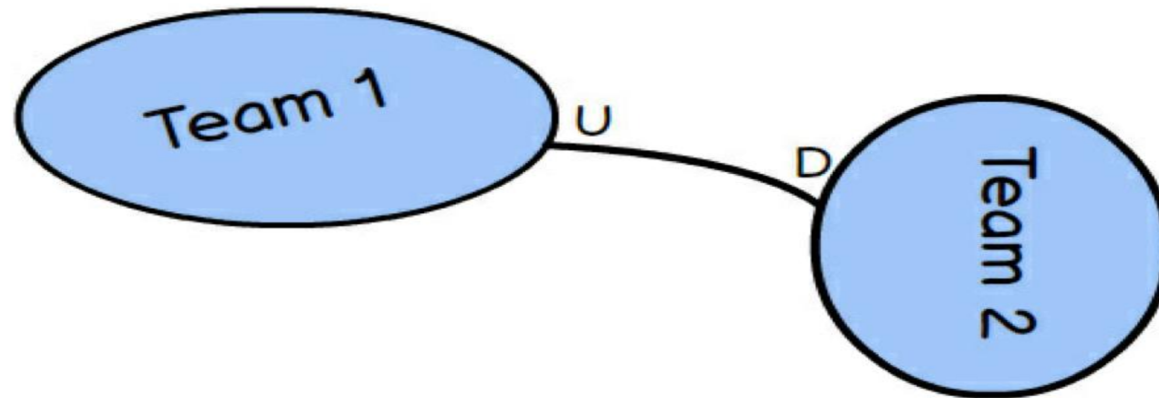
- Synchronized, Continuous Integration.
- Hard to keep for long time. Limits need to be set.



Context Mapping – Shared Kernel

Teams (two or more) share a small common model

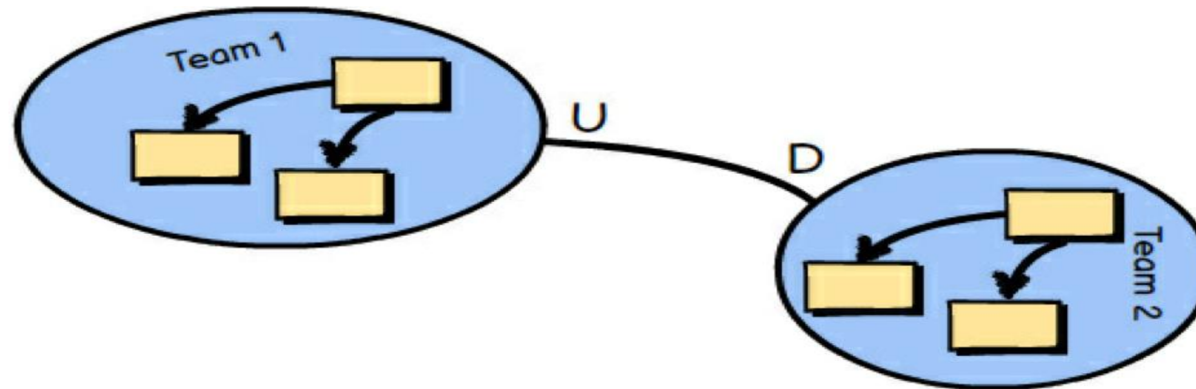
- Possibly on team owns and maintain that shared kernel
- Open communication between the teams and **Constant** agreement



Context Mapping – Customer-Supplier

Supplier (U) holds the sway and Determine what the Customer (D) will get and when.

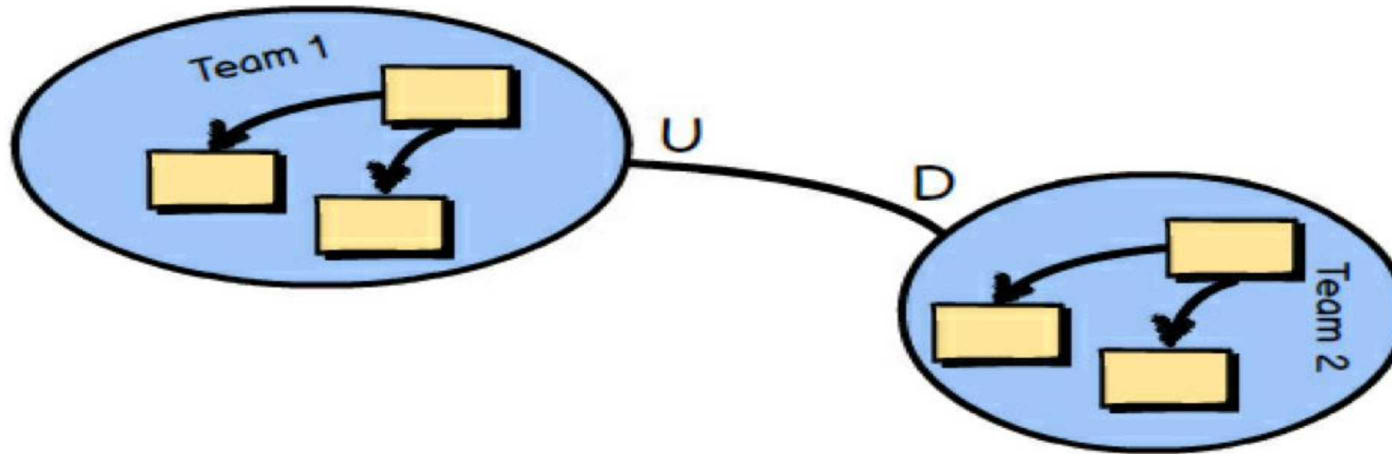
- Supplier plans to meet some/all customer needs
- Customer plan with Supplier to meet various expectations



Context Mapping - Conformist

Upstream team has no motivation to support specific needs for Downstream team

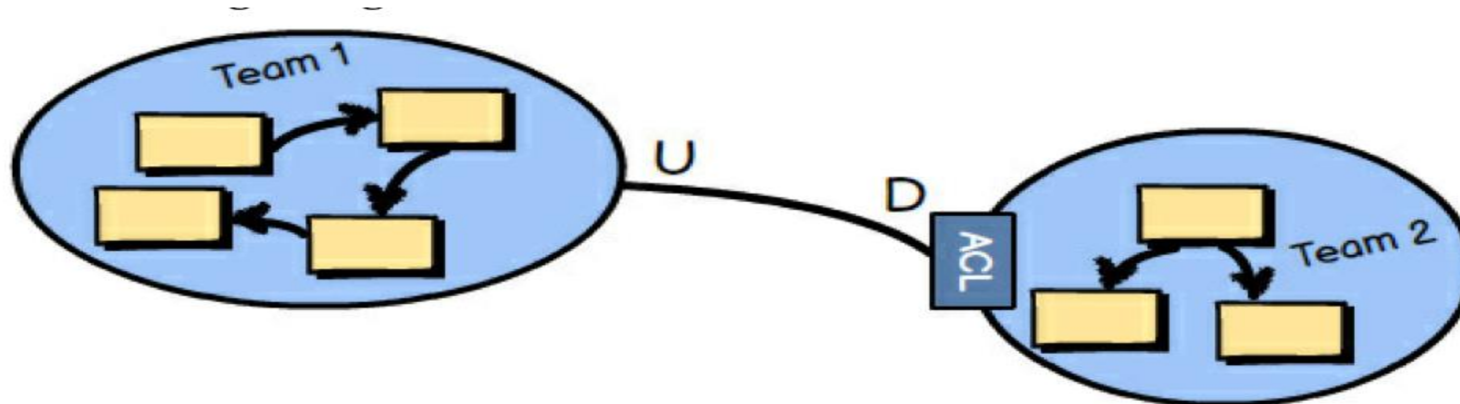
- Downstream team **Conforms** with Upstream specs
- Example Amazon.com and its sellers



Context Mapping – Anticorruption Layer

Downstream teams creates translation layer to isolate them from Upstream team changes

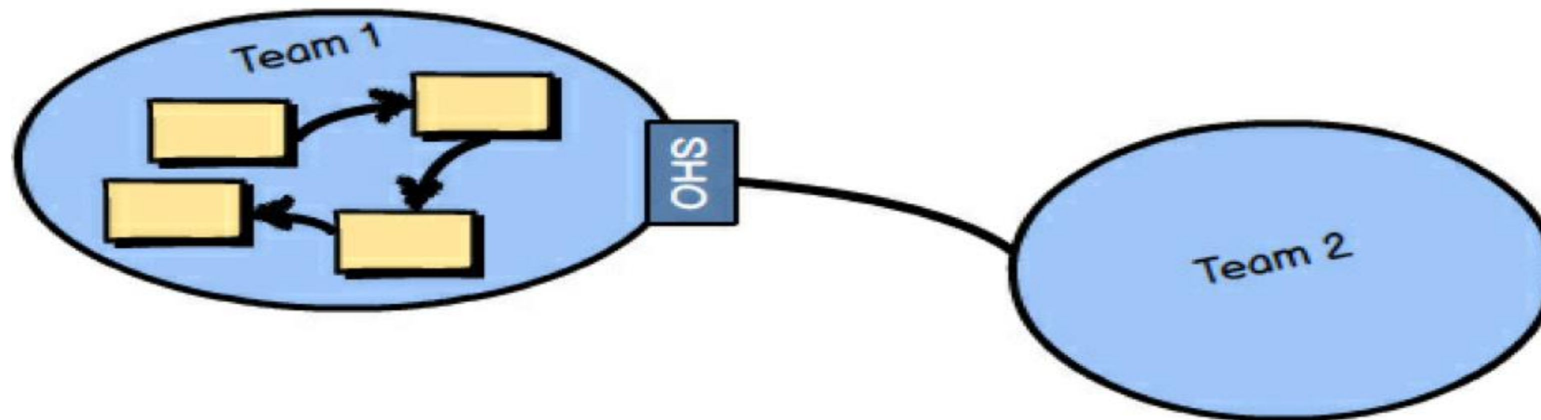
- Most Defensive relationship
- A common solution used to integrate with legacy systems



Context Mapping – Open Host Service

Upstream system defines a protocol/Interface to downstream access to their bounded context set of services

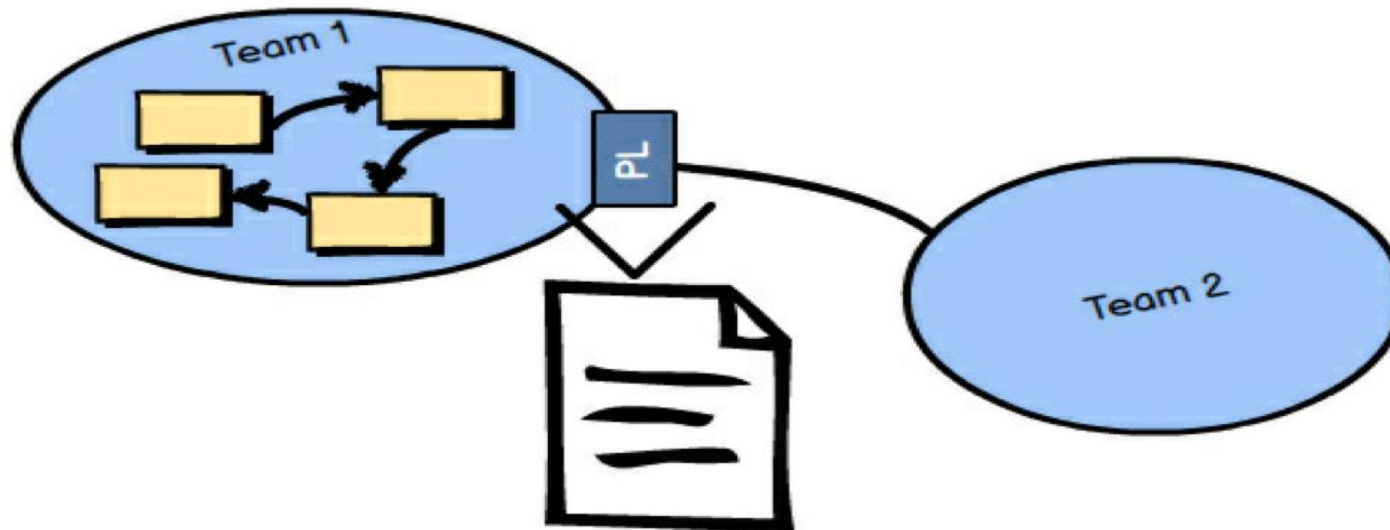
- Well documented API from Team 1
- No need for Anti corruption layer, Team 2 can be comfortably Conformist



Context Mapping – Published Language

Upstream system Publishes communication language of some standard

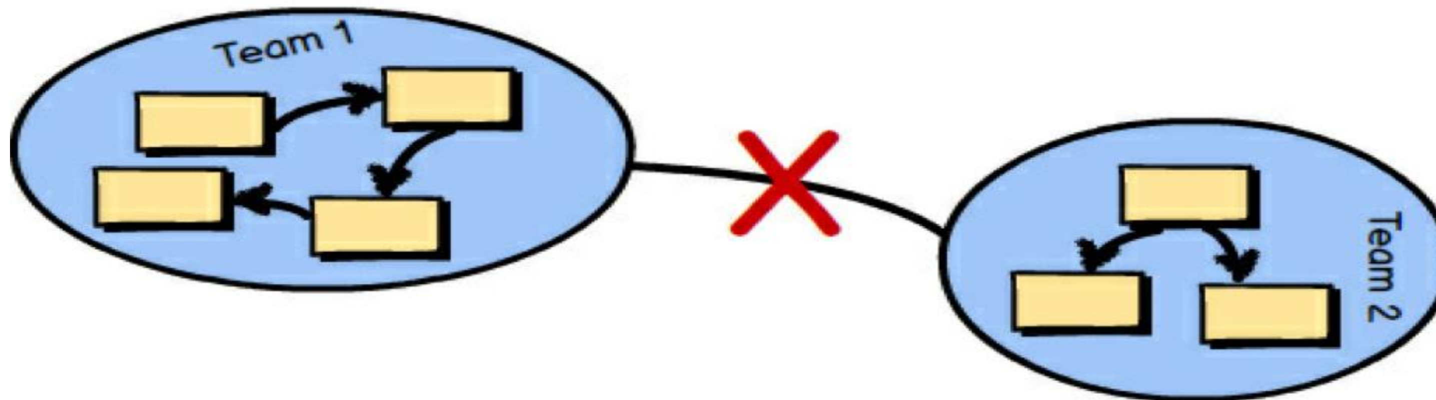
- Examples XML or JSON Schema, RESTful API, Async Messaging
- Consumer can translate from and to the language easily



Context Mapping – Separate Ways

The integration will provide No significant payoff

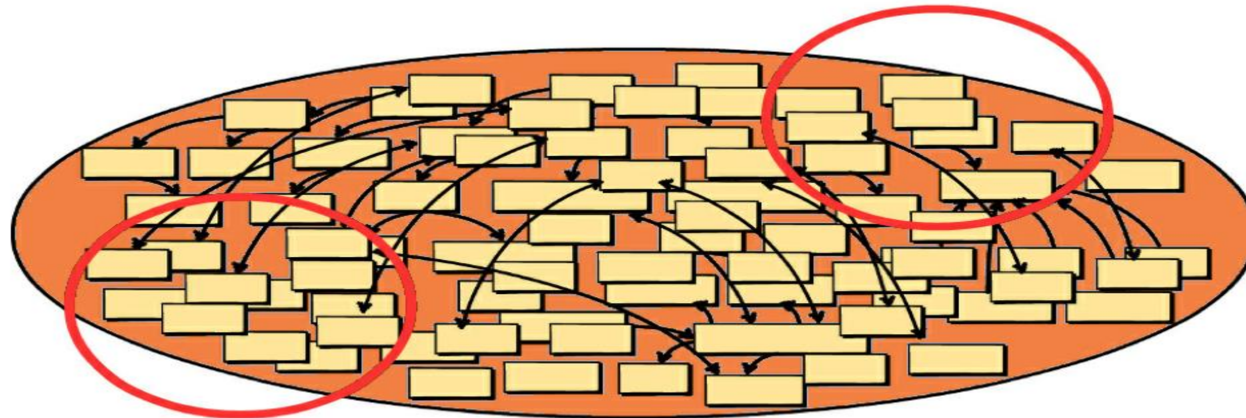
- No one ubiquitous language provides the needed functionality
- Consumer builds its own solution and forget about integration to the provider ubiquitous language



Context Mapping – Big Ball of Mud

This should be avoided at all costs

- Overtime, more and more cross aggregate relationship will be created, making system maintenance harder costly
- System will completely collapse eventually



Source: Vernon, Vaughn. Domain-Driven Design Distilled