PREGRADO

UNIDAD 2 | HIGH-LEVEL SOFTWARE DESIGN & PATTERNS

DOMAIN-DRIVEN DESIGN

PART 1

SI424 | Diseño y Patrones de Software

Al finalizar la unidad, el estudiante elabora y comunica artefactos de diseño de software aplicando principios básicos y patrones empresariales de diseño para un dominio y contexto determinados.

AGENDA

INTRO

DOMAIN-DRIVEN DESIGN ELEMENTS

STRATEGIC LEVEL

UBIQUITOUS LANGUAGE

STUDY CASE

What is the Domain?

Domain in the realm of software engineering commonly refers to the subject area on which the application is intended to apply. In other words, during application development, the domain is the “sphere of knowledge and activity around which the application logic revolves.”

Another common term used during software development is the domain layer or domain logic, which may be better known to many developers as the business logic. The business logic of an application refers to the higher-level rules for how business objects interact with one another to create and modify modelled data.

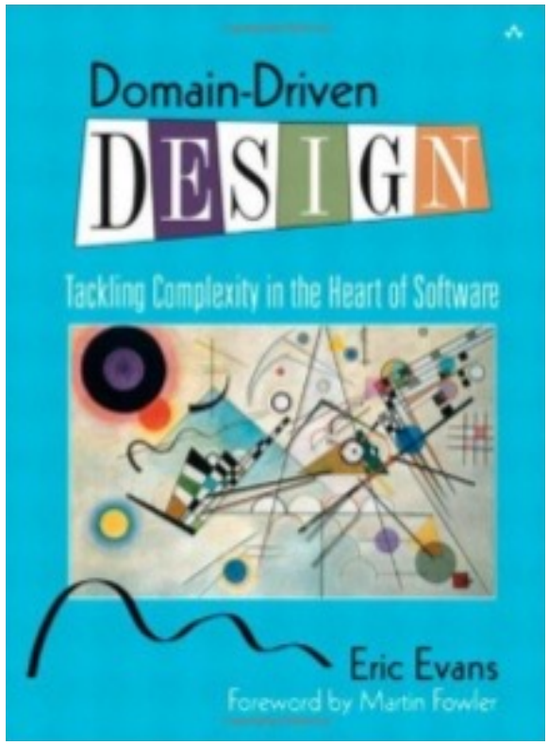
Domain = your business

Domain model = Code that the represents this business

Domain-Driven Design

Introducido por Eric Evans en el libro *Domain-Driven Design: Tackling Complexity in the Heart of Software*.

DDD es un approach utilizado para necesidades complejas, que conecta la implementación a un modelo evolutivo de los conceptos core del negocio.



Se enfoca en el dominio del problema y básicamente ayuda a identificar la arquitectura e informar al equipo sobre la mecánica del negocio que el software necesita replicar.

Valor estratégico

DDD tiene valor estratégico dado que establece el mapping entre los conceptos del dominio del negocio y los artefactos de software.

No es una metodología, está más un approach al diseño de la arquitectura del software, brindando una estructura de prácticas para tomar decisiones de diseño que sirvan de ayuda a los proyectos con dominios complicados.

Focus

- The core domain and domain logic.
- Complex designs on models of the domain.
- Improving the application model and resolving emerging domain-related issues by collaborating with domain experts.

AGENDA

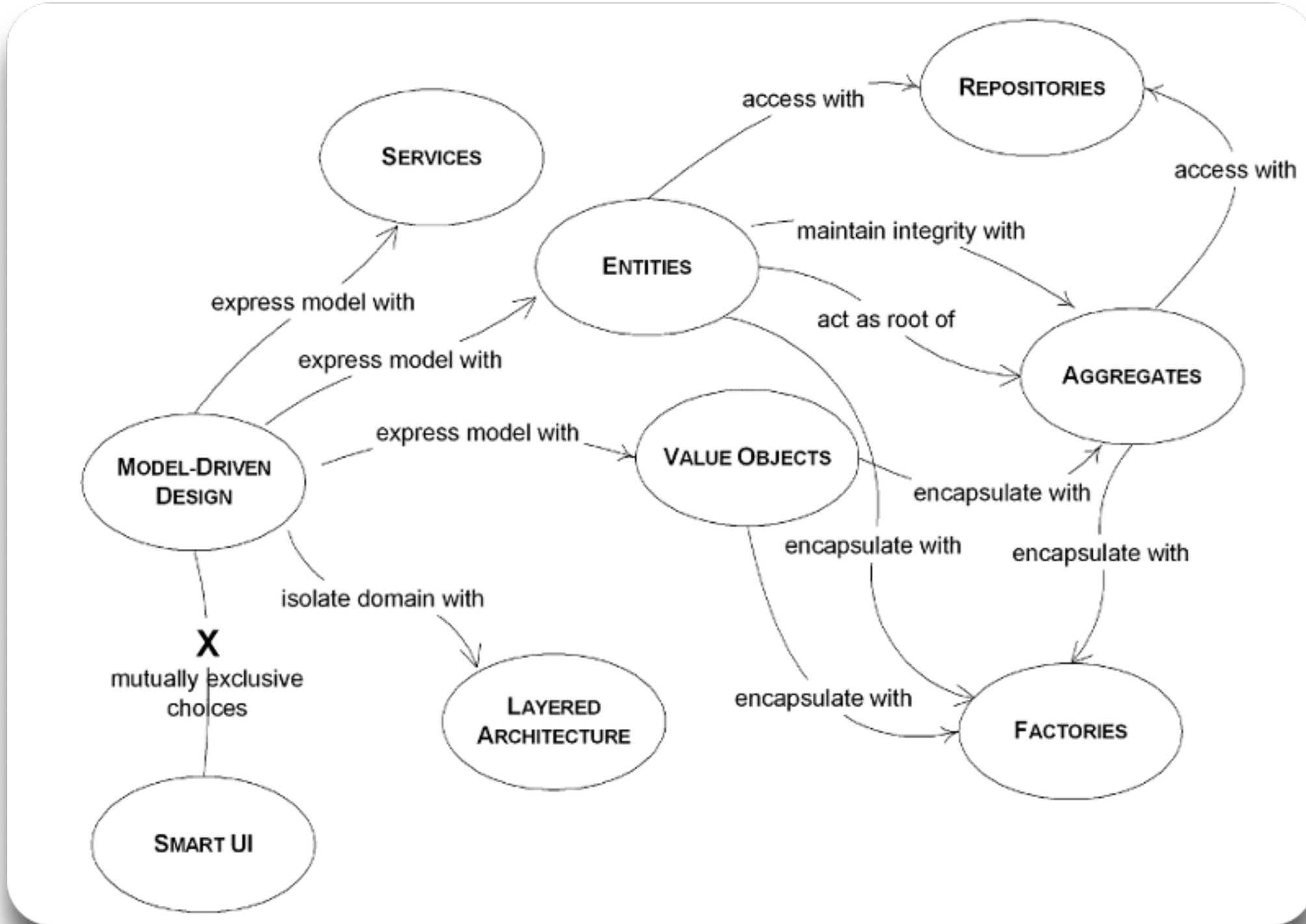
INTRO

DOMAIN-DRIVEN DESIGN ELEMENTS

STRATEGIC LEVEL

UBIQUITOUS LANGUAGE

STUDY CASE


SERVICES
access with
REPOSITORIES
access with
ENTITIES
maintain integrity with
express model with
act as root of
express model with
AGGREGATES
express model with
MODEL-DRIVEN DESIGN
VALUE OBJECTS
encapsulate with
encapsulate with
encapsulate with
isolate domain with
X
mutually exclusive choices
LAYERED ARCHITECTURE
encapsulate with
FACTORIES
SMART UI

Topics

Domain driven design is divided into two (2) major topics:

- Strategic
- Tactical

Do not start applying DDD by Tactical (creation of Entities, Repositories, Value Objects and etc.).

Spend time on Strategic because that's where your architecture will be defined!

Ubiquitous Language

El idioma que es usado por el equipo para conectar todas las actividades de dicho equipo con el software.

Ayuda cuando hay que conocer más sobre los términos que utilizan los expertos en el negocio.

El equipo técnico es capaz de conocer si el idioma cambia y si un término específico se va a utilizar para un significado distinto.

Context

Una configuración que determina el significado de un statement.

Context Mapping

Gráfico que conecta los contextos.

Para cada contexto, hay un idioma, una implementación independiente y una interfaz para comunicarse con otros contextos delimitados (bounded contexts).

Bounded Contexts

Bounded context es el contexto dentro del cual, el ubiquitous language y modelo correspondiente son válidos.

Le da al equipo un claro entendimiento de qué tiene que ser consistente y qué puede desarrollarse de forma independiente.

Model

Un sistema que describe los aspectos seleccionados de un dominio y que a menudo se usa para resolver problemas relacionados con un dominio en particular.

Strong Domain Model Characteristics

Being Aligned with the business' model, strategies, and processes.

Being isolated from other domains and layers in the business.

Be loosely designed with no dependencies on the layers of the application on either side of the domain layer.

Being reusable to avoid models that are duplicated.

Be an abstract and cleanly separated layer to create easier maintenance, testing, and versioning.

Minimum dependencies on infrastructure frameworks to avoid outliving those frameworks and tight coupling on external frameworks.

Designed with a "Plain Old Java Object" programming model without having any technology or framework dependencies.

Key Benefits

Business necessities are oriented.

A common set of terms and definitions used by the entire team.

Keeping track is made easier.

Better code.

Agility is a standard.

Get a good software architecture.

Communication counts.

A balanced application.

Stay focused on the solution.

Purely flexible.

Building Blocks

Entity.

Value Object.

Domain Event.

Aggregate.

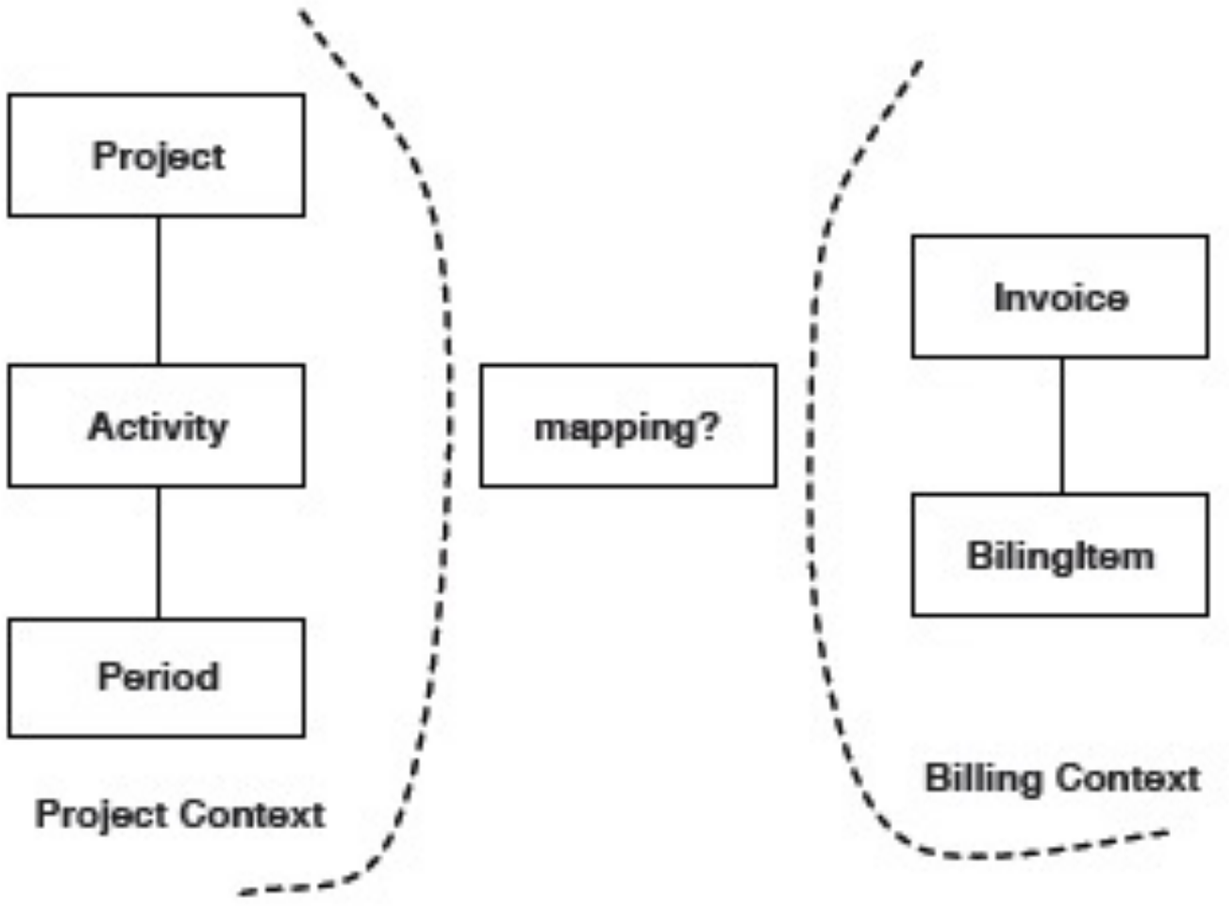
Service.

Repositories.

Factories.

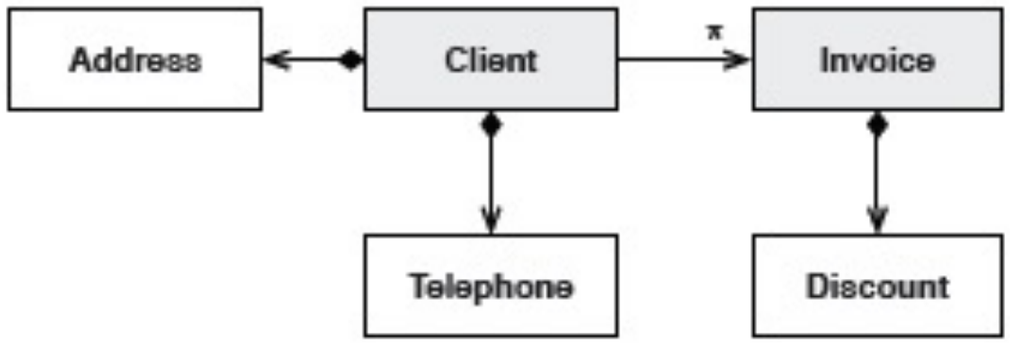
Strategic Design

Context Maps

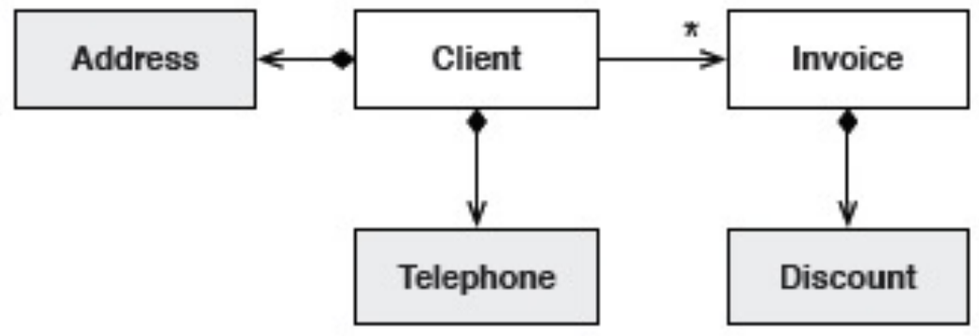


Modeling the Domain

Entities

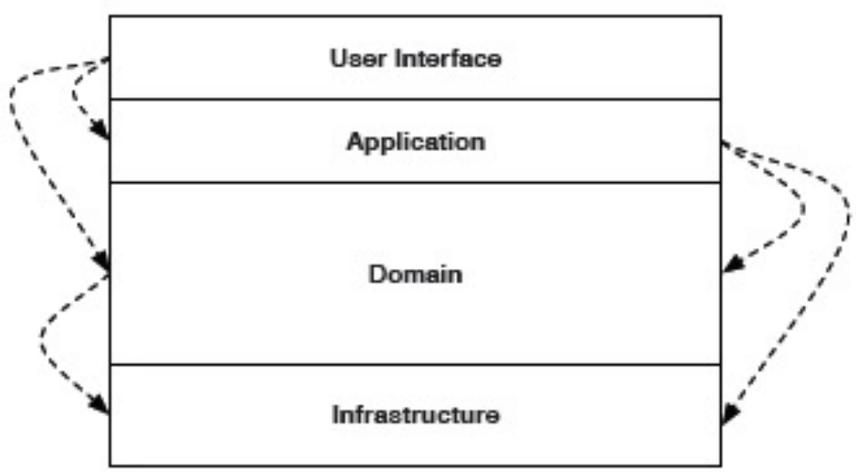


Value Objects



Application Architecture

Layered architecture



User Interface	Responsible for constructing the user interface and managing the interaction with the domain model. Typical implementation pattern is model-view-controller.
Application	Thin layer that allows the view to collaborate with the domain. Warning: it is an easy 'dumping ground' for displaced domain behavior and can be a magnet for 'transaction script' style code.
Domain	An extremely behavior-rich and expressive model of the domain. Note that repositories and factories are part of the domain. However, the object-relational mapper to which the repositories might delegate are part of the infrastructure, below this layer.
Infrastructure	Deals with technology specific decisions and focuses more on implementations and less on intentions. Note that domain instances can be created in this layer, but, typically, it is the repository that interacts with this layer, to obtain references to these objects.

AGENDA

INTRO

DOMAIN-DRIVEN DESIGN ELEMENTS

STRATEGIC LEVEL

UBIQUITOUS LANGUAGE

STUDY CASE

Strategic

Domain Experts (DEs) are people with a good knowledge of the problem to be solved in that specific domain. We can say that today Product Managers play this role, but don't limit yourself to them.

What does the developer have to do with the business?

- Developers need to understand the business, and for this to happen there must be intense interaction between devs and DEs.

How do DEs understand a technical language and how can devs understand the business language?

- That's exactly where the Ubiquitous Language comes in!

Ubiquitous Language

A language structured around the domain model and used by all team members to connect all the activities of the team with the software.

From now, there is no longer the business language (Domain) and the development language. The team will speak the same language!

Developers must make clear in code the actions that are used by DEs as well.

Agile manifesto:

- Individuals and interactions over processes and tools
- Customer collaboration over contract negotiation

Bounded Context

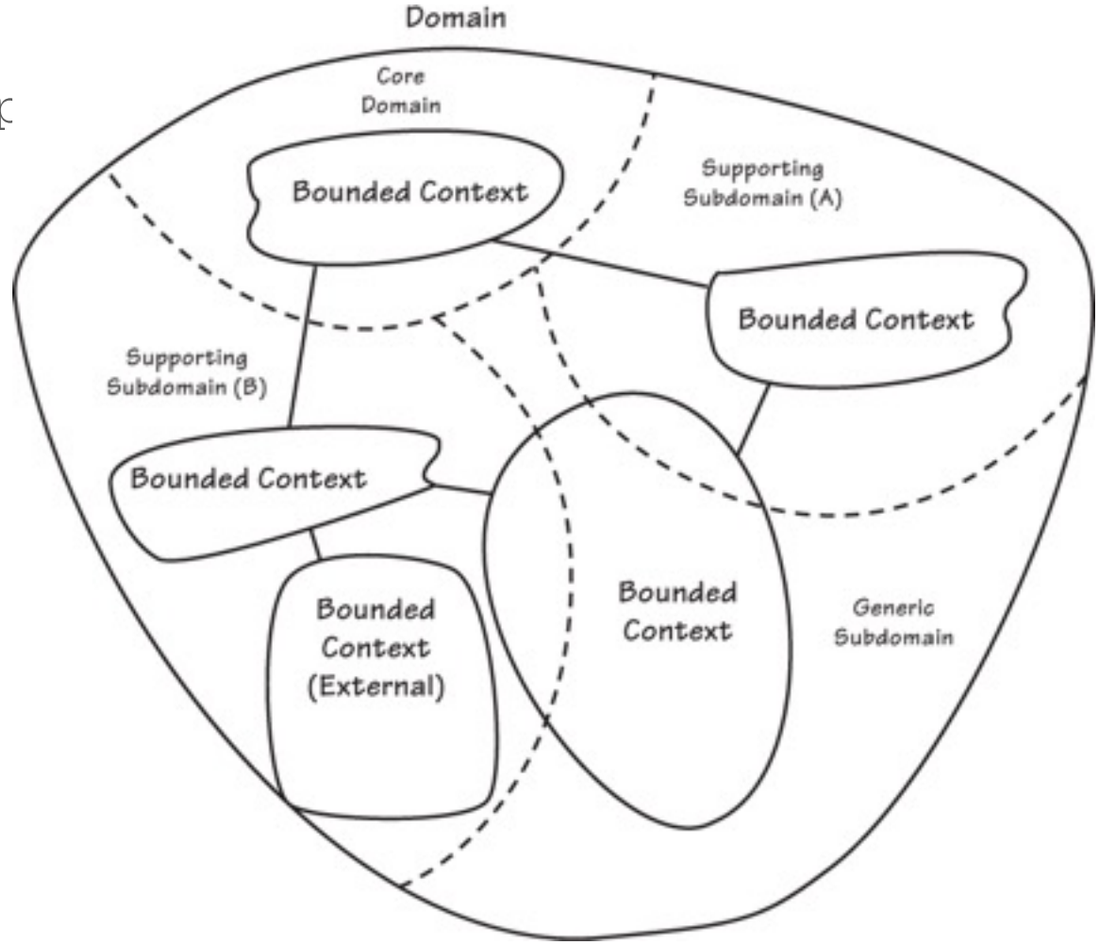
A description of a boundary (typically a subsystem, or the work of a specific team) within which a particular model is defined and applicable.

A domain can be divided into subdomains that are separated into contexts.

- Delimited Space (External Interface)
- Each context has its own ubiquitous language.
- Each context has its own architecture (Independent Imp

Agile manifesto:

- Working software over comprehensive documentation
- Responding to change over following a plan



Benefits

- The whole company talking the same language, reduced risk of misunderstandings.
- Now you have a segregated architecture.
- Good integration with Business layer and Engineering layer.
- Maintainability. Smaller software is easier to maintain.
- Flexible. Now your services are independent.
- Domain and Architecture mapped. Side effects are not a surprise anymore.
- Quality.

Considerations

- Complex architecture.
- Additional efforts. Now we need to have a lot of meetings and spend some time mapping our domain.
- New mindset. For a good implementation of DDD, everyone needs to be aligned, both in vocabulary and ownership (here is the most important point of DDD).
- Prioritize tasks. For DDD, technical debts are not tasks that will be in the backlog forever.

AGENDA

INTRO

DOMAIN-DRIVEN DESIGN ELEMENTS

STRATEGIC LEVEL

UBIQUITOUS LANGUAGE

STUDY CASE

Representing the Model

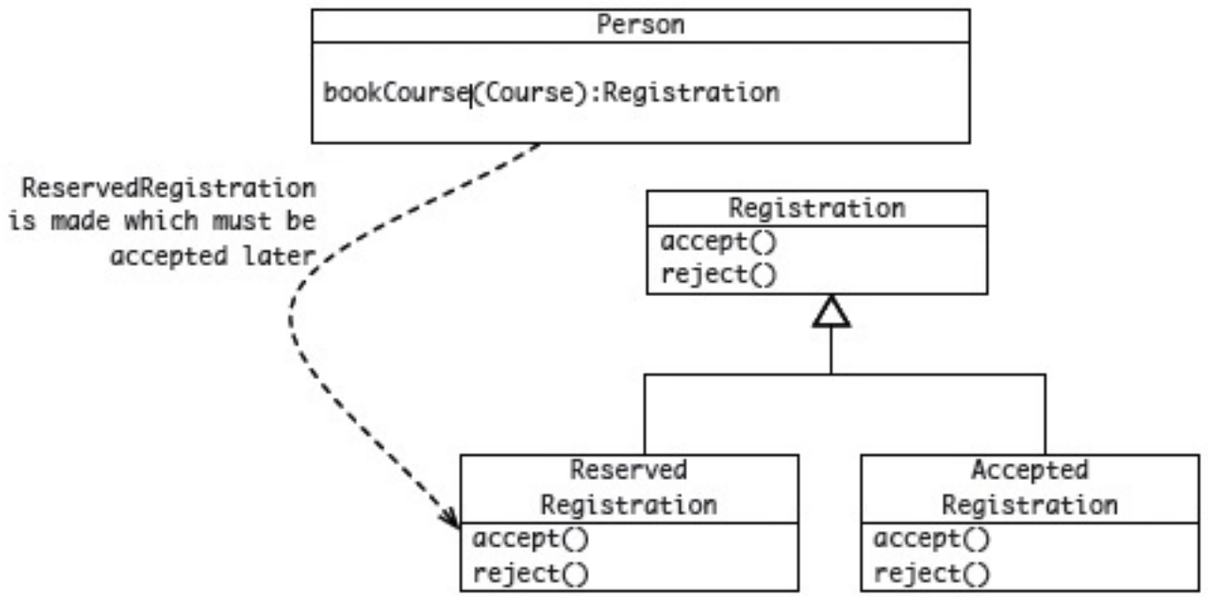
Using Language

A person that is looking at attending a training course searches for courses based on topic, cost and the course schedule. When a course is booked, a registration is issued which the person can cancel or accept at a later date.

Using Code

```
class Person {
   public Registration bookCourse(Course c) { ... }
}

abstract class Registration {
    public abstract void accept();
    public abstract void cancel();
}

class ReservedRegistration extends Registration { ..
class AcceptedRegistration extends Registration { ..

interface CourseRepository {
    public List<Course> find(...);
}
```

Using a UML Sketch



Ubiquitous Language

Reveal the Intention not the Implementation

A conversation:

"When a person books a course, and the course is full, then the person has a status of 'waiting'. If there was space available, then the person's details must be sent via our message bus for processing by the payment gateway".

Obstructions

person has a status	Status seems to be a flag or field. Perhaps the domain expert is familiar with some other system, maybe a spreadsheet, and is suggesting this implementation.
sent via our message bus	This is a technical implementation. The fact that it is sent via a message bus is of no consequence in the domain.
processing	This is ambiguous and obscure. What happens during processing?
payment gateway	Another implementation. It is more important that there is some form of payment, but the implementation of the payment is insignificant at this point.

Ubiquitous Language

Refactor the Language

The refactored conversation:

"When a person registers for a course, a reserved registration is issued. If there is a seat available, and payment has been received, then the reserved registration is accepted. If there are no seats available on the course, then the reserved registration is placed on a waiting list as a standby registration. The waiting list is managed on a first in basis".

Ubiquitous Language

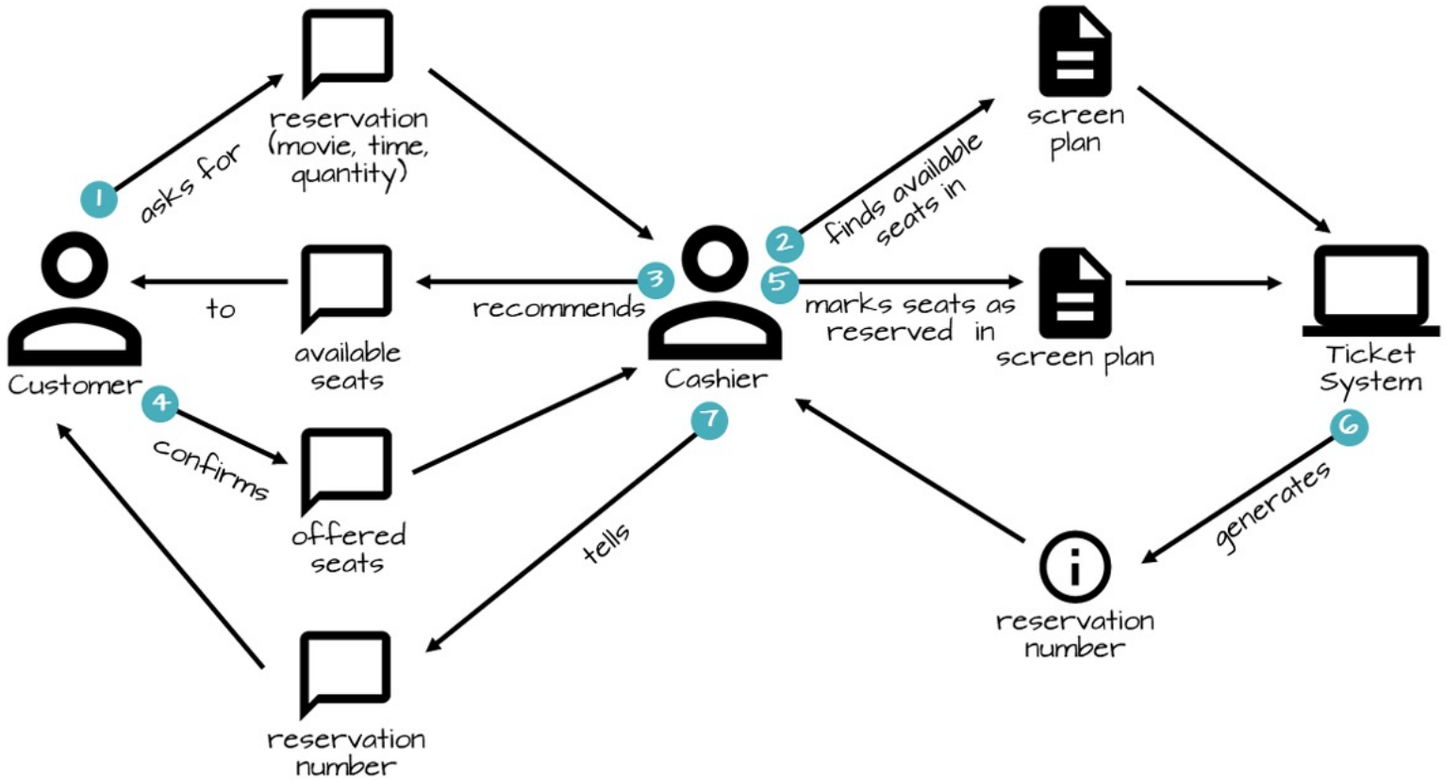
A concrete Story with BDD

```
Story: Register for a course

As a person looking for training

I want to book a course

So that I can learn and improve my skills.
```

```
Scenario: Course is full

Given that the Python 101 course accommodates 10 seats

and there are already 10 people with confirmed registrations for Python 101

When I register for 'Python 101'

Then there should be a standby registration for me for Python 101

and my standby registration should be on the waiting list.
```

Domain Storytelling

Es un lenguaje pictográfico que busca registrar de manera práctica los domain stories, de tal forma que los domain experts puedan ver si el equipo está entendiendo su story correctamente.



Domain Storytelling

Domain Stories ayudan en la elaboración para el product backlog de los epics y user stories.
Los user stories pueden derivarse directamente de los domain stories.

Por ejemplo

“As a cashier, I want to determine the free seats for a show so that I can offer these seats to my customers.”

AGENDA

INTRO

DOMAIN-DRIVEN DESIGN ELEMENTS

STRATEGIC LEVEL

UBIQUITOUS LANGUAGE

STUDY CASE

Cargo Tracker

Cargo Business (lifecycle of cargos).

Capabilities:

Booking

Routing

Tracking

Handling

Cargo Tracker Domain

Sub-domains

Booking

- Booking of cargos
- Assigning of routes to cargos
- Modification of cargos (e.g. change of destination)
- Cancelation of cargos

Routing

- Optimal Itinerary allocation for cargos based on Route Specification.
- Voyage Maintenance for carriers that will carry cargos (e.g. addition of new rules).

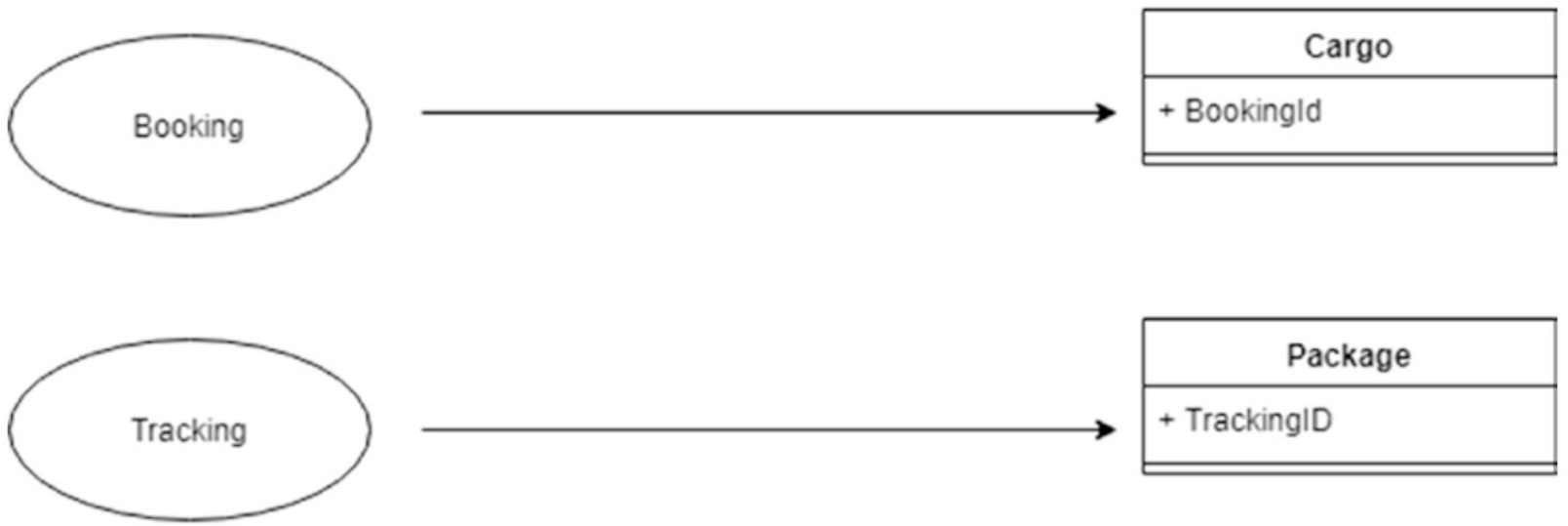
Handling

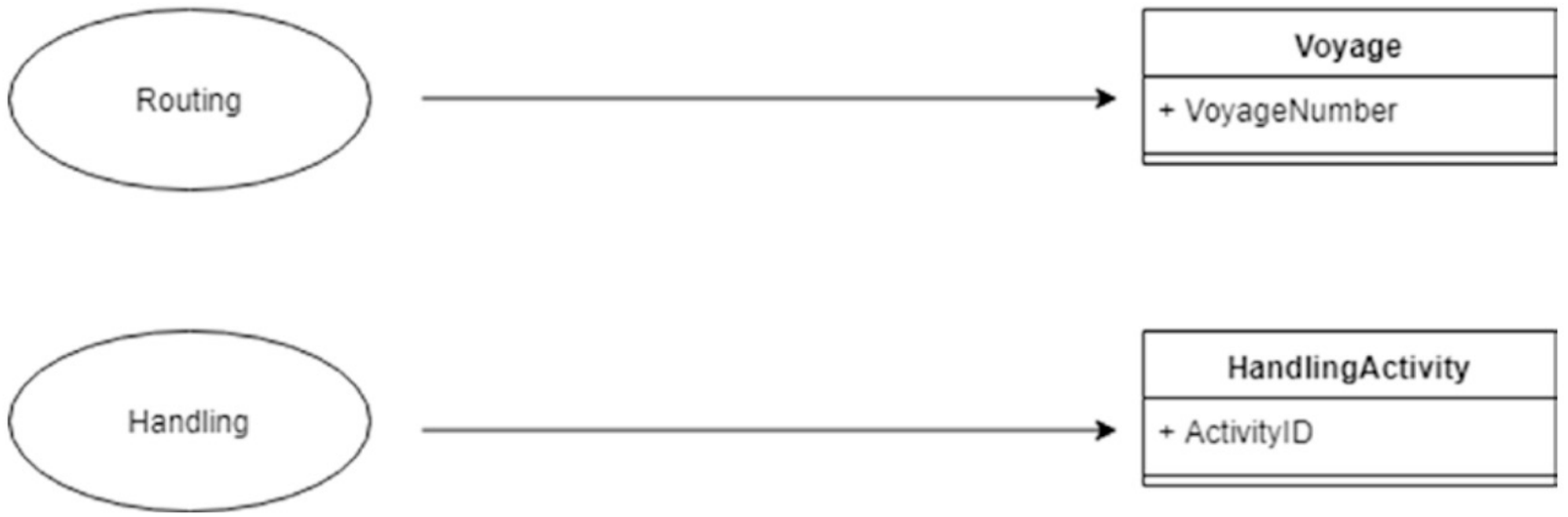
- Cargo need to be inspected/handled as it progresses along its assigned route.

Tracking

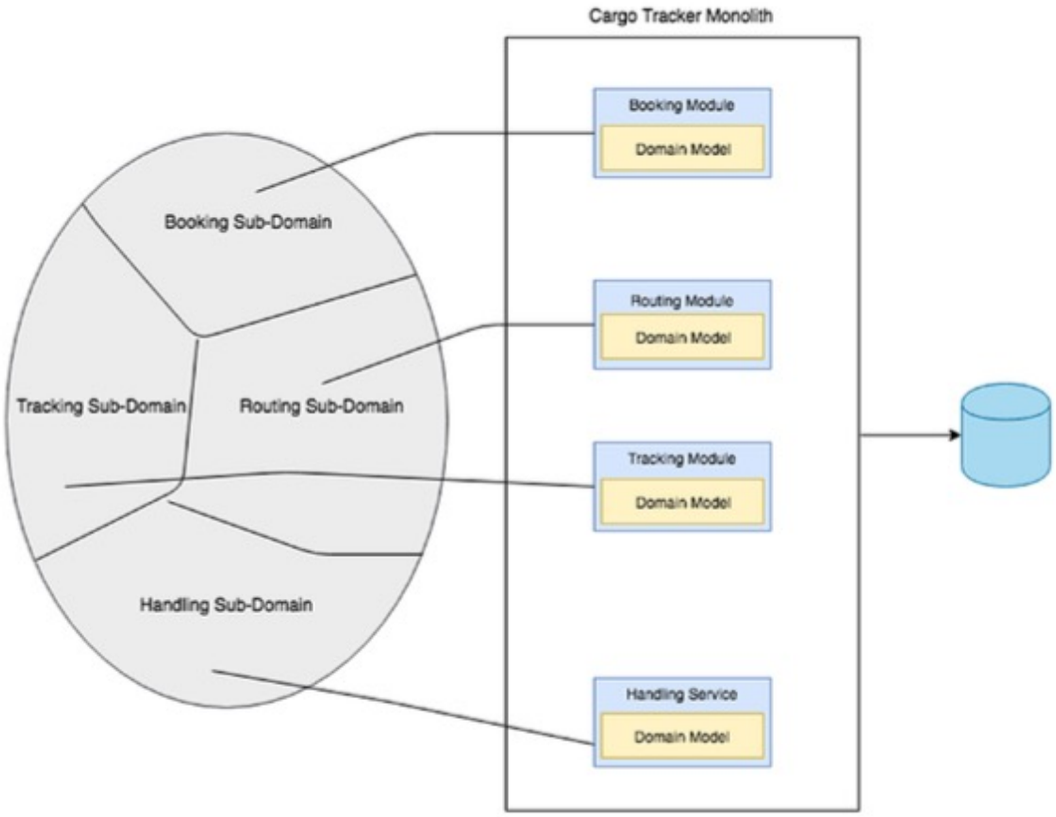
- Customers need comprenhensive, detailed, and up-to-date information of booked cargos.

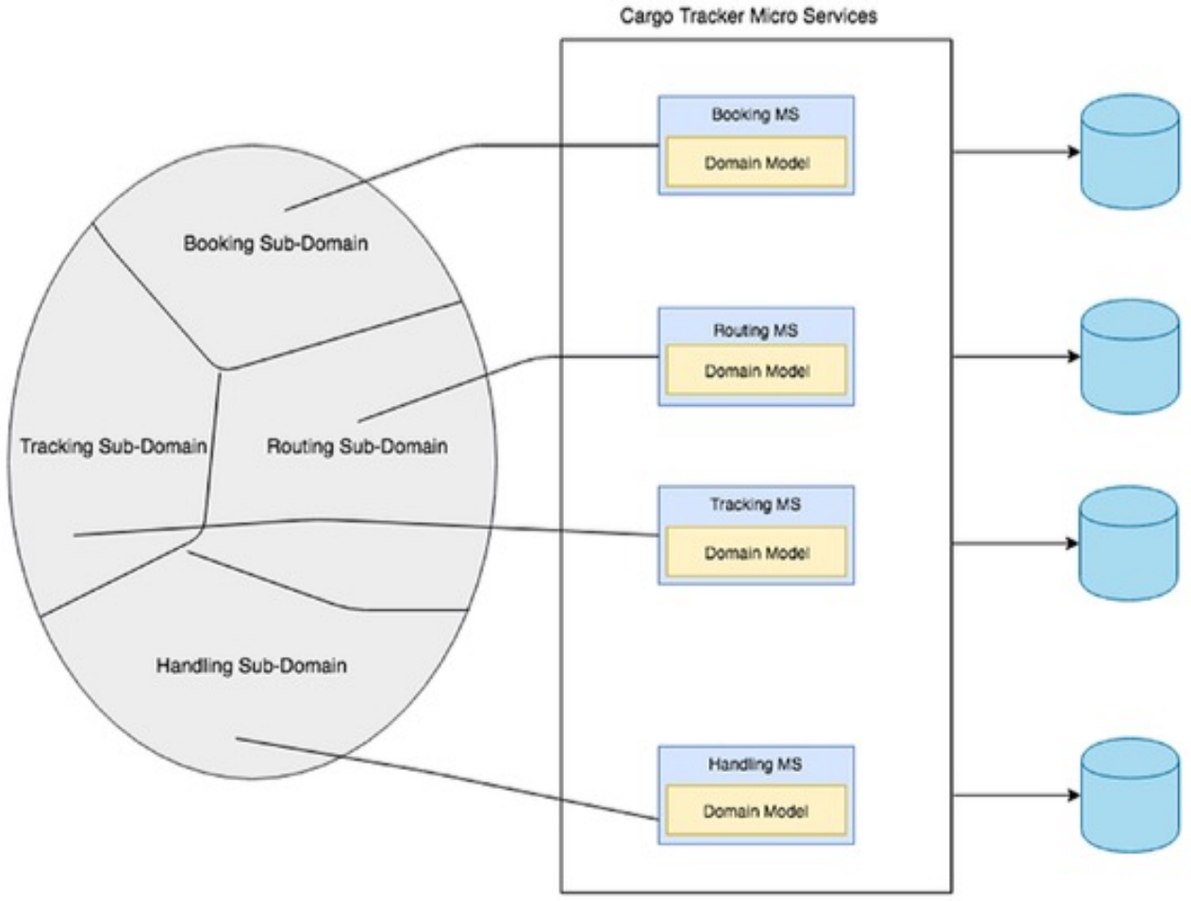
Boundaries


Booking
Cargo
+ BookingId
Tracking
Package
+ TrackingID

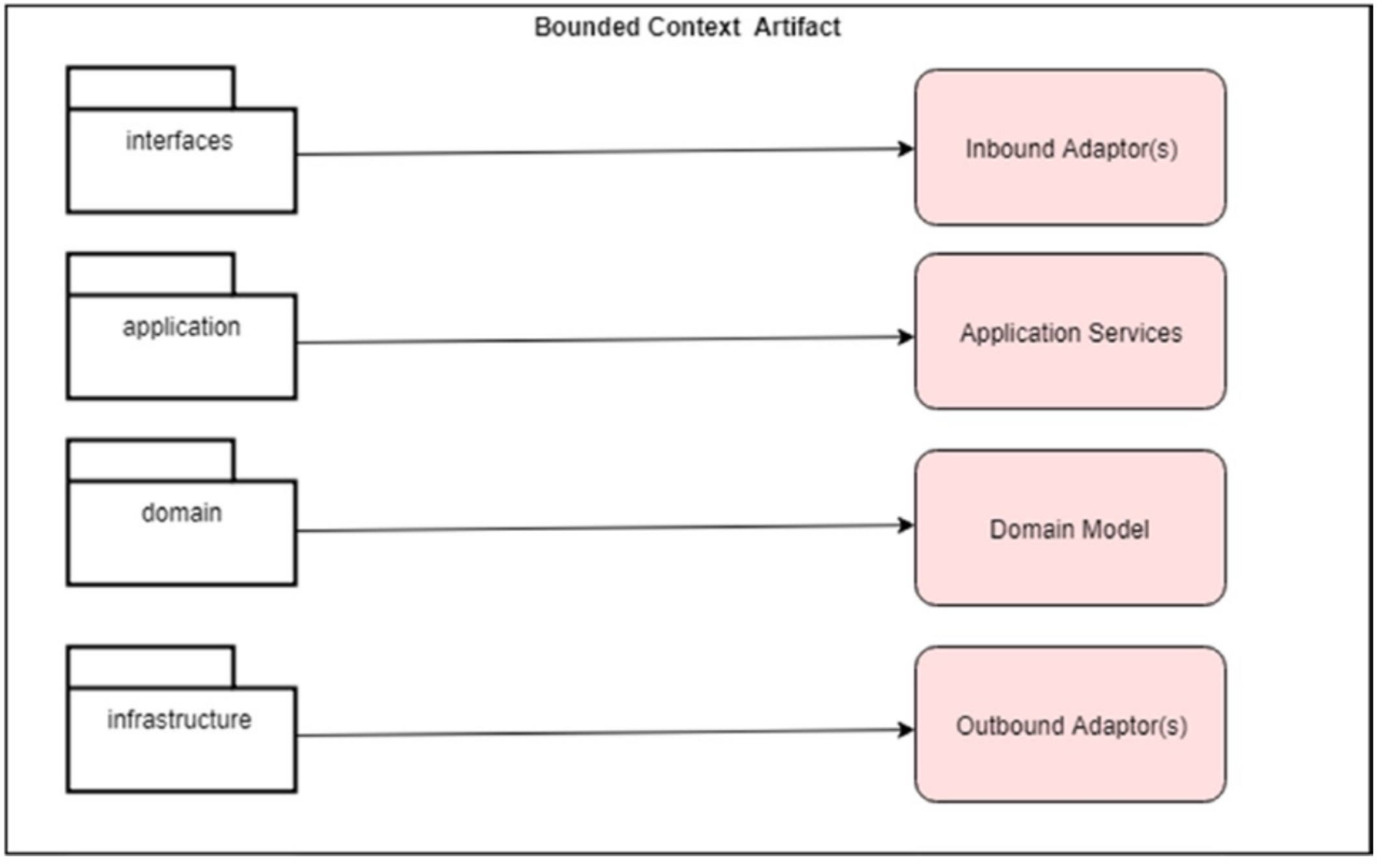

Routing
Voyage
+ VoyageNumber
Handling
HandlingActivity
+ ActivityID

Cargo Tracker Sub-domains





Bounded Context



RESUMEN

Recordemos

Domain-Driven design es un approach para realizar el diseño estratégico de un software.

Prioriza el uso de un lenguaje común, que entiendan los domain experts y el equipo.

Domain Storytelling ofrece una manera práctica para interactuar con los domain experts.

REFERENCIAS

Para profundizar

Domain Language
https://domainlanguage.com/ddd/

Domain Storytelling official Site
https://domainstorytelling.org

REFERENCIAS

Para profundizar

https://airbrake.io/blog/software-design/domain-driven-design

http://olivergierke.de/2020/03/Implementing-DDD-Building-Blocks-in-Java/

https://medium.com/tradeshift-engineering/my-vision-as-a-software-engineer-about-ddd-domain-driven-design-2f36ec18a1ec

https://cargo-tracker.gitbook.io/documentation/java-ee-and-ddd

PREGRADO

Ingeniería de Software

Escuela de Ingeniería de Sistemas y Computación | Facultad de Ingeniería

UPC
Universidad Peruana
de Ciencias Aplicadas

Prolongación Primavera 2390,
Monterrico, Santiago de Surco
Lima 33 - Perú
T 511 313 3333
https://www.upc.edu.pe

exígete, innova