Enterprise System Integration (MTAT.03.229)

LECTURE 4: DOMAIN-DRIVEN DESIGN (DDD) APPROACH II

MOHAMAD GHRARIB
UNIVERSITY OF TARTU
Domain-driven design (DDD)

**DDD** is a software *design approach* that focus on modelling software to match a domain according to input from domain's experts\(^1\).

**DDD** is used for developing software for *complex software systems*.

Applying the **DDD** require deep understanding of the *domain* of discloser.

---

Why do we need the DDD?

What is a plant?
Why do we need the DDD?

What is a plant?

Booking unit

A plant is a thing that is booked or not.
Why do we need the DDD?

What is a plant?

**Booking unit**

A plant is a thing that is booked or not.

**Shipping unit**

A plant is a thing that can be delivered one location to another.
Why do we need the DDD?

What is a plant?

**Booking unit**
A plant is a thing that is booked or not.

**Shipping unit**
A plant is a thing that can be delivered one location to another.

**Business unit**
A plant is a thing that can/cannot be rented to a specific location depending on the rent period.

The definition of what a **plant** is **depends** on who you ask.
Why do we need the DDD?

What is a plant?

This means **language** is important

Booking unit
A plant is a thing that is booked or not.

Shipping unit
A plant is a thing that can be delivered one location to another.

Business unit
A plant is a thing that cannot be rented to a specific location depending on the rent period.

The definition of what a **plant** is **depends** on who you ask.
Domain-driven design (DDD)

**DDD** is an approach that “design” the system from top to down.
Domain-driven design (DDD)

**DDD** is an approach that “design” the system from top to down
DDD is an approach that “design” the system from top to down
**Domain-driven design (DDD)**

**Tactical design**

- Service/functionality
  - Modules
  - Classes
  - Objects

**Strategic design**

- Domain
  - Subdomain
    - Service/functionality
  - Subdomain
    - Service/functionality
  - Subdomain
    - Service/functionality

**DDD** is an approach that “design” the system from top to down
Domain-driven design (DDD)

**Ubiquitous language** is a common language that connects all the parts of the design.

**Strategic design** defines the bounded contexts and the Context Maps, and contributes the Ubiquitous Language.

**Tactic design** provides a tool to link Strategic design to implementation.
Domain-driven design (DDD)

**Ubiquitous language** a common language that connects all the parts of the design.

**Strategic design** defines the bounded contexts and the Context Maps, and contributes the Ubiquitous Language.

**Tactic design** provides a tool to link Strategic design to implementation.
**Ubiquitous language** a common language that connects all the parts of the design.
DDD - Ubiquitous language

Ubiquitous language a common language that connects all the parts of the design.
**Bounded context** is the boundary within a domain where a particular domain model applies.

A key aspect to define a bounded context is that, it should be able to function by itself.

Note: Bounded contexts are not necessarily isolated from one another
**Bounded context** is the boundary within a domain where a particular domain model applies.

A key aspect to define a bounded context is that, it should be able to function by itself.

Note: Bounded contexts are not necessarily isolated from one another.
**Bounded context** is the boundary within a domain where a particular domain model applies.

A key aspect to define a bounded context is that, it should be able to function by itself.

Note: Bounded contexts are not necessarily isolated from one another.
**Bounded context** is the boundary within a domain where a particular domain model applies.

A key aspect to define a bounded context is that, it should be able to function by itself.

Note: Bounded contexts are not necessarily isolated from one another.
**Bounded context** is the boundary within a domain where a particular domain model applies.

A key aspect to define a bounded context is that, it should be able to function by itself.

Note: Bounded contexts are not necessarily isolated from one another.
context Maps captures the relationships between bounded contexts.
DDD - Tactic design

Services

express model with

Model driven design

entities

repositories

maintain integrity with

act as a root of

aggregate

encapsulate with

factories

value objects

encapsulate with

encapsulate with

encapsulate with
An *entity* is an object that is identified by its consistent thread of continuity, as opposed to traditional objects, which are defined by their attributes.

An **entity** do have distinctive identity.
An entity is an object that is identified by its consistent thread of continuity, as opposed to traditional objects, which are defined by their attributes. An entity do have distinctive identity.
Value Objects are used to describe certain aspects of a domain. Value Objects do not have distinct identity. E.g., measures, quantity or describe the things in domain.
**Value Objects** are used to describe certain aspects of a domain. **Value Objects** do not have distinct identity. E.g., measures, quantity or describe the things in domain.
Value Objects are used to describe certain aspects of a domain. Value Objects do not have distinct identity. E.g., measures, quantity or describe the things in domain
An aggregate: a cluster of entities and value objects with a defined boundary.

An aggregate has one root entity that is known as the aggregate root.

External objects have access only to the aggregate root, and use that to pass along instructions to aggregate’s enteritis.
An aggregate: a cluster of entities and value objects with a defined boundary.

An aggregate has one root entity that is known as the aggregate root.

External objects have access only to the aggregate root, and use that to pass along instructions to aggregate’s enteritis.
An **aggregate**: a cluster of entities and value objects with a defined boundary.

An **aggregate** has one root entity that is known as the **aggregate root**.

External objects have access only to the **aggregate root**, and use that to pass along instructions to aggregate’s enteritis.
A repository can be seen as an interface that is used to access to the data persistence system (the database).

It is preferred to use a single repository per aggregate.
A repository can be seen as an interface that is used to access to the data persistence system (the database).

It is preferred to use a single repository per aggregate.
DDD - Tactic design – An example

Site engineer

- Name: name

Plant catalog

- list

Plant item entry

- Availability:

Plant item

- Plant id:
  - Plant type:
  - Plant daily cost:

Plant reservation

Engagement duration

- Start date: Date
- End date: Date
DDD - Tactic design – An example
DDD - Tactic design – An example

//Entity
public class PlantReservation
{
    PlantItem: plantItem
    ........
}

Site engineer
- Name: name

Plant catalog
- list

Plant item entry
- Availability:

Plant item
- Plant id:
- Plant type:
- Plant daily cost:

Plant reservation

Engagement duration
- Start date: Date
- End date: Date
DDD - Tactic design – An example

//Entity
public class PlantReservation
{
    PlantItem: plantItem
      ...... 
}

Entity

Value Object
DDD - Tactic design – An example

//Entity
class PlantReservation {
    PlantItem plantItem
    EngagementDuration: engDuration;
}

//Value Object
class EngagementDuration {
    Start state: Date;
    End date: Date;
    ...
}
// Entity
public class PlantReservation
{
    PlantItem plantItem
    EngagementDuration engDuration;
}

// Value Object
public class EngagementDuration
{
    Start date: Date;
    End date: Date;
    ...
}
DDD - Tactic design – An example

```
//Entity
public class PlantReservation
{
    PlantItem: plantItem
    EngagementDuration: engDuration;
}

// Value Object
public class EngagementDuration
{
    Start date: Date;
    End date: Date;
    ...
}
```
DDD - Tactic design – An example

Repository

```java
//Interface
interface PlantReservation {
    boolean isAvailable();
}

//Entity
public class PlantReservation {
    String plantItemId;
    Date engagementDuration;
}

//Value Object
public class EngagementDuration {
    Date startDate;
    Date endDate;
}
```
DDD - Tactic design – An example

Repository

<<interface>>
Plant catalog

fetchAllPlants()
fetchPlanById(id)
fetchPlansByType(type)
fetchAvailabilityofaPlant(id, engagementDuration)

//Entity
class PlantReservation
{
    PlantItem: plantItem
    EngagementDuration: engDuration;
}

// Value Object
class EngagementDuration
{
    Start date: Date;
    End date: Date;
    ...
}

Aggregate

Site engineer
Name: name

Plant catalog

Plant item
Availability:

Plant item entry

Engagement duration
Start date: Date
End date: Date

Value Object

Plant reservation

Entity

Aggregate

list
query
Additional resources


- Vernon, Vaughn. *Implementing domain-driven design*. Addison-Wesley, 2013.
Thank You
for your attention

Mohamad Gharib
mohamad.gharib@ut.ee

unitartu
tartuuniversity