



DDD workshop dag 1

Daniël Mertens

daniel.mertens@infosupport.com

- Daniël Mertens
- Consultant bij Info Support
- 5 jaar actief in DDD
- Padel
- Coding puzzles



▲ Enkele regels

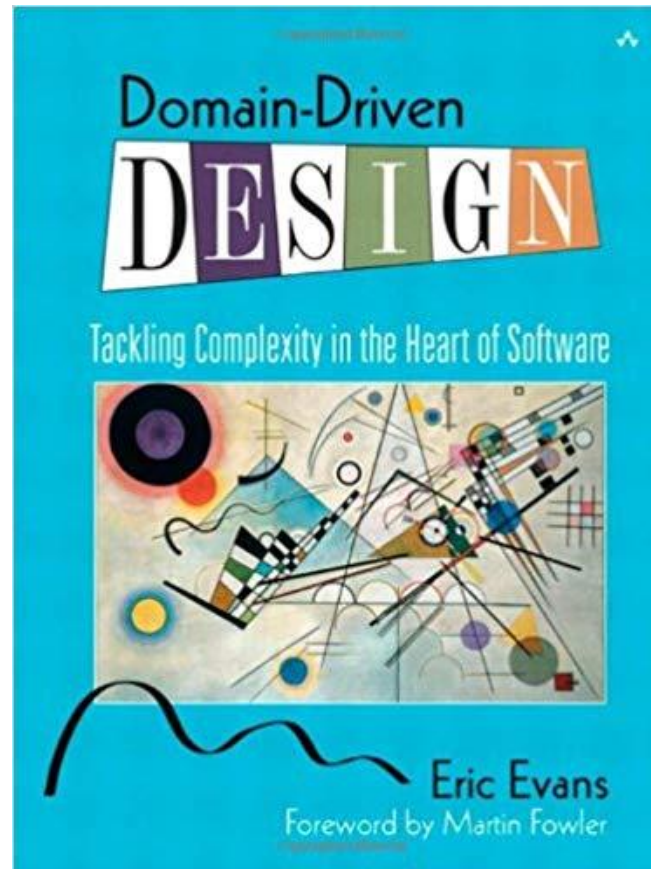
- Gebruik jouw favoriete taal: C#, Java, Python, ...
- Stel vragen wanneer je wil.
- Pauze




▲ Slides en code

Go to: <https://bit.ly/PXL-DDD-2024>

▀ DDD: Tackling Complexity



Picture adopted from Amazon



“A good domain model can be incredibly valuable,
but it’s not something that’s easy to make.
Few people can do it well, and it’s very hard to teach.”

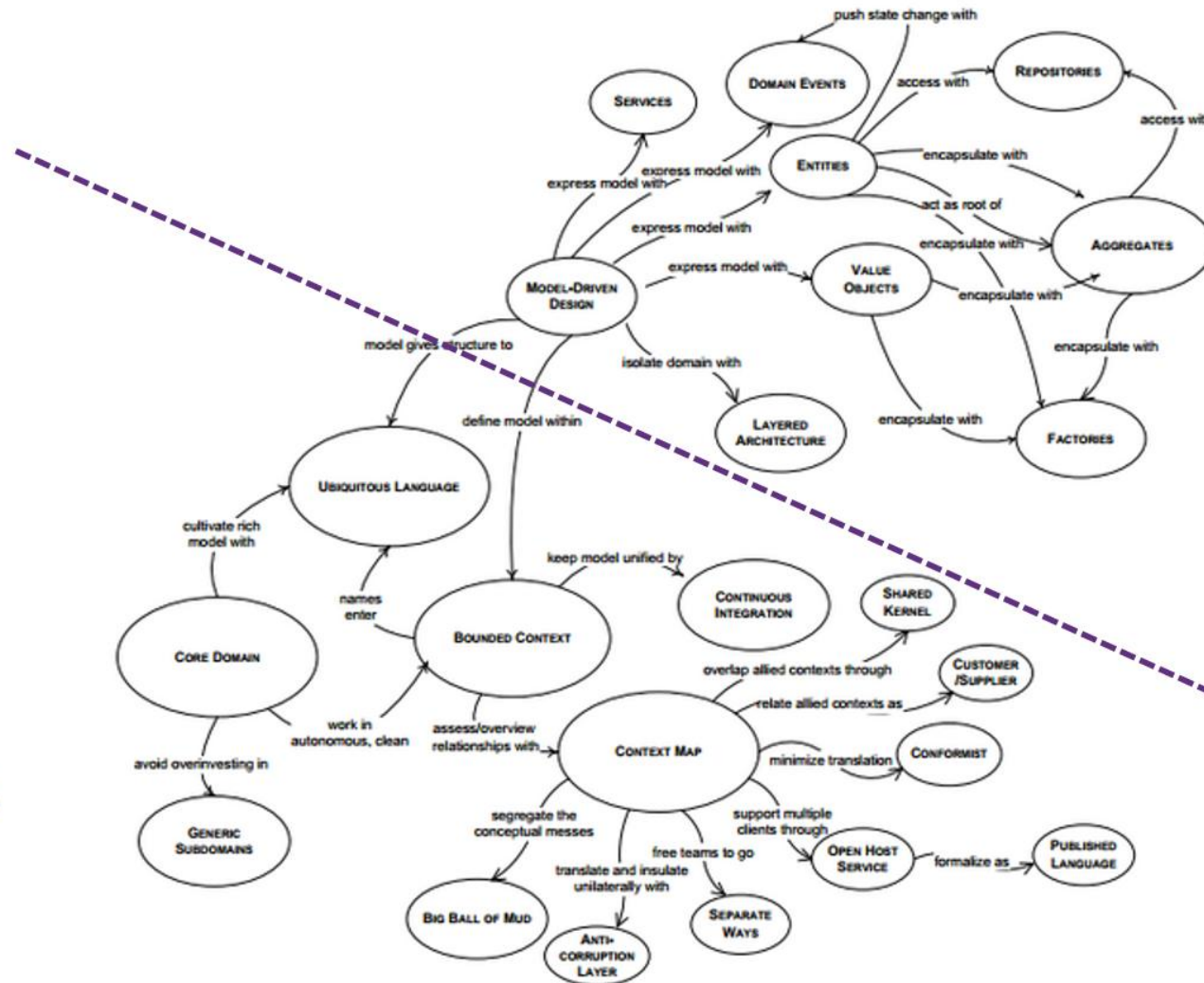
– Martin Fowler, in foreword of “Domain Driven Design”



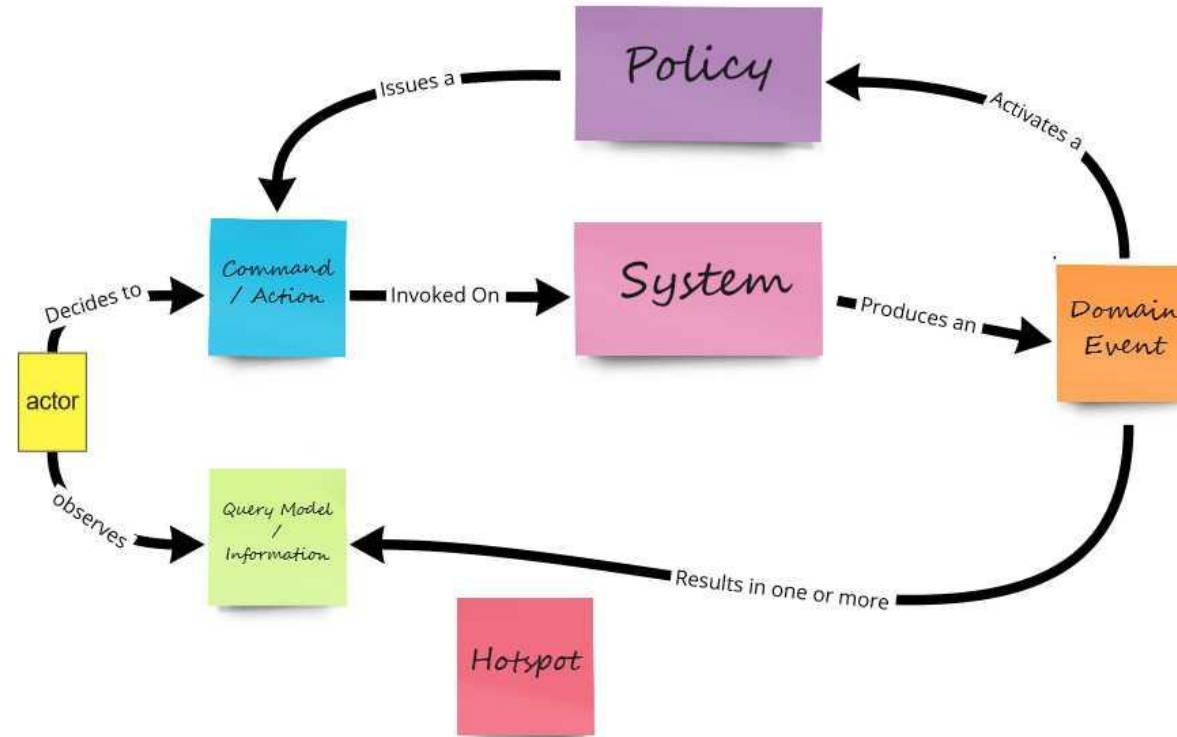
▲ Tactical vs Strategic

STRATEGIC

TACTICAL



Strategic



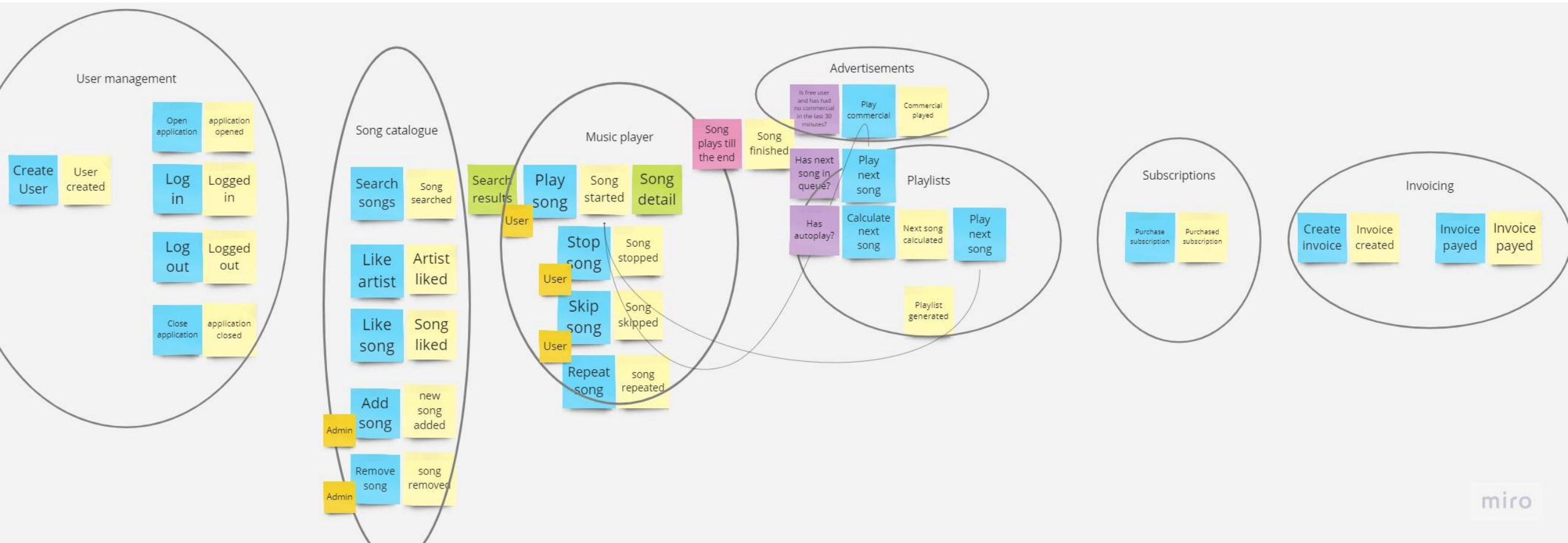
Example



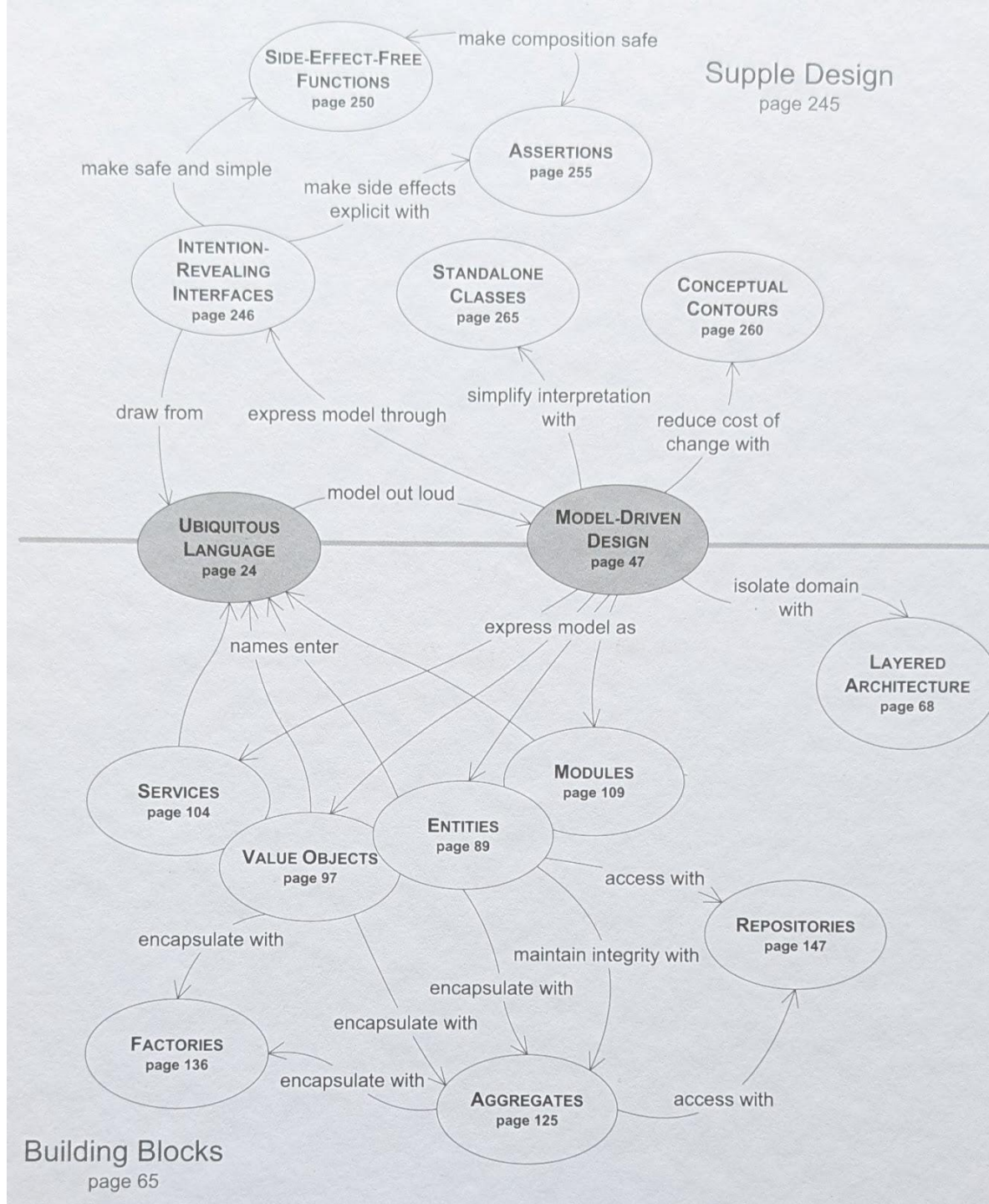


▲ Event storming

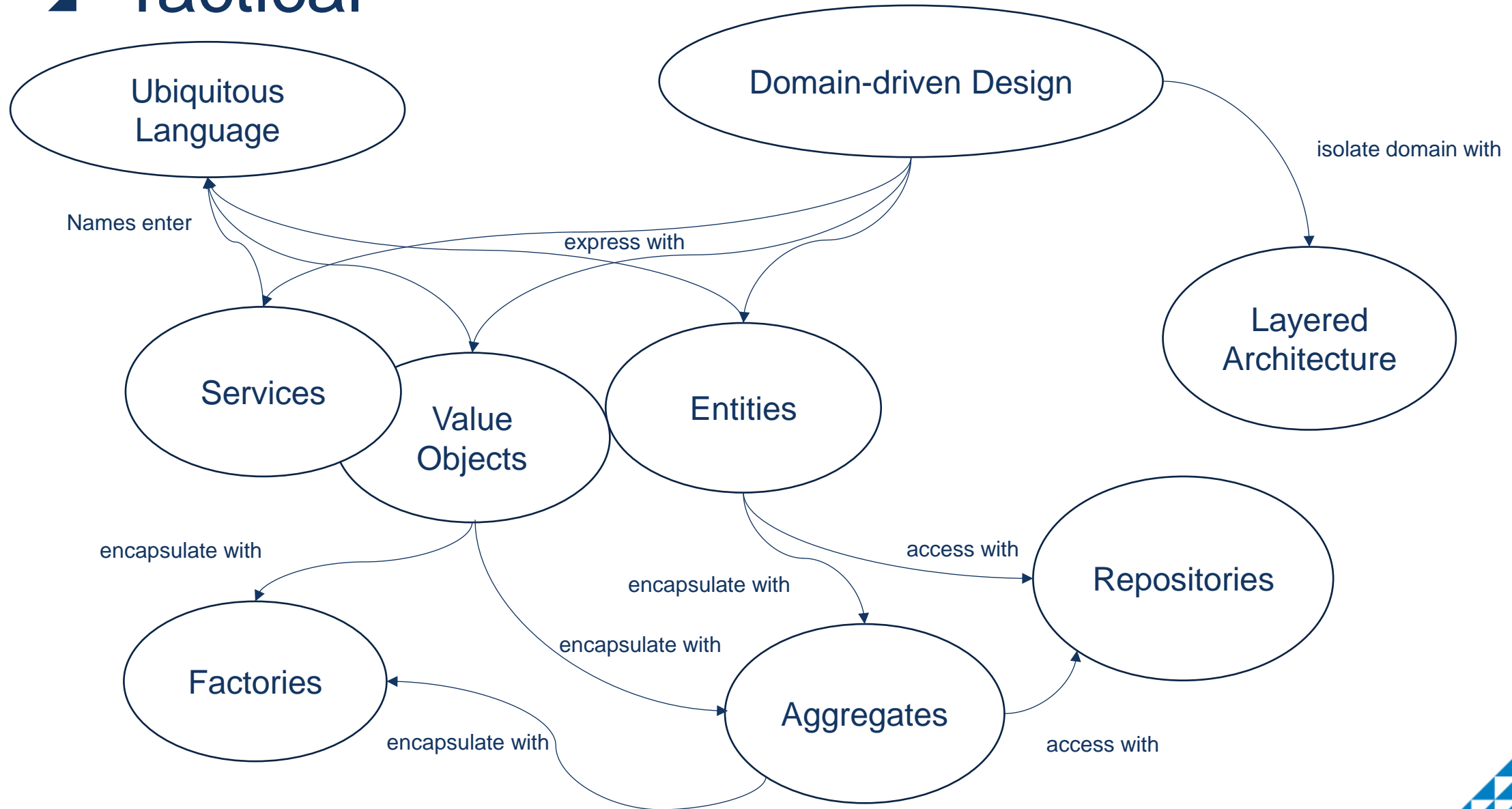
- Wordt uitgevoerd met mensen uit de hele business
- Creëert een representatie van een business proces
- Representeert waar de bottlenecks zitten
- Legt je ubiquitous language vast



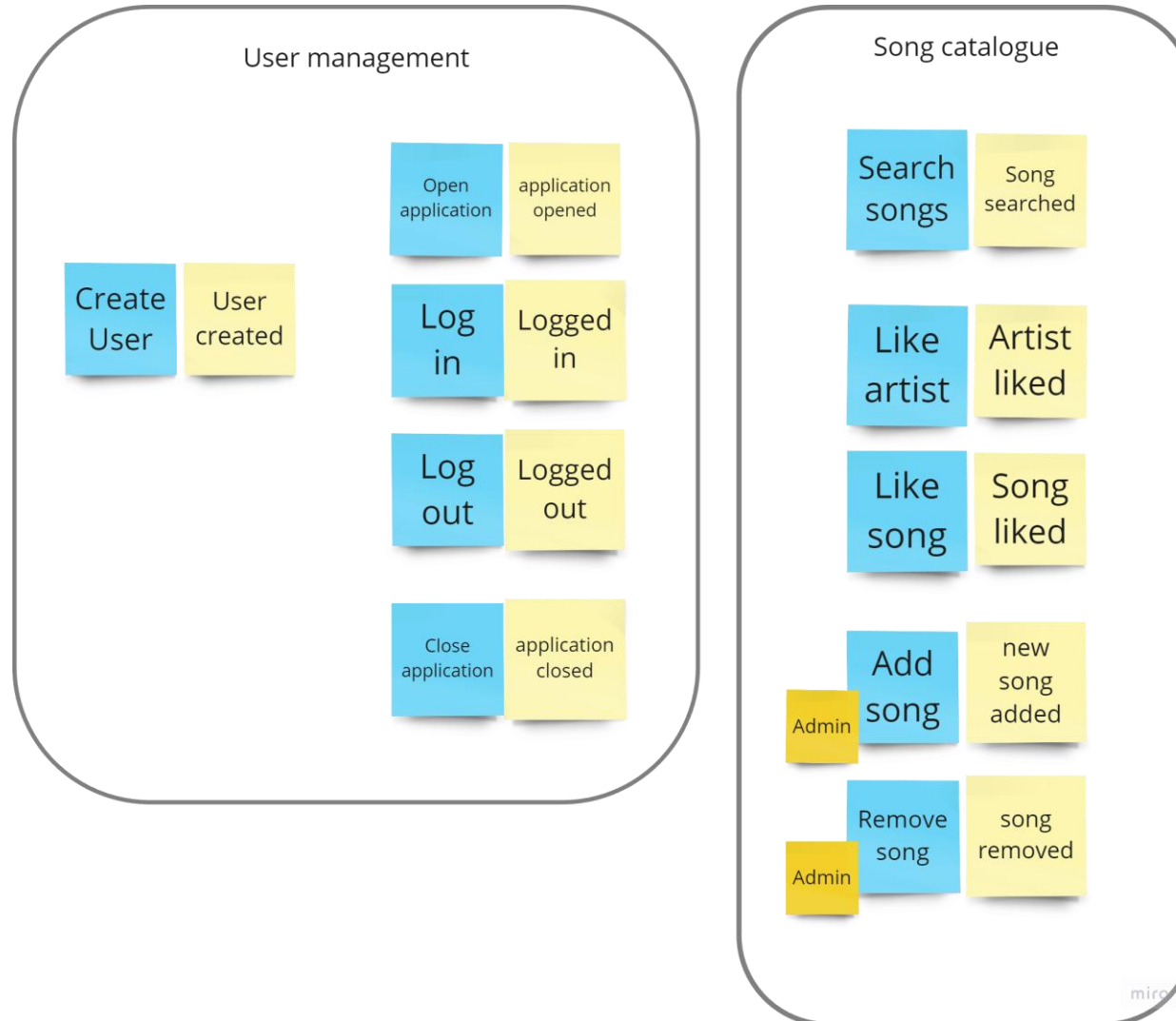
Tactical



▲ Tactical



Ons domain



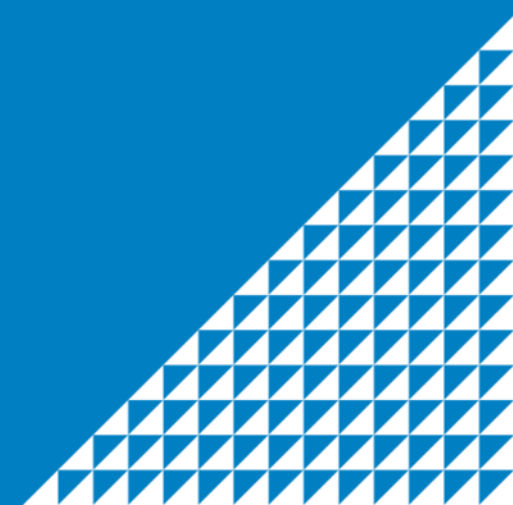


Oefening 1:

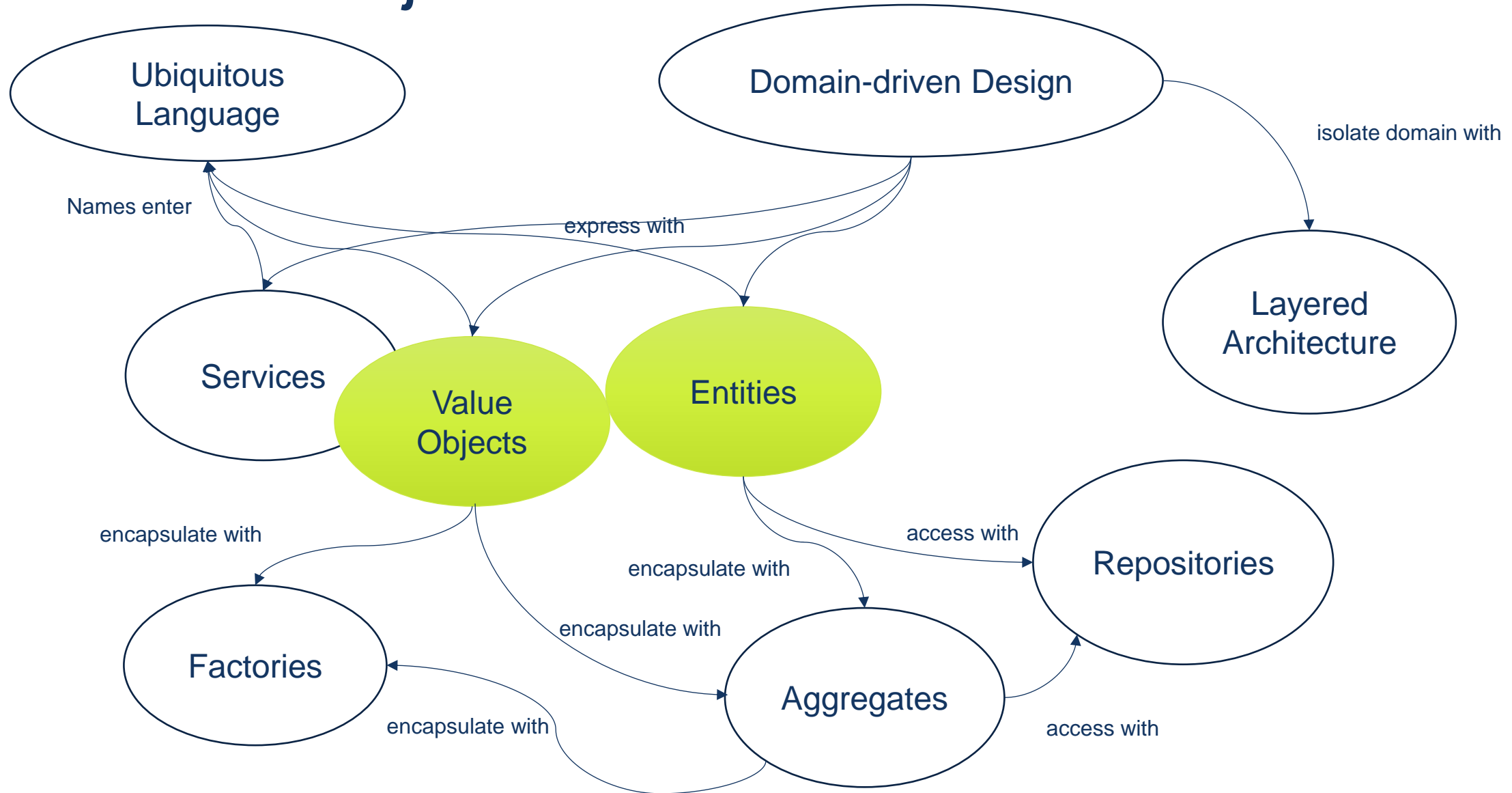
Maak een model voor de gebruikers van onze applicatie.

Maak een model voor liedjes in onze applicatie

Voeg de nodige properties toe aan de modellen.



Value objects en entities





▲ Value objects

- “Represent a descriptive aspect of the domain with no conceptual identity.”
- Immutable
- Distinguishable only on value
- Easy to use

Value objects

```
public class Age
{
    public int Value { get; }

    public Age(int age)
    {
        Value = age;
    }

    public override bool Equals(object obj)
    {
        if (obj == null || GetType() != obj.GetType())
        {
            return false;
        }

        return Value == ((Age)obj).Value;
    }
}
```



Entity

- “Objects which have continuity through a life cycle and are being distinct independent of its attribute values over time.”
- Mutable
- Has a unique id in the domain
- More complicated to use

Entity

```
public class User {  
    public UserId Id { get; }  
    public Name Name { get; }  
    public Age Age { get; }  
    public Address Address { get; }  
  
    public User(UserId id, Name name, Age age, Address address)  
    {  
        Id = id;  
        Name = name;  
        Age = age;  
        Address = address;  
    }  
}
```



Identity of Entity

- Globaal uniek
 - Toegewezen of gegenereerd tijdens creatie.
- Lokaal uniek
 - › Liedje van een album
 - Lokaal toegewezen of gegenereerd



Oefening 2:

Pas de modellen van oefening 1 aan en gebruik value objects en entities.

Voeg de nodige validatie toe op de value objects.

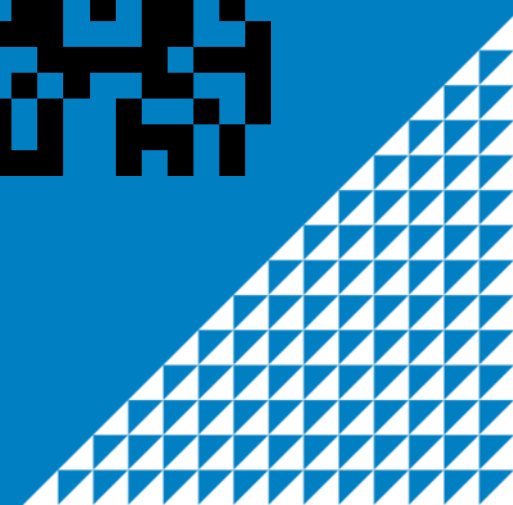
Standaarden zijn belangrijk in DDD. Creëer ook een basis abstracte klasse voor value objects en entities.

Maak entities aan. Wat merk je?

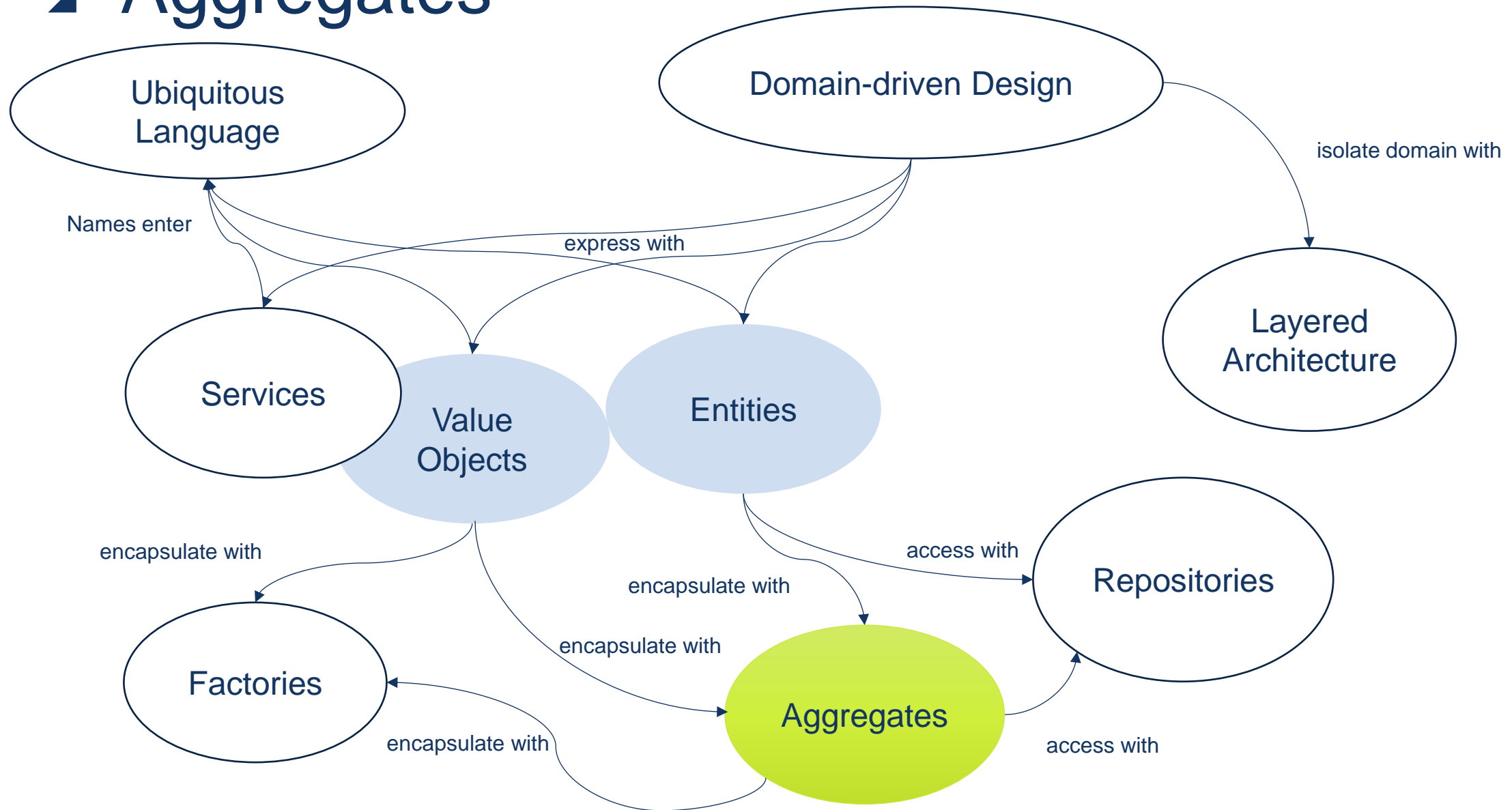


Klaar voor een quiz?

<https://app.sli.do/event/c9Qoc3AWvkFww0AvjV28PY>



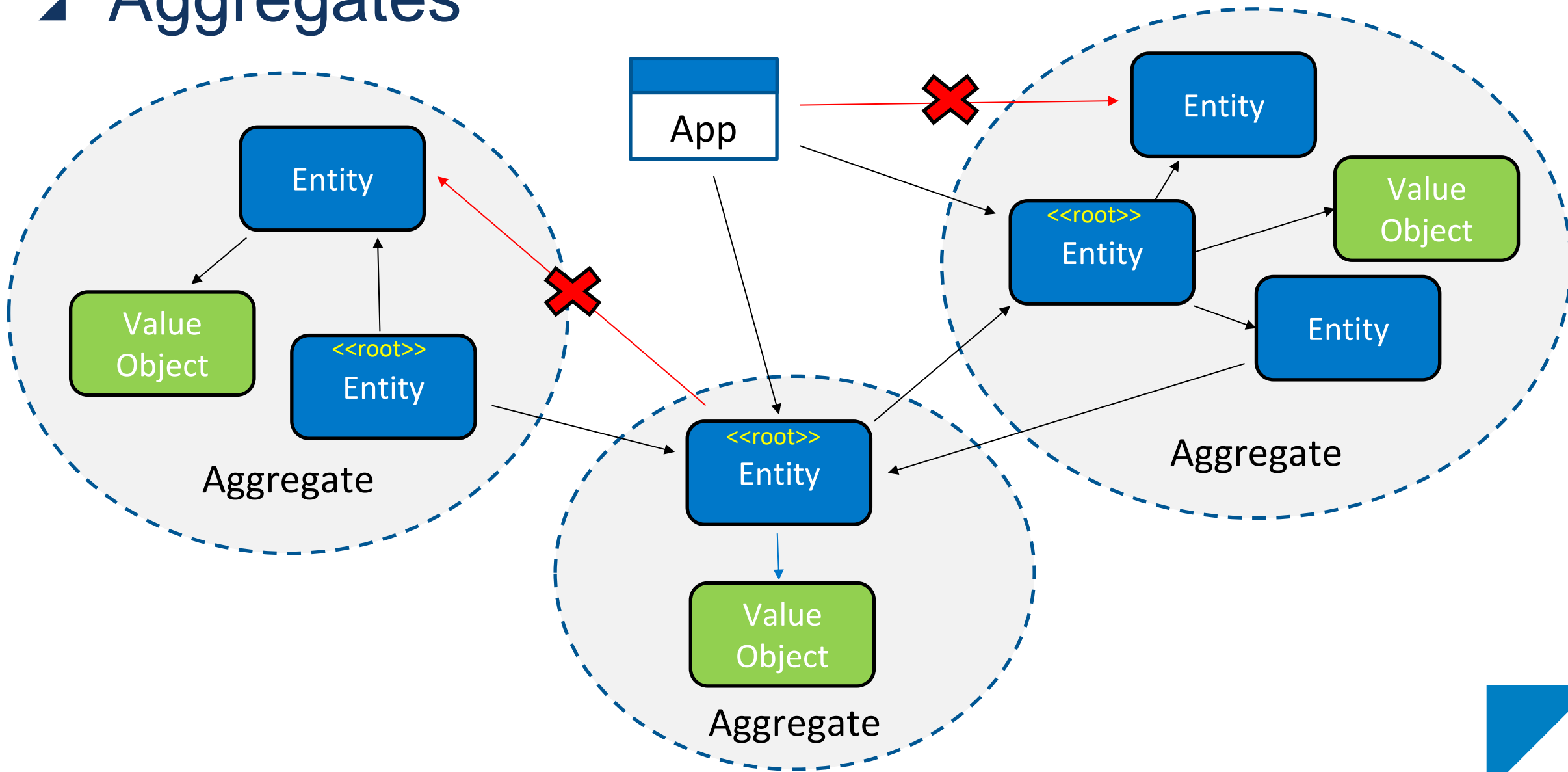
▲ Aggregates



▲ Aggregates

- “A cluster of associated objects that is treated as a unit for the purpose of data changes.”
- Zijn verantwoordelijk voor hun eigen consistency.
- Een verzameling van Entities en Value Objects.
- Er is exact 1 Aggregate Root (entity)
 - Verantwoordelijk voor de consistency
 - Enige toegang voor changes

▲ Aggregates

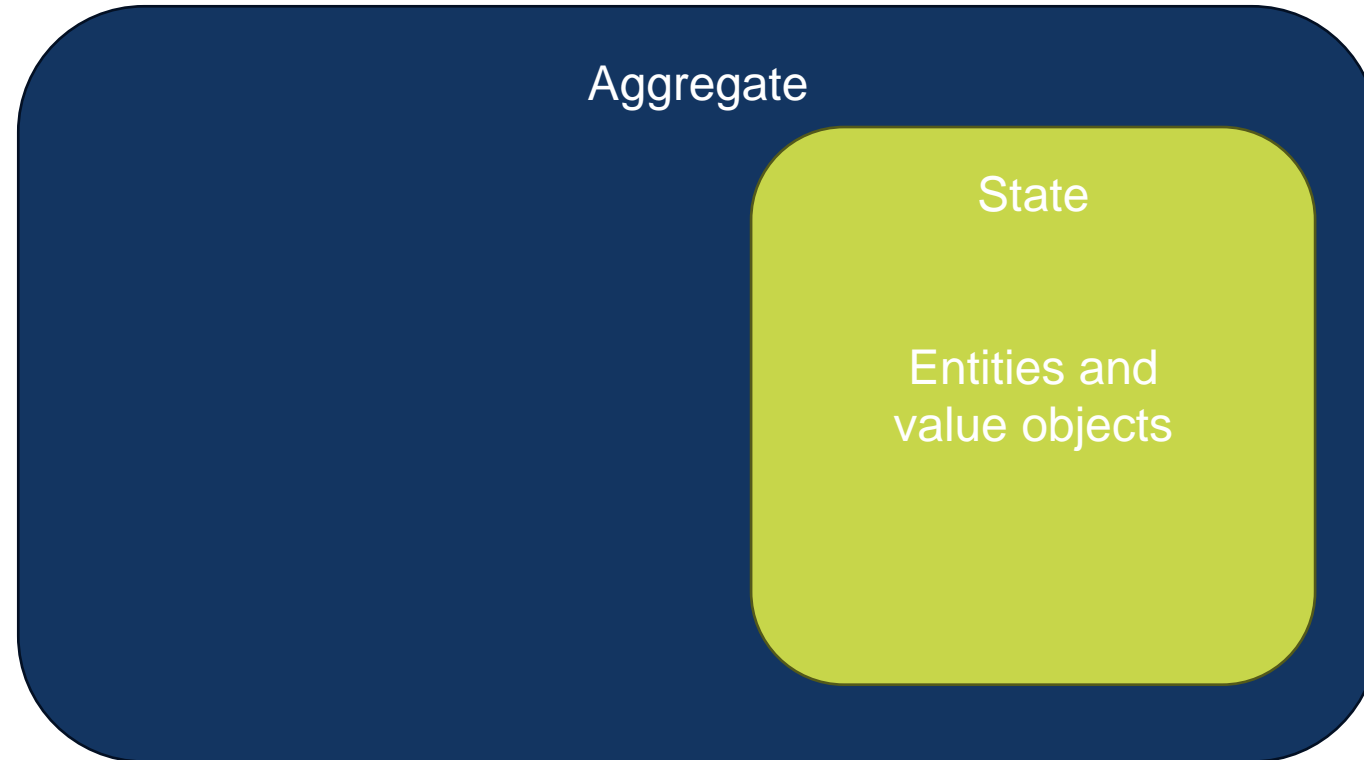


▲ Aggregate

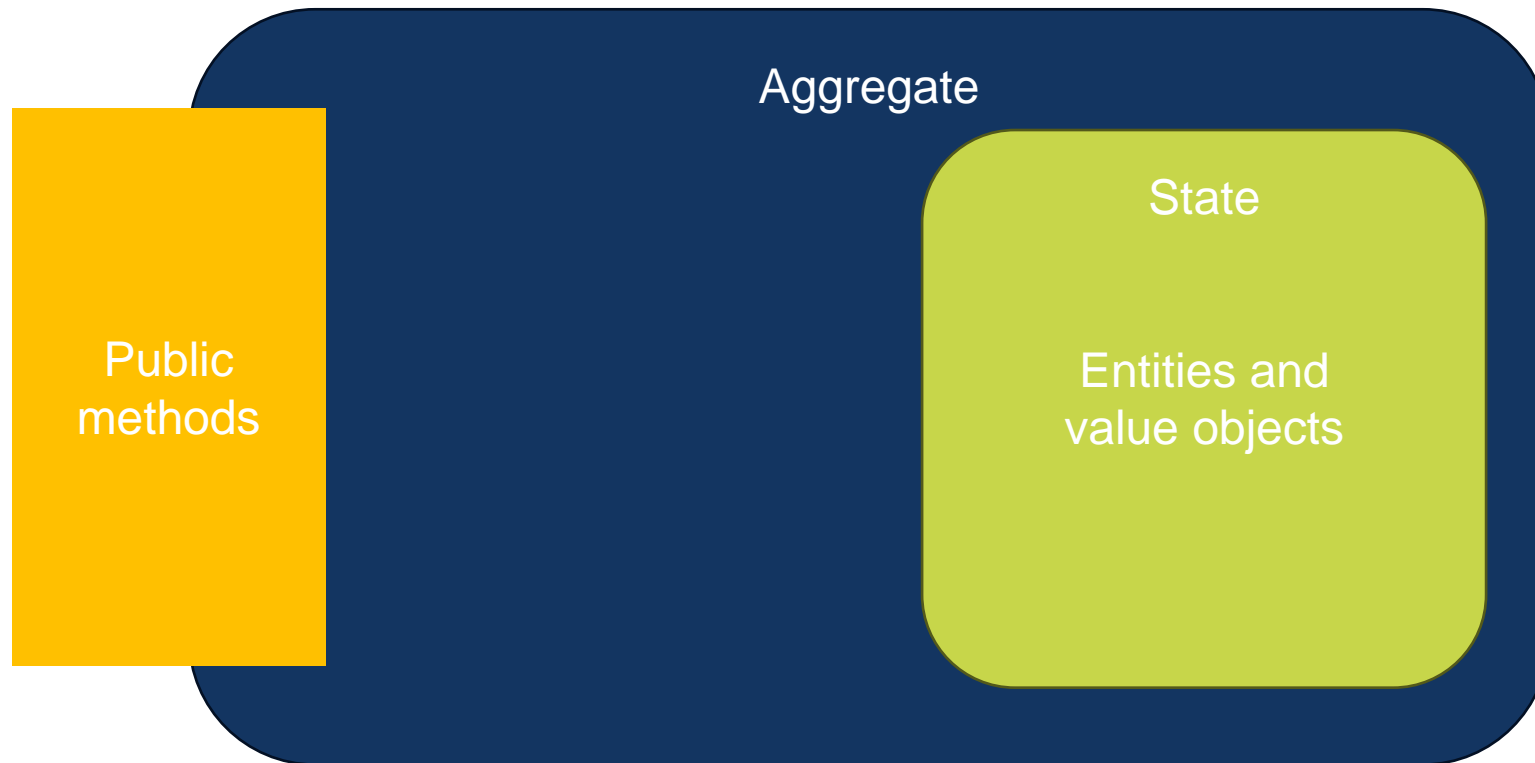
Aggregate



▲ Aggregate



▲ Aggregate



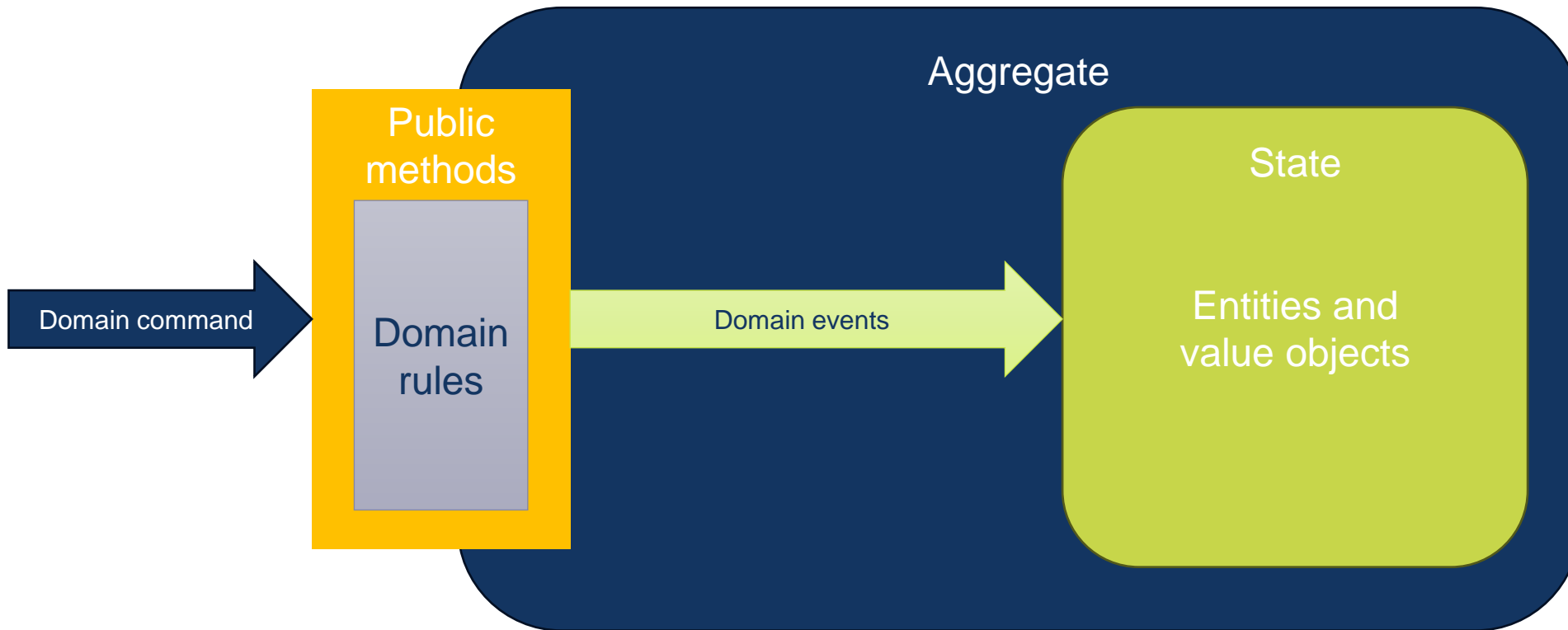
▲ Aggregate

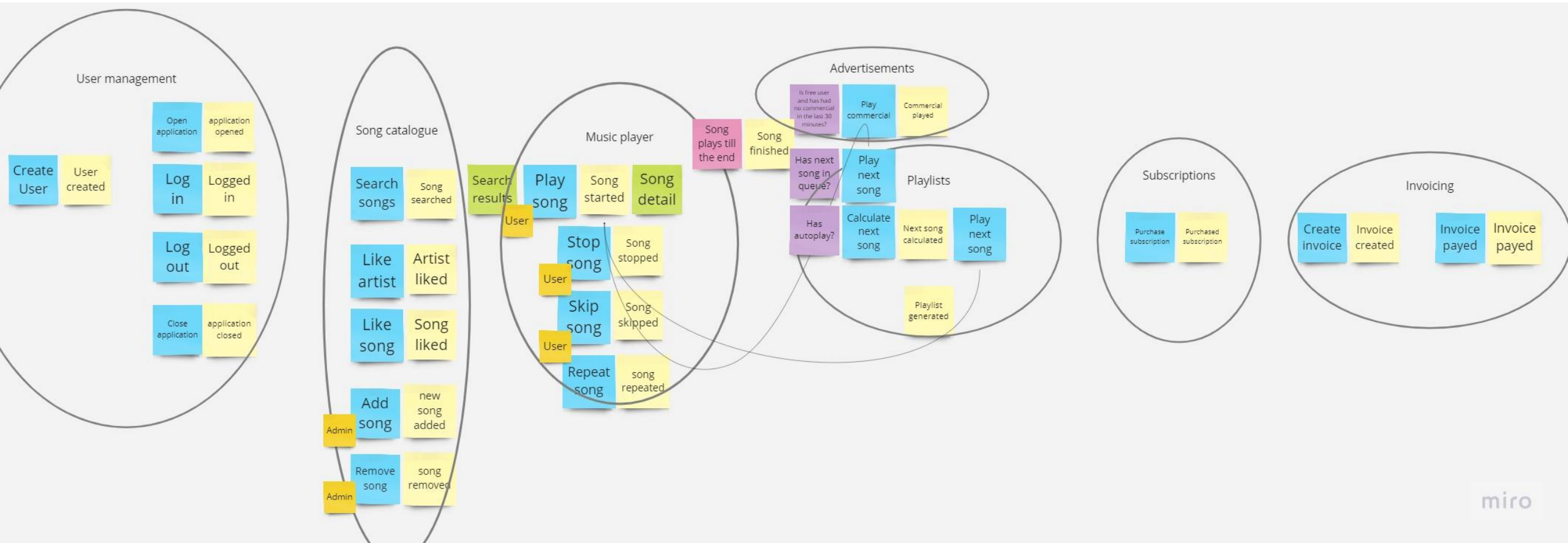


▲ Aggregate



▲ Aggregate





▲ Hands on exploration

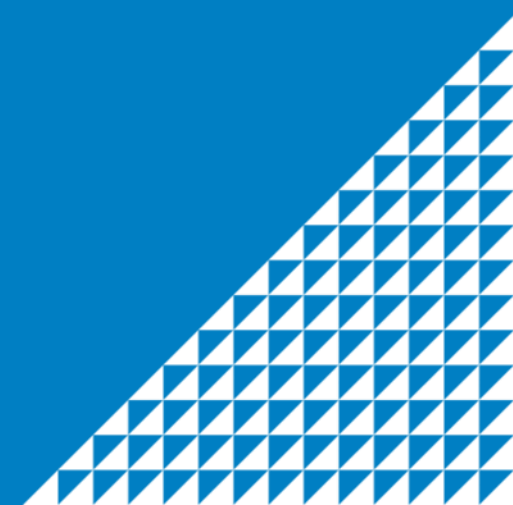
- Creatie van aggregates
- Afhandelen van commando's (domain commands)
- Validatie met domain rules
- Muteren van de state met domain events





Oefening 3:

Creëer een aggregate en implementeer commands en events.



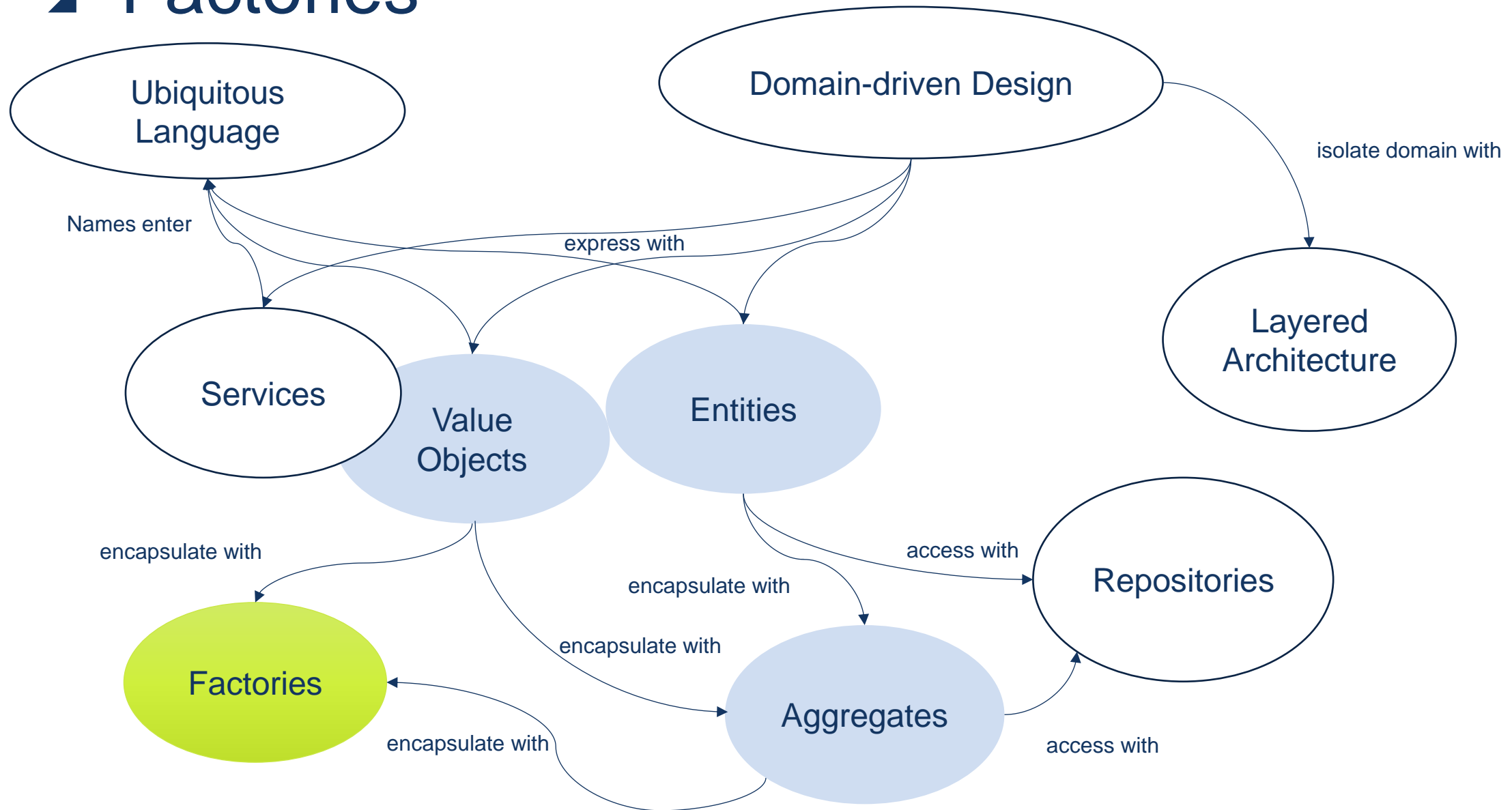


Quizen maar...

<https://app.sli.do/event/c9Qoc3AWvkFww0AvjV28PY>



Factories

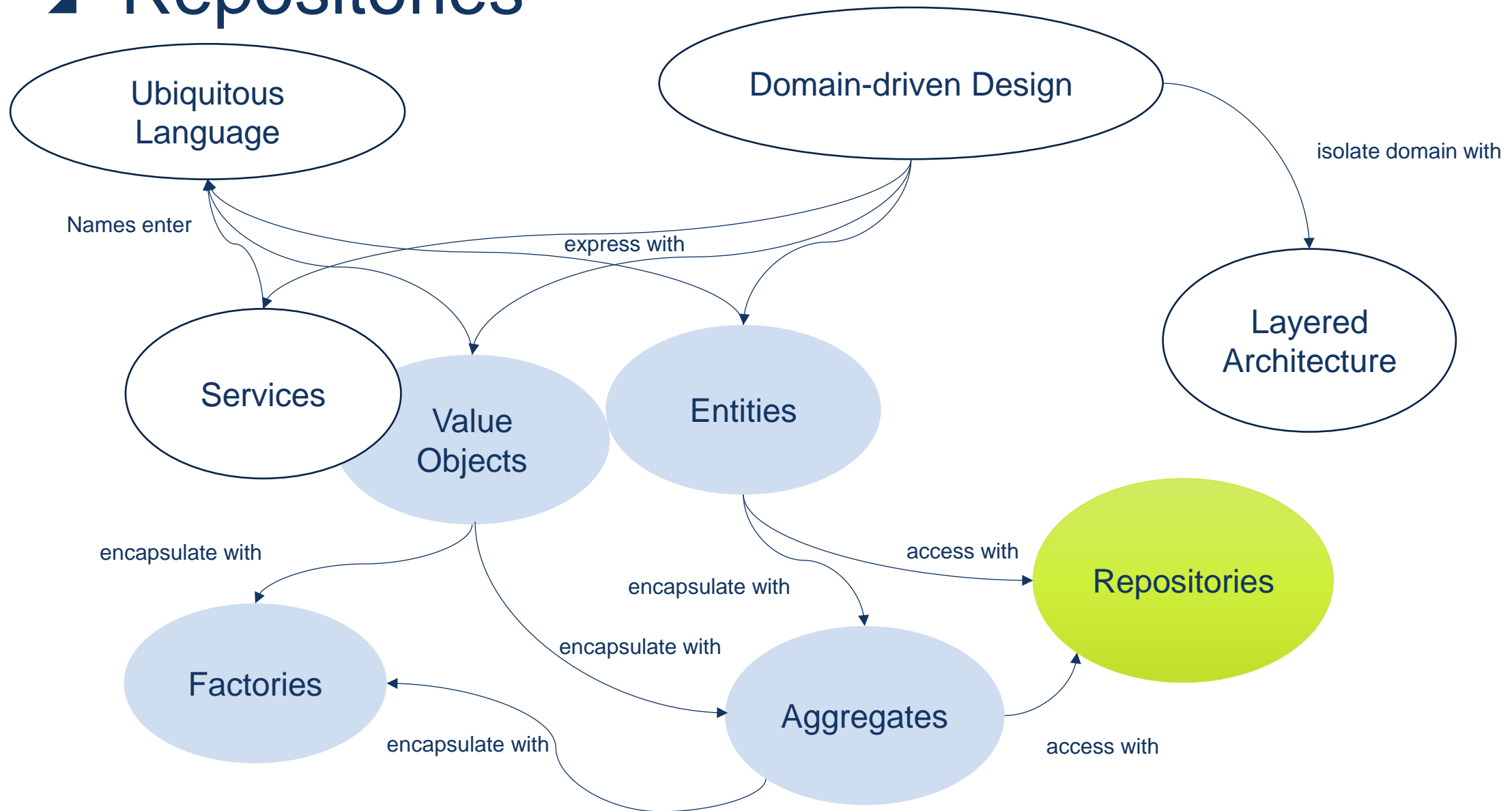


▲ Factories

- “Aggregates can become complex object graphs of which the construction should be the responsibility of factories.”
- Factories worden gebruikt wanneer normale creatie van een value object of entity to complex wordt.
- Gebruikt vaak DI om services aan te roepen tijdens creatie.



▲ Repositories

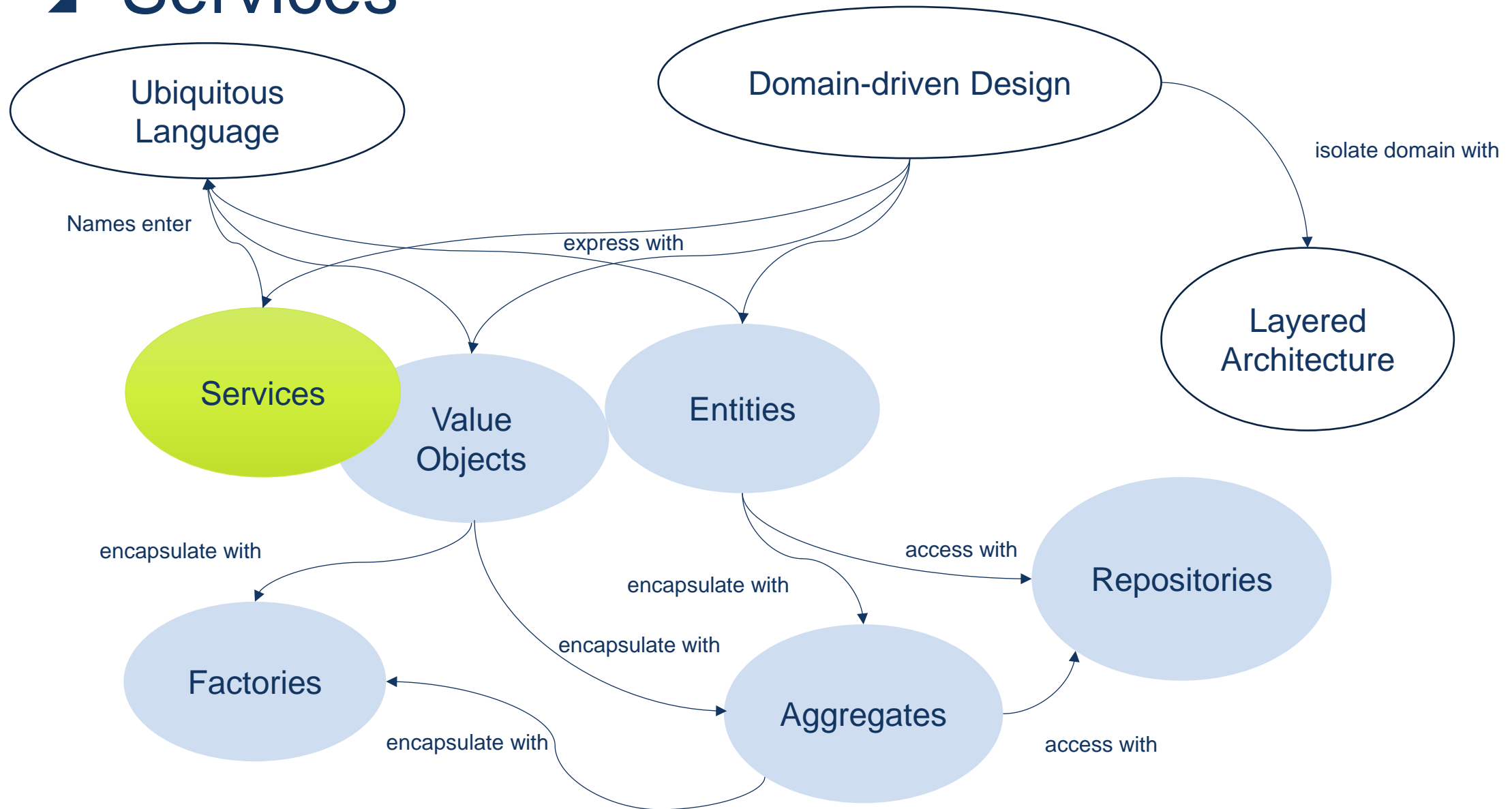




▲ Repositories

- “Represent all objects of a certain type as a collection with querying capabilities.”
- Houdt entities bij.
- Kan connecteren met
 - Database
 - Cache
 - Text file
 - ...

Services





▲ Services

- “A process or transformation in the domain which is not a natural responsibility of an Entity or a Value Object.”
- Domain services helpen met
 - het encapsuleren van domain logica
 - het ophalen van informatie binnen of buiten het domein dat nodig is voor validatie
 - ...

Services

```
public void ChangeName(Name name, INameValidator nameValidator,
                        INameQueryService nameQueryService)
{
    if (!nameValidator.IsValid(name))
    {
        throw new InvalidNameException("Invalid name");
    }

    if (nameQueryService.Exists(name))
    {
        throw new NameAlreadyExistsException("Name already exists");
    }

    ...
}
```

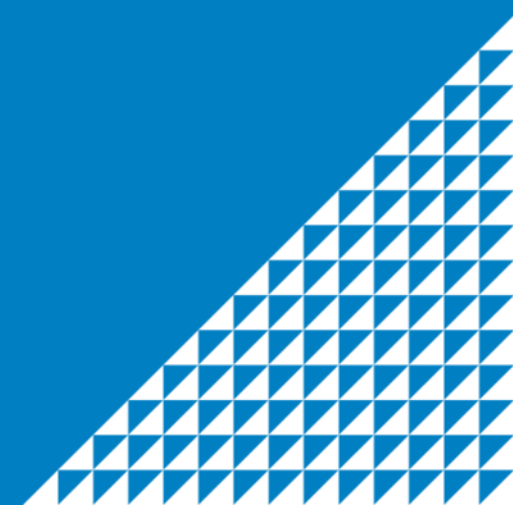


Oefening 4:

Add some domain services to your aggregates or value objects.

Add a factory for a value object

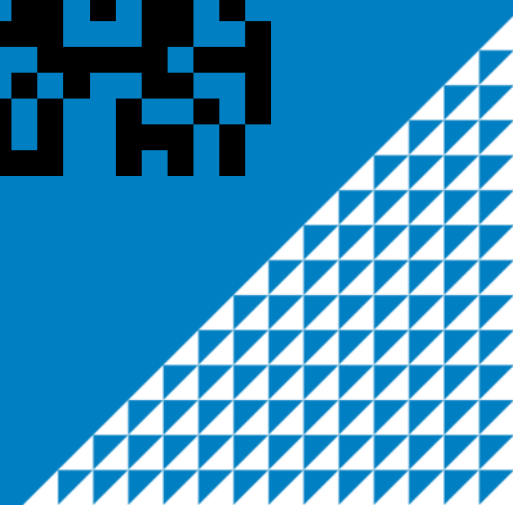
What do you notice?



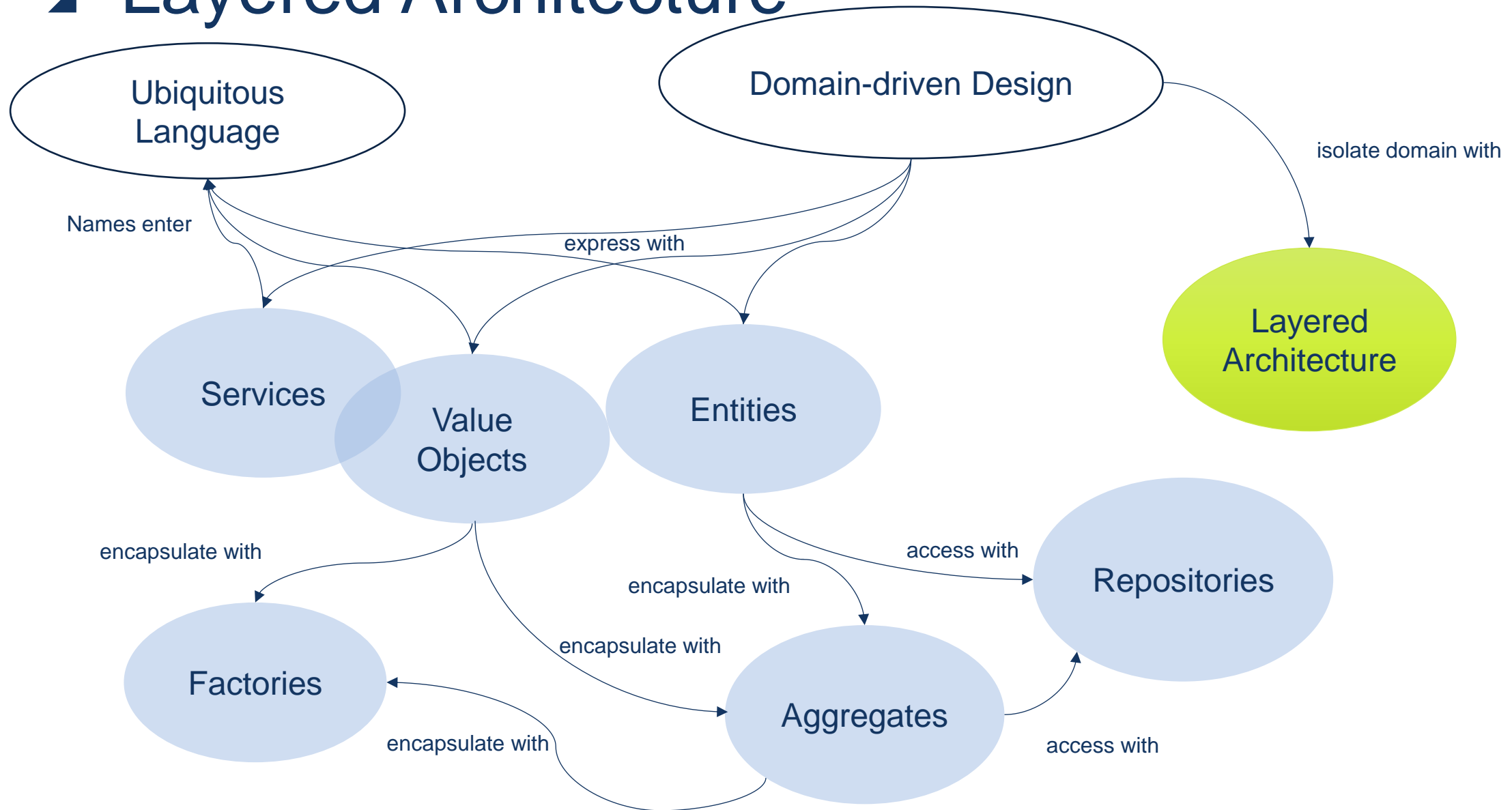


Quiz't je dat?

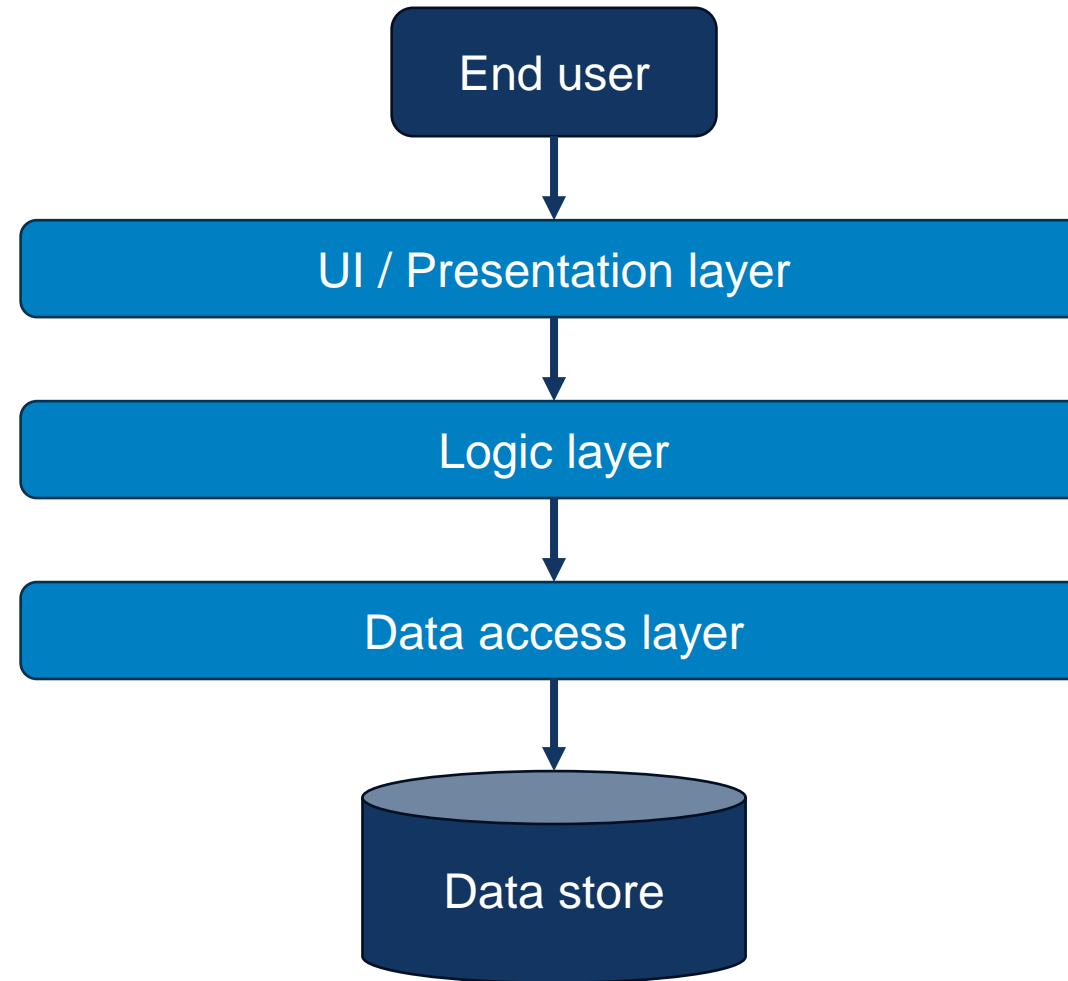
<https://app.sli.do/event/c9Qoc3AWvkFww0AvjV28PY>



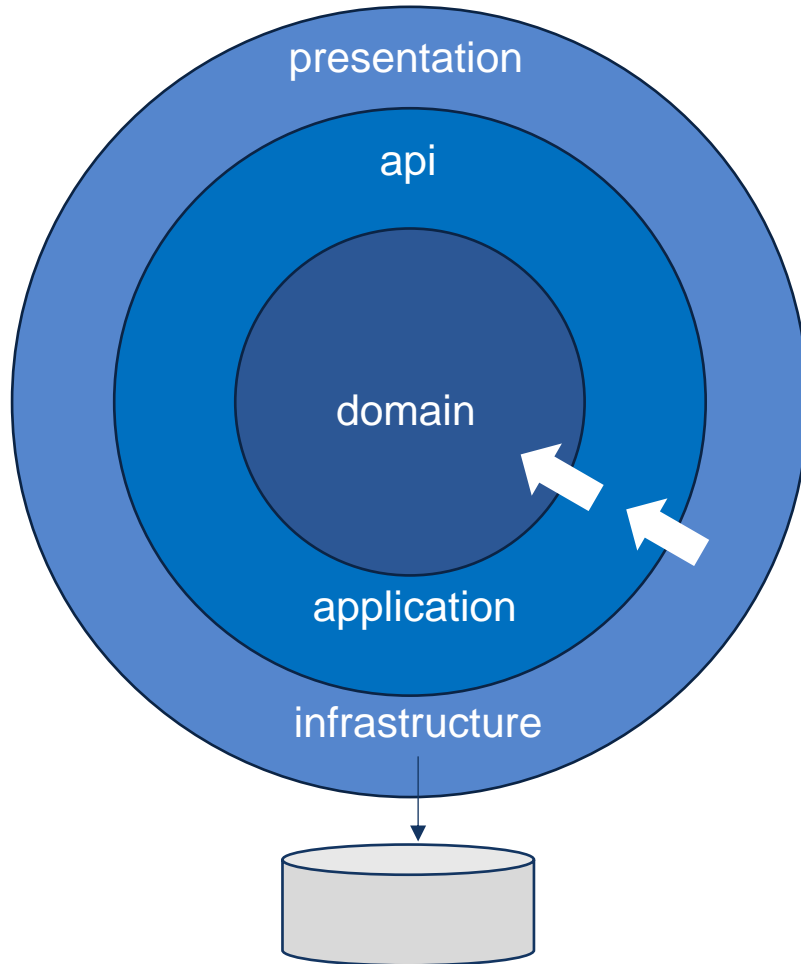
Layered Architecture



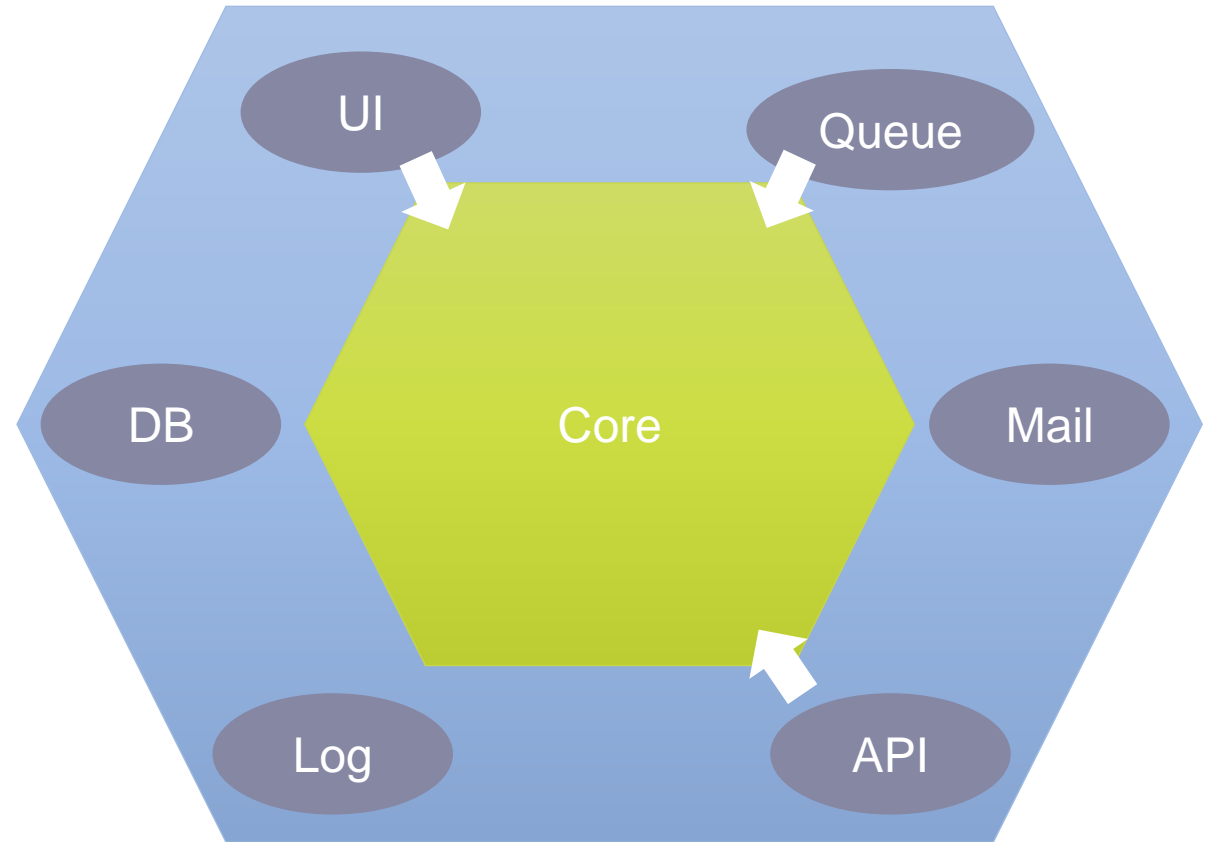
Architecture



Architecture



Onion architecture



Hexagonal architecture



▲ Architecture

- Onion/hexagonal architecture zorgt voor beter separation of concerns
 - Domain logica staat centraal
 - Afhankelijkheid is naar binnen toe (inversion of control)
 - Domain logica is niet afhankelijk van de infrastructuur.
- Alles is losgekoppeld via interfaces
 - Interfaces beschreven in de application en domain layer
 - Losse koppeling met de concrete implementatie
- Mocking is simpel in je testen.



▲ Application services

- Use case oriented
 - Gefocust op het vervullen van een business use case
- Bevat geen domain logica
- Allerlaatste moment waarop er geconverteerd wordt naar domain types
- Beschrijft een process. Voornamelijk stateless services.



▲ Application services

Typisch volt een application service de volgende stappen:

- 1) Validatie van de input
- 2) Ophalen van domain entities
- 3) Uitvoegen van domain commands op een aggregate
- 4) Opslaan van domain entities
- 5) Notificaties verzenden van belangrijke events

▲ Hands on exploration

- Creating an application layer
- Creating an ports layer





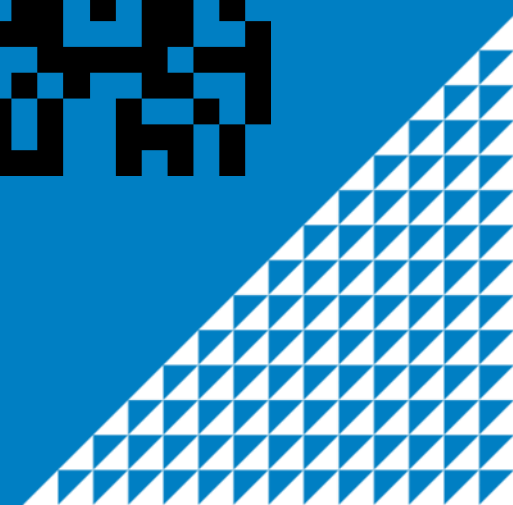
Oefening 5:

Expand your ports layer. Implement an API or other interface to make changes domain objects

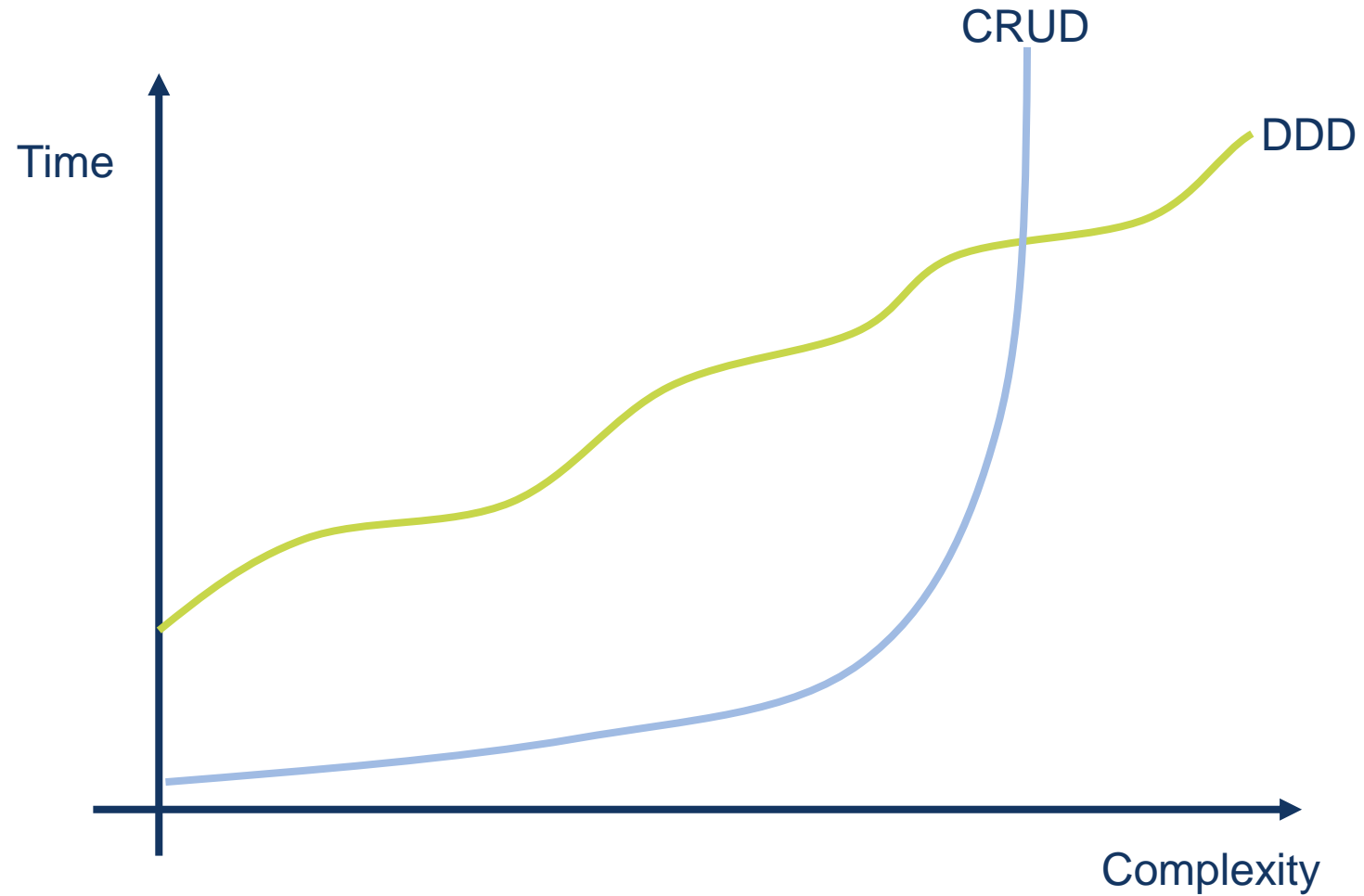


De laatste punten

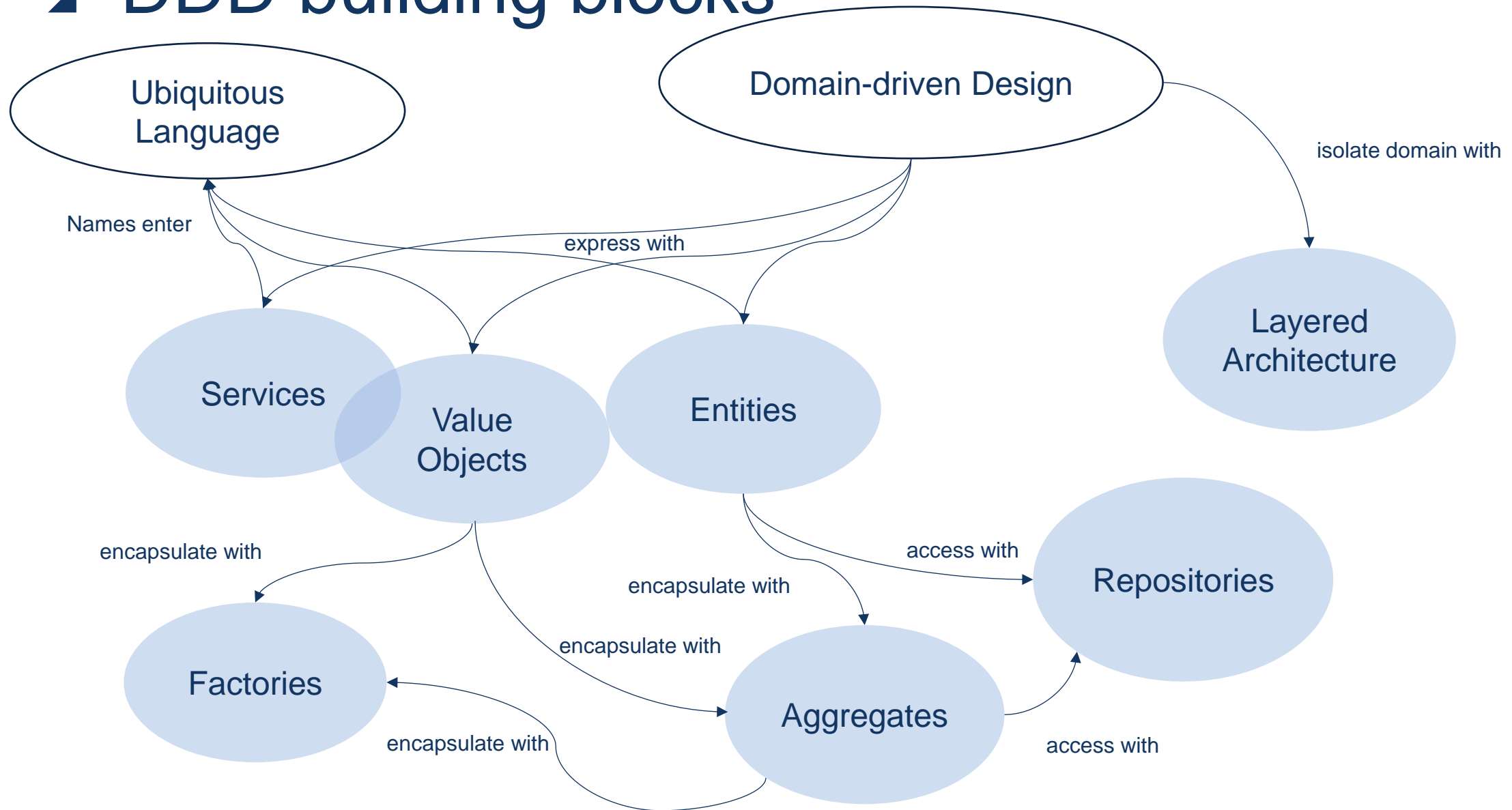
<https://app.sli.do/event/c9Qoc3AWvkFww0AvjV28PY>



▲ DDD versus CRUD



DDD building blocks



▲ Nog op zoek naar een leuke, creatieve stage opdracht?



- Maak een spel waar een LLM de controller is van het spel. Jij geeft instructies, de LLM voert uit.



Tot volgende week voor dag 2

Daniel.mertens@infosupport.com

