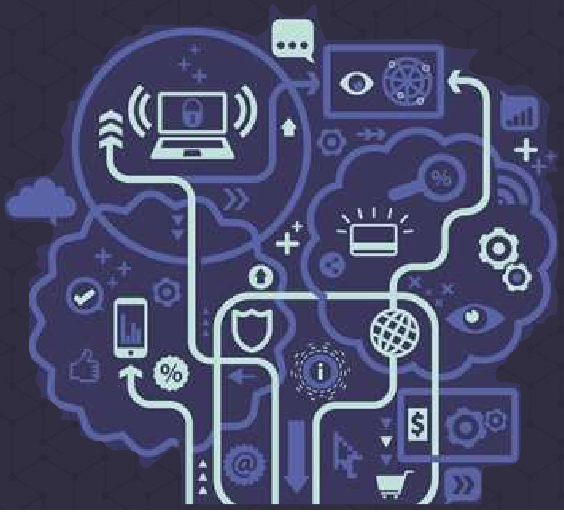
PREGRADO

UNIDAD 3 | HIGH-LEVEL SOFTWARE DESIGN & PATTERNS

DOMAIN DRIVEN DESIGN



SI720 | 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

STRATEGIC

TACTICAL

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

AGENDA

INTRO

DOMAIN DRIVEN DESIGN

STRATEGIC

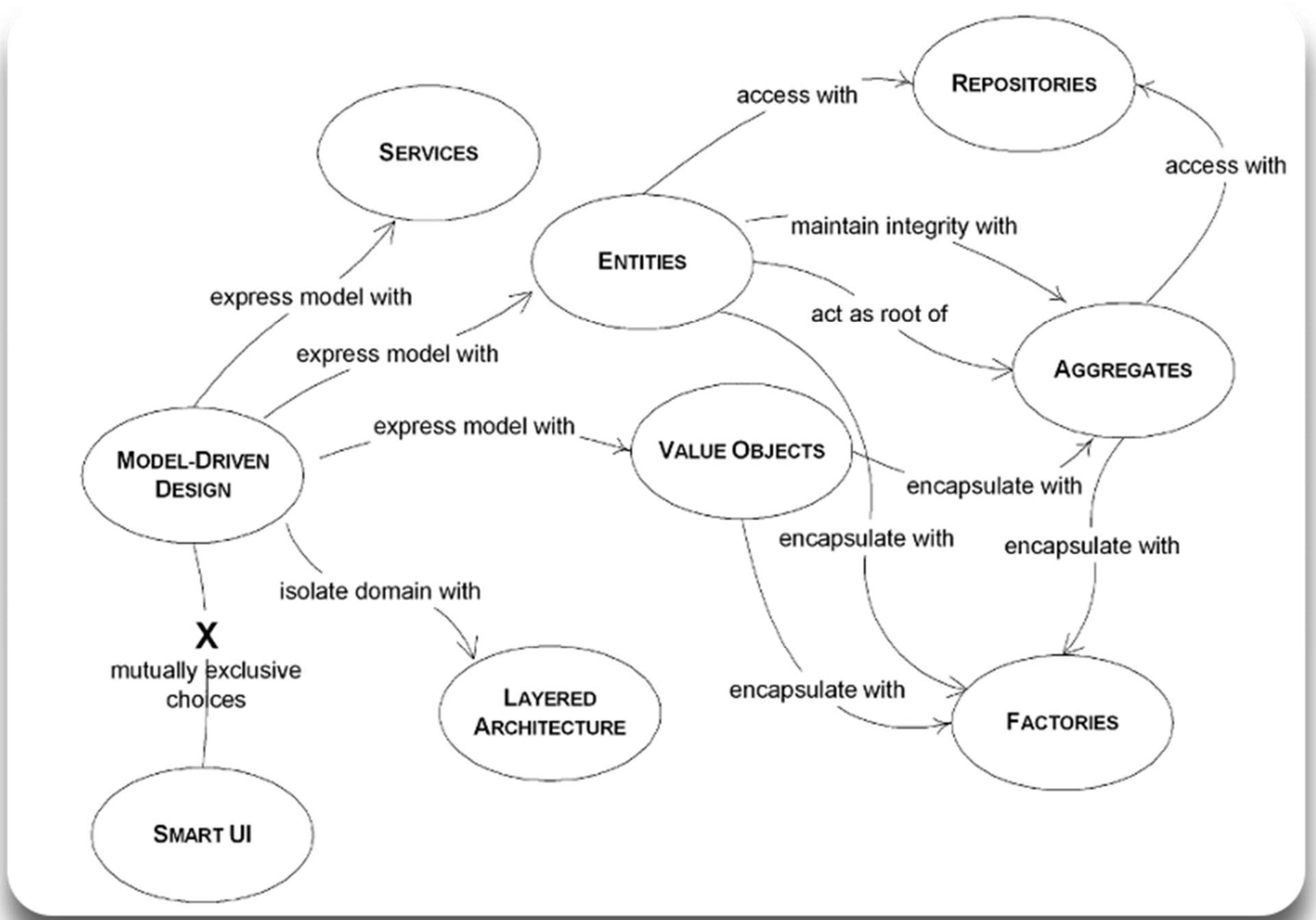
TACTICAL

What is the Domain Driven Design?

Initially introduced and made popular by programmer Eric Evans in his 2004 book, ***Domain-Driven Design:*** *Tackling Complexity in the Heart of Software*, domain-driven design is the expansion upon and application of the domain concept, as it applies to the development of software.

It aims to ease the creation of complex applications by connecting the related pieces of the software into an ever-evolving model. DDD focuses on three core principles:

- Focus on the core domain and domain logic.
- Base complex designs on models of the domain.
- Constantly collaborate with domain experts, in order to improve the application model and resolve any emerging domain-related issues.


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

AGENDA

INTRO

DOMAIN DRIVEN DESIGN

STRATEGIC

TACTICAL

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

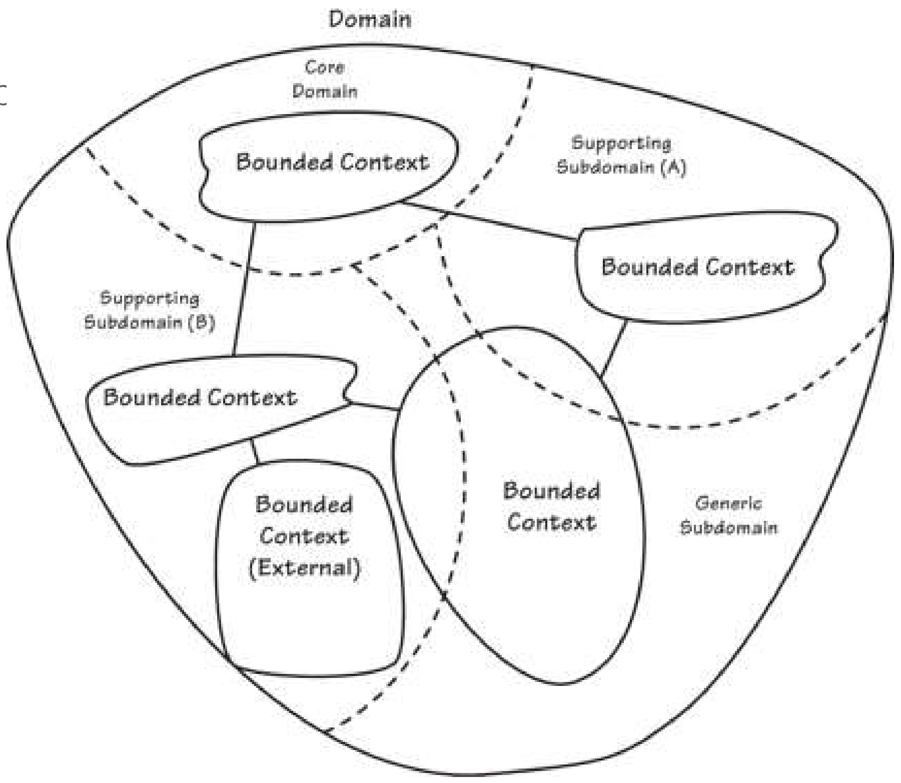
A description of a boundary (typically a subsystem, or the work of a specific team) within which a particular model is defined and applicable.

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



AGENDA

INTRO

DOMAIN DRIVEN DESIGN

STRATEGIC

TACTICAL

Tactical

According to Eric Evans, there are three (3) options for modeling our domains and they are:

- Entity
- Value Object
- Service

Entity

Martin Fowler has this definition of what Evans calls Entity Object.

“Objects that have a distinct identity that runs through time and different representations. You also hear these called ‘reference objects’.”

Entities are actors important enough to be unique and have identifiers.

Value Object

Martin Fowler defines the Value Object like this

“Objects that matter only as the combination of their attributes. Two value objects with the same values for all their attributes are considered equal.

Value Objects do not have identities and should be objects less complex than entities and **immutable**, thus facilitating their creation and manipulation.

Service

A service is an operation or form of business logic that doesn't naturally fit within the realm of objects.

An object where the business logic of one or more entities is concentrated. When using this strategy make sure your Service is stateless, meaning it has no state and only pure functions.

Domain Events

Domain Events strategy is nothing more than an event fired from a Context with information that may interest other contexts

Aggregates

According to Martin Fowler, a *DDD aggregate is a cluster of domain objects that can be treated as a single unit. An example may be order and its line-items, these will be separate objects, but it’s useful to treat the order (together with its line items) as a single aggregate.*

An aggregate is a set of Entities and Value Objects that do not make sense alone.

Repository

Repositories will be responsible for storing our Entities, Aggregates and Value Objects.

In the DDD context, a repository can be, from a database to a simple list (collection) in Java memory, where you can add your objects through the .add method, retrieve through the .get method maintaining the same state when it was inserted.

Factory

DDD factories are nothing but the use of Creation Patterns of GoF (Gang of Four), for example.

Some of these patterns are: Factory Method, Abstract Factory, and Builder.

These patterns advocate the creation of an interface or class that will be responsible for creating your objects.

Continuous integration

Domain-driven design also heavily emphasizes the practice of continuous integration, which asks the entire development team to use one shared code repository and push commits to it daily (if not multiple times a day).

An automatic process executes at the end of the work day, which checks the integrity of the entire code base, running automated unit tests, regression tests, and the like, to quickly detect any potential problems that may have been introduced in the latest commits.

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.

RESUMEN

Recordemos

Domain Driven Design

Strategic

Tactical

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