ARQUITECTURA DE SOFTWARE

UNIVERSIDAD
DE ANTIOQUIA
1 8 0 3

DOCENTE
Robinson Coronado

Bryan Zuleta Vélez

Facultad de Ingeniería
Ingeniería de Sistemas
Universidad de Antioquia
2019-2

Es un enfoque de Desarrollo de Software en donde se conecta la implementación, los conceptos de modelos y el núcleo del negocio.
Sus principios se basan en:
-Colocar los modelos y reglas de negocio de la organización, en el core de la aplicación
-Basar nuestro dominio complejo, en un modelo de software
-Se utiliza para tener una mejor perspectiva a nivel de colaboración entre expertos del dominio y desarrolladores, para concebir un software con los objetivos bien claros

El Diseño guiado por el dominio se puede dividir en dos partes que serían el diseño táctico y el diseño estratégico, en donde el diseño táctico se puede interpretar como la parte de código y el estratégico en cómo hablarle y manejar las palabras hacia las personas y también hacia los equipos de trabajo
Entonces el **MODEL-DRIVEN DESIGN** Son los Services, Layered Architecture, Domain Events, Entities, Value Objects, Aggregates, Repositories y Factories
y en el **STRATEGIC DESIGN** son Ubiquitous Language, Big Ball Of Mud, Anticorruption Layer, Bounded Context, Context Map, Separate Ways, Customer/Supplier Teams, Continuous Integration, Shared Kernel, Conformist, Open Host Service y Published Language.
Entonces cada diseño tiene varios conceptos y vamos a mencionar algunos, por la parte de diseño táctico tenemos el:

MODEL-DRIVEN DESIGN: Qué resumidamente es como vamos a modelar el proyecto, en una relacion seria coger un pedazo de plastilina y darle forma a lo que buscamos o en palabras más técnicas es dirigir un diseño en base a modelos existentes y eso lo expresamos con servicios que podrían ser los casos de usos. también manejamos una **entidades** y unos **value-objects** que son las clases que instanciamos en el proyecto, en que se diferencian estas dos pues con un ejemplo sencillo sería que una entidad tiene un identificador y el value-object no, como si fueras a un cine y las sillas están numeradas sería una entidad, si no lo están entonces sería lo otro porque no tiene entidad. de ahí vienen los agregados que sería que es un conjunto que encapsula las entidades y los value-objects con una lógica o semántica, un ejemplo sería que un vídeo tenga comentarios, como los comentarios están dentro del vídeo y todo esto se guarda en **repositorios** en donde se tendrá acceso a todo lo que hemos hablado, y de ahí entramos a eventos de dominio que sería algo que ha pasado, que en un ejemplo sería que has publicado un vídeo a youtube o un usuario se ha registrado a algún **servicio**, luego se encapsula en factorías .y todo esto se maneja en una arquitectura en capas, y todo esto acabado de explicar está estructurado en un l**enguaje ubicuo** es simplemente el modelo de negocio reflejarlo en el código. y ahì entramos al **Bounded Context** es un poquito extenso de explicar pero resumidamente es que cuando se tiene un modelo muy grande se dividan en contextos delimitados poniendo de forma explícita la relación entre ellos. un ejemplo es el concepto que se tiene de <Cliente> en una organización grande como tenemos en el área de ventas el cliente son las personas que compran los productos, para el área de sistemas viene ser el área a quien se está dirigiendo el sistema que se está desarrollando esto se mantiene modelado en una integración continua que es cuando se haga un cambio se mantenga la lógica y esta no se pierda. Todo se relaciona con un **Context Map** que es mapear todo lo que se ha hecho pero

en código, es decir, todo lo que hemos estado modelando ponerlo en código. y esto se Context Map se divide en 7 partes:
Big Ball Of Mud: Es donde el código está unido por todas las entidades entonces es separarlo en partes que sean entendibles.
Anticorruption Layer: es la interfaz en donde se va a manejar todo el repositorio
Separate Ways: darle alas a todos los equipos que están trabajando, es decir que se de la libertad de manejar el lenguaje siempre y cuando se mantenga la lógica y estructura
Open Host Service: Es abrir un servicios para que otra entidad pueda sacar la información que necesite que se formaliza en Published Language que es en el lenguaje que esté publicado el código.
Shared Kernel: Son todas las cosas que podemos compartir, como sería en una bases de datos se podría compartir los identificadores como sería el usuario.
Customer/Supplier Teams: Un equipo tercero que se tenga que adaptar al contexto que se está manejando dentro del grupo principal.
Conformist: Donde serían dos equipos que se conforman con lo que se compartan entre ambos.