DOMAIN-DRIVEN DESIGN

POR

ASTRID CAROLINA GOMEZ

LUIS CARLOS MARÍN CAMPOS

ANDRES DARIO HIGUITA PEREZ

MARIO DE JESUS PUENTES SANCHEZ

THE DIVINES

PRESENTADO A:

ROBINSON CORONADO GARCIA

ARQUITECTURA DE SOFTWARE

INGENIERÍA DE SISTEMAS

UNIVERSIDAD DE ANTIOQUIA

2020 - 1

Domain-driven design

A la hora de construir software son muchas las técnicas, paradigmas, patrones arquitectónicos, modelos y tecnologías que existen y que se han desarrollado para brindarnos múltiples herramientas a los desarrolladores.
Una de estas múltiples herramientas es Domain-Driven-Design, que es el conjunto de prácticas y principios que ayudan a crear software a partir de una mejor interpretación de los términos de negocios, y está mejor interpretación del dominio ayuda a mejorar los procesos que tienen que ver con la confección del código. En la búsqueda de comprender este diseño no podemos confundirlo como una tecnología o metodología, ya que es una técnica que está estructurada con varias prácticas con las cuales nos podríamos ayudar a tomar decisiones de diseño con el fin de darle manejo a dominios complejos durante el desarrollo de proyectos ágiles.

Se hace necesario que los equipos de desarrollo tengan claro cuál es el dominio del negocio, es decir, el problema en específico que se está tratando de resolver, para no incurrir en soluciones no óptimas y costos extras, que perjudican a cualquier proyecto de software. Se trata de respetar el dominio lo máximo posible ya que el DDD representa las distintas claves, la terminología y los patrones que se utilizan para el desarrollo del software donde el dominio es lo más central e importante de una determinada organización.
Normalmente, no debemos considerar el dominio de una forma común o universal, sino que depende del contexto. El modelo del dominio para una empresa específica será distinto del de otra empresa. Obviamente habrá muchísimos elementos comunes, pero también existirán suficientes diferencias como para que no sean intercambiables Esto no impide que se pueda utilizar el paradigma del Domain Driven Design en desarrollar productos genéricos, pero en ese caso hay que pensar no tanto en el dominio genérico, sino en el de las soluciones de software para ese dominio.

Ahora bien, la pregunta que surge es ¿porque necesitamos un diseño basado en dominios? primero debemos tener presente que la comunicación en todo desarrollo de software es fundamental para que el equipo hable un mismo lenguaje, y cuando el equipo comprende el problema comercial en términos comerciales, esto les permite a los desarrolladores comunicarse más claramente con las partes interesadas comerciales y entre el equipo técnico.
DDD hace que el software sea más cercano al dominio, y por lo tanto es más cercano al cliente. y permite el testing a distintas partes del dominio de manera aislada.

Este diseño principalmente se basa en:
Colocar los modelos y reglas de negocio de la organización, en el punto central 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 los desarrolladores, además para concebir un software con los objetivos bien claros.

Ventajas de usar DDD:

- Permite la comunicación efectiva entre expertos del dominio y expertos técnicos a través de lenguaje ubicuo.
- Permite enfocarse en el desarrollo de un área dividida del dominio a través del límite del contexto.
- Hace que el software sea más cercano al dominio, y por lo tanto sea más cercano al cliente.
- Permite que el código se encuentre bien organizado, permitiendo el testeo de las distintas partes del dominio de manera aislada.
- Permite que la lógica de negocio resida en un solo lugar, y se encuentre dividida por contextos.
- Permite una mantenibilidad a largo plazo.

Desventajas

- Utilizar este diseño conlleva a que se presente un aislamiento de la lógica de negocio con un experto de dominio y el equipo de desarrollo suele llevar mucho esfuerzo a nivel tiempo.
- Para utilizarlo es necesario contar con un experto de dominio.
- Al implementarlo se presenta una curva de aprendizaje alta, con patrones, procedimientos, etc.
- Este enfoque de diseño que se sugiere utilizar en aplicaciones donde el dominio sea complejo, no es recomendado para unos simples CRUD’s que se puedan implementar.

Al trabajar con el Diseño orientado a el dominio es necesario tener presente la definición de algunos términos como lo son:

Dominio: El cual es el problema específico que estamos intentando resolver del mundo real. Representa la terminología y los conceptos clave del dominio del problema. Identifica las relaciones entre las entidades incluidas dentro del ámbito del dominio del problema, identifica sus atributos y proporciona una visión estructural del dominio.

Core Domain: Este es el diferenciador clave para el negocio del cliente, siendo algo que deben hacer bien y no se puede hacer una subcontratación.

Subdominio: Es la cual hace la segregación de un dominio más general en uno con forma más acotada, con más cohesión y más comprensible.

Contexto: Es el escenario en el cuak aparece una palabra o declaración que determina su significado.

Modelo: Este es unn sistema de abstracciones que describe aspectos seleccionados de un dominio y se puede usar para resolver problemas relacionados con ese dominio.

Lenguaje ubicuo (común): Este es un lenguaje estructurado en torno al modelo de dominio y utilizado por todos los miembros del equipo para conectar todas las actividades del equipo con el software. La comunicación efectiva entre los desarrolladores y los expertos del dominio es esencial para el proyecto.

Entidades: Las entidades son objetos del modelo que se caracterizan por tener identidad en el sistema, los atributos que contienen no son su principal característica. Deben poder ser distinguidas de otros objetos, aunque tengan los mismos atributos. Tienen que poder ser consideradas iguales a otros objetos aun cuando sus atributos difieren.

Value objects: Estos representan conceptos que no tienen identidad. Simplemente describen características. Por lo tanto, solo nos interesan sus atributos. Los value object representan elementos del modelo que se describen por el que son.

Services: Estos son los servicios que representan operaciones, acciones o actividades que no pertenecen conceptualmente a ningún objeto de dominio concreto. Los servicios no tienen ni estado propio ni un significado más allá que la acción que los definen, tienden a ser nombrados como verbos.

Arquitectura por capas: Es necesario dividir el sistema en al menos cuatro capas: presentación, aplicación, dominio e infraestructura.

Repositorio: Los métodos para recuperar objetos de dominio deben delegarse en un objeto Repository especializado de modo que las implementaciones de almacenamiento alternativas se puedan intercambiar fácilmente.

Factory: Los métodos para crear objetos de dominio deben delegarse en un objeto Factory especializado, de modo que las implementaciones alternativas se puedan intercambiar fácilmente.

Aggregate: Una colección de objetos que están unidos por una entidad raíz, también conocida como raíz agregada. La raíz agregada garantiza la coherencia de los cambios que se realizan dentro del agregado al prohibir que los objetos externos tengan referencias a sus miembros.

En conclusión, el diseño orientado al dominio es una técnica que está estructurada por varias prácticas que pueden ayudar a tomar las decisiones de diseño con el fin de enfocar y acelerar el manejo de dominios complejos durante el desarrollo del software, en donde este debe ser iterativo y debe existir una estrecha relación entre los desarrolladores y los expertos del dominio.

Esta técnica fue ideada para el desarrollo de aplicaciones complejas y está orientada a proyectos que usen metodologías ágiles.