Diseño Dirigido por el Dominio (Domain Driven Design - DDD)

Sebastián Bedoya Restrepo

Jose Andrés López Castaño

Andrés Eduardo Rossi Herreno

Luis Felipe Salazar Ríos

Arquitectura de software

Ingeniería de Sistemas

Universidad de Antioquia

Ude@

Hay mucha información sobre Diseño Dirigido por el Dominio en el cual se explica la importancia y relevancia que toma este tipo de actitudes y comportamientos por parte de los equipos involucrados en un proyecto de software a la hora de desarrollar una arquitectura exitosa y sostenible en el tiempo; pero ¿cómo se podría definir DDD de manera sencilla?

DDD es una manera para aproximarse al diseño del software apoyado en el conocimiento y pericia de los expertos en el dominio de un negocio o actividad determinada que, aunque no posean el conocimiento del lenguaje en que se construye un software, nos dan la materia prima para un buen desarrollo del mismo; podemos sustraer la mayor cantidad de información posible de las actividades y dinámicas del negocio aprovechando la experiencia para evitar fallos u errores de funcionalidad desde la arquitectura, este accionar permite profundizar en los procesos, crear todas las conexiones, actividades y comportamientos de los mismos dentro de la dinámica de una actividad para tener todos los parámetros y puntos de vista posible en la construcción del proyecto.

Lo que se busca a través del DDD es que el software desde su construcción posea diversas características que le permitan desarrollar una vida útil sostenible y exitosa; para ello debe cumplir con varios requisitos funcionales y no funcionales como son disponibilidad, continuidad, eficiencia, escalabilidad, interoperabilidad, mantenibilidad, portabilidad, estabilidad, adaptabilidad, reusabilidad y ser verificable entre otros. Para ello, se lleva a cabo una estrategia de diseño en la cual se debe separar durante el análisis los espacios del problema y la solución para tener un mejor entendimiento del problema que permita aportar el mejor diseño para la solución del mismo y que permita verificar el funcionamiento de la solución mediante el solapamiento de la información adquirida del proceso; por esta razón se busca que los expertos de negocio y los técnicos trabajen de manera concatenada para entender el dominio que es el problema manteniendo la separación de este de los aspectos tecnológicos, permitiendo tener una mejor comprensión de su dinámica y posibilitando dividir el dominio en subdominios que se pueden trabajar de manera individual para la asignación de cargas.

En términos de DDD el problema del negocio es el dominio para el cual se está estructurando el proyecto y diseñando la solución, por ello se recomienda llevar a cabo actividades que sean dinámicas e inclusivas cómo Event Stroming que permitan que se hable de manera amplia, exclusiva y extendida sobre el problema sin tener en cuenta la tecnología y de esta manera abarcar todos los procesos de la empresa e ir individualizando los subdominios.

Desde que Eric Evans presentó esta idea en su libro "Domain-Driven Design — Tackling Complexity in the Heart of Software" acuño el término lenguaje ubicuo para referirse a la creación de un lenguaje participativo entre los expertos y los técnicos a la hora de parametrizar el dominio y que permita que se lleve a cabo una dinámica comunicativa eficiente entre los actores que aunque están hablando de un mismo problema, normalmente lo hacen en idiomas diferentes y eso representa la principal

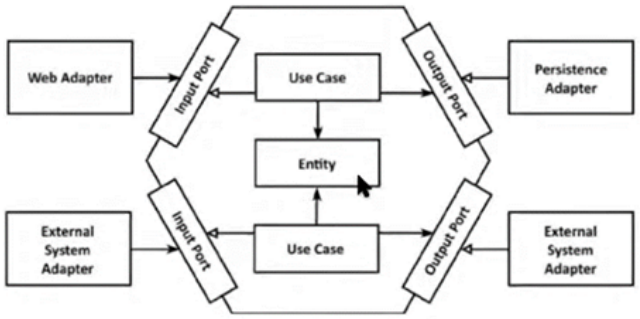
dificultad para el proyecto en el momento de establecer el levantamiento de requisitos. Por esta razón se recomienda desarrollar distintos lenguajes ubicuos o ir evolucionándolos conforme se avanza en la individualización de los subdominios y se va adquiriendo una mejor percepción y conocimiento del modelo.

Una práctica recomendada es dividir el dominio en subdominios mediante "Bounded context" que es delimitar o acotar los subdominios que compartan un mismo contexto, esto mejora la especialización de los equipos en el momento de distribuir cargas de trabajo y crea un ambiente propicio para el uso de herramientas tecnológicas comunes; la división del problema de esta manera lleva a una solución con mayor cohesión interna y reduce la dependencia de los equipos de agentes externos al mismo y permite crear desacoplamiento fuerte entre los subdominios a la hora de implementar los desarrollos con tecnologías distintas.

Los subdominios no están aislados, tienen dependencias entre ellos y esto impacta en la capacidad de evolución de cada uno, por eso se debe crear un diagrama que permita especificar las relaciones que existen entre los subdominios con el fin de mantener claridad y transparencia en los procesos de trabajo e instancias colaborativas.

Existen dos arquitecturas bastante conocidas y que se fundamentan en los principios que establece el DDD las cuales son la Hexagonal y capa de cebollas en las cuales se busca que todo parta desde el conocimiento del dominio y se vaya estableciendo el desarrollo conforme se avanza en la parametrización e individualización de los subdominios de manera que se pueden utilizar diferentes tecnologías que convergen a través de la funcionalidad de los servicios y microservicios.

Arquitectura Hexagonal



Arquitectura en capas de cebolla.