ARQUITECTURA DE SOFTWARE

ENSAYO DDD

Andrés Carrascal, Luis Carlos Rendón, Wulfram Polo
Facultad de Ingeniería – Ingeniería de Sistemas
Universidad de Antioquia

2020

DOMAIN DRIVEN DESIGN (DDD)

Dentro del mundo de arquitectura de software, la arquitectura de microservicios busca maximizar la forma en que se adaptan las soluciones mediante la distribución de las responsabilidades dentro de un software en servicios con ciclos de vida independiente, poder lograr esta independencia, seria clave para obtener una ventaja en la arquitectura, logrando un dominio funcional que se puede hacer mediante el Domain-Driven-Design (DDD) definido por Eric Evans en su libro publicado en 2004.

En el contexto de la creación de aplicaciones, DDD hace referencia a los problemas como dominios. Describe áreas con problemas independientes como contextos delimitados correlacionados con un microservicio y resalta un lenguaje común para hablar de dichos problemas. También sugiere muchos patrones y conceptos técnicos, como entidades de dominio con reglas de modelos enriquecidos.

Inicialmente, tenemos que recalcar que el DDD no es una tecnología ni metodología, más bien, es una técnica que nos puede ayudar a la toma de decisiones en cuanto al diseño más adecuado a enfocar en una aplicación, esto dependiendo el proyecto que vayamos a realizar, para tener una mejor claridad de esto, vamos primero a definir por partes lo que esta técnica hace referencia, ¿Qué es el dominio?, no es más que el campo donde el usuario hará aplicación del software que quiere realizar, es decir aquello de lo que trata el negocio y, por tanto, de lo que tratará la aplicación, El objetivo del DDD es representar el dominio en un modelo que luego se implementará en forma de software.
Otra definición, que debemos tener en cuenta es la de Modelo, el cual es un conjunto de los conocimientos que el equipo tiene sobre el dominio, la idea es que el conocimiento se vaya refinando y adaptando constantemente a medida que se aprenda sobre el dominio. El objetivo es ahorrar energía en soluciones técnicas y sobrecargo de los desarrolladores a aprender los principios del negocio. El DDD nos ayuda a tener un diseño más cercano al lenguaje natural que al lenguaje técnico, pero evidentemente, para que esto tenga sentido, debe complementarse con la adecuada decisión de patrones de diseño, inversión de dependencia, encapsulación y ACL’s, lo cual tiene múltiples ventajas en el contexto del software, pero nos puede aportar valor en la forma en la que se administran nuestras empresas. Conforme estas crecen, se empiezan a superponer procesos y dependencias entre los mismos.

¿Cómo se conforman los Dominios? Un dominio, está dividido en subdominios más pequeños conocidos como Bounded Contexts los cuales son un espacio lógico en el que compartimos un lenguaje ubicuo. Es decir, en este espacio lógico, todos utilizamos las mismas palabras para referirnos a los mismos elementos en donde a cada uno de los servicios no le interesa cómo haga el trabajo el otro contexto, sino las entradas y salidas; de esta manera, lo más importante es que cada uno de los contextos se pueda comunicar. La situación ideal sería cuando un Bounded Context cubre un único sub-dominio, lo que equivale a **"**para cada problema una solución**".** Sin embargo, esto suele no cumplirse cuando los negocios empiezan a agregar complejidades pequeñas a los sub-dominios y estos relacionan recursos de otro sub-dominio.

DDD se divide en dos grandes partes: estratégica y táctica, cuyo orden de implementación es definido de acuerdo al estado de avance del proyecto que se va a trabajar con este enfoque. Para un proyecto que esté en etapa de definición y análisis, el punto de partida va a ser la parte estratégica, donde se busca un modelo que nos ayude a entender y resolver las necesidades o problemas que se tienen en un dominio de negocio; seguidamente se inicia el proceso táctico donde se eligen los patrones de arquitectura a utilizar para diseñar la solución**,** al igual que las tecnologías a usar en la solución de cada sub-dominio modelado.

En forma de conclusión, podemos decir que el modelo juega un papel central para asegurar la correspondencia entre el software y el negocio al que debe representar. Las ventajas que nos va a ofrecer es que su aplicación a proyectos de larga vida y cambios constantes, nos va a ofrecer un modelado constante, iterativo, justificado, consistente y robusto. Además, nos da la oportunidad de que todos los miembros de un proyecto estén alineados, hablando de los mismos conceptos