ENSAYO DOMAIN DRIVEN DESIGN

Paola Andrea Posada Restrepo
Arquitectura de Software

Para entender de manera más sencilla que es DDD deberíamos iniciar con un ejemplo que encontré en la web, donde se supone que se va a realizar un software para una clínica veterinaria y al momento de modelar el dominio del negocio surgen preguntas como ¿un cliente como una persona, dueño de un animal? o ¿un cliente como un animal, que tiene un dueño persona?, otros interrogantes que pueden surgir entorno a este ejemplo, como la definición, la similitud o la diferencia entre los términos paciente, animal, mascota, cliente, dueño y el papel que juegan dentro del negocio.

A primera vista estos planteamientos no serían de gran importancia para el desarrollo y cualquier decisión sobre estos no influirá en el producto final; sin embargo cada uno de estos planteamientos son de vital importancia y marcan el éxito o el fracaso del software. Es decir, suponga que las decisiones las tomo el equipo de desarrollo y el cliente será la mascota (para ellos es quien hace uso del servicio veterinario) sin embargo para el negocio, el cliente es el dueño de la mascota (ya que es quien paga por los servicios prestados). Lo que causa que posiblemente el cliente no acepte la aplicación o se tengan que hacer cambios y eso se traduce en tiempo y recursos perdidos, lo que se había podido evitar con una correcta comunicación entre las partes involucradas.

Es por lo anterior que surge DDD (Domain Driven Design/ Diseño guiado por dominios) el cual se puede definir como una técnica (ideada para el desarrollo de aplicaciones complejas y está orientada a proyectos que usen metodologías ágiles) que podrá ayudar a tomar decisiones de diseño desde la parte del dominio, es decir DDD busca concebir un software con objetivos claros, implementando la comunicación y la colaboración entre los expertos en el dominio (son personas que conocen el negocio perfectamente, no tienen que tener conocimientos de software) y los desarrolladores; ya que al tener unos objetivos bien definidos y aceptados por ambas partes se puede comenzar a materializar la idea y que esta tenga buenos resultados. Cabe aclarar que esta técnica (DDD) no define implementaciones, define conceptos y reglas de implementación.

Ahora ya conocemos un poco lo que es DDD debemos definir algunos de sus elementos para entender más a fondo cómo se trabaja con dicha técnica.

Aunque ya se ha hablado del dominio considero necesario definirlo de manera más explícita, el dominio se puede ver cómo el negocio, las reglas que este tiene y como trabaja. Para entenderlo mejor se puede dar un pequeño ejemplo, para una aplicación bancaria quien conoce más del dominio de esta

es un banquero o el gerente del banco, es una o varias personas que saben a la perfección cómo funciona el negocio y sus conocimientos son indispensables para llevar a buen término un proyecto. Toda esta información será el centro del software y se aloja en la capa del dominio.

Otro concepto es Ubiquitous Language (Lenguaje Ubicuo), es un término que trae Eric Evans en su libro sobre DDD como propuesta para crear un lenguaje común entre los programadores y los expertos del dominio, ya que como se dijo al principio entablar una correcta comunicación entre estas partes es esencial para el éxito de un proyecto.

DDD hace referencia a los siguientes elementos que se abordarán de forma superficial.
Entidades: son objetos del modelo que se caracterizan por tener identidad
Objetos de valor: representan conceptos que no tienen identidad y se encargan de describir características.
Servicios: representan operaciones, acciones o actividades que no pertenecen conceptualmente a ningún objeto de dominio concreto. Es decir, son comportamientos que no le pertenecen a ninguna entidad porque simplemente son utilizados por ellas.
Módulos: son "espacios" separados que sirven para organizar el código, como un contenedor para un conjunto de clases específicas de la aplicación.
Agregados: son grupos de entidades relacionadas entre sí son dependientes entre ellas a nivel de negocio. Se debe definir una entidad raíz y se dará acceso público únicamente a esta.
Factorías: Se usan cuando la creación de una entidad o un agregado se convierte en un proceso complejo, por lo tanto se debe delegar esa responsabilidad en una Factoría. Las factorías son clases encargadas de crear entidades o agregados, construyendo todas las entidades contenidas en ellos.
Repositorios: actúa como la capa de persistencia de la aplicación, además los repositorios tendrían que tener las operaciones básicas de Create, Retrieve, Update, Delete.

Finalmente, es conveniente definir las capas de la arquitectura, DDD propone cuatro capas, sin embargo estas y sus componentes pueden variar dependiendo de las necesidades del software. Interface de usuario que se encarga presentar la información al usuario, interpretar sus acciones y enviarlas a la aplicación. Aplicación es la responsable de coordinar todos los elementos de la aplicación y su flujo. Dominio es el centro de la aplicación, contiene las reglas de negocio. Infraestructura esta capa es la capa de soporte para el resto de capas. Provee la comunicación entre capas, implementa la persistencia de los objetos de negocio y las librerías de soporte.

DDD se centra en el lenguaje, en un lenguaje natural para que ambas partes puedan entender, comprender y conciliar los objetivos, lógica y reglas del negocio; ya que con estos factores completamente claros y sin ninguna ambigüedad el desarrollo tiene una dirección, la cual es el negocio y su dominio. Ahora que se tiene un modelo bien definido y aceptado por ambas partes, el desarrollo se enfocará completamente a este.

Cabe aclarar que estas prácticas además de estar enfocadas al dominio tiene como objetivo mantener el principio de responsabilidad única, codigo desacoplado, principio de cohesión, integridad de la información, encapsulamiento, principio SOLID abierto a extensión cerrado a modificación, permitir la reusabilidad y la testabilidad; además de muchas otras técnicas, principios y patrones.

Para finalizar, DDD fue pensado y se recomienda su uso en desarrollos empresariales complejos, aplicaciones que generalmente tienen una tiempo de vida largo, que esté en constante evolución y que esta sea en gran volumen. Sin embargo, también tiene como objetivo los sistemas distribuidos (ya que estos se centran principalmente en el dominio) y microservicios.

Referencias

- Loscalzo, J. (2018, June 18). Domain Driven Design: Principios, beneficios y elementos - Primera Parte. Recuperado de https://medium.com/@jonathanloscalzo/domain-driven-design-principios-beneficios-y-elementos-primera-parte-aad90f30aa35

- Loscalzo, J. (2018, June 18). Domain Driven Design: Principios, beneficios y elementos - Segunda Parte. Recuperado de https://medium.com/@jonathanloscalzo/domain-driven-design-principios-beneficios-y-elementos-segunda-parte-337d77dc8566

- Tubio, J. (2019). Resumen sobre DDD. Domain Driven Design. Recuperado de https://github.com/jatubio/5minutos_laravel/wiki/Resumen-sobre-DDD.-Domain-Driven-Design