Ensayo (Domain Driven Design)

Por:

Anderson Barrientos Parra
C.C: 1017242181
Stiven Guerra Chaverra
C.C: 1037672655
Sebastián Gómez Ramírez
C.C: 1045026756
Hermen Esteban Theran Martínez
C.C: 1035875072

Robinson Coronado García
Profesor

UNIVERSIDAD
DE ANTIOQUIA
1 8 0 3

Arquitectura de Software - Ingeniería de Sistemas
Universidad De Antioquia

Medellín 2020

Introducción

En el presente documento se busca dar a conocer, explicar y representar el enfoque de desarrollo de software conocido como Domain Driven Design (Desde ahora DDD para abreviar), utilizado por Eric Evans en su libro *"Domain-Driven Design - Tackling Complexity in the Heart of Software"*, en el 2004. El cual busca comprender una serie de técnicas, metodologías y patrones a través de los cuales el desarrollo de productos software debe ser una tarea sencilla si se conoce el campo con el que se trata.

Desarrollo

Traducido al español, el DDD significa *Diseño Dirigido por Dominios,* al interiorizar este nombre y con un poco de conocimientos en el área, se puede comprender un poco de lo que busca atender este enfoque de desarrollo, caso contrario, para definir el concepto se debe empezar por definir lo que es un *Dominio*.

Un Dominio se puede definir como la abstracción del área sobre la cual se está trabajando y todo lo relacionado a esta, desde las entidades presentes en el negocio hasta las interacciones que hay entre ellas y las tareas que desempeña cada una. Como ejemplo a la hora de trabajar en una aplicación de viajes en avión donde un usuario puede entrar, buscar vuelos y comprar tiquetes, el dominio se definiría como La Venta de Tiquetes de Avión. Pero al igual que objetos tangibles, el dominio puede pertenecer a un mundo intangible como un sistema de manejo y venta de videojuegos virtuales (valga el pleonasmo), donde un usuario posee una librería virtual en la cual puede añadir videojuegos que compra en línea y puede descargar en cualquier momento desde cualquier dispositivo.
Bajo la definición anterior, los dominios presentan diferentes elementos, los cuales son fáciles de identificar y categorizar en base a lo que representan o la tarea que cumplen:

- Objetos. Entidades que existen a lo largo del dominio, los cuales existen para representar elementos interactuantes en la aplicación. En la aplicación de tiquetes se tendrían "Vuelos", "Aviones", "Tiquetes". En la aplicación de videojuegos se tendrían "Videojuegos", "Librerías".

- Comportamientos. Interacciones que pueden existir entre los objetos mencionados anteriormente o acciones que cada objeto efectúa de manera independiente sobre el sistema. En la aplicación de tiquetes se tendrían "Comprar", "Reservar", "Buscar". En la aplicación de videojuegos se tendrían "Descargar", "Actualizar".

A la hora de desarrollar cualquier tipo de aplicación la comunicación juega un papel importante para la correcta abstracción del dominio, ya que aquí entra en juego el concepto de "experto temático", como la persona que tiene un total conocimiento de los detalles, aspectos y reglas que harán parte de la aplicación y/o proyecto que se desarrollará. Un ejemplo de esto podría ser un software de simulaciones astronómicas, donde los distintos miembros del equipo de trabajo como desarrolladores, analistas de software o arquitectos no tienen conocimiento sobre cómo funcionan estas simulaciones, sino el cliente (los

astrónomos) que son los que conocen a profundidad y detalle sus necesidades y pueden guiar el desarrollo del software en cuestión.

Para que esta comunicación se dé de manera efectiva, el *DDD* define el concepto de "lenguaje omnipresente", el cual consiste en un lenguaje y terminología únicos para todas las etapas del proyecto. ¿Por qué es necesario un lenguaje en común entre los desarrolladores y los expertos del dominio? Para dar respuesta a esta pregunta, se puede partir del hecho de que por lo general el desarrollador está acostumbrado a un lenguaje propio de su área o lenguaje técnico, pero este lenguaje muchas veces no es de entendimiento por los expertos temáticos y/o clientes, por lo que un lenguaje en común es de vital importancia para evitar malentendidos o ambigüedades que pueden presentarse si no hay una compatibilidad en el uso de términos. Un ejemplo sencillo de esto, es la forma en la que cada miembro del equipo se refiere a la persona que usará el software. Los desarrolladores lo llaman "usuario", y los analistas "cliente", esto genera ambigüedades para ambas partes, pues aunque se refieren al mismo objeto, puede entenderse como algo totalmente distinto.

Ventajas

- Posee una comunicación efectiva entre expertos del dominio y expertos técnicos a través de *lenguaje omnipresente*.
- El software es más cercano al dominio, y por lo tanto es más cercano al cliente.
- La lógica de negocio reside en un solo lugar, y dividida por contextos.
- Posee una mantenibilidad a largo plazo.
- Tiene un código bien organizado, permitiendo el testing de las distintas partes del dominio de manera aisladas.

Desventajas

- El proceso de aislamiento del dominio suele tomar mucho tiempo
- Posee una curva de aprendizaje alta
- Este enfoque sólo es sugerido para aplicaciones donde el dominio sea complejo.

Conclusión

El uso de este enfoque en el desarrollo de software, mejora a gran escala, el desarrollo de software, debido a su unificación del lenguaje y aislamiento de la lógica del negocio, aunque como todo enfoque arquitectónico, lo ideal es evaluar las necesidades del software para establecer esta.

Bibliografía

- Echeverría, A., López, G., Grossi, M., Servetto, A., Jeder, A. y Linares, P. (2010). “DDD (diseño dirigido por el dominio) y aplicaciones Enterprise: ¿fidelidad al modelo o a las herramientas? Recuperado de: http://sedici.unlp.edu.ar/bitstream/handle/10915/18328/Documento_completo.pdf?sequence=1&isAllowed=y

- Bass, L., Clements, P., & Kazman, R. (2012). *Software Architecture in Practice* (3rd Revised ed.) [Libro electrónico]. Addison-Wesley Professional. Recuperado de: http://jz81.github.io/course/sa/Software%20Architecture%20in%20Practice%20(3rd).pdf

- Guzmán, S. (2016). *Functional Domain Driven Design*. Recuperado de: https://repositorio.uam.es/bitstream/handle/10486/673434/Herrera_Guzman_Sergio_tfg.pdf?sequence=1&isAllowed=y