DDD - Domain Driver Design

Equipo de Trabajo

Berserkers

Docente
Robinson Coronado Garcia

Responsables
Alejandro Mesa Gómez
Jhon Janer Torres Restrepo
Hernán Javier Aguilar Cruz
Cristhian Javier González Rodríguez

Arquitectura de Software
Universidad de Antioquia
Medellín
2020

Índice

Acerca de DDD

El DDD o domain driver design es un enfoque para el desarrollo de software, definido así por Eric Evans en su libro *Domain driven design: Tackling Complexity in the Heart of Software* publicado en el 2004*,* el cual se enfoca en un modelo evolutivo para la resolución de problemas del dominio. El libro recopila una serie de patrones y recomendaciones para el diseño de un modelo de dominio que permita tratar la complejidad de las aplicaciones.

El DDD en su enfoque para el desarrollo de software propone una estructura en lugar de una tecnología o una metodología, con la cual se tomen decisiones respecto al diseño para acelerar el manejo de los dominios más complejos dentro de un proyecto. Por ejemplo, este modelo sería adecuado para la creación de un software que tiene complejidad en su proceso comercial. Así mismo, el DDD no es muy útil para aplicaciones CRUD ya que su considerable simplicidad no lo demanda.

Es importante que se lleve a cabo una colaboración armoniosa entre expertos técnicos y expertos en el dominio, para lograr mejor eficiencia y profundidad a la hora de analizar los dominios correspondientes. Una buena comunicación evita malos entendidos y da más enfoque al objetivo principal.

Aplicaciones

El uso de DDD no es dependiente de una herramienta o un software preciso. Sin embargo, hay un número cada vez más grande de aplicaciones en relación a los patrones presentados y defendidos en el libro de Evans. Entre ellos, se incluyen:

Actifsource: Complemento para Eclipse donde se incluye el desarrollo de software combinado DDD e ingeniería basada en modelos, así como generación de código. **Cubic Web:** es un web frame de código abierto donde se impulsa el modelado de datos. Se define un modelo donde es suficiente para obtener datos de una aplicación web que funcione. **OpenMDX:** Código abierto basado en Java, MDA framework soportado por Java, Java SE, Java EE y .NET, este se diferencia ya que utiliza modelos que ayudan a impulsar directamente el comportamiento en tiempo de ejecución de los sistemas operativos. **OpenXava:** Genera una aplicación AJAX con entidades JPA. Necesita escribir clases de dominio para obtener una aplicación lista.
Restful Objects: Estándar para una API Restful, en un modelo de objeto de dominio.

Aparte, el uso de DDD está relacionado con varias ideas como lo son: Análisis y diseño orientado a objetos, ingeniería basada en modelos (MDE), arquitectura basada en modelos (MDA), plain old java objects (conocidos también como POJOs) y plain old CLR objects (también conocidos como POCOs). El patrón de objetos desnudos (the naked objects pattern), modelado específico de dominio (DSM), lenguaje específico de dominio (DSL), programación orientada a objetos (AOP), segregación de responsabilidad de consultas de comando (CQRS), suministro de eventos (ES).

Ventajas y desventajas

En el diseño basado en dominios, todo el equipo utiliza un mismo lenguaje y comparte un modelo. Los desarrolladores se comunican mejor con el equipo empresarial y el trabajo es más eficiente cuando se trata de establecer soluciones para los modelos que reflejen cómo opera el negocio, en lugar de cómo opera el software. Además, resulta bastante sencillo realizar un seguimiento de la implementación de los requisitos, un código más legible y menos duplicación.

Al seguir un enfoque ágil que es iterativo e incremental, el DDD aclara el modelo mental de los expertos en el dominio en un modelo útil para el negocio, todos los equipos pueden comprender dónde son importantes determinadas integraciones y por qué, al final esto brinda una buena arquitectura de software. El DDD da la vuelta a los conceptos de diseño orientado a objetos, esto implica que casi todo en el modelo de dominio se basa en un objeto y, por lo tanto, será modular y encapsulado, lo que permitirá cambiar y mejorar el sistema de forma regular y continua.

Sin embargo, es habitual que surja el anti-patrón “Modelo del Dominio Anémico”, en inglés (Anemic Domain Model). Si esto sucede, es común encontrar objetos que llevan nombres sacados del dominio y forman una estructura que, a primera vista, parece un modelo del dominio pero la realidad es que estos objetos son solo un conjunto de datos sin comportamiento, implementados por la lógica en objetos servicio.

Conclusiones

Es evidente que en toda aplicación se presentan complejidades, sin importar que tan simple pueda parecer. El problema yace de que estas complejidades muchas veces no se encuentran en la parte técnica, sino en la lógica del negocio (dominio). Así, cuando intentamos resolver problemas de dominio con tecnología, se desata la incomprensión de su verdadero funcionamiento, aunque hayamos resuelto el problema.

Referencias

[1] Justdigital. (2018, Julio 18). ¿Qué aporta Domain-Driven Design al software? AGILE, AGILE METHODOLOGIES, DDD, DOMAIN-DRIVEN DESIGN, SOFTWARE DEVELOPMENT, Consultado el 16 de Diciemnbre 2020, en https://justdigital.agency/blog/domain-driven-design/

[2] M Yauri at-Tamimi (2017, Diciembre 14). DDD: Part I (Introduction) Learn about DDD- Domain Driven Design- which focuses on software development through collaboration between technical experts and domain experts. Consultado el 16 de Diciemnbre 2020, en https://dzone.com/articles/ddd-part-i-introduction

[3] Karam, L. (2020). An Introduction to Domain Driven Design and Its Benefits - DZone Agile. Consultado el 16 de Diciemnbre 2020, en https://dzone.com/articles/an-introduction-to-domain-driven-design-and-its-be