METODOLOGÍA DE SISTEMAS

PechaKucha 2

Profesor:

SANTORO, EXEQUIEL

CURSO: 2W2

GRUPO: 05

Integrantes:

BRACAMONTE, GASTÓN.....110139

CHACHAGUA, VANINA.....110316

FERRUFINO, MARIO GASTÓN.....110318

LAMBRECHT, FRANCO.....110276

SANCHEZ, MILENA.....110655

Domain Driven Design (DDD)

¿Qué es?

Se refiere al «diseño guiado por el dominio», conocido en inglés como “Domain-Driven Design”.

El **DDD** no es una tecnología ni una metodología, es una técnica que está estructurada por varias prácticas que ayudan a tomar decisiones de diseño con el fin de enfocar y acelerar el manejo de dominios complejos durante el desarrollo de proyectos digitales.

Representa distintas claves, terminología y patrones utilizados para desarrollar software donde el dominio es lo más central e importante que ayudan a tratar la complejidad de las aplicaciones.

¿Cómo nació?

Es una técnica que ha surgido y evolucionado en las últimas dos décadas, pero se ha cristalizado más claramente durante los últimos años: diseño orientado al dominio. Eric Evans ha hecho una gran contribución a este tema escribiendo en un libro gran parte del conocimiento acumulado sobre diseño de dominios.

Este enfoque para desarrollo de software definido por Eric Evans, en su libro Domain-Driven Design: Tackling Complexity in the Heart of Software, en el que expone un modelo rico, expresivo y en constante evolución que busca resolver problemas del dominio de una forma semántica.

¿Para qué sirve?

Fue ideada para el desarrollo de aplicaciones complejas y está orientada a proyectos que usen metodologías ágiles.

Centrado en el ¿Qué? Y no en el ¿Cómo?

Principios

Sus principios se basan en:

- Colocar los modelos y reglas de negocio de la organización, en el core de la aplicación
- Basar nuestro dominio complejo, en un modelo de software.
- Se utiliza para tener una mejor perspectiva a nivel de colaboración entre expertos del dominio y los desarrolladores, para concebir un software con los objetivos bien claros.

Beneficios

- Comunicación efectiva entre expertos del dominio y expertos técnicos a través de Ubiquitous Languge.
- Foco en el desarrollo de un área dividida del dominio (subdominio) a través de Bounded Context's.
- El software es más cercano al dominio, y por lo tanto es más cercano al cliente.
- Código bien organizado, permitiendo el testing de las distintas partes del dominio de manera aisladas.
- Lógica de negocio reside en un solo lugar, y dividida por contextos.
- Mantenibilidad a largo plazo.

Inconvenientes

- Aislar la lógica de negocio con un experto de dominio y el equipo de desarrollo suele llevar mucho esfuerzo a nivel tiempo.
- Necesitamos un experto de dominio —¿ DDD sin Domain-Experts?
- Una curva de aprendizaje alta, con patrones, procedimientos,…
- Este enfoque solo es sugerido para aplicaciones donde el dominio sea complejo, no es recomendado para simples CRUD's.

Dominio

- Es el problema específico que estamos intentando resolver del mundo real.
- Nosotros tenemos que abstraernos de este dominio, generando una abstracción a través de un modelo.
- Representa todo el conocimiento de un experto de negocio; pero este conocimiento es difícil de plasmar en un desarrollo de software.
- Un dominio "real" tiene demasiada información que plasmar y es tarea en conjunto la decisión de las abstracciones a lograr.
- El primer y principal requerimiento de un modelo de dominio es ser consistente y sin contradicciones.
- El dominio se va a convertir en una capa en nuestra arquitectura, es un lugar central donde nada externo tiene que influir, complicar o distraer la idea principal: el dominio como el corazón de nuestro software.
- La capa de dominio es la responsable de representar la información del negocio, la lógica del negocio, situaciones del negocio, a pesar de las dificultades que pueden conllevar la infraestructura. Es decir, nos abstraemos de nuestro detalles, que pueden ser que base de datos vamos a utilizar, como vamos a persistir, que framework frontend vamos a utilizar.

Ubiquitous Language

La comunicación efectiva entre los desarrolladores y los expertos del dominio es esencial para el proyecto. Un lenguaje común, que sea representado en el dominio es muy importante, para evitar tener problemas futuros y desarrollar un software exitoso, donde la comunicación sea el pilar para su obtención.

A project faces serious problems when it’s languages is fractured — Eric evans

Elementos del dominio

- **Entities**: Las entidades son objetos que tienen identidad, normalmente un GUID, UUID o ID. Esto significa que aunque dos objetos de la misma clase, tengan los mismo valores de atributos, no son iguales, sino que la identidad* es quien los identifica como el mismo objeto.
- **Value Objects**: Existen casos en el que un objeto en particular no se comporta como una entidad, no todos los objetos deben ser identificables. En estos casos, las entidades necesitan objetos que tengan un comportamiento bien definido en el que se describe cierto aspecto del dominio. Son llamados Value Objects.
- **Services**: Existe lógica o reglas de negocios que no pertenecen particularmente a un objeto del dominio y no tiene sentido que se encuentre en alguno de ellos. Este comportamiento representa una parte importante del dominio, pero no encaja en ningún value object o entidad. Para estos casos, seguramente necesitemos agrupar esta lógica en objetos particulares, los llamados Servicios de Dominio.
- **Modules**: Los módulos son “espacios” separados que sirven para organizar el código: sirve como un contenedor para un conjunto de clases específicas de la aplicación.
- **Aggregates**: Una agregación es un cluster de objetos asociados que pueden ser tratados como una sola unidad con el propósito del cambio de datos
- **Factories**: Puede ocurrir que la construcción de Entidades y Agregaciones, internamente sean estructuras complejas, y conocer estructuras internas para su fabricación sería violar el principio de encapsulamiento. Por ende, vamos a delegar la fabricación en un objeto Factory.
- **Repositories**: Toda la lógica, debería estar encapsulada en los Repositorios.

Bibliografía:

- https://www.infoq.com/minibooks/domain-driven-design-quickly/
- https://justdigital.agency/que-es-domain-driven-design-ddd/
- https://medium.com/@jonathanloscalzo/domain-driven-design-principios-beneficios-y-elementos-primera-parte-aad90f30aa35
- https://medium.com/@mattcarroll/book-review-domain-driven-design-42c96a75a72
- https://medium.com/@jonathanloscalzo/domain-driven-design-principios-beneficios-y-elementos-segunda-parte-337d77dc8566