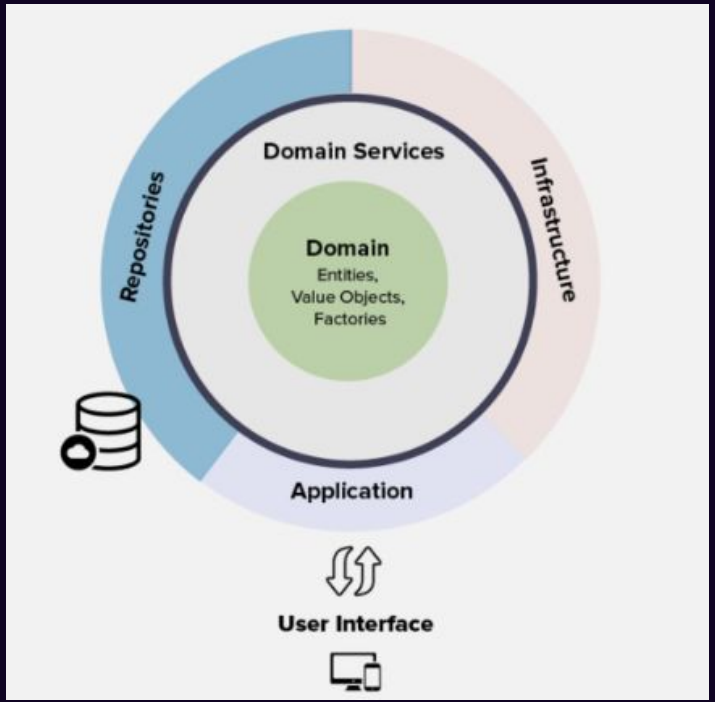
Domain Driven Design

Doctrina del desarrollador

Indice

¿Que es?

El diseño basado en dominios es un importante enfoque de diseño de software, que se centra en modelar software para que coincida con un dominio de acuerdo con las aportaciones de los expertos de ese dominio. Según el diseño basado en dominios, la estructura y el lenguaje del código de software deben coincidir con el dominio empresarial



3 principios fundamentales

3 principios fundamentales

El enfoque principal del proyecto es el dominio central y la lógica del dominio

Los diseños complejos se basan en modelos del dominio

La colaboración entre expertos técnicos y de dominio es crucial para crear un modelo de aplicación que resuelva problemas de dominio particulares

Conceptos importantes

Conceptos importantes

- **Lógica de dominio**
  La lógica de dominio es el propósito de su modelado. Más comúnmente, se la conoce como lógica de negocios. Aquí es donde sus reglas de negocio definen la forma en que se crean, almacenan y modifican los datos

- **Modelo de dominio**
  El modelo de dominio incluye las ideas, el conocimiento, los datos, las métricas y los objetivos que giran en torno al problema que estás intentando resolver. Contiene todas las reglas y patrones que le ayudarán a lidiar con una lógica empresarial compleja. Además, serán útiles para cumplir con los requisitos de su negocio

- **Servicio de dominio**
  Es una capa adicional que también contiene lógica de dominio. Es parte del modelo de dominio, al igual que las entidades y los objetos de valor

- **Subdominio**
  Un dominio consta de varios subdominios que hacen referencia a diferentes partes de la lógica empresarial. Por ejemplo, una tienda minorista en línea podría tener un catálogo de productos, un inventario y una entrega como subdominios

- **Patrones de diseño**
  Tienen que ver con la reutilización de código. No importa la complejidad del problema que encuentre, alguien que haya estado haciendo programación orientada a objetos probablemente ya haya creado un patrón que lo ayudará a resolverlo

- **Servicio de aplicaciones**
  Es otra capa que no contiene lógica empresarial. Sin embargo, está aquí para coordinar la actividad de la aplicación, ubicada encima del modelo de dominio

Conceptos importantes

- **Contexto**
  El contexto acotado es un patrón central en el diseño basado en dominios que contiene la complejidad de la aplicación. Maneja grandes modelos y equipos. Aquí es donde implementas el código, después de haber definido el dominio y los subdominios. Los contextos acotados en realidad representan límites en los que se define y es aplicable un determinado subdominio. Aquí, el subdominio específico tiene sentido, mientras que otros no. Una entidad puede tener diferentes nombres en diferentes contextos. Cuando cambia un subdominio dentro del contexto acotado, no es necesario que todo el sistema cambie también

- **Repositorio**
  Es una colección de entidades comerciales que simplifica la infraestructura de datos. Libera el modelo de dominio de preocupaciones de infraestructura. El concepto de estratificación impone la separación de preocupaciones entre capas

- **El lenguaje omnipresente**
  Es una metodología que se refiere al mismo lenguaje que utilizan los expertos y desarrolladores cuando hablan del dominio en el que están trabajando

- **Entidades**
  Las entidades son una combinación de datos y comportamiento, como un usuario o un producto. Tienen identidad, pero representan puntos de datos con comportamiento

- **Objetos de valor y agregados**
  Es un objeto que no tiene identidad y está definido por sus atributos. Por ejemplo, una cantidad de dinero, una fecha o una dirección son objetos de valor

Ventajas

Ventajas

El dominio es más importante que UI/UX
Como el dominio es el concepto central, los desarrolladores crearán aplicaciones adecuadas para el dominio en particular. Esta no será otra aplicación centrada en la interfaz. Aunque no debes dejar de lado la UX, usar el enfoque DDD significa que el producto se dirige exactamente a los usuarios que están directamente conectados al dominio

Comunicación más sencilla
Gracias al Ubiquitous Language, la comunicación entre desarrolladores y equipos se vuelve mucho más fácil. Como es probable que el lenguaje omnipresente contenga términos más simples a los que se refieren los desarrolladores, no hay necesidad de términos técnicos complicados

Más flexibilidad
Como DDD está orientado a objetos, todo lo relacionado con el dominio se basa en un objeto y es modular y enjaulado. Gracias a esto, todo el sistema se puede modificar y mejorar periódicamente

Desventajas

Desventajas

Contiene prácticas repetitivas
Aunque muchos dirían que esto es una ventaja, el diseño basado en dominios contiene muchas prácticas repetitivas. Se fomenta el uso de la integración continua para crear aplicaciones sólidas que puedan adaptarse cuando sea necesario. Muchas organizaciones pueden tener dificultades con estos métodos. Más particularmente, si su experiencia previa está generalmente ligada a modelos de crecimiento menos flexibles, como el modelo en cascada

Puede que no funcione para proyectos altamente técnicos
Es perfecto para aplicaciones que tienen una lógica empresarial compleja. Sin embargo, puede que no sea la mejor solución para aplicaciones con una complejidad de dominio menor pero una complejidad técnica alta. Las aplicaciones con gran complejidad técnica pueden resultar muy desafiantes para los expertos en dominios orientados a los negocios

Se necesita un conocimiento profundo del dominio
Tiene que haber al menos un especialista en el dominio en el equipo que comprenda las características precisas del área temática que es el centro de la aplicación. A veces es necesario incorporar al equipo de desarrollo varios miembros del equipo que conozcan a fondo el dominio

Ejemplo de código

Ejemplo de código

Github

En el siguiente link encontraran un ejemplo creado en typescript con la implementación de domain-driven-design
https://github.com/julianmerlo95/domain-driven-design-example

Referencias

- **Medium:** https://medium.com/microtica/the-concept-of-domain-driven-design-explained-3184c0fd7c3f
- **Atlassian:** https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment