Domain driven design

Es un enfoque de desarrollo que le permite a un equipo de trabajo alcanzar los objetivos planteados de una forma más fácil, a partir de un modelado basado en el dominio del negocio a tratar.
No es un patrón ni un marco de trabajo como tal, pero sí describe un par de fases que **pueden** trazar la ruta a seguir en todo el desarrollo.

La primera de estas fases, es la fase estratégica.
En esta fase se plantea y se decide el diseño del modelo, que muchas veces se describe en servicios, entidades y las clases en general,y todas estas siempre manejadas desde una encapsulación accediendo a ellas por medio de repositorios, para evitar así el acoplamiento.
A partir del modelo, y de la lógica en general, se generan eventos de los cuales se va a tener un seguimiento, escuchándolos y publicando su estado actual; por ejemplo: “El video se ha subido con éxito”. Además de estos, existen otros conceptos como *factorys* y otros patrones de diseño, especialmente de creación que ayudan a mantener limpia y agnóstica la forma de trabajar con los elementos ya mencionados.
Todo esto, se puede globalizar como la arquitectura por capas, o como muchas veces se ha definido en los textos, como un arquitectura hexagonal.

A partir de acá, y de tener un modelo y una arquitectura bien definida basada en el dominio, se puede empezar a mudar a la siguiente fase que describe el *Domain driven design,* la cual es la fase técnica.
Esta transición bien hecha, es lo que le da el verdadero potencial a este enfoque de desarrollo, y es que, poder **implementar** en el código los conceptos del modelo y núcleo del negocio de una forma coherente, le da un verdadero valor al flujo de todo el proceso que llevó en la fase estratégica y obviamente un éxito en general en el proyecto como tal.

Y el inicio de esta transición se da por medio del *Ubiquitous language* lo cual es un lenguaje en común para todos los miembros del equipo, y el cual se debe definir en el momento exacto de elegir la API que conectará el back con el frontend. La definición de este lenguaje, permite definir límites o una autonomía en los contextos en los cuales cada uno se está desempeñando, incluso genera un factor de **conformidad**, lo que significa que simplemente se conforman con poder consumir la API que les provee un contexto a otro, sin interesarse realmente como se hizo o qué necesita.
En este campo también entra el concepto de *continuous integration,* lo cual posibilita un alta mitigación de errores en el código.

Y así, de una forma muy superficial, se puede tener un concepto base de lo que significa el ***Domain driven design.***