

Watch: **DDD en 20 minutos**

Aprende DDD en 20 minutos ⚡ | Domain-Driven Design



¡Bienvenido!

Si quieres saber qué es eso del Domain Driven Design estás en el sitio correcto 🙌 🙌

Veremos en 20 minutos no sólo de qué trata DDD y de los patrones tácticos, también veremos la parte estratégica que con frecuencia se pasa de largo o se describe muy superficialmente.

DDD: Tactical Design

- **Model-Driven Design:** Hace referencia a que modelemos desde nuestro contexto, es decir, dirigir nuestro diseño en base a los modelos. Esto lo conseguiremos a través de:
 - Expresar la semántica de Dominio en términos de
 1. **Services**
 2. **Entities**

3. Value Objects

- Los **Aggregates** son elementos conceptuales que nos permitirán mantener la integridad de las entidades y encapsular los value objects
- **Repositories** en los que persistiremos nuestras entidades
- Cada vez que se produzcan cambios en nuestras entidades lanzaremos **Domain Events**, reflejando que algo ha pasado en nuestro sistema
- Gracias a los **Factories** podremos encapsular nuestras entidades, agregados y value objects. En los [repositorios](#) que utilizamos en los cursos podeis ver ejemplos de cómo implementamos el [Factory Pattern](#)
- Generalmente cuando trabajemos con DDD lo haremos a través de una **Arquitectura por Capas** como es el caso de la Arquitectura Hexagonal.
- Propone el uso de **Ubiquitous Language**, es decir, que los nombres de variables, metodos y clases se establezcan con el lenguaje de nuestro dominio.
- Todo ello se define dentro de un **Bounded Context**, que nos va a permitir una mayor autonomía entre equipos además de establecer límites en el lenguaje

DDD: Strategic Design

- Dentro de nuestro Bounded Context, mantendremos nuestro modelo unificado gracias a la **Integración Continua**
- Hablamos de **Context Map** para referirnos al mapeo realizado de los términos que utilizamos en real a conceptos lógicos o físicos. Este mapeo nos ayudará además a
 - Separar el **Big Ball of Mud** de nuestra aplicación, y llevar a cabo la publicación de eventos en lugar de adentrarnos en un código costoso de desacoplar
 - Identificar cómo aplicar el **Anticorruption Layer**
 - Cada equipo puede **Trabajar de forma separada** (Podremos, por ejemplo utilizar distintos lenguajes de programación para los diferentes contextos de nuestra aplicación)
 - Facilita la publicación de **Open Host Services** para la comunicación entre los Bounded Contexts y entre la estructura

organizacional de nuestra aplicación (**Published Language**)

- Posibilita la relación entre contextos a través de un **Shared Kernel** y en términos de **Customer/Supplier Team** y de **Conformist**







Conclusión

- Los Bounded Contexts son una pieza crítica del desarrollo
- El lenguaje obicuo debe definirse desde el momento inicial en que establecemos el contrato de la API con la que interactuará frontend y backend, así todos los elementos que interactúan deben impregnarse de dicho lenguaje

Hemos visto un montón de conceptos muy de pasada y seguramente todo resulte por ahora bastante confuso, ¡Pero no os preocupéis! En las próximas lecciones del curso los veremos con mucho más detalle acompañado de ejemplos de código en [PHP](#), [Java](#) y [Scala](#), y aprenderemos todo lo que puede aportarnos el DDD ¡Nos Vemos!

Cursos relacionados




















Cursos Previos:

-  [Principios SOLID Aplicados](#)
-  [Arquitectura Hexagonal](#)
-  [Testing: Introducción y buenas prácticas](#)
-  [CQRS](#)
-  [Event-Driven Architecture](#)
-  [CQRS+Event Sourcing](#)

Siguientes Cursos:

-  [DDD en PHP](#)
-  [DDD en Java](#)

Cursos complementarios:

- Tooling: ¡Sácale el máximo provecho a tus herramientas de trabajo!
 -  [Exprimiendo IntelliJ IDEA](#) (PhpStorm, Rider, Goland, PyCharm y derivados)
 -  [Terminal 100% productiva con Zsh](#)
 -  [Git: Introducción y trabajo en equipo](#)
 -  [Integración Continua con GitHub Actions](#)
- Sistemas: ¡Prepárate para un entorno DevOps!
 -  [Git: Introducción y trabajo en equipo](#)
 -  [Docker: De o a deploy](#)
 -  [Kubernetes para desarrolladores](#)
 -  [ELK+Beats: Centraliza logs con Elastic Stack](#)
 -  [Agile: Retrospectivas](#)
 -  [AWS: Tu primer deploy en EC2](#)
 -  [AWS: Autoescalado de aplicaciones con ALB y ASG](#)
- Front web y mobile:
 -  [Migración progresiva a VueJS](#)
 -  [ReactJS: De o a deploy](#)
 -  [Crea una app con VueJS y Jest aplicando TDD](#)
 -  [Testing unidirectional dataflow con Vuex y Jest](#)
 -  [Testing en iOS y Android](#)
 -  [Primera app en Go](#)
 -  [Introducción a Scala](#)
 -  [Buenas prácticas de BDD con Gherkin \(Cucumber, Behat...\)](#)

Textos del curso contribuidos por Pablo León (¡Gracias 🙌!)