


Watch: **Diferencias entre Bounded Contexts, Subdomains, Modules, y Shared Kernel**
17m · 100 Points


1.00		Framework coupled code	Modules	Bounded Contexts	Microservices
Learning curve		Low	Medium	High	High++
Teams autonomy		Low	Medium+	High	High++
Infrastructure		Shared & coupled	Shared	Isolated & distributed	Isolated & distributed
Code maintainability/ extensibility		Low	High	High+	High++
Infrastructure complexity		Low	Medium	High	High+++
Integration			Buses	Buses	Distributed buses

Diferencias entre Bounded Contexts, Microservicios, Subdomains, Modules y Shared Kernel

Aunque ya hemos visto algunos conceptos como Bounded Contexts, Modules... En este video profundizaremos algo más en ellos y en sus diferencias con otros elementos que, cuando empezas, podrían resultar similares entre sí o cuanto menos confusos.

- **Subdomain:** Hace referencia exactamente a lo mismo que los Bounded Contexts. Mientras que *Subdomain* estaría en el terreno de la problemática que queremos resolver, *Bounded Context* estaría en el terreno de la solución que damos a dicho problema.
- **MicroServicio:** Es una práctica habitual desplegar aplicaciones como Microservicios. Aunque normalmente suele ir en relación 1 a 1 con los Bounded Contexts, también es posible que se relacionen 2 microservicios (Por ejemplo la parte backend y frontend) con un mismo "contexto"
- **Shared Kernel:** Se trata de código compartido entre Bounded Contexts y entre Módulos dentro de un mismo Bounded Context (También podemos encontrarlo bajo el nombre de Common o Shared). Aquí dejaremos las cosas con la menor lógica posible para no generar acoplamiento en los contextos.
- **CodelyTV Tip** 📌: Si nos llevásemos un Bounded Context en otro proyecto distinto, tendríamos que llevarnos tanto ese Bounded Context como la carpeta de Shared Kernel

Diferencias

	Framework coupled code	Modules	Bounded Contexts	Microservices
Learning curve	Low	Medium	High	High++
Teams autonomy	Low	Medium+	High	High++
Infrastructure	Shared & coupled	Shared	Isolated & distributed	Isolated & distributed
Code maintainability/ extensibility	Low	High	High+	High++
Infrastructure complexity	Low	Medium	High	High+++

Curva de Aprendizaje ↗

La curva de aprendizaje se incrementa considerablemente cuando nos enfrentamos a un sistema distribuido entre distintos microservicios frente a aquel código acoplado al framework en el que todo está junto. Por eso, lo ideal es avanzar progresivamente, iterando en los distintos estadios hasta llegar a una estructura de microservicios.

Autonomía de los equipos 👤

Trabajar con Bounded Contexts y Microservicios nos va a permitir separar los equipos de modo que puedan desarrollar y desplegar de manera independiente.

Al otro lado, trabajar con un código acoplado nos obliga a que los equipos coincidan en el mismo código tanto al desarrollarlo como en los despliegues.

Infraestructura 🏠

En un código acoplado al framework, puesto que todo está unido, podemos toparnos con un efecto dominó 🧨 de modo que si una de las piezas implicadas falla, todo lo demás falle.

En el otro extremo, al estar aislada la infraestructura, si una de ellas falla (Por ejemplo la referente a Usuarios) el resto seguiría funcionando con normalidad.

Mantenibilidad/Extensibilidad del código 🛠️

Por supuesto, cuanto más desacoplado y aislado esté nuestro código, más fácil será mantenerlo y extenderlo. Como hemos comentado en otras ocasiones, tener nuestro código fuertemente acoplado nos obligará a que cada modificación que hagamos lleve a tocar en múltiples sitios.

Complejidad de la Infraestructura 🐼

Tener menos elementos implicados, como en el caso del código acoplado, supone una menor complejidad en términos de infraestructura. Por eso al trabajar con Bounded Contexts y especialmente con Microservicios, estaremos aumentando bastante su complejidad (No será lo mismo tener un sólo deploy y una sola BD que tener múltiples deploys y trabajar con varias BDD).

Comunicación entre ellos 💬

En el caso del código acoplado, la comunicación será inevitablemente acoplada (Por ejemplo a través de Inyección de Dependencias).

Por otro lado, la comunicación puede ser a través de Buses en el caso de Módulos y Bounded Contexts, o por medio de Buses distribuidos entre Microservicios

Despliegue 🚀

Respecto a nivel de Despliegue, el cambio se observa en el paso a Microservicios, donde será aislado para cada uno de ellos, lo cual nos permitirá una mayor escalabilidad (Se despliega una parte concreta de nuestra aplicación)

¿Alguna duda?

Si tienes cualquier duda o sugerencia para mejorar esta lección, abre una nueva discusión justo aquí abajo y compártela 😊.

Nos vemos en el siguiente video: ¡Estructuras de Carpetas en un Monorepositorio! 📁

Complete Step

100pts

Group Discussion

Start a discussion

-  **Bounded Context con Multi-tenant**
Vladimir S. 4 days ago 🗨️ +0 +0 [Answer >](#)
-  **Comunicación via buses internos?**
Leandro B. 3 months ago 🗨️ +2 +2 [Answer >](#)
-  **¿Bounded context en distintos repositorios?**
Iván R. 4 months ago 🗨️ +1 +1 [Answer >](#)
-  **Equipos de trabajo y Shared Kernel**
Pablo L. 8 months ago 🗨️ +1 +2 [Answer >](#)
-  **No carga el video**
Harold P. a year ago 🗨️ +1 +0 [Answer >](#)