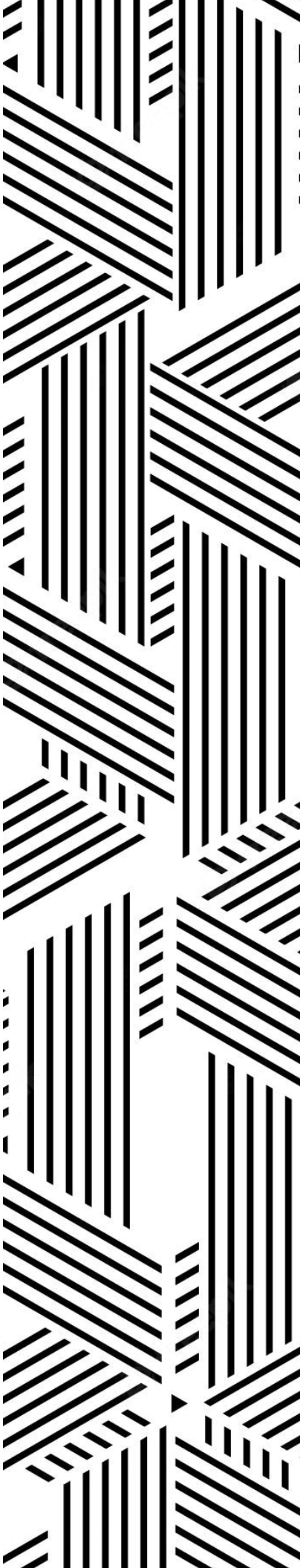
DOMAIN-DRIVEN DESIGN

por Alejandro Chacón



22 de junio del 2023

Contenidos

1. Introducción

Parafraseando las palabras de Eric Evans (2017), *el dominio es el corazón del software*. El software se hace para la resolución de problemas relacionados con el dominio — problemas del usuario final. Todos los demás atributos de un sistema están allí para soportar esta meta. El dominio es, a su vez, la fuente de la complejidad del desarrollo de software. Irónicamente, el dominio es comúnmente ignorado por equipos de desarrollo, los cuales se concentran en aspectos técnicos y tecnológicos del software, situación que puede devenir en software que, a pesar de su robustez tecnológica, falla en cumplir el requerimiento más importante de todo sistema: la resolución de los problemas del usuario final.

2. Domain-Driven Design: ¿Qué es?

Para abordar el dominio del software con mayor atención, se creó el Domain Driven Design, metodología arquitectónica centrada en el modelado del software con la intención de hacer su estructura y funcionamiento análogos con los del dominio al cual va destinado, para así alinearlo íntimamente con los requisitos del negocio.

Por dominio, nos referimos al área específica o problema en el que el sistema de software debe operar. Ya sea comercio electrónico, finanzas, el sector salud o cualquier otra industria o mercado, el dominio representa un conjunto de conocimientos, experiencias y desafíos asociados con esa industria. Ergo, al ganar un entendimiento profundo de un dominio en particular, los desarrolladores pueden establecer un lenguaje compartido y *ubicuo* con los expertos d el dominio para los cuales están desarrollando la solución de software, permitiendo así una comunicación y colaboración efectiva entre ambos grupos de interesados.

3. Tipos de diseño

3.1. Diseño estratégico

Dentro del DDD, se encuentran dos tipos de enfoques de diseño; el diseño estratégico y el diseño táctico. El primero se refiere a trazar una ruta (*roadmap*) que permita visualizar la evolución planeada del software a largo plazo. El enfoque estratégico encara las preocupaciones de alto nivel, e.g. la identificación de dominios, el establecimiento de relaciones entre los contextos limitados del dominio, y la definición de patrones de mapeo. El diseño estratégico permite que los equipos de desarrollo alineen sus decisiones técnicas con las metas y prioridades del negocio, asegurando que el sistema evoluciona de manera que apoye a los requerimientos cambiantes de su dominio.

3.1.1. Lenguaje Ubicuo

El lenguaje ubicuo, permitido gracias al DDD, consiste en un conjunto de términos y conceptos bien definidos y relacionados con el dominio, los cuales han sido acordados tanto por el equipo de desarrollo como por los expertos del dominio. Este lenguaje se vuelve un puente entre el dominio y la implementación del software, ayudando a limar dificultades de la comunicación y facilitando la colaboración. Este lenguaje compartido influencia todo el ciclo de vida del software, desde discusiones y documentación, hasta la implementación del sistema y sus pruebas.

3.1.2. Modelo del Dominio

Hablar de DDD es hablar de modelos. Para poder crear un buen sistema, es necesario tomar el conocimiento de los expertos del dominio, regado en sus mentes y nebuloso, para después traducirlo a modelos que expresen de forma clara y concisa la forma que debe tomar el sistema a implementar. Eric Evans lo llama "Triturar" (crunch) conocimiento para convertirlo en modelos: mientras más se acerquen nuestros modelos al dominio, más fácil será modificarlos y extenderlos a futuro, pues cuando se descubra algo nuevo del dominio, se sabrá exactamente en dónde se debe colocar dentro de los modelos.

Para efectos del DDD, el modelo más importande es el del dominio. Este diagrama es una representación gráfica de todos los conceptos de un negocio: sus interrelaciones, la forma en que se componen, etcétera. El modelo del dominio es la visualización del lenguaje ubicuo promovido por el DDD.

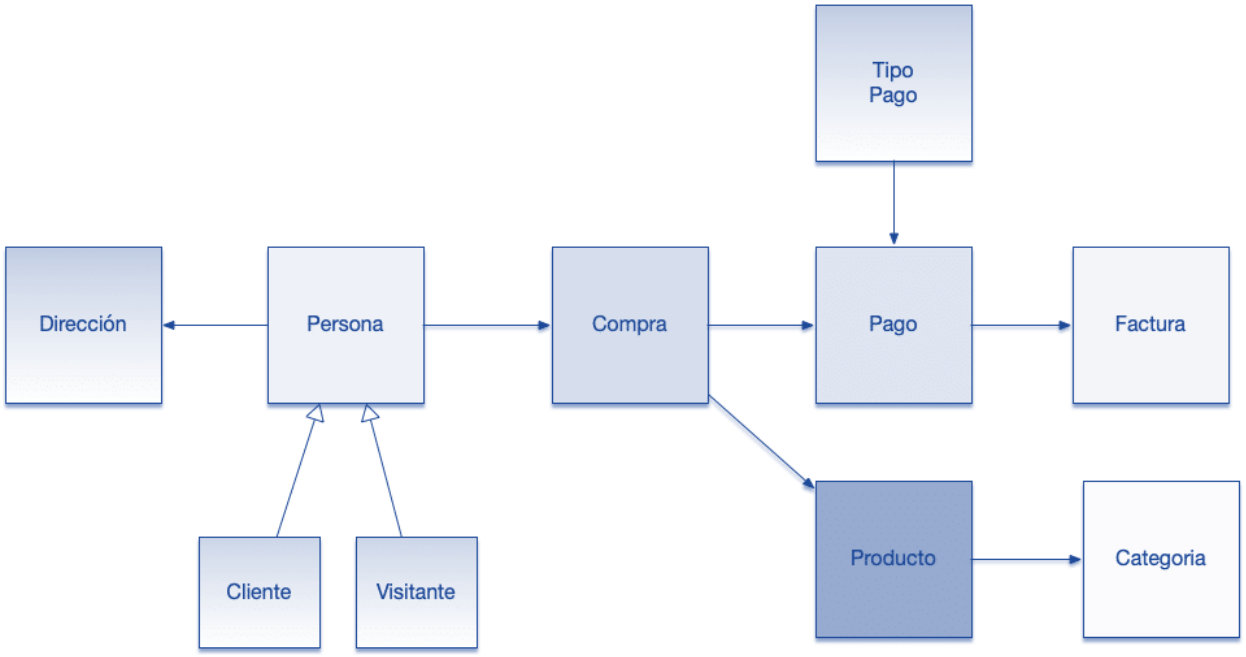


Ilustración 1. Modelo del dominio

3.1.3. Contextos Limitados

Ahora bien, cuando se *mapea* un dominio particularmente grande y/o complejo, es normal que nuestro "diccionario" de términos compartidos crezca enormemente.

En estos casos, DDD recurre a los contextos limitados (*bounded contexts*). Este término fundamental ayuda al desarrollador a manejar la complejidad de un sistema grande, pues divide el dominio en subdominios, cada uno especializado en un área del dominio general. Cada contexto limitado representa un subsistema cohesivo y autocontenido, permitiendo a equipos de desarrollo concentrarse en un área específica del dominio sin sentirse aturdidos por la complejidad del sistema

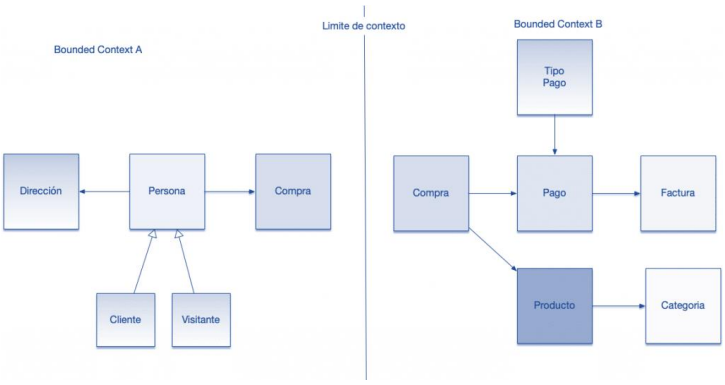


Ilustración 2. Modelo del Dominio con Contextos Limitados

entero, y así tomar decisiones arquitectónicas que sean acotadas a ese subsistema en específico. Desde esta perspectiva, podríamos decir que un contexto limitado es análogo a una clase o a un componente escrito en un lenguaje OOP, pues sigue los mismos principios de cohesión, encapsulamiento y modularidad que permiten al desarrollador dividir y preocuparse por fragmentos de funcionalidad.

3.1.4. Mapeo de Contextos

No es suficiente con dividir los contextos, sin embargo. Una vez acotados, se deben definir tanto las relaciones entre los contextos, como las relaciones entre los equipos responsables por su desarrollo — se deben *mapear*. Vaughn Vernon describe varias formas en que se pueden integrar múltiples contextos limitados, incluyendo: asociación, kernel compartido, proveedor, conformista y capa anticorrupción.

Asociación	Describe la relación entre los equipos de desarrollo, y es útil cuando dos equipos distintos están construyendo dos contextos limitados distintos, los cuales tienen conjuntos dependientes de metas. Cada equipo debe entender parte del Lenguaje Ubicuo de su equipo asociado, específicamente aquellos conceptos que son de interés para su contexto particular.
Kernel compartido	Dado cuando múltiples contextos limitados utilizan el mismo modelo. En términos de diseño, esto significa que ambos equipos de desarrollo manejan el mismo Lenguaje Ubicuo; en cuanto al código, ambos equipos compartirán librerías, o servicios.
Proveedor	Sucede cuando un contexto limitado (upstream) ofrece su diseño a otro contexto limitado (downstream), donde el upstream intenta cumplir las expectativas del downstream. Pasa cuando los contextos están en la misma empresa
Conformista	Similar a proveedor, pero el upstream no se preocupa por alcanzar los requerimientos del downstream. Sucede cuando se integra un contexto al contexto de una solución grande y ya establecida, o cuando el contexto requiere usar de un API, la cual sirve muchos clientes aparte del contexto nuevo.
Capa anticorrupción	Relación upstream/downstream, donde el downstream tiene una capa que lo protege del upstream, conocida en inglés como ACL. Cuando el upstream pasa sus modelos al downstream, la capa ACL traduce esos modelos a modelos nuevos que se acoplen a las necesidades y reglas del downstream, garantizando que la integridad del downstream se quede intacta, y que no se introduzcan conceptos que no perjudiquen su lógica.

Tabla 1. Formas de relación en el mapeo de contextos

3.2. Diseño Táctico

Y mientras el diseño estratégico aborda la dirección general del proyecto, el diseño táctico se concentra en los detalles individuales de cada contexto limitado. En este nivel del diseño, los desarrolladores aplican técnicas de modelado del dominio para expresar conceptos del negocio y reglas, en la forma del ya mencionado modelo del dominio. Para esto, utiliza los siguientes conceptos:

Entidades	Son objetos que pueden cambiar, con un identificador único. Tienen una vida propia dentro del modelo del dominio.
Objetos valor.	Son inmutables y no tienen una identidad propia, a diferencia de las entidades. Se identifican sólo por los valores en sus atributos. Por esta razón, para actualizar a un objeto valor, se debe crear una instancia nueva que reemplace a la anterior.

Agregados	Un agregado es un conjunto o clúster de una o más entidades, y puede contener objetos valor; el padre de un clúster se llama raíz del agregado.
Servicios	Objetos sin estado que realizan alguna lógica que no embona con alguna operación de una entidad u objeto valor. Realizan operaciones específicas del dominio, las cuales pueden relacionarse con múltiples objetos del dominio.
Repositorios	Usados principalmente para lidiar con almacenamiento. Se responsabilizan por persistir a los agregados.
Fábricas	Se encargan de abstraer la construcción de un objeto, y pueden regresar la raíz de un agregado, una entidad, o un objeto valor.
Eventos	Sucesos significativos que ocurren en el dominio y necesitan ser reportados a otros interesados del dominio. Es normal que los agregados publiquen eventos.

Tabla 2. Elementos del diseño táctico

4. Conclusión

El Diseño Orientado al Dominio ofrece múltiples beneficios al desarrollo de software. Conlleva a sistemas que se alinean mejor al dominio del negocio, haciéndolos más mantenibles y adaptables a los requerimientos cambiantes. DDD alienta la colaboración entre distintos equipos técnicos y expertos del dominio, fomentando un entendimiento universal y reduciendo el riesgo de problemas de mala comunicación. También, al concentrarse en los conceptos del negocio más importantes, DDD ayuda a los equipos a identificar y priorizar las funcionalidades más importantes, lo que promueve procesos de desarrollos más centrados y eficientes.

5. Referencias

Batista, F. (2022a, March 4). *What is Strategic Design ? - DDD - The Domain Driven Design*. DDD. https://thedomaindrivendesign.io/what-is-strategic-design/

Batista, F. (2022b, March 4). *What is Tactical Design ? - DDD - The Domain Driven Design*. DDD. https://thedomaindrivendesign.io/what-is-tactical-design/

Caules, C. Á. (2021). ¿Que es un Bounded Context? *Arquitectura Java*. https://www.arquitecturajava.com/que-es-un-bounded-context/

Evans, E., & Evans, E. J. (2004). *Domain-driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional.

Kexugit. (2016, January 22). *Best Practice - An Introduction To Domain-Driven Design*. Microsoft Learn. https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/february/best-practice-an-introduction-to-domain-driven-design