**Domain Driven Design**

Desarrollá las preguntas de forma clara y concisa. El desarrollo de todas las preguntas no debe exceder dos hojas.

1. ¿Qué es el **lenguaje ubicuo** y cómo contribuye a la alineación entre el equipo técnico y los expertos del dominio?
2. Diferenciá claramente una **entidad** de un **value object** en el modelo del dominio, ejemplifique.
3. Qué es un **bounded context** y explica por qué su correcta delimitación es clave para manejar la complejidad y la evolución de sistemas grandes.
4. ¿Cómo ayudan los diagramas UML (como diagramas de clases, secuencia y casos de uso) a definir y comunicar el modelo del dominio? ¿De qué manera favorecen el lenguaje común entre desarrolladores y expertos del negocio?

Respuestas:

1. El **lenguaje ubicuo** es un lenguaje común, preciso, compartido y acotado del negocio que se construye entre los desarrolladores y los expertos del dominio. Hace referencia a un vocabulario que se utiliza en el código, en las reuniones, en la documentación y en las conversaciones del equipo.

Contribuye a la alineación entre el equipo técnico y los expertos del dominio de manera tal que:

* Reduce la ambigüedad: todos usan las mismas palabras con el mismo significado.
* Facilita la comunicación: se eliminan malentendidos entre los desarrolladores y los expertos del negocio.

Su objetivo es que no se presenten confusiones respecto de los términos que se utilicen, ya que una palabra puede ser percibida con distintos significados. Por ejemplo, si hablamos de transacciones, métodos de pagos, se hace referencia al pago, y este sería el lenguaje ubicuo porque se sabe de que se está hablando.

1. Una **entidad** es un elemento que se caracteriza por ser identificable por un único valor o un conjunto de estos. Por ejemplo, una persona tiene dni, nombre, apellido, hobbies, etc (el dni sería un valor único). Su identidad no depende de sus atributos, ya que estos pueden cambiar pero sigue siendo la misma entidad (puede cambiar de nombre, de apellido, pero el número de dni siempre va a seguir siendo el mismo).

En cambio, un **value object** es un elemento que es identificable por el conjunto completo de su contenido y, en caso de modificar cualquier valor del mismo, se obtendría otro elemento distinto. Por ejemplo, una persona vive en una determinada dirección, la cual tiene una calle, ciudad, código postal, etc. La dirección, es definida por sus atributos y estos son inmutables (una vez creados, no se cambian). En el caso de que la persona se mude, no se cambiaría el objeto dirección, sino que se crea uno completamente nuevo.

ACA HAY DUDAS: SEGÚN EL POWER POINT DE LA PROFE: “Objeto de valor: elemento similar a la entidad, salvo que es identificable por el conjunto completo de su contenido y, **en caso de modificar cualquier valor del mismo, se obtendría otro elemento distinto.** Ejemplo: EmailComprador. El objeto de valor es lo que contiene el atributo (o sea, el valor del atributo).”  
En el ejemplo del pedido, dice que el value object es la cantidad, pero si esa cantidad pasa de 50 a 100, no se tendría que crear un nuevo pedido, no tiene sentido. Solamente cambias el número y ya está.  
Con el ejemplo del mail, si vos sos una persona y tenes un mail, pero despues lo cambias, no vas a crear una nueva persona porque cambio de mail, vas a crear un nuevo mail que va a tener los mismos atributos, pero con diferentes valores (otro nombre, otra terminación como .hotmail, etc).

1. **Bounded context** es un ámbito delimitado donde se aplica un lenguaje ubicuo coherente. Las reglas y conceptos tienen un significado claro y consistente, pero solo dentro de ese ámbito. Por ejemplo, la palabra “producto” , en ventas, puede referirse a lo que el cliente ve y compra, o, en logística, puede referirse a lo que se mueve dentro del almacén, o, en facturación, puede referirse a lo que se factura legalmente. El significado de esta palabra cambia según su contexto.

Su correcta delimitación es clave para manejar la complejidad y la evolución de sistemas grandes ya que:

* Reduce la complejidad del sistema (si surge un error, será en un módulo específico y no en todo el sistema, lo cual facilita encontrarlo y arreglarlo).
* Evita malentendidos.
* Organiza equipos y responsabilidades de forma clara.
* Hace que el sistema sea más escalable, mantenible y modular.

1. Los diagramas UML ayudan a definir y comunicar el modelo de dominio de varias maneras:

* Clarifican conceptos y relaciones: representa visualmente la entidades del dominio, sus atributos, métodos y relaciones.Por ejemplo, es mucho más claro y preciso ver una relación de herencia en un diagrama de clases que explicarlo solo con texto.
* Prevén errores: ayudan a detectar confusiones tempranas (que luego se verían reflejadas en el código y su desarrollo).

Favorecen el lenguaje común entre desarrolladores y expertos del negocio de manera tal que los diagramas UML permiten comunicar ideas entre roles distintos; el equipo técnico (devs) y el funcional (negocio) pueden usar los mismo diagramas como base para las conversaciones. También estos se convierten en una herramienta visual del lenguaje ubicuo.