UNIVERSIDAD DE ANTIOQUIA

1 8 0 3

Domain Driven Design: Ensayo

Andres Grisales Gonzalez[1] Cristian Camilo Serna Betancur[2] John Fredy Mejia Serna[3]
supervisado por Prof. Robinson Coronado G.
Arquitectura de Software

***Domain Driven Design*—El aspecto más complicado de los grandes proyectos de software no es la implementación, es el dominio del mundo real al que sirve el software. Domain Driven Design es una visión y un enfoque para tratar con dominios altamente complejos que se basa en hacer que el dominio en sí sea el foco principal del proyecto y mantener un modelo de software que refleje una comprensión profunda del dominio.**

DOMAIN DRIVEN DESIGN: ENSAYO

EL Domain Driven Design o Diseño Dirigido por Dominio es una metodología y prescripción de procesos para el desarrollo de sistemas complejos; muchas empresas con dominios extremadamente complejos confían en él para producir software que pueda evolucionar rápidamente junto con el negocio. El énfasis del diseño impulsado por el dominio es comprender el dominio del problema para crear un modelo abstracto del dominio del problema que luego pueda implementarse en un conjunto particular de tecnologías.

El Diseño Dirigido por Dominio nos introduce el lenguaje ubicuo—el cual consideramos una forma abreviada de enfatizar el principio fundamental del Diseño Dirigido por Dominio—que se define como un lenguaje estructurado en torno al modelo de dominio y utilizado por todos los miembros del equipo para conectar todas las actividades del equipo con el software. Este lenguaje trata sobre el dominio empresarial y utiliza terminología del ámbito empresarial, no términos técnicos de TI. En otras palabras, haz uso de la terminología de dominio en todas partes; hazlo omnipresente. Podemos afirmar con seguridad que uno de los problemas que enfrentan muchos esfuerzos de desarrollo de software es la fricción constante que introduce la traducción entre dos vocabularios técnicos, el del dominio empresarial por un lado y el de los desarrolladores por el otro. Hasta cierto punto, esta dualidad es inevitable: los desarrolladores deben enmarcar su trabajo en términos de algoritmos y computación, que generalmente no tienen un equivalente directo en el vocabulario empresarial. De ahí la necesidad de adoptar una política lingüística que mitigue estas dificultades.

El Diseño Dirigido por Dominio se divide en dos grandes mundos, estos son: el estratégico y el táctico, hay quienes dicen que se puede aplicar el táctico sin el estratégico, otros dicen que si se está aplicando el Diseño Dirigido por Dominio se debe aplicar el estratégico y el táctico; esta decisión queda a consideración del lector. En el estratégico se habla de que se deben identificar unos contextos delimitados que tienen definidos cada uno de ellos un lenguaje ubicuo: Se propone entonces que por cada contexto limitado se tenga un repositorio de código propio para evitar confusiones en la terminología—por eso en cada uno se define una terminología con un significado en especifico de acuerdo al contexto—. Por otro lado se tiene al táctico que nos invita a utilizar ciertas herramientas y tener prácticas que ayuden precisamente a centrarse en el negocio, entre esas invitaciones está hacer uso de modelos de dominio—pueden ser entidades—, utilizar el patrón repositorio para la persistencia de las entidades, identificar agregados de acuerdo a las necesidades del negocio e intentar tener agregados lo más pequeños posibles para favorecer tasa de éxito de transaccionalidad en el sentido que las entidades que hagan parte de un mismo agregado se referencian directamente mientras que las entidades de agregados diferentes se referencie únicamente por su identificador, y finalmente hacer uso de valores de objeto para modelar conceptos de dominio que no tienen identidad propia como tal y solo valen por lo que tienen.

El proceso de Diseño Dirigido por Dominio implica la colaboración entre desarrolladores y no desarrolladores. Idealmente, habrá un modelo compartido con lenguajes compartidos para que, a medida que personas de diferentes dominios con diferentes perspectivas discutan la solución, tengan una base de conocimientos compartida con conceptos compartidos. Entre sus ventajas destacamos la facilidad de comunicación: en términos generales, el Diseño Dirigido por Dominio es muy útil cuando se trata de ayudar al equipo a crear un modelo común. Los equipos del lado del negocio y del lado del desarrollador pueden usar este modelo para comunicarse sobre los requisitos comerciales, las entidades de datos y los modelos de proceso. Proporciona flexibilidad: el Diseño

Dirigido por Dominio da la vuelta a los conceptos de diseño orientado a objetos. Esto implica que casi todo en el modelo de dominio se basa en un objeto y, por lo tanto, será modular y encapsulado, lo que permitirá cambiar y mejorar el sistema de forma regular y continua. Patrones mejorados: el Diseño Dirigido por Dominio brinda a los desarrolladores de software los principios y patrones para resolver problemas difíciles en software y, a veces, en negocios. Riesgo reducido de malentendidos: Los requisitos siempre se explican desde la perspectiva del dominio. La conceptualización del sistema de software en términos del dominio empresarial reduce el riesgo de malentendidos entre los expertos del dominio y el equipo de desarrollo. Mejor coordinación del equipo: La coordinación está más impulsada por las personas—ya que todos usan la misma terminología—.

No obstante y como los estudiantes objetivos que somos, nos vemos obligados a destacar algunas de las desventajas que encontramos dentro de este enfoque. Esfuerzos adicionales necesarios: La principal desventaja de introducir el Diseño Dirigido por Dominio en el desarrollo de software es el esfuerzo adicional requerido para adoptar este enfoque de desarrollo de software. Se necesita más esfuerzo y tiempo para crear un modelo sustancial del dominio empresarial antes de que se hagan evidentes los efectos positivos en el proceso de desarrollo. Requiere expertos en dominio: Los proyectos que plantean hacer uso del Diseño Dirigido por Dominio requieren expertos en el dominio—debe existir una estrecha relación entre los desarrolladores y los expertos del dominio, el conocimiento profundo del dominio es esencial, al igual que la colaboración con los expertos del desarrollo durante la vida del proyecto. Esto evitará malos entendidos entre las partes del equipo y ofrecerá la oportunidad de obtener un conocimiento más profundo del dominio—. Solo apto para aplicaciones complejas: el Diseño Dirigido por Dominio fue diseñado para simplificar la complejidad. Es un gran enfoque para el desarrollo de software si existe la necesidad de simplificarlo, pero para aplicaciones simples, no vale la pena usarlo. La alta encapsulación puede ser un problema: El alto nivel de aislamiento y encapsulación en el modelo de dominio puede representar un desafío para los expertos en el dominio empresarial.

Para concluir, se puede decir que el enfoque del Diseño Dirigido por Dominio mueve la óptica de las metodologías, herramientas y gestión de proyectos al núcleo del sistema de software que se está desarrollando y el dominio de expertos con el que se involucrará. Algunas personas piensan que no es importante, realmente es muy importante cuando se requiere crear sistemas complejos—no son simples CRUDs—y que necesitan escalar bien en el tiempo sin agregar complejidad accidental. También es cierto que si no se hace bien se pueden cometer errores que agregan más complejidad accidental. Como dijo en algún momento Steve Jobs: *“El diseño no es solo lo que ves, sino cómo funciona, el diseño es cómo funciona”*.