# MODELAMIENTO BASADO EN EL DOMINIO:
## Estado del Arte

**Verónica Macías Mendoza**
**Ingeniera en Computación, Profesora ESPOL, mmacias@fiec.espol.edu.ec , 593 4 2269313**

**Resumen**
El Modelamiento basado en el Dominio o Domain-Driven Modeling, es un nuevo paradigma de Software que destaca la construcción de herramientas de software sobre la base del Modelo del negocio o dominio. El modelo debe abstraer y mostrar de forma clara los principios fundamentales que rigen al dominio particular de los detalles tecnológicos tales como lenguajes, herramientas de comunicación, etc., que pueda requerir el producto final; permitiendo así que el modelo original sea independiente de la plataforma de desarrollo PIM -Platform Independent Model-. Este artículo describe en que consiste este paradigma, denotando las necesidades que motivaron el desarrollo del mismo y las bases sobre las que ha sido construido; se citan ejemplos de herramientas y arquitecturas que soportan este paradigma y las principales ventajas y desventajas que este ofrece a los encargados de la producción y mantenimiento de herramientas de software. Se concluye indicando que el Modelamiento basado en el Dominio brinda a los sistemas construidos ciertos aspectos importantes en cuanto a calidad; más también se concluye que esta es una rama joven aún, en la cual queda mucho por investigar.

**Palabras Claves**
Modelamiento basado en el dominio, Domain-Driven Modeling, MDD, Model-driven development, MDA, Model-driven architecture

## 1. Introducción

En el siempre cambiante mundo de la computación, donde cada día las computadoras son más veloces y la complejidad de los sistemas a desarrollar incrementa constantemente; el éxito de los departamentos y compañías de tecnología de información (TI) están siendo juzgados por la habilidad de construir sistemas de acuerdo a las cada vez más estrictas restricciones de tiempo y calidad que impone el mercado, exigiendo además que estos sistemas sean los suficientemente flexibles para responder de forma ágil a los cambios de requerimientos tecnológicos o del negocio mismo [1]. Es fácil suponer entonces, que aquello que buscan todos los métodos, tecnologías, arquitecturas y lenguajes existentes, es brindar a la gente que trabaja en TI herramientas y/o procesos capaces de acelerar la construcción de sistemas de software de calidad.

Son muchos los motivos por los que un proyecto de software puede quedar relegado en el olvido: burocracia, falta de recursos, falta de personal preparado, etc.; pero la forma en como fue diseñada la solución es la que finalmente determina cuan complejo puede llegar a ser un problema de software [2]. Por ello es que todos los autores de los más importantes libros y artículos de Software e inclusive el común de los trabajadores en TI, coinciden en que un software de calidad solo puede ser realizado amparado por un buen diseño.

En las disciplinas tradicionales de Ingeniería estos buenos diseños constan siempre de modelos del producto final. Así por ejemplo, nadie imagina la construcción de un avión, casa, puente, etc. sin que antes haya existido un modelo del mismo. Sin embargo, a pesar de que alrededor de la Ingeniería de Software se han establecido varios paradigmas de modelamiento, la construcción y uso de modelos es frecuentemente considerada por los desarrolladores como una molestia en lugar de una ayuda para su trabajo.

El Modelamiento basado en el Dominio es un paradigma que establece que la construcción de productos software debe estar basada en un modelo, lo que no es una nueva idea dentro de la Ingeniería de Software, y que este modelo debe estar principalmente enfocado en el dominio –espacio conceptual del problema, negocio- y la lógica del mismo, separando a estos de las especificaciones tecnológicas del sistema [3]. Es esta última característica la que “asegura” la producción eficaz y efectiva de productos de software más flexibles; siendo estos posibles resultados la bandera que portan los defensores de este paradigma para motivar a los que trabajan en TI al uso e investigación de métodos y herramientas que soporten este paradigma.

## 2. El Modelo

Un modelo es una representación abstracta de alguna cosa que puede o no existir en la realidad; es una simplificación que muestra ciertos aspectos de lo que se desea modelar, escondiendo aquellos elementos que no son de interés a los que usaran el modelo [2]. Por ejemplo; podemos tener varios modelos de un auto, el constructor del auto necesitará que su modelo tenga *todos* los detalles técnicos necesarios para la posterior construcción del auto; pero para la persona que comprará el auto, el modelo del mismo debe mostrar solo las características físicas exteriores del auto y cuando mucho conocer de las especificaciones técnicas más relevantes del mismo, aquellas que a él, como usuario del auto, le interesan -fuerza del motor, cilindraje, etc.-.

Los modelos pueden ser expresados de muchas formas; ya que pueden ser representados en todas las formas de lenguaje gráficas o textuales existentes. Así puedo tener un modelo de un sistema representado por Casos de Uso escritos en lenguaje UML, o puedo tener el código en lenguaje Java del sistema, que también, para sorpresa de algunos, es un modelo del sistema (recordemos que el sistema final es un conjunto de 0s y 1s, de los que el desarrollador del sistema no se entera, es decir hay una abstracción de estos detalles); la diferencia entre los dos modelos mencionados radica en el nivel de abstracción.

Un modelo puede tener múltiples vistas, las mismas que deben ser consistentes entre sí [4]. Así por ejemplo podemos tener varias vistas de un carro, de perfil, de frente, etc. En todo caso, es importante aclarar que no todos los modelos que realizamos necesitan ser ejecutables o formales; pero aquellos que lo sean, podrían alcanzar el beneficio de la automatización [5].

Los modelos son útiles para resaltar los aspectos importantes de la realidad representada, dejando de lado los elementos que puedan distraer la atención de aquello en lo que debemos enfocarnos. De forma específica, han sido aprovechados dentro la ingeniería para representar los problemas planteados y sus posibles soluciones; ya que la abstracción realizada en el modelo se ha utilizado como base para una mejor comunicación entre los diferentes participantes del problema, así como para una mejor compresión del mismo.

Sin embargo, más allá del modelo en sí, lo que realmente interesa a todas las ramas de la ingeniería es la construcción de métodos, herramientas, etc. que sirvan de base para la construcción de modelos que sean realmente útiles; es decir el tema de estudio es el *modelamiento* en sí.

## 3. Modelamiento

El modelamiento es también conocido como análisis; y entre las más importantes metas del mismo, expresadas por Bertrand Meyer [6], tenemos las siguientes:

- Comprender el problema o problemas que el eventual producto, si es que llega a existir alguno, podrá resolver
- Promover la generación de preguntas relevantes acerca del problema y del producto
- Proveer las bases para responder a preguntas acerca de las propiedades específicas del problema o del producto

En caso de que el objetivo final no sea la construcción de un producto, las metas listadas son suficientes para realizar el análisis del problema.

Uno de los principales beneficios del proceso de análisis es el expresado por la meta 2 del listado, ya que la generación de preguntas relevantes sobre el problema nos conduce a una mejor compresión de este, así como a disipar aquellas dudas o ambigüedades que pueden resultar desastrosas para la construcción de un producto final. Para citar un ejemplo de la importancia que representa lo citado, supongamos que un arquitecto contratado para manejar la construcción de una casa pasa por alto preguntar a sus clientes el número de dormitorios que esta debe tener, porque asume que, como todo el resto de los clientes atendidos anteriormente, los clientes actuales quieren tener 3. Sin embargo, cuando el arquitecto se siente a realizar la distribución de los cuartos de los cuartos de la casa en el plano, tendrá que preguntar a los clientes su opinión acerca de ello, lo que seguramente sacará a la luz cuántos dormitorios realmente desean sus clientes. Si la construcción de planos no existiera, existiría una alta probabilidad de que los clientes no queden muy satisfechos con sus casas.

Es evidente que en las otras áreas de la ingeniería, la construcción de modelos ha sido muy útil; sin embargo dentro de la Ingeniería de Software, especialmente dentro del círculo de personas que implementan los sistemas, existe una gran apatía hacia la construcción y uso de los mismos. Son algunas las razones que han llevado a esta consecuencia. Muchos desarrolladores opinan que el modelamiento no es otra cosa más que la construcción de

toneladas de documentos que finalmente no sirven de mucho al momento de desarrollar un sistema [7]. Otros consideran que el problema radica en que estos modelos son difíciles de mantener a medida que avanza el proyecto, ya que mientras esto sucede van surgiendo cambios o se añaden nuevos requerimientos al sistema. Estos cambios a los requerimientos iniciales del sistema llevan a que el desarrollador realice los cambios directamente en el código y, por no querer "perder" tiempo, no actualiza los cambios en el modelo.

Cuando un modelo no representa el producto final construido, éste no servirá para identificar que parte del sistema debe ser analizada para resolver los problemas que puedan surgir cuando este se ponga en producción. Mucho menos este modelo servirá para localizar las partes del sistema que deban ser cambiadas o sustituidas para atender cambios posteriores en los requerimientos tecnológicos o del negocio en sí. Y, sin duda, algo que es muy claro para todos, inclusive para los detractores del modelamiento, es que si tenemos un buen modelo escrito en algún lenguaje de abstracción superior a la del código fuente, será mucho más fácil *visualizar* cuál es la parte del sistema a cambiar –todo el que trabaje en TI entenderá con mucha más facilidad un modelo escrito en UML, que todas las líneas de código fuente escritas por el mejor de los desarrolladores-. Es decir, nuestro sistema será más flexible si el modelo que lo representa es una abstracción correcta y actualizada del mismo [2].

## 4. Modelamiento basado en el Dominio

El problema que se puede observar en los paradigmas de modelamiento de sistemas tradicionales, es que estos se basaban en la generación de documentos que eran muy difíciles de mantener en sincronía con los cambios realizados en el código que atendían a los cambios en los requerimientos iniciales del sistema. O sea que, a menos que no hubiese cambios dentro de estas especificaciones iniciales, el modelo no correspondía al sistema desarrollado. Una de las causas principales de este problema es que estos métodos de modelamiento y los ambientes de desarrollo existentes no son muy buenos en separar los detalles arquitectónicos y tecnológicos del espacio conceptual del problema. Como consecuencia de los anterior los sistemas producidos son extremadamente sensibles a los cambios en el ambiente de desarrollo, y al mismo tiempo, se vuelve muy difícil, sino imposible, reflejar de forma directa en la implementación del sistema los cambios producidos en el dominio. Además, aparte de lo citado anteriormente, esta gran dependencia de la infraestructura tecnológica requiere un alto nivel de experiencia y conocimiento por parte de los equipos de desarrollo; lo que incrementa los costos de contratación de personal o entrenamiento del mismo. [1]

El modelamiento basado en el dominio, también conocido como Desarrollo basado en el Modelo -MDD por sus siglas en inglés Model Driven Development- o como Desarrollo basado en el Dominio –DDD por sus siglas en inglés Domain Driven Development-; toma como base la idea de que la complejidad de un sistema de información está dada por la complejidad del contexto del problema. Entonces, dando una especial atención a comprender los detalles esenciales del dominio, antes que a las especificaciones del sistema en sí, los desarrolladores serán capaces de lidiar de una mejor manera con la complejidad del sistema. [3] Basado en esta idea, el MDD plantea la construcción y uso de un modelo independiente de la plataforma, que represente los conceptos principales del dominio y las reglas del mismo. Por supuesto, la construcción de un sistema no puede estar respaldada únicamente en un modelo del problema, por lo que encima del modelo independiente de la plataforma, debe existir al menos un modelo que represente los detalles tecnológicos del sistema a construir. Este último modelo se conoce como PSM –Platform Specific Model- por sus siglas en inglés. La figura 1 representa la arquitectura básica de un sistema modelado usando el paradigma MDD.

Modelo Específico de la
Plataforma (PSM)

Modelo Independiente de la
Plataforma (PIM)

**Figura 1.** Arquitectura básica de un sistema modelado usando MDD

Por supuesto, dado que esta es la arquitectura básica, la mayoría de los métodos que soportan este paradigma definen su propia subdivisión de la capa superior; pero en todas ellas se mantiene constante la existencia de un modelo que representa el dominio y la lógica del mismo.

Como se había mencionado antes, la separación de los detalles tecnológicos del dominio del problema, siempre que esta separación se mantenga en la implementación misma del sistema, provee al sistema de una mayor

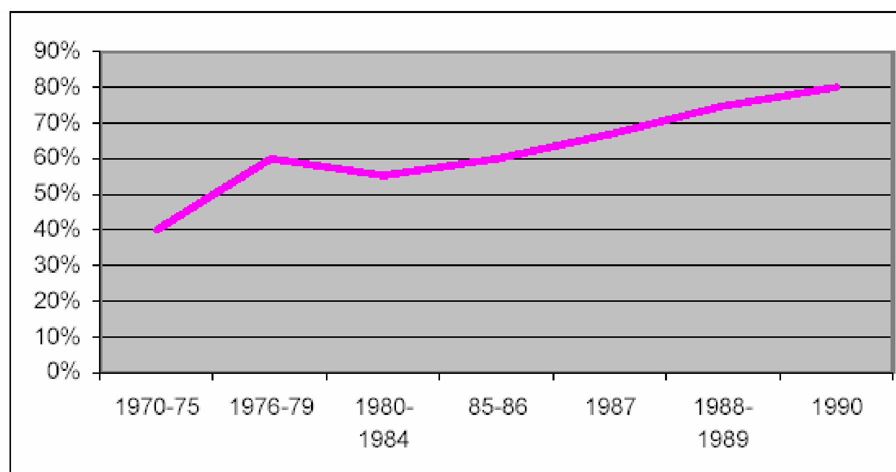
flexibilidad. Cuando se producen cambios en la parte tecnológica del sistema, que normalmente suceden con mayor frecuencia que los cambios del negocio y sus reglas, el análisis del espacio conceptual del problema, no necesita ser evaluado nuevamente, ahorrando tiempo valioso en la mantenibilidad del sistema. Es por eso que MDD propone una conexión más directa entre el modelo y la implementación, a través de lo que define como una implementación por *transformación* que será específica para cada ambiente particular de desarrollo [3]. La consecuencia final es que un método que soporte el paradigma MDD, una vez haya madurado, sea capaz de proveer la base para la construcción de herramientas que permitan la generación automática del código fuente; tal como ahora los compiladores generan el código ejecutable a partir del código fuente de nuestros sistemas.

### 4.1 Ventajas del Modelamiento basado en el dominio

Antes de enumerar las ventajas que ofrece el paradigma del MDD, es necesario revisar algunos detalles que ayudaran a una mejor comprensión de la importancia que estas ventajas pueden ofrecer a los diversos actores participantes en la construcción de productos de software.

Existen muchos factores que afectan la calidad de los productos de software. Existen algunos factores externos, que son aquellos que pueden ser apreciados por el usuario final del sistema, y que por tanto son los que a él le interesan. Entre estos factores podemos mencionar la correcta funcionalidad del sistema, así como la extensibilidad, reusabilidad, compatibilidad y portabilidad del mismo. Por supuesto el más importante de estos factores es la correcta funcionalidad del sistema, que implica que el sistema es capaz de realizar las tareas exactamente en la forma como se supone debe hacerlo. Pero esto no implica que el resto de factores sean menos importantes; ya que es muy probable que el usuario espere que si más adelante desea cambios, estos no sean más costosos o que tomen más tiempo, que el desarrollo del sistema completo. Para poder satisfacer estas necesidades del cliente el sistema debe cumplir con los requerimientos de extensibilidad y reusabilidad. Estas dos últimas características forman parte de lo que se conoce como mantenibilidad del sistema [6].

La mantenibilidad del sistema ha sido considerada por mucho tiempo el patito feo del software, no se conoce muchos desarrolladores a los que les agrade trabajar en el mantenimiento de sistemas existentes, y mucho menos cuando ellos no fueron parte del equipo que desarrollo el sistema original. Pero, para disgusto de nuestros desarrolladores, los estudios realizados demuestran que la fase de mantenimiento del sistema es usualmente la parte que consume mayor tiempo y presupuesto dentro del ciclo de vida de un proyecto.




[Lientz and Swanson, 1980], [Rock-Evans and Hales, 1990], [Schach, 1990], [Pressman, 1993], [Frazer, 1992], [Pigoski, 1997]

**Figura 2.** Historial de la tasa de costo de mantenimiento [8]

Sin embargo es importante anotar, que este incremento en la tasa de costo de mantenimiento no es necesariamente malo. Lo que puede llegar a ser malo es que el retorno de la inversión realizada sea afectado por este incremento.

Un estudio realizado hace varios años por Lientz y Swanson [9], y que puede ser encontrado como referencia válida entre los más importantes libros de Ingeniería de Software -Schach, 2002; Pressman, 2001; Sommerville, 2001-, divide el mantenimiento en 4 tipos de actividades principales: adaptativas, correctivas, preventivas y

perfectivas. El mismo estudio menciona como la más importante de estas actividades a la perfectiva, que consiste en las modificaciones realizadas por cambios en la especificación inicial del sistema o la adición de nuevas funcionalidades al mismo. La siguiente figura muestra la distribución de las actividades de mantenimiento según los resultados de ese estudio:

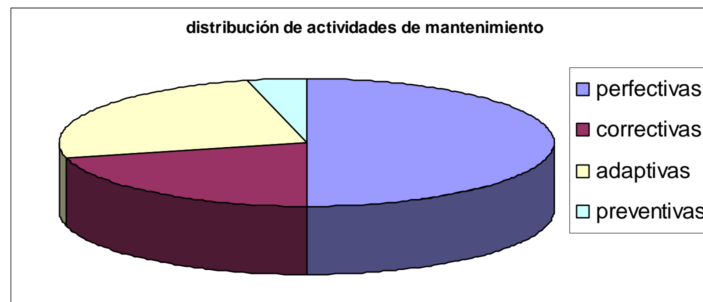



**Figura 3.** Distribución de actividades de mantenimiento

En el gráfico podemos observar claramente que las actividades perfectivas de mantenimiento son aquellas que suceden de forma más frecuente; lo que incentiva al desarrollo de método que garanticen una mejor relación costo-beneficio de estas actividades.

El paradigma de Modelamiento basado en el dominio lista entre sus ventajas principales la de ofrecer mayor flexibilidad en los sistemas, es decir que la correcta aplicación del mismo nos conduce a producir sistemas que son más fáciles de mantener.

Sabemos que el paradigma de programación Orientado a Objetos nos ha ofrecido también esta ventaja, ya que la reusabilidad, modularidad y encapsulamiento han sido sin duda alguna una gran ayuda para alcanzar este objetivo. A pesar de ello cuando un sistema es demasiado complejo, y tenemos una cantidad de objetos muy grande, la modularidad que nos había ofrecido este paradigma se vuelve deficiente. Es allí donde el paradigma MDD nos ofrece una mejora, ya que nos obliga a organizar los objetos en capas y particiones [3]. Modelar a nivel del dominio nos obliga a pensar acerca de la arquitectura y el modelo de objetos subyacentes de nuestro sistema, en lugar de que perdamos el foco del problema a resolver pensando en como codificaremos ese requerimiento que representa un desafío para nuestras habilidades de programación.

Dada la importancia que la flexibilidad tiene en la calidad de los sistemas de información actuales, esta es sin duda una de las piedras angulares del paradigma MDD.

Además de la flexibilidad en los sistemas desarrollados usando MDD, este paradigma ofrece las siguientes ventajas [10]:

- Incremento de portabilidad entre arquitecturas, middleware y plataformas de desarrollo
- Mejoras en la consistencia del Código
- Incremento en la Productividad (tiempo de desarrollo más rápido)

En cuanto al incremento de portabilidad, podemos mencionar que la misma ha sido la base para la creación de una de las arquitecturas más conocidas que soportan MDD, esta es MDA –por sus siglas en inglés Model Driven Architecture[1]-.

Según Richard Soley y el equipo de estrategia de la OMG [11], uno de los problemas principales que debe ser enfrentado por los que trabajan en TI en la actualidad, es la proliferación de ambientes middleware tales como CORBA, Enterprise Java Beans, message-oriented middleware, XML/SOAP, COM+ y .NET entre otros. Lo cierto es que a pesar de las ventajas o desventajas que pueda ofrecer cada uno de estos ambientes, la migración de un ambiente a otro siempre es una actividad costosa.

MDD propone la creación de modelos del dominio, que por definición esconden los detalles tecnológicos de la aplicación, y también propone la transformación de este modelo a uno específico para la plataforma, es decir que ahora tenemos un modelo "estable" sobre el podemos migrar de un ambiente a otro de una forma más ágil. Lo que se desea es la creación de una herramienta MDD, que pueda ser configurada para producir código para un ambiente particular. Actualmente tenemos ya herramientas que nos proveen de generación automática de código

1 MDA nació dentro de la OMG –Object Managment Group-, que es una organización preocupada por la creación de estándares dentro de la industria del software. Entre sus creaciones más conocidas están CORBA y UML.

que soportan métodos o arquitecturas basadas en MDD –Rational Rapid Developer de IBM, OptimalJ de Compuware-, y aunque aún no son capaces aún de generar programas completos a partir de un modelo específico, estudios, cuyos resultados citaremos más adelante, han demostrado que el código generado por estas herramientas hace uso eficiente de las prácticas de las técnicas de programación más conocidas. Es importante notar, que aunque estas herramientas de generación de código facilitan la portabilidad mucho más, aquellas que soporten MDD que no posean las herramientas de generación de código asociadas también proporcionan esta ventaja gracias a que el modelo conceptual se mantiene estable independientemente de la plataforma tecnológica que se utilice para desarrollar el sistema.

Con respecto a las mejoras en la consistencia del Código, estas en realidad también se derivan de la existencia de herramientas MDD automáticas de generación de código. El problema que se tiene cuando tenemos un equipo de trabajo con varios programadores, es que cada quien, como individuo que es, tiene su propio bagaje de conocimientos y experiencia. Así mientras uno de los programadores puede tener un estilo limpio de programación y hacer uso de patrones de diseño estandarizados, podemos tener otro que con menos experiencia en el campo, decida tratar de inventar lo que ya ha sido inventado. Entonces el uso de una herramienta automática de generación de código en base a un modelo, que debe por supuesto ser lo suficientemente inteligente como para proveer a nuestro código de patrones estandarizados, uso de componentes, etc., hará que al final tengamos un código más consistente. Y si vemos más allá, nos daremos cuenta de que la consistencia en el código nos da una gran ventaja desde la perspectiva de mantenimiento del sistema [10].

Lo citado anteriormente no quiere decir que usando MDD "mataremos" la creatividad de los desarrolladores, sino más bien que la creatividad de estos será elevada a un nivel que va más allá de cómo resolver un problema que ya fue resuelto por otro. Por supuesto lo expresado tampoco significa que ya no necesitaremos a los desarrolladores, después de todo el que tengamos una herramienta de generación de código automática no implica que no se tenga que realizar ajustes al código final, y sin la experiencia y conocimiento de los desarrolladores esto no se podría realizar.

Por último en lo que concierne al incremento en la productividad, que también está relacionado con la generación automática de código, debido a la importancia que conlleva para los gerentes de proyectos de TI, esta es sin duda la segunda ventaja más importante ofrecida para los usuarios de MDD.

Antes de que apareciera en la arena de la Ingeniería de Software el concepto de MDD, existían herramientas que generaban código a partir de especificaciones formales tales como un modelo de datos. La ventaja de las herramientas que pertenecen a alguna arquitectura o método que soporte MDD sobre las otras, es que estas permiten al usuario definir un dominio completo de una aplicación –entidades del negocio-, y luego traducen este modelo a código basado en especificaciones formales de la arquitectura tecnológica. Así el modelo conceptual podría tener por ejemplo una entidad que representa una orden de cliente; en la aplicación esa misma orden puede ser implementada como un objeto del negocio persistente llamado orden, un mensaje requiriendo esa orden, un JSP que muestra la orden, un servicio web –web service en inglés- que retorna la información de la orden al cliente, o todo lo anterior. Lo interesante es que la elección apropiada es realizada independientemente del modelo básico [12].

### 4.2 Resultados de estudios acerca del Modelamiento basado en el dominio

Ninguna ventaja ofrecida sería creíble sin la base de investigaciones realizadas que prueben la existencia real de las mismas; por ello esta parte del artículo está dedicada a citar resultados de investigaciones realizadas por diferentes grupos de trabajo a fin de demostrar que en efecto existen ventajas al usar alguna herramienta, método o arquitectura particular que soporta MDD.

El primer estudio que citaremos realiza una comparación de productividad en el desarrollo de una aplicación J2EE particular realizada por dos equipos de trabajo, uno que utilizó una herramienta de soporte para MDA y otro que utilizó una metodología centrada en el código soportada por un IDE tradicional [13].

Aquí la tabla I que muestra los datos cuantitativos generados por el estudio:

**Tabla I.** Comparación de Horas de Trabajo de equipos

| Equipo | No. de Horas de Trabajo Estimadas | No. de Horas de Trabajo realizadas |
|---|---|---|

| | | |
|---|---|---|
| Tradicional | 499 | 507,5 |
| MDA | 442 | 330 |

Como podemos observar en la Tabla I, los resultados del estudio demostraron que la productividad del equipo que trabajó con la arquitectura MDA se incrementó considerablemente respecto a la mostrada por el equipo que trabajó con la metodología tradicional.

Además del incremento en la productividad, en las conclusiones del estudio se citan como ventaja de MDA que el PIM es un modelo de larga vida, ya que usualmente las reglas básicas de un negocio no cambian con tanta frecuencia como la tecnología. Como resultado este modelo serviría como base para la comunicación entre desarrolladores que usen diferentes herramientas tecnológicas. La otra ventaja mencionada está relacionada con la calidad del código producido y de la aplicación misma. Esto porque en el equipo que usó la metodología tradicional se encontraron errores lógicos que necesitaron de las correcciones y por supuesto de pruebas extras a las ya realizadas.

El segundo estudio que citaremos, realizado por el mismo grupo de investigación, realiza una comparación de productividad de desarrollo de una aplicación entre un equipo de trabajo que usó RRD una herramienta que soporta ARAD –Architectured Rapid Application Development-, que es un tipo de aplicaciones que soporta MDD, y otro que utilizó una metodología centrada en el código soportada por un IDE tradicional [10]. El equipo que usó RRD como herramienta de desarrollo recibió las mismas especificaciones que se entregaron al equipo del estudio anterior, por lo que esta vez se midió la productividad del equipo RRD contra las generadas en el estudio anterior por el equipo que usó el IDE tradicional.

En la Tabla II se muestran los resultados cuantitativos del estudio:

**Tabla II.** Comparación de Horas de Trabajo de equipos

| Equipo | No. de Horas de Trabajo Estimadas | No. de Horas de Trabajo realizadas |
|---|---|---|
| Tradicional | 499 | 507,5 |
| RRD | 40-50 | 47,5 |

Los resultados de este estudio muestran una diferencia de productividad mucho más impactante que la del estudio anterior, lo que según el estudio, se debe en mucho a que la herramienta utilizada RRD posee una habilidad importante en la generación del código de la aplicación.

Un tercer estudio realizado, muestra una comparación de mantenibilidad entre dos aplicaciones desarrolladas por dos equipos de trabajo, uno usando una herramienta de soporte para MDA y otro usando un IDE tradicional [14].

Los resultados cuantitativos de este estudio, que se pueden observar en la tabla III, demostraron que para completar todos los cambios pedidos a los dos equipos, el que utilizó MDA registró una mejora de 37% en la productividad respecto al equipo que usó una arquitectura tradicional.

**Tabla III.** Horas de Trabajo de equipos

| Equipo | No. de Horas de Trabajo realizadas |
|---|---|
| Tradicional | 260,5 |
| MDA | 164,8 |

Los resultados del estudio van muy alineados con los del anterior, en el cual se observó un incremento del 35% de productividad en el desarrollo de una aplicación completamente nueva por parte del equipo que usó MDA. Lo expresado indica que MDA ofrece el mismo incremento de productividad tanto al desarrollar una aplicación nueva, como al darle mantenimiento.

Es importante acotar que como uno de los factores que, según el estudio, afectan a la productividad está la complejidad de la tarea. El equipo de investigación encontró que los beneficios obtenidos usando MDA son mayores mientras mayor es la complejidad del cambio a realizar.

Hay otros estudios que no han realizado tales comparaciones entre equipos de desarrollo que utilizan diferentes paradigmas; sin embargo los resultados de los mismos no dejan de ser importantes en cuanto a respaldar las ventajas que presenta el paradigma MDD. Así, podemos citar un caso de estudio usando MERODE -el cual es un método que se basa en MDD- en el que se definió un modelo del dominio para una empresa de telecomunicaciones que nacía dentro del mercado Holandés [15]. En dicho caso de estudio se mencionan como conclusiones del mismo que el modelamiento del dominio es definitivamente un proceso iterativo, ya que no

todo puede ser especificado desde el inicio, siempre a medida que iban avanzando con el estudio se iban encontrando nuevos requerimientos o restricciones que influían de una forma u otra al modelo anterior. Además se concluyó también que el uso del modelo del dominio permitió de una forma más limpia la integración de varias aplicaciones que utilizaban plataformas diferentes. Otra importante conclusión del estudio, es que al ser la empresa nueva en el mercado, tenía deficiencias en las especificaciones de sus propios procesos, y que el hecho de realizar el análisis del problema usando modelamiento del dominio ayudó a que se establezcan procesos formales dentro del negocio en sí.

### 4.3 Desventajas

Dado que MDD es un paradigma relativamente nuevo, la mayoría de las herramientas disponibles en el mercado que permiten el paso de un modelo formal al código de forma automática son muy costosas. Sin embargo, vale la pena recalcar que la ventaja que estas proveen en cuanto a productividad puede hacer que la inversión puede ser recuperada con facilidad si se la usa de forma correcta.

En el mercado podemos encontrar herramientas gratis de modelamiento. El problema que puede surgir aquí es que el modelo creado no permanezca en sincronización con el código.

Algunos detractores de MDD mencionan que UML, uno de los principales lenguajes de modelamiento, aún presentan deficiencia en ciertos aspectos tales como la semántica de los casos de uso [16] entre otros, por lo que mal estaría pasar directamente de un modelo UML al código. Sin embargo podemos decir que cada vez se sigue trabajando en cuanto a esto, en talleres y conferencias que analizan en detalle las posibles soluciones a problemas que presenta este lenguaje; y también en algunos casos los métodos crean sus propios lenguajes, como es el caso de MERODE, de modelamiento basándose en especificaciones estrictamente formales que le dan la consistencia necesaria al modelo generado.

Otra cosa que es importante anotar, es que el tener un buen método de Modelamiento no nos asegura tener un buen modelo final. La construcción del modelo siempre descansa sobre las personas que analizan el problema, por lo cual si el modelo construido no corresponde a la realidad, si nuestro código fue generado a partir del modelo, esto puede significar el fracaso del proyecto.

## 5. Conclusiones

En base a los estudios realizados, podemos decir que el Modelamiento basado en el Dominio, correctamente utilizado y con la ayuda de las herramientas apropiadas, brinda un incremento importante en la productividad de las empresas desarrolladoras de software.

También es evidente por los resultados y teorías ampliamente aceptadas, que aún sin las herramientas que nos permitirían el paso del modelo al código; se gana muchísimo en cuanto a calidad cuando basamos nuestro desarrollo en un modelo bien definido del sistema.

El modelamiento basado en el Dominio, es un paradigma relativamente joven; aún quedan ciertos aspectos, como la definición de lenguajes de modelamiento formales y estandarizados, que deben ser cubiertos por los investigadores; pero es seguro que gracias a las ventajas y beneficios que esto podría significar para las empresas, los estudios en esta área seguirán surgiendo.

**Referencias**


**[1]** G. Koutsoukos, L. Andrade, J.L. Fiadeiro, J. Gouveia, "Architectural concerns and use of a model-driven development framework", ATX Software S.A., http://www.softmetaware.com/oopsla2002/koutsoukosg.pdf

**[2]** Eric Evans, "Domain-Driven Design: Tackling complexity in the heart of software", Addison Wesley, 2004, pp. xxi, 2

**[3]** M. Snoeck, G. Dedene, M. Verhelst, A. Depuydt, "Object-Oriented Enterprise Modeling with MERODE", Leuven University Press, 1999, pp. 12, 14

**[4]** M. Snoeck, C. Michiels, G. Dedene, "Consistency by construction: the case of MERODE", ER 2003 Workshops ECOMO, IWCMQ, AOIS, and XSDM, Chicago, IL, USA, October 13, 2003, Proceedings, 2003 XVI, 410 p., Lecture Notes in Computer Science, Volume 2814, pp.105-117, http://www.econ.kuleuven.ac.be/public/ndbaa30/default.htm

**[5]** Stephen J. Mellor, Anthony N. Clark, Takao Futagami, "Model-Driven Development", IEEE Software, 2003, http://www.computer.org/software/homepage/2003/s5gei_print.htm

**[6]** Bertrand Meyer, "Object Oriented Software Construction", Prentice Hall PTR, 1997, pp. 904, 3-19

**[7]** Scott Ambler, "Debunking Modeling Myths", Software Development, Agosto 2001, www.sdmagazine.com

**[8]** F. Ruiz, M. Polo, "Mantenimiento del Software", Grupo Alarcos, Departamento de Informática de la Universidad de Castilla-La Mancha, 2001, http://alarcos.inf-cr.uclm.es/doc/mso/slides/s9.pdf

**[9]** B. P. Lientz, E. B. Swanson, "Software Maintenance Management". Addison-Wesley: Reading, MA. 1980.

**[10]** The Middleware Company, "Model Driven Development for J2EE with IBM Rational Rapid Developer (RRD): Productivity Analysis", Noviembre 2003, http://www.MiddlewareRESEARCH.com

**[11]** R. Soley, Equipo de estrategia del grupo OMG, "Model Driven Architecture", Noviembre 2000, http://www.omg.org/mda/mda_files/model_driven_architecture.htm

**[12]** The Middleware Company, "Model Driven J2EE Development: comparing two approaches and tools. Productivity Analysis", Noviembre 2003, http://www.MiddlewareRESEARCH.com

**[13]** The Middleware Company, "Model Driven Development for J2EE Utilizing a Model Driven Architecture Approach: Productivity Analysis", June 2003, http://www.MiddlewareRESEARCH.com

**[14]** The Middleware Company, "Model Driven Development for J2EE Utilizing a Model Driven Architecture Approach: Maintainability Analysis", June 2003, http://www.MiddlewareRESEARCH.com

**[15]** M. Snoeck, C. Michiels, "Experiences with domain modeling and the co-design of Business Rules in the Telecommunication Business Area", 2003 http://www.econ.kuleuven.ac.be/public/ndbaa30/default.htm

**[16]** Dave Thomas, "Revenge of the Modelers or UML Utopia", IEEE Software, Mayo/Junio 2004