# Tema 5. Desarrollo dirigido por modelos

## Desarrollo de Software
## Curso 2023-2024
## 3º - Grado Ingeniería Informática

Dpto. Lenguajes y Sistemas Informáticos
ETSIIT
Universidad de Granada

15 de mayo de 2024


# Tema 5. Desarrollo dirigido por modelos


5.1 Fundamentos del desarrollo dirigido por modelos (MDD)
- 5.1.1 Introducción a la Ingeniería Dirigida por Modelos
- 5.1.2 Modelos y metamodelos
- 5.1.3 Lenguaje de modelado (ML)
- 5.1.4 Clasificación de un lenguaje de modelado
- 5.1.5 Productos software, plataformas y transformaciones
- 5.1.6 De enfoques dirigidos por modelos abstractos a concretos

5.2 Arquitectura dirigida por modelos (MDA)


- Un modelo software representa un aspecto o vista del sistema usando un lenguaje (a menudo gráfico) para describirlo.
- Por un lado, son utilizados como forma de tener una imagen común del sistema a desarrollar, compartida por todos los interesados en el sistema y que permita estar seguros de que se desarrolla lo que de verdad cada uno imagina.
- Pero lo más importante es que existe una propuesta de desarrollo software, el enfoque MDE (del inglés Model-driven Engineering), que saca mucho más partido de los modelos. En concreto a partir de ellos se pueden crear y ejecutar de forma automática sistemas software, a partir de técnicas complejas tales como meta-modelado, transformación de modelos, generación de código e interpretación de modelos. Algunos ejemplos son:
  - Model Driven Architecture (MDA), definida por el Object Management Group (OMG)
  - Factorías software, que unen al uso de modelos el uso de patrones, marcos de trabajo y herramientas
  - Ingeniería DSL, del inglés Domain Specific Languages

**Modelo**

Un modelo en una abstracción de un sistema bajo estudio

El modelo es en sí también un sistema, con su propia identidad, complejidad, elementos, relaciones, etc. Si pensamos en un modelo sobre un modelo, debemos considerar que el segundo actúa como sistema bajo estudio y que, por tanto, es un sistema.

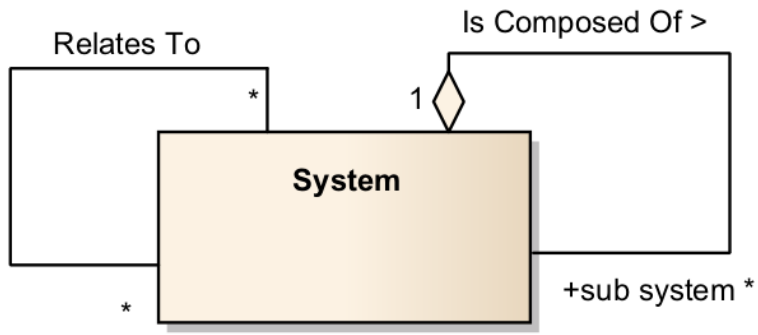


Figura 1: Definición de sistema. [Fuente:[1]]

[1] **Silva.2015**.

**DEFINICIÓN: Modelo**

Sistema que ayuda a definir y dar respuesta sobre el sistema bajo estudio sin necesidad de tener que considerarlo directamente

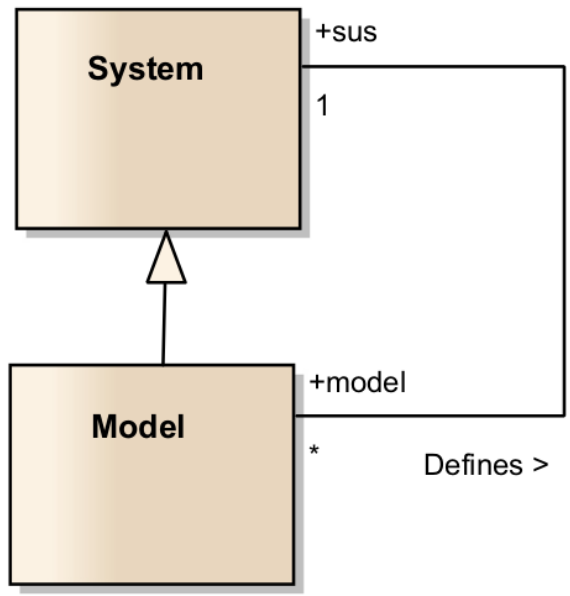


Figura 2: Definición de modelo y su relación con el sistema que describe. [Fuente:[2]]

[2] **Silva.2015**.

Tres criterios para distinguir un modelo software de cualquier otro artefacto:

- Criterio de mapeo: Debe ser posible identificar en la realidad (si existe) cada objeto representado en el modelo
- Criterio de reducción: El modelo simplifica la realidad, destacando solo algunos aspectos de ella que interesan (también pueden amplificarse otros)
- Criterio de pragmatismo: El modelo debe ser útil para algún propósito, por ejemplo, para Booch los modelos sirven para:
  - visualizar un sistema, como es o queremos que sea
  - especificar la estructura y el comportamiento de un sistema
  - funcionar como plantilla que guía el proceso de desarrollo; y
  - ayudar a documentar las decisiones tomadas a lo largo del ciclo de vida de un proyecto

**DEFINICIÓN: metamodelo**

Modelo que define la estructura de un Modeling Language (ML).

1. Relación *ElementOf*: Un ML es un conjunto de modelos (o un modelo es un elemento de un ML)
2. Relación *Defines*: Un metamodelo es un modelo de la estructura de un ML (o un ML es definido por un metamodelo)
3. Un metamodelo es un modelo de un conjunto de modelos o es un modelo de modelos
4. Relación *ConformsWith*: Un modelo se ajusta a un metamodelo, i.e. debe satisfacer las reglas definidas al nivel de su metamodelo

## 5.1.2. Modelos y metamodelos
### Metamodelo

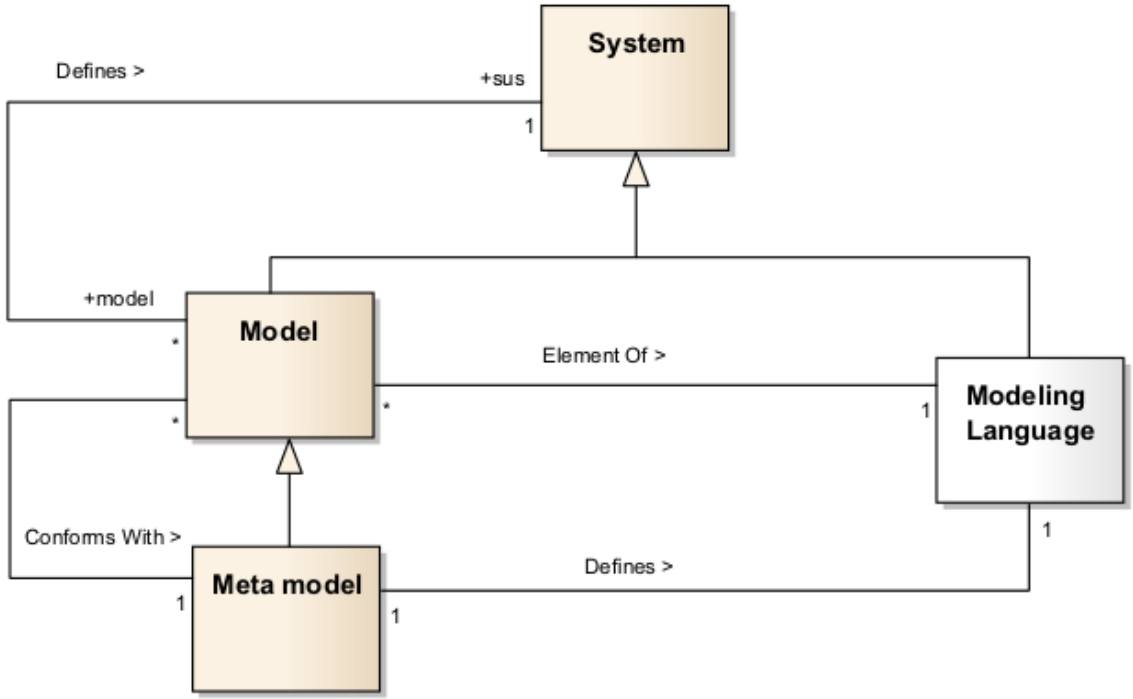


Figura 3: Definición de metamodelo y su relación con el modelo. [Fuente:[3]]

[3] **Silva.2015**.

## 5.1.2. Modelos y metamodelos

### Meta-metamodelo, metamodelo y modelo

- En el campo de los ML: un ejemplo es el estándar Meta Object Facility (MOF), el núcleo de MDA (de OMG), el cual asigna tipos (e interfaces para usarlos), a las distintas entidades de una arquitectura. MOF se define a sí mismo usando MOF. MOF está formado por una arquitectura de cuatro capas, capas que se describen a continuación de la más alta a la más baja (ver Figura 4):
  - M3: Un meta-metamodelo, el lenguaje que usa MOF para especificar metamodelos (o modelos M2). MOF se crea a sí mismo instanciándose de su propio modelo
  - M2: Metamodelos, instanciados a partir del meta-metamodelo M3, siendo cada elemento del metamodelo una instancia de un elemento definido en el meta-metamodelo M3. Un ejemplo de metamodelo (es decir, una instancia de MOF) es UML, y otro es Common Warehouse Metamodel (CWM)
  - M1: Modelos, definidos según los intereses y necesidades de sus usuarios (distintos dominios de aplicación, distintos niveles de abstracción) y descritos mediante los metamodelos en M2. Ejemplos sería todos los modelos escritos en UML. Un modelo de usuario puede contener tanto elementos de modelo (como clases –Video, por ejemplo–) o instancias de las clases –instance:Video, por ejemplo–.
  - M0: Datos, que describen los objetos que existen en un entorno de cómputo concreto o incluso en el mundo real (un Video concreto en

## 5.1.2. Modelos y metamodelos

### Meta-metamodelo, metamodelo y modelo

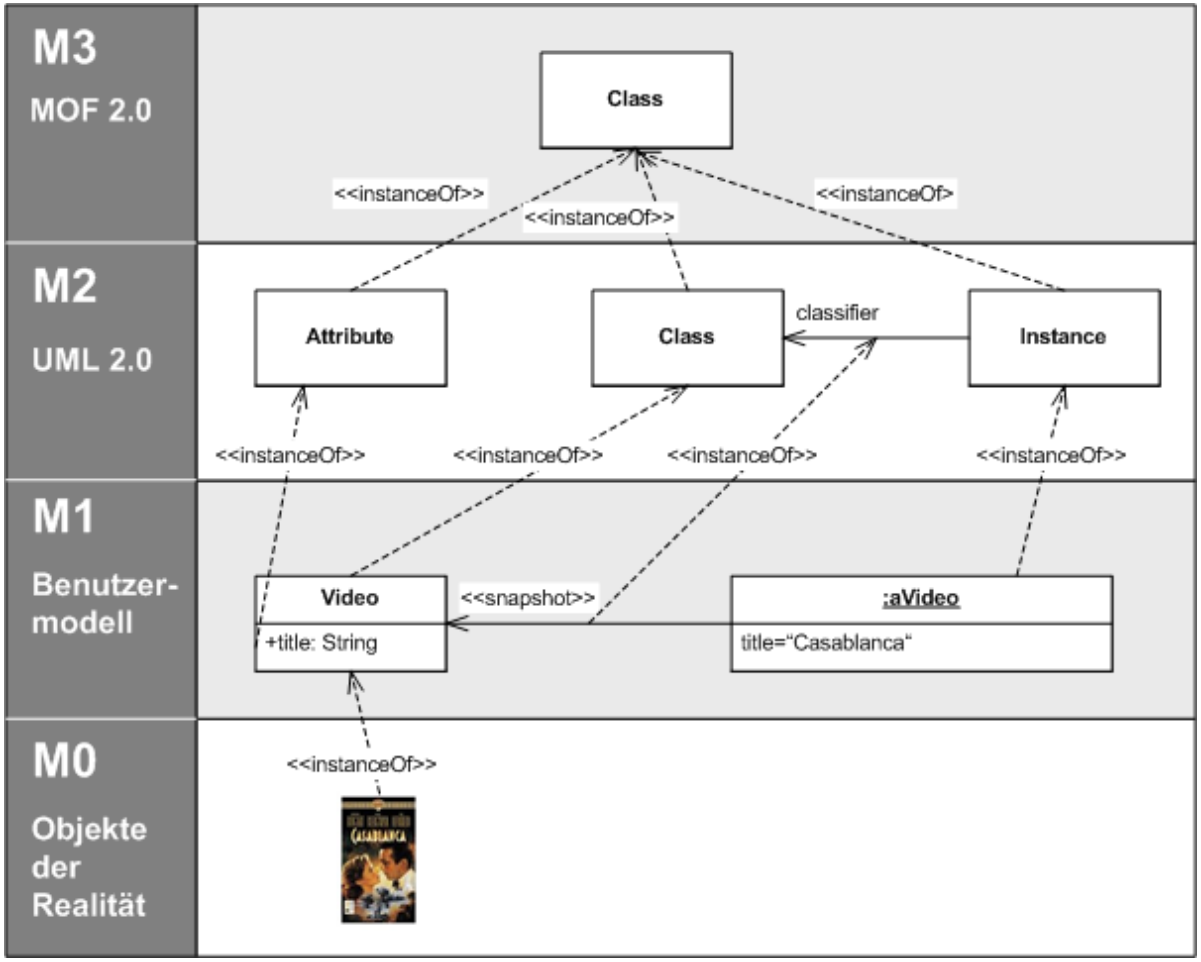


Figura 4: Un ejemplo de la jerarquía de 4 niveles de MOF.

## 5.1.2. Modelos y metamodelos

### Meta-metamodelo, metamodelo y modelo

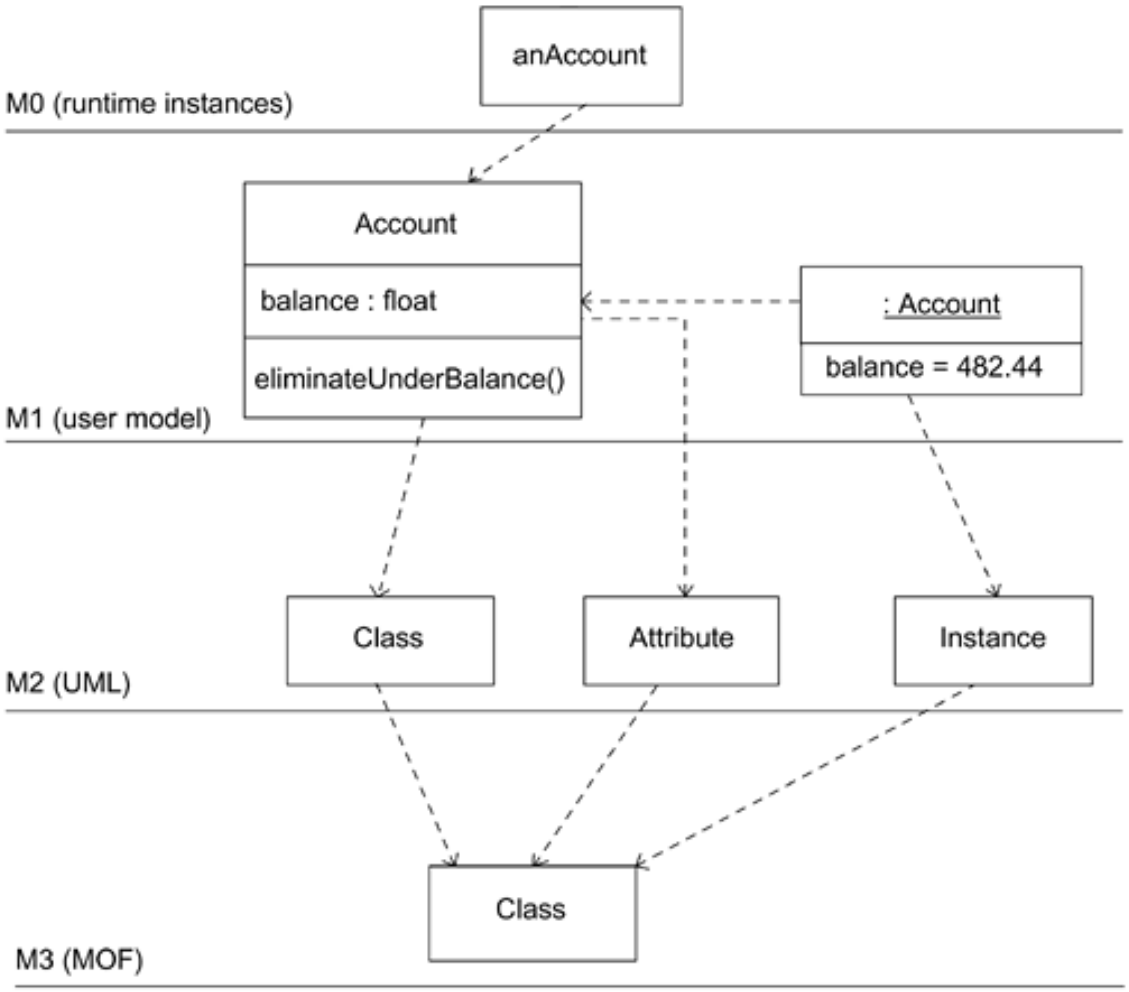


Figura 5: Otro ejemplo de la jerarquía de 4 niveles de MOF. [Fuente:[4]]

[4] **Mellor.zz.2004**.

## ¿Por qué usar metamodelos?

- Especificación del lenguaje: Todas las partes interesadas pueden entender un modelo y rebatirlo si el significado de cada elemento del modelo está bien especificado.
- Comunicación acerca de los modelos: Se simplifica la comunicación entre las partes interesadas.
- Funciones de mapeo: Facilita pasar de un modelo a otro.

## 5.1.3. Lenguaje de modelado (ML)

**DEFINICIÓN 2: Lenguaje de modelado ML**

ML(metamodelo: *m*, notación *n*, semántica: *s*, pragmática: *p*): Conjunto de todos los posibles modelos que se ajustan a *m*, se representan por *n*, satisfacen *s* y tienen en *p* una guía de la forma de uso más apropiada.

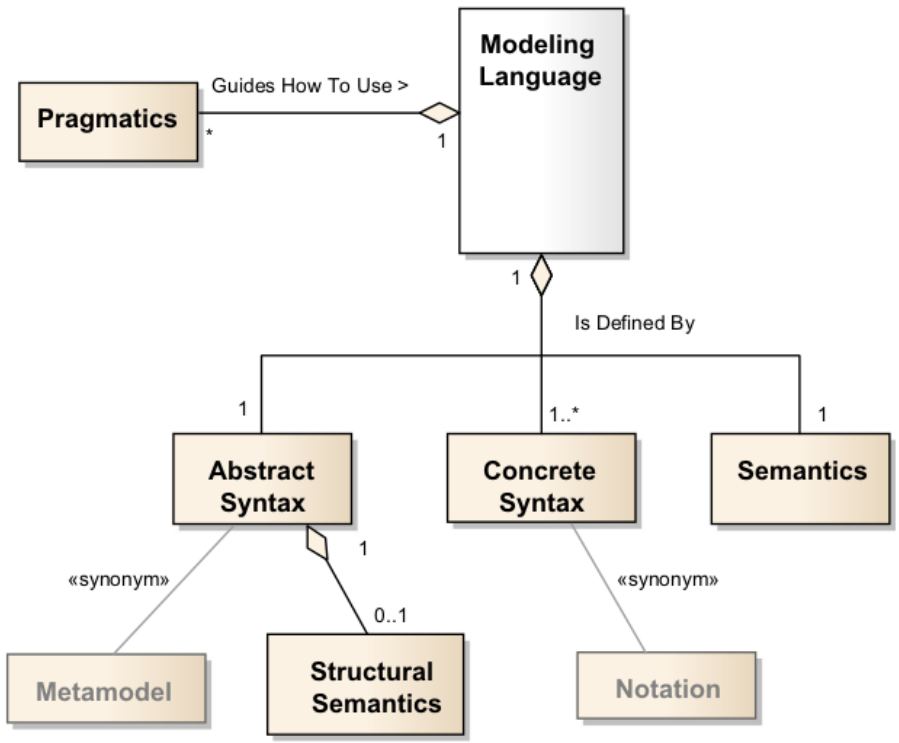


Figura 6: Definición de lenguaje de modelado. [Fuente:[5]]

[5] **Silva.2015**.

Según el dominio de la aplicación:

- GPML: de uso general (del inglés, General Purpose Modeling Language). Tienen un número mayor de constructos o conceptos generales, de forma que invitan a un uso más amplio en diferentes campos de aplicación. Ej. UML, SysML[6].
- D(S)ML: de uso específico (del inglés, Domain-Specific Modeling Language). Usan un número más reducido de constructos y además más relacionados con el campo específico de su dominio de aplicación. Esa mayor especificidad los hace más fáciles de entender y, por tanto, de validar. Sin embargo también tiene mayor costo el que sea específico de un dominio, pues un lenguaje tiene que aprenderse, implementarse, mantenerse, y aprender las herramientas que permiten desarrollo de software usando dicho lenguaje.

[6] SysML es un lenguaje usado para generar documentación en sistemas software OO.

## 5.1.4. Lenguaje de modelado (ML)

### Puntos de vista como formas de estructuración de un lenguaje de modelado

Los lenguajes de modelado pueden estructurarse en base a distintos puntos de vista, que se clasifican en base a dos criterios:

- Nivel de abstracción (terminología MDA):
  - Punto de vista independiente de la computación: Computation Independent Model (CIM)
  - Punto de vista independiente de la plataforma: Platform Independent Model (PIM)
  - Punto de vista específico de una plataforma: Platform Specific Model (PSM)
  - Punto de vista múltiple: incluye distintos niveles de abstracción (CIM, PIM, PSM)
- Perspectiva o dimensión funcional:
  - Punto de vista estático: el lenguaje permite describir el sistema desde una perspectiva funcional estructural (clases, objetos, nodos, bloques, relaciones entre ellos). Ej. diagrama de clases UML o diagrama de componentes
  - Punto de vista dinámico: el lenguaje permite describir la funcionalidad del sistema en base al comportamiento del mismo (tareas, operaciones, estados, eventos, mensajes y sus relaciones). Ej. diagramas de actividad o máquinas de transición de estados en UML; diagramas de procesos de negocio en Business Process Model & Notation (BPMN)

**DEFINICIÓN: Producto software**

Sistema compuesto de la integración no trivial de

- plataformas software,
- artefactos (código) generados por transformaciones modelo-a-texto,
- artefactos (código) escritos directamente por desarrolladores,
- eventualmente, modelos directamente ejecutables en una plataforma software particular.

## Plataformas software

- Una plataforma software es un conjunto integrado de elementos computacionales que permiten el desarrollo y ejecución de una clase de productos sofware.
- Por lo general, estos elementos proporcionan diferentes funcionalidades a través de la reutilización y mecanismos de extensibilidad y se refieren a tecnologías como middleware, librerías software, marcos de aplicaciones, y componentes de software, pero también sistemas de gestión de bases de datos, servidores Web, gestores de contenidos, sistemas de gestión de documentos, sistemas de gestión de flujo de trabajo, etc.

## Artefactos

- Los artefactos, tanto generados de forma automática desde el modelo como escritos por desarrolladores, pueden ser relevantes durante el tiempo de desarrollo, durante el tiempo de ejecución o en ambas fases del ciclo de vida del producto software.
- En todo caso, todos dependen estrechamente de las plataformas involucradas.
- Algunos ejemplos son: archivos de código fuente y binario, scripts de configuración e implementación, scripts de bases de datos e incluso archivos de documentación, incluidos los propios modelos.

## Transformaciones

En el enfoque MDE, se consideran dos tipos de transformaciones:

- Transformaciones de modelo a texto (M2T, del inglés Model To Text).- Producen artefactos software textuales –generalmente código, XML y otros ficheros de texto– a partir de modelos. La técnica más común dentro de la transformación M2T es la de generación de código
- Transformaciones de modelo a modelo (M2M, del ingles Model to Model).- Transforman un modelo en otro, siendo generalmente el segundo más cercano al dominio de la solución o satisfaciendo las necesidades de alguna parte interesada. Estas transformaciones se especifican a través de distintos lenguajes, como lenguajes de programación, pero también por lenguajes especializados de transformación de modelos, para diferentes propósitos y con diferentes paradigmas de modelado (algunos ejemplos son QVT, Acceleo, ATL, VIATRA y DSLTrans)

## Modelos

- Los modelos se utilizan para producir artefactos de software.
- En algunas situaciones particulares, los modelos pueden ser directamente interpretados y ejecutados por plataformas específicas integradas con el producto software. Para ello deben definirse de manera consistente y rigurosa. En general, se requiere un cierto nivel de calidad para que los modelos se puedan usar correctamente en transformaciones M2M o M2T.

## 5.1.6. De enfoques dirigidos por modelos abstractos a concretos

Los enfoques MDE pueden clasificarse en tres tipos según el nivel de abstracción:

- Enfoques de alto nivel
- Enfoques de nivel medio: meta-herramientas dirigidas por modelos
- Enfoques concretos: herramientas dirigidas por modelos

## 5.1.6. De enfoques dirigidos por modelos abstractos a concretos

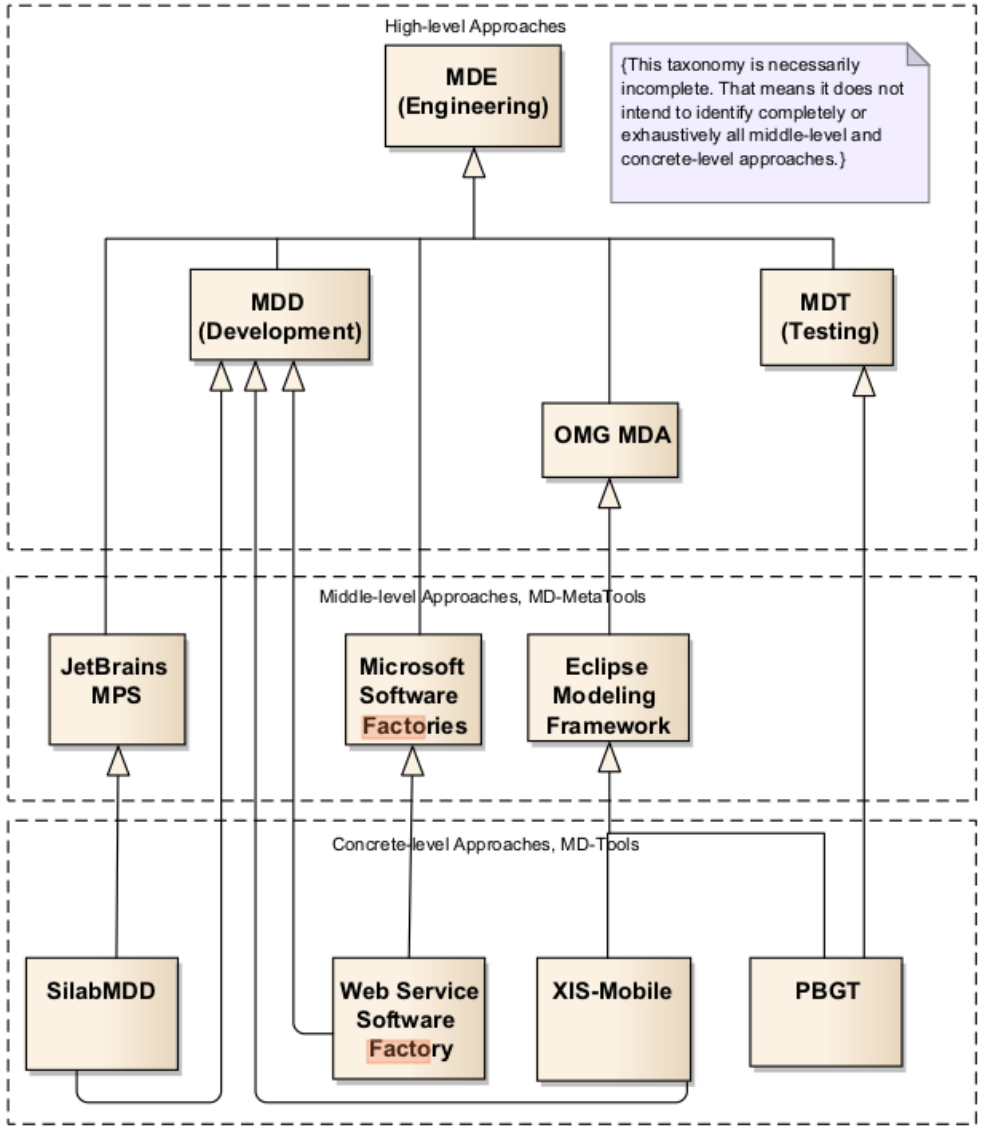


Figura 7: MDE y distintos niveles de cocreción. [Fuente:[7]]

[7] **Silva.2015**.

## 5.1.6. De enfoques dirigidos por modelos abstractos a concretos
## Enfoques de alto nivel

A este nivel se muestran tres tipos de MDE:

- Orientados al desarrollo (MDD Model Drivel Development).- Se enfocan en las tareas de especificación de requisitos, análisis, diseño e implementación. Las concreciones de MDD suelen proporcionar lenguajes para modelar con distintos niveles de abstracción y transformaciones M2M and M2T par mejorar tanto productividad en el desarrollo como calidad en el producto software generado
- Orientados a la prueba (MDT o MBT Model Driven/Based Testing).- Se enfocan en la automatización de las pruebas. Las concreciones MDT suelen representar el comportamiento que se desea del sistema, las estrategias de prueba y el entorno de prueba. Los casos de prueba generados están al mismo nivel de abstracción que el modelo y podrán convertirse en tests ejecutables que conectan con el sistema usando unas herramientas de prueba y marcos de trabajo concretos
- MDA, de OMG.- Enfocado principalmente en la definición de modelos y sus transformaciones. Admite la definición de modelos en diferentes niveles de abstracción (CIM, PIM, PSM). Tipos de transformaciones: M2M (CIM-CIM, CIM-PIM, PIM-PIM, PIM-PSM y PSM-PSM), M2T (PSM-text). Varias especificaciones concretas

## Enfoques de nivel medio (MD MetaTools)

- Eclipse Modeling Project (EMP).- Se centra en la evolución y promoción de tecnologías de desarrollo MD dentro de la comunidad Eclipse, proporcionando un conjunto integrado de marcos y herramientas de modelado extensibles e implementaciones de estándares. Varias herramientas y marcos, la mayoría populares, relativamente fáciles de usar y mantener y con soporte comunitario abierto y fuerte.
- Microsoft Software Factories.- Se inspira en la metáfora que hace de las "línea de montaje" en áreas de automatización industrial de las fábricas.

**DEFINICIÓN: Fábrica de software**

Colección estructurada de activos software relacionados, tales como procesos, DSLs, plantillas, IDEs, configuraciones y vistas, que son utilizados para crear tipos específicos de software

- JetBrains Meta Programming System (MPS).- De código abierto, es un workbench de proyección para lenguajes, lo que significa que no existen ni gramáticas ni parsers, sino que se cambia directamente el árbol subyacente de sintaxis abstracta, mostrado como texto, con un simple editor.

- XIS-Mobile (del inglés, eXtensible Interactive Systems): para aumentar la productividad del desarrollo de aplicaciones móviles multiplataforma. Su DSL se define como perfil UML, con una arquitectura de múltiples vistas que permite un enfoque básico de diseño y otro inteligente. Compuesto por cuatro componentes principales, este marco sugiere desarrollar una aplicación móvil en cuatro pasos siempre que sea posible:
  1. Definir las vistas requeridas usando el editor gráfico (Visual Editor),
  2. validarlas con el validador de modelos,
  3. generar los modelos de las vistas de las interfaces de usuario, usando el generador de modelos y
  4. generar el código fuente de la aplicación a través del generador de código.
- WebService Software Factory, un ejemplo de las fábricas de software de Microsoft y un enfoque MDD concreto que proporciona una colección integrada de recursos diseñados para construir servicios Web de forma rápida y consistente, que se adhieran a patrones de arquitectura y diseño bien conocidos.

- SilabMDD, enfocado especialmente en la especificación de requisitos. Incluye el lenguaje SilabReq, que impone una definición rigurosa de especificación de casos de uso, basada en la descripción de secuencias de acciones, pre- y post-condiciones, y las relaciones entre los casos de uso y los elementos definidos en los modelos de dominio del sistema (especificados de forma textual).
- GTGT (Pattern Based GUI Testing): MBT que proporciona estrategias de prueba genéricas, basadas en patrones de prueba de interfaz de usuario.

- El proceso iterativo a nivel práctico se reduce de forma significativa con respecto al teórico: ante fallos, los programadores, que conocen bien el código, se olvidan de la documentación (en forma de diagramas y texto) y cambian directamente el código
- En la fase de explotación del sistema, suele cambiar el equipo de mantenimiento y la documentación está incompleta
- Para los desarrolladores resulta siempre tediosa la elaboración de la documentación, mucho más si se hace con mucho detalle, con una Descripción Arquitectónica (DA) elaborada teniendo en cuenta los puntos de vista de los distintos interesados y las distintas perspectivas para asegurar la calidad

## Ciclo de vida tradicional

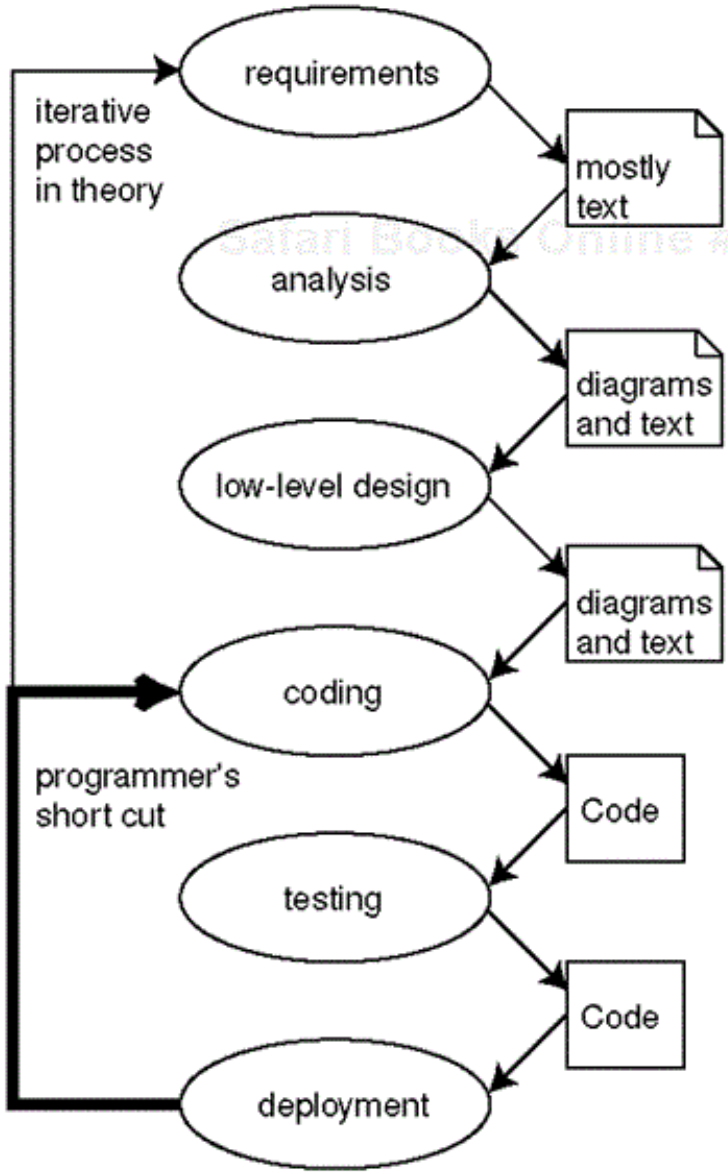


Figura 8: El ciclo de vida software tradicional: teoría y práctica. [Fuente: [8]]

- Las metodologías ágiles (Scrum o Extreme Programming (XP)), conscientes de esta realidad, la resuelven de una forma "extrema", de manera que deciden prescindir en la mayor medida posible de las fases iniciales y la documentación generada en ellas y enfocarse sobre todo en las fases de codificación y prueba, para lo que solo usan diagramas de diseño (de bajo nivel).
- La solución que proponen en el caso de que cambie el equipo de desarrollo o mantenimiento del producto software para no "perderse en el código" es la de dejar "marcadores" para los que vengan detrás, que finalmente son también texto y diagramas de más alto nivel.[9]

[9] **Warmer.zz.2003**.

- En ninguno de los enfoques se puede prescindir realmente de la documentación del alto nivel
- No es posible ser siempre realmente productivos, en el sentido de construir algo ejecutable, como es el código
- El enfoque MD plantea una alternativa distinta, en la que desde las primeras fases del ciclo de vida lo que se construye (modelos, expresados por texto y diagramas, principalmente) puedan ya ser productivos, en el sentido de que puedan ser ejecutados

# Fin Tema 5

Se espera que los estudiantes comprendan los conceptos de modelado y metamodelado, utilizando UML para representar sistemas de software desde puntos de vista estáticos y dinámicos. Deben conocer y saber reconocer herramientas de transformación de modelos (M2M y M2T) y su importancia en la automatización del desarrollo de software. Además, deben entender y aplicar los principios de Model-Driven Architecture (MDA) para asegurar la portabilidad e interoperabilidad entre plataformas. En resumen, se espera que utilicen técnicas de Ingeniería Dirigida por Modelos (MDE) para diseñar, desarrollar y mantener sistemas de software eficientes y adaptables, con una sólida base teórica y práctica en estas metodologías.