# Modelado Ágil


Trabajo realizado por:

Fernando Calvillo Parejo
Jerónimo Chaves Caballero
Javier Gómez Luzón
Elena María Gómez Ríos
Javier Oliva Cruz

# Índice:

# 1. Introducción

El modelado ágil, en inglés Agile Modeling (AM), es un nuevo enfoque para realizar actividades de modelado que fue introducido en el 2002 por Scott W. Ambler.

Como definición, es una metodología basada en la práctica para modelado efectivo de sistemas de software, siendo una colección de prácticas, guiadas por principios y valores que pueden ser aplicados por profesionales de software en el día a día. AM no es un proceso prescriptivo, ni define procedimientos detallados de como crear un tipo de modelo dado. En lugar de eso, sugiere prácticas para ser un modelador efectivo.

Es "*suave al tacto*" , no es duro y es rápido - piense en AM como un arte, no una ciencia.

AM tiene tres objetivos:

1. Definir y mostrar como poner en práctica una colección de valores, principios y prácticas que conlleven a un modelado ligero efectivo.

2. Explorar la aplicación de técnicas de modelado en proyectos de software a través de un enfoque ágil, tal como XP, DSDM o SCRUM.

3. Explorar el cómo mejorar el modelado bajo procesos prescriptivos, tales como el Proceso Racional Unificado (RUP), o el Proceso Unificado Empresarial (EUP).

# 2. Valores

Los cinco valores del modelado ágil (AM) son:

1. **Comunicación**: Los modelos promueven la comunicación entre su equipo y las partes interesadas de su proyecto, así como entre los desarrolladores de su equipo.

2. **Sencillez**: Es importante que los desarrolladores entiendan que los modelos son fundamentales para simplificar tanto el software como el proceso del software; es mucho más fácil explorar una idea y mejorarla a medida que aumenta su comprensión, dibujando un diagrama o dos en lugar de escribir decenas o incluso cientos de líneas. de código.

3. **Retroalimentación**: La retroalimentación es vital para que los equipos de desarrollo ágil comprendan si el equipo va en la dirección que se espera. El desarrollo del producto por parte del equipo ágil estará sujeto a revisión por el propietario del producto, y cualquier feedback se podrá adjuntar al product backlog.

4. **Valor**: El valor es importante porque será necesario tomar decisiones importantes y ser capaz de cambiar de dirección descartando o refactorizando su trabajo cuando algunas de las decisiones resultan inadecuadas.

5. **Humildad**: Los mejores desarrolladores tienen la humildad de reconocer que no saben todo, que sus compañeros desarrolladores, sus clientes y, de hecho, todas las partes interesadas del proyecto también tienen sus propias áreas de experiencia y tienen valor para agregar a un proyecto. Un enfoque eficaz es asumir que todos los involucrados en su proyecto tienen el mismo valor y, por lo tanto, deben ser tratados con respeto.

# 3. Principios

Los principios o fundamentos están basados en los valores anteriormente citados. Se deben de cumplir si queremos desarrollar software de forma ágil.

Los principios se dividen en dos tipos:

- **Importantes**: Son los principales y que deben de cumplirse en todo el proceso de modelado.

- **Adicionales**: No son obligatorios seguirlos, pero suponen un mejor proceso de modelado al incluirlos.

## 3.1. Principios importantes

- **Adoptar simplicidad**: Cuando estás desarrollando debes de asumir que la solución más simple es la mejor solución.

- **Aceptar los cambios**: La percepción de los requisitos cambia durante el tiempo.

- **Llegar a la siguiente fase es un objetivo secundario**: Debemos de pensar antes en un proyecto consistente y robusto que realizar todas las tareas para la fecha límite.

- **Cambio incremental**: Debemos de partir de modelos básicos e ir incrementando su complejidad a medida que avanzamos en el proyecto.

- **Maximizar la inversión del cliente**: Tenemos que tener siempre en cuenta que es lo prioritario para el cliente.

- **Modelar con un propósito**: Ten claro qué y para quién estás modelando.

- **Múltiples modelos**: Necesitamos utilizar múltiples modelos para desarrollar software porque cada modelo describe un único aspecto de nuestro software.

- **Calidad del trabajo**: Lo que realicemos debemos de hacerlo de la mejor manera posible para que sea más fácil de entender y de actualizar.

- **Retroalimentación rápida**: Debemos de comunicar al cliente de manera rápida todo avance que realicemos para obtener un feedback necesario para corregir posibles errores que hayamos cometido lo antes posible.

- **El software es el principal objetivo**: Todos queremos tener terminado un producto de calidad aunque tardemos en ello.

- **Viajar ligero**: Todos los artefactos y documentación que hagamos debemos de mantenerlos. Por tanto, es importante tener el número justo y necesario de artefactos.

### 3.2. Principios adicionales

- **El contenido es más importante que la representación**: da igual el tipo de método de representación, lo importante es tener un contenido claro y comprensible.

- **Comunicación honesta y abierta**: Esto ayuda a que el equipo pueda proponer sugerencias y mejoras sin miedo a represalias.

### 3.3. Principios adicionales obsoletos.

Los siguientes principios se declararon como obsoletos en enero de 2005 por las siguientes razones:

- **Todos podemos aprender de todos**: Es una gran idea, pero es muy general y no es necesario para una metodología de modelado.

- **Conoce tus modelos**: No es necesario ser tan explícito.

- **Conoce tus herramientas**: El mismo problema que el elemento anterior.

- **Adaptación local**: Aunque sea una buena idea, no es algo que se de en el modelado, sino en el proceso general de producción de software.

- **Trabajar con el instinto de la gente**: El mismo problema de “Todos podemos aprender de todos”.

# 4. Prácticas

Las prácticas en la metodología ágil están organizadas en dos listas las prácticas principales y las suplementarias. Las prácticas principales se deben adoptar para poder afirmar que realmente se está adoptando un desarrollo basado en modelos ágiles. Las prácticas suplementarias se deben considerar para cumplir con las necesidades del entorno.

## 4.1. Prácticas Principales

- **Participación activa del cliente**: Se puede promover fácilmente la participación activa de los interesados en tus proyectos si adopta técnicas de modelado inclusivo.

- **Aplicar los artefactos adecuados**: Cada artefacto tiene sus propias aplicaciones específicas. Se necesita conocer las fortalezas y debilidades de cada tipo de artefacto para saber cuándo usarlos y cuándo no.

- **Propiedad colectiva**: Todos pueden trabajar en cualquier modelo y artefacto en el proyecto, si es necesario.

- **Información en un único sitio**: La información debe de almacenarse en un único sitio y solamente ahí.

- **Crear varios modelos en paralelo**: Debido a que cada tipo de modelo tiene sus fortalezas y debilidades, ningún modelo es suficiente para tus necesidades de modelado.

- **Simplicidad de contenido**: Debe mantener el contenido real de sus modelos (sus requisitos, su análisis, su arquitectura o su diseño) tan simple como sea posible y al mismo tiempo satisfacer las necesidades de los interesados de tu proyecto. No debe agregar aspectos adicionales a tus modelos a menos que sean justificables.

- **Describir los modelos de forma simple**: Cuando consideramos los posibles diagramas que podríamos aplicar (diagramas UML, diagramas de interfaz de usuario, modelos de datos, etc.) rápidamente nos damos cuenta de que la mayoría de las veces solo necesita un subconjunto de los diagramas que tenemos.

- **Mostrar los modelos públicamente**: Debes mostrar tus modelos públicamente. Esto permite una comunicación abierta y honesta en tu equipo, ya que todos los modelos actuales son fácilmente accesibles para ellos, así como para las partes interesadas de tu proyecto porque no les estás ocultando nada.

- **Alternar entre artefactos**: Cuando estés trabajando en un artefacto de desarrollo y descubras que estás atascado, entonces deberías considerar trabajar en otro artefacto por el momento. Cada artefacto tiene sus fortalezas y debilidades, cada artefacto es bueno para cierto tipo de trabajo.

- **Modelar en pequeños incrementos**: Esto significa que nuestro desarrollo debe de ser incremental, realizando pequeños avances que debemos de lanzar en semanas o incluso en uno o dos meses, que permite dar a los usuarios un producto de una manera más rápida.

- **Modelar en grupo**: Cuando modelas con un propósito, a menudo encuentras que estás modelando para entender algo, que estás modelando para comunicar sus ideas a otros, o que está buscando desarrollar una visión común de tu proyecto. Esta es una actividad grupal, en la que desea que la información de varias personas trabajen juntas de manera efectiva.

- **Probarlo con código**: Para determinar si un modelo tiene el funcionamiento que esperamos de él debemos realizar pruebas al modelo, todo esto, mediante código.

- **Usar las herramientas más simples**: Para representar modelos utilizamos herramientas simples, como una pizarra, un dibujo, echarles una foto y, ocasionalmente, use¡amos una herramienta de modelado si y sólo si proporcionan valor a los esfuerzos de programación, como la generación de código. Esto es así, porque muchos modelos serán desechados

### 4.2. Prácticas Suplementarias

- **Aplicar los estándares de modelado**: La idea básica es que los desarrolladores deben aceptar y seguir un conjunto común de estándares de modelado en un proyecto de software.

- **Aplicar los patrones suavemente**: Si sospecha que se aplica un patrón, debe modelarse de tal manera que implemente la cantidad mínima que necesite al principio, lo que facilitará refactorizar el software en un futuro cuando se tenga claro qué patrón se utilizará.

- **Deseche los modelos temporales**: La gran mayoría de los modelos que se crean son modelos que han cumplido su propósito y ya no agregan ningún valor. Después se debe decidir si adaptar los modelos, pero solo si esto agrega valor al proyecto, o descartar esta posibilidad si la inversión de actualizar los modelos no se recuperará.

- **Formalizar modelos de contrato**: Un modelo de contrato es algo que ambas partes deben acordar y cambiar si es necesario. Su objetivo es minimizar la cantidad de modelos de contrato para que el sistema cumpla con los principios

de XP.

- **Actualizar solo cuando duele**: Debe actualizar un modelo solo cuando sea absolutamente necesario, cuando no tener el modelo actualizado es más doloroso que el esfuerzo de actualizarlo. Con este enfoque, se actualizan una cantidad de modelos muy pequeña, ya que los modelos no deben ser perfectos para aportar valor.

### 4.3. Really Good Ideas

Las siguientes prácticas son complementarias de AM, pero no se incluyen explícitamente como parte de ellas:

- **Refactorización**: Esta es una práctica de codificación en la que realiza pequeños cambios, llamados refactorizaciones, al código para añadir nuevos requisitos o para mantener su diseño lo más simple posible. Esta práctica garantiza que su diseño se mantenga limpio y claro.

- **Prueba de primer diseño**: Primero considera y luego codifica un caso de prueba antes de escribir el código de negocio que satisface este caso de prueba. Esta práctica obliga a pensar en su diseño antes de escribir su código, eliminando la necesidad de un modelado de diseño detallado.

# 5. AM y los procesos software base

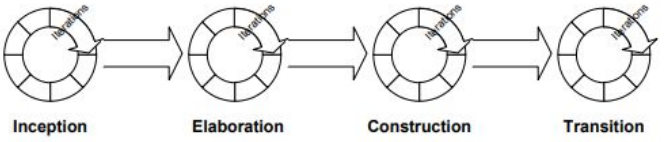
Si definimos el modelado ágil (AM) como un proceso software basado en prácticas cuyo objetivo es describir cómo modelar y documentar de una manera eficaz y ágil, podemos decir también que las prácticas del modelado ágil se deben utilizar, idealmente en su totalidad, para mejorar otros procesos de software más completos como la eXtreme Programming (XP), el Rational Unified Process (RUP) , Disciplined Agile Delivery (DAD), la Enterprise Unified Process (EUP)... Estos procesos cubren un alcance más amplio que el modelado ágil, en los primeros tres casos el proceso de desarrollo y en el cuarto el proceso de software completo que incluye tanto el desarrollo como la producción. Aunque todos estos procesos incluyen actividades de modelado y documentación, de una forma u otra, definitivamente hay espacio para mejorar. Con DAD, las prácticas de modelado ágil están integradas en el marco, con XP, los procesos de modelado deberían definirse mejor, y con RUP, los procesos de modelado podrían definitivamente hacerse más ágiles. Ahora pasaremos a comentar el RUP como ejemplo.

## 5.1. RUP

RUP es un enfoque iterativo para sistemas orientados a objetos, y abarca en gran medida los casos de uso para los requisitos de modelado y la creación de las bases para un sistema. RUP está inclinado hacia el desarrollo orientado a objetos. No descarta implícitamente otros métodos, aunque el método de modelado propuesto, UML, es particularmente adecuado para el desarrollo de OO.

### Proceso

La vida útil de un proyecto RUP se divide en cuatro fases denominadas Inicio, Elaboración, Construcción y Transición. Estas fases se dividen en iteraciones, cada una de las cuales tiene el propósito de producir un software demostrable. La duración de una iteración puede variar desde dos semanas o menos hasta seis meses.




- En la fase inicial, los objetivos del ciclo de vida del proyecto se establecen de manera que se tengan en cuenta las necesidades de todos los interesados (el alcance, criterios de aceptación, casos de uso críticos, arquitecturas candidatas).

- La fase de elaboración es donde se sientan las bases de la arquitectura del software. RUP asume que la fase de elaboración producirá una arquitectura suficientemente sólida junto con requisitos y planes suficientemente estables. Se describe, además, la arquitectura del software y se crea un prototipo ejecutable de la arquitectura.

- En la fase de construcción, todos los componentes restantes y las características de la aplicación se desarrollan e integran en el producto y se prueban. Los resultados de la fase de construcción (alfa, beta y otras versiones de prueba) se crean lo más rápido posible, sin dejar de alcanzar la calidad adecuada. Una o más versiones se realizan durante la fase de construcción, antes de pasar a la fase final de transición.

- La fase de transición se ingresa cuando el producto de software está lo suficientemente maduro para ser enviado a la comunidad de usuarios. En función de la respuesta del usuario, las versiones posteriores se realizan para corregir los problemas pendientes o para finalizar las funciones pospuestas. Distribución y equipos de ventas a menudo realizan varias iteraciones, con versiones beta y de disponibilidad general. También se produce documentación del usuario (manuales, material del curso).

### Roles y responsabilidades

En RUP hay un rol para cada actividad. RUP define treinta roles, llamados trabajadores. La conversión es bastante convencional (por ejemplo, Arquitecto, Diseñador, Revisor de diseño, Administrador de configuración), con la excepción de los roles definidos en los flujos de trabajo de modelado de negocios y entorno.

# 6. Bibliografía


1. https://es.wikiversity.org/wiki/Modelado_%C3%81gil
2. http://agilemodeling.com/
3. http://www.lsi.us.es/docencia/get.php?id=567
4. http://wikis.uca.es/wikiCE/index.php/Agile_modeling
5. Abrahamsson, Pekka & Salo, Outi & Ronkainen, Jussi & Warsta, Juhani. (2002). Agile Software Development Methods: Review and Analysis. Proc. Espoo 2002. 3-107.