



Ingeniería de Software I

Metodologías Ágiles

Metodologías Ágiles

En los años 80 y principios de los 90, existía una opinión general de que la mejor forma de obtener un mejor software era a través de una planificación cuidadosa del proyecto, la utilización de métodos de análisis y diseño, y procesos de desarrollo de software controlados y rigurosos.

En general se realizaban sistemas críticos, desarrollados por grandes equipos, a menudo dispersos geográficamente.

Sin embargo, cuando este enfoque fue aplicado a sistemas de negocio pequeños y de tamaño medio, el esfuerzo invertido era grande, y cuando cambiaban los requerimientos, se hacía esencial rehacer el trabajo.

Del descontento nacieron las metodologías ágiles.

Metodologías Ágiles

Introducción

El éxito de un desarrollo esta dado por la metodología empleada la cual nos da una dirección a seguir para su correcta conclusión.

Generalmente esta metodología lleva asociado un marcado énfasis en el control del proceso, definiendo roles, actividades, herramientas y documentación detallada.

Este enfoque no resulta ser muy adecuado para proyectos actuales donde el entorno del sistema es muy cambiante y se exige una reducción de tiempo.

Ante estas dificultades, muchos equipos se resignan a prescindir de las buenas prácticas, asumiendo los riesgos.

En este contexto, las metodologías ágiles emergen como una posible solución.

Metodologías Ágiles Introducción

“Es un enfoque iterativo e incremental (evolutivo) de desarrollo de software”

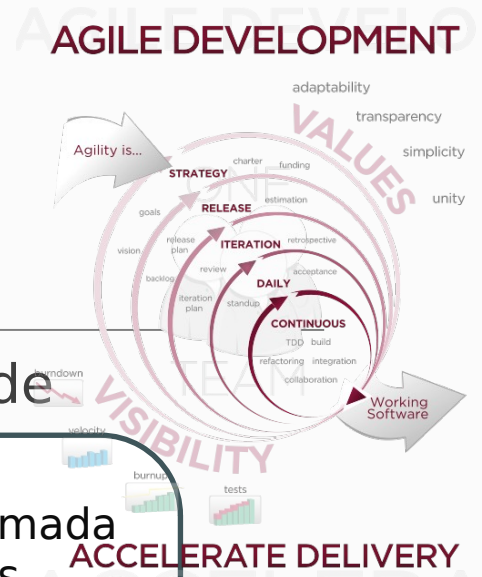
Objetivos :

- Producir software de alta calidad con un costo apropiado.
- Esbozar los valores y principios que deben guiar el desarrollo de software para que puedan surgir a lo largo del tiempo.
- Ofrecer una alternativa a las metodologías tradicionales, características de poca documentación que se desarrolla en etapas.

El desarrollo iterativo es una estrategia de reproceso en la que el tiempo se separa para revisar y mejorar partes del sistema.

es una estrategia programada y en etapas, en la que las diferentes partes del sistema se desarrollan en diferentes momentos o a diferentes velocidades, y se integran a medida que se completan

desarrollo de software y dirigidos por la planificación de las actividades



Reseña...

Una Metodología Ágil es aquella en la que “se da prioridad a las tareas que dan resultados directos y que reducen la burocracia tanto como sea posible” [Fowler], adaptándose además rápidamente al cambio de los proyectos.

Ese enfoque ha sido utilizado desde hace más de dos décadas por un grupo de profesionales de software.

Un Poco De Historia...

Así es como nace “The Agile Alliance”(AA), [AAlliance-www], una organización dedicada a promover los conceptos de desarrollo de software ágil, y de ayudar a las organizaciones a adoptar dichos conceptos. Estos conceptos están resumidos en el Manifiesto para el Desarrollo Ágil de Software y consta de valores y principios.
<https://www.agilealliance.org/>

El Manifiesto [AManifesto-www] fue redactado, entre otros, por Kent Beck, el “padre” de XP.

La definición moderna de desarrollo ágil de software evolucionó a mediados de los años 1990 y en el año 2001, miembros prominentes de la comunidad se reunieron en Snowbird, Utah”

Valores...

- **Individuos e interacciones** más que procesos y herramientas.
- **Software operante** más que documentaciones completas.
- **Colaboración con el cliente** más que negociaciones contractuales.
- **Respuesta al cambio** más que apegarse a una rigurosa planificación.

Es importante comprender que aún cuando se deben valorar los conceptos que se encuentran del lado derecho, debemos valorar aún más aquellos que están a la izquierda.

Una buena manera de interpretar el manifiesto, es asumir que éste define preferencias, no alternativas.

Principios

1. Nuestra mayor prioridad es satisfacer al cliente a través de fáciles y continuas entregas de software valuable.
2. Los cambios de requerimientos son bienvenidos, aún tardíos, en el desarrollo. Los procesos Ágiles capturan los cambios para que el cliente obtenga ventajas competitivas.
3. Entregas frecuentes de software, desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre una entrega y la siguiente.
4. Usuarios y desarrolladores deben trabajar juntos durante todo el proyecto.

Principios...

- 5. Construir proyectos alrededor de motivaciones individuales.
- 6. Darles el ambiente y el soporte que ellos necesitan y confiar el trabajo dado. El diálogo cara a cara es el método más eficiente y efectivo de intercambiar información entre el equipo de desarrolladores.
- 7. El software que funciona es la medida clave de progreso.
- 8. Los procesos ágiles promueven un desarrollo sostenible. Los stakeholders, desarrolladores y usuarios deberían ser capaces de mantener un paso constante indefinidamente.

Principios...

- 9. Atención continua a la excelencia técnica y buen diseño incrementa la agilidad.
- 10. Simplicidad (el arte de maximizar la cantidad de trabajo no dado) es esencial.
- 11. Las mejores arquitecturas, requerimientos y diseños surgen de la propia organización de los equipos.
- 12. A intervalos regulares, el equipo reflexiona sobre cómo volverse más efectivo, entonces afina y ajusta su comportamiento en consecuencia.

Comparación Ágil vs. No Ágil

Metodología Ágil	Metodología No Ágil
Pocos Artefactos	Más Artefactos
Pocos Roles	Más Roles
No existe un contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo (además in-situ)	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (< 10 integrantes) y trabajando en el mismo sitio	Grupos grandes
Menos énfasis en la arquitectura	La arquitectura es esencial

Desventajas

En la práctica, los principios que subyacen a los métodos ágiles son a veces difíciles de cumplir:

- . **Aunque es atractiva la idea de involucrar al cliente en el proceso de desarrollo,** los representantes del cliente están sujetos a otras presiones, y no intervienen por completo en el desarrollo del software.
- . **Priorizar los cambios podría ser difícil,** sobre todo en sistemas donde existen muchos participantes. Cada uno por lo general ofrece diversas prioridades a diferentes cambios.
- . **Mantener la simplicidad requiere trabajo adicional.** Bajo la presión de fechas de entrega, es posible que los miembros del equipo carezcan de tiempo para realizar las simplificaciones deseables al sistema.
- . **Muchas organizaciones, especialmente las grandes compañías, pasan años cambiando su cultura, de tal modo que los procesos se definan y continúen.** Para ellas, resulta difícil moverse hacia un modelo de trabajo donde los procesos sean informales y estén definidos por equipos de desarrollo.

Desventajas

Por lo general, el documento de requerimientos del software forma parte del contrato entre el cliente y el proveedor. Como en los métodos ágiles se minimiza la documentación, suele ser complejo reglamentarlo.

La mayoría de los libros que describen los métodos ágiles y las experiencias con éstos hablan del uso de dichos métodos para el desarrollo de nuevos sistemas. Sin embargo, una enorme cantidad de esfuerzo en ingeniería de software se usa en el mantenimiento y la evolución de los sistemas de software existentes. Al no existir documentación se complejizaría.

Principales Metodologías Agiles

XP Extreme Programming

www.extremeprogramming.org

Scrum

DSDM (Dynamic Systems Development Method)

www.dsdm.org

Crystal Methods (Cockburn's Crystal Family Methodologies) www.crystallmethodologies.org

ASD Adaptative Software Development

www.adaptivesd.com

FDD Feature-Driven Development

eXtreme Programming

- » Es una disciplina de desarrollo de software basado en los valores de la *sencillez*, la *comunicación*, la *retroalimentación*, la *valentía* y el *respeto*
- » Su acción consiste en llevar a todo el equipo reunido en la presencia de prácticas simples, con suficiente información para ver dónde están y para ajustar las prácticas a su situación particular.

eXtreme Programming

- ☐ Desarrollo iterativo e incremental: pequeñas mejoras, unas tras otras.
- ☐ Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión.
- ☐ Programación en parejas
- ☐ Frecuente integración del equipo de programación con el cliente o usuario.
- ☐ Corrección de todos los errores antes de añadir nueva funcionalidad.
- ☐ Refactorización del código
- ☐ Propiedad del código compartida
- ☐ Simplicidad en el código

EXtreme P rogramming



Los principios básicos de *Programación Extrema (XP)* [Bec 1999] son :

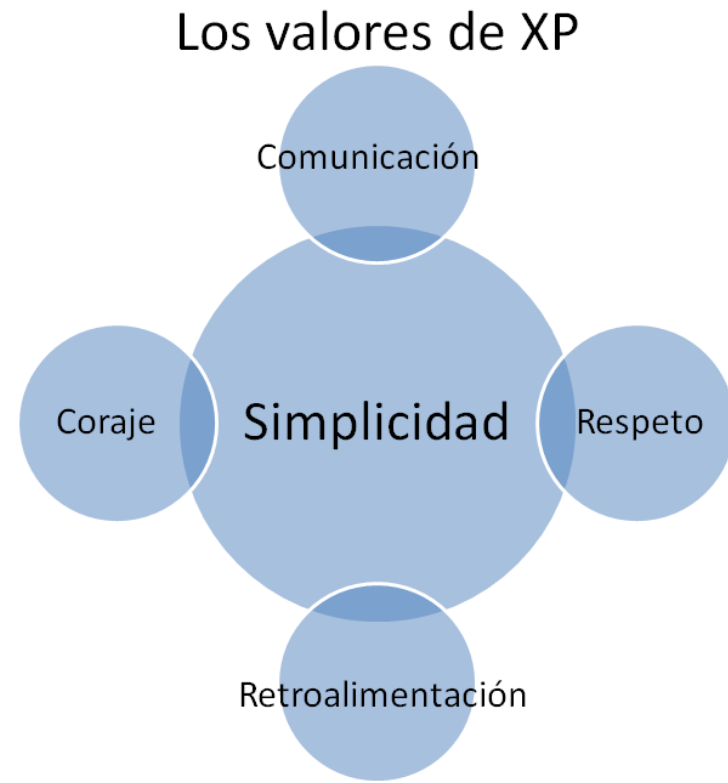


Imagen elaborada por oficinaproyectosinformatica.blogspot.com

Extreme Programming - Características

Las características esenciales :

- ☐ Historias de usuario
- ☐ Roles
- ☐ Proceso
- ☐ Prácticas

XP - Roles

- **Programador**
(Programmer)

- Responsable de decisiones técnicas
- Responsable de construir el sistema
- Sin distinción entre analistas, diseñadores o codificadores
- En XP, los programadores diseñan, programan y realizan las pruebas

- **Jefe de Proyecto** *(Manager)*

- Organiza y guía las reuniones
- Asegura condiciones adecuadas para el proyecto

- **Cliente** *(Customer)*

- Es parte del equipo
- Determina qué construir y cuándo
- Establece las pruebas funcionales

XP - Roles

- Entrenador (*Coach*)

- Responsable del proceso
- Tiende a estar en un segundo plano a medida que el equipo madura

- Encargado de Pruebas (*Tester*)

- Ayuda al cliente con las pruebas funcionales
- Se asegura de que las pruebas funcionales se superan

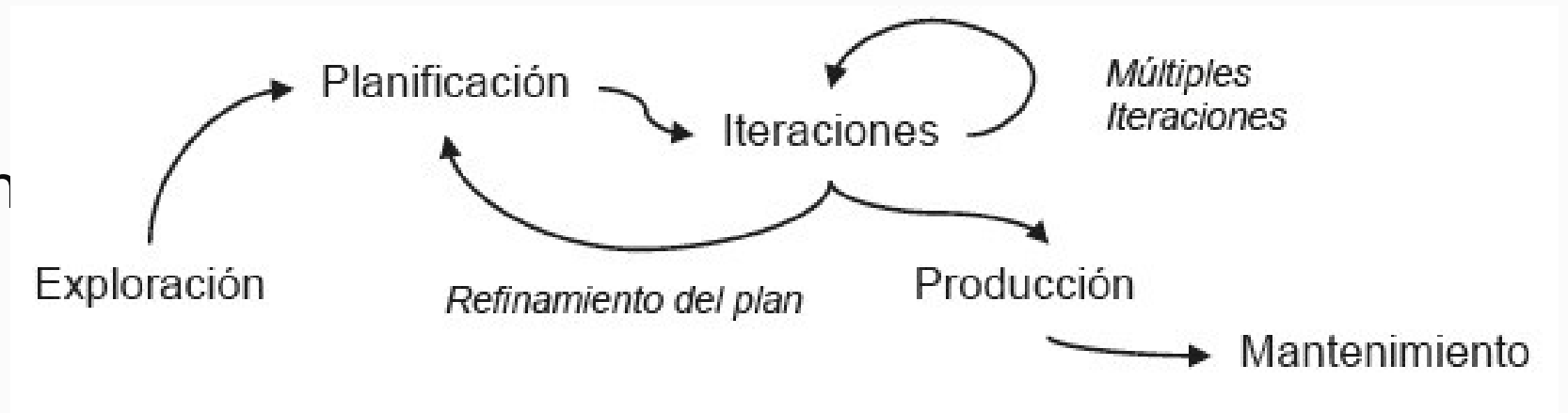
- Rastreador (*Tracker*)

- *Metric Man*
- Observa sin molestar
- Conserva datos históricos

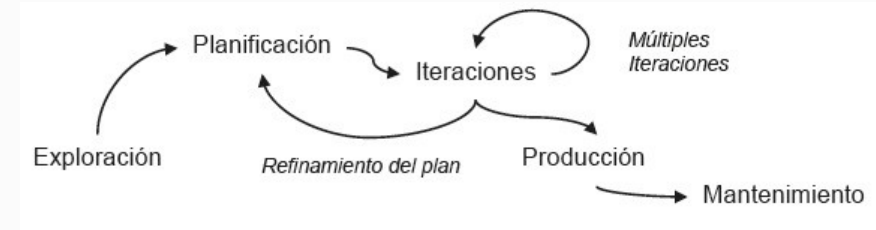
XP - Proceso

El ciclo de vida consiste en:

1. Exploración
2. Planificación
3. Iteraciones
4. Producción
5. Mantenimiento
6. Muerte



XP - Proceso



1. Exploración

- Los clientes plantean las historias de usuario que son de interés para la primera entrega del producto.
- El equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto.
- Se construye un prototipo.

La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

XP - Proceso



2. Planificación

- El cliente establece la prioridad de cada historia de usuario.
- Los programadores realizan una estimación del esfuerzo.
- Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente.

Esta fase dura unos pocos días.

XP - Proceso

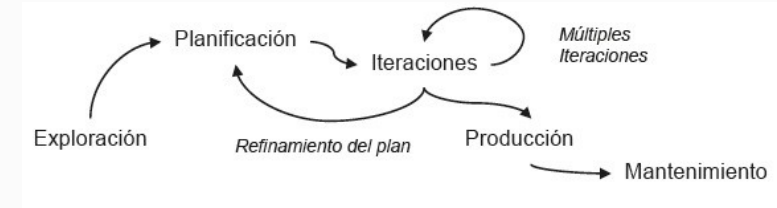


3. Iteración

- El Plan de Entrega está compuesto por iteraciones de no más de tres semanas.
- El cliente es quien decide qué historias se implementarán en cada iteración
- Al final de la última iteración el sistema estará listo para entrar en producción.

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado.

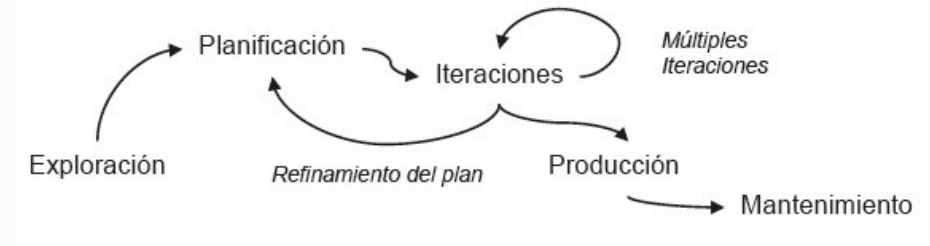
XP - Proceso



4 - Producción

- Esta fase requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente.
- Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase.

XP - Proceso



5. Mantenimiento

- Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones.
- La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.

XP - Proceso

6. Muerte

- Es cuando el cliente no tiene más historias para ser incluidas en el sistema.
- Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura.
- La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo.

Extreme Programming - Prácticas

☐ Testing:

Los programadores continuamente escriben pruebas unitarias, las cuales deben corregir problemas para que el desarrollo continúe.

Los clientes escriben pruebas demostrando que las funcionalidades están terminadas.



☐ Refactoring:

Actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios.

☐ Programación de a Pares:

Todo el código de producción es escrito por dos programadores en una máquina.



Extreme Programming - Prácticas



☐ Propiedad Colectiva del Código:

Cualquiera puede cambiar código en cualquier parte del sistema en cualquier momento.

Motiva a contribuir con nuevas ideas, evitando a la vez que algún programador sea imprescindible.

☐ Integración Continua:

Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.

Reduce la fragmentación de los esfuerzos de los desarrolladores por falta de comunicación sobre lo que puede ser reutilizado o compartido.



Extreme Programming - Prácticas

- ❑ Semana de 40-horas:

Se debe trabajar un máximo de 40 horas por semana.

El trabajo extra desmotiva al equipo.

Los proyectos que requieren trabajo extra para intentar cumplir con los plazos suelen al final ser entregados con retraso. En lugar de esto se puede realizar el juego de la planificación para cambiar el ámbito del proyecto o la fecha de entrega.

- ❑ Cliente en el Lugar de Desarrollo:

El cliente tiene que estar presente y disponible todo el tiempo para el equipo.

- ❑ Estándares de Codificación:

Los programadores escriben todo el código de acuerdo con reglas que enfatizan la comunicación a través del mismo.

SCRUM



Es un marco de trabajo utilizado para desarrollar productos complejos donde se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente y obtener el mejor resultado posible de un proyecto

Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.

Scrum se define como “una manera simple de manejar problemas complejos”, proporcionando un paradigma de trabajo que soporta la innovación y permite que equipos auto-organizados entreguen resultados de alta calidad en tiempos cortos.

En Scrum se realizan entregas parciales y regulares del resultado final del proyecto, priorizadas por el beneficio que aportan al receptor del proyecto

Scrum - Principios



Eliminar el desperdicio: no generar artefactos, ni perder el tiempo haciendo cosas que no le suman valor al cliente.

Construir la calidad con el producto: la idea es inyectar la calidad directamente en el código desde el inicio.

Crear conocimiento: En la práctica no se puede tener el conocimiento antes de empezar el desarrollo.

Diferir las decisiones: tomar las decisiones en el momento adecuado, esperar hasta ese momento, ya que uno tiene mas información a medida que va pasando el tiempo. Si se puede esperar, mejor.

Scrum - Principios



Entregar rápido: Debe ser una de las ventajas competitivas más importantes.

Respetar a las personas: la gente trabaja mejor cuando se encuentra en un ambiente que la motive y se sienta respetada.

Optimizar el todo: optimizar todo el proceso, ya que el proceso es una unidad, y para lograr tener éxito y avanzar, hay que tratarlo como tal.

Roles - Scrum

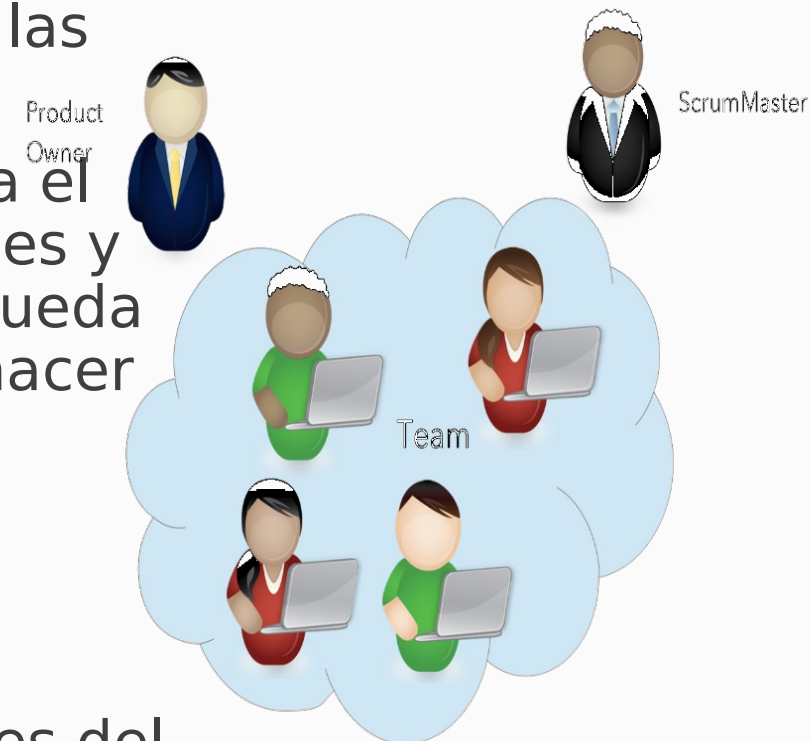


El **Product Owner (Propietario)** conoce y marca las prioridades del proyecto o producto.

El **Scrum Master (Jefe)** es la persona que asegura el seguimiento de la metodología guiando las reuniones y ayudando al equipo ante cualquier problema que pueda aparecer. Su responsabilidad es entre otras, la de hacer de paraguas ante las presiones externas.

El **Scrum Team (Equipo)** son las personas responsables de implementar la funcionalidad o funcionalidades elegidas por el Product Owner.

Los **Usuarios o Cliente**, son los beneficiarios finales del producto, y son quienes viendo los progresos, pueden aportar ideas, sugerencias o necesidades.



Artefactos - Scrum

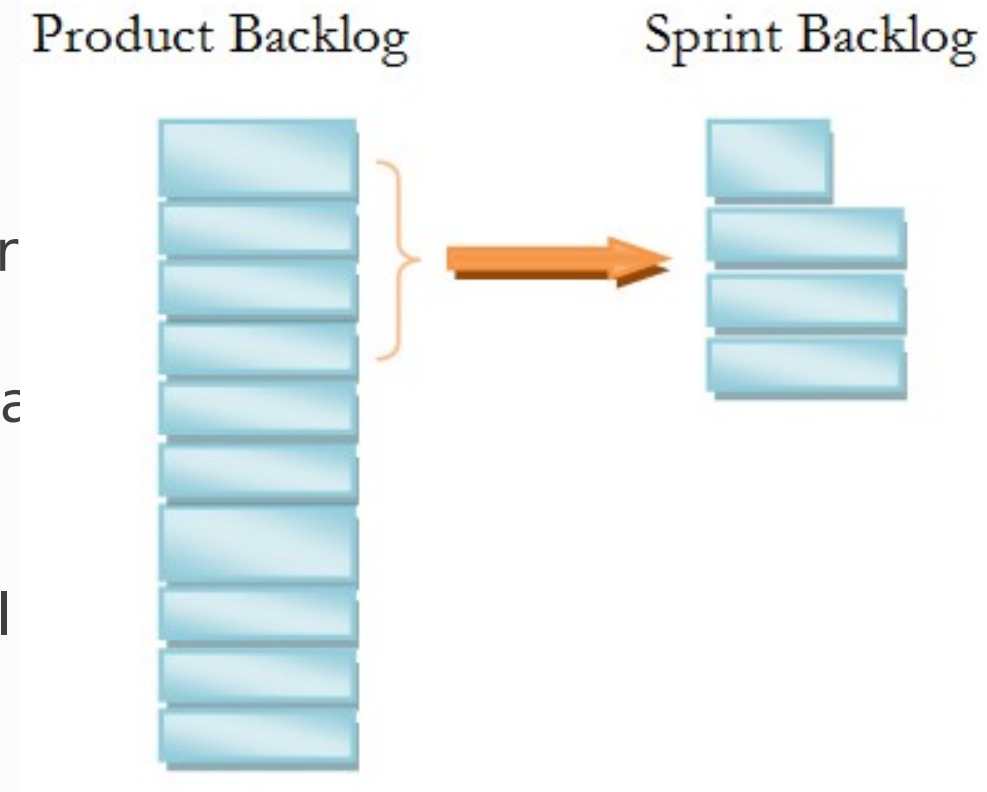


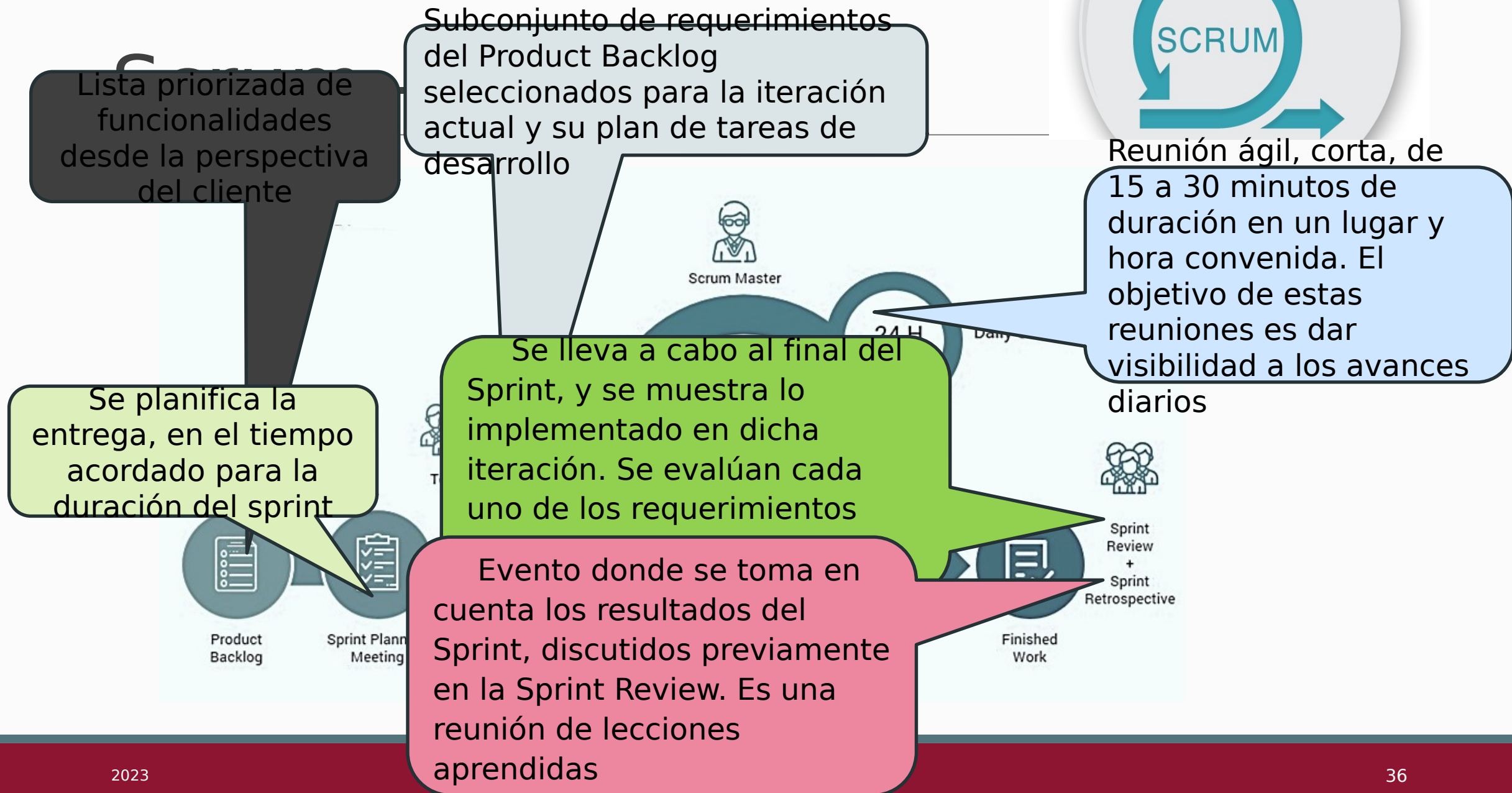
Product Backlog: es la lista maestra que contiene toda la funcionalidad deseada en el producto. La característica más importante es que la funcionalidad se encuentra ordenada por un orden de prioridad.

Sprint Backlog: es la lista que contiene toda la funcionalidad que el equipo se comprometió a desarrollar durante un Sprint determinado.

Burndown Chart: muestra un acumulativo del trabajo hecho, día-a-día.

Entre otros...





Scrum - Proceso



Scrum es **iterativo e incremental**

Se busca poder atacar todos los problemas que surgen durante el desarrollo del proyecto.

El nombre Scrum se debe a que durante los Sprints, lo que serían las fases de desarrollo, se solapan, de manera que no es un proceso de cascada por cada iteración, si no que tenemos todas éstas etapas juntas que se ejecutan una y otra vez, hasta que se crea suficiente.

Scrum - Proceso



Este solapamiento de fases se puede asemejar a un scrum de rugby, en el cual todos los jugadores (o roles, en nuestro caso), trabajan juntos para lograr un objetivo.



¿Cuándo usar Scrum?



Scrum está pensado para ser aplicado en proyectos en donde el “caos” es una constante, aquellos proyectos en los que tenemos requerimientos dinámicos, y que tenemos que implementar tecnología de punta.

Esos proyectos difíciles, que con los enfoques tradicionales se hace imposible llegar a buen puerto.

KANBAN

Es un enfoque Lean de desarrollo de software ágil.

Literalmente, Kanban es una palabra japonesa. En Toyota, Kanban es el término que describe un sistema de control visual y física que une todo el sistema de producción.

Kanban se presenta como una herramienta de gestión de trabajo con las siguientes prácticas:

- limitar el trabajo en curso
- visualizar el flujo de trabajo
- medir el tiempo promedio de entrega.

Filosofía que hace hincapié en la eliminación de residuos o de no valor añadido a través de la mejora continua para agilizar las operaciones. Está centrado en el cliente y enfatiza el concepto de eliminar cualquier actividad que no agregue valor a la creación o entrega de un producto o servicio. Lean se centra en ofrecer una mayor calidad, reducir el tiempo de ciclo y reducir los costos.

Esto la convierte en una herramienta simple, pero a la vez muy potente



KANBAN

Esta metodología ha ganado popularidad en el mantenimiento de multiproyectos de diferentes características.

Kanban trata de administrar el flujo de trabajo. Inicialmente, no reemplaza nada que la organización haga, simplemente impulsa el cambio.

A diferencia de otras metodologías no define ningún rol y no prescribe una secuencia de pasos definidos para llevar a cabo el desarrollo del proyecto. Su implementación es sencilla (3 prácticas)

Es habitual combinar en las prácticas de otras metodologías ágiles los tableros Kanban que son la herramienta ágil por excelencia de esta metodología, donde se visualizan las tareas a realizar, las realizadas y lo pendiente. Estos, son una herramienta para el control visual del sistema que ayuda a un equipo a visualizar explícitamente el proceso de desarrollo.

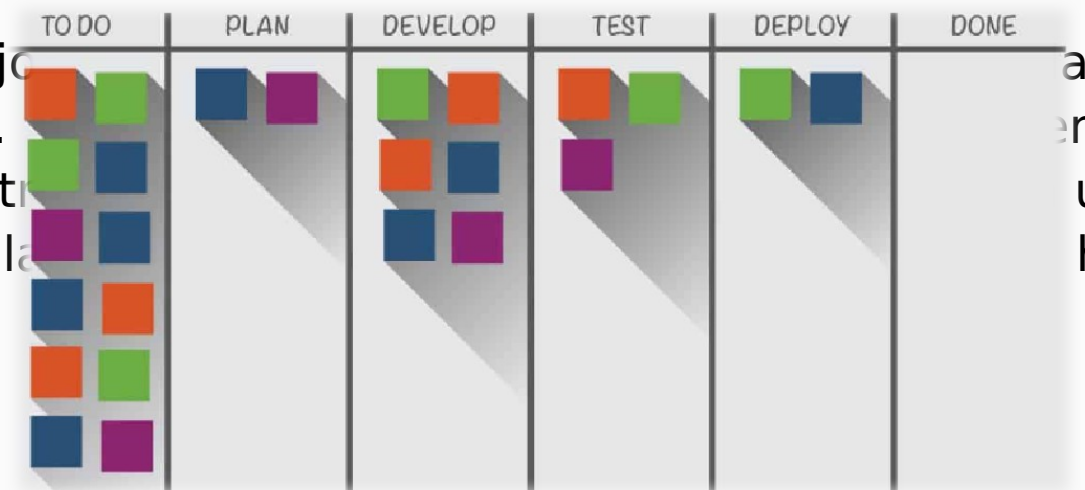
KANBAN



A primera vista, el tablero Kanban se asemeja a un proceso en cascada. Pero en la práctica, evita explícitamente los problemas de un proceso de estas características mediante la aplicación de lotes pequeños de trabajo y el desarrollo incremental.

Está compuesto por una serie de columnas que representan los diversos estados que atraviesa un requerimiento durante el proceso de desarrollo. Las tarjetas se mueven de un estado a otro mostrando la evolución hasta que haya sido aceptado por el cliente.

Cada columna tendrá un límite para el trabajo. Cuando una tarjeta se mueve de una columna a la siguiente, esta se mueve a la siguiente. La columna actual representando capacidad de trabajo. Una tarjeta terminada desde el estado anterior y la capacidad para procesarla.





Desarrollo de Software Basado en Modelos

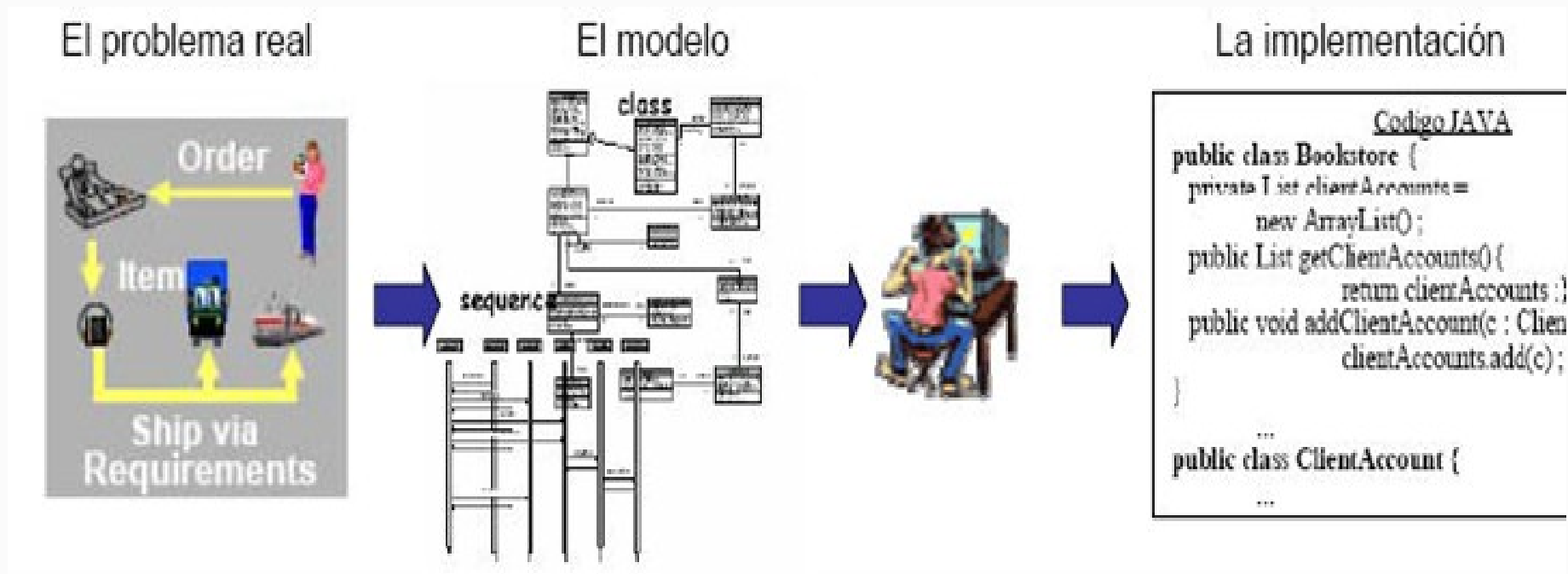
El Desarrollo de Software Basado en Modelos. (MBD)

Hacia fines de los 70' De Marco introdujo el concepto de desarrollo de software basado en modelos. Destacó que la construcción de un sistema de software debe ser precedida por la construcción de un modelo, tal como se realiza en otros sistemas ingenieriles.

Un modelo del sistema consiste en una conceptualización del dominio del problema y actúa como una especificación precisa de los requerimientos que el sistema de software debe satisfacer. (Abstracción de elementos del problema, comunicación, negociación con el usuario)

El Desarrollo de Software Basado en Modelos. (MBD)

Construcción de un Sistema de Software



Desarrollo de Software Dirigido por Modelos. (MDD)

El adjetivo «dirigido» en MDD, a diferencia de «basado», enfatiza que este paradigma asigna a los modelos un rol central y activo: son al menos tan importantes como el código fuente.

Model Driven Development (MDD) promueve enfatizar los siguientes puntos claves:

- Mayor nivel de abstracción en la especificación tanto del problema a resolver como de la solución correspondiente.
- Aumento de confianza en la automatización asistida por computadora para soportar el análisis, el diseño y la ejecución.
- Uso de estándares industriales como medio para facilitar las comunicaciones, la interacción entre diferentes aplicaciones y productos, y la especialización tecnológica.
- Los modelos son los conductores primarios en todos los aspectos del desarrollo de software.

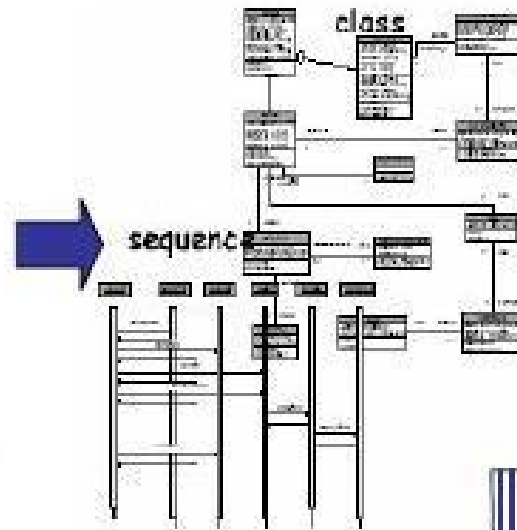
Desarrollo de Software Dirigido por Modelos. (MDD)

Los modelos pasan de ser entidades contemplativas (es decir, artefactos que son interpretados por los diseñadores y programadores) para convertirse en entidades productivas a partir de las cuales se deriva la implementación en forma automática.

El problema real



El modelo



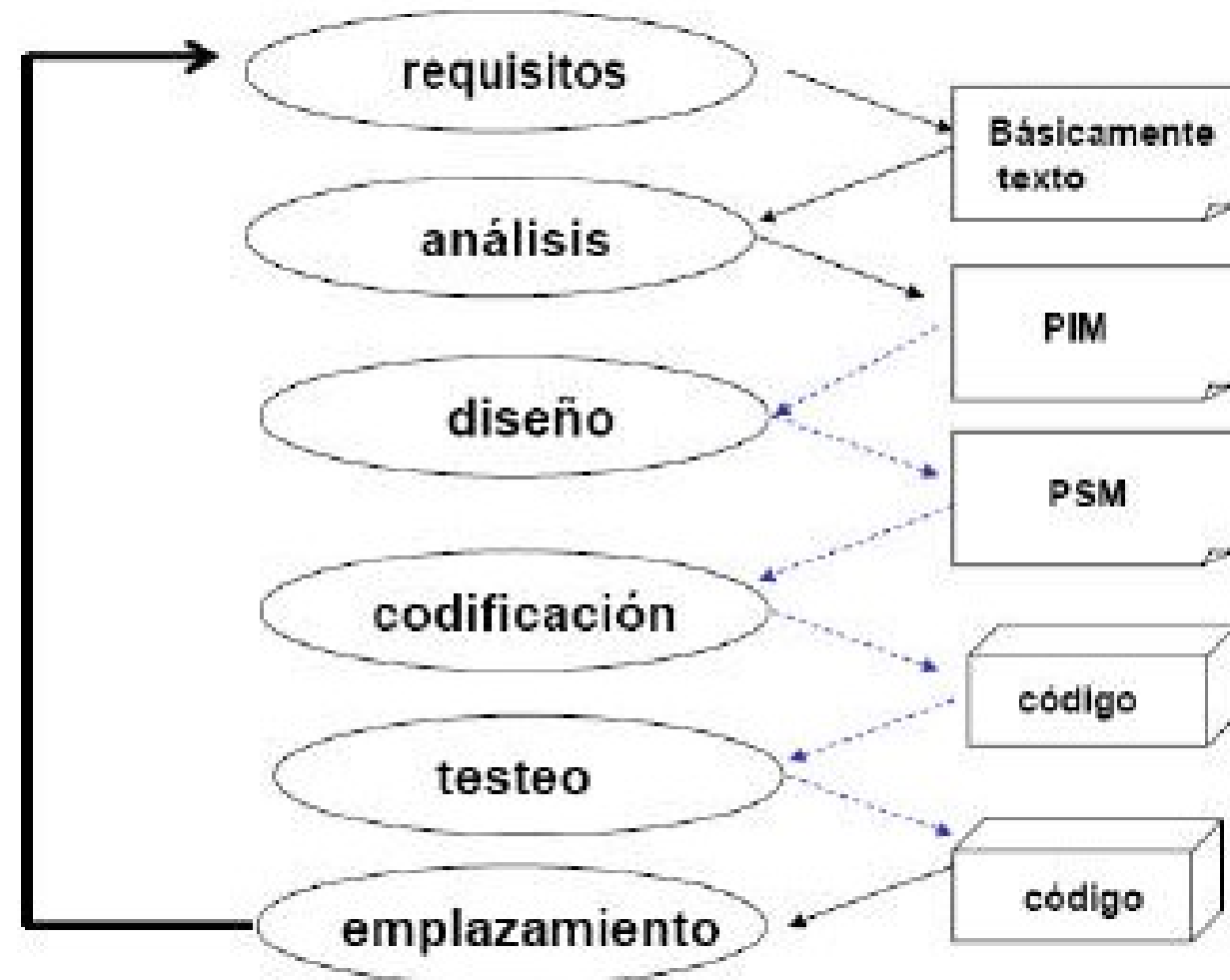
La implementación

Código JAVA

```
public class Bookstore {  
    private List clientAccounts =  
        new ArrayList();  
    public List getClientAccounts(){  
        return clientAccounts;  
    }  
    public void addClientAccount(c : ClientAccount){  
        clientAccounts.add(c);  
    }  
    ...  
    public class ClientAccount {  
        ...  
    }  
}
```

A blue arrow points from the 'El modelo' diagram to this implementation. Below the implementation, a computer icon is shown with a blue arrow pointing to it from the model, and a curved blue arrow pointing from the implementation back to the model, indicating a feedback loop.

Ciclo de Vida del Software Dirigido por Modelos.



Modelos de MDD. (PIMs, y PSMs)

Platform Independent Model (PIM): “Un modelo de un sistema que no contiene información acerca de la plataforma o la tecnología que es usada para implementarlo”

Platform Specific Model (PSM): “Un modelo de un sistema que incluye información acerca de la tecnología específica que se usará para su implementación sobre una plataforma específica”

Transformación de modelos: “Especifica el proceso de conversión de un modelo en otro modelo del mismo sistema.”

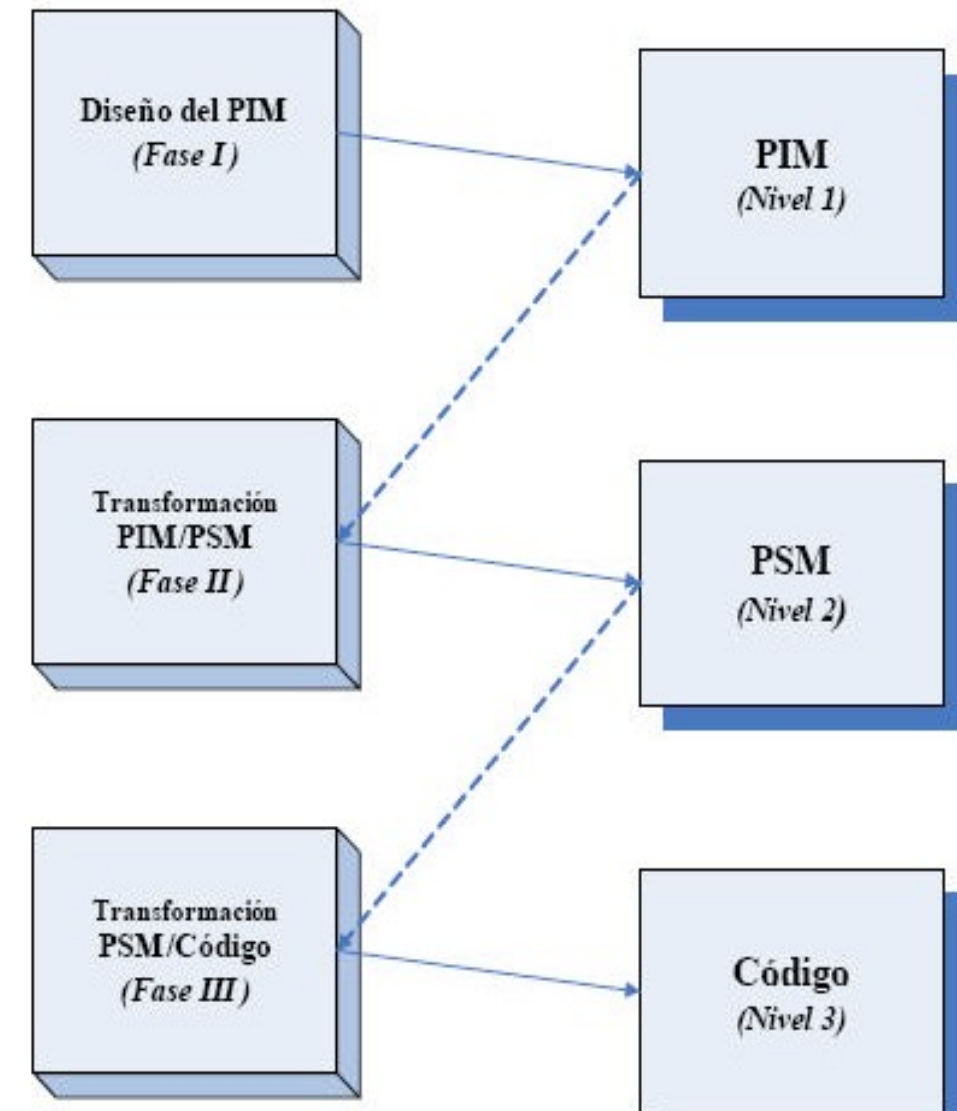
- Cada transformación incluye (al menos):
 - un PIM,
 - un Modelo de la Plataforma,
 - una Transformación, y
 - un PSM

Los tres pasos principales en el proceso de desarrollo MDD.

- » Platform Independent Model (PIM): “Un modelo de un sistema que no contiene información acerca de la plataforma o la tecnología que es usada para implementarlo”
- » Platform Specific Model (PSM): “Un modelo de un sistema que incluye información acerca de la tecnología específica que se usará para su implementación sobre una plataforma específica”
- » Transformación de modelos: “Especifica el proceso de conversión de un modelo en otro modelo del mismo sistema.”

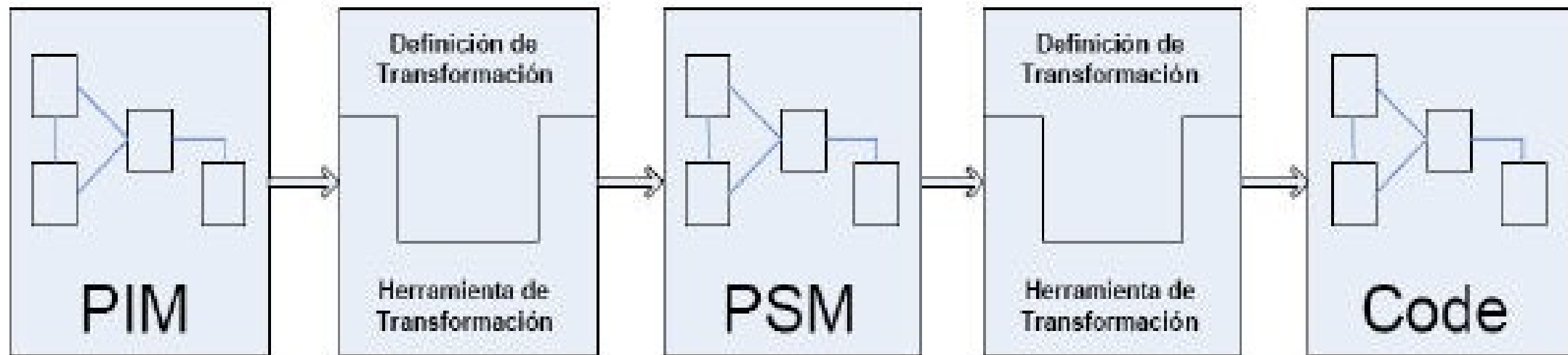
Cada transformación incluye (al menos):

*un PIM,
un Modelo de la Plataforma,
una Transformación, y
un PSM*



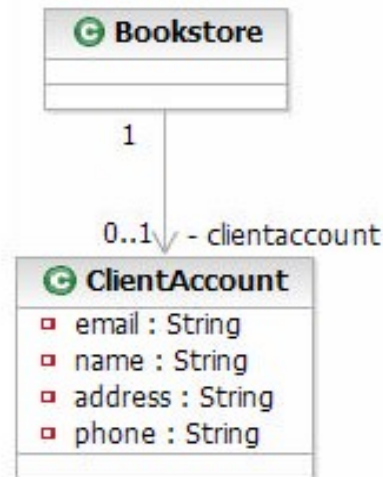
¿Qué es una transformación?

En general, se puede decir que una definición de transformación consiste en una colección de reglas, las cuales son especificaciones no ambiguas de las formas en que un modelo (o parte de él) puede ser usado para crear otro modelo (o parte de él).

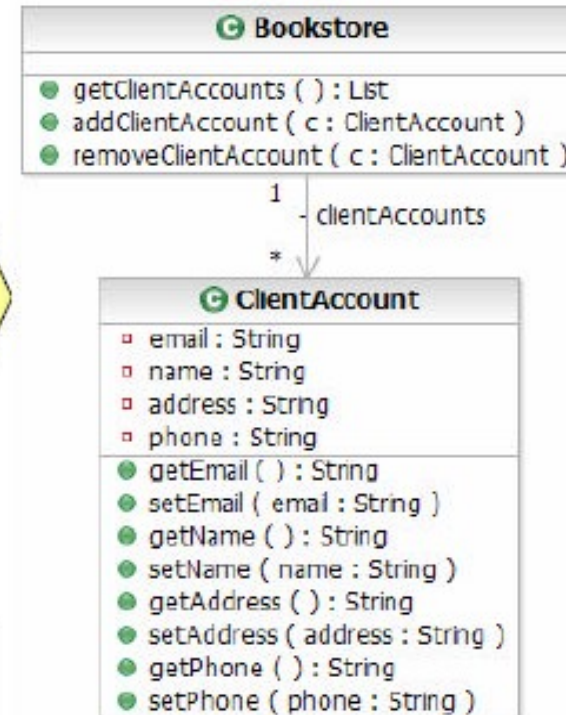


El patrón MDD es normalmente utilizado sucesivas veces para producir una sucesión de transformaciones.

Ejemplo de una transformación



Primer Paso
PIM al PSM



Segundo Paso
PSM al código

Código JAVA

```
public class Bookstore {
    private List clientAccounts =
        new ArrayList();
    public List getClientAccounts(){
        return clientAccounts;
    }
    public void addClientAccount(c : ClientAccount) {
        clientAccounts.add(c);
    }
}

...

public class ClientAccount {
    ...
}
```

Orígenes de MDD.

MDD es la evolución natural de la ingeniería de software basada en modelos enriquecida mediante el agregado de transformaciones automáticas entre modelos.

Si bien MDD define un nuevo paradigma para el desarrollo de software, sus principios fundamentales no constituyen realmente nuevas ideas sino que son reformulaciones y asociaciones de ideas anteriores.

La técnica de transformación se asemeja al proceso de abstracción y refinamiento presentado por Dijkstra.

Beneficios de MDD.

Incremento en la productividad (modelos y transformaciones).

Adaptación a los cambios tecnológicos.

Adaptación a los cambios de requisitos.

Consistencia (automatización).

Re-uso (de modelos y transformaciones).

Mejoras en la comunicación con los usuarios y la comunicación entre los desarrolladores (los modelos permanecen actualizados).

Captura de la experiencia (cambio de experto).

Los modelos son productos de larga duración (resisten cambios).