

# 2º

## Ingeniería del Software I

### TEMA 3 - Anexo

Introducción desarrollo ágil e software



# T3. Análisis de requisitos de un sistema.

## 3.1 Desarrollo ágil de software

### Objetivos

- Conocer los **métodos de desarrollo ágil** de software.
  - Razones de los métodos de desarrollo ágil, el manifiesto ágil, las diferencias entre el desarrollo ágil y el dirigido por un plan.
  - Prácticas clave en la programación extrema y cómo se relacionan con los principios generales de los métodos ágiles.
  - Enfoque de Scrum para la administración de un proyecto ágil.
  - Reconocerá los conflictos y problemas de escalar los métodos de desarrollo ágil para el diseño de grandes sistemas

# T3. Análisis de requisitos de un sistema.

## 3.1 Desarrollo ágil de software

### Contenido

3.1.1 Métodos ágiles

3.1.2 Desarrollo dirigido por un plan y desarrollo ágil

3.1.3 Programación extrema

Pruebas XP

# T3. Análisis de requisitos de un sistema.

## 3.1 Desarrollo ágil de software

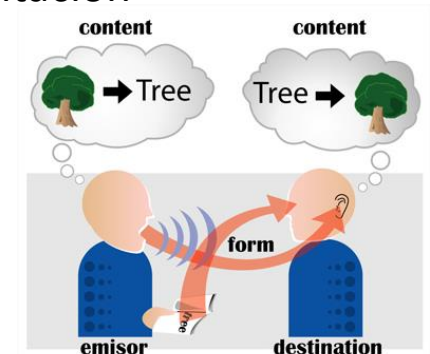
### Introducción

- En la actualidad la **entrega y el desarrollo rápidos** son por lo general el requerimiento fundamental de los sistemas de software.
- Es probable que debido a **factores externos**, los **requerimientos cambien rápida e impredeciblemente**.
  - En tal caso, el software podría ser obsoleto al momento de entregarse.
- Un **proceso convencional en cascada** o uno basado en especificación se prolongan con frecuencia, en tanto que el software final se entrega al cliente mucho después de lo que se especificó originalmente. **No es aplicable**.
  - **Mucho menos en sistemas de control críticos** para la seguridad, donde es esencial un análisis completo del sistema, resulta oportuno un enfoque basado en un plan.
- IBM introdujo el desarrollo incremental en la década de 1980. **Para desarrollo de sistemas dinámicos en el 2000**:
  - la noción prosperó realmente a finales de la década de 1990, con el desarrollo de la noción de enfoques ágiles como el DSDM, Scrum, y la programación extrema.

# T3. Análisis de requisitos de un sistema.

## 3.1 Desarrollo ágil de software

- Existen muchos enfoques para el desarrollo de software rápido, pero comparten algunas características fundamentales:
  - Los procesos de **especificación, diseño y desarrollo están entrelazados**. El documento de requerimientos del usuario define sólo las características más importantes del sistema.
  - El sistema se **desarrolla en diferentes versiones**. Los usuarios finales y otros colaboradores del sistema intervienen en la especificación y evaluación de cada versión.
  - Las **interfaces de usuario del sistema** se desarrollan usando con frecuencia un **sistema de elaboración interactivo**.
    - Diseño interactivo de interfaz + «client in the loop» + documentación minimalista.
- Involucran a los clientes en el proceso de desarrollo** para conseguir una **rápida retroalimentación** sobre los requerimientos cambiantes.
- Minimizan la cantidad de documentación** con el uso de comunicaciones informales.



# T3. Análisis de requisitos de un sistema.

## 3.1 Desarrollo ágil de software. 3.1.1 Métodos

### 3.1.1 Métodos ágiles

- En **sistemas pequeños**, los costes de planificación y gestión dominan al desarrollo.
- En la década de 1990 el descontento con estos enfoques condujo a algunos desarrolladores de software a **proponer «nuevos métodos ágiles»** los cuales permitieron que el equipo de desarrollo se **enfocara en el software** en lugar del diseño y la documentación.
  - Los métodos ágiles se apoyan universalmente en el **enfoque incremental** para la especificación, el desarrollo y la entrega del software.
  - Son más adecuados **para el diseño de aplicaciones** en que los requerimientos del sistema cambian, por lo general, rápidamente durante el proceso de desarrollo.
  - Se dirigen a **simplificar el proceso burocrático** al evitar trabajo con valor dudoso a largo plazo, y a **eliminar documentación** que quizá nunca se emplee.

# T3. Análisis de requisitos de un sistema.

## 3.1 Desarrollo ágil de software. 3.1.1 Métodos

- La filosofía de los métodos ágiles se refleja en el **manifiesto ágil**. Afirma:
  - *Estamos descubriendo mejores formas para desarrollar software, al hacerlo y al ayudar a otros a hacerlo. Gracias a este trabajo llegamos a valorar:*
    - A los **individuos y las interacciones** sobre los procesos y las herramientas.
    - Al **software operativo** sobre la documentación exhaustiva.
    - La **colaboración con el cliente** sobre la negociación del contrato.
    - La **respuesta al cambio** sobre el seguimiento de un plan.
  - *Esto es, aunque exista valor en los objetos a la derecha, valoraremos más los de la izquierda.*
- El éxito de dichos métodos condujo a cierta **integración con métodos más tradicionales de desarrollo**, basados en el modelado de sistemas, lo cual resulta en la noción de modelado ágil.
- Aunque los **métodos ágiles sean diferentes, comparten una serie de principios**, según el manifiesto ágil y, por ende, tienen mucho en común (los principios)



# T3. Análisis de requisitos de un sistema.

## 3.1 Desarrollo ágil de software. 3.1.1 Métodos

Los **principios de los métodos ágiles**

Principio	Descripción
<b>Participación del cliente</b>	Los clientes deben intervenir estrechamente durante el proceso de desarrollo. Su función consiste en ofrecer y priorizar nuevos requerimientos del sistema y evaluar las iteraciones del mismo.
<b>Entrega incremental</b>	El software se desarrolla en incrementos y el cliente especifica los requerimientos que se van a incluir en cada incremento.
<b>Personas, no procesos</b>	Tienen que reconocerse y aprovecharse las habilidades del equipo de desarrollo. Debe permitirse a los miembros del equipo desarrollar sus propias formas de trabajar sin procesos establecidos.
<b>Adoptar el cambio</b>	Esperar a que cambien los requerimientos del sistema y, de este modo, diseñar el sistema para adaptar dichos cambios.
<b>Mantener simplicidad</b>	Enfocarse en la simplicidad tanto en el software a desarrollar como en el proceso de desarrollo. Siempre que sea posible, trabajar de manera activa para eliminar la complejidad del sistema.



# T3. Análisis de requisitos de un sistema.

## 3.1 Desarrollo ágil de software. 3.1.1 Métodos

- Los métodos ágiles han tenido mucho **éxito** para ciertos **tipos de desarrollo de sistemas**:
  1. Desarrollo del producto, donde una compañía de software elabora un **producto pequeño o mediano** para su venta.
  2. **Diseño de sistemas internos** a la medida dentro de una organización, donde hay un claro compromiso del cliente por intervenir en el proceso de desarrollo, y donde no existen muchas reglas ni regulaciones externas.
- El éxito de los métodos ágiles se debe al interés considerable por usar dichos métodos **para otros tipos de desarrollo del software**.
  - **Dado su enfoque en equipos reducidos firmemente integrados, hay problemas en escalarlos hacia grandes sistemas (sistemas críticos).**

# T3. Análisis de requisitos de un sistema.

## 3.1 Desarrollo ágil de software. 3.1.1 Métodos

- **A veces**, en la práctica, los principios que subyacen a **los métodos ágiles son difíciles de cumplir**:
  1. Su **éxito radica en tener un cliente que desee y pueda pasar tiempo con el equipo de desarrollo**. ¿Está disponible el cliente?
  2. Quizás algunos **miembros del equipo** no cuenten con la **personalidad adecuada para la participación intensa** característica de los métodos ágiles.
  3. **Priorizar los cambios** sería extremadamente difícil. Sobre todo en sistemas grandes.
  4. **Mantener la simplicidad requiere trabajo adicional**. Bajo presión, es difícil.
  5. Muchas organizaciones. Resulta difícil moverse hacia un modelo de trabajo donde los procesos sean informales y estén definidos por equipos de desarrollo. **Difícil implantación en algunas compañías**.
- Cuando se recurre a una organización externa para el desarrollo del sistema, el documento de requerimientos forma parte del contrato entre el cliente y el proveedor.
  - Quizás sea difícil elaborar contratos para este tipo de desarrollo.
  - **Los métodos ágiles deben apoyarse en contratos, en los que el cliente pague por el tiempo requerido para el desarrollo del sistema**, en vez de hacerlo por el desarrollo de un conjunto específico de requerimientos.
    - Siempre que marche bien, beneficiará tanto al cliente como al desarrollador.

# T3. Análisis de requisitos de un sistema.

## 3.1 Desarrollo ágil de software. 3.1.1 Métodos

- Hay sólo un pequeño número de reportes de experiencia sobre el uso de métodos ágiles para el mantenimiento de software.
  - Tres preguntas que deberían considerarse junto con los métodos y el mantenimiento ágil:
    1. ¿Los sistemas que se desarrollan usando un enfoque ágil **se mantienen**, a pesar del énfasis en el proceso de desarrollo de minimizar la documentación formal?
    2. ¿Los métodos ágiles pueden usarse con efectividad para **evolucionar un sistema como respuesta a requerimientos** de cambio por parte del cliente?
    3. ¿**Cuándo se da por finalizado** el sistema?
- Se estima que la **documentación formal describe el sistema** y, por lo tanto, facilita la comprensión a quienes cambian el sistema.
  - Sin embargo, en la práctica, con frecuencia la documentación formal **no se conserva actualizada** y, por ende, no refleja con precisión el código del programa.
  - Los apasionados de los métodos ágiles argumentan que **escribir esta documentación es una pérdida de tiempo** y que la clave para **implementar software que se pueda mantener** es producir un **código legible de alta calidad**.
    - **Código apoyado de pseudocódigo detallado.**

# T3. Análisis de requisitos de un sistema.

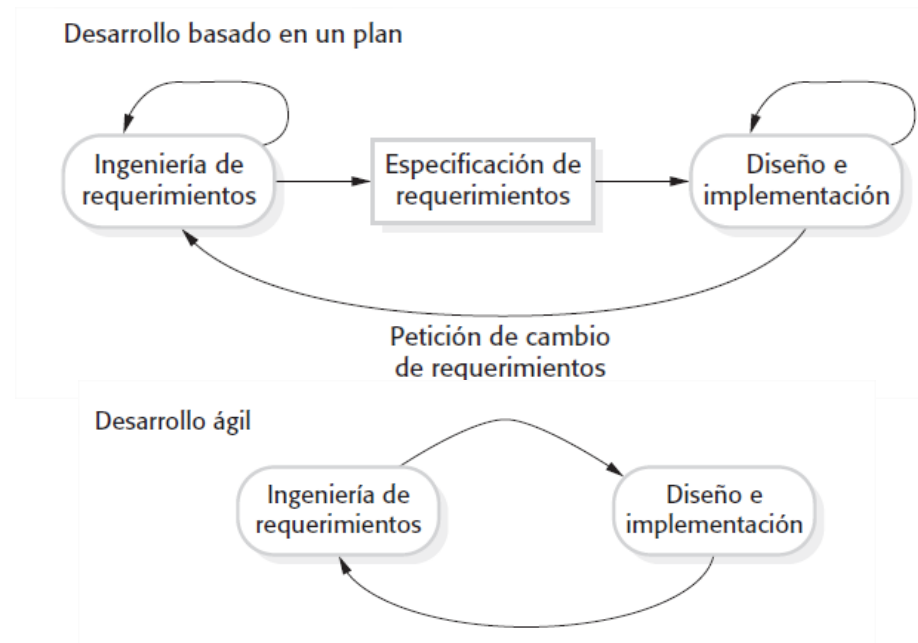
## 3.1 Desarrollo ágil de software. 3.1.1 Métodos

- No obstante, el **documento clave** es el documento de requerimientos del sistema, el cual indica al ingeniero de software lo que se supone que debe hacer el sistema.
- **Es probable que el uso de métodos ágiles haga más difícil y costoso el mantenimiento posterior del sistema.**
  - **Sin documentación no hay mantenimiento.**
- **Problema 1:** Sin embargo, quizá la principal dificultad luego de entregar el software sea **mantener al cliente interviniendo en el proceso**. ¿Continuará el cliente en el mantenimiento?
  - **Más factible durante el desarrollo, menos probable durante el mantenimiento.**
- **Problema 2: Continuidad del equipo de desarrollo.** Los métodos ágiles se apoyan en aquellos miembros del equipo que comprenden los aspectos del sistema sin que deban consultar la documentación.
  - Si se separa un equipo de desarrollo ágil, entonces **se pierde este conocimiento implícito** y es difícil que los nuevos miembros del equipo acumulen la misma **percepción del sistema y sus componentes**.
    - Algunos autores apuestan como solución híbrida para este problema, método ágil y técnicas de desarrollo dirigidas por un plan.

# T3. 3.1 Desarrollo ágil de software. 3.1.2

## Desarrollo dirigido por un plan y metodología ágil

- Los **enfoques ágiles** en el desarrollo de software consideran **el diseño y la implementación como las actividades centrales** en el proceso del software. Incorpora la adquisición de requerimientos y pruebas.
- Por contraste, **el enfoque basado en un plan**, identifica etapas separadas , las cuales producen **sus salidas**.



- El equipo de desarrollo ágil puede incluir un **“pico” de documentación** donde, en vez de producir una nueva versión de un sistema, el equipo **generará documentación** del sistema

## T3. 3.1 Desarrollo ágil de software. 3.1.2

### Desarrollo dirigido por un plan y metodología ágil

- **Preguntas para decidir sobre el equilibrio entre un enfoque basado en un método ágil o en un plan:**
  1. ¿Es importante tener una **especificación y un diseño muy detallados** antes de dirigirse a la implementación? → Sí, **plan**.
  2. ¿Es práctica una estrategia de entrega incremental, donde se dé el software a los clientes y se obtenga así una **rápida retroalimentación** de ellos? Sí, **métodos ágiles**.
  3. ¿Qué tan grande es el sistema que se desarrollará?
    - Los **métodos ágiles** son más efectivos cuando el sistema logra **diseñarse con un pequeño equipo** asignado que se **comunique de manera informal**.
    - Esto sería imposible para los **grandes sistemas** que precisan equipos de desarrollo más amplios. → **Plan**.
  4. ¿Qué tipo de sistema se desarrollará?
    - Los sistemas que **demandan mucho análisis antes de la implementación** (por ejemplo, sistema en tiempo real con requerimientos de temporización compleja), por lo general, necesitan un diseño bastante detallado para realizar este análisis. → **Plan**

## T3. 3.1 Desarrollo ágil de software. 3.1.2

### Desarrollo dirigido por un plan y metodología ágil

5. ¿Cuál es el tiempo de vida que se espera del sistema?
  - Los sistemas con **lapsos de vida prolongados** podrían requerir **más documentación de diseño**, para comunicar al equipo de apoyo los propósitos originales de los desarrolladores del sistema.
  - Los defensores de los **métodos ágiles** argumentan acertadamente que con frecuencia la documentación no se conserva actualizada, ni se usa mucho para el mantenimiento del sistema a largo plazo.
6. ¿Qué tecnologías se hallan disponibles para apoyar el desarrollo del sistema?
  - Los **métodos ágiles se auxilian a menudo de buenas herramientas** para seguir la pista de un diseño en evolución.
  - Si se **desarrolla un sistema con un IDE** sin contar con buenas herramientas para visualización y análisis de programas, entonces posiblemente se **requiera más documentación de diseño**.
7. ¿Cómo está organizado el equipo de desarrollo?
  - Si el equipo de desarrollo está **distribuido**, o si parte del desarrollo se subcontrata, entonces tal vez se requiera **elaborar documentos de diseño para comunicarse a través de los equipos de desarrollo**. Quizá se necesite planear por adelantado cuáles son.

## T3. 3.1 Desarrollo ágil de software. 3.1.2

### Desarrollo dirigido por un plan y metodología ágil

8. ¿Existen problemas culturales que afecten el desarrollo del sistema?
  - Las **organizaciones de ingeniería tradicionales** presentan una cultura de desarrollo basada en un **Plan**, pues es una norma en ingeniería. Esto requiere comúnmente una amplia documentación de diseño.
  - En vez del **conocimiento informal** que se utiliza en los **procesos ágiles**.
9. ¿Qué tan buenos son los diseñadores y programadores en el equipo de desarrollo?
  - Se argumenta en ocasiones que los métodos ágiles requieren niveles de habilidad superiores a los enfoques basados en un plan, en que los programadores simplemente traducen un diseño detallado en un código.
    - Si usted tiene un equipo con **niveles de habilidad relativamente bajos**, es probable que necesite del mejor **personal para desarrollar el diseño**, siendo otros los responsables de la programación. **Plan**
10. ¿El sistema está sujeto a regulación externa?
  - Si un regulador externo tiene que aprobar el sistema (por ejemplo, la Agencia de Aviación Federal [FAA] estadounidense aprueba el software que es crítico para la operación de una aeronave), entonces, tal vez se le **requerirá documentación detallada** como parte del sistema de seguridad. **Plan**



## T3. 3.1 Desarrollo ágil de software

- En realidad, es irrelevante el conflicto sobre si un proyecto puede considerarse dirigido por un plan o ágil. → software ejecutable que cubra sus necesidades y realice funciones útiles.
- En la práctica, muchas compañías que afirman haber usado métodos ágiles adoptaron algunas habilidades ágiles y las integraron con sus procesos dirigidos por un plan.

### 3.1.3 Programación extrema

- Metodología basada en la comunicación, la reutilización del código desarrollado y la retroalimentación.
- El ciclo de entrega de la programación extrema



- En la programación extrema, los requerimientos se expresan como escenarios (llamados **historias de usuario**), que se **implementan** directamente como una **serie de tareas**. Los programadores **trabajan en pares** y **antes** de escribir el código, se **desarrollan las pruebas** para cada tarea.

# T3. 3.1 Desarrollo ágil de software. 3.1.3 Programación extrema

## 3.1.3 Programación extrema

- **Clientes:** determinan las prioridades y el alcance del proyecto. **Programadores.** **Testers:** ayudan al cliente sobre los requisitos del producto. **Coach:** asesoran a los demás miembros del equipo e identifican el siguiente paso a seguir. **Manager:** gestiona los recursos, el proyecto, se encarga de la comunicación.
- Por una parte se deben planificar los plazos temporales del proyecto basándose en las exigencias del **cliente**.
  - En base al coste y la complejidad del proyecto se marcan las prioridades y se definen fechas (muchas veces de forma orientativa).
  - Se desarrolla la iteración cada dos semanas, se marca el rumbo a seguir y se entrega el **software útil** después de cada uno de estos periodos bisemanales.
- Se han de efectuar una serie de **pruebas automatizadas** en base a los requisitos del cliente para comprobar que todo funcione correctamente.
  - Éstas han de hacerse **de forma periódica y automática**.
- La programación del **software** se hace siempre en pareja
  - Se asegura con este método que al menos un programador conoce y controla el modelo y la otra persona, realiza la revisión como mínimo.
- **El código es de todos**, con el desarrollo de las pruebas automáticas y la programación por pares se incluye también la posibilidad de que cualquiera pueda añadir y retocar parte del código.

# T3. 3.1 Desarrollo ágil de software. 3.1.3

## Programación extrema

### Prácticas de programación extrema

Principio/Práctica	Descripción
<b>Planificación incremental</b>	Los <b>requerimientos</b> se registran en tarjetas de historia ( <b>story cards</b> ) y las historias que se van a incluir en una liberación se determinan por el tiempo disponible y la prioridad relativa
<b>Entregas pequeñas</b>	<b>Al principio se desarrolla el conjunto mínimo de funcionalidad útil</b> , que ofrece valor para el negocio.
<b>Diseño simple</b>	Se realiza un diseño suficiente para cubrir sólo aquellos requerimientos actuales.
<b>Desarrollo de la primera prueba</b>	<b>Se usa un marco de referencia de prueba de unidad automatizada al escribir las pruebas para una nueva pieza de funcionalidad</b> , antes de que esta última se implemente.
<b>Reconstrucción (Refactorización)</b>	Se espera que todos los desarrolladores reconstruyan de manera continua el código y, tan pronto como sea posible, se encuentren mejoras de éste. Lo anterior conserva el código simple y fácil de mantener.
<b>Programación en pares</b>	<b>Los desarrolladores trabajan en parejas, y cada uno comprueba el trabajo del otro</b> ; además, ofrecen su apoyo para asegurar que siempre se entregue un buen trabajo.
<b>Propiedad colectiva</b>	<b>Los desarrolladores en pares trabajan en todas las áreas del sistema</b> , de manera que no se desarrollan islas independientes de experiencia, ya que todos los desarrolladores se responsabilizan de todo el código.
<b>Integración continua</b>	<b>Tan pronto como esté completa una tarea, se integra en todo el sistema</b> . Después de tal integración, deben aprobarse todas las pruebas de unidad en el sistema.
<b>Ritmo sostenible</b>	<b>No se consideran aceptables grandes cantidades de tiempo extra</b> , pues el efecto neto de este tiempo con frecuencia se traduce en reducir la calidad del código y la productividad.
<b>Cliente in situ</b>	<b>Un representante del usuario final del sistema (el cliente) tiene que disponer de tiempo completo para formar parte del equipo XP</b> . En un proceso de programación extrema, el cliente es miembro del equipo de desarrollo y responsable de llevar los requerimientos.

# T3. 3.1 Desarrollo ágil de software. 3.1.3

## Programación extrema

- Las **historias de usuario o User Stories** son partes del comportamiento que va a llevar a cabo un sistema de software.
  - Se utilizan en enfoques de software ágiles para dividir una gran cantidad de funcionalidad en partes más pequeñas para planificar.
  - También se entiende como "historia" o "historia de usuario" a una funcionalidad a desarrollar, entendido en los entornos Agile.
- Su objetivo es identificar funcionalidad útil que pueda desarrollarse en dos semanas aproximadamente, cuando la siguiente entrega del sistema esté preparada para el cliente.
- **Durante la planificación**, salen a la luz preguntas que no pueden responderse fácilmente y se requiere trabajo adicional para explorar posibles soluciones.
- El equipo puede elaborar algún **prototipo** o tratar de desarrollarlo para entender el problema y la solución.
  - En términos XP, **éste es un “pico” (spike)**, es decir, un incremento donde no se realiza programación.
- **Se construyen nuevas versiones de software varias veces al día y las versiones se entregan a los clientes aproximadamente cada dos semanas.**
  - **No se diseña, se realiza el cambio.**

## T3. 3.1 Desarrollo ágil de software. 3.1.3

### Programación extrema

- Un problema general con el desarrollo incremental es que **tiende a degradar la estructura del software**, de modo que los cambios al software se vuelven cada vez más difíciles de implementar.
- La programación extrema aborda este problema al sugerir que el software debe reconstruirse continuamente. **Controla la degradación del desarrollo.**
- Algunas características y cambios nuevos no pueden ajustarse con facilidad al reconstruir el código y al requerir modificar la arquitectura del sistema.
  - **A veces no es posible volver a desarrollar y hay que modificar la arquitectura.**

#### Pruebas en XP

- Con el desarrollo incremental, **no hay especificación de sistema de la que pueda partir un equipo de prueba externo** para desarrollar **pruebas del sistema**.
- En consecuencia, algunos enfoques del desarrollo incremental tienen un **proceso de pruebas muy informal**, comparado con las pruebas de un proyecto tradicional.

### Product Owner

- Única persona responsable de maximizar el retorno de la inversión (ROI) del esfuerzo de desarrollo.
- Responsable de la visión del producto.
- Decide si se debe liberar (entregar) o continuar con el desarrollo.

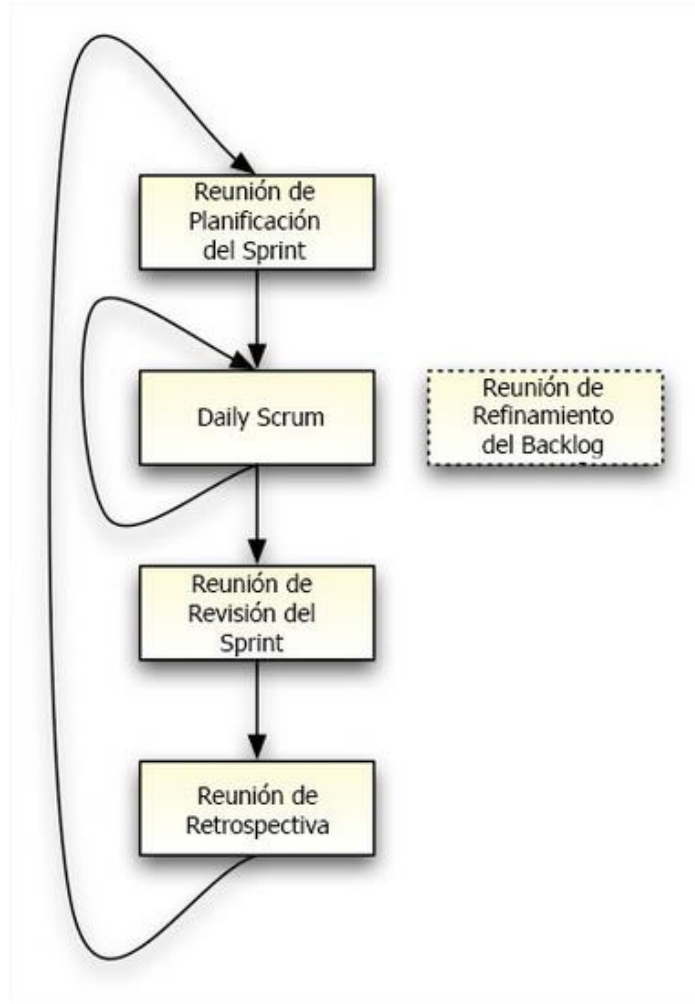
### Equipo de Desarrollo de Scrum

- Multifuncional (incluye miembros con habilidades de testing y a menudo otros no llamados tradicionalmente desarrolladores: analistas de negocio, expertos de dominio, etc.)
- Auto-organizado/ auto-gestionado, sin roles asignados externamente.
- Negocia los compromisos con el Product Owner, de un Sprint a la vez.

### Scrum Master (facilitador)

- Facilita el proceso de Scrum.
- Crea un ambiente propicio para la organización del equipo
- Captura datos empíricos para ajustar las previsiones

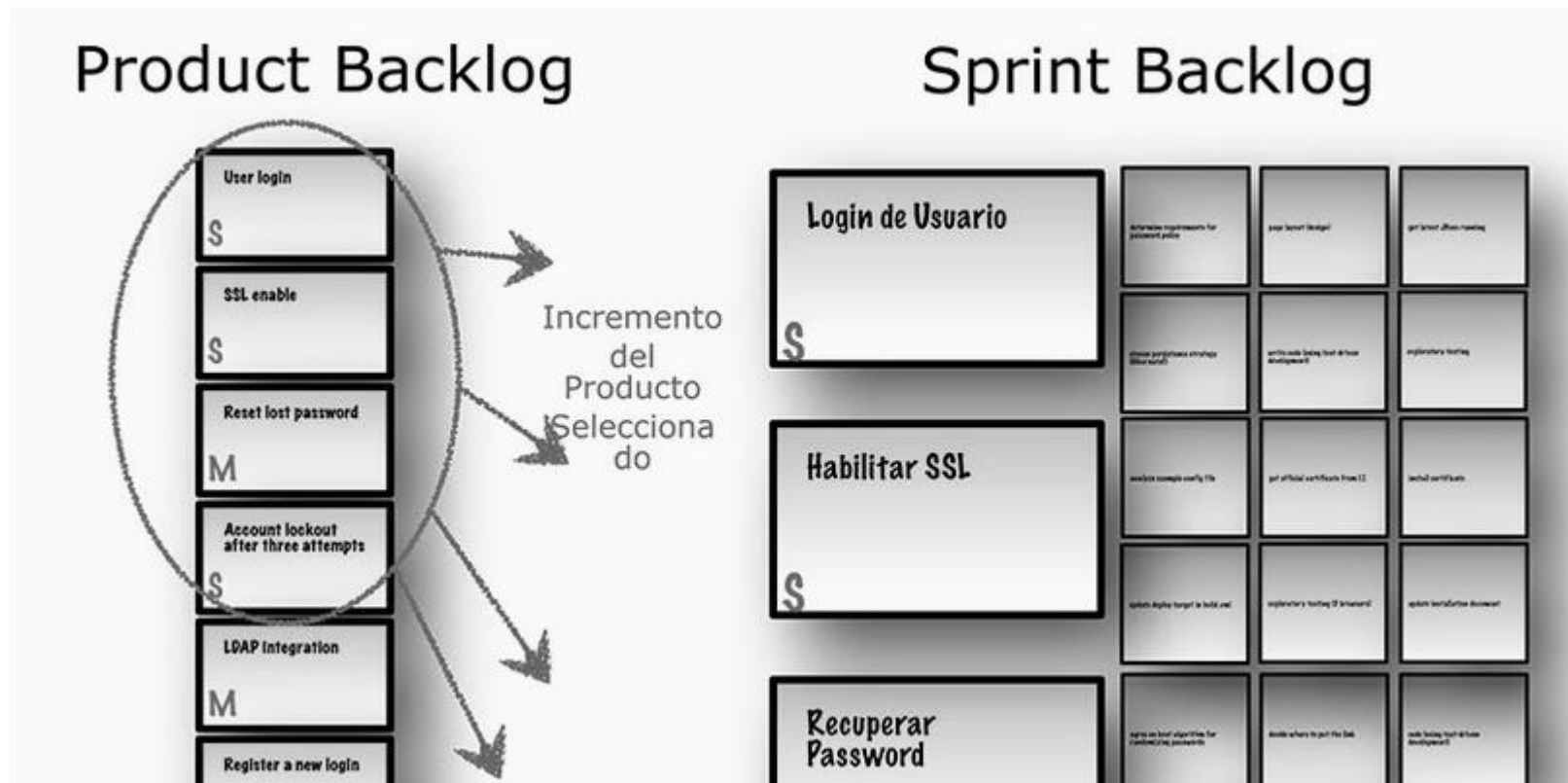
## 1. Reunión de Planificación de Sprint



Al comienzo de cada Sprint, el Product Owner y el equipo tienen una Reunión de Planificación del Sprint donde negocian qué elementos del Backlog de producto intentarán desarrollar durante el Sprint.

El Product Owner es el responsable de declarar cuáles son los ítems más importantes para el negocio. El equipo es responsable de seleccionar la cantidad de trabajo que cree que podrán realizar sin acumular deuda técnica.

Si el equipo no es capaz de desarrollar un incremento con funcionalidad correcta en un Sprint, debe reducir la cantidad de funcionalidad comprometida.



## 2. Daily Scrum y Ejecución del Sprint

Cada día, en el mismo lugar y a la misma hora, los miembros del equipo de desarrollo se reúnen durante 15 minutos reportándose entre sí.

Cada miembro del equipo resume lo que hizo el día anterior, lo que hará hoy, y qué problemas ha encontrado. Mantenerse de pie en el Daily Scrum ayuda a que sea breve.



### 3. Reunión de Revisión del Sprint

Después de la ejecución del Sprint, el equipo mantiene una reunión de revisión del Sprint para mostrar una evolución del producto al Product Owner y a los demás interesados.

La reunión debe mostrar una demo

El Product Owner **revisa los compromisos** de la Reunión de Planificación del Sprint y decide qué ítems considera terminados.

El **Scrum Master ayuda al Product Owner y a los stakeholders** a convertir su feedback en los nuevos Ítems del Product Backlog.

### 4. Reunión de Retrospectiva del Sprint

**Cada Sprint finaliza con una retrospectiva.**

El equipo reflexiona sobre su propio proceso.

Inspeccionan el comportamiento y toman medidas para posibles cambios en futuros Sprints.

Una vez que un equipo ha resuelto los problemas encontrados, el Scrum Master trabaja para facilitar el proceso de cambio

## 5. Reunión de Refinamiento del Backlog

El equipo estima la cantidad de esfuerzo que van a dedicar para completar los ítems del Backlog de Producto y proporciona información técnica para ayudar al Product Owner a priorizarlos.

La reunión de refinamiento también se llama “Backlog Grooming”, “Mantenimiento del Backlog” o “Story Time”.

### Backlog de producto

Lista ordenada de funcionalidad























Ítems

### Ítems del Backlog del Producto

Especifica el qué y el cómo



### Backlog del Sprint

Ítems del Backlog Comprometidos	Tareas No Iniciadas	Tareas En Progreso	Tareas Completadas
	  		 
	  		
	     		
			

Ítems comprometidos entre el equipo y el Product Owner en la reunión de Planificación del Sprint

## T3. 4. Scrum

### Tareas del Sprint

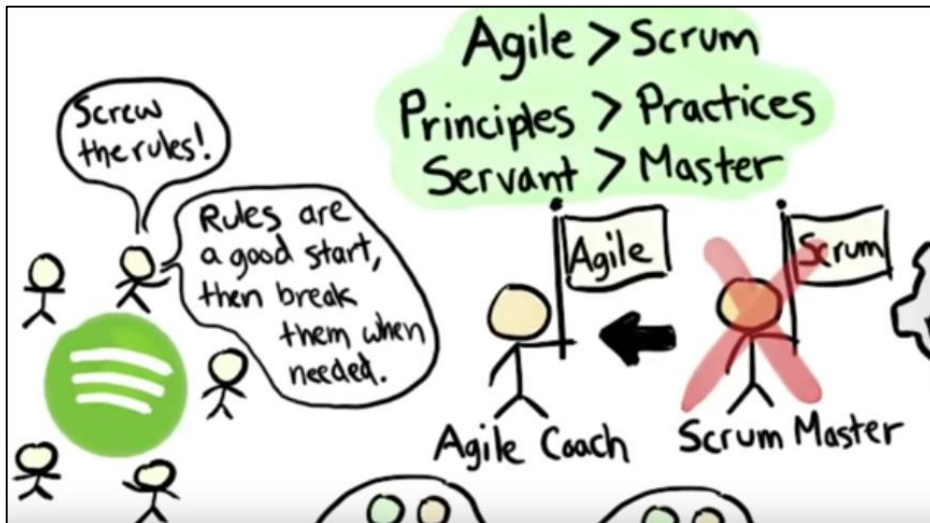
Especifica qué alcanzar y cómo lograr el ítem comprometido

El esfuerzo se re-estima a diario, por lo general en horas

En la ejecución del Sprint, una persona se ofrece como responsable de la tarea

**Spotify Engineering Culture (by Henrik Kniberg)**

<https://www.youtube.com/watch?v=4GK1NDTWbkY>



**Agile way of working at ING Belgium**

<https://www.youtube.com/watch?v=TaV-d7eKWFc&list=RD4GK1NDTWbkY&index=5>