

Metodologías

Ingeniería del Software Avanzada
Proceso software

Concepto

- “Conjunto integrado de técnicas y métodos que permiten obtener la forma homogénea y abierta, cada una de las actividades del ciclo de vida del SW (Hilera et al., “Metodología de desarrollo de Sistemas de Información”, UAH, 1996)
 - Método. Actividades llevadas a cabo para conseguir un objetivo. Ejemplo: método de análisis, de diseño,...
 - Integrado. Significa que las técnicas se utilizan como parte de los métodos.
 - Técnicas. Utilizadas en la aplicación de un método.
 - Homogénea. Sistemática, que se utilice en todos los proyectos de una organización. No debería definirse una nueva metodología para cada proyecto.
 - Abierta. A cambios y adaptación según el proyecto que se va a llevar a cabo.

Concepto

- Una metodología debe definir:
 - Cómo dividir un proyecto en etapas
 - Qué trabajos se llevan a cabo en cada etapa.
 - Qué salida se produce en cada etapa, y cuando se debe producir. Las salidas normalmente son documentos y productos SW.
 - Qué técnicas y herramientas se van a utilizar en cada etapa. Normalmente se utilizan técnicas estándar
 - Qué restricciones se aplican (de tiempo, coste, objetivos, etc.)
 - Cómo se gestiona y controla un proyecto

Diferencias con ciclo de vida

- Ciclo de vida es más abstracto:
 - Se refiere sólo al esquema de fases y tareas
 - Es una definición específica de los procesos
 - Suele incluir sólo el Qué hacer
- Una metodología:
 - Puede seguir uno o varios modelos de ciclo de vida
 - Además suele incluir referencias y guías más explícita sobre cómo aplicar métodos, técnicas y herramientas.
 - Incluye el Cómo hacer

Tipos

- Por su aplicabilidad:
 - Metodologías oficiales (de iure)
 - Métrica v3: España
 - SSADM: Reino Unido
 - Metodologías no oficiales (de facto)
 - Unified process
 - Yourdon Structured Analysis
- Por su tecnología:
 - Estructuradas
 - Orientadas a objetos, orientadas a componentes
 - Web
- Por su gestión del proceso:
 - Pesadas
 - Métrica V3
 - Ágiles
 - XP, Scrum

Metodologías estructuradas

- Es la aplicada desde mediados de los años 70.
- Idea:
 - Software se compone de funciones que procesan datos
- Propone crear modelos del sistema que representen de manera descendente los siguientes aspectos:
 - Las funciones (también llamadas “procesos”) del sistema
 - Los flujos de datos de entrada, salida e internos de cada función
 - La estructura de los datos procesados por las funciones del sistema.
- Metodologías: Metrica, SSADM, Merise

Técnicas estructuradas

- **Análisis:**
 - Diagrama de Flujo de Datos
 - Diagrama de Entidad Relación
 - Matriz de funciones-entidades
- **Diseño:**
 - Diagrama de Estructura Modular (o “de Constantine”)
 - Diagrama de Módulos (o “de Jackson”)
 - Diagrama de Estructura Lógica de un programa (o “de Warnier”)
 - Diagrama de estructura de datos (LDS)
 - Diagrama de tablas (de una Base de Datos)
 - Diagrama de flujo de control (flowchart, organigrama, etc.) y pseudocódigo
- **Programación estructurada:**
 - Estructuras de control (secuencia, decisión (if), repetición (loop, for, while, repeat,..)
 - Subprogramas: programación modular
 - Lenguajes de programación estructurada: COBOL, FORTRAN, C, PASCAL, NATURAL, PL/SL(Oracle).

Metodologías Orientadas a Objetos. Ejemplo RUP

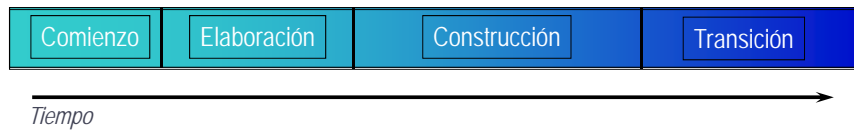
Orientación a objetos

- Se empieza a aplicar a finales de los años 80.
- Se basa en la idea de que un sistema software se compone de objetos software que interactúan entre sí.
- La funcionalidad de un sistema se reparte entre los objetos, asignando a cada objeto funciones específicas.
- El objetivo final de la ISOO es construir software de la misma forma mediante el ensamblaje de componentes.
- Las ventajas de la orientación a objetos son:
 - **Facilidad para la reutilización del software**
 - **Simplificación del mantenimiento del software**
- Metodologías: MÉTRICA 3 (España), UNIFIED PROCESS (Rumbaugh, Booch y Jacobson), FUSION (Hewlett-Packard), OPEN (Henderson-Sellers)

Técnicas OO

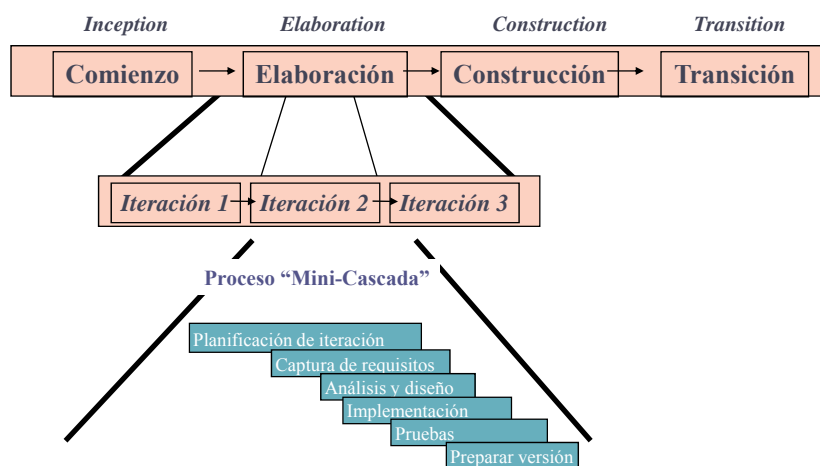
- **Análisis (www.uml.org) :**
 - Diagrama de Casos de Uso
 - Diagrama de Actividades
 - Diagramas de interacción: Secuencia y Colaboración
 - Diagrama de Estados
- **Diseño (www.uml.org)**
 - Diagrama de Componentes
 - Diagrama de Despliegue
 - Patrones de diseño (independientes de UML)
- **Programación OO**
 - Herencia: Objetos basados en otros objetos
 - Polimorfismo: Funciones con el mismo nombre
 - Agregación: Objetos que contienen objetos
 - Lenguajes de programación orientada a objetos: JAVA, C++, C#, VISUAL BASIC, SMALLTALK, EIFFEL.

Fases de RUP

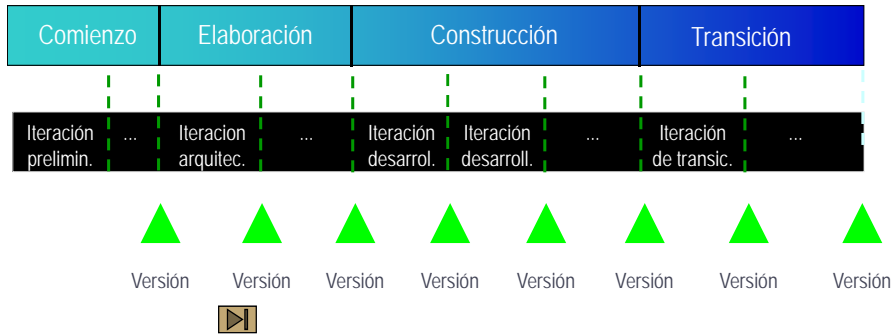


- Comienzo: define el enfoque del proyecto y desarrolla un análisis de negocio (*business case*)
- Elaboración: planear proyecto, especificar características y configuración de arquitectura
- Construcción: programación y prueba
- Transición: traspasar producto a sus usuarios

Proceso unificado



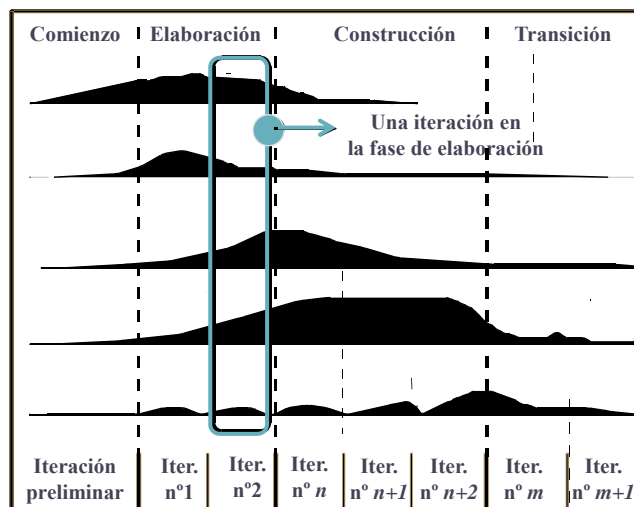
Fases e iteraciones



Fases

Flujos de trabajo

- Requisitos
- Análisis
- Diseño
- Implementación
- Pruebas



Claves del proceso iterativo

- Integración continua
 - No se hace en un momento cercano a la fecha de entrega
- Versiones ejecutables, frecuentes
 - Algunas internas; algunas para entregar
 - Regularidad: rompe síndrome 90% acabado con 90% por acabar
- Aborda los riesgos a través de una progresión demostrable
 - El progreso se mide en productos, no en documentación o en estimaciones de ingeniería
- Incorpora problemas/elementos/cambios:
 - En futuras iteraciones más que interrupción de producción en marcha

Metodologías Orientadas a Componentes

¿Qué es la ISBC?

- Proceso para diseño y construcción de sistemas que utilizan componentes de software reutilizables.
- Componente:
 - Parte HW o SW de un sistema (IEEE std. 610)
 - “Una parte reemplazable, casi independiente y no trivial de un sistema que cumple una función clara en el contexto de una arquitectura bien definida”
 - Los componentes se identifican por las características de su interfaz (servicios)
- La ISBC se parece bastante a la OO:
 - Análisis de requisitos y diseño de alto nivel
 - ¿Hay componentes comerciales ya desarrollados (COTS) para cada requisito? ¿Se dispone de componentes reutilizables desarrollados internamente? ¿Son compatibles las interfaces de los componentes disponibles dentro de la arquitectura del sistema a construir?

Elementos de apoyo

- Patrones (*patterns*):
 - Existen patrones específicos para mejorar la reutilización y para un diseño y composición apropiada de componentes
- Marcos de trabajo (*frameworks*):
 - Diseño de subsistemas compuesto de colección de clases abstractas o concretas y las interfaces entre ellas.
 - Una aplicación semicompleta con componentes estáticos y dinámicos personalizables para aplicaciones específicas.
 - Un esquema para el desarrollo y/o la implementación de una aplicación

Frameworks

- **Diferencias con aplicaciones:**
 - Frameworks son soluciones genéricas, incompletos, no cubren toda la funcionalidad sino que abstraen diseño común
- **Diferencias con patrones:**
 - Frameworks ejecutables e implementados; los patrones son más generales y abstractos, más pequeños y menos especializados
- **Diferencias con librerías:**
 - Framework no es solo un conjunto de clases (librería), define una estructura de aplicación. "el framework llama a tu código" es distinto a "tu código llama a la librería"

Metodologías Web

Ing. de software web

- **Publicación del estándar IEEE Std 2001-2002:**
 - Software Engineering — Recommended Practice for the Internet — Web Site Engineering, Web Site Management, and Web Site Life Cycle
- **Se trata de un tipo de ingeniería del software orientada a la naturaleza multidimensional de las aplicaciones web, que implican, además de programación:**
 - Definición de estructuras complejas de información (XML, ...)
 - Diseño de estructuras de navegación
 - Gestión de contenidos
 - Diseño gráfico
 - Gestión de seguridad
- **Metodologías:**
 - OPEN (Henderson-Sellers), HDM (Hypermedia Design Methodology), OOHDM (Object Oriented HDM), RMM (Relationship Management Methodology), EORM (Enhanced Object Relationship Model)

Indicaciones para web

- **Inclusión de aspectos importantes para web:**
 - Estudio de la navegación y multimedia (tanto interfaz como contenidos)
 - Análisis de grupos de usuarios (universo de usuarios amplio y variado):
 - Accesibilidad
 - Usabilidad (facilidad de uso)
 - Estandarización
 - Documentar bien la implementación. Mantenimiento complejo y con necesidad de respuesta rápida
- **Variaciones en los equipos de trabajo:**
 - Diseñador web: combina la parte gráfica con la de código para implementar
 - Editor web: responsable de los resultados de web
 - Diseñador gráfico: control y desarrollo de elementos gráficos y multimedia

Metodologías pesadas. Ejemplo Métrica V3

Metodologías pesadas

- Se centran en planificar y documentar cada tarea
- Mucho esfuerzo de gestión
- Rígidas respecto a cambios en los requisitos durante el desarrollo
- Cada tarea lleva asociados entregables y responsables de distintas acciones
- Predisposición a contratos, estándares y auditorías
- Facilitan el mantenimiento (intentan minimizarlo definiendo requisitos y arquitecturas con visión de futuro)

Estructura de Métrica V3

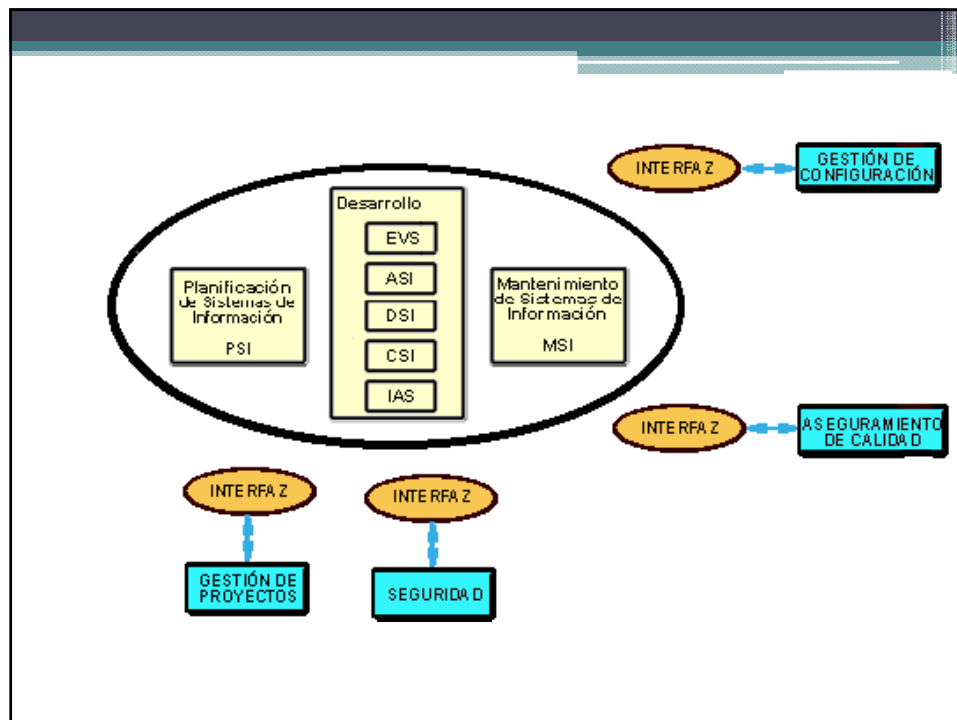
Es una única estructura la metodología MÉTRICA Versión 3 cubre desarrollo estructurado y orientado a objetos, y facilita a través de interfaces la realización de los procesos de apoyo u organizativos. Los procesos se dividen en Principales e Interfaces.

Cada Proceso detalla las Actividades y Tareas a realizar. Para cada tarea se indican:

- Las técnicas y prácticas a utilizar
- Los responsables de realizarla
- Sus productos de entrada y salida

Patrón de tarea





Metodologías Ágiles. Ejemplos XP y SCRUM

Metodologías ágiles

- **Manifiesto ágil:**
 - Varios críticos (K.Beck) de las metodologías tradicionales publican en 2001 este manifiesto de nuevos valores para el desarrollo de software
- **Contenido** (<http://www.agilemanifesto.org/>)
 - Descubrimos mejores métodos para desarrollar software, haciéndolo y ayudando a otros a que lo hagan. Con este trabajo hemos llegado a valorar más el lado izquierdo aunque hay valor en elementos de la derecha

Ágil	Tradicional
Los individuos y su interacción	Los procesos y las herramientas
El software que funciona	La documentación exhaustiva
La colaboración con el cliente	La negociación contractual
La respuesta al cambio	El seguimiento de un plan

Principios

1. La prioridad principal es satisfacer al cliente mediante tempranas y continuas entregas de software que le den valor
2. Dar la bienvenida a los cambios. Los métodos ágiles capturan los cambios para que el cliente tenga una ventaja competitiva
3. Entregar frecuentemente software que funcione, desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre una entrega y la siguiente
4. La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto
5. Construir proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos
6. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo
7. A intervalos regulares, el equipo reflexiona cómo llegar a ser más efectivo, y según esto ajusta su comportamiento

Comparación Ágil - NoÁgil

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
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

Principales métodos ágiles

- Crystal Methodologies, Alistair Cockburn:
 - https://en.wikiversity.org/wiki/Crystal_Methods
- **SCRUM**, Ken Schwaber & Jeff Sutherland:
 - www.controlchaos.com
- DSDM (Dynamic Systems Development Method):
 - <https://www.agilebusiness.org/resources/dsdm-handbooks/the-dsdm-agile-project-framework-2014-onwards>
- Lean Programming, Mary Poppendieck:
 - www.poppendieck.com
- Extreme Programming, Kent Beck:
 - www.extremeprogramming.org
- Adaptive Software Development, Jim Highsmith:
 - www.adaptivesd.com

Roles XP

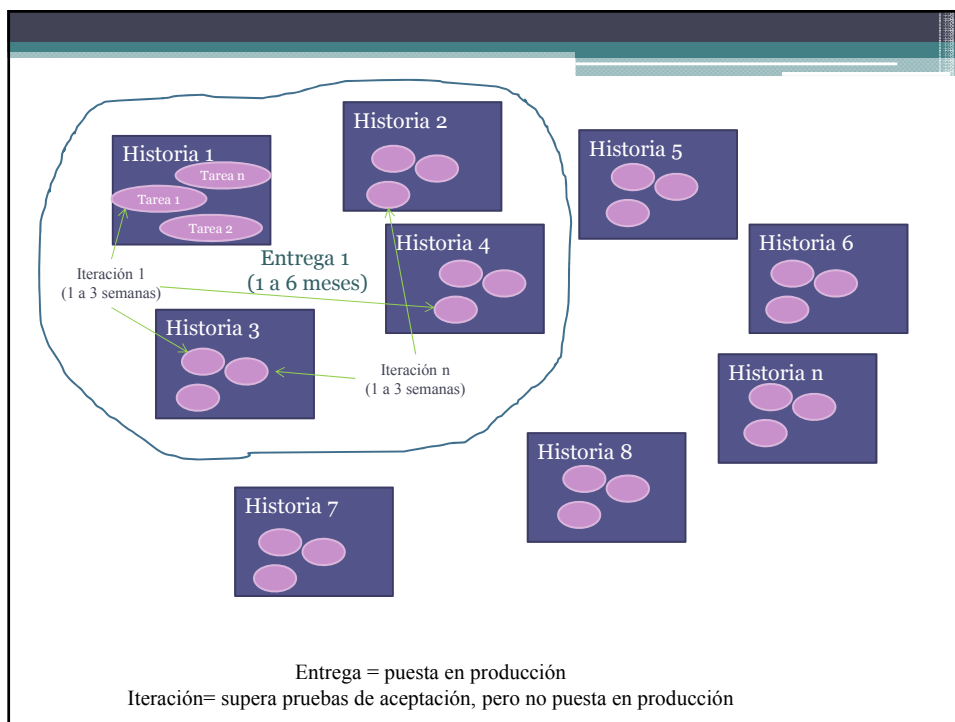
- **Programador (*Programmer*)**
 - Responsable de decisiones técnicas
 - Responsable de construir el sistema
 - Sin distinción entre analistas, diseñadores o codificadores
 - En XP, los programadores diseñan, programan y realizan las pruebas
- **Encargado de Pruebas (*Tester*)**
 - Ayuda al cliente con las pruebas funcionales
 - Se asegura de que las pruebas funcionales se superan
- **Rastreador (*Tracker*)**
 - Metric Man
 - Observa sin molestar
 - Conserva datos históricos
- **Jefe de Proyecto (*Manager*)**
 - Organiza y guía las reuniones
 - Asegura condiciones adecuadas para el proyecto
- **Cliente (*Customer*)**
 - Parte permanente del equipo
 - Determina qué construir y cuándo
 - Establece pruebas funcionales
- **Entrenador (*Coach*)**
 - Responsable del proceso
 - Tiende a estar en un segundo plano a medida que el equipo madura

Captura de Requisitos en XP

- **Historias del Usuario (*User-Stories*)**
 - Establecen los requisitos del cliente
 - Trozos de funcionalidad que aportan valor
 - Se les asignan tareas de programación con un n° de horas de desarrollo
 - Las establece el cliente
 - Son la base para las pruebas funcionales
- **Muy similares a los escenarios de casos de uso**

Planificación en XP

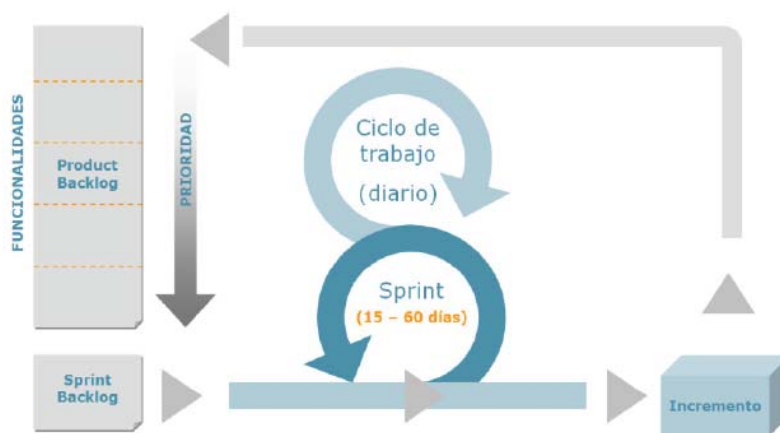
- Planificación por entregas al cliente (*releases*)
- Se priorizan aquellas *user-stories* que el cliente selecciona porque son más importantes para el negocio
- Entregas:
 - Son lo más pequeñas posibles
 - Se dividen en iteraciones (iteración = 2 o 3 semanas)
 - Están compuestas por historias
- A cada programador se le asigna una tarea de la *user-story*



Prácticas XP

- El cliente debe estar permanentemente con el quipo
 - Decisiones del cliente: prioridad, composición y fecha de las entregas
 - Decisiones técnicas (programadores y otros): estimaciones de tiempo, organización del proceso de trabajo, planificación detallada (dentro de una entrega)
- Reunión diaria de todo el equipo, de pie, en círculo
- El código es de todos, por tanto es imprescindible:
 - Estándares y normas de estilo de programación
- Todo el código se escribe en parejas: uno programa, otro revisa y asesora y a la inversa (*pair-programming*)
- Pruebas (se diseñan antes de codificar): unitarias e integración, el programador (mejor automatizadas); funcionales, el cliente
- Existe un ordenador para la integración

SCRUM



SCRUM. Definiciones

- **Product backlog. Requisitos del sistema.**
 - Inventario de características y funciones que el propietario del producto desea obtener, ordenado por prioridad.
 - Es un documento “vivo”, en constante evolución.
- **Sprint Backlog.**
 - Lista de los trabajos que realizará el equipo durante el sprint para generar el incremento previsto.

SCRUM. Reuniones

Reunión diaria. Cada miembro responde a tres preguntas:

1. Trabajo realizado ayer
2. Trabajo que se va a realizar hoy
3. Impedimentos para desarrollar el trabajo

→ **SEGUIMIENTO DEL SPRINT**

Se genera la “sprint backlog” o lista de tareas que se van a realizar, y el “objetivo del sprint”: lema que define la finalidad de negocio que se va a lograr.

→ **PLANIFICACIÓN DEL SPRINT**



Ciclo de trabajo

Sprint
(15 – 30 días)

Presentación de lo realizado. Análisis y revisión del incremento generado

→ **REVISIÓN DEL SPRINT**



Incremento

SCRUM. Roles

- **Propietario del producto.**
 - Una persona, y sólo una, conocedora del entorno de negocio del cliente y de la visión del producto.
 - Representa a todos los interesados en el producto final y es el responsable del Product Backlog.
- **Responsabilidad del desarrollo: El equipo**
 - Multidisciplinar que cubre todas las habilidades
 - Se auto-gestiona y auto-organiza
- **Scrum Manager**
 - Responsabilidad de funcionamiento del modelo.
 - A nivel de proyecto o a nivel de organización
 - En algunos casos un rol exclusivo (Scrum Master) y en otros, varias personas (responsables de calidad o procesos o de gestión de proyectos...)

Métodos ágiles

- Actualmente ya hay bastantes experiencias para analizar sus resultados
- **Ventajas y limitaciones:**
 - Beneficios en equipos y aplicaciones pequeñas que requieren muchos cambios y dinámica
 - Problemas de gestión en aplicaciones más grandes
 - Buenos resultados necesitan muy buenos profesionales
- **Bibliografía:**
 - B.Meyer. *Agile! The Good, the Hype and the Ugly*. Springer, 2014

Nuevos desafíos: Seguridad en Systems Development Life Cycle (SDLC) - I

- Microsoft Security Development Lifecycle (Microsoft SDL)
 - Contiene una colección de actividades de seguridad agrupada por las fases del SDLC tradicional.
- Seven Touchpoints for Software Security
 - Conjunto de mejores prácticas. Han sido adoptadas por el Departamento de Seguridad Nacional de EEUU y por Erns and Young entre otros.

Nuevos desafíos: Seguridad en Systems Development Life Cycle (SDLC) - II

- Oracle Software Security Assurance (OSSA)
 - Metodología de Oracle para abordar la seguridad en las fases de diseño, codificación, pruebas y mantenimiento para productos Oracle y su nube con el fin de evitar problemas de seguridad derivados en parches.
- NIST 800-64-Seguridad en el ciclo de vida de desarrollo software
 - Se describen las funciones y responsabilidades clave de seguridad en cada fase de su ciclo de vida.
 - No es una metodología flexible, ya que se proyecta principalmente para organizaciones gubernamentales.