Ingeniería del software: procesos, ciclo de vida y enfoques ágiles

1. Introducción a la Ingeniería del Software

¿Qué es la ingeniería del software?

La **ingeniería del software** es la disciplina que aplica principios de ingeniería, gestión y buenas prácticas al desarrollo de programas informáticos. Su objetivo es **producir software de alta calidad**, dentro de **plazos y costes razonables**, y que sea **mantenible y fiable** a lo largo del tiempo.

El software no solo son líneas de código: implica **procesos, documentación, pruebas, mantenimiento y evolución**.

Proceso software y ciclo de vida del software

El **proceso software** es el conjunto de **actividades, métodos, prácticas y transformaciones** que permiten desarrollar y mantener un sistema software.

El **ciclo de vida del software** describe **todas las etapas** por las que pasa un producto software desde que se concibe hasta que deja de utilizarse. Estas etapas pueden organizarse de diferentes maneras según el modelo de desarrollo que se

adopte.

Fases del desarrollo de una aplicación

Aunque cada metodología puede adaptarlas, tradicionalmente se distinguen las siguientes **fases principales**:

1. **Análisis de requisitos**
   - Se identifican las **necesidades del cliente o usuario**.
   - Se definen los **requisitos funcionales** (qué debe hacer el sistema) y **no funcionales** (rendimiento, seguridad, usabilidad…).
   - Se obtiene un documento de especificación de requisitos.

2. **Diseño**
   - Se traduce lo anterior a una **arquitectura técnica y funcional** del sistema.
   - Incluye diseño de **bases de datos**, **interfaces**, **componentes** y **módulos de software**.
   - Objetivo: preparar una estructura sólida antes de programar.

3. **Codificación (implementación)**
   - Se transforma el diseño en **código fuente**.
   - Aquí los programadores desarrollan los módulos en uno o varios lenguajes.
   - Se aplican **buenas prácticas de programación** y **control de versiones**.

4. **Pruebas (testing)**
    - Se verifica que el software **funciona correctamente** y cumple los requisitos.
    - Tipos: pruebas unitarias, de integración, de sistema y de aceptación.
    - El objetivo es detectar y corregir errores antes de la entrega.

5. **Documentación**
    - Es transversal a todo el proceso.
    - Incluye **manuales técnicos**, **manuales de usuario**, **documentación de código**, etc.
    - Facilita la **mantenibilidad** y la **transferencia de conocimiento**.

6. **Explotación (implantación o despliegue)**
    - Se instala y pone en marcha el software en su entorno real.
    - Puede incluir **formación a usuarios** y **configuración del sistema**.

7. **Mantenimiento**
    - Es la fase más larga.
    - Consiste en **corregir errores**, **mejorar prestaciones** y **adaptar el sistema** a nuevos requisitos o entornos.

□ Modelos de proceso de desarrollo

software

Los modelos de proceso indican **cómo se organizan las fases anteriores**. Los principales son:

1. **Modelo en cascada (Waterfall)**
   - Fases **secuenciales**, donde una debe completarse antes de comenzar la siguiente.
   - Ventajas: estructura clara, buena para proyectos con requisitos estables.
   - Inconvenientes: poca flexibilidad ante cambios; los errores se detectan tarde.

2. **Modelo iterativo**
   - El desarrollo se realiza en **ciclos repetidos (iteraciones)**.
   - Cada iteración produce una **versión mejorada** del software.
   - Permite **ajustar requisitos** y **revisar** después de cada ciclo.

3. **Modelo evolutivo**
   - El software se **construye progresivamente** según la retroalimentación del usuario.
   - Se generan **prototipos** y versiones sucesivas hasta alcanzar el producto final.
   - Muy útil cuando los requisitos **no están completamente definidos**.

Metodologías de desarrollo software

Concepto

Una **metodología de desarrollo** define **cómo aplicar** los modelos de proceso: qué pasos seguir, qué roles intervienen, qué artefactos se generan y qué herramientas se usan.

Objetivos

- Aumentar la **productividad** y **calidad del software**.
- Estandarizar y controlar el proceso.
- Reducir **errores y costes de mantenimiento**.
- Favorecer la **colaboración** entre equipos.

Características comunes

- División del trabajo en **fases o iteraciones**.
- **Roles definidos** (analista, programador, tester, etc.).
- **Documentación y entregables** por fase.
- **Control de calidad** en cada etapa.

Técnicas empleadas

- Diagramas UML (modelado de sistemas).
- Diseño orientado a objetos.
- Análisis estructurado.
- Gestión de versiones y control de cambios.

Tipos de metodologías

Podemos agruparlas en dos grandes familias:

1. **Metodologías tradicionales o pesadas**
    - Basadas en planificación rígida y documentación extensa (ej. Cascada, V-Model, RUP).
    - Priorizan **control y previsibilidad**.

2. **Metodologías ágiles**
    - Priorizan la **adaptación al cambio**, la **colaboración** y la **entrega rápida de valor**.
    - Ejemplos: **Scrum**, **Kanban**, **XP (Extreme Programming)**.

□□ Herramientas CASE (Computer Aided Software Engineering)

Son herramientas informáticas que **asisten en la automatización** de tareas del proceso de desarrollo.

Tipos:

- **Upper CASE**: ayudan en fases iniciales (análisis y diseño).
- **Lower CASE**: se centran en fases finales (codificación, pruebas, mantenimiento).
- **Integrated CASE**: cubren el ciclo de vida completo.

Ejemplos:

- **Enterprise Architect**, **Visual Paradigm**, **Rational Rose**, **StarUML**, **PowerDesigner**.

Beneficios:

- Mejoran la **productividad** y **consistencia**.
- Facilitan la **documentación automática**.
- Reducen **errores humanos**.
- Aumentan la **trazabilidad** del proceso.

⚡ 2. Metodologías Ágiles y Scrum en contraposición a las tradicionales

🧱 Enfoque tradicional

- Planificación **detallada al inicio** del proyecto.
- Requisitos **fijos y documentados**.
- El desarrollo es **secuencial**.
- Escasa flexibilidad ante cambios.
- Mayor burocracia y documentación.

📌 Ejemplo: modelo **cascada** o **RUP**.

🌱 Enfoque ágil

Surge como **respuesta a las limitaciones** de los modelos tradicionales.

Basado en el **Manifiesto Ágil (2001)**, que valora:

- **Personas e interacciones** sobre procesos y herramientas.

- **Software funcionando** sobre documentación exhaustiva.
- **Colaboración con el cliente** sobre negociación contractual.
- **Respuesta al cambio** sobre seguir un plan.

⚙ Características

- Desarrollo **incremental e iterativo**.
- Entregas frecuentes de **versiones funcionales**.
- **Equipos autoorganizados** y comunicación constante.
- **Revisión continua** de objetivos y prioridades.

Scrum: una metodología ágil concreta

Scrum es el marco ágil más popular. Se organiza en **sprints** (iteraciones cortas de 2-4 semanas) donde el equipo entrega una **versión funcional** del producto.

Roles:

1. **Product Owner**: representa al cliente y prioriza los requisitos (Product Backlog).
2. **Scrum Master**: facilita el proceso y elimina impedimentos.
3. **Equipo de desarrollo**: multidisciplinar y autoorganizado.

Elementos clave:

- **Product Backlog**: lista priorizada de funcionalidades.
- **Sprint Backlog**: tareas seleccionadas para el sprint.

- **Incremento**: versión funcional al final del sprint.

Eventos:

- **Sprint Planning**: planificación del sprint.
- **Daily Scrum**: reunión diaria de 15 minutos.
- **Sprint Review**: revisión del producto entregado.
- **Sprint Retrospective**: reflexión sobre el proceso y mejoras.

⚖ Comparación: Tradicional vs. Ágil (Scrum)

Aspecto	Tradicional (Cascada, RUP)	Ágil (Scrum, Kanban)
Planificación	Completa al inicio	Iterativa y adaptativa
Requisitos	Fijos	Evolutivos
Entregas	Una al final	Frecuentes (cada sprint)
Documentación	Extensa	Mínima y esencial
Comunicación	Formal	Directa y constante
Roles	Jerárquicos	Colaborativos
Cambio de alcance	Costoso	Natural y aceptado
Orientación	Procesos	Personas y valor entregado