UNIDAD 1: ELEMENTOS DE DESARROLLO DEL SOFTWARE

Yussef El Hassani Chairi Kachaf

1. TIPOS DE SOFTWARE:

De sistema: Interactúa directamente con el hardware (de bajo nivel, muy cerca del hardware).

De aplicación: Programas que están más lejos que el microprocesador y se valen del SO.

De desarrollo: Programas para hacer programas (VisualStudioCode, Eclipse, etc).

Firmware: Es software de sistema pero muy cercano al hardware (ej: cualquier tipo de código de programa que se ejecuta en un microcontrolador (periféricos: raton teclado etc..)

Drivers: Programa entre el SO y el firmware del hardware. Son los controladores de dispositivos y mandan instrucciones básicas al hardware del firmware.

1.1 RELACIÓN HARDWARE Y SOFTWARE

Disco duro: Almacena permanentemente los datos y archivos ejecutables.

CPU (Central Processing Unit): cerebro del ordenador lee y ejecuta instrucciones que están almacenadas en la ram.

Memoria RAM (Memoria principal): Retiene temporalmente el código binario de los archivos ejecutables y los datos necesarios

E/S: Recoge datos mediante la entrada, para posteriormente mostrar los resultados.

1.2 CÓDIGOS FUENTE, OBJETO Y EJECUTABLE

Código fuente: Es el código que ha desarrollado el programador.

Código objeto: Archivo binario no ejecutable, se basa en el código fuente y no es legible

Código ejecutable: Archivo binario ejecutable.

Byte code: código objeto que genera JAVA.

2. CICLO DE VIDA DE SOFTWARE

Ingeniería de software- disciplina que astucia principios y metodologías para el mantenimiento y desarrollo de sistemas software.

2. 1 DESARROLLO DE SOFTWARE

Fases principales

Análisis: Plantear los requisitos.

Diseño: Se descompone y organiza el sistema en elementos componentes que pueden ser desarrollados por separado. (**mock up**: es un diseño estático de una página web o aplicación que presenta muchos de sus elementos de diseño finales pero no es funcional.)

Codificación: Realización del código fuente (en lenguajes compilados, interpretados, de marca, etc)

Pruebas: conseguir que el programa funcione mal, para buscar los fallos y arreglarlos.

Mantenimiento:
Correctivo: Corregir los defectos una vez ocurrido el fallo.
Preventivo: Cuando se arregla el problema con antelación para prevenir fallos.
Perfectivo: Se mejoran las funcionalidades.
Evolutivo: Se añaden funcionalidades.
Adaptativo: Se adapta a nuevos entornos
.

2.2 Resultado tras cada fase

Análisis: Definición de requisitos del software

Diseño arquitectónico: Creación de documento de arquitectura de software

Diseño detallado: Especificación de módulos y funciones

Codificación: Desarrollo del código fuente

Pruebas de unidades: Evaluación de módulos funcionales

Pruebas de integración: Comprobación del sistema integrado

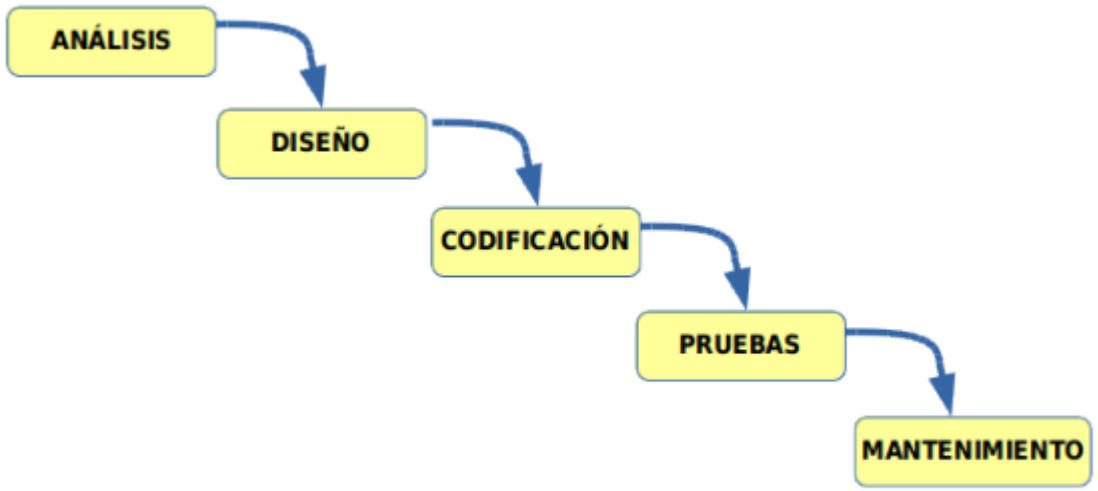
Pruebas del sistema: Validación del sistema completo

Documentación: Creación de documentación técnica y de usuario

Mantenimiento: Gestión de informes de errores y cambios

3. MODELOS DE DESARROLLO SOFTWARE

3. 1 MODELO EN CASCADA



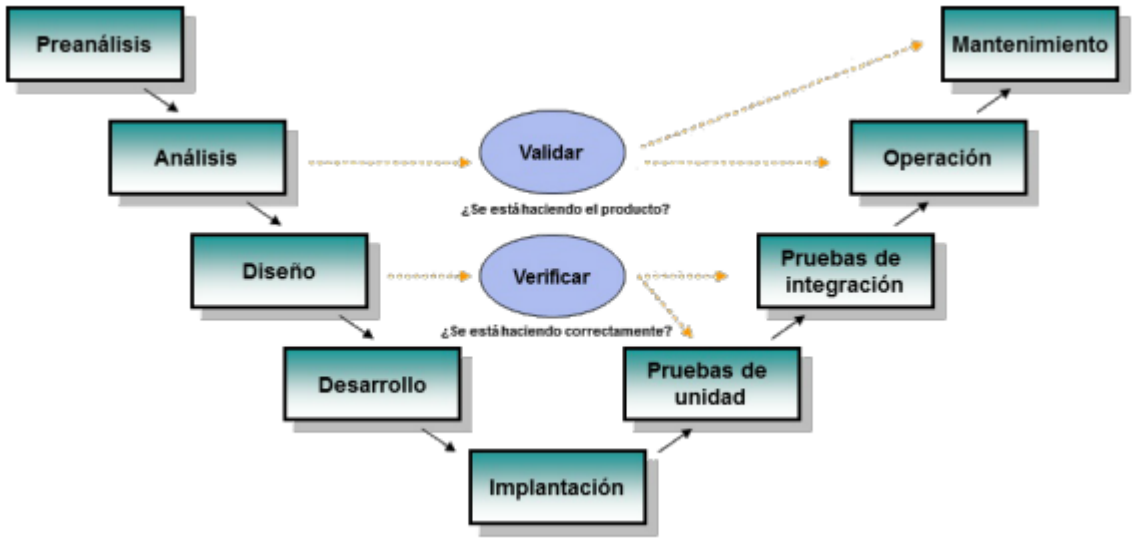
Ventajas:

- Estructura simple y fácil de comprender.
- Facilita la planificación y gestión del proyecto.

Inconvenientes:

- Poca flexibilidad para cambios en los requisitos.
- Riesgo de detectar problemas al final del ciclo, lo que puede resultar en costos significativos.

3.2 MODELO EN V



Ventajas:

- Enfoque estructurado que integra pruebas tempranas, lo que ayuda a detectar y corregir problemas de manera eficiente.

- Mayor visibilidad y control de las fases del proyecto, lo que facilita la gestión y asegura una alineación con los objetivos.

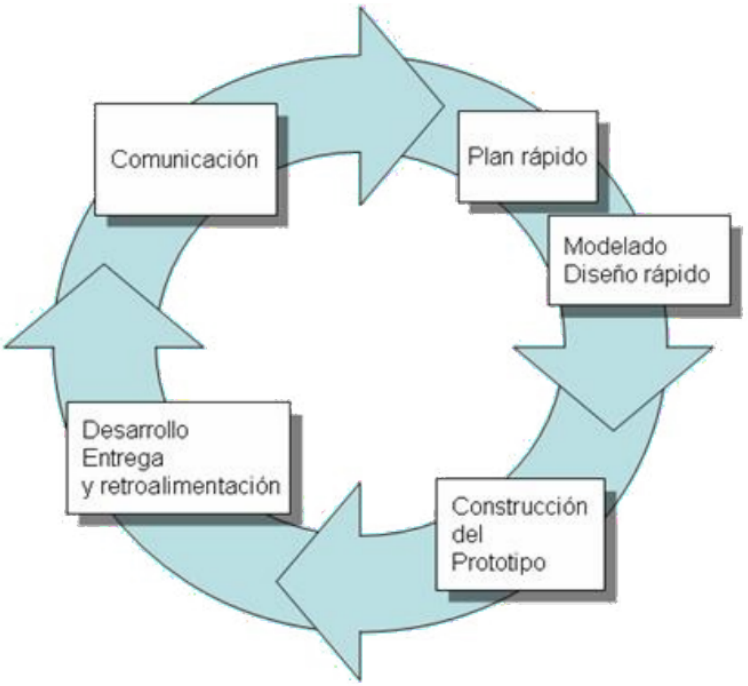
Inconvenientes:

- Menos flexibilidad para cambios de requisitos, ya que se espera que estén bien definidos desde el principio.
- Requiere una planificación detallada desde el inicio, lo que puede ser un desafío en proyectos donde los requisitos no son claros o cambian con frecuencia.

3.3 PROTOTIPOS

Los protótipos son **modelos iniciales de software** que se crean cuando los requisitos no están completamente definidos, ya sea debido a la falta de experiencia previa, omisiones o falta de claridad por parte del usuario/cliente.

Este proceso implica **crear un protótipo durante la fase de análisis**, probado por el usuario/cliente para refinar los requisitos del software.
Hay **dos tipos de protótipos**: **los rápidos**, que pueden desarrollarse con diferentes herramientas y lenguajes y finalmente descartarse, y **los evolutivos**, que están en el mismo lenguaje y herramientas del proyecto y se utilizan como base para el desarrollo del software.



3.4 MODELO EN ESPIRAL (I)

3.5 MODELO EN ESPIRAL (II)

3.6 METODOLOGÍAS ÁGILES

- Son métodos de ingeniería del software basados en el desarrollo iterativo, es decir, yendo poco a poco.

- El trabajo se organiza en pequeños grupos multidisciplinarios (un equipo debe de ser autónomo, es decir, sus componentes deben de tener conocimientos en todos los aspectos).

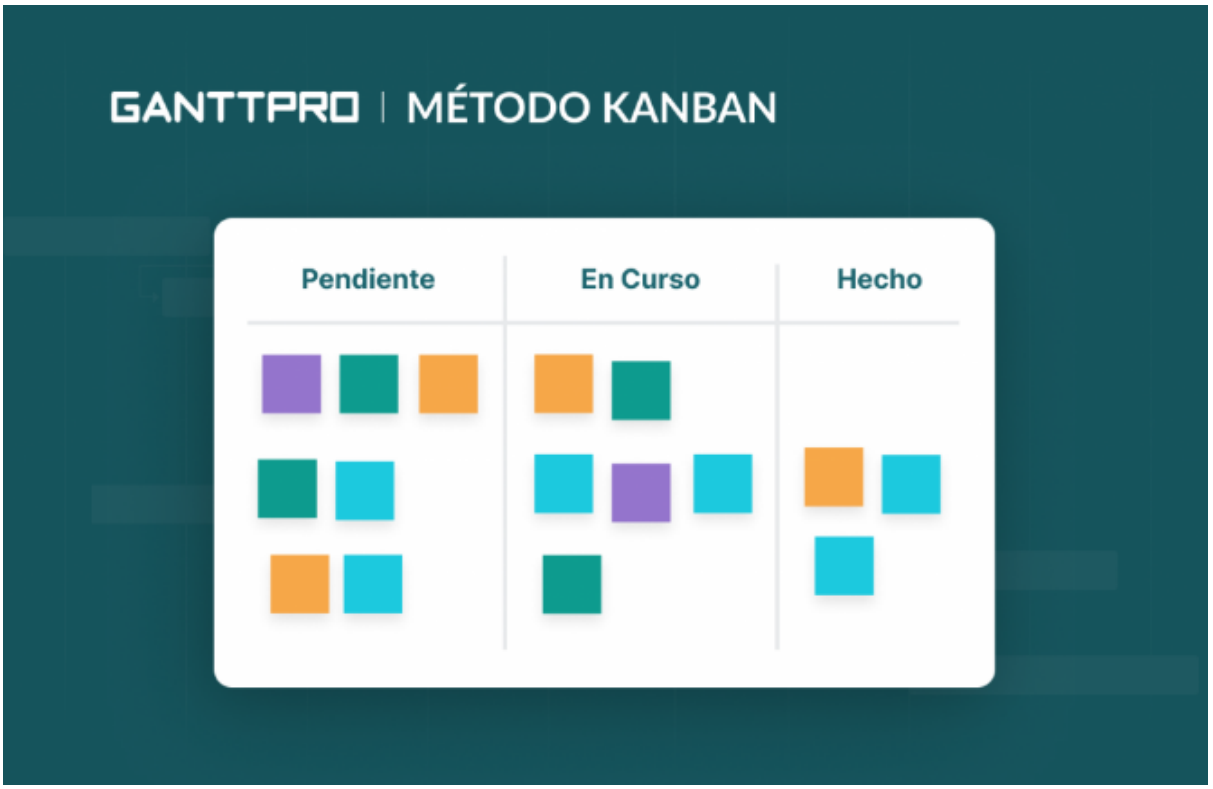
Las metodologías más conocidas son:

-Kanban

-Scrum

-XP (eXtreme Programming)

3.7 PIZARRA KANBAN



Columnas: La pizarra Kanban tiene columnas que representan diferentes etapas del proceso, como "Por Hacer," "En Progreso," y "Hecho."

Tarjetas: Cada tarea o elemento de trabajo se representa con una tarjeta, que se coloca en la columna correspondiente.

Movimiento: A medida que una tarea avanza en el proceso, se mueve la tarjeta de una columna a la siguiente.

Visualización: La pizarra Kanban proporciona una representación visual clara del flujo de trabajo, permitiendo una gestión visual de las tareas.

Transparencia: Todos los miembros del equipo pueden ver el estado de las tareas en tiempo real, lo que fomenta la transparencia y la colaboración.

Agilidad: Facilita la toma de decisiones ágil y una gestión más eficiente de proyectos y tareas.

3.8 SCRUM: Principios

- Enfoque de desarrollo progresivo.

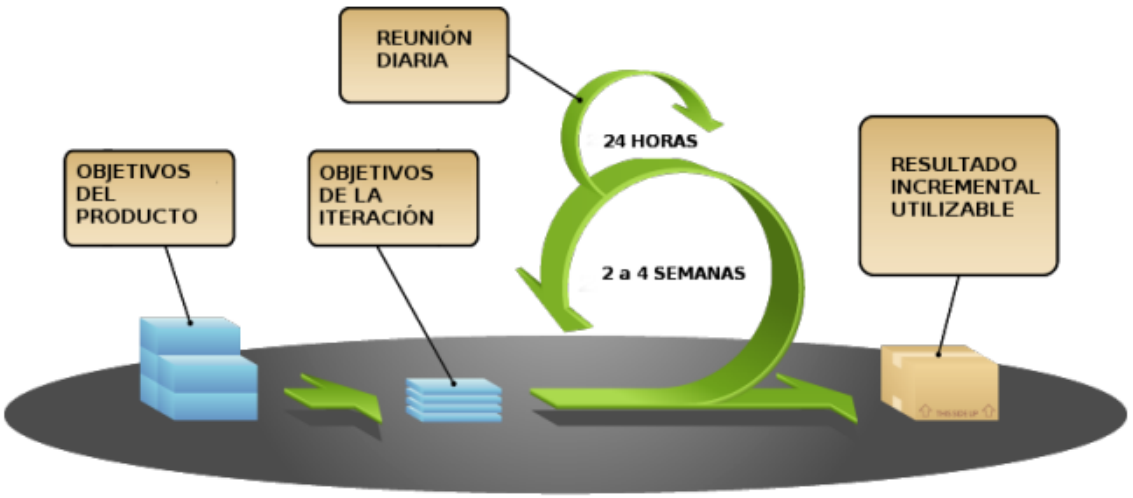
- Se realizan ciclos de trabajo (llamados sprints) cada 2 a 4 semanas.

- Al inicio de cada ciclo, se definen los objetivos de manera prioritaria (conocidos como backlog del sprint).

- Al concluir cada ciclo, se logra una entrega parcial que es funcional para el cliente.

- Se llevan a cabo encuentros diarios para discutir el progreso del ciclo.

3.8.1 SCRUM: Proceso



3.8.2 Roles Clave en SCRUM

• El ***Product Owner*** desempeña el papel de la "voz del cliente" al definir los criterios de aceptación y garantizar su cumplimiento.

• El ***Scrum Master*** se encarga de asegurar la adhesión a la metodología Scrum, fomentando y facilitando la labor del equipo.

• El ***Equipo de Desarrollo***, compuesto por entre 6 y 10 miembros, opera de manera autónoma y multifuncional.

3.8.3 Componentes Clave de SCRUM

- El **Product Backlog** es una lista ordenada que contiene los requisitos del producto.

- El **Sprint Backlog** es una lista de requisitos seleccionados del backlog para su desarrollo durante el sprint.

- El **Incremento** representa el estado del producto tras cada sprint.

3.8.4 Eventos Relevantes en SCRUM

- El **Sprint** es el evento principal que engloba a todos los demás eventos y tiene una duración máxima de 1 mes.

- El **Sprint Planning** es una reunión inicial donde se planifica el trabajo del sprint y tiene una duración máxima de 8 horas.

- El **Daily Scrum** es una reunión diaria de breve duración (máximo 15 minutos) que permite la puesta en común sobre el progreso del sprint.

- El **Scrum Review** es una reunión final que evalúa el incremento logrado y tiene una duración máxima de 4 horas.

- **Scrum Retrospective** es una reunión final que evalúa la aplicación adecuada de la metodología Scrum y tiene una duración máxima de 3 horas.

3.9 VALORES DE LA PROGRAMACIÓN EXTREMA (XP)

3.9.1 PRINCIPALES CARACTERÍSTICAS DE LA PROGRAMACIÓN EXTREMA (XP)

Diseño sencillo: Prioriza la simplicidad en el diseño del software.

Pequeñas mejoras continuas: Realiza mejoras iterativas y frecuentes en lugar de cambios radicales.

Pruebas y refactorización: Incluye pruebas exhaustivas y ajustes regulares para mantener la calidad del código.

Integración continua: Integra el código regularmente para verificar su funcionamiento.

Programación en pareja: Dos programadores colaboran en el mismo código para mayor calidad.

Participación activa del cliente: El cliente es parte integral del equipo de desarrollo.

Propiedad compartida del código: Todos los miembros del equipo son responsables del código.

Estándares de codificación: Establece normas y convenciones para mantener la consistencia.

Jornadas de trabajo limitadas: Se limitan las horas de trabajo para garantizar la salud y la eficiencia del equipo.

4 Lenguajes de Programación y Ejecución

4.1 Obtención de Código Ejecutable:

Dos opciones: Compilar e Interpretar.

4.2 Proceso de Compilación/Interpretación:

Fases: Análisis Léxico y Sintáctico.

Genera código objeto si no hay errores.

Falta análisis semántico.

4.3 Lenguajes Compilados:

Ejemplos: C, C++

Ventaja: Ejecución eficiente.
Desventaja: Requiere compilación tras cambios.

4.4 Lenguajes Interpretados:

Ejemplos: PHP, Javascript

Ventaja: Interpretación directa del código.
Desventaja: Ejecución menos eficiente.

5 JAVA (I):

Compilado e interpretado.
Código fuente Java se compila a bytecode.
Bytecode es para la máquina virtual de Java.
Bytecode se interpreta para ejecutar.

5.1 JAVA (II):

Ventajas:
- Estructurado,
- Orientado a Objetos
- Fácil de aprender
- Buena documentación.

Desventajas:
- Menos eficiente que lenguajes compilados.

5.2 Tipos (I):

Según la operación:
    Declarativos: Resultado sin pasos.
    Imperativos: Pasos para el resultado.

5.3 Tipos (II):

- Lenguajes Declarativos:
    Lógicos: Reglas (Prolog).
    Funcionales: Funciones (Lisp, Haskell).
    Algebraicos: Sentencias (SQL).
- Normalmente interpretados.

5.4 Tipos (III):

- Lenguajes Imperativos:
  - Estructurados: C.
  - Orientados a Objetos: Java.
  - Multiparadigma: C++, Javascript.
- Lenguajes Orientados a Objetos son también estructurados.

5.5 Tipos (IV):

- Según Nivel de Abstracción:
  - Bajo Nivel: Ensamblador.
  - Medio Nivel: C.
  - Alto Nivel: C++, Java.

5.6 Evolución:

Código binario.
Ensamblador.
Lenguajes estructurados.
Lenguajes orientados a objetos.