1. Software y programa. Tipos de software

- **Definición de Software**: Conjunto de programas que controlan el hardware para ejecutar las órdenes del usuario.
- **Tipos de Software**:
    1. **Software de sistema**: Objetivo: Administrar y coordinar el hardware y los recursos básicos. Ejemplos: Sistemas operativos como Windows, Linux, macOS.
    2. **Software de programación**: Objetivo: Herramientas para desarrollar programas. Ejemplo: Entornos de desarrollo, compiladores.
    3. **Software de aplicaciones**: Objetivo: Realizar tareas específicas para el usuario. Ejemplos: Procesadores de texto, hojas de cálculo, videojuegos.
- **Programa**: Conjunto de instrucciones escritas en un lenguaje de programación.
- **Interés del tema**: Desarrollo de aplicaciones informáticas y sus fases: conceptos básicos, lenguajes de programación y procesos para ejecutar un programa.

2. Relación hardware-software

- **Hardware**: Dispositivos físicos de un ordenador.
- **Relación**: El software necesita estar correctamente instalado y configurado para funcionar sobre el hardware.
    1. **Arquitectura Von Neumann (1946)**: Primera arquitectura hardware con programa almacenado.
- **Puntos de vista de la relación**:
    1. **Sistema Operativo**:
        - Actúa como intermediario entre hardware y aplicaciones.
        - Administra recursos como CPU, memoria RAM, y dispositivos de entrada/salida.
        - Oculta la complejidad del hardware a las aplicaciones.
    2. **Aplicaciones**:
        - Conjunto de programas escritos en lenguajes entendibles por humanos, pero traducidos a código binario para ser ejecutados por el hardware.
        - Proceso necesario: Traducción de lenguajes de programación a señales eléctricas interpretables por el hardware.

3. Desarrollo de software

- **Definición**: Proceso desde la idea hasta el programa implementado y funcionando.
- **Objetivo**: Crear programas eficientes, seguros y que cumplan las necesidades del usuario final.
- **Etapas del desarrollo**:
    - Fases secuenciales que garantizan un desarrollo ordenado y coordinado.
    - Estas etapas forman el **Ciclo de Vida del Software**.

3.1. Ciclos de vida del software

1. **Modelo en Cascada**:
    - Secuencia rígida de etapas sin retroceso.
    - Útil para pequeños proyectos con requisitos claros.
2. **Modelo en Cascada con Realimentación**:
    - Permite volver a etapas anteriores para corregir errores.
    - Adecuado para proyectos con pocos cambios.
3. **Modelos Evolutivos**:
    - Consideran la naturaleza cambiante del software:
        - **Iterativo Incremental**: Refinamiento continuo de fases, mejorando en cada repetición.
        - **Modelo en Espiral**: Construcción por versiones mejoradas que aumentan funcionalidades.

4. Lenguajes de programación

- **Definición**: Idiomas artificiales con normas y símbolos que generan un código interpretable por el hardware.
- **Función**: Herramienta para comunicarse con el hardware y ejecutar tareas.
- **Tipos y características**:
    1. **Lenguaje máquina**:
        - Combinaciones de 0s y 1s (código binario).
        - Único para cada procesador, no portable.
        - No se usa actualmente para programar.
    2. **Lenguaje ensamblador**:
        - Usa mnemotécnicos en lugar de binario.
        - Traducción al lenguaje máquina necesaria.
        - Referencia directa a ubicaciones físicas de archivos.
        - Dificultad de uso.
    3. **Lenguajes de alto nivel**:
        - Sentencias derivadas del inglés.
        - Más próximos al razonamiento humano.
        - Traducción al lenguaje máquina requerida.
        - Uso actual, aunque en disminución.
    4. **Lenguajes visuales**:
        - Programación gráfica (diseño visual del software).
        - Generación automática de código.
        - Totalmente portables.
        - Requieren traducción al lenguaje máquina.

4.1.- Concepto y características.

- **Clasificación:**
    - Según cercanía al lenguaje humano (alto/bajo nivel).
    - Según técnica (estructurados, objetos, visuales).
- **Ejemplos destacados:** Java, C, C++, PHP y Visual Basic.

4.2.- Lenguajes de programación estructurados.

- Basados en tres estructuras: secuencial, selectiva (condicional) y repetitiva (bucles).
- **Ventajas:** Sencillez de lectura y mantenimiento.
- **Inconvenientes:** Código monolítico, sin reutilización eficiente.
- **Ejemplo:** Pascal, C.
- Evolución hacia programación modular (división por bloques reutilizables).

4.3.- Lenguajes de programación orientados a objetos.

- **Concepto:** Basados en objetos independientes que colaboran entre sí.
- **Ventajas:**
    - Código reutilizable.
    - Fácil detección y solución de errores.
- **Características:**
    - Objetos con atributos.
    - Clases como colecciones de objetos similares.
    - Comunicación entre objetos mediante métodos.
- **Ejemplos:** C++, Java, VB.NET.

5.- Fases en el desarrollo y ejecución del software.

1. **Análisis de requisitos:** Especificación de necesidades funcionales y no funcionales.
2. **Diseño:** División del sistema en partes, definición de relaciones y herramientas (bases de datos, lenguajes).
3. **Codificación:** Transformación de los diseños en código fuente.
    - **Código Fuente:** Lenguaje de alto nivel.
    - **Código Objeto:** Resultado de compilar (binario intermedio).
    - **Código Ejecutable:** Binario final comprensible por el hardware.
4. **Pruebas:** Detección y corrección de errores.
5. **Documentación:** Registro completo del proyecto.
6. **Explotación:** Instalación y configuración en el cliente.
7. **Mantenimiento:** Actualización y soporte técnico.

5.1.- Análisis.

- **Definición:** Es la primera etapa del proyecto, la más complicada y la que más depende de la capacidad del analista. Se especifican y analizan los requisitos funcionales y no funcionales del sistema.
- **Requisitos:**
    - **Funcionales:** Qué funciones realiza la aplicación.
    - **No funcionales:** Rendimiento, seguridad, normativa.
- **Documento ERS:** Contiene especificaciones detalladas de objetivos, requisitos y planificación.

5.2.- Diseño.

- **Objetivo:** Crear un modelo funcional-estructural del sistema.
- **Decisiones importantes:**

    - Entidades y relaciones de bases de datos.
    - Lenguaje de programación y herramientas seleccionadas.
- **Herramientas:** UML, diagramas.

5.3.- Codificación.

- **Definición:** Consiste en elegir un determinado lenguaje de programación, codificar toda la información anterior y llevarlo a código fuente.
- **Deseable:** Código modular, correcto, legible, eficiente y portable.
- **Fases del código:**
    1. **Fuente:** Instrucciones escritas por el programador.
    2. **Objeto:** Binario intermedio tras compilar.
        - La **compilación** es la traducción de una sola vez del programa, y se realiza utilizando un compilador. La interpretación es la traducción y ejecución simultánea del programa línea a línea. El código objeto es un código intermedio entre el código fuente y el ejecutable y sólo existe si el programa se compila.
    3. **Ejecutable:** Código binario final listo para el hardware (código máquina). El sistema operativo será el encargado de cargar el código ejecutable en memoria RAM y proceder a ejecutarlo.

5.4.- Fases en la obtención de código

5.4.1.- Código Fuente

- **Definición**: Conjunto de instrucciones en lenguajes de alto nivel, no ejecutables directamente.
- **Proceso de obtención**:
    - Partir del análisis y diseño.
    - Diseñar un algoritmo en pseudocódigo.
    - Elegir el lenguaje adecuado.
    - Codificar el algoritmo.
- **Tipos según la licencia**:
    - **Abierto**: Modificable y reutilizable.
    - **Cerrado**: No editable por usuarios.

5.4.2.- Código Objeto

- **Definición**: Código binario intermedio obtenido al compilar el código fuente.
- **Traducción**:
    - **Compilación**: Traduce todo el código en un paso, genera código objeto.
    - **Interpretación**: Traduce y ejecuta línea por línea; no genera código objeto.
- **Características**: No ejecutable directamente, es un paso previo al ejecutable.

5.4.3.- Código Ejecutable

- **Definición**: Código binario final que la computadora puede ejecutar directamente.
- **Proceso**:
    - Enlazar archivos de código objeto con un *linker*.

    - Resultado: Archivo ejecutable controlado por el sistema operativo.

5.5.- Máquinas virtuales

- **Definición**: Capa de software que separa el hardware físico del software, garantizando portabilidad.
- **Funciones principales**:
    - Portabilidad de aplicaciones.
    - Gestión de memoria.
    - Control de hardware y cumplimiento de seguridad.
- **Características**:
    - Aísla la aplicación del hardware físico.
    - Actúa como puente entre bytecode y dispositivos físicos.
    - Verifica y protege el bytecode antes de ejecutarlo.

5.5.1.- Frameworks

- **Definición**: Estructuras de ayuda al programador para desarrollar proyectos más rápido y uniforme.
- **Ventajas**:
    - Desarrollo rápido.
    - Reutilización de código.
    - Portabilidad y diseño uniforme.
- **Inconvenientes**:
    - Dependencia del framework.
    - Consumo de recursos.
- **Ejemplos**:
    - .NET (Visual Studio .NET).
    - Spring (Java).

5.5.2.- Entornos de ejecución

- **Definición**: Servicio que permite ejecutar programas, formado por máquina virtual y APIs.
- **Funciones principales**:
    - Configurar memoria.
    - Enlazar programas con bibliotecas y subprogramas.
    - Depurar errores semánticos.
- **Características**:
    - Intermediario entre lenguaje fuente y sistema operativo.
    - Distribuido junto a APIs compatibles.
- **Nota**: Para desarrollar nuevas aplicaciones se requiere un **entorno de desarrollo**, no solo un entorno de ejecución.

5.5.3.- Java Runtime Environment (JRE)

- **Concepto**: JRE permite ejecutar programas Java en cualquier plataforma.
- **Componentes**:
    - **Máquina Virtual Java (JVM)**: Interpreta el código Java.

    - **Bibliotecas estándar (API)**: Implementan funcionalidades básicas.
- JVM y API son consistentes y se distribuyen conjuntamente.

5.6.- Pruebas

- **Objetivo**: Validar y verificar el software usando datos de prueba límite.
- **Tipos de pruebas**:
    - **Unitarias**: Verifican partes individuales del software (independientes). Ejemplo: *JUnit* en Java.
    - **Integración**: Comprueban el sistema completo, con todas sus partes interrelacionadas.
    - **Beta Test**: Prueba final en el entorno de producción del cliente.

5.7.- Documentación

- **Objetivo**: Asegurar el uso, mantenimiento y evolución del software.
- **Tipos de guías**:
    1. **Técnica**:
        - Contenido: Diseño, codificación y pruebas realizadas.
        - Dirigida a: Técnicos (analistas y programadores).
        - Objetivo: Facilitar desarrollo, correcciones y mantenimiento.
    2. **Uso**:
        - Contenido: Funcionamiento, requisitos, ejemplos, y resolución de problemas.
        - Dirigida a: Usuarios finales.
        - Objetivo: Informar a los clientes para utilizar el software.
    3. **Instalación**:
        - Contenido: Puesta en marcha, explotación y seguridad.
        - Dirigida a: Técnicos responsables y clientes.
        - Objetivo: Garantizar una instalación precisa y segura.

5.8.- Explotación

- **Definición**: Fase en que los usuarios finales comienzan a usar la aplicación.
- **Pasos principales**:
    1. **Instalación**: Transferir programas al equipo del cliente.
    2. **Configuración**: Ajustar parámetros de uso según las necesidades del cliente (manual o automatizada).
    3. **Producción normal**: El software queda en uso cotidiano.
- **Nota importante**: Última fase crítica, con Beta Test bajo condiciones reales del cliente.

5.9.- Mantenimiento

- **Definición**: Proceso continuo de mejora, corrección y actualización del software.
- **Razones del mantenimiento**:
    - **Perfectivos**: Mejorar la funcionalidad existente.
    - **Evolutivos**: Adaptarse a nuevas necesidades del cliente.
    - **Adaptativos**: Ajustes por cambios en el hardware o tendencias del mercado.

    - **Correctivos**: Solucionar errores detectados tras la entrega.
- Es la etapa más larga del ciclo de vida del software.

XAMPP

Definición: Es un paquete de software libre que proporciona un entorno de desarrollo local para probar y ejecutar aplicaciones web. Es ampliamente utilizado por desarrolladores web para trabajar en proyectos sin necesidad de configurar manualmente un servidor en la nube.

Características:

1. Multiplataforma: Funciona en Windows, Linux y macOS.
2. Fácil de instalar.
3. Gratuito y de código abierto.

Componentes:

1. **X**: Multiplataforma.
2. **A**: Apache (servidor web).
3. **M**: MariaDB/MySQL (base de datos).
4. **P**: PHP (programación).
5. **P**: Perl (programación).

Usos:

- Desarrollar y probar aplicaciones web.
- Ejecutar scripts PHP.
- Gestionar bases de datos localmente.

Ventajas:

1. Configuración rápida.
2. Preconfigurado y fácil de usar.
3. Ideal para aprender desarrollo web.

Desventajas:

1. No apto para producción (riesgos de seguridad).
2. Puede consumir recursos del sistema.

CRUD

Definición: representa las cuatro operaciones básicas que se pueden realizar sobre una base de datos o cualquier sistema de almacenamiento de datos. Estas operaciones son:

1. **Create (Crear)**: Añadir nuevos datos al sistema.
2. **Read (Leer)**: Consultar o recuperar datos existentes.
3. **Update (Actualizar)**: Modificar datos existentes.
4. **Delete (Eliminar)**: Eliminar datos del sistema.

Ejemplo:

- **Create**: Registrar un nuevo usuario.
- **Read**: Ver detalles de un usuario.
- **Update**: Editar su información.
- **Delete**: Borrar al usuario.

SCRUM

Definición: Es un marco de trabajo (framework) dentro del desarrollo ágil de software que se utiliza para gestionar proyectos y coordinar equipos de trabajo de manera eficiente.

Principios fundamentales del SCRUM:

1. **Iterativo e Incremental**: Divide el trabajo en ciclos cortos llamados **sprints** (2-4 semanas) con el objetivo de entregar un incremento de producto funcional.
2. **Colaboración y Transparencia**: Trabajo colaborativo y claro.
3. **Adaptabilidad**: Ajustes según necesidades del proyecto.

Roles:

1. **Product Owner**: Prioriza tareas y gestiona el **Product Backlog**.
2. **Scrum Master**: Facilita el proceso y resuelve impedimentos.
3. **Equipo de Desarrollo**: Realiza el trabajo técnico.

Artefactos:

1. **Product Backlog**: Lista priorizada de tareas.
2. **Sprint Backlog**: Tareas seleccionadas para un sprint.
3. **Incremento**: Resultado funcional del sprint.

Eventos:

1. **Sprint Planning**: Planificación de tareas para el sprint.
2. **Daily Scrum**: Reunión breve para sincronizar el equipo.
3. **Sprint Review**: Presentación del incremento.
4. **Sprint Retrospective**: Reflexión sobre el proceso y mejoras.