Definiciones

Software: Conjunto de programas y datos que coexisten en una computadora

Programas: Conjunto de instrucciones que indican al ordenador lo que debe hacer.

Datos: Información con la que trabajan las instrucciones de los programas. Es un conjunto de información que interpretada por un programa adecuado representa números, texto, imágenes, sonido, vídeo, etc..

Software de Sistema: Es el que trabaja directamente con el hardware, haciéndolo funcionar y poniéndolo al servicio del resto del software y, por tanto, del usuario (Sistema operativo y drivers).

Software de aplicación: Aquel que el usuario utiliza para realizar su trabajo (ofimática, contabilidad, etc)

Software de programación: Aquel utilizado para el desarrollo de programas.

Software de Utilidades: Aquel utilizado para analizar, configurar, optimizar y mantener un equipo.

Aplicación: conjunto de programas escritos en algún lenguaje de programación que el hardware del equipo debe interpretar y ejecutar.

Herramientas CASE: Conjunto de aplicaciones que se utilizan en el desarrollo de software con el objetivo de reducir costes y tiempo del proceso, mejorando por tanto la productividad del proceso.

1. IDE (Entorno de Desarrollo Integrado)
Un IDE es una aplicación que proporciona herramientas para el desarrollo de software. Incluye un editor de código, herramientas de depuración, un compilador y, a veces, un sistema de gestión de versiones. Su objetivo es facilitar el proceso de escritura, prueba y depuración del código.

2. Librería
Una librería de programación es una colección de código desarrollado previamente. Estas librerías se pueden importar a otros códigos facilitando la programación y las tareas comunes, como manipulación de datos o manejo de archivos.

3. Framework
Un entorno de trabajo (framework) es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática. Un framework dicta la arquitectura de una aplicación, ofreciendo reglas y convenciones que el desarrollador debe seguir. Los frameworks son comunes en el desarrollo web (por ejemplo, Django, Angular) y suelen incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas

4. SDK (Kit de Desarrollo de Software)
Un SDK es un conjunto de herramientas y recursos que permiten a los desarrolladores crear aplicaciones para una plataforma específica. Incluye librerías, documentación, ejemplos de código y herramientas de desarrollo.

5. Código Fuente
El código fuente es el conjunto de instrucciones escritas por un programador en un lenguaje de programación. Este código es legible por humanos y se compone de líneas de texto que describen la lógica y el funcionamiento de un programa. El código fuente se traduce a un formato que una computadora puede ejecutar.

7. Ejecutable
Un ejecutable es un archivo que contiene código objeto que ha sido enlazado y puede ser ejecutado directamente por el sistema operativo.

9. Traductor
Un traductor es un programa que recibe como entrada código escrito en un cierto lenguaje y produce como salida código en otro lenguaje. Esto puede incluir compiladores (que traducen código fuente a código objeto) e intérpretes (que traducen y ejecutan el código fuente directamente).

Tipos de Software

Software Libre: Permite el uso, modificación, estudio y distribución sin restricciones.
Ejemplo: Linux.

Software de Fuente Abierta: Permite el acceso, modificación y redistribución del código fuente. Puede requerir que versiones mejoradas del software lleven un nombre o una versión diferente al original.
Ejemplo: Mozilla Firefox.

Software con Copyleft: Permite modificar y redistribuir, pero obliga a que las versiones modificadas tengan las mismas restricciones que la original.
Ejemplo: WordPress.

Freeware: Es gratuito, pero el código no está disponible ni se puede modificar.
Ejemplo: Skype.

Shareware: Permite el uso gratuito por un tiempo o con funcionalidad limitada, pero requiere compra para desbloquear el acceso completo.
Ejemplo: WinRAR.

Software comercial: Es desarrollado por una empresa que pretende ganar dinero por su uso.
Ejemplo: Microsoft Office.

Software privativo: Restringe el uso, la modificación y la distribución del código fuente, sin acceso al mismo.
Ejemplo: macOS.

Ciclo de vida del software

Modelo clásico en Cascada: En este modelo para empezar una etapa es necesario finalizar la etapa anterior sin posibilidad de vuelta atrás.

Modelo en Cascada con Retroalimentación: Igual pero se puede volver atrás.

Modelo iterativo incremental: El software se construye en pequeñas partes llamadas "incrementos." Cada incremento añade funcionalidad al software.

Modelo en Espiral: El software se va construyendo repetidamente en forma de versiones que son cada vez mejores.

Etapas:
• Análisis.
• Diseño.
• Codificación y Obtención código ejecutable.
• Ejecución y Pruebas.
• Documentación.
• Explotación.
• Mantenimiento.

Análisis
Fase de mayor importancia y más complicada, se revisarán los requisitos funcionales y no funcionales del sistema, acabando en el documento ERS (Especificación de Requisitos Software).

Diseño
Se divide el sistema en partes y se especifica qué se hará en cada parte. Se establecen las entidades, relaciones, el sistema gestor de base de datos y lenguaje de programación.

Codificación y Obtención código ejecutable
Proceso de programación, teniendo en cuenta: modularidad, corrección, legibilidad, eficiencia, portabilidad y mantenimiento.

Pruebas

- Pruebas unitarias: se prueban una a una las diferentes partes de software y se comprueba su funcionamiento por separado.

- Pruebas de integración : Se comprueba el funcionamiento del sistema completo.

Documentación, explotación y mantenimiento
Documentos en el desarrollo de software: guía técnica, guía de uso, guía de instalación.
La explotación es la fase en que los usuarios finales conocen la aplicación y comienzan a utilizarla.
Tipos de mantenimiento:

- Perfectivos: Para mejorar la funcionalidad del software.
- Evolutivos: El cliente tendrá en el futuro nuevas necesidades.
- Correctivos: La aplicación tendrá errores en el futuro que habrá que corregir.

Tipos de Lenguaje de Programación

Lenguaje máquina: Instrucciones en unos y ceros que entiende directamente el procesador, único para cada procesador.

Lenguaje ensamblador (bajo nivel): Usa mnemotécnicos en lugar de unos y ceros pero su estructura se sigue acercando al lenguaje máquina; requiere traducción.

Lenguaje de alto nivel: Utiliza sentencias más simples en inglés que pueden equivaler a varias instrucciones complicadas de bajo nivel, permitiendo escribir código de manera más fácil y rápida . Requiere traducción por parte de un traductor.
Ejemplos: C, Java o PHP.

Lenguajes visuales: Se programa gráficamente, el código se genera automáticamente y necesita traducción; son completamente portables.
Ejemplos: SQL, Unix o Shell.

Quinta generación: esta generación se encuentra todavía en desarrollo y se caracteriza por un uso del lenguaje muy cercano al natural gracias al uso de la inteligencia artificial.

También se dividen en:

Lenguaje compilado: el código fuente se convierte directamente en lenguaje máquina mediante un compilador, generando un ejecutable que puede ser ejecutado directamente por la computadora. Ejemplos: C, C++.

Lenguaje interpretado: el código fuente es ejecutado línea por línea por un intérprete sin necesidad de una compilación previa. Ejemplos: Python, JavaScript.

(Java es un lenguaje compilado e interpretado, primero se compila en bytecode (usando javac) y luego ese bytecode es interpretado por su máquina virtual.)

Metodologías ágiles

La metodología ágil es un conjunto de técnicas aplicadas en ciclos de trabajo cortos con el objetivo de que el proceso de entrega de un proyecto sea más eficiente. Así, con cada etapa completada, ya se pueden entregar avances y se deja de lado la necesidad de esperar hasta el término del proyecto. Esta metodología permite controlar mejor el presupuesto, reducir los riesgos y que el cliente participe de manera activa en el proyecto.

Esta metodología es adecuada para proyectos de alta complejidad, con necesidad de flexibilidad e innovación en el proceso y de flujo dinámico.

Algunos inconvenientes de este tipo de metodologías son:

- El presupuesto no suele estar cerrado.
- Requiere de equipos capaces de adaptarse a cambios frecuentes.
- Es necesaria la comunicación y colaboración continua.

Tipos de metodologías ágiles

Scrum

La metodología ágil Scrum se basa en ciclos de trabajo o sprints que suelen durar entre dos semanas y un mes en los que se entrega alguna parte del proyecto. En cada nuevo sprint se genera una versión del producto que supera a la anterior.

Los pilares fundamentales de esta metodología son la planificación del sprint, las reuniones diarias, la revisión del sprint y la retrospectiva del equipo sobre el sprint.

Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, los requisitos son cambiantes o están poco definidos y donde la innovación, la flexibilidad y la productividad son fundamentales.

Algunas de sus ventajas son la adaptabilidad, la entrega incremental, la proactividad del cliente, la reducción de riesgos y la gran calidad del producto final.

Los inconvenientes de esta metodología son:

- Dependencia de la colaboración y la comunicación.
- Gran carga de reuniones que pueden ralentizar el proceso.
- Dificultades en la planificación.

Kanban

La estrategia Kanban, también conocida como “Tarjeta Visual” consiste en la creación de un panel de tareas en el que se reflejan tres columnas: pendientes, en proceso y terminadas. Este panel está al alcance de todos permitiendo la visualización rápida de lo que se ha hecho o falta por hacer lo que mejora la productividad y eficiencia del equipo.

En esta estrategia es indispensable la identificación de las tareas al comenzar el proyecto, y su desglose en subtareas que se deben representar en el tablero. También son necesarias

las reuniones regulares para revisar el flujo de trabajo y buscar maneras de optimizar el proceso.

Las ventajas de esta metodología son:

- Visualización del proceso.
- Flexibilidad y adaptabilidad.
- Mejora continua.
- Elusión de sobrecarga.

Por otro lado, sus inconvenientes pueden ser la falta de estructura, la dependencia de la autoorganización y la dificultad para priorizar