UNIDAD 1. ELEMENTOS DE DESARROLLO DEL SOFTWARE

Introducción.

En esta Unidad aprenderemos a:

- **Reconocer la relación de los programas con los componentes del sistema informático:** Los programas de software son esenciales en el funcionamiento de los sistemas informáticos. Estos programas interactúan con el hardware para realizar tareas específicas.

- **Diferenciar código fuente, objeto y ejecutable:**

    - Código fuente: Es el texto legible escrito en un lenguaje de programación. Es la versión en la que los programadores escriben el software.
    - Código objeto (intermedio): Este archivo binario no es ejecutable por sí mismo, pero es una etapa intermedia en el proceso de compilación.
    - Código ejecutable: Este archivo binario es la versión final que puede ser ejecutada por una computadora. Es válido principalmente para lenguajes compilados como C, C++, y Java.

- **Identificar las fases de desarrollo de una aplicación informática:** El desarrollo de software sigue un proceso que incluye varias fases: análisis, diseño, codificación, pruebas y mantenimiento. Cada fase tiene su propósito y se realiza de manera secuencial para garantizar un producto final de alta calidad.

- **Clasificar los lenguajes de programación:** Existen diversos lenguajes de programación, cada uno con sus propias características y aplicaciones. Algunos ejemplos incluyen C, C++, Java, JavaScript, HTML, XML y JSON. Estos lenguajes se eligen según las necesidades y objetivos del proyecto.

Conceptos básicos

Tipos de software:

1. De sistema: Incluye el sistema operativo y los controladores (drivers) que permiten que el hardware funcione correctamente.
2. De aplicación: Estas son las aplicaciones que utilizamos en nuestro día a día, como suites ofimáticas, navegadores web y herramientas de edición de imágenes.
3. De desarrollo: Este tipo de software se utiliza para crear otros programas, como editores de código, compiladores e intérpretes.

Relación Hardware-Software.

- Disco duro: Almacena de forma permanente los archivos ejecutables y los archivos de datos, proporcionando almacenamiento a largo plazo.
- Memoria RAM: Almacena temporalmente el código binario de los archivos ejecutables y los archivos de datos necesarios mientras la computadora está en funcionamiento.
- CPU (Unidad Central de Procesamiento): Lee y ejecuta las instrucciones almacenadas en la memoria RAM, así como los datos necesarios para realizar las tareas.
- E/S (Entrada/Salida): Se encarga de recoger nuevos datos desde la entrada, mostrar los resultados en la salida y leer/guardar datos en el disco, facilitando la interacción entre la computadora y el usuario.

El disco duro se considera un periférico de E/S (Entrada/Salida).

La CPU también se llama UCP (por sus siglas en inglés), procesador o microprocesador.

Códigos fuente, objeto y ejecutable.

- **Código fuente**: Es un archivo de texto legible escrito en un lenguaje de programación. Es la forma en que los programadores escriben y entienden el software.
- **Código objeto** (intermedio): Este archivo binario no es ejecutable, pero es una etapa intermedia en el proceso de compilación. Puede contener

instrucciones y datos en un formato que la computadora pueda comprender.
- **Código ejecutable**: Es un archivo binario que la computadora puede ejecutar. Solo es válido para lenguajes compilados como C, C++ y Java. En los lenguajes interpretados, como PHP y JavaScript, no se produce un código objeto separado.

Ciclo de vida del software

Ingeniería de software:

- Es una disciplina que estudia los principios y metodologías para el desarrollo y mantenimiento de sistemas de software.
- Algunos autores consideran que "desarrollo de software" es un término más apropiado que "ingeniería de software", ya que este último implica niveles de rigor y prueba de procesos que no son apropiados para todos los proyectos de desarrollo de software.

Desarrollo de software

- Comprende diversas fases principales: análisis, diseño, codificación, pruebas y mantenimiento.

Fases principales:

- **ANÁLISIS**: En esta fase, se determinan y definen claramente las necesidades del cliente y se especifican los requisitos que debe cumplir el software a desarrollar. La especificación de requisitos debe ser completa, concisa, sin ambigüedades y entendible por el cliente.

- **DISEÑO**: En esta fase, se descompone y organiza el sistema en elementos componentes que pueden ser desarrollados por separado. Se especifica la interrelación y funcionalidad de estos elementos componentes. Las actividades habituales incluyen el diseño arquitectónico, el diseño detallado, el diseño de datos y el diseño de interfaz de usuario.
    - mock up: diseño estático de una página web o aplicación que presenta muchos de sus elementos de diseño finales pero no es funcional. ejemplo:prototipo

- **CODIFICACIÓN**: Aquí es donde se escribe el código fuente de cada componente del software. Se pueden utilizar diferentes lenguajes informáticos, como C, C++, Java, JavaScript, así como lenguajes de otro tipo como HTML, XML y JSON.

- **PRUEBAS**: El objetivo principal de las pruebas es identificar defectos y asegurarse de que el programa funcione correctamente. Se somete al programa a diversas situaciones para descubrir posibles errores.

- **MANTENIMIENTO**: Durante la explotación del software, es necesario realizar cambios ocasionales para corregir defectos, mejorar la funcionalidad, agregar nuevas características o adaptarse a nuevos entornos. Hay 3 tipos de mantenimiento:
    - **Correctivo**: se refiere a un conjunto de actividades técnicas que tienen como objetivo solucionar problemas en los activos y dispositivos cada vez que surja la necesidad de repararlos o incluso reemplazarlos.
    - **Perfectivo**: su propósito es adaptar el software mediante la incorporación de nuevas funcionalidades según sea requerido, y suprimiendo aquellas características que carecen de relevancia o resultan poco efectivas en el contexto del software en cuestión.
    - **Evolutivo**: Se esfuerza por perfeccionar el sistema con el propósito de prevenir dificultades o inconvenientes en el futuro.
    - **Adaptativo**: Implica realizar ajustes en el software sin perturbar su operatividad, con la finalidad de adecuarlo a las transformaciones en el entorno de la aplicación.
    - **Preventivo**: Se enfoca en llevar a cabo una evaluación metódica y siguiendo ciertos estándares en dispositivos de diversas naturalezas, ya sean mecánicos, eléctricos, informáticos, entre otros, con el objetivo de prevenir fallos causados por el uso, desgaste o el paso del tiempo.

Modelos de desarrollo

Modelos de desarrollo de software:

- Existen varios enfoques para el desarrollo de software, que se pueden clasificar en modelos clásicos (predictivos), modelos evolutivos o incrementales y metodologías ágiles (adaptativos).

Modelo en cascada:

- Este modelo, de mayor antigüedad, identifica las fases principales del desarrollo de software y se realiza de manera secuencial. Cada fase debe completarse antes de pasar a la siguiente. Es un modelo rígido que no se adapta bien a cambios continuos en las especificaciones.

Modelo en V:

- Similar al modelo en cascada, este enfoque organiza las fases en un formato jerarquizado. Los niveles superiores indican mayor abstracción, mientras que los niveles inferiores indican mayor detalle. Cada fase se valida a través de una fase correspondiente en el nivel opuesto.

Modelo de construcción de prototipos:

- A menudo, los requisitos no están especificados con claridad, y este modelo implica crear prototipos de software para refinar y clarificar los requisitos en colaboración con el cliente.

Modelo en espiral:

- Este modelo, desarrollado por Boehm, se utiliza en la programación orientada a objetos y enfatiza la reutilización de código. Se basa en un ciclo de desarrollo que incluye análisis, diseño, codificación y otras fases.

Metodologías ágiles:

- Estas metodologías se basan en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan con el tiempo según las necesidades del proyecto. Los equipos de desarrollo trabajan en colaboración y toman decisiones a corto plazo. Algunas de las metodologías ágiles más conocidas incluyen Kanban, Scrum y eXtreme Programming (XP).

Kanban:

- Originado en la industria de fabricación de productos, se enfoca en controlar la producción por demanda y entregar el máximo valor a los clientes utilizando los recursos necesarios.

Scrum:

- Basado en un modelo de desarrollo incremental, utiliza iteraciones regulares llamadas "sprints" para lograr entregas parciales utilizables por el cliente. Define roles, artefactos y eventos, como reuniones diarias, revisiones y retrospectivas.

XP (Programación Extrema):

- Se centra en valores como la simplicidad, la comunicación, la retroalimentación, el valor y el respeto. Promueve prácticas como el diseño sencillo, pequeñas mejoras continuas, pruebas y refactorización, integración continua y programación en parejas. El cliente se integra en el equipo de desarrollo, y la propiedad del código es compartida. Se enfoca en un trabajo de 40 horas semanales.

Lenguajes de programación.

Obtención de código ejecutable:
Para obtener código binario ejecutable, se tienen dos opciones: compilar o interpretar.

Proceso de compilación/interpretación:
La compilación/interpretación del código fuente se lleva a cabo en dos fases:

- **Análisis léxico**: En esta etapa, el código fuente se descompone en tokens o palabras clave, lo que permite al compilador o intérprete reconocer y entender la estructura del programa. Esto es importante para identificar palabras reservadas, variables, operadores, y otros elementos.
- **Análisis sintáctico**: En esta fase, el compilador o intérprete verifica si la estructura del programa es gramaticalmente correcta. Se construye un árbol sintáctico que representa la jerarquía y la relación entre las distintas partes del código. Si no se encuentran errores, se genera el código objeto correspondiente.

Es importante destacar que un código fuente correctamente escrito desde un punto de vista léxico y sintáctico no garantiza que el programa funcione según lo deseado. No se realiza un análisis semántico en esta etapa, lo que significa que errores lógicos pueden pasar desapercibidos.

Lenguajes compilados:
Ejemplos de lenguajes compilados incluyen C y C++. La principal ventaja de estos lenguajes es que ofrecen una ejecución muy eficiente. Sin embargo, la principal desventaja es que es necesario compilar el código fuente cada vez que este es modificado. Esto puede ser un proceso que consume tiempo y recursos.

Lenguajes interpretados:
Ejemplos de lenguajes interpretados son PHP y Javascript. La ventaja clave de los lenguajes interpretados es que el código fuente se interpreta directamente y no requiere una etapa de compilación separada. Sin embargo, la ejecución suele ser menos eficiente que la de los lenguajes compilados.

JAVA:
Java es un lenguaje que combina características de lenguajes compilados e interpretados. El código fuente Java se compila en un código binario intermedio llamado bytecode. Este bytecode se interpreta en la máquina virtual de Java para su ejecución. Algunas ventajas de Java incluyen su estructura orientada a objetos, relativa facilidad de aprendizaje, buena documentación y una amplia base de usuarios. Sin embargo, su principal desventaja es que suele ser menos eficiente que los lenguajes puramente compilados.

Tipos de lenguajes según la forma en que operan:

- **Declarativos**: Estos lenguajes indican el resultado deseado sin especificar los pasos a seguir para obtenerlo.
- **Imperativos**: En cambio, los lenguajes imperativos especifican los pasos concretos que deben seguirse para alcanzar un resultado.

Tipos de lenguajes declarativos:

- **Lógicos**: Utilizan reglas lógicas para definir el comportamiento. Un ejemplo es Prolog.
- **Funcionales**: Se basan en el uso de funciones matemáticas para describir las operaciones. Ejemplos son Lisp y Haskell.
- **Algebraicos**: Estos lenguajes emplean sentencias para realizar operaciones, como SQL. Suelen ser lenguajes interpretados.

Tipos de lenguajes imperativos:

- **Estructurados**: Ejemplos incluyen C y Pascal. Se centran en la secuencia lógica de instrucciones.
- **Orientados a objetos**: Java y C# son ejemplos de lenguajes orientados a objetos. Estos lenguajes se centran en objetos y sus interacciones.
- **Multiparadigma**: Lenguajes como C++ y Javascript son versátiles y pueden utilizarse en diferentes estilos de programación. Los lenguajes orientados a objetos también son lenguajes estructurados.

Tipos de lenguajes según nivel de abstracción:

- **Bajo nivel:** Incluye lenguajes como ensamblador, que se relacionan estrechamente con la arquitectura de la máquina.
- **Medio nive**l: C es un ejemplo de lenguaje de nivel medio que proporciona un equilibrio entre bajo y alto nivel.
- **Alto nivel**: Ejemplos de lenguajes de alto nivel son C++, Java y Python, que ofrecen una mayor abstracción y facilidad de uso.

Evolución:
A lo largo de la historia, los lenguajes de programación han evolucionado desde el código binario inicial hasta los lenguajes orientados a objetos y las abstracciones de alto nivel. Esta evolución ha permitido a los programadores trabajar de manera más eficiente y con un enfoque en la resolución de problemas en lugar de detalles de bajo nivel.

Criterios para la selección de un lenguaje:

- **Campo de aplicación**: El tipo de proyecto o aplicación que se está desarrollando puede influir en la elección del lenguaje.
- **Experiencia previa**: La familiaridad con un lenguaje puede ser un factor importante.
- **Herramientas de desarrollo**: La disponibilidad de herramientas y entornos de desarrollo puede facilitar el trabajo.
- **Documentación disponible**: Una buena documentación puede ser esencial para aprender y utilizar eficazmente un lenguaje.
- **Base de usuarios**: Un lenguaje con una comunidad activa de usuarios puede proporcionar soporte y recursos adicionales.
- **Reusabilidad**: La capacidad de reutilizar código existente puede ahorrar tiempo y esfuerzo.
- **Transportabilidad**: La portabilidad del código a diferentes plataformas puede ser importante en proyectos multiplataforma.
- **Imposición del cliente**: En algunos casos, el cliente o la empresa pueden tener restricciones o preferencias específicas en cuanto al lenguaje a utilizar.