**RESUMEN ENTORNOS DE DESARROLLO**

**UNIDAD 1**

# **Sofware development elements**.

## CONCEPTO DE PROGRAMA INFORMÁTICO Y DE APLICACIÓN INFORMÁTICA.

Un ordenador es un dispositivo electrónico utilizado para procesar información y obtener resultados. Los datos y la información se pueden introducir en el ordenador por la entrada (input) y a continuación se procesan para producir una salida (output, resultados).

La mayoría de los ordenadores, constan fundamentalmente de tres componentes principales: el procesador (CPU); la memoria principal o central y el programa. Los elementos físicos del ordenador reciben el nombre de hardware en cambio los programas, son llamados software.

Programación: es automatizar y definir una serie de procesos para resolver un problema y obtener un resultado final. Un programa es el conjunto de instrucciones que se le dan al ordenador para resolver un problema o tarea determinada.

Lenguaje de Programación: Se utilizan para indicar al ordenador las acciones que ha de realizar para resolver un determinado problema. Básicamente los lenguajes de programación se componen de ordenes que le dicen al ordenador lo que tiene que hacer.

Un programa informático: es un conjunto de instrucciones que producirán la ejecución de una determinada tarea. En esencia, un programa es un medio para conseguir un fin. El desarrollo de un programa requiere las siguientes fases:

1. Definición y análisis del problema.
2. Diseño de algoritmos. - diagrama de flujo; - pseudocódigo.
3. Codificación del programa.
4. Depuración y verificación del programa.
5. Documentación.
6. Mantenimiento.

## CONCEPTO DE LENGUAJE DE PROGRAMACIÓN

Para poder entender el concepto de lenguaje de programación, debemos explicar antes que significa el término algoritmo. Un algoritmo es un método para resolver un problema. La resolución de un problema exige el diseño de un algoritmo que resuelva el problema propuesto. Los pasos para la resolución de un problema son:

1. Diseño del algoritmo, que describe la secuencia ordenada de pasos que conducen a la solución de un problema dado.
2. Expresar el algoritmo como un programa en un lenguaje de programación adecuado (Fase decodificación).
3. Ejecución y validación del programa por el ordenador.

Para llegar a la realización de un programa es necesario el diseño previo de un algoritmo, de modo que sin algoritmo no puede existir un programa. Los algoritmos son independientes tanto del lenguaje de programación en que se expresan como del ordenador que los ejecuta. El diseño de la mayoría de los algoritmos requiere creatividad y conocimientos profundos de la técnica de la programación.

Así pues, los Lenguajes de programación (Lenguaje máquina, ensamblador y de alto nivel), al igual que los idiomas sirven de vehículo de comunicación entre seres humanos y los ordenadores.

Ahora bien, es el procesador el que debe ser capaz de interpretar el algoritmo, lo que significa:

1. Comprender las instrucciones de cada paso
2. Realizar las operaciones correspondientes.

Cuando el procesador es un ordenador, el algoritmo se ha de expresar en un formato que se denomina programa. El proceso de traducir un algoritmo en pseudocódigo a un lenguaje de programación se denomina codificación, y el algoritmo escrito en un lenguaje de programación se denomina código fuente.

En la realidad el ordenador no entiende directamente los lenguajes de programación, sino que se requiere un programa que traduzca el código fuente a otro lenguaje que sí entiende la máquina directamente, pero muy complejo para las personas. Este lenguaje se conoce como lenguaje máquina y el código correspondiente código máquina. Los programas que traducen el código fuente escrito en un lenguaje de programación a código máquina se denominan traductores.

Se utilizan tres métodos generales de implementación, de lenguajes de programación:

1. Interpretación: Se dispone de un programa especial (intérprete) capaz de leer, analizar, decodificar y ejecutar una a una las sentencias de un programa fuente. Su inconveniente suele ser menor rapidez que otras implementaciones.

2. Compilación: Se dispone de un programa especial (compilador) capaz de leer y analizar sentencias y traducirlas a un conjunto de instrucciones de máquina equivalentes (programa objeto).

3. Implementación híbrida: Se traducen los programas fuente escritos en un lenguaje de alto nivel, a un lenguaje intermedio diseñado para permitir una interpretación fácil.

Las características fundamentales que debe cumplir todo algoritmo son:

1. Un algoritmo debe ser preciso e indicar el orden de realización de cada paso.

2. Un algoritmo debe estar definido. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.

3. Un algoritmo debe ser finito. Si se sigue un algoritmo, se debe terminar en algún momento; o sea, debe tener un número finito de pasos.

## TIPOS DE LENGUAJE DE PROGRAMACIÓN

Existen, al menos, tres formas en que podemos establecer una clasificación de los lenguajes de programación:

1. SEGÚN EL NIVEL DE ABSTRACCIÓN:
   1. LENGUAJES DE MÁQUINA: Se llama lenguaje máquina a las instrucciones que se dan directamente al ordenador, utilizando una serie de dígitos binarios o bits, representados por los números 0 y 1 que especifican una operación.
   2. LENGUAJES ENSAMBLADOR: Son representaciones simbólicas del lenguaje maquina asociado, lo cual permite una programación menos tediosa que con el anterior; sin embargo, es necesario un profundo conocimiento de la arquitectura hardware subyacente para realizar una programación efectiva.
   3. LENGUAJES DE ALTO NIVEL: Estos lenguajes permiten que los algoritmos se codifiquen en un nivel y estilo de escritura fácilmente legible y comprensible por otros programadores. Son los más utilizados actualmente como lenguaje de programación.
   4. LENGUAJES DE CUARTA GENERACIÓN: Son lenguajes en los que apenas hay código y en su lugar aparecen indicaciones sobre qué es lo que el programa debe de obtener.
   5. LENGUAJES DECLARATIVOS: Basados en el pensamiento humano y no en el procesamiento del computador; utilizan para ello fundamentos matemáticos y concretamente la lógica de predicados.
2. SEGÚN EL PARADIGMA QUE IMPLEMENTEN
   1. PROGRAMACIÓN DESORDENADA: En este estilo de programación, predomina el instinto del programador por encima del uso de cualquier método lo que provoca que la corrección y entendimiento de este tipo de programas sea casi ininteligible.
   2. LENGUAJES DE PROGRAMACIÓN IMPERATIVA: Basados en la perspectiva de funcionamiento del procesador, están orientados a una ejecución secuencial de las órdenes y el uso de almacenes de datos modificables.
   3. LENGUAJE DE PROGRAMACIÓN ESTRUCTURADA: En esta programación se utiliza una técnica que genera programas que sólo permiten utilizar tres estructuras de control:

1. Secuencias (instrucciones que se generan secuencialmente)

2. Alternativas (sentencias if)

3. Iterativas (bucles condicionales)

* 1. LENGUAJES DE PROGRAMACIÓN ORIENTADA A OBJETOS: Es la más novedosa, se basa en intentar que el código de los programas se parezca lo más posible a la forma de pensar de las personas. Cada objeto posee datos y métodos propios, por lo que los programadores se concentran en programar independientemente cada objeto y luego generar el código que inicia la comunicación entre ellos.
  2. LENGUAJES DE PROGRAMACIÓN LÓGICA: Centrados en la perspectiva del razonamiento humano, tratan de solucionar el problema desde un punto de vista lógico
  3. LENGUAJES DE PROGRAMACIÓN FUNCIONAL: Orientados directamente al proceso de solución del problema y basados generalmente en estructuras de datos subyacentes como las listas encadenadas
  4. LENGUAJES DE PROGRAMACIÓN CONCURRENTE: Estos lenguajes incorporan aspectos de los paradigmas anteriores, principalmente del imperativo, y han sido diseñados para permitir a descripción de varios procesos que se ejecutan simultáneamente.

1. SEGÚN LAS APLICACIONES A QUE SE DESTINEN

## CÓDIGO FUENTE, CÓDIGO OBJETO Y CÓDIGO EJECUTABLE; MÁQUINAS VIRTUALES:

Un programa escrito en un lenguaje de alto nivel, no puede ser ejecutado directamente por un ordenador, sino que debe ser traducido a lenguaje máquina. Las etapas por las que debe pasar un programa escrito en un lenguaje de programación, hasta poder ser ejecutable son:

1. Programa fuente: Programa escrito en un lenguaje de alto nivel.
2. Compilador: Programa encargado de traducir los programas fuentes escritos en un lenguaje de alto nivel a lenguaje máquina y de comprobar que las llamadas a las funciones de librería se realizan correctamente.
3. Programa (o código) objeto: Es el programa fuente traducido (por el compilador) a código máquina. Aún no es directamente ejecutable.
4. Programa Ejecutable: Traducción completa a código máquina, realizada por el enlazador, del programa fuente y que ya es directamente ejecutable.
5. Linker (montador o enlazador): Es el programa encargado de insertar al programa objeto el código máquina de las funciones de las librerías (archivos de biblioteca) usadas en el programa y realizar el proceso de montaje, que producirá un programa ejecutable

Como cada lenguaje de programación tiene unas reglas especiales (sintaxis) debe existir un compilador específico para cada lenguaje de programación. Si el programa fuente es sintácticamente correcto, el compilador generará el código objeto, en caso contrario mostrará una lista con los errores encontrados.

Los errores en tiempo de compilación son los que se producen antes de la ejecución del programa, durante el proceso de compilación del programa. Los errores en tiempo de ejecución son los que se producen durante la ejecución del programa.

Aunque al compilar un programa no de errores, el programa puede funcionar incorrectamente y/o a dar errores durante su ejecución.

Los errores que se pueden producir en la fase de compilación son:

• Errores fatales: Son raros. Indican errores internos del compilador. Cuando ocurren, la compilación se detiene inmediatamente.

• Errores de sintaxis: Son los errores típicos de sintaxis. No detienen la compilación sino que al finalizar ésta se mostrará la lista con todos los errores encontrados.

• Advertencias o avisos (warnings): Indican que hay líneas de código sospechosas que a pesar de no infringir ninguna regla sintáctica, el compilador las encuentra susceptibles de provocar un error.

**BYTECODE(MÁQUINA VIRTUAL)**

Este bytecode es un código intermedio entre el código fuente y el código máquina. Suele tratárselo como un fichero binario que contiene un programa ejecutable similar a un módulo objeto. Los programas en bytecode suelen ser interpretados por un intérprete de bytecode (en general llamado máquina virtual, dado que es análogo a un ordenador). Su ventaja es su portabilidad: el mismo código binario puede ser ejecutado en diferentes plataformas y arquitecturas. Es la misma ventaja que presentan los lenguajes interpretados.

**Librerías:**

Junto con los compiladores, se incluyen ciertos ficheros llamados librerías. Las librerías contienen el código objeto de muchos programas que permiten hacer cosas comunes.

## PROCESO DE OBTENCIÓN DE CÓDIGO EJECUTABLE A PARTIR DEL CÓDIGO FUENTE; HERRAMIENTAS IMPLICADAS:

1. **TRADUCTORES DEL LENGUAJE:**

Los traductores de lenguaje son programas que traducen a su vez los programas fuente escritos en lenguajes de alto nivel a código máquina. Los traductores se dividen en compiladores e intérpretes.

* 1. INTÉRPRETES: Un intérprete es un traductor que toma un programa fuente, lo traduce y, a continuación, lo ejecuta.
  2. COMPILADORES: Como a hemos visto anteriormente, un compilador es un programa que traduce los programas fuente escritos en lenguaje de alto nivel a lenguaje máquina. La traducción del programa completo se realiza en una sola operación denominada compilación del programa; es decir, se traducen todas las instrucciones del programa en un solo bloque.

1. **DEPURADORES:**

Como hemos visto anteriormente, además de los errores de compilación, también existen los errores de ejecución. Para este tipo de errores aparecen los depuradores. Encontrar la causa que los provoca es una labor en ocasiones complicada, razón por la cual los EDI nos proporcionan una herramienta llamada Depurador que nos ayuda a encontrar los errores lógicos y demás errores producidos en tiempo de ejecución.

Un depurador (debugger), es un programa diseñado específicamente para la detección, verificación y corrección de errores. Los depuradores nos permiten trazar el programa (ejecutarlo sentencia a sentencia) y visualizar el contenido de las variables y direcciones de memoria durante la ejecución del programa.

# [**Modelos de Desarrollo de Software**](https://educacionadistancia.juntadeandalucia.es/centros/cordoba/mod/resource/view.php?id=120814)

## Ingeniería de software.

Disciplina que estudia los principios y metodologías para el desarrollo y mantenimiento de sistemas software. El desarrollo de software se estructura habitualmente en las siguientes fases:

1. Análisis
2. Diseño
3. Codificación
4. Pruebas
5. Mantenimiento

1. Análisis: Se determina y define claramente las necesidades del cliente y se especifica los requisitos que debe cumplir el software a desarrollar.

2. Diseño: Se descompone y organiza el sistema en elementos componentes que pueden ser desarrollados por separado. Se especifica la interrelación y funcionalidad de los elementos que componen el sistema.

3. Codificación: Se realiza la implementación en código fuente de cada uno de los componentes diseñados en la fase anterior.

4. Pruebas: El principal objetivo de las pruebas debe ser conseguir que el programa funcione incorrectamente y que se descubran defectos. Por tanto, debemos someter al programa al máximo número de casos diferentes que nos permitan encontrar la mayor cantidad de errores posibles.

5. Mantenimiento: Durante la explotación del sistema software es necesario realizar cambios ocasionales. Para ello hay que rehacer parte del trabajo realizado en las fases previas. Tipos de mantenimiento:

* Correctivo: se corrigen defectos.
* Perfectivo: se mejora la funcionalidad.
* Evolutivo: se añade funcionalidades nuevas.
* Adaptativo: se adapta a nuevos entornos.

## Modelos de desarrollo de Software

Según el planteamiento que se haga existen distintos modelos de desarrollo de software. Algunos tipos son:

-Modelos clásicos (predictivos)

* Modelo en cascada
* Modelo en V

-Modelo de construcción de prototipos

-Modelo evolutivos o incrementales

* Modelo en espiral (iterativos)
* Metodologías ágiles

## Explicación Modelos

Modelo Cascada: Modelo de mayor antigüedad. Identifica las fases principales del desarrollo software. Las fases han de realizarse en el orden indicado. El resultado de una fase es la entrada de la siguiente fase. Es un modelo bastante rígido que se adapta mal al cambio continuo de especificaciones. Existen diferentes variantes con mayor o menor cantidad de actividades. Entre sus mayores inconvenientes es que la identificación de errores se produce muy tarde.

Modelo En V: Modelo muy parecido al modelo en cascada. Visión jerarquizada con distintos niveles. Los niveles superiores indican mayor abstracción. Los niveles inferiores indican mayor nivel de detalle. El resultado de una fase es la entrada de la siguiente fase. Existen diferentes variantes con mayor o menor cantidad de actividades.

Modelo de construcción de prototipos (I): A menudo los requisitos no están especificados claramente por no existir experiencia previa o por omisión o falta de concreción del usuario/cliente.

Proceso de elaboración:

* Se crea un prototipo durante la fase de análisis y es probado por el cliente para refinar los requisitos del software a desarrollar.
* Se repite el paso anterior las veces necesarias.

Según la estrategia de prototipado que se haya utilizado podemos distinguir dos tipos de prototipos:

• Prototipos rápidos:

– El prototipo puede estar desarrollado usando otro lenguaje y/o herramientas.

– Finalmente el prototipo se desecha.

• Prototipos evolutivos:

– El prototipo está diseñado en el mismo lenguaje y herramientas del proyecto.

– El prototipo se usa como base para desarrollar el proyecto.

Metodologías ágiles (I): Son métodos de ingeniería del software basados en el desarrollo iterativo e incremental. Los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto. El trabajo es realizado mediante la colaboración de equipos auto-organizados y multidisciplinarios, inmersos en un proceso compartido de toma de decisiones a corto plazo. Dos de las metodologías ágiles más habituales son:

• Scrum

• Kanban

Kanban: Desarrollado inicialmente por Toyota para la industria de fabricación de productos. Enfocado a entregar el máximo valor para los clientes, utilizando los recursos justos.