UT 1. Desarrollo de software.

1. Concepto de programa informático. Instrucciones y datos.

1.1. Programa.

Definición software RAE

1. Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.
2. Estructuras de datos que permiten que los programas manipulen de forma adecuada la información.
3. Información descriptiva tanto en papel como en formas virtuales que describen la operación y usos del programa.

Para comprender mejor el concepto software, es necesario hacer referencia a las distintas partes del hardware.

Concepto de programa informático

Un programa informático es un grupo de instrucciones que están escritas en un

lenguaje de programación sobre el que se aplican una serie de datos para resolver un problema. Es decir, el ordenador necesita que esté en lenguaje máquina y para ello tendremos que usar un compilador. Una vez hecho esto tendremos que procesar todas las instrucciones pasándolas a la memoria principal.

1.2. Componentes del sistema informático.

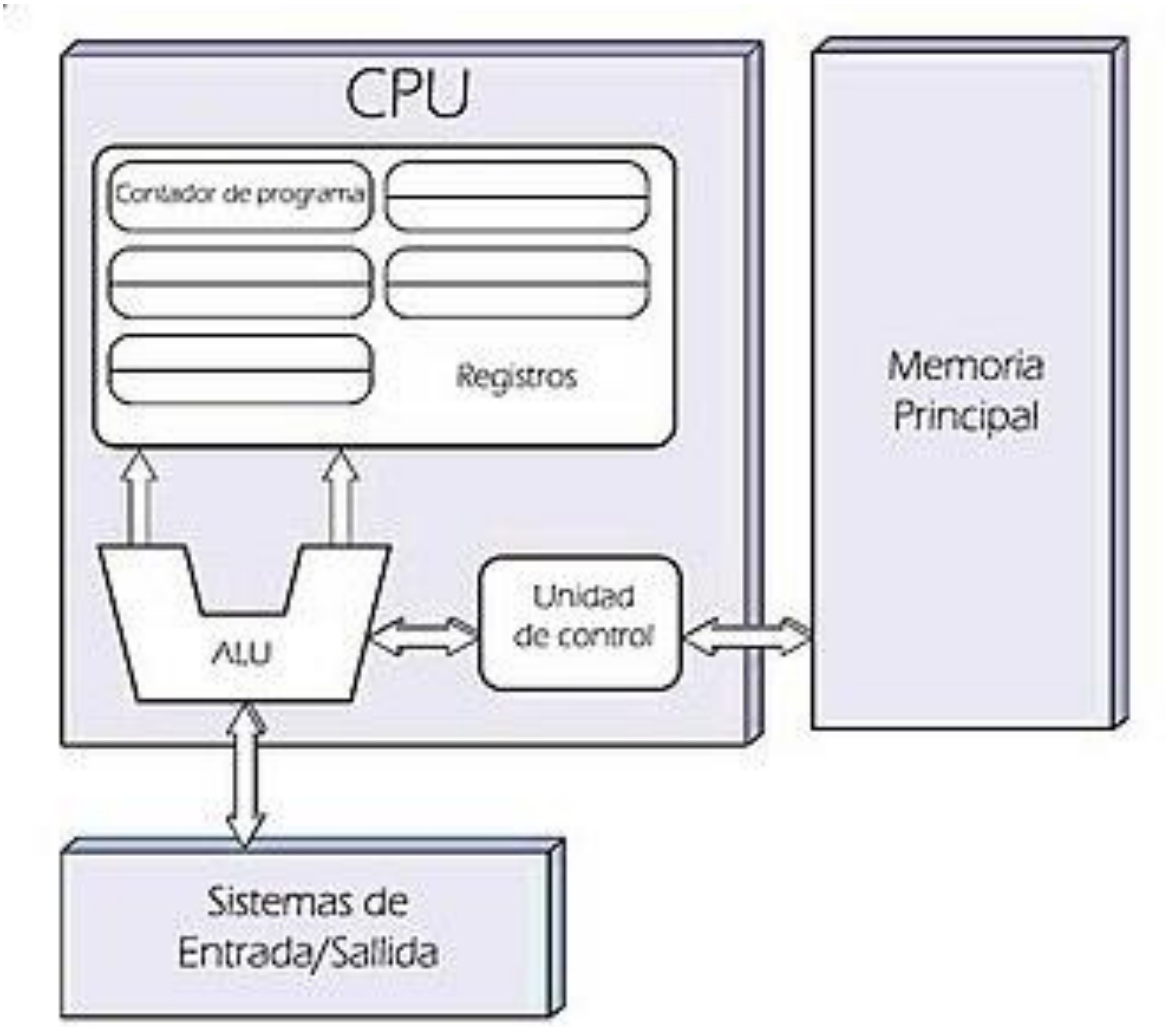
Si queremos iniciar un programa necesitaremos recursos hardware del ordenador, como son el procesador, la memoria RAM, dispositivos E/S, etc. Las instrucciones para inicializar el programa se cargan en la memoria principal y se ejecutarán en la CPU (en inglés, Central Processing Unit). Si vemos la arquitectura Von Neumann entenderemos cómo funcionan los componentes que conforman la CPU:

1. Unidad central de proceso (CPU): Es la parte del ordenador que ejecuta las instrucciones contenidas en los programas, aunque sean complejas se materializan en operaciones aritméticas(sumas, restas) y lógicas (OR, AND) que se realizan sobre bits. La CPU consta de :

• La Unidad Aritmético-Lógica (ALU) (UAL): es la que recibe los datos y ejecuta operaciones de cálculo y comparaciones, además de tomar decisiones lógicas (si son verdaderas o falsas), pero siempre supervisada por la Unidad de Control.

• La Unidad de Control (UC): se encarga de interpretar y ejecutar las instrucciones que se almacenan en la memoria principal y, además, genera las señales de control necesarias para ejecutarlas, mandando señales a la ALU y los registros.

• Los registros: son los que almacenan la información temporal, almacenamiento interno de la CPU. Intervienen en la ejecución de las instrucciones. Contiene el contador de programa y el registro de instrucción.



A continuación, vamos a ver los diferentes registros que posee la UC:

• **Contador de programa (CP):** contendrá la dirección de la siguiente instrucción para realizar, su valor será actualizado por la CPU después de capturar una instrucción.

• **Registro de Instrucción (RI):** es el que contiene el código de la instrucción, se analiza dicho código. Consta de dos partes: el código de la operación y la dirección de memoria en la que opera.

• **Registro de dirección de memoria (RDM):** tiene asignada una dirección correspondiente a una posición de memoria que va a almacenar la información mediante el bus de direcciones.

• **Registro de intercambio de memoria (RIM):** recibe o envía, según si es una operación de lectura o escritura, la información o dato contenidos en la posición apuntada por el RDM.

• **Decodificador de instrucción (DI):** extrae y analiza el código de la instrucción contenida en el RI y, además, genera las señales para que se ejecute correctamente la acción.

• **El Reloj**: marca el ritmo del DI y nos proporciona unos impulsos eléctricos con intervalos constantes a la vez que marca los tiempos para ejecutar las instrucciones.

• **El secuenciador**: son órdenes que se sincronizan con el reloj para que ejecuten correctamente y de forma ordenada la instrucción.

2. Memoria principal (RAM): contiene instrucciones del programa que hay que ejecutar y los datos sobre los que se debe operar estas instrucciones. La CPU toma las instrucciones de la RAM y envía las órdenes para su ejecución. Memoria volátil cuando se apaga desaparece.
3. Unidad o sistema de entrada salida: proporciona información al ordenador con el exterior transfiriendo información a través de periféricos
    a. Entrada
    b. Salida
    c. Entrada/Salida.

Para la interconexión de todos los elementos existen las conexiones llamadas Buses.

Cuando ejecutamos una instrucción podemos distinguir dos fases:

1) **Fase de búsqueda**: se localiza la instrucción en la memoria principal y se envía a la Unidad de Control para poder procesarla.

2) **Fase de ejecución:** se ejecutan las acciones de las instrucciones.

Para que podamos realizar operaciones de lectura y escritura en una celda de memoria se utilizan el RDM, el RIM y el DI. El decodificador de instrucción es el encargado de conectar la celda RDM con el registro de intercambio RIM, el cual posibilita que la transferencia de datos se realice en un sentido u otro según sea de lectura o escritura.

1.3. Tipos de software

Podemos dividir el software en dos categorías: según las tareas que realiza y según su método de distribución. Además, si tenemos en cuenta la licencia por la cual se distribuye, se clasifica en: software libre, software propietario y software de dominio público.

1.3.1. Software basado en el tipo de función que realizan

sistema operativo,
software de programación
software de aplicaciones.

1.3.2. Software basado en el tipo de trabajo que realizan

• **Software de sistema:** es el que hace que el hardware funcione. Está formado por programas que administran la parte física e interactúa entre los usuarios y el hardware. Algunos ejemplos son los sistemas operativos, los controladores de dispositivos, las herramientas de diagnóstico, etc.
• **Software de aplicación:** aquí tendremos los programas que realizan tareas específicas para que el ordenador sea útil al usuario. Por ejemplo, los programas ofimáticos, el software médico o el de diseño asistido, etc.
• **Software de programación o desarrollo:** es el encargado de proporcionar al programador las herramientas necesarias para escribir los programas informáticos y para hacer uso de distintos lenguajes de programación. Entre ellos encontramos los entornos de desarrollo integrado (IDE).

1.3.3. Software basado en el método de distribución

Según esta clasificación, distinguimos tres tipos de software:
• **Shareware**: donde los usuarios pueden pagar y después descargar el aplicativo desde internet. Por ejemplo, PowerDVD.
• **Freeware:** donde los usuarios Software pueden descargar el aplicativo de forma gratuíta, pero que mantiene los derechos de autor. Por ejemplo, Avast.
• **Adware**: es un aplicativo donde se ofrece publicidad incrustada, incluso en la instalación del mismo. Por ejemplo, CCleaner.

1.3.4. Licencias de software. Software libre y propietario.

Una licencia es un contrato entre el desarrollador de un software y el usuario final. En él se especifican los derechos y deberes de ambas partes. Es el desarrollador el que especifica qué tipo de licencia distribuye.
Existen tres tipos de licencias:
• **Software libre**: el autor de la licencia concede libertades al usuario, entre ellas están:

- Libertad para usar el programa con cualquier fin.

- Libertad para saber cómo funciona el programa y adaptar el código a nuestras propias necesidades.
- Libertad para poder compartir copias a otros usuarios.
- Libertad para poder mejorar el programa y publicar las modificaciones realizadas.

• **Software propietario**: este software no nos permitirá acceder al código fuente del programa y de forma general nos prohibirá la redistribución, la reprogramación, la copia o el uso simultáneo en varios equipos. Pueden darse dos variantes vistas anteriormente: freeware y shareware

• **Software de dominio público**: este software no pertenece a ningún propietario y carece de licencia, por lo que todo el mundo lo puede utilizar. Incluso podremos realizar una oferta para adquirirlo bajo el código fuente de dominio público.
La licencia que más se usa en el software libre es la licencia GPL (GNU General Public License – Licencia Pública General) que nos dejará usar y cambiar el programa, con el único requisito que se hagan públicas las modificaciones realizadas

2. Código fuente, código objeto y código ejecutable; máquinas virtuales.

En la etapa de diseño construimos las herramientas de software capaces de generar un código fuente en lenguaje de programación. Estas pueden ser los diagramas de flujo o el pseudocódigo.

Le etapa de codificación es la encargada de generar el código fuente y pasa por diferentes estados.

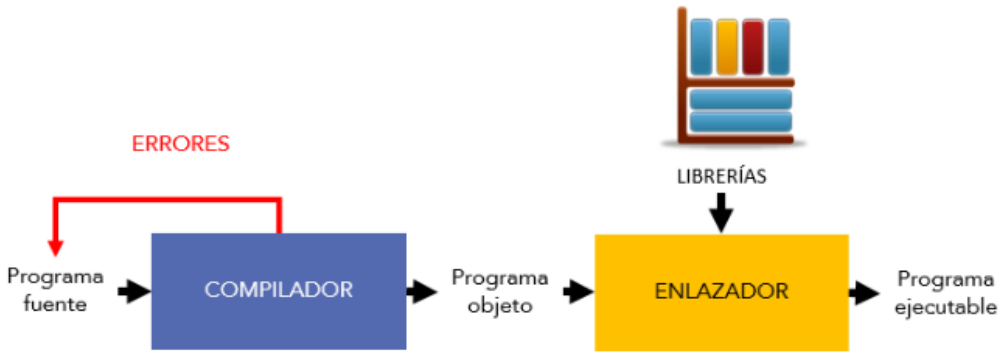
2.1. Tipos de código

Cuando escribimos un código pasa por distintos estados hasta que se ejecuta:

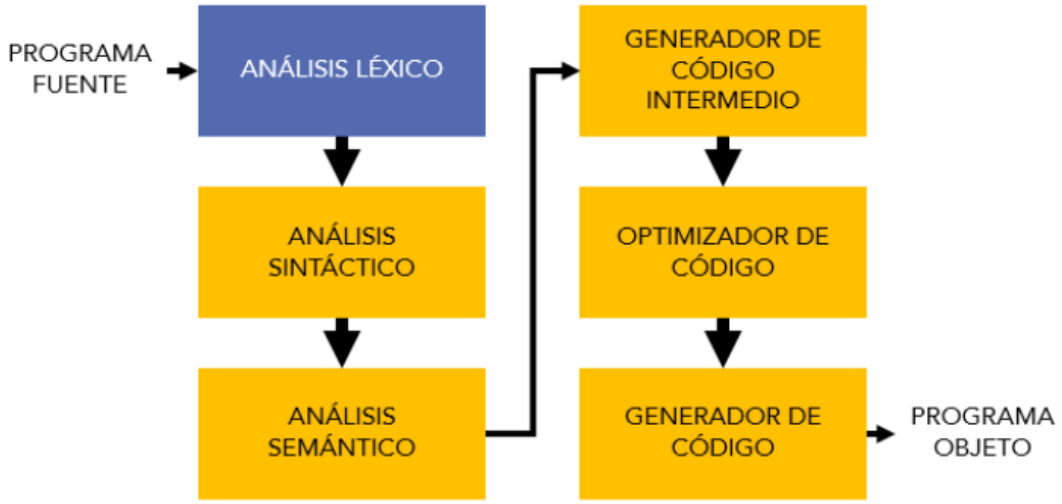
- **Código fuente**: es el código realizado por los programadores usando algún editor de texto o herramienta de programación. Posee un lenguaje de alto nivel y para escribirlo se parte de los diagramas de flujo o pseudocódigos. No se puede ejecutar directamente en el ordenador.
- **Código objeto**: es el código que se crea tras realizar la compilación del código fuente. Este código no es entendido ni por el ordenador ni por nosotros. Es una representación intermedia de bajo nivel.
- **Código ejecutable**: este código se obtiene tras unir el código objeto con varias librerías para que así pueda ser ejecutado por el ordenador.

Compilación
La compilación es el proceso a través del cual se convierte un programa en lenguaje máquina a partir de otro programa de computadora escrito en otro lenguaje. La compilación se realiza a través de dos programas: el compilador y el enlazador. Si en el compilador se detecta algún tipo de error no se generará el

código objeto y tendremos que modificar el código fuente para volver a pasarlo por el compilador



Dentro del compilador tendremos varias fases en las que se realizan distintas operaciones:



Análisis léxico: se lee el código obteniendo unidades de caracteres llamados tokens.
Ejemplo: la instrucción resta = 2 – 1, genera 5 tokens: resta, =, 2, -, 1.

- **Análisis sintáctico**: recibe el código fuente en forma de tokens y ejecuta el análisis para determinar la estructura del programa, se comprueba si cumplen las reglas sintácticas.
- **Análisis semántico**: revisa que las declaraciones sean correctas, los tipos de todas las expresiones, si las operaciones se pueden realizar, si los arrays son del tamaño correcto, etc.
- **Generación de código intermedio**: después de analizarlo todo, se crea una representación similar al código fuente para facilitar la tarea de traducir al código objeto.

- **Optimización de código**: se mejora el código intermedio anterior para que sea más fácil y rápido a la hora de interpretarlo la máquina.
- **Generación de código**: se genera el código objeto.

El enlazador insertará, en el código objetos, las librerías necesarias para que se pueda producir un programa ejecutable. Si se hacen referencia a otros ficheros que contengan las librerías especificadas en el código objeto, se combina con dicho código y se crea el fichero ejecutable.

2.2. Máquinas Virtuales

Una máquina virtual es un tipo de software capaz de ejecutar programas como si fuese una máquina real. Se clasifican en dos categorías:

- **Máquinas virtuales de sistema**. Nos permiten virtualizar máquinas con distintos sistemas operativos en cada una.
Un ejemplo son los programas VMware o Virtual Box que podremos usar para probar nuevos sistemas operativos o ejecutar programas.

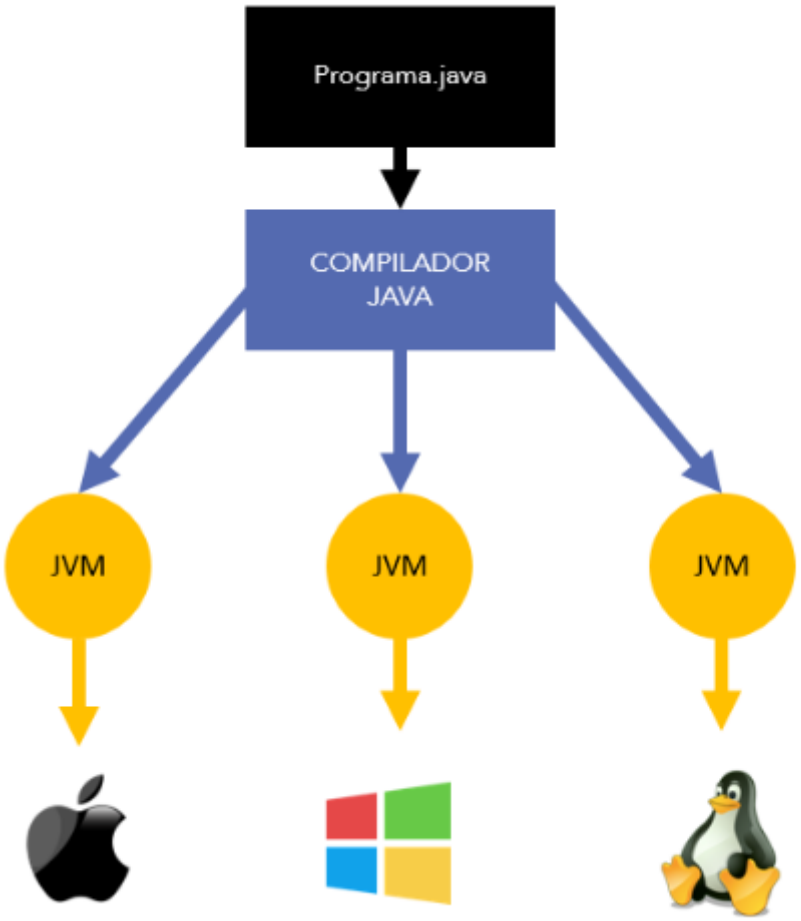
- **Máquinas virtuales de proceso**. Se ejecutan como un proceso normal dentro de un SO y solo soporta un proceso. Se inician cuando lanzamos el proceso y se detienen cuando este finaliza. El objetivo es proporcionar un entorno de ejecución independiente del hardware y del sistema operativo y permitir que el programa sea ejecutado de la misma forma en cualquier plataforma.
Ejemplo de ello es la máquina virtual de Java.

Las máquinas virtuales requieren de grandes recursos por lo que hay que tener cuidado y ejecutarlas en ordenadores capaces de soportar los procesos que requieren dichas máquinas para que no nos funcionen lentas o se colapsen.

La máquina virtual de Java
Los programas que se compilan en lenguaje Java son capaces de funcionar en cualquier plataforma (UNIX, Mac, Windows, Solaris, etc.). Esto se debe a que el código no lo ejecuta el procesador del ordenador sino la propia Máquina Virtual de Java (JVM).

El funcionamiento básico de la máquina virtual es el siguiente:

1º) El código fuente estará escrito en archivos de texto planos con la extensión .java.
2º) El compilador javac generará uno o varios archivos siempre que no se produzcan errores y tendrán la extensión .class.
3º) Este fichero .class contendrá un lenguaje intermedio entre el ordenador y el SO y se llamará bytecode.
4º) La Java VM coge y traduce el bytecode en código binario para que el procesador de nuestro ordenador sea capaz de reconocerlo. Los ficheros .class podrán ser ejecutados en múltiples plataformas.



Entre las tareas que puede realizar la máquina virtual Java pueden estar:
• La reserva de espacio para objetos creados y liberar aquella memoria que no se usa.
• Comunicación con el sistema en el que se ejecuta la aplicación para varias funciones.
• Observar que se cumplen las normas de seguridad para las aplicaciones Java.
Una de las desventajas de usar este tipo de lenguajes que se basan en una máquina virtual puede ser más lentos que los lenguajes ya compilados, debido a

la capa intermedia. No obstante, cabe destacar que no es una desventaja demasiado crítica.

3. Lenguajes de programación

a. Tipos de lenguaje
b. Características de los lenguajes más difundidos.

3.1. Tipos de lenguaje

Tipos de lenguajes de programación. Clasificación y características de los lenguajes más difundidos

un programa informático es un conjunto de instrucciones escritas en un lenguaje de programación.

Lenguaje de programación hace referencia al conjunto de caracteres, reglas y acciones combinadas y consecutivas que un equipo debe ejecutar.

Constará de los siguientes elementos:

• **Alfabeto o vocabulario**: conjunto de símbolos permitidos.
• **Sintaxis**: reglas para realizar correctamente construcciones con los símbolos.
• **Semántica**: reglas que determinan el significado de construcción del lenguaje

Elementos básicos de los lenguajes:

Identificadores: nombres simbolicos
Constantes: datos que no cambian
Operadores: símbolos que respresentas operaciones entre variables
Instrucciones: símbolos especiales que representas estructuras de procesamiento y de definición de elementos de programacion
Comentarios: texto que se usa para comentarios de programa.

3.2. Clasificación y características

Podemos clasificar los lenguajes de programación basándonos en los siguientes criterios:

Según su nivel de abstracción:	Lenguajes de bajo nivel
	Lenguajes de nivel medio
	Leguajes de alto nivel
Según la forma de ejecución:	Lenguajes compilados
	Lenguajes interpretados
Según el paradigma de programación:	Lenguajes imperativos
	Lenguajes funcionales
	Lenguajes lógicos
	Lenguajes estructurados
	Lenguajes orientados a objetos

3.1.1. Según su nivel de abstracción.

- **Lenguajes de bajo nivel:**

El lenguaje de más bajo nivel por excelencia es el lenguaje máquina, el que entiende directamente la máquina. Utiliza el lenguaje binario (0 y 1) y los programas son específicos para cada procesador.
Al lenguaje máquina le sigue el lenguaje ensamblador. Es complicado de aprender y es específico para cada procesador. Cualquier programa escrito en este lenguaje tiene que ser traducido al lenguaje máquina para que se pueda ejecutar. Se utilizan nombres mnemotécnicos y las instrucciones trabajan directamente con registros de memoria física.

- **Lenguajes de nivel medio:**

Posee características de ambos tipos de nivel, tanto del nivel bajo como del alto, y se suele usar para la creación de sistemas operativos. Un lenguaje de nivel medio es el lenguaje C.

- **Lenguajes de alto nivel:**

Este tipo de lenguaje es más fácil a la hora de aprender, ya que para usarlo lo hacemos con palabras que solemos utilizar. El idioma que se suele emplear es el inglés y para poder ejecutar lo que escribamos necesitaremos un compilador para que traduzca al lenguaje máquina las instrucciones.
Este lenguaje es independiente de la máquina ya que no depende del hardware del ordenador.

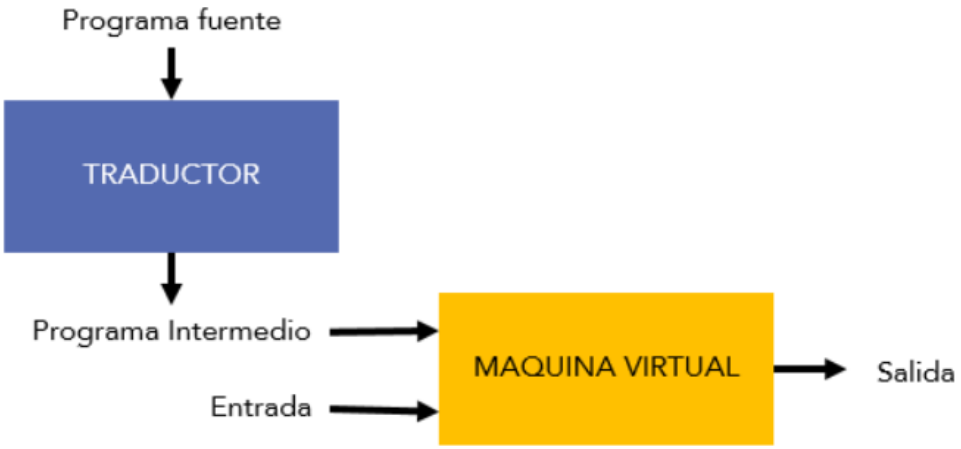
Algunos ejemplos de lenguajes de alto nivel son: ALGOL, C++, C#, Clipper, COBOL, Fortran, Java, Logo, Pascal, etc.

3.1.2. Según la forma de ejecución.

- **Lenguajes compilados:** Al programar en alto nivel hay que traducir ese lenguaje a lenguaje máquina a través de compiladores. Los compiladores traducen desde un lenguaje fuente a un lenguaje destino. Devolverá errores si el lenguaje fuente está mal escrito y lo ejecutará si el lenguaje destino es ejecutable por la máquina.

Ejemplo: C, C++, C#, Objetive-C.

Lenguajes interpretados:
Son otra variante para traducir programas de alto nivel. En este caso nos da la apariencia de ejecutar directamente las instrucciones del programa fuente con las entradas proporcionadas por el usuario. Cuando ejecutamos una instrucción, se debe interpretar y traducir al lenguaje máquina.
El compilador es, de forma general, más rápido que un intérprete al asignar las salidas. Sin embargo, al usar el intérprete evitaremos tener que compilar cada vez que hagamos alguna modificación.
Ejemplos de algunos lenguajes son: PHP, JavaScript, Python, Perl, Logo, Ruby, ASP, Basic, etc.
El lenguaje Java usa tanto la compilación como la interpretación. Un programa fuente en Java puede compilarse primero en un formato intermedio, llamado bytecodes, para que luego una máquina virtual lo interprete.



Según el paradigma de programación

El paradigma de programación nos detalla las reglas, los patrones y los estilos de programación que usan los lenguajes. Cada lenguaje puede usar más de un paradigma, el cual resultará más apropiado que otro según el tipo de problema que queramos resolver.
Existen diferentes categorías de lenguaje:

- **Lenguajes imperativos:** Al principio, los primeros lenguajes imperativos que se usaron fueron el lenguaje máquina y, más tarde, el lenguaje ensamblador. Ambos lenguajes consisten en una serie de sentencias que establecen cómo debe manipularse la información digital presente en cada memoria o cómo se debe enviar o recibir la información en los dispositivos. La sentencia principal es la asignación. A través de las estructuras de control podemos establecer el orden en que se ejecutan y modificar el flujo del programa según los resultados de las acciones.

Algunos ejemplos de estos lenguajes son: Basic, Fortran, Algol, Pascal, C, Ada, C++, Java, C#.
Casi todos los lenguajes de desarrollo de software comercial son imperativos.
Dentro de esta categoría podremos englobar:
• Programación estructurada.
• Programación modular.
• Programación orientada a objetos (usa objetos y sus interacciones para crear programas).

- **Lenguajes funcionales**: Están basados en el concepto de función y estarán formados por definiciones de funciones junto con argumentos que se aplican. Entre sus características, destacan que no existe la operación de

asignación las variables almacenan definiciones a expresiones, la operación fundamental es la aplicación de una función a una serie de argumentos y, además, la computación se realiza evaluando expresiones.
Algunos ejemplos de este tipo de lenguaje son: Lisp, Scheme, ML, Miranda o Haskell. Apenas se usan para el software comercial.

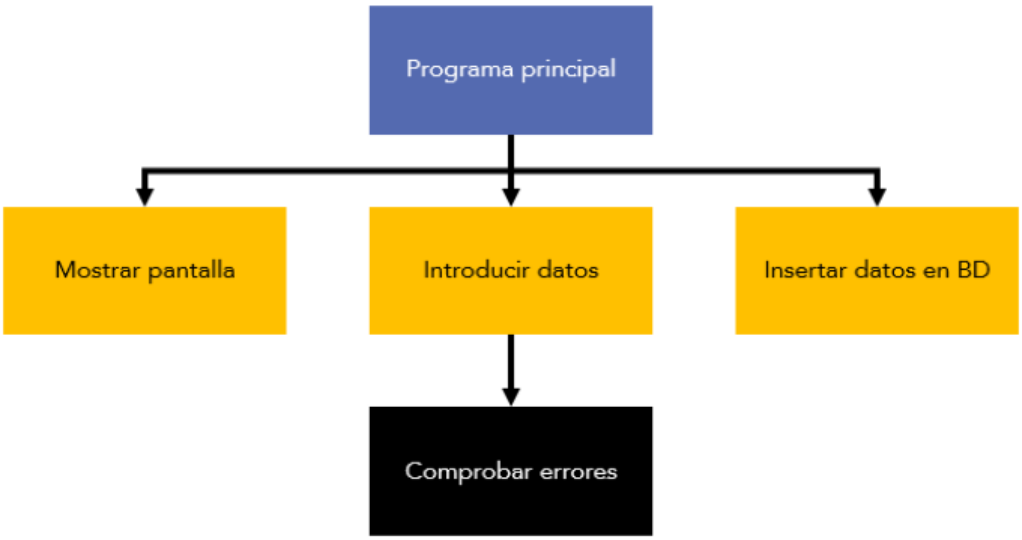
- **Lenguajes lógicos**: Están basados en el concepto de razonamiento, ya sea de tipo deductivo o inductivo. A partir de una base de datos consistente en un conjunto de entidades, propiedades de esas entidades o relaciones entre entidades, el sistema es capaz de hacer razonamientos.

Los programas escritos en este lenguaje suelen tener forma de una base de datos, la cual está formada por declaraciones lógicas que podremos consultar. La ejecución será en forma de consultas hacia esa base de datos.
El lenguaje lógico más importante es Prolog, especialmente preparado para sistemas expertos, demostración de teoremas, consultas de bases de datos relacionales y procesamiento de lenguaje natural.

- **Lenguajes estructurados**: Utilizan las tres construcciones lógicas nombradas anteriormente y resulta fácil de leer. El inconveniente de estos programas estructurados es el código, que está centrado en un solo bloque, lo que dificulta el proceso de hallar el problema.

Cuando hablamos de programación **estructurada** nos estamos refiriendo a programas creados a través de módulos, es decir, pequeñas partes más manejables que, unidas entre sí, hacen que el programa funcione. Cada uno de los módulos poseen una entrada y una salida, y deben estar perfectamente comunicados, aunque cada uno de ellos trabaja de forma independiente.

A continuación, vemos un programa estructurado en módulos:

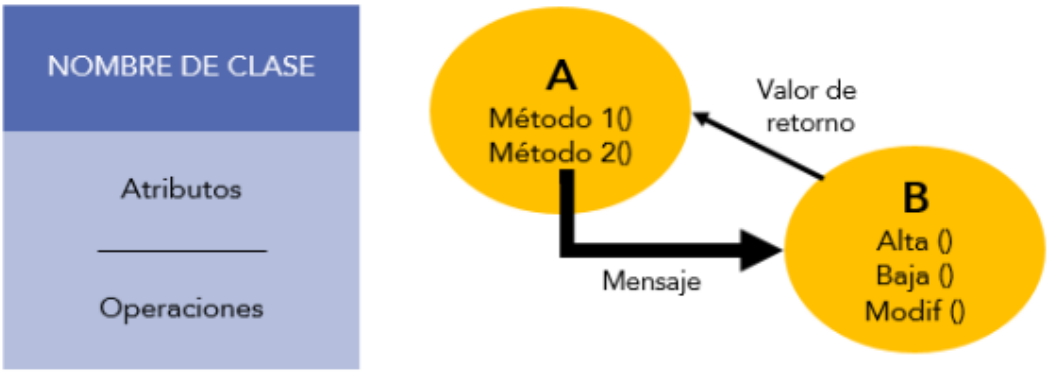


La evolución a esta programación mediante módulos se le denomina

programación modular y posee las siguientes ventajas:
• Al dividir el programa en módulos, varios programadores podrán trabajar a la vez en cada uno de ellos.
• Estos módulos pueden usarse para otras aplicaciones.
• Si surge algún problema será más fácil y menos costoso detectarlo y abordarlo, ya que se puede resolver de forma aislada.
Algunos ejemplos de este lenguaje son: Pascal, C, Fortran, Modula-2, etc.

- **Lenguajes orientados a objetos**: Este lenguaje estará definido por un conjunto de objetos en vez de por módulos como hemos visto anteriormente. Estos objetos están formados por una estructura de datos y por una colección de métodos que interpretan esos datos. Los datos que se encuentran dentro de los objetos son sus atributos y las operaciones que se realizan sobre los objetos cambian el valor de uno o más atributos

La comunicación entre objetos se realiza a través de mensajes, como se plasma en la siguiente figura:



Una clase es una plantilla para la creación de objetos. Al crear un objeto se ha de especificar a qué clase pertenece para que el compilador sepa qué características posee.
Entre las ventajas de este tipo de lenguaje, hay que destacar la facilidad para reutilizar el código, el trabajo en equipo o el mantenimiento del software. Por el contrario, su principal desventaja es la dificultad para programar en este lenguaje, ya que es muy poco intuitivo.
Algunos ejemplos de lenguajes orientados a objetos son: C++, Java, Ada, Smalltalk, etc.

4. Introducción a la ingeniería del software.

Ingeniería del software: (término propuesto a finales de los 60).
"La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software", IEEE93.
La ingeniería de Software es una ***disciplina de la ingeniería*** *que está relacionada con* ***todos los aspectos de la producción de software.***
"**Disciplina de la ingeniería":**

Se aplican teorías, métodos y herramientas en la forma adecuada para encontrar soluciones a problemas incluso cuando no hay teorías que lo soporten.
"**Todos los aspectos de la producción de software**":

No sólo están relacionados con los temas tecnológicos, sino que también interviene en la planificación y gestión del proyecto, selección de herramientas, etc.

Nuestro enfoque:
La Ingeniería de Software es una disciplina de diseño y desarrollo de software de ALTA CALIDAD.

4.1. Proceso software y ciclo de vida del software

Modelo de proceso del software: Es una estrategia de desarrollo que se debe emplear para resolver problemas de la industria de software.

Ciclo de vida es el periodo de tiempo que comienza cuando se concibe un producto Software y finaliza cuando el producto pierde su utilidad.

No existe un modelo de propósito general, pero si las normas ISO/IEC 12207 y la IEEE 1074.

4.2. Fases del desarrollo software.

Estas etapas son:

1. **Fase inicial**

Planificación del proyecto y resolución de problemas.

- Planificación del proyecto
- Estimación de costes (viabilidad del proyecto)
- Es la fase más **compleja**
- Precisa de expertos en planificación de proyectos
- Se desarrollan documentos muy importantes para el proyecto (viabilidad, estimación…)

2. **Fase de análisis**

Se especifican los requisitos funcionales y no funcionales del sistema.

- Consiste en **analizar** el problema
- Recopilar, examinar y formular los requisitos del cliente (especificación de requisitos)
- Análisis de restricciones, entrevistas con el cliente y los usuarios finales
- Se genera un **documento vinculante** (a modo de contrato) entre el cliente y el desarrollador

3. **Fase de diseño**

Se divide el sistema en partes y se determina la función de cada una.

- Consiste en determinar los **requisitos** generales de la arquitectura de la aplicación
- Se define cada subconjunto de la aplicación
- Los documentos son mucho más técnicos
- Esta fase involucra a los jefes de proyecto, arquitectos de software y analistas
- Los programadores aún no intervienen en esta fase

4. **Fase de implementación**

Se elige un Lenguajes de Programación y se codifican los programas.

- Consiste en **implementar** el software usando lenguajes de programación, librerías, frameworks…
- Se crea documentación muy detallada en la que se incluye y documenta el código fuente
- Parte del código suele comentarse sobre el propio código fuente generado
- Se detallan las entradas, salidas, parámetros… de cada uno de los módulos del programa.
- El detalle es máximo, pensando en el mantenimiento y soporte futuro que tendrá el programa

5. **Pruebas**

Se prueban los programas para detectar errores y se depuran.

- Servirán para corregir posibles errores, tendremos dos tipos de errores:
    - Sintácticos: Producidos por un mal uso del lenguaje.
    - Semánticos: Lo que está equivocado es la solución que yo he ideado.
- Se realizan pruebas para garantizar que la aplicación se ha programado según las especificaciones
- A modo preliminar, se pueden considerar dos categorías de pruebas:
    - **Pruebas funcionales**: se prueba que a la aplicación hace lo que tiene que hacer (con el cliente)
    - **Pruebas estructurales**: se efectúan pruebas técnicas sobre el sistema (estrés, carga, integración…)

6. **Explotación**

Instalamos, configuramos y probamos la aplicación en los equipos del cliente.

- Se instala el software en el entorno real de uso
- Se trabaja con el software de forma cotidiana (nuevas necesidades, incidencias…)
- Se recogen los errores y las correcciones en un nuevo documento de mantenimiento

- Los programadores y analistas revisan esos **fallos** para mejorar el software y **aprender** de los errores

7. **Mantenimiento**

Se mantiene el contacto con el cliente para actualizar y modificar la aplicación el futuro.

- Se realizan **procedimientos correctivos** sobre el programa
- Siempre hay que tener delante la documentación técnica de la aplicación
- Las operaciones de mantenimiento se deben documentar para dejar constancia de los cambios
- Mantenimiento: Hay tres tipos:
    - Correctivo: Sirve para corregir posibles errores o fallos del programa.
    - Perfectivo: Se usa para perfeccionarlo.
    - Adaptativo: Sirve para adaptarlo a nuevas situaciones.

8. **Retirada**

- El software ha llegado al **final** de su vida útil
- No resulta rentable seguir ampliándolo o manteniéndolo
- Llegados a este punto, el ciclo puede comenzar de nuevo:
- Comprando un nuevo software
- Desarrollando un nuevo software (a medida)

9. **Documentación**

De todas las etapas, se documenta y guarda toda la información.

- Como hemos explicado en la sección anterior, la documentación es vital
- En cada fase se generan uno o más documentos
- La documentación es vital para saber cómo usar la aplicación final
- Como mínimo, cada aplicación debe tener estos documentos:
    - **Manual de usuario**: explica cómo usará el usuario la aplicación
    - **Manual técnico**: documentación dirigida a los técnicos (administradores y programadores)
    - **Manual de instalación**: detalla el proceso de instalación de la aplicación

4.3. Roles del desarrollo de software

Arquitecto de software:

- Decide cómo se realiza el proyecto y cómo va a estructurarse
- Tiene un amplio conocimiento de las tecnologías, los frameworks y las librerías
- Decide y forma los recursos del desarrollo de un proyecto

Jefe de proyecto:

- Dirige el curso del proyecto
- Puede ser un analista con experiencia, un arquitecto o una persona dedicada a ese puesto en exclusividad
- Debe saber gestionar un equipo y lidiar con los tiempos
- Trata de manera continua y fluida con el cliente

Analista de sistemas:

- Esta persona realiza un estudio exhaustivo del problema que va a llevarse a cabo
- Efectúa el análisis y el diseño de todo el sistema
- Este perfil requiere mucha experiencia, y también suele involucrarse en reuniones con el cliente

Analista programador:

- Esta persona realiza un estudio exhaustivo del problema que va a llevarse a cabo
- Efectúa el análisis y el diseño de todo el sistema
- Este perfil requiere mucha experiencia, y también suele involucrarse en reuniones con el cliente

Programador:

- Conoce en profundidad el lenguaje de programación
- Se encarga de codificar las tareas encomendadas por el analista o el analista programador
- Su misión es la de codificar y probar los diferentes módulos de la aplicación

4.4. Modelos de proceso de desarrollo software

Modelos de desarrollo de software

- **Modelo en cascada.** Consiste en dividir el proceso de desarrollo en fases, cada una de las cuales se ejecuta una vez que la anterior ha finalizado. El modelo en cascada es un modelo secuencial, es decir, las fases se ejecutan una tras otra.
    - **Sin realimentación**: es el modelo de vida clásico del software. Es prácticamente imposible que se pueda utilizar, ya que requiere conocer de antemano todos los requisitos del sistema. Sólo es aplicable a pequeños desarrollos, ya que las etapas pasan de una a otra sin retorno posible. (se presupone que no habrá errores ni variaciones del software).
    - **Con realimentación**: es uno de los modelos más utilizados. Proviene del modelo anterior, pero se introduce una realimentación entre etapas, de

forma que podamos volver atrás en cualquier momento para corregir, modificar o depurar algún aspecto. No obstante, si se prevén muchos cambios durante el desarrollo no es el modelo más idóneo. Es el modelo perfecto si el proyecto es rígido (pocos cambios, poco evolutivo) y los requisitos están claros.

- **Modelo evolutivo**. tiene en cuenta la naturaleza cambiante y evolutiva del software. La idea detrás de este modelo es el desarrollo de una implantación del sistema inicial, exponerla a los comentarios del usuario, refinarla en N versiones hasta que se desarrolle el sistema adecuado. Una ventaja de este modelo es que se obtiene una rápida realimentación del usuario, ya que las actividades de especificación, desarrollo y pruebas se ejecutan en cada iteración. Distinguimos dos variantes:
    - Modelo Iterativo Incremental. Está basado en el modelo en cascada con realimentación, donde las fases se repiten y refinan, y van propagando su mejora a las fases siguientes.
    - Modelo en Espiral. Es una combinación del modelo anterior con el modelo en cascada. En él, el software se va construyendo repetidamente en forma de versiones que son cada vez mejores, debido a que incrementan la funcionalidad en cada versión. Es un modelo bastante complejo.

4.5. Herramientas de apoyo al desarrollo del software.

En la práctica, para llevar a cabo varias de las etapas vistas en el punto anterior contamos con herramientas informáticas, cuya finalidad principal es automatizar las tareas y ganar fiabilidad y tiempo.

Esto nos va a permitir centrarnos en los requerimientos del sistema y el análisis de este, que son las causas principales de los fallos del software.

Las herramientas **CASE** son un 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.

El desarrollo rápido de aplicaciones o **RAD** es un proceso de desarrollo de software que comprende el desarrollo iterativo, la construcción de prototipos y el uso de utilidades CASE. Hoy en día se suele utilizar para referirnos al desarrollo rápido de interfaces gráficas de usuario o entornos de desarrollo integrado completos.

La tecnología CASE trata de automatizar las fases del desarrollo de software para que mejore la calidad del proceso y del resultado final.

En concreto, estas herramientas permiten:

- Mejorar la planificación del proyecto.
- Darle agilidad al proceso.
- Poder reutilizar partes del software en proyectos futuros.

- Hacer que las aplicaciones respondan a estándares.
- Mejorar la tarea del mantenimiento de los programas.
- Mejorar el proceso de desarrollo, al permitir visualizar las fases de forma gráfica.

CLASIFICACIÓN

Normalmente, las herramientas CASE se clasifican en función de las fases del ciclo de vida del software en la que ofrecen ayuda:

U-CASE: ofrece ayuda en las fases de planificación y análisis de requisitos.

M-CASE: ofrece ayuda en análisis y diseño.

L-CASE: ayuda en la programación del software, detección de errores del código, depuración de programas y pruebas y en la generación de la documentación del proyecto.

Ejemplos de herramientas CASE libres son: ArgoUML, Use Case Maker, ObjectBuilder...