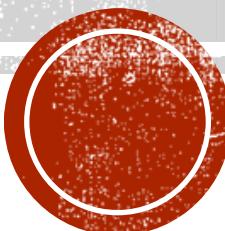


# **1. DESARROLLO DE SOFTWARE**



# 1 EL SOFTWARE DEL ORDENADOR

- El ordenador se compone de dos partes: **Hardware (HW) y Software (SW)**
  - **Hardware** está formado por los componentes físicos que se pueden ver y tocar.
  - El Software forma la parte lógica del ordenador que no se puede tocar.
- Entre las definiciones que podemos encontrar de Software podemos encontrar las siguientes:
  - La RAE, lo define como “el conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en un ordenador”
  - La IEEE (Institute of Electrical Engineers), lo define como “el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación”
- En cualquier caso, **el software es todo aquello que se refiere a los programas y datos almacenados en un ordenador.**
- Podemos clasificar el software de dos maneras: según la tarea que la realiza y según el método de distribución.



# 1.1 SOFTWARE BASADO EN EL TIPO DE TAREA QUE REALIZA

- Se distinguen tres tipos de software:
- 1. **Software de sistema**
  - Permite que el HW funcione. Formado por programas que permiten la administración de la parte física. Ej: Sist. operativos, controladores de dispositivos, herramientas de diagnóstico, etc.
- 2. **Software de aplicación**
  - Lo forman los programas que nos ayudan a realizar tareas específicas en cualquier campo. Este software hace que el ordenador sea una herramienta útil para el usuario. Ejemplos: Aplicaciones ofimáticas, software educativo, aplicaciones de contabilidad, etc.
- 3. **Software de programación o desarrollo**
  - Es el que proporciona al programador herramientas para ayudarle a escribir programas informáticos y a usar diferentes lenguajes de programación de forma práctica.
  - Entre ellos se encuentran los entornos de desarrollo integrados (IDE), que agrupan las anteriores herramientas, normalmente en un entorno visual, de forma que el programador no necesite introducir múltiples comandos para compilar, interpretar, depurar, etc. Habitualmente cuentan con una avanzada interfaz gráfica de modo usuario (GUI).



# 1.2 SOFTWARE BASADO EN EL MÉTODO DE DISTRIBUCIÓN

- 1. **Shareware.**

- Es una modalidad de distribución de software, tanto juegos como programas utilitarios, para que el usuario pueda evaluar de forma gratuita el producto por un tiempo especificado.
  - Para adquirir una licencia de SW que permita su uso completa se requiere un pago.

- 2. **Freeware.**

- Se distribuye sin cargo. A veces se incluye el código fuente, pero no es lo usual.
  - Suele incluir una licencia de uso, que permite su redistribución pero con algunas restricciones, como no modificar la aplicación en sí, ni venderla y dar cuenta de su autor.
  - No todos los programas de software libre son necesariamente freeware

- 3. **Adware**

- Son programas Shareware que de forma automática descargan publicidad en nuestro ordenador cuando los ejecutamos o instalamos. Al comprar la licencia del programa se elimina la publicidad.



- 4. **Software multimedia**

- Se refiere a los programas utilizados para presentar de una forma integrada texto, gráficos, sonidos, etc.
- Este tipo de software es considerado como una nueva tecnología.

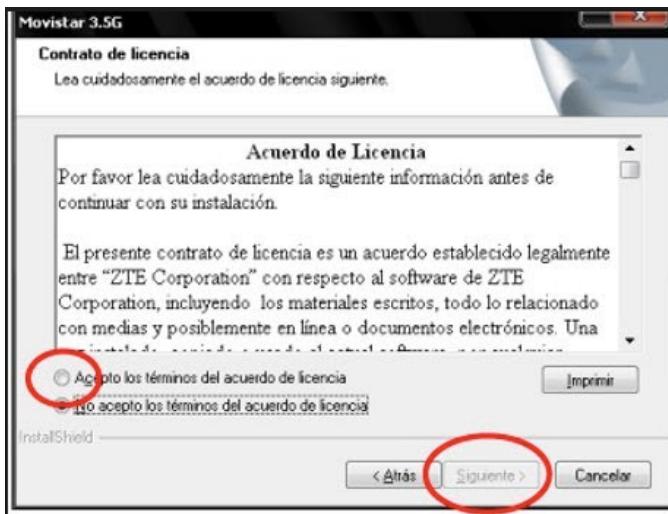
- 5. **Software de uso específico**

- Es el que se desarrolla para resolver un problema determinado de alguna organización o persona.
- Este tipo de software puede ser muy variado, desde programas de gestión de un videoclub, los que se usan en un instituto para registrar las calificaciones o los que se usan en los bancos para el control de las cuentas y los clientes.



# 1.3 LICENCIAS DE SOFTWARE. SOFTWARE LIBRE Y PROPIETARIO

- Una **licencia de software** es un contrato que se establece entre el desarrollador de un software, sometido a propiedad intelectual y derechos de autor, y el usuario, en el cual se definen con precisión los derechos y deberes de ambas partes.
- Es el desarrollador, o aquel a quien este haya cedido los derechos y deberes de explotación, quién eligen la licencia según la cual distribuye el software.



- El **software libre** es aquel en el que el autor cede una serie de libertades básicas al usuario, en el marco de una licencia, que establece las siguientes libertades:
  - 1. Libertad de utilizar el programa con cualquier en cuantos ordenadores se deseé.
  - 2. Libertad de estudiar cómo funciona el programa y de adaptar su código a necesidades específicas; para ello, como condición previa, es necesario poder acceder al código fuente.
  - 3. Libertad de distribuir copias a otros usuarios (con o sin modificaciones)
  - 4. Libertad de mejorar el programa (ampliar, añadir funciones) y de hacer públicas y distribuir al públicas las modificaciones. Para ello, como condición previa, es necesario poder al código fuente.
- El **software propietario** es aquel que, habitualmente, se distribuye en formato binario, sin posibilidad de acceder al código fuente según una licencia según la cuál el propietario, por regla general, prohíbe alguna o todas las siguientes posibilidades: redistribución, modificación, copia, uso en varias máquinas, transferencia de titularidad, etc. Freeware y Shareware son software propietario.
- Finalmente, un **software de dominio público** es aquel que carece de licencia o no hay forma de determinarla pues se desconoce al autor.

- La licencia más utilizada en los productos y desarrollos de software libre y de fuentes abiertas es la licencia GPL (GNU General Public License – Licencia Pública General) que da derecho al usuario a usar y modificar el programa con la obligación de hacer públicas las versiones modificadas de este.
- Aquí puedes ver más información sobre licencias de software libre compatibles con la licencia GPL. <http://www.gnu.org/licenses/license-list.es#SoftwareLicenses>

# 2 EL PROGRAMA INFORMÁTICO

- **Definición de programa informático:** “Un programa informático es un conjunto de instrucciones que se ejecutan de una manera secuencial con el objetivo de realizar una o varias tareas en un sistema”
- Un programa informático es creado por un programador en un lenguaje determinado, que será compilado y ejecutado por un sistema. Cuando un programa es llamado para ser ejecutado, el procesador ejecuta el código compilado del programa instrucción por instrucción.
- Imaginemos que tenemos un programa extremadamente sencillo que pide por teclado dos números y los suma. Aunque este programa tendría más instrucciones (por ejemplo para pedir la entrada de los números), la instrucción que realizaría la operación de nuestro programa se podría corresponder con la siguiente:
  - $\text{resultado} = \text{valor1} + \text{valor2}$



- Definición de **lenguaje de programación**: “Un lenguaje de programación es un conjunto de instrucciones, operadores y reglas de sintaxis y semánticas, que se ponen a disposición del programador para que éste puede comunicarse con los dispositivos hardware y software existentes”
- Este idioma artificial que se crea y que está constituido por los operadores, instrucciones y reglas tiene como objetivo facilitar la tarea de crear programas.
- En un principio, todos los programas eran creados por el único código que el ordenador era capaz de entender: **el código máquina**, un conjunto de ceros y unos de grandes proporciones. Este método hacía que fuera una labor sumamente tediosa.
- Por eso se decidió como solución establecer un nombre a las secuencias de programación más frecuentes (instrucciones) e indicando a la vez la posición de memoria donde se colocaban los resultados y el todo, dando paso al **lenguaje ensamblador**.
- Este lenguaje ensamblador aunque facilitó la tarea, seguía siendo complicado programar, por lo que aparecieron otros lenguajes de alto nivel como FORTRAN. Los lenguajes de alto nivel tratan de que nos olvidemos del lenguaje máquina y programar sea una tarea lo más liviana, entendible e intuitiva posible. Al final el compilador es el que conseguirá que nuestro código sean 1s y 0s.

## 2.1 TIPOS DE CÓDIGO

- Los distintos tipos de código por los que pasará nuestro programa antes de ser ejecutado por el sistema son:
  - **Código fuente.**- El código fuente de un programa informático es un conjunto de instrucciones escritas en un lenguaje de programación determinado. Es decir, el código en el que nosotros escribimos nuestro programa.
  - **Código objeto.**- El código objeto es el resultante de compilar el código fuente. Si se trata de un lenguaje de programación compilado, el código objeto será código máquina, mientras que si se trata de un lenguaje de programación virtual, será código bytecode.
  - **Código ejecutable.**- El código ejecutable es el resultado obtenido de enlazar nuestro código objeto con las librerías. Este código ya es nuestro programa ejecutable, programa que se ejecutará directamente en nuestro sistema o sobre una máquina virtual en el caso de los lenguajes de programación virtuales.



## 2.2 COMPILACIÓN

- Aunque el proceso de obtener nuestro código ejecutable pasa tanto por un compilador como por un enlazador, se suele llamar al proceso completo "**compilación**".
- Todo este proceso se lleva a cabo mediante dos programas: compilador y enlazador. Mientras que el **enlazador** une el código objeto con las librerías, el trabajo del compilador es mucho más completo.
- **Fases de compilación:**
  - Análisis lexicográfico: Se buscan palabras reservadas, operaciones, caracteres de puntuación, etc.
  - Análisis sintáctico-semántico: Se revisa la coherencia. Por ejemplo, si los tipos de datos son correctos.
  - Generación de código intermedio.
  - Optimización de código
  - Generación de código
  - Enlazador de librerías



# 3 PROCESOS DE DESARROLLO

- El proceso de desarrollo del software implica un conjunto de actividades que se tienen que planificar y gestionar de tal manera que aseguren un producto final que dé solución a las necesidades de todas aquellas personas que lo van a utilizar. A Todo este proceso es lo que se conoce como el CICLO DE VIDA DEL SOFTWARE.
- El **ciclo de vida de un producto software** comprende el periodo que transcurre desde que el producto es concebido hasta que deja de estar disponible o es retirado.
- Normalmente, se divide en etapas y en cada etapa se realizarán una serie de tareas.
- Hay más de un modelo de etapas de desarrollo como el iterativo incremental o en espiral, que de modo recurrente suelen ser compatibles y usadas entre sí, sin embargo vamos a estudiar uno de los modelos más extendidos y completos, el **modelo en cascada**.
- En este modelo, cada etapa tiene como entrada uno o varios documentos procedentes de las etapas anteriores y produce otros documentos de salida, por ello una tarea importante en cada etapa es la DOCUMENTACIÓN.



## 3.1 MODELO EN CASCADA



## 3.2 ANÁLISIS

- Lo más importante del éxito de un proyecto software es entender y comprender el problema que se necesita resolver, y una vez comprendido darle solución.
- En esta fase se analizan y especifican los requisitos o capacidades que el sistema debe tener porque el cliente así lo ha pedido.
- La obtención de requisitos no es tarea fácil
  - El cliente puede no tenerlos claros
  - Pueden surgir nuevos requisitos
  - Puede cambiar lo especificado
  - Pueden existir malos entendidos por falta de conocimiento del equipo de desarrollo sobre el problema a resolver
  - El cliente puede no expresarse de forma clara debido a la falta de conocimientos informáticos.
- Para realizar un proyecto satisfactorio es necesario obtener unos buenos requisitos, y para ello es esencial una buena comunicación entre el cliente y desarrolladores.



- Para facilitar esta comunicación se utilizan varias técnicas, algunas son las siguientes:
  - 1. Entrevistas.- Es la más tradicional. Consiste en hablar con el cliente.
  - 2. Desarrollo conjunto de aplicaciones (JAD, Joint Application Development).- Se apoya en la dinámica de grupos, es un tipo de entrevista muy estructurada aplicable a grupos de personas (usuarios, administradores, analistas, desarrolladores, etc.). Cada persona juega un rol concreto.
  - 3. Planificación conjunta de requisitos (JRP, Joint Requirements Planning).- Es un subconjunto de JAD, se caracterizan por estar dirigidas a la alta dirección y en consecuencia los productos resultantes son los requisitos de alto nivel o estratégicos.
  - 4. Brainstorming.- Es un tipo de reuniones en grupo cuyo objetivo es generar ideas desde diferentes puntos de vista para la resolución de un problema.



- 5. Prototipos.- Es una versión inicial del sistema, se utiliza para clarificar algunos puntos, demostrar los conceptos, en definitiva, para enterarse más acerca del problema y sus posibles soluciones. Después se tira o bien se usa como base para añadir más cosas.
- 6. Casos de uso.- Es la técnica definida en la UML (Unified Modeling Language), se basa en la representación de escenarios que describen el comportamiento deseado del sistema, es decir, lo que queremos que haga el sistema. Representan requisitos funcionales del mismo. Es importante resaltar que describen qué hace el sistema, no cómo lo hace. Se especifican dos tipos de requisitos
  - Funcionales.- Describen con detalle la función que realiza el sistema, como reacciona ante determinadas entradas, usos básicos, etc.
  - No funcionales.- Tratan sobre las características del sistema, como puede ser la fiabilidad, mantenibilidad, sistema operativo, plataforma HW, restricciones, limitaciones, etc.

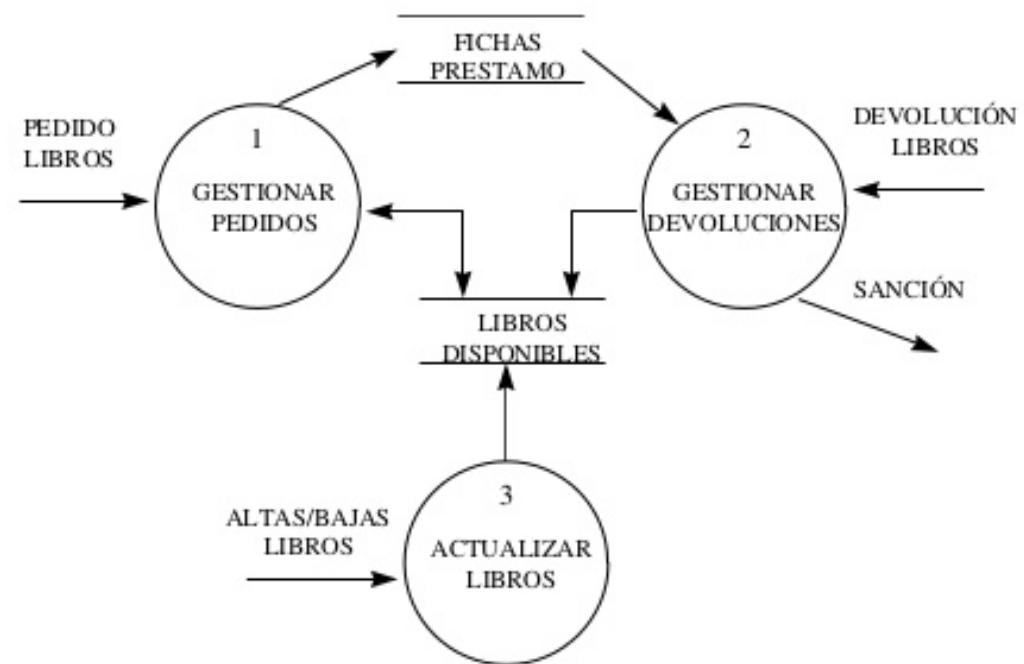


<b>Requisitos funcionales</b>	<b>Requisitos no funcionales</b>
El usuario puede agregar un nuevo contacto	La aplicación debe funcionar en sistemas operativos Windows y Linux
El usuario puede ver una lista con todos los contactos	El tiempo de respuesta a consultas, altas, bajas y modificaciones debe ser inferior a 5 segundos
A partir de la lista de contactos el usuario puede acceder a los datos de un contacto	Utilizar un sistema gestor de base de datos para almacenar los datos
El usuario puede eliminar un contacto o varios desde la lista	Espacio libre en disco: 1GB

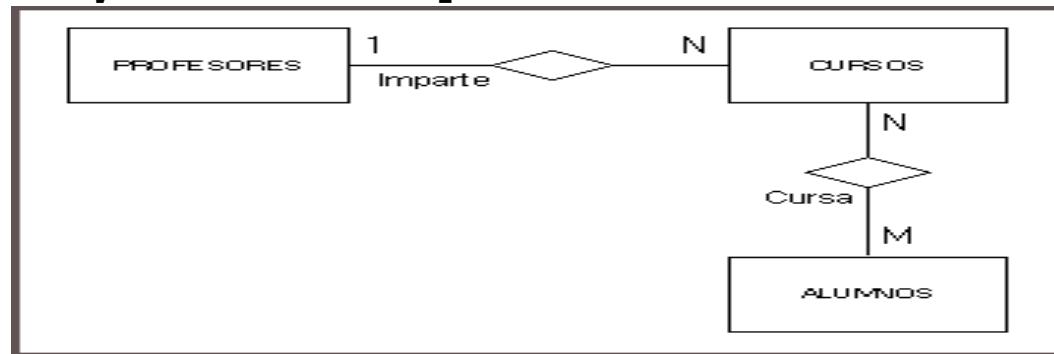
- Para especificar los requisitos se utilizan diferentes técnicas:
- 1. **Diagramas de flujo de datos, DFD.**
  - Es un diagrama que represente el flujo de datos entre los distintos procesos, entidades y almacenes que forman el sistema.
  - Los procesos identifican funciones dentro del sistema y se representan mediante burbujas ovaladas o circulares.
  - Las entidades externas representan componentes que no forman parte del sistema (persona, departamento, etc.), pero proporcionan datos al sistema y se representan mediante rectángulos.
  - Los almacenes representan los datos desde el punto de vista estático, es decir, representan el lugar donde se almacenan los datos procesados y se representan mediante dos líneas horizontales y paralelas.
  - Por último, el flujo de datos representa el movimiento de datos dentro del sistema y se representa mediante flechas.



### DIAGRAMA 0: GESTIONAR BIBLIOTECA



- 2. **Diagramas de flujo de control, DFC.** Similar a los DFD pero con la diferencia de que muestra el flujo de control, en lugar de datos.
- 3. **Diagramas de transición de estados, DTE.** Representa cómo se comporta el sistema como consecuencia de sucesos externos.
- 4. **Diagrama entidad/relación, DER (también conocido, como Modelo E/R o MER).** Usado para representar los datos y la forma en la que se relacionan entre ellos.



- 5. **Diccionario de datos.** Es una descripción detallada de los datos utilizados por el sistema que gráficamente se encuentran representados por los flujos de dato y almacenes sobre el DFD.



- Al final de esta etapa debemos tener un documento de Especificación de Requisitos del Software (ERS).
- Este documento no debe tener ambigüedades, debe ser completo, consistente, fácil de verificar y modificar, fácil de utilizar en la fase de explotación y mantenimiento y fácil de identificar el origen y consecuencias de los requisitos.
- Sirve como entrada para la siguiente para la siguiente fase en el desarrollo de la aplicación.
- La estructura de este documento según el estándar 830 (IEEE, 1998) es la siguiente: <https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf>



- De manera general, en este documento quedan especificados:
  - La planificación de las reuniones que van a tener lugar.
  - Relación de los objetivos del usuario cliente y del sistema.
  - Relación de los requisitos funcionales y no funcionales del sistema.
  - Relación de objetivos prioritarios y temporización.
  - Reconocimiento de requisitos mal planteados o que conllevan contradicciones, etc.
- Todo aquello que no se detecte, o resulte mal entendido en la etapa inicial provocará un fuerte impacto negativo en los requisitos, propagando esta corriente degradante a lo largo de todo el proceso de desarrollo e incrementando su perjuicio cuanto más tardía sea su detección.



## 3.3 DISEÑO

- Una vez identificados los requisitos es necesario componer la forma en que se solucionará el problema.
- En esta etapa se traducen los requisitos funcionales y no funcionales en una representación de software aunque sin entrar en detalles.
- Se define por tanto el entorno que requerirá el sistema, aunque también se puede establecer en sentido contrario, es decir, diseñar el sistema en función de los recursos de los que se dispone.
- Se crearán los diagramas de casos de uso y de secuencia para definir la funcionalidad del sistema.
- Se definirá también el formato de la información de entrada y salida, las estructuras de datos y la división modular (se debe dividir el sistema en partes y establecer qué relaciones habrá entre ellas así como qué hace cada parte).
- En definitiva, debemos crear un modelo funcional-estructural de los requerimientos del sistema global, para poder dividirlo y afrontar las partes por separado.



## 3.4 CODIFICACIÓN

- Una vez realizado el diseño se realiza el proceso de codificación.
- En esta etapa, el programador recibe las especificaciones del diseño y las transforma en un conjunto de instrucciones escritas en un lenguaje de programación, almacenadas dentro de un programa.
- A este conjunto de instrucciones se le llama código fuente.
- Cuanto más exhaustivo ha sido el análisis y diseño, esta tarea será más sencilla aunque siempre es posible que se necesite un nuevo análisis o rediseño al encontrar problemas al programar el software.
- En cualquier proyecto en el que trabaja un grupo de personas debe haber unas normas de codificación y estilo, claras y homogéneas. Estas normas facilitan las tareas de corrección y mantenimiento de los programas, sobre todo cuando se realizan por personas que no lo han desarrollado.



- Las características deseables de todo código son:
  - Modularidad: que esté dividido en trozos más pequeños.
  - Corrección: que haga lo que se le pide realmente.
  - Fácil de leer: para facilitar su desarrollo y mantenimiento futuro.
  - Eficiencia: que haga un buen uso de los recursos.
  - Portabilidad: que se pueda implementar en cualquier equipo.
- Se recomienda a la vez que se va desarrollando código ir escribiendo los manuales técnicos y de referencia necesarios, así como la parte inicial correspondiente del manual de usuario. Esta documentación es esencial para la etapa de pruebas y mantenimiento, así como para la entrega final del producto.



## 3.5 PRUEBAS

- En esta etapa ya se dispone del software y se trata de encontrar errores, no solo de codificación sino también relativos a la especificación o el diseño.
- Durante la prueba del software se realizarán tareas de verificación y validación del software.
- El objetivo en esta etapa es planificar y diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.
- Una prueba tiene éxito si descubre un error no detectado hasta entonces (en caso de existirlo).
- Un caso de prueba es un documento que especifica los valores de entrada, salida esperada y las condiciones previas para la ejecución de la prueba.



- **Algunas recomendaciones sobre las pruebas:**
  - Cada prueba debe definir los resultados de salida esperados.
  - Un programador o una organización debe evitar sus propios programas.
  - Es necesario revisar los resultados de cada prueba en profundidad.
  - Las pruebas deben incluir datos de entrada válidos y esperados, así como no válidos e inesperados.
  - No planear pruebas asumiendo que no se encontrarán errores.
  - La probabilidad de encontrar errores en un parte del software es proporcional al número de errores ya encontrados.
  - Las pruebas son una tarea creativa.
- **No es un proceso estático, y es usual realizar pruebas después de otras etapas, como la documentación.**



- Entre todas las pruebas que se efectúan sobre el software podemos distinguir básicamente.
- 1) **PRUEBAS UNITARIAS**
  - Consisten en probar, una a una, las diferentes partes de software y comprobar su funcionamiento (por separado, de manera independiente).
  - Junit es el entorno de pruebas para Java
- 2) **PRUEBAS DE INTEGRACIÓN**
  - Se realizan una vez que se han con éxito las pruebas unitarias y consistirán en comprobar el funcionamiento del sistema completo: con todas sus partes interrelacionadas.
  - La parte final se denomina comúnmente Beta Test y se realiza a ser posible en los equipos del cliente y bajo un funcionamiento normal de su empresa (entorno de producción del software).



## 3.6 DOCUMENTACIÓN

- Todas las etapas del desarrollo deben quedar perfectamente documentadas.
- En esta etapa será necesario reunir todos los documentos generados y clasificarlos según el nivel técnico de sus descripciones.
- Esta documentación que se realiza del software tiene dos caras:
  - La documentación disponible para el usuario.
  - La documentación destinada al propio equipo de desarrollo.
- La documentación para el usuario debe mostrar una información completa y de calidad que ilustre mediante los recursos adecuados cómo manejar la aplicación. Una buena documentación debería permitir a un usuario cualquier comprender el propósito y el modo de uso de la aplicación sin información previa o adicional.
- La documentación técnica, destinada a equipos de desarrollo, que explica el funcionamiento interno del programa, haciendo especial hincapié en la codificación del mismo.



- Para los sistemas más pequeños, la documentación suele ser menos amplia, como mínimo se debe tratar de mantener la especificación del sistema, el documento de diseño y el código fuente del programa.
- Por desgracia, el mantenimiento de la documentación a menudo se descuida y puede no estar en sintonía con el software asociado, causando problemas para los usuarios y mantenedores del sistema.
- La mejor solución al problema de una documentación desfasada consiste en apoyar el mantenimiento de los documentos a herramientas software que registran los documentos, las relaciones entre ellos, avisan a los desarrolladores cuando hay cambios en un documento, registran posibles inconsistencias en la documentación, etc.

## 3.7 EXPLOTACIÓN

- Una vez que tenemos nuestro software y hemos comprobado tras las pruebas que el software es fiable y carece de errores, hay que prepararlo para su distribución. Esto es lo que comúnmente se conoce como Explotación.
- Aunque diversos autores consideran la explotación y el mantenimiento como la misma etapa, nosotros vamos a diferenciarlas en base al momento en que se realizan.
- La explotación es la fase en que los usuarios finales conocen la aplicación y comienzan a utilizarla y compone las fases de Instalación, puesta a punto y funcionamiento de la aplicación en el equipo final del cliente.
- Cabe destacar, que en caso de que nuestro software sea una versión sustitutiva de un software anterior es recomendable valorar la convivencia de sendas aplicaciones durante un proceso de adaptación.



## 3.8 MANTENIMIENTO

- Sería muy lógico pensar que con la entrega de nuestra aplicación (instalación y configuración del proyecto) hemos terminado nuestro trabajo. Sin embargo, son muy escasas las ocasiones en las que un software no vaya a necesitar de un mantenimiento continuado.
- En esta fase del desarrollo de software se arreglan los fallos o errores que suceden cuando el programa ya ha sido implementado en un sistema y se realizan las ampliaciones necesitadas o requeridas.
- Si el mantenimiento es para una ampliación, este modelo en cascada que hemos visto se vuelve cíclico, ya que será necesario analizar los requisitos, diseñar la ampliación, codificar la misma, probarla, documentarla, implementarla y, por supuesto, dar soporte de mantenimiento sobre la misma.
- Existen cuatro tipos de mantenimiento sobre un software: Adaptativo, Correctivo, perfectivo y preventivo



- **1. Mantenimiento adaptativo.**
  - Es posible que se produzcan cambios en el entorno original (CPU, SSOO, reglas de empresa, etc.)
  - Tiene como objetivo la modificación del producto por los cambios que se produzcan, tanto en el hardware como en el software del entorno rápido en el que se ejecuta.
  - Es el más usual debido a los cambios que se producen en la tecnología informática y que suelen dejar rápidamente los productos informáticos desarrollados.
- **2. Mantenimiento correctivo**
  - Se corrigen errores o defectos que, a pesar de las pruebas no se detectaron.
- **3. Mantenimiento perfectivo**
  - Conforme utiliza el cliente puede descubrir funciones adicionales que le pueden aportar beneficios. Se modifica el producto software para incorporar mejoras de rendimiento o mantenibilidad del producto.
- **4. Mantenimiento preventivo**
  - Se realizan mejoras y se facilitan tareas sin alterar especificaciones. Por ejemplo, se adapta, se mejora o se añaden comentarios al software.



# **3.9 ROLES QUE INTERACTÚAN EN EL DESARROLLO**

- Se deben realizar diversas y diferentes tareas en el desarrollo del software.
- Todo esto no lo realiza una única persona, sino que el personal que interviene en el desarrollo de un software es tan diverso como las diferentes tareas que se van a realizar.
- Los roles no son necesariamente rígidos y es habitual que participen en varias etapas del proceso de desarrollo.
- 1) **Analista de sistemas**
  - Uno de los roles más antiguos en el desarrollo del software. Su objetivo consiste en realizar un estudio del sistema para dirigir el proyecto en una dirección que garantice las expectativas del cliente determinando el comportamiento del software.
  - Participa en la etapa de análisis
- 2) **Diseñador de software**
  - Realiza el diseño de la solución que hay que desarrollar.
  - Participa en la etapa de diseño



- 3) **Analista programador**
  - Domina una visión más amplia de la programación, aportando una visión general del proyecto más detallada diseñando una solución más amigable para la codificación y participando activamente en ella.
  - Participa en las etapas de diseño y codificación.
- 4) **Programador**
  - Se encarga de manera exclusiva de crear el resultado del estudio realizado por analistas y diseñadores.
  - Escribe el código fuente del software.
  - Participa en la etapa de codificación.
- 5) **Arquitecto software**
  - Cohesiona todo el proceso de desarrollo.
  - Participa en las etapas de análisis, diseño, documentación y explotación.



# 4. LENGUAJES DE PROGRAMACIÓN

- Podemos definir un lenguaje de programación como un conjunto de caracteres, las reglas para la combinación de esos caracteres y un conjunto de reglas.
- Los lenguajes de programación se pueden clasificar atendiendo a varios criterios
  - Según su nivel de abstracción: Lenguajes de bajo nivel, lenguajes de nivel medio y lenguajes de alto nivel.
  - Según la forma de ejecución: lenguajes compilados y lenguajes interpretados
  - Según el paradigma de programación: lenguajes imperativos, funcionales, lógicos, estructurados y lenguajes orientados a objetos.



# 4.1 SEGÚN SU NIVEL DE ABSTRACCIÓN

## ▪ Lenguajes de bajo nivel

- Se acercan al funcionamiento de un ordenador.
- Se conocen también como lenguaje máquina, que es entendible únicamente por la máquina. Las instrucciones están formadas por ceros y unos.
- Son específicos para cada procesador.
- A este le sigue el lenguaje ensamblador. Este lenguaje es difícil de aprender y es específico para procesador. Un programa escrito en este lenguaje necesita ser traducido a lenguaje máquina para poder ejecutarse. Se utilizan nombres nemotécnicos.

-u 100 1a	OCFD :0100 BA0B01 OCFD :0103 B409 OCFD :0105 CD21 OCFD :0107 B400 OCFD :0109 CD21	MOV DX,0108 MOV AH,09 INT 21 MOV AH,00 INT 21	48 6F 6C 61 2C
-d 10b 13f	OCFD :0100 OCFD :0110 OCFD :0120 OCFD :0130 OCFD :0140	20 65 73 74 65 20 65 73-20 75 6E 20 70 72 6F 67 72 61 60 61 20 68 65 63-68 6F 20 65 6E 20 61 73 73 65 60 62 6C 65 72 20-70 61 72 61 20 6C 61 20 57 69 6B 69 70 65 64 69-61 24	

Hola,  
este es un progra  
rama hecho en as  
sembler para la  
wikipedia\$



- **Lenguajes de nivel medio**

- Estos lenguajes tienen ciertas características que los acercan a los lenguajes de bajo nivel, pero a la vez también tienen características de los lenguajes de alto nivel.
- Un lenguaje de programación de este tipo es el lenguaje C. Se suelen utilizar para aplicaciones como la creación de sistemas operativos.

- **Lenguajes de alto nivel**

- Son más fáciles de aprender porque están formados por palabras del lenguaje natural (inglés).
- Para poder ejecutarlos en el ordenador se necesita un programa intérprete o compilador que traduzca las instrucciones escritas en este lenguaje, en instrucciones de lenguaje máquina que el ordenador pueda entender.
- Son independientes de la máquina.
- Existen muchos: Basic, C++, COBOL, C#, JAVA, Pascal, Python, PL/SQL, etc.



## 4.2 SEGÚN LA FORMA DE EJECUCIÓN

### ▪ Lenguajes compilados

- Un programa que se escribe en un lenguaje de alto nivel tiene que traducirse a un código que pueda utilizar la máquina. Los programas traductores que realizan esta operación se llaman compiladores o intérpretes.
- Un compilador es un programa que puede leer un programa escrito en un determinado lenguaje (un lenguaje fuente) y traducirlo en un programa equivalente en otro lenguaje (destino).
- Hay que tener en cuenta que el compilador devolverá errores si el programa fuente no está bien escrito. El programa destino se podrá ejecutar si el lenguaje destino es directamente ejecutable por la máquina.

### ▪ Lenguajes interpretados

- Los intérpretes, son otra alternativa para traducir los programas escritos en lenguaje de alto nivel.
- En este caso, en lugar de producir un programa destino, nos da la apariencia de ejecutar directamente las operaciones especificadas por el fuente.
- Algunos ejemplos de lenguajes interpretados son: PHP, Java Script, Python, etc.
- Los procesadores del lenguaje JAVA combinan la compilación y la interpretación. Se compila primero a un lenguaje intermedio, llamado bytecodes, después una máquina virtual los interpreta.



## 4.3 SEGÚN EL PARADIGMA DE PROGRAMACIÓN

- Un paradigma de programación es un enfoque particular para la construcción de software.
- Define un conjunto de reglas, patrones y estilos de programación que usan los lenguajes de programación.
- Un lenguaje de programación puede usar más de un paradigma.
- Dependiendo del problema a resolver un paradigma resultará más apropiado que otro.
- Aunque hemos dicho que se pueden clasificar en lenguajes imperativos, funcionales, lógicos, estructurados y lenguajes orientados a objetos, únicamente vamos a ver los lenguajes imperativos y una parte especial de los lenguajes imperativos, los lenguajes orientados a objetos.



- **Lenguajes imperativos**

- En los lenguajes imperativos un cálculo consiste en una serie de sentencias que establecen explícitamente cómo se debe manipular la información digital presente en memoria, y/o cómo se debe recoger o enviar información desde/hacia los dispositivos.
- La sentencia principal es la asignación.
- Las estructuras de control permiten establecer el orden de la asignación y cambiar el flujo del programa dependiendo de los resultados de las acciones primitivas.
- La mayoría de los lenguajes usados para desarrollo de software comercial son imperativos.
- Ejemplos de lenguajes imperativos son: Basic, Fortran, Algol, C, Ada, C++, Java, C#.
- Dentro de esta categoría se engloba la programación estructura, la programación modular y la programación orientada a objetos.



- **Lenguajes de programación orientados a objetos**

- En la programación orientada a objetos un programa está compuesto por un conjunto de objetos no por un conjunto de instrucciones o un conjunto de módulos (como la no vista programación estructurada).
- Un objeto consta de una estructura de datos y de una colección de métodos u operaciones que manipulan esos datos. Los datos definidos dentro de ese objeto son sus atributos. Las operaciones definen el comportamiento y pueden cambiar el valor de uno o más atributos.
- Entre las ventajas de la programación orientada a objetos destacamos que facilita la reutilización de código, el trabajo en equipo o el mantenimiento de software. Sin embargo, la principal desventaja es la complejidad para adaptarse a esta programación ya que es menos intuitiva que la imperativa (de una manera simple) o la estructurada.
- Ejemplos de lenguajes orientados a objetos: C++, C#, Java, etc.



# 5. MÁQUINAS VIRTUALES

- Una máquina virtual es una aplicación software de una máquina (por ejemplo, un ordenador) que ejecuta los programas como si fuese una máquina real. Se pueden clasificar en dos categorías:
- 1) Máquinas virtuales de sistema (System Virtual Machine). Permiten ejecutar en la misma máquina física varias máquinas virtuales cada una con un sistema operativo, de esta manera pueden coexistir diferentes sistemas operativos sobre una misma máquina. Ejemplos de este software es Virtual Box o VMWare Workstation.
- Se pueden utilizar para evaluar y probar nuevos sistemas operativos, para ejecutar aplicaciones sobre distintos sistemas operativos, etc.
- Las máquinas virtuales agregan gran complejidad al sistema en tiempo de ejecución y pueden hacer que al ejecutar algún proceso el ordenador vaya lento; por lo tanto hay que tener en cuenta que la máquina donde se ejecute la máquina virtual debe tener una capacidad notable.

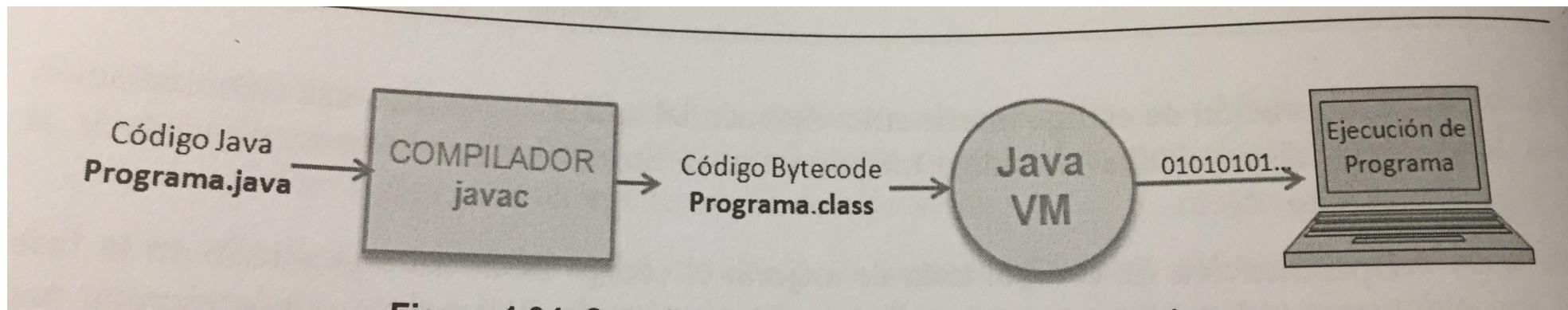


- 2) Máquinas virtuales de proceso (Process Virtual Machine). Una máquina virtual de proceso, a veces llamada “máquina virtual de aplicación” se ejecuta como un proceso normal dentro de un sistema operativo y soporta un solo proceso. Esta máquina se inicia automáticamente cuando se lanza el proceso que se desea ejecutar y se detiene cuando finaliza.
- Su objeto es proporcionar un entorno de ejecución independiente de la plataforma hardware y del sistema operativo, que oculte los detalles de la programación subyacente y permita que un programa se ejecute siempre de la misma forma sobre cualquier plataforma.
- El ejemplo más conocido actualmente es la máquina virtual de Java.



# 5.1 MÁQUINA VIRTUAL DE JAVA

- Los programas compilados en lenguaje Java se pueden ejecutar en cualquier plataforma: es decir, pueden ejecutarse en entornos UNIX, Mac, Windows, Solaris, etc.
- Esto es debido a que el código generado por el compilador no lo ejecuta el procesador del ordenador, sino que lo ejecuta la Máquina Virtual de Java (JVM, Java Virtual Machine).



- El proceso de compilación y ejecución de un programa Java contiene los pasos:
  - El código fuente del programa está escrito en ficheros de texto plano que tienen la extensión .java.
  - La compilación del fichero, mediante el compilador Java (javac), genera un fichero (o varios) con la extensión .class (siempre y cuando no haya errores).
  - Un fichero .class contiene código en un lenguaje intermedio cercano al lenguaje máquina pero independiente del ordenador y del sistema operativo en que se ejecuta, este código se llama bytecode.
  - La máquina virtual toma y traduce el bytecode en código binario para el procesador que se utiliza para ejecutar el programa.
  - Como está disponible (la máquina Virtual Java) en muchos sistemas operativos diferentes, los ficheros .class pueden ser ejecutados en distintas plataformas.
  - La principal desventaja de los lenguajes basados en máquina virtual es que son más lentos que los lenguajes completamente compilados



## 5.2 ENTORNOS DE EJECUCIÓN

- Un entorno de ejecución es un servicio de máquina virtual que sirve como base software para la ejecución de programas. En ocasiones pertenece al propio sistema operativo, pero también se puede instalar como software independiente que funcionará por debajo de la aplicación.
- Básicamente son un conjunto de utilidades que permiten la ejecución de programas.
- En Java el entorno de Ejecución es lo que se conoce como JRE (Java Runtime Environment o entorno en tiempo de ejecución de Java).
- El JRE se compone de un conjunto de utilidades que permitirá la ejecución de programas Java sobre cualquier tipo de plataforma.
- JRE está formado por:
  - Una máquina virtual Java.
  - Bibliotecas de clase estándar que implementan el API de JAVA.



# 6. HERRAMIENTAS UTILIZADAS EN LA PROGRAMACIÓN

- Para llevar a cabo la codificación y prueba de los programas se suelen utilizar entornos de programación (no confundir con entornos de ejecución). Estos entornos nos permiten realizar diferentes tareas:
  - Crear, editar y modificar el código fuente del programa.
  - Compilar, montar y ejecutar el programa.
  - Examinar el código fuente
  - Ejecutar el programa en modo depuración
  - Realizar pruebas del programa de forma automática
  - Generar documentación
  - Gestionar los cambios que se van haciendo en el programa (control de versiones).
  - Muchas más.
- A estos entornos de programación se les suele llamar entornos de desarrollo integrado o IDE (Integrated Development Environment).



- Los IDEs están diseñados para maximizar la productividad del programador.
- Un IDE es un programa informático formado por un conjunto de herramientas de programación que facilitan las tareas de creación, modificación, compilación, implementación y depuración del software.
- La mayoría de los IDEs actuales proporcionan un entorno de trabajo visual formado por ventanas, barras de menús, barras de herramientas, asistentes que ayudan al programador a desarrollar parte del proyecto o del código, etc.
- Un mismo IDE puede funcionar con varios lenguajes de programación, este es el caso de Eclipse o Netbeans que mediante la instalación de plugins (en muchas casos ya instalados por defecto) se le puede añadir soporte para lenguajes adicionales.



# 7. DESARROLLO EN TRES CAPAS

- El desarrollo en capas nace de la necesidad de separar la lógica de la aplicación del diseño, separando a su vez los datos de la presentación al usuario.
- Para solventar esa necesidad, se ideó el desarrollo en 3 capas, que separa la lógica del negocio, el acceso a datos y la presentación del usuario en tres capas que pueden tener cada una tantos niveles como se quiera o necesite.
- 1) Presentación
  - Comunica a la aplicación con el usuario.
  - Solo se comunica con la capa del negocio.
- 2) Negocio
  - Alberga los programas de la aplicación.
  - Se comunica con la capa de presentación y de datos.
- 3) Datos
  - Es el acceso a datos, tanto para solicitar como para persistir (guardado de los cambios).
  - Se comunica con la capa de negocio.



- El desarrollo por capas no solo nos mejora y facilita la estructura de nuestro propio software, sino que nos aporta la posibilidad de interoperar con otros sistemas ajenas a nuestra aplicación.
- Dentro del desarrollo por capas, encontramos diferentes modelos de software, el más utilizado es el MVC (Modelo Vista Controlador).
- El MVC define tres componentes para las capas del desarrollo del software.
  - 1) Vista.- Recibe los datos del modelo y los muestra al usuario.
  - 2) Controlador.- Actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.
  - 3) Modelo. Accede a la capa de almacenamiento de los datos.

