



# CFGs Desarrollo de Aplicaciones Multiplataforma Módulo

## Entornos de desarrollo

Título: RD 686/2010 de 20 mayo  
(BOE 12 de junio de 2010)

Currículo: D 230/2011 de 28 /08/2011  
(DOCM 9 de agosto de 2011)

## Unidades por evaluaciones y duración aproximada:

Ev.	Unidades	Semanas
1ª	1. Desarrollo de Sw	7
	2. Instalación y uso de entornos de desarrollo	3
2ª	3. Diseño y realización de pruebas	6
	4. Optimización y documentación	5
3ª	5. Diseño OO. Diagramas estructurales	6
	6. Diseño OO. Diagramas comportamiento	5

# I. Sw y programa informático.

---

## SISTEMA DE INFORMACIÓN:

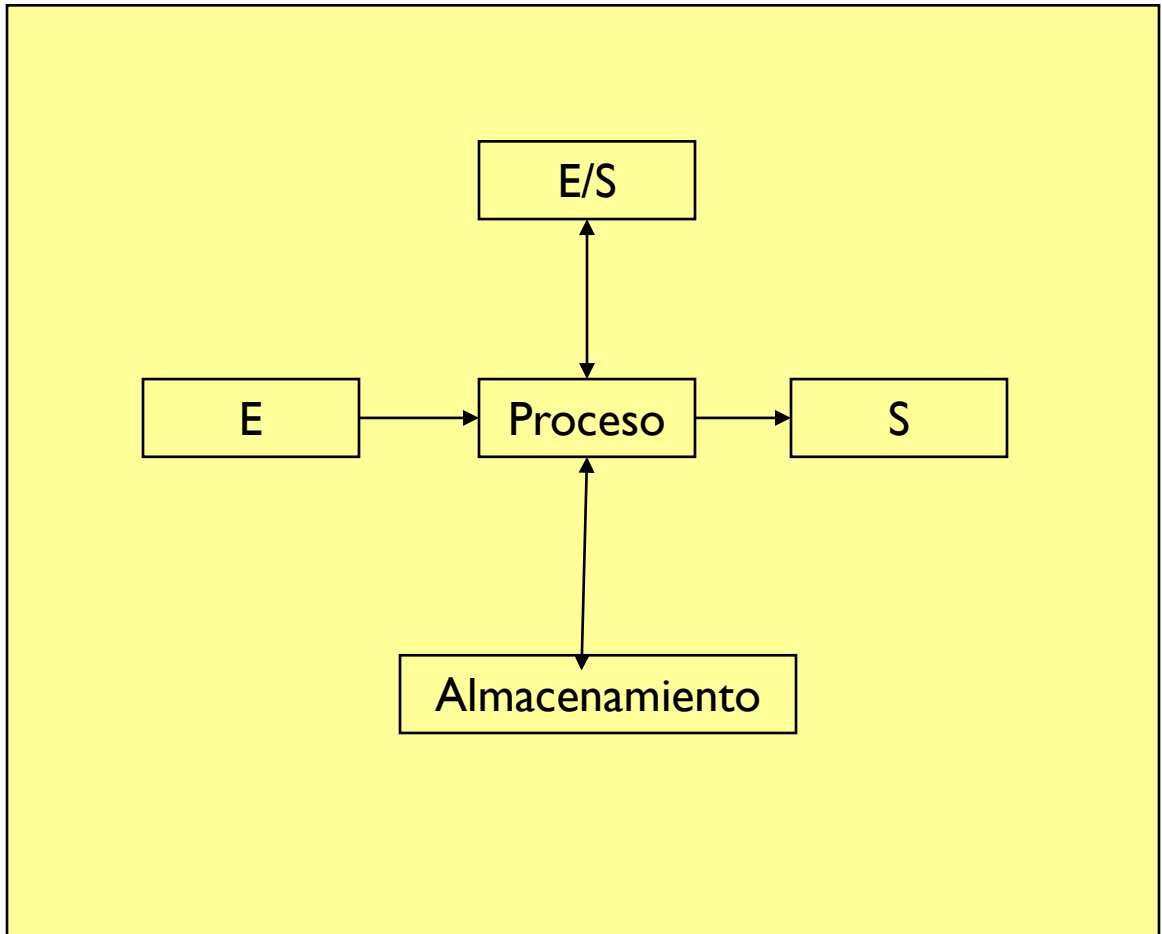
Un conjunto formal de **procesos** que, operando sobre una colección de **datos** estructurada según las necesidades de la empresa, **recopilan, elaboran, y distribuyen** la información (o parte de ella) necesaria para las operaciones de dicha empresa y para las actividades de dirección y control correspondientes (decisiones) para desempeñar su actividad de acuerdo a su estrategia de negocio

## OBJETIVO:

Ayudar al desempeño de las actividades en todos los niveles de la organización mediante el suministro de la información adecuada, con la calidad suficiente, a la persona apropiada, en el momento y lugar oportunos, y con el formato más útil para el receptor

# I. Sw y programa informático.

## SISTEMA DE INFORMACIÓN:



## SISTEMA INFORMÁTICO:

Sistema de información cuyos procesos se llevan a cabo con ordenadores

$$\text{S.I.} = \text{Sw} + \text{Hw} + \text{Usuarios}$$

# I. Sw y programa informático.

---

## **HW: Parte física**

- **CPU**
- **Memoria**
- **Periféricos**
- **Soportes almacenamiento**
- **Otros**

## **SW: Parte lógica**

- **Programas**
- **Datos**
- **Documentación**
- **Métodos de desarrollo de sw**

## **PROGRAMA:**

Conjunto de órdenes (instrucciones) que son ejecutadas por la máquina para resolver un problema determinado

## **APLICACIÓN INFORMÁTICA:**

Conjunto de programas para realizar un trabajo determinado

*(Programa y Aplicación se usan como sinónimos)*

# I. Sw y programa informático.

## Relación Hw-Sw

---

El software se ejecuta sobre el Hw

- ❑ Los elementos Hw necesitan estar instalados y configurados correctamente para que el equipo funcione
- ❑ Relación Hw-Sw desde dos puntos de vista:
  - del Sistema Operativo
  - de las aplicaciones.

# I. Sw y programa informático.

## Relación Hw-Sw

---

### **Desde el punto de vista del sistema operativo :**

- El sistema operativo es el encargado de coordinar al hardware durante el funcionamiento del ordenador, actuando como intermediario entre éste y las aplicaciones que están corriendo en un momento dado.
- Todas las aplicaciones necesitan recursos hardware durante su ejecución (tiempo de CPU, espacio en la RAM, gestión de los dispositivos de Entrada/Salida, ....etc.).
- Será siempre el sistema operativo el encargado de controlar todos estos aspectos de manera "oculta" para las aplicaciones (y para el usuario).

# I. Sw y programa informático.

## Relación Hw-Sw

---

### **Desde el punto de vista de las aplicaciones :**

- Los programas están escritos en algún lenguaje de programación que el hardware del equipo debe interpretar y ejecutar.
- Las sentencias de los lenguajes de programación son fáciles de aprender y usar por el ser humano.
- El hardware de un ordenador sólo es capaz de interpretar señales eléctricas (ausencias o presencias de tensión) que, en informática, se traducen en secuencias de 0 y 1 (código binario).
- Para que el ordenador ejecute las instrucciones escritas en un lenguaje de programación tendrá que pasar por un proceso de traducción de código.



# I. Sw y programa informático.

## Tipos de Sw

---

### Definiciones de SW:

#### ☐ **Según la RAE:**

- *“Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora”*

#### ☐ **El estándar IEEE:**

- *“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 Sw es todo aquello que se refiere a los programas y datos almacenados en un ordenador, programas encargados de dar instrucciones para realizar con el hw o para comunicarnos con otro sw, y datos necesarios para la ejecución de los programas

# I. Sw y programa informático.

## Tipos de Sw

---

Clasificación del SW, según:

### ❑ **Su función**

- De sistema
- De aplicación
- De desarrollo

### ❑ **El método de distribución:**

- Shareware (Por tiempo o limitado)
- Freeware → (Free → Gratis (No libre))
- Adware (Publicidad)

### ❑ **Tipo de licencia:**

- Libre → (en utilización y modificación → no tiene por qué ser gratuito)
- Propietario
- De dominio público (no tiene autor)
- De código abierto (similar a libre, el código estará disponible)

# I. Sw y programa informático.

## Tipos de Sw

---

Según su función, el Sw se clasifica en:

### □ **Sw de sistema (S.O. + drivers):**

- software base que ha de estar instalado y configurado en nuestro ordenador para que las aplicaciones puedan ejecutarse y funcionar.
- Ej: Windows, Linux, MAC.....

### ■ **Sw de aplicación:**

- conjunto de programas que realizan tareas específicas
- Ej: Procesador de textos, una hoja de cálculo, un videojuego, etc.

### ■ **Sw de desarrollo o programación:**

- conjunto de herramientas que nos permiten desarrollar programas informáticos
- Ej: Lenguajes de programación, compiladores, Entornos de desarrollo,...

# I. Sw y programa informático.

## Tipos de Sw

---

Ejercicio:

Completa la clasificación de los tipos de SW

Más información sobre los tipos de Sw:

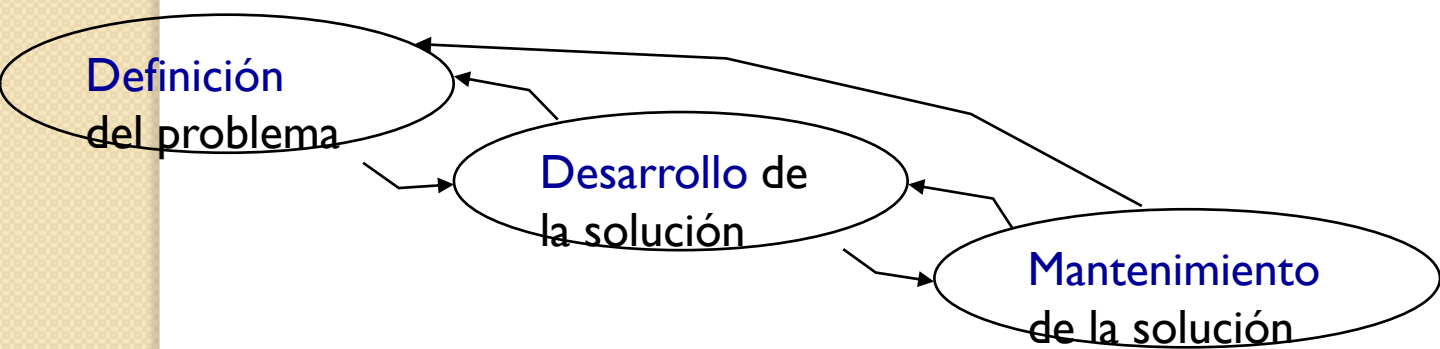
<http://www.tiposdesoftware.com/>

(o en otros lugares)

## 2. Desarrollo de Sw

Todo el proceso que ocurre desde que se concibe una idea hasta que un programa está implementado en el ordenador y funcionando.

- La construcción de software es un proceso que puede llegar a ser muy complejo y que exige gran coordinación y disciplina del grupo de trabajo que lo desarrolle.
- En todo proceso de desarrollo se ha de pasar una serie de etapas



## 2. Desarrollo de Sw

---

### Definición del problema

= el **QUÉ**

Se identifican los requisitos clave del sistema y del Sw :

- Qué información se va utilizar y cuáles son sus características,
- Qué restricciones, funcionalidad y rendimiento se desea,
- Se definen el sistema y el Sw a emplear, los recursos, tiempos y costes.

= Proyecto del Sistema.

Tareas principales:

- Planificación
- Análisis de requisitos

## 2. Desarrollo de Sw

---

### **Desarrollo de la solución propuesta:**

= el **CÓMO**

- El diseño de las estructuras de datos e interfaces
- Especificación detallada del diseño que permitirá obtener el código fuente (pseudocódigo).
- Cómo se va a traducir el diseño en un lenguaje de programación,
- Se diseña un plan de desarrollo y las pruebas necesarias para garantizar el funcionamiento correcto.

Al término de esta fase se entregará el sistema.

### **Tareas principales en la creación de software:**

- Análisis y Diseño del Sw.
- Codificación
- Prueba del Sw
- Documentación

## 2. Desarrollo de Sw

---

### Mantenimiento de la solución:

Se centra en el **CAMBIO:**

### **Tipos de cambios:**

- Corrección
- Adaptación
- Mejora o evolutivos
- Prevención (Ejemplo: efecto 2000)

Tareas:

- Explotación
- Mantenimiento



## 2. Desarrollo de Sw

---

Según estimaciones:

- el 26% de los grandes proyectos de software fracasan,
- el 48% deben modificarse drásticamente
- sólo el 26% tienen rotundo éxito.

La principal causa del fracaso de un proyecto es la falta de una buena planificación de las etapas y mala gestión de los pasos a seguir.

## 2. Ciclo de vida del Sw

---

### Ciclo de vida:

**“Conjunto de fases por las que pasa un sistema de información desde su concepción hasta que ya no se utiliza”**

“una aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del Sw”

*(IEEE 1064, 1991: estándar IEEE para el desarrollo de procesos de ciclo de vida Sw)*

“un marco de referencia que contiene los procesos, las actividades, y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto Sw, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso”

*(ISO 12207-1, 1994: procesos de ciclo de vida de Sw)*

## 2. Ciclo de vida del Sw

---

Los procesos generales (o fases) que se deben realizar en el ciclo de vida de un S.I. se pueden desarrollar de distintas formas (distintas estrategias), dando lugar a distintos **modelos de desarrollo** o **paradigmas del Sw**.

La elección de un paradigma u otro para llevar a cabo un proyecto Sw depende de la naturaleza del mismo y de la aplicación de los métodos y herramientas a usar así como de los controles y entregas requeridos.

## 2. Ciclo de vida del Sw

---

Existen varios paradigmas, los más conocidos son:

- ☐ Modelo en cascada (ciclo de vida clásico)
- ☐ Prototipado
- ☐ Evolutivos: Incremental y Espiral
- ☐ Desarrollo rápido de aplicaciones
- ☐ Técnicas de 4G
- ☐ Métodos formales (apoyados en lógica matemática)
- ☐ Modelo OO

*(existen variantes en las que intervienen fases de varios de los paradigmas anteriores)*

**Ejercicio: Buscar las características principales de los paradigmas anteriores**

(Muchas empresas tienen sus propios modelos basados en los anteriores: [Junta de Andalucía](#))

## 2. Herramientas de apoyo al desarrollo del Sw

---

### **Herramientas CASE:**

*(Computer Aided Software Engineering)*

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

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.

## 2 Herramientas de apoyo al desarrollo del Sw

---

**CLASIFICACIÓN** en función de las fases del ciclo de vida del software en la que ofrecen ayuda:

- ☐ **U-CASE** (*upper CASE o de alto nivel*):  
Primeras fases: planificación y análisis
- ☐ **M-CASE** (*de medio nivel*) : análisis y diseño
- ☐ **L-CASE** (*lower CASE o de bajo nivel*):  
Ultimas fases: programación del software, detección de errores del código, depuración de programas y pruebas y en la generación de la documentación del proyecto.
- ☐ **I-CASE** (*Integrated CASE*):  
Todas las fases

## 2 Herramientas de apoyo al desarrollo del Sw

---

### **CLASIFICACIÓN** según Fuggetta (1993):

#### ☐ **Herramientas:**

ayudan a un proceso concretos  
(compilación, comparación de la prueba,...)

#### ☐ **Bancos de trabajo (WorkBench):**

conjunto de herramientas que ayudan a  
fases o actividades del proceso de  
desarrollo (análisis, análisis y diseño)

#### ☐ **Entornos:**

Conjunto de bancos de trabajo integrados  
de alguna forma que ayudan a todo el  
proceso (o a una parte substancial del  
mismo).

### 3. Fases en el desarrollo y ejecución de Sw

---

Independientemente del modelo o paradigma elegido (ciclo de vida), siempre hay una serie de etapas que debemos seguir para construir software fiable y de calidad.

Estas etapas son:

- **ANÁLISIS DE REQUISITOS.**
- **DISEÑO.**
- **CODIFICACIÓN.**
- **PRUEBAS.**
- **DOCUMENTACIÓN.**
- **EXPLOTACIÓN.**
- **MANTENIMIENTO.**



### 3. Fases en el desarrollo y ejecución de Sw

---

#### **Definición del problema:**

#### **ANÁLISIS DE REQUISITOS.**

Se especifican los requisitos funcionales y no funcionales del sistema.

#### **Requisitos Funcionales:**

- Qué funciones tendrá que realizar la aplicación.
- Qué respuesta dará la aplicación ante todas las entradas.
- Cómo se comportará la aplicación en situaciones inesperadas.

#### **Requisitos No Funcionales:**

- Tiempos de respuesta del programa, legislación aplicable, tratamiento ante la simultaneidad de peticiones, etc.

### 3. Fases en el desarrollo y ejecución de Sw **Desarrollo de la solución:**

#### **DISEÑO.**

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

#### **CODIFICACIÓN.**

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

#### **PRUEBAS.**

Se prueban los programas para detectar errores y se depuran.

#### **DOCUMENTACIÓN.**

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

#### **EXPLOTACIÓN.**

Instalamos, configuramos y probamos la aplicación en los equipos del cliente (el cliente lo usa y se pueden detectar errores a solucionar).

### 3. Fases en el desarrollo y ejecución de Sw

---

#### **Mantenimiento de la solución:**

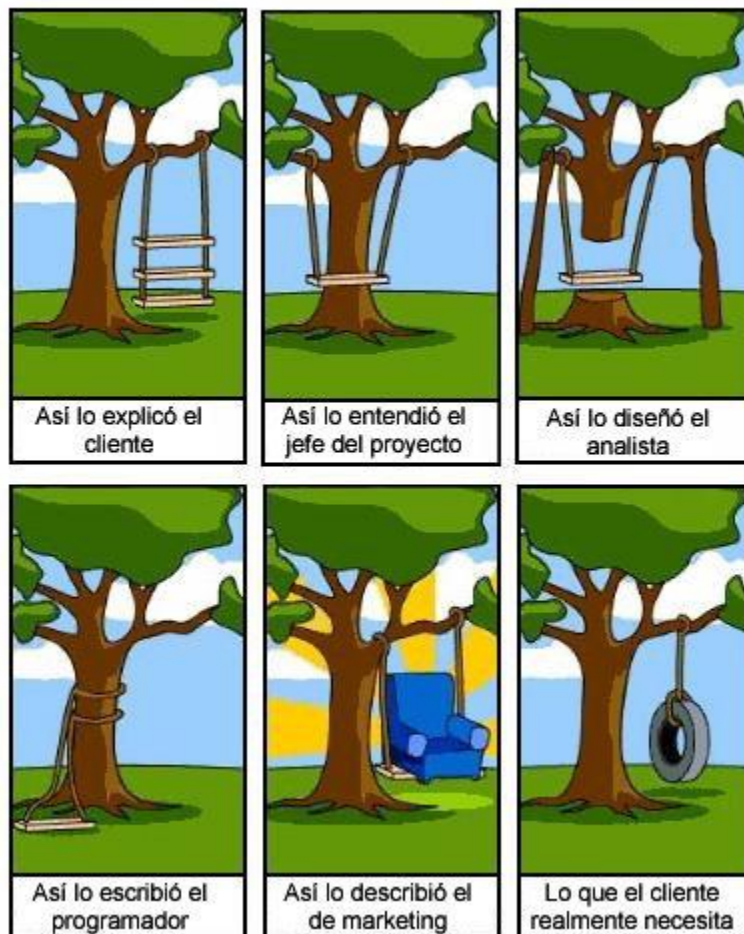
#### **MANTENIMIENTO.**

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

### 3. I Análisis

- la primera etapa del proyecto,
- la más complicada y
- la que más depende de la capacidad del analista.

- Es fundamental una buena comunicación entre el analista y el cliente para que la aplicación que se va a desarrollar cumpla con sus expectativas.
- La culminación de esta fase es el documento ERS (Especificación de Requisitos Software).



### 3. I Análisis

---

#### **ERS (Especificación de Requisitos Software) :**

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.

## 3.2 Diseño

---

Ya sabemos lo que hay que hacer, el siguiente paso es **¿Cómo hacerlo?**

- Se debe dividir el sistema en partes y establecer qué relaciones habrá entre ellas.
- Decidir qué hará exactamente 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.

En este punto, se deben tomar decisiones importantes, tales como:

- Entidades y relaciones de las bases de datos.
- Selección del lenguaje de programación que se va a utilizar.
- Selección del Sistema Gestor de Base de Datos.
- Etc.

### 3.3 Codificación

---

- **Tiene que cumplir exhaustivamente con todos los datos impuestos en el análisis y en el diseño de la aplicación.**
- El programador se encarga de codificar toda la información anterior en un lenguaje de programación.

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 ejecutar en cualquier equipo.

## 3.5 PRUEBAS

---

Son imprescindibles para asegurar la validación y verificación del software construido

Algunas de las pruebas del sw son:

### **PRUEBAS UNITARIAS :**

- Se prueban, una a una, las diferentes partes del software y se comprueba su funcionamiento (por separado, de manera independiente).
- JUnit es el entorno de pruebas para Java.

### **PRUEBAS DE INTEGRACIÓN:**

- Consistirán en comprobar el funcionamiento del sistema completo: con todas sus partes interrelacionadas.



## 3.5 PRUEBAS

---

### **Beta Test:**

Prueba final que se realiza sobre el entorno de producción donde el sw va a ser utilizado por el cliente (a ser posible, en los equipos del cliente y bajo un funcionamiento normal de su empresa).

El período de prueba será normalmente el pactado con el cliente.

## 3.6 DOCUMENTACION

Todas las etapas en el desarrollo de software deben quedar perfectamente documentadas.

### ➤ **¿Por qué?**

Para dar toda la información a los usuarios de nuestro software y poder acometer futuras revisiones del proyecto.

- Tenemos que ir documentando el proyecto en todas las fases del mismo, para pasar de una a otra de forma clara y definida.
- Una correcta documentación permitirá la reutilización de parte de los programas en otras aplicaciones, siempre y cuando se desarrollen con diseño modular.

## 3.6 DOCUMENTACION

---

Los documentos a elaborar en el proceso de desarrollo de software se pueden agrupar en:

- ☐ Guía técnica
- ☐ Guía de uso
- ☐ Guía de instalación

## 3.6 DOCUMENTACION

---

### **GUÍA TÉCNICA:**

#### ☐ Documentos:

- El diseño de la aplicación
- La codificación de los programas
- Las pruebas realizadas

#### ☐ Objetivo:

Facilitar un correcto desarrollo, realizar correcciones en los programas y permitir un mantenimiento futuro

#### ☐ Dirigido a:

Personal técnico en informática (analistas y programadores)

## 3.6 DOCUMENTACION

---

### **GUÍA de USO:**

#### ☐ Documentos:

- Descripción de la funcionalidad de la aplicación.
- Forma de comenzar su ejecución
- Ejemplos de uso del programa.
- Requerimientos software de la aplicación.
- Solución de los posibles problemas que se pueden presentar.

#### ☐ Objetivo:

Dar a los usuarios finales toda la información necesaria para utilizar la aplicación

#### ☐ Dirigido a:

Los usuarios que van a usar la aplicación (clientes).

## 3.6 DOCUMENTACION

---

### **GUÍA de INSTALACIÓN:**

#### ☐ Documentos:

Toda la información necesaria para:

- Puesta en marcha.
- Explotación.
- Seguridad del sistema.

#### ☐ Objetivo:

Dar toda la información necesaria para garantizar que la implantación de la aplicación se realice de forma segura, confiable y precisa.

#### ☐ Dirigido a:

Personal informático responsable de la instalación, en colaboración con los usuarios que van a usar la aplicación (clientes).

## 3.7 EXPLOTACIÓN

La explotación es la instalación, puesta a punto y funcionamiento de la aplicación en el equipo final del cliente

- Una vez que las pruebas nos demuestran que el software es fiable, carece de errores y hemos documentado todas las fases, el siguiente paso es la **explotación**, fase en la que los usuarios finales conocen la aplicación y comienzan a utilizarla.

### **1º Instalación:**

- Proceso en el que los programas son transferidos al computador del usuario cliente y posteriormente configurados y verificados.
- Es recomendable que los futuros clientes estén presentes en este momento e irles comentando cómo se va planteando la instalación.

## 3.7 EXPLOTACIÓN

---

### **2º Configuración:**

- Se asignan los parámetros de funcionamiento normal de la empresa y se prueba que la aplicación es operativa.
- También puede ocurrir que la configuración la realicen los propios usuarios finales, siempre y cuando les hayamos dado previamente la guía de instalación.
- Y también, si la aplicación es más sencilla, podemos programar la configuración de manera que se realice automáticamente tras instalarla. (Si el software es "a medida", lo más aconsejable es que la hagan aquellos que la han fabricado).



## 3.7 EXPLOTACIÓN

---

### 3° Producción normal.

- La aplicación pasa a manos de los usuarios finales y se da comienzo a la explotación del software.
- Es muy importante tenerlo todo preparado antes de presentarle el producto al cliente:  
**¡será el momento crítico del proyecto.!**

## 3.8 MANTENIMIENTO

---

- En cualquier sector laboral, con la entrega del producto, ha terminado su trabajo
- PERO en el caso de la construcción de software, esto NO es así con la entrega de nuestra aplicación (la instalación y configuración de nuestro proyecto en los equipos del cliente).
- ADEMÁS, la etapa de mantenimiento es la más larga de todo el ciclo de vida del sw
- el sw deberá actualizarse y evolucionar con el tiempo, deberá ir adaptándose a las mejoras del hw y afrontar situaciones nuevas que no existían cuando el software se construyó.
- Siempre surgen errores (BUGS) que habrá que ir corrigiendo y nuevas versiones del producto mejores que las anteriores
- Pactar con el cliente un servicio de mantenimiento de la aplicación (que también tendrá un coste temporal y económico).

## 3.8 MANTENIMIENTO

---

Tipos de cambios:

- ❑ **Correctivos:** La aplicación tendrá errores en el futuro (sería utópico pensar lo contrario).
- ❑ **Adaptativos:** Modificaciones, actualizaciones... para adaptarse a las nuevas tendencias del mercado, a nuevos componentes hardware, etc.
- ❑ **Mejora o Evolutivos:** El cliente tendrá en el futuro nuevas necesidades. Por tanto, serán necesarias modificaciones, expansiones o eliminaciones de código.
- ❑ **Preventivos:** cambios para facilitar las futuras operaciones de mantenimiento

## 4. Lenguajes de programación

---

### Lenguaje de Programación:

idioma creado de forma artificial, formado por un conjunto de símbolos y normas que se aplican sobre un alfabeto para obtener un código, que el hardware de la computadora pueda entender y ejecutar.

### Es un conjunto de:

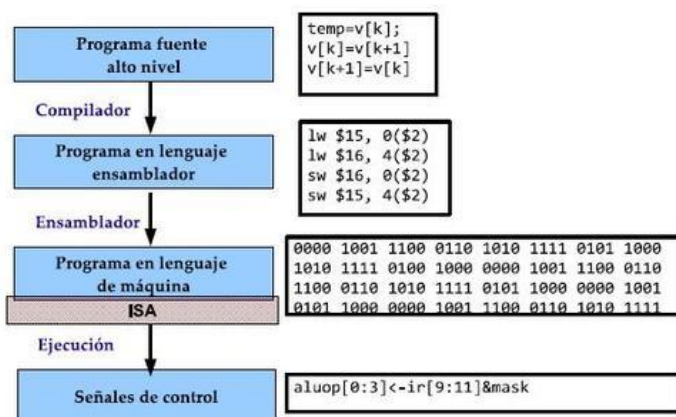
- ❑ **Alfabeto:** conjunto de símbolos permitidos
- ❑ **Sintaxis:** Normas de construcción permitidas de los símbolos del lenguaje
- ❑ **Semántica:** significado de las construcciones para hacer acciones válidas

## 4. Lenguajes de programación

### Clasificación:

- Según lo cerca que está del lenguaje humano o de la máquina:

- ☐ L. máquina
- ☐ L. ensamblador
- ☐ L. de alto nivel

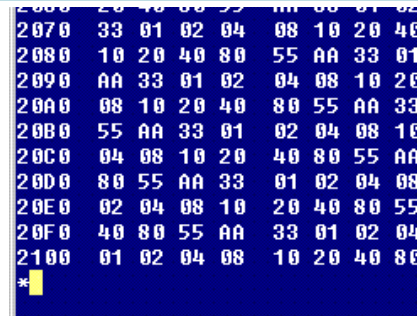


- Según la técnica de programación utilizada:

- ☐ L. Estructurados
- ☐ L.OO
- ☐ L.Visuales

## 4. Lenguajes de programación

### Lenguaje máquina:



2070	33	01	02	04	08	10	20	40
2080	10	20	40	80	55	AA	33	01
2090	AA	33	01	02	04	08	10	20
20A0	08	10	20	40	80	55	AA	33
20B0	55	AA	33	01	02	04	08	10
20C0	04	08	10	20	40	80	55	AA
20D0	80	55	AA	33	01	02	04	08
20E0	02	04	08	10	20	40	80	55
20F0	40	80	55	AA	33	01	02	04
2100	01	02	04	08	10	20	40	80
*								

- Sus instrucciones son combinaciones de unos y ceros.
- Es el único lenguaje que entiende directamente el ordenador. (No necesita traducción).
- Fue el primer lenguaje utilizado.
- Es único para cada procesador (no es portable generalmente de un equipo a otro (si tienen distintos juegos de instrucciones)).
- Hoy día se programa muy poco en este lenguaje.

## 4. Lenguajes de programación

```
main:  .CODE
        MOV AX,DGROUP
        MOV DS,AX
        MOV [VarB],42
        MOV [VarD],-7
        MOV BX,Offset[S]
        MOV AX,[VarW]
```

### Lenguaje ensamblador:

- Sustituyó al lenguaje máquina para facilitar la labor de programación.
- En lugar de unos y ceros se programa usando mnemotécnicos (instrucciones complejas) (una instrucción equivale a otra en lenguaje máquina).
- Necesita traducción al lenguaje máquina para poder ejecutarse.
- Muchas de sus instrucciones son sentencias que hacen referencia a la ubicación física de los registros en el procesador.
- Es difícil de utilizar.

## 4. Lenguajes de programación

### Lenguaje de alto nivel:

ejemplo C: Hola Mundo!

```
#include <stdio.h>

int main()
{
    printf("Hola Mundo!\n");
    return 0;
}
```

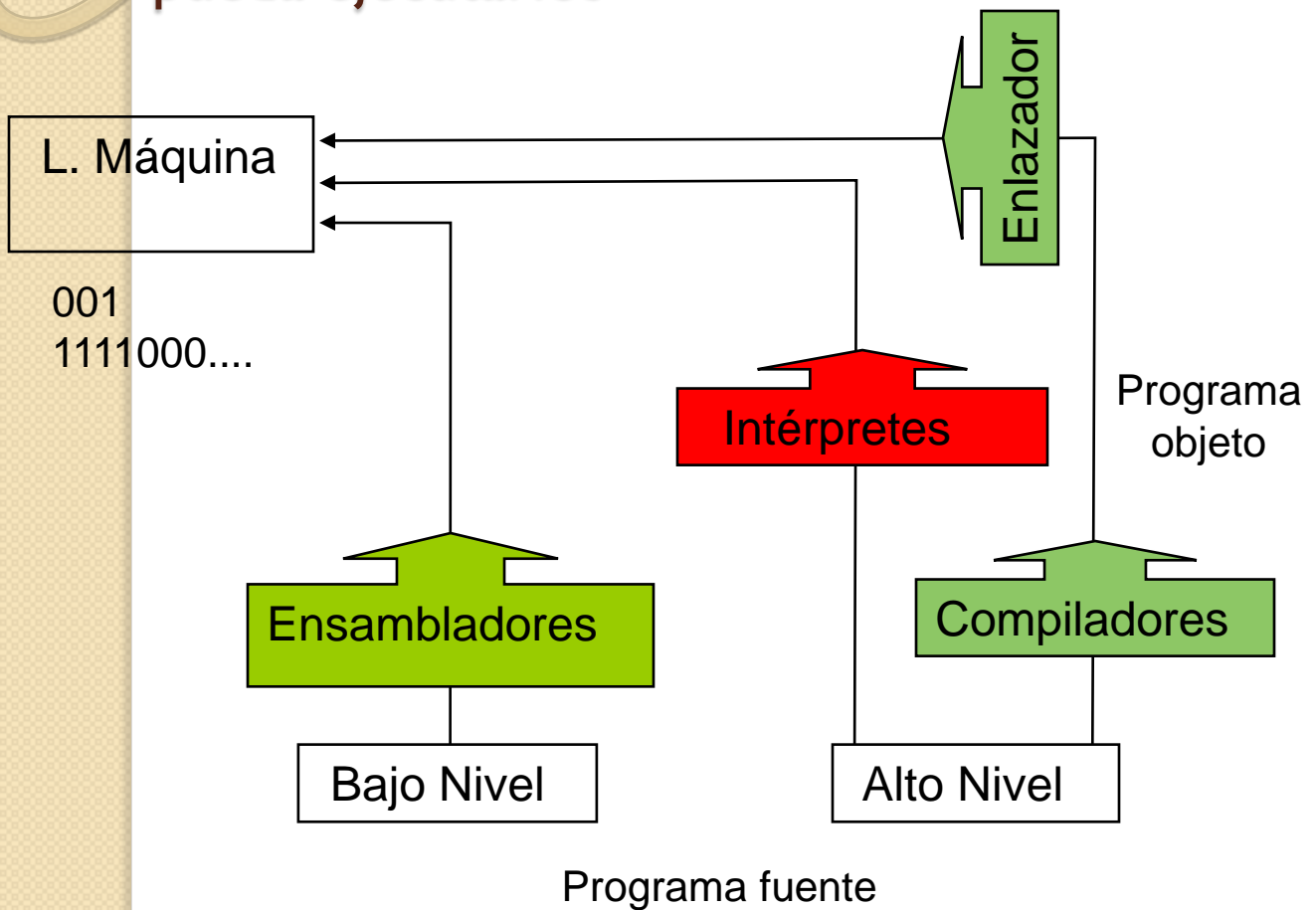
- Sustituyeron al lenguaje ensamblador para facilitar más la labor de programación.
- En lugar de mnemotécnicos, se utilizan sentencias y órdenes derivadas del idioma inglés. (Necesita traducción al lenguaje máquina).
- Son más cercanos al razonamiento humano.
- Son utilizados hoy día.



## 4. Lenguajes de programación

### TRADUCTORES

Programas que traducen programas al lenguaje máquina para que un ordenador pueda ejecutarlos



El Compilador+Enlazador o el Ensamblador: obtienen un programa ejecutable independiente que ya podremos ejecutar en cualquier momento.

El Intérprete: cada vez que se ejecute el programa tiene que traducir las instrucciones de Alto Nivel a Lenguaje Máquina (implica mayor lentitud de ejecución).

## 4. Lenguajes de programación

---

### **Lenguajes de programación estructurada:**

- Los programas se tratan como un conjunto ordenado de sentencias.
- Sentencias básicas:
  - ☐ Secuencia
  - ☐ Alternativa
  - ☐ Repetitiva

### **VENTAJAS:**

- Los programas son fáciles de leer.
- El mantenimiento de los programas es sencillo.
- La estructura del programa es sencilla y clara.

(la programación antigua que no era estructurada se denominaba de forma peyorativa “programación *espagueti*”)

## 4. Lenguajes de programación

---

### Lenguajes de programación estructurada:

#### **INCONVENIENTES :**

- Todo el programa se concentra en un único bloque (si se hace demasiado grande es difícil manejarlo).
  - No permite reutilización eficaz de código, ya que todo va "en uno".
  - No es útil para programas muy largos
- Por esto a la programación estructurada le sustituyó la programación **modular**, donde los programas se codifican por módulos y bloques, permitiendo mayor funcionalidad.
- Ejemplos de lenguajes estructurados:  
Pascal, C, Fortran.

## 4. Lenguajes de programación

---

### **Lenguajes de programación OO:**

- Los programas se componen de objetos independientes que colaboran entre ellos para realizar acciones
- Los objetos son reutilizables para proyectos futuros.

### **Características:**

- Los objetos del programa tendrán una serie de atributos que diferencian unos de otros y que indicarán el estado de los objetos.
- Se define clase como una colección de objetos con características similares.
- Mediante los llamados métodos, los objetos se comunican con otros produciéndose un cambio de estado de los mismos.
- Los objetos son, pues, como unidades individuales e indivisibles que forman la base de este tipo de programación.

## 4. Lenguajes de programación

---

### **Lenguajes de programación OO:**

#### **VENTAJAS:**

- El código es reutilizable.
- Si hay algún error, es más fácil de localizar y depurar en un objeto que en un programa entero.

#### **INCONVENIENTES:**

- Programación menos intuitiva que la estructurada

Ejemplos de lenguajes orientados a objetos:

Java, Ada, C++, VB.NET, Delphi, PowerBuilder,

## 4. Lenguajes de programación

---

### **Lenguajes visuales:**

- Están sustituyendo a los lenguajes de alto nivel basados en código.
- En lugar de sentencias escritas, se programa gráficamente usando el ratón y diseñando directamente la apariencia del software.
- Su correspondiente código se genera automáticamente.
- Necesitan traducción al lenguaje máquina.
- Son completamente portables de un equipo a otro.

## 4. Lenguajes de programación

---

Para saber más:

Realiza un pequeño trabajo con los tipos de lenguajes de programación que encontrarás en:

<http://www.monografias.com/trabajos38/tipos-lenguajes-programacion/tipos-lenguajes-programacion.shtml>

## 5. Tipos de código

Según el lenguaje en el que está escrito un programa, se le llama:

**Código Fuente:** escrito en algún lenguaje de programación de alto nivel.

**Código Objeto:** código binario resultado de compilar el código fuente.

El código objeto no es directamente inteligible por el ser humano ni por la computadora (al código fuente se le ha pasado un compilador).

**Código Ejecutable (máquina):** código binario resultante de enlazar los archivos de código objeto con ciertas rutinas y bibliotecas necesarias . Es directamente inteligible por la computadora (al código objeto se le ha pasado un linker (enlazador), o bien al código ensamblador se le ha pasado un ensamblador) (también es el resultado de la traducción de una instrucción de alto nivel por parte de un intérprete) .



## 5. Tipos de código

---

### **CODIGO FUENTE**

- **Conjunto** de instrucciones que el ordenador deberá realizar, escritas en un lenguaje de alto nivel.
- Para obtener el código fuente de una aplicación informática:
  - Se debe partir de las etapas anteriores de análisis y diseño
  - Se diseñará un algoritmo que simbolice los pasos a seguir para la resolución del problema (generalmente en pseudocódigo)
  - Se elegirá un Lenguaje de Programación de alto nivel apropiado para las características del software que se quiere codificar.
  - Se procederá a la codificación del algoritmo antes diseñado.

## 5. Tipos de código

---

### **CODIGO FUENTE**

La culminación de la obtención de código fuente es un documento con la codificación de todos los módulos, funciones, bibliotecas y procedimientos necesarios para codificar la aplicación.

Tipos de código fuente según licencias:

#### **Código fuente abierto.**

Es aquél que está disponible para que cualquier usuario pueda estudiarlo, modificarlo o reutilizarlo.

#### **Código fuente cerrado.**

Es aquél que no tenemos permiso para editarlo.

## 5. Tipos de código

### **CODIGO OBJETO**

Es el resultado de la compilación del código fuente (ejecución del compilador).

Es un programa en un código equivalente formado por unos y ceros (bytecode) que aún no puede ser ejecutado directamente por la computadora.

Sólo se genera código objeto una vez que el código fuente está libre de errores léxicos (ejem: algún carácter no válido; la ñ no es reconocida en muchos lenguajes de programación), sintácticos (ejem: falta el signo de = en una asignación) y semánticos (ejem: una variable no declarada, en C java utilizar = en lugar de == para una comparación, incompatibilidad entre asignación de variables de distintos tipos), un error semántico se produce cuando la sintaxis del código es correcta, pero la semántica o significado no es el que se pretendía.

*(aunque dependiendo del compilador, es posible que algún error semántico no sea detectado, por ejemplo en la suma de dos variables «incompatibles», se debería detectar posteriormente junto con los errores lógicos, pero ya en tiempo de ejecución)*

## 5. Tipos de código

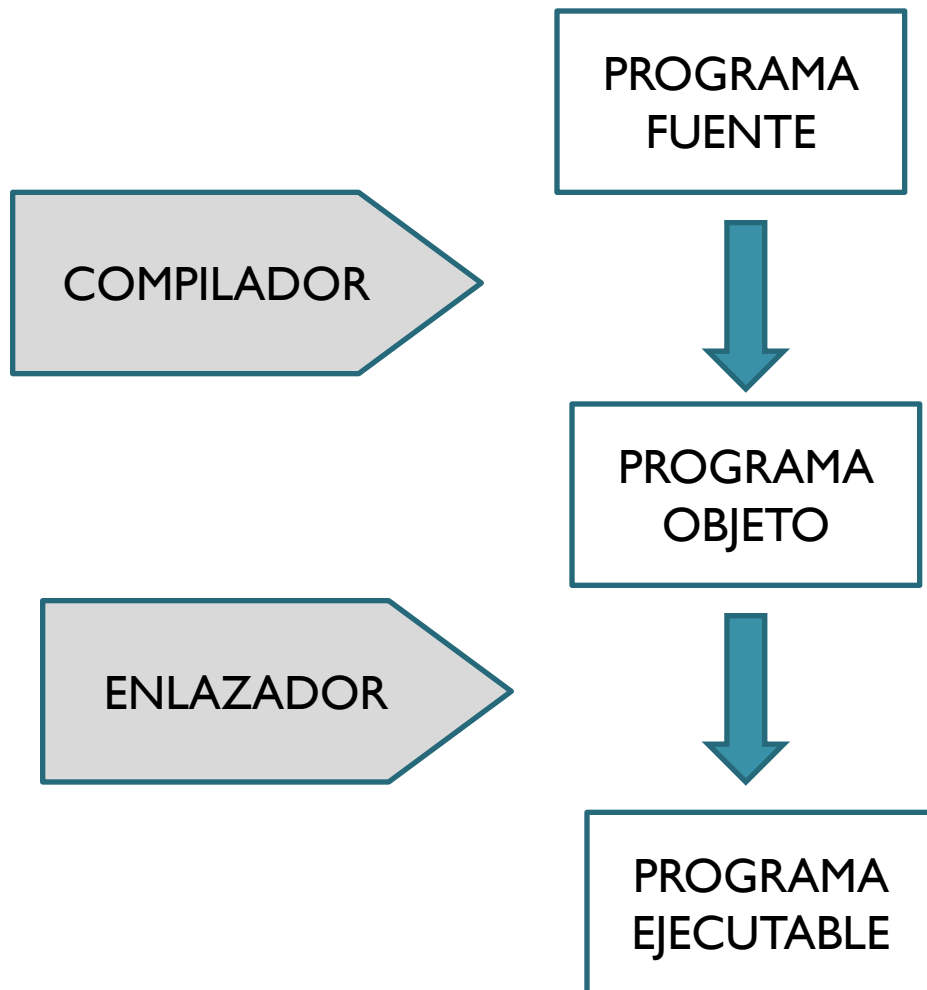
---

### **CODIGO EJECUTABLE**

- Resultado de enlazar (linker) los archivos de código objeto, (programa y librerías) consta de un único archivo que puede ser directamente ejecutado por la computadora (ya es código máquina).
- No necesita ninguna aplicación externa. Este archivo es ejecutado y controlado por el sistema operativo.

## 5. Tipos de código

### Proceso de obtención de código ejecutable:

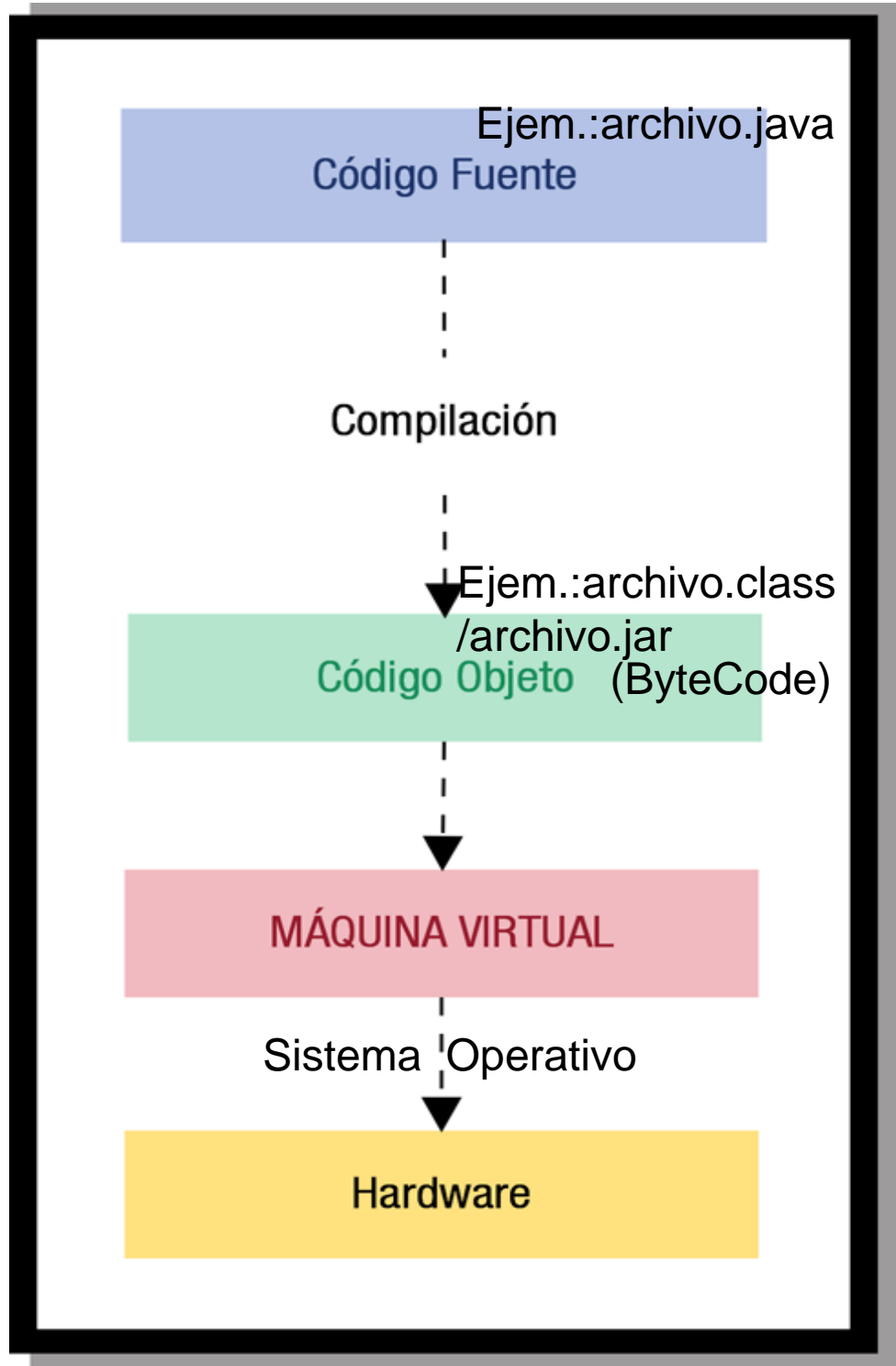


## 5. Máquinas virtuales

*(No confundir MV desde el punto de vista de programación con un Ordenador Virtual al que comúnmente llamamos MV)*

- Una Máquina Virtual (MV) de proceso es un tipo especial de software cuya misión es separar el funcionamiento del ordenador de los componentes hardware instalados. Ejemplo: la máquina virtual java.
- Actúa de puente entre la aplicación y Sistema Operativo concreto del equipo donde se instale.
- Con el uso de máquinas virtuales podremos desarrollar y ejecutar una aplicación sobre cualquier equipo, independientemente de las características concretas de la plataforma de los componentes físicos instalados (sólo habrá que tener instalada la MV para esa plataforma).
- Esto garantiza la **portabilidad** de las aplicaciones («Write once, run anywhere»).

## 5.4 Máquinas virtuales



## 5. Máquinas virtuales

---

Funciones principales de una máquina virtual:

- ☐ Conseguir que las aplicaciones sean portables.
- ☐ Reservar memoria para los objetos que se crean y liberar la memoria no utilizada.
- ☐ Comunicarse con el sistema donde se instala la aplicación (huésped), para el control de los dispositivos hardware implicados en los procesos.
- ☐ Cumplimiento de las normas de seguridad de las aplicaciones.



## 5. Máquinas virtuales

---

Características de la máquina virtual:

- ☐ Cuando el código fuente se compila se obtiene código objeto (bytecode/código intermedio).
- ☐ Para ejecutarlo en cualquier máquina se requiere tener independencia respecto al hardware concreto que se vaya a utilizar.
- ☐ Para ello, la máquina virtual aísla la aplicación de los detalles físicos del equipo en cuestión.
- ☐ Funciona como una capa de software de bajo nivel y actúa como puente entre el bytecode de la aplicación y los dispositivos físicos del sistema (realmente el SO está también en medio).
- ☐ La Máquina Virtual verifica todo el bytecode antes de ejecutarlo.
- ☐ La Máquina Virtual puede proteger direcciones de memoria para evitar “código maligno.”

## 5. ENTORNO DE EJECUCION

---

Conjunto de utilidades que permiten la ejecución de programas.

Es un servicio de máquina virtual que sirve como base software para la ejecución de programas.

Está formado por:

- máquina virtual, y
- API's (bibliotecas de clases estándar necesarias para que pueda ser ejecutada)

Puede:

- pertenecer al propio sistema operativo, o
- instalarse como software independiente

**Runtime:** tiempo que tarda un programa en ejecutarse en la computadora / tiempo de ejecución de un programa.

## 5. ENTORNO DE EJECUCION

---

Durante la ejecución, los entornos se encargarán de:

- ❑ **Configurar la memoria** principal disponible en el sistema.
- ❑ **Enlazar** los archivos del programa con las bibliotecas existentes y con los subprogramas creados. Considerando que las bibliotecas son el conjunto de subprogramas que sirven para desarrollar o comunicar componentes software pero que ya existen previamente y los subprogramas serán aquellos que hemos creado a propósito para el programa.
- ❑ **Depurar** los programas: comprobar la existencia (o no existencia) de errores semánticos no detectados en compilación, y lógicos (los léxicos y sintácticos ya se detectaron en la compilación). Normalmente se utiliza un debugger que nos proporcionan los entornos de programación.

## 5. JAVA RUNTIME ENVIRONMENT

---

Funcionamiento, instalación, configuración y primeros pasos del Runtime Environment del lenguaje Java (extensible a los demás lenguajes de programación).

Empresa creadora: Sun Microsystem y adquirido por Oracle

### **Concepto.**

- JRE = Java Runtime Environment (entorno en tiempo de ejecución Java).
- El JRE se compone de un conjunto de utilidades que permitirá la ejecución de programas java sobre cualquier tipo de plataforma para la que haya sido implementado el JRE.

*(si lo que se desea es desarrollar aplicaciones Java, entonces se debe descargar JDK (Java Development Kit que incluye JRE+API+ compilador Java))*

## 5. JAVA RUNTIME ENVIRONMENT

---

**Componentes:** JRE está formado por:

- Una Máquina virtual Java (JMV o JVM si consideramos las siglas en inglés), que es el programa que interpreta el código de la aplicación escrito en Java.
- Bibliotecas de clase estándar que implementan el API de Java (Interfaz de Programación de Aplicaciones: las clases facilitadas por Java para poder ejecutar programas en el lenguaje Java).

Las dos: JMV y API de Java son consistentes entre sí, por ello son distribuidas conjuntamente.

**Instalación:** Java es software libre, Descargar el programa JRE de la página de Java. ([www.java.com](http://www.java.com) o bien de [www.oracle.com](http://www.oracle.com)) (Java2 Runtime Environment JRE 1.6.0.21). Instalar siguiendo los pasos del asistente.

## 5. FRAMEWORKS

---

- Son Plataformas Software de ayuda al programador para desarrollar proyectos sin partir de cero.
- Se tiene definidos programas de soporte, bibliotecas, etc, que ayudan a desarrollar y unir los diferentes módulos o partes de un proyecto

Ejemplos de Frameworks:

- **.NET:**
  - para aplicaciones sobre windows
  - “Visual Studio .net” nos da facilidades para construir aplicaciones en distintos lenguajes (J#, C#, C++, Visual Basic .net).
  - Su motor es el “.Net framework” que permite ejecutar dichas aplicaciones.
  - Es un componente que se instala sobre el sistema operativo (máquina virtual).
- **Spring de Java** (→ misma idea con Java)

## 5. FRAMEWORKS

---

### **VENTAJAS** de utilizar un framework:

- **Desarrollo rápido** de software.
- **Reutilización** de partes de código para otras aplicaciones.
- **Diseño** uniforme del software.
- **Portabilidad** de aplicaciones de un computador a otro, ya que los bytecodes que se generan a partir del lenguaje fuente podrán ser ejecutados sobre cualquier máquina virtual.

### **INCONVENIENTES:**

- Gran dependencia del código respecto al framework utilizado (si cambiamos de framework, habrá que reescribir gran parte de la aplicación).
- La instalación e implementación del framework en nuestro equipo consume bastantes recursos del sistema.