



Unión Europea
Fondo Social Europeo
“El FSE invierte en tu futuro”



Comunidad
de Madrid

UT1 - DESARROLLO DE SOFTWARE

Entornos de Desarrollo

Ciclo Formativo de grado superior en Desarrollo de Aplicaciones
DAM/DAW

ÍNDICE

- 1. Concepto de programa informático. Instrucciones y datos.
- 2. Ejecución de programas en ordenadores.
- 3. Datos y programas.
- 4. Hardware vs. Software.
- 5. Estructura funcional de un ordenador: procesador, memoria.
- 6. Tipos de software. BIOS. Sistema. Aplicaciones.
- 7. Código fuente, código objeto y código ejecutable; máquinas virtuales.

ÍNDICE

- 8. Lenguajes de programación:
 - 8.1 Tipos de lenguajes de programación.
 - 8.2 Características de los lenguajes más difundidos.
- 9. Introducción a la Ingeniería del Software:
 - 9.1 Proceso software y ciclo de vida del software.
 - 9.2 Fases del desarrollo de una aplicación: análisis, diseño, codificación, pruebas, documentación, explotación y mantenimiento.

ÍNDICE

- 9.3 Modelos de proceso de desarrollo software (cascada, iterativo, evolutivo).
- 9.4 Metodologías de desarrollo software. Características. Técnicas. Objetivos. Tipos de metodologías.
- 9.5 Herramientas CASE (Computer Aided Software Engineering).
- 10. Proceso de obtención de código ejecutable a partir del código fuente, herramientas implicadas (editores, compiladores, enlazadores, etc.).
- 11. Errores en el desarrollo de programas.
- 12. Importancia de la reutilización de código.

1. Concepto de programa informático. Instrucciones y datos.

- Ordenador = Autómata programable.
- Dato = información introducida por el usuario.
- Instrucciones = Operadores + Variables + Estructuras de Control que operan con los datos.

Ejemplo programa en C

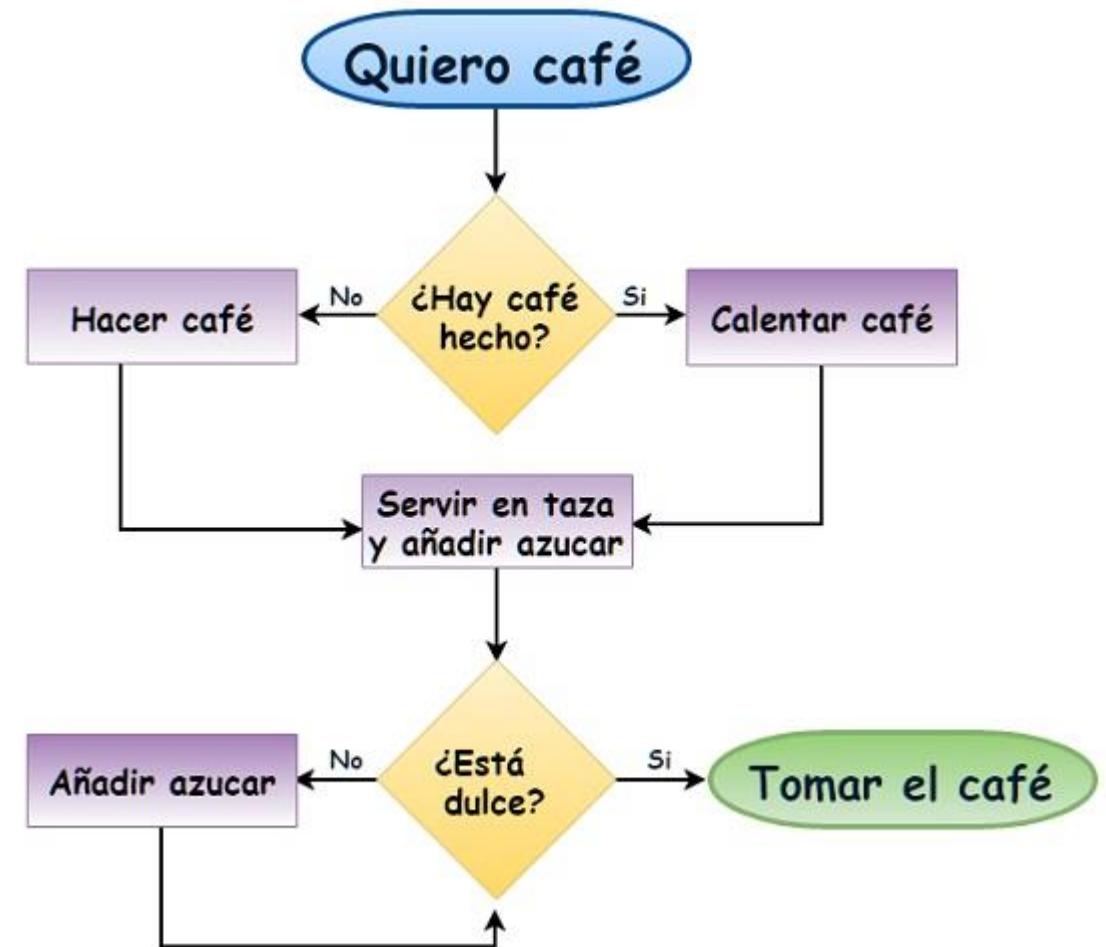
El texto anterior representa un ejemplo de lo que sería un programa. Se compone de 6 líneas, que van a comentarse a continuación:

```
/* Programa holamundo.c */  
Esta línea no realiza ninguna función solo dice cuál es el nombre del pro-  
grama.  
#include <stdio.h>  
Esta línea es necesaria si va a sacarse algo por pantalla.  
main ()  
Esta línea indica que esto es lo primero que va ejecutar el programa (lo  
contenido entre { y }).{  
    printf ("Hola Mundo"); .  
    Esta línea muestra las palabras Hola mundo por pantalla.  
}
```

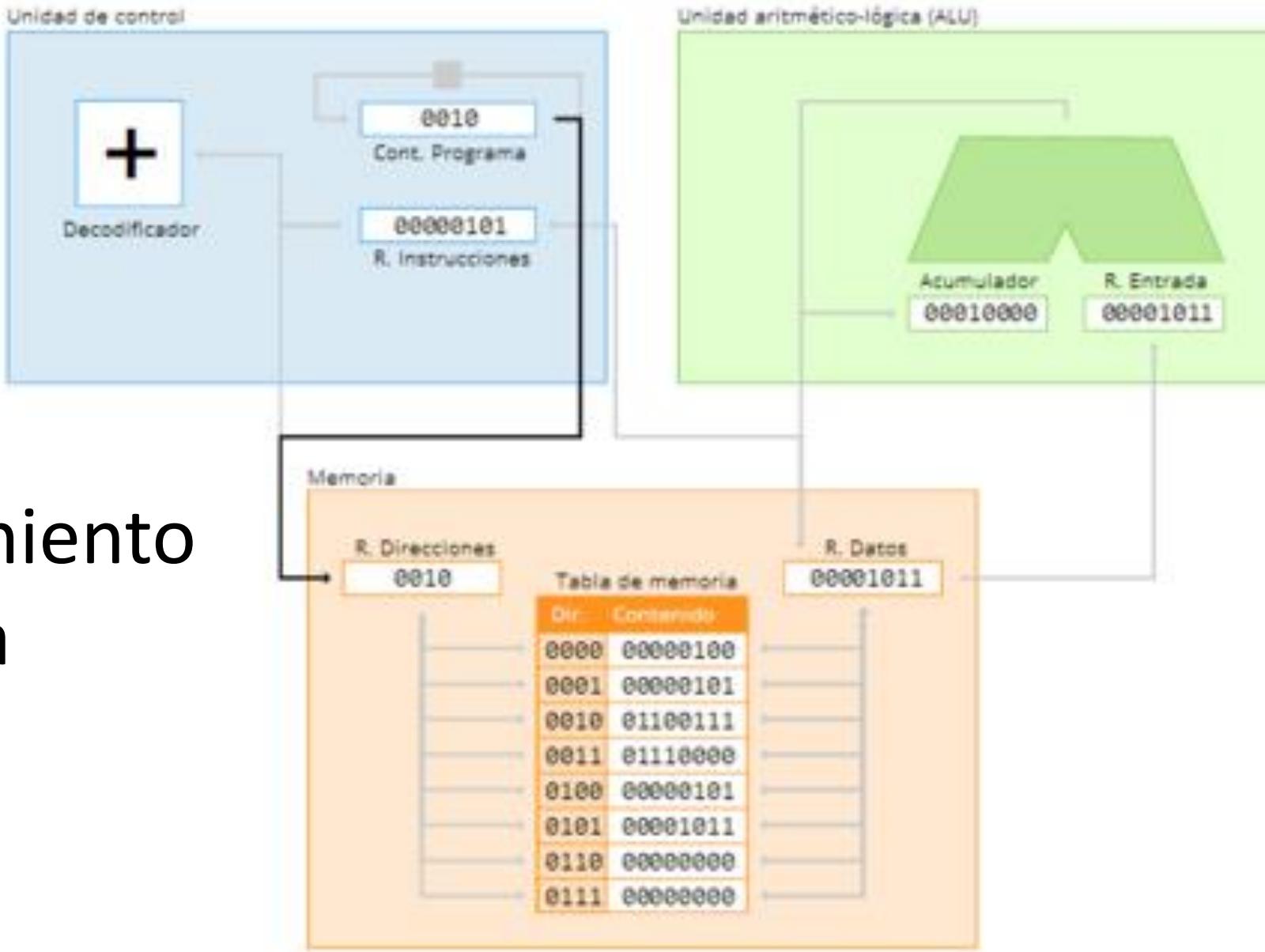
En resumen, este programa mostrará las palabras *Hola mundo* por pantalla.

2. Ejecución de programas en ordenadores.

- Algoritmo.
- Instrucciones.
- Recibe datos. Emite un resultado.



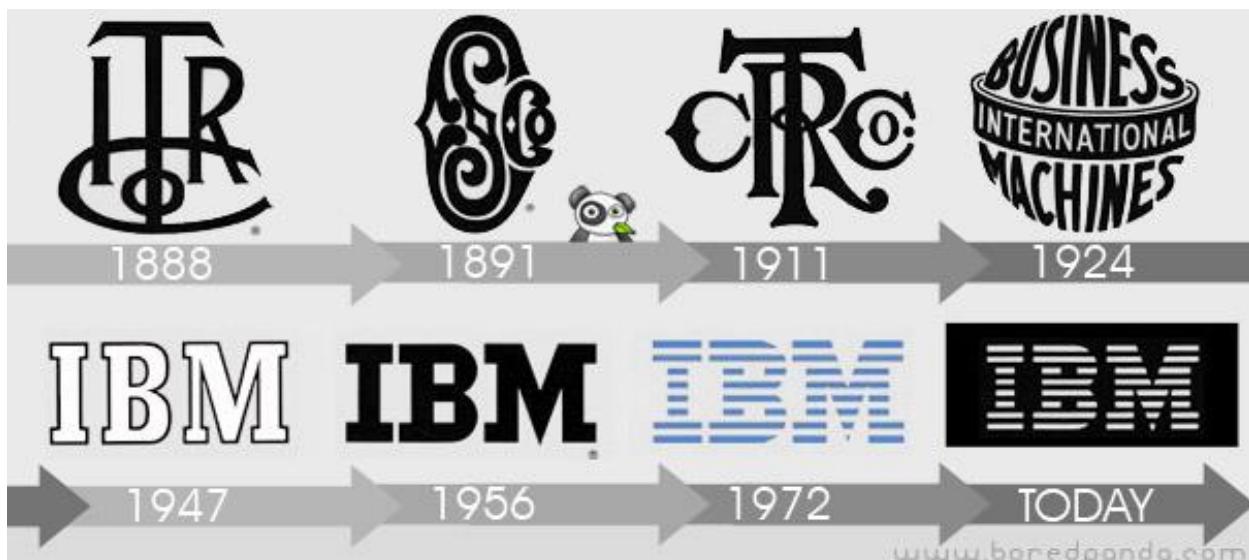
- Direccionamiento de Memoria



3. Datos y programas.

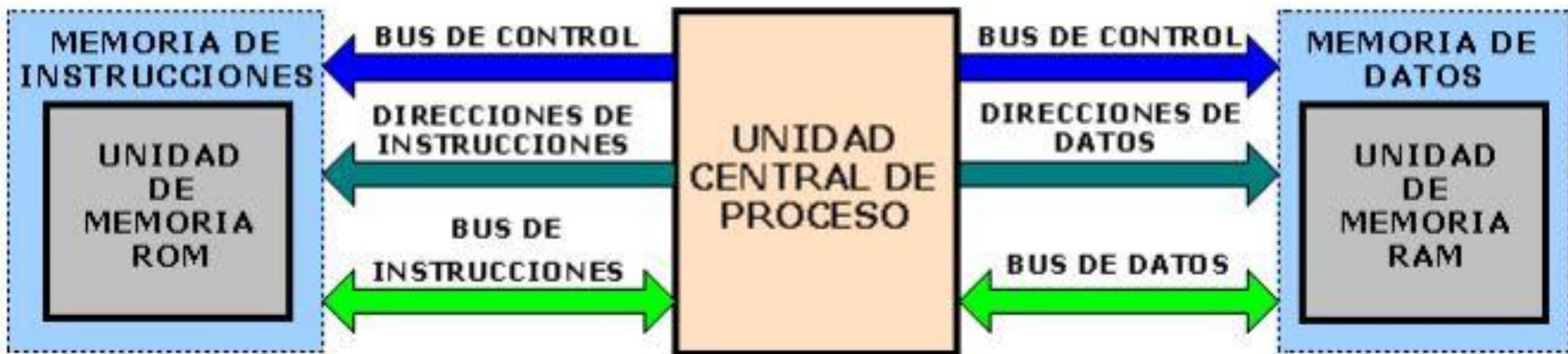


- Arquitectura Harvard. 1944.
- Harvard-IBM Mark 1 basado en relés.
- Programación realizada mediante el encendido y apagado de relés.
 - Interruptor eléctrico que permite el paso de la corriente eléctrica cuando está cerrado e interrumpirla cuando está abierto. → Lógica Binaria →(0,1)

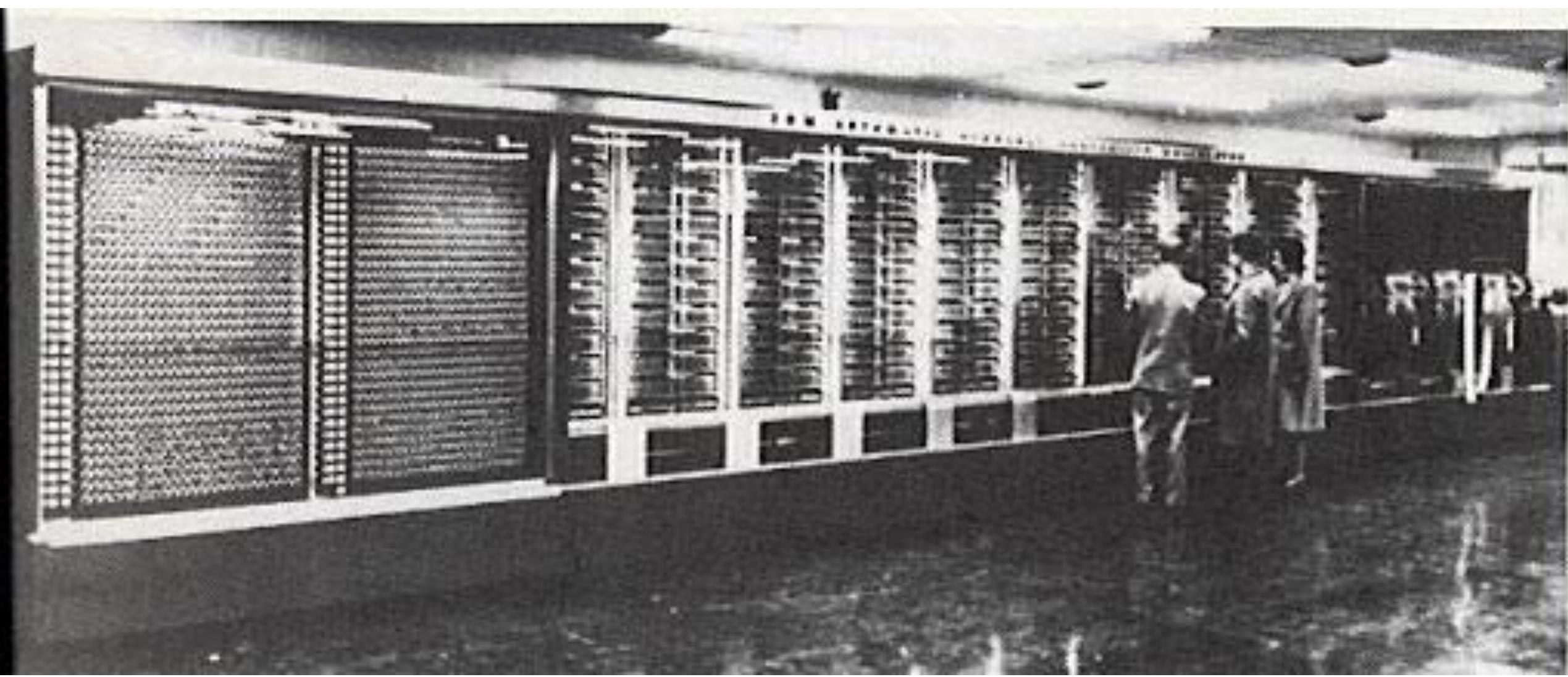


INSTRUCCIONES + DATOS

ARQUITECTURA HARVARD



IBM - MARK 1

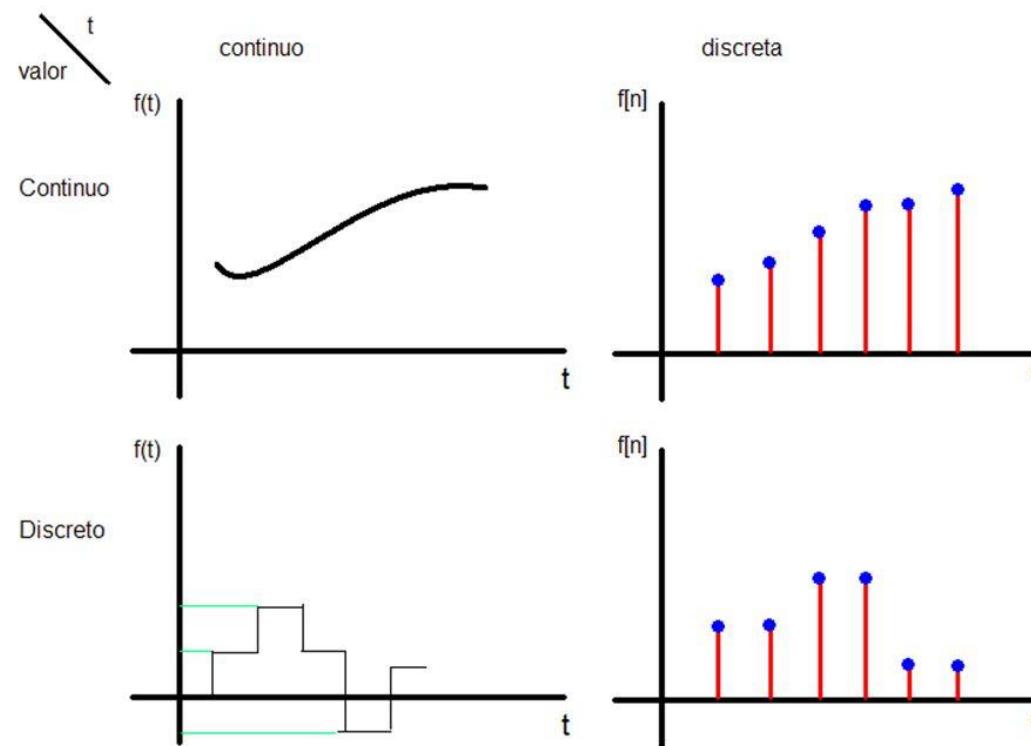


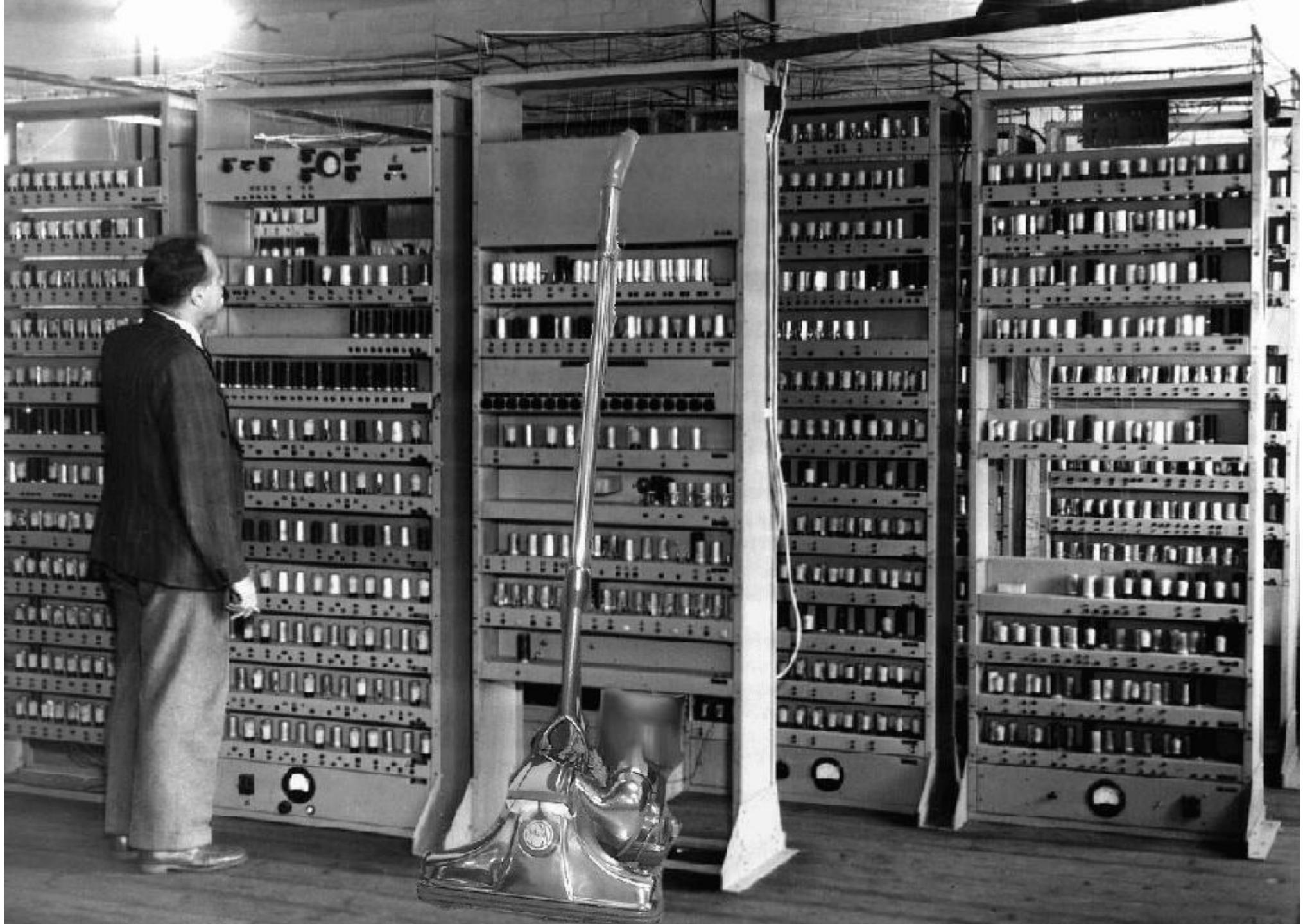
3. Datos y programas.

- Arquitectura Von Neumann.
- 1945.
- Húngaro nacionalizado estadounidense.
- Se aplicó al primer ordenador EDVAC.
- **Electronic Discrete Variable Automatic Computer.**



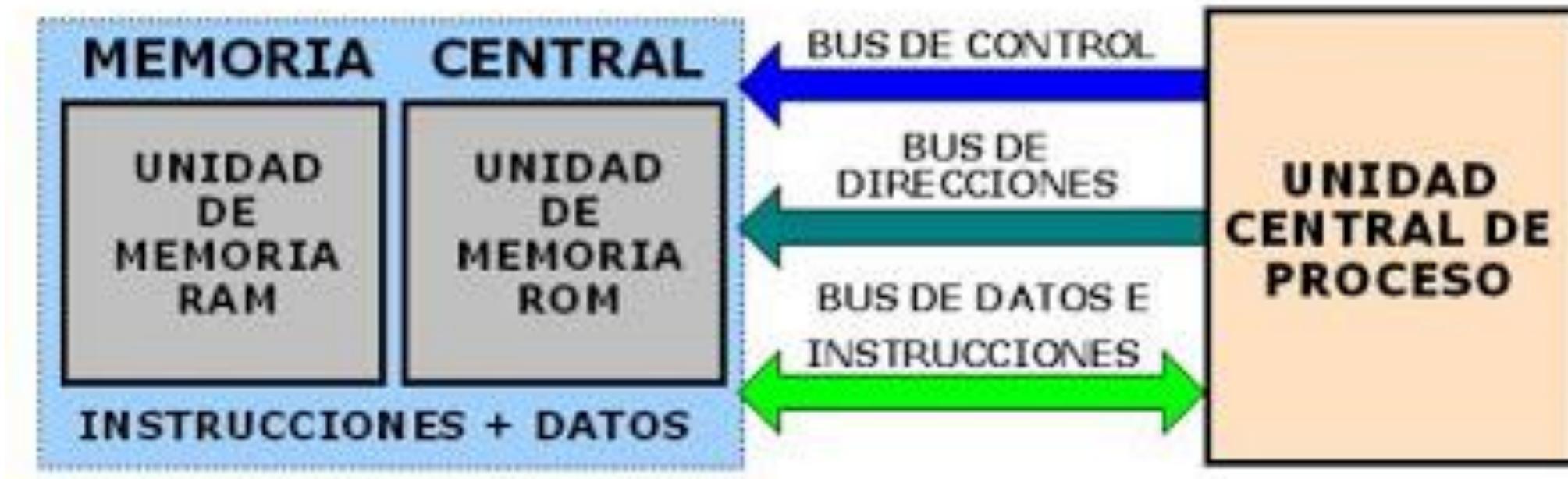
1903-1957





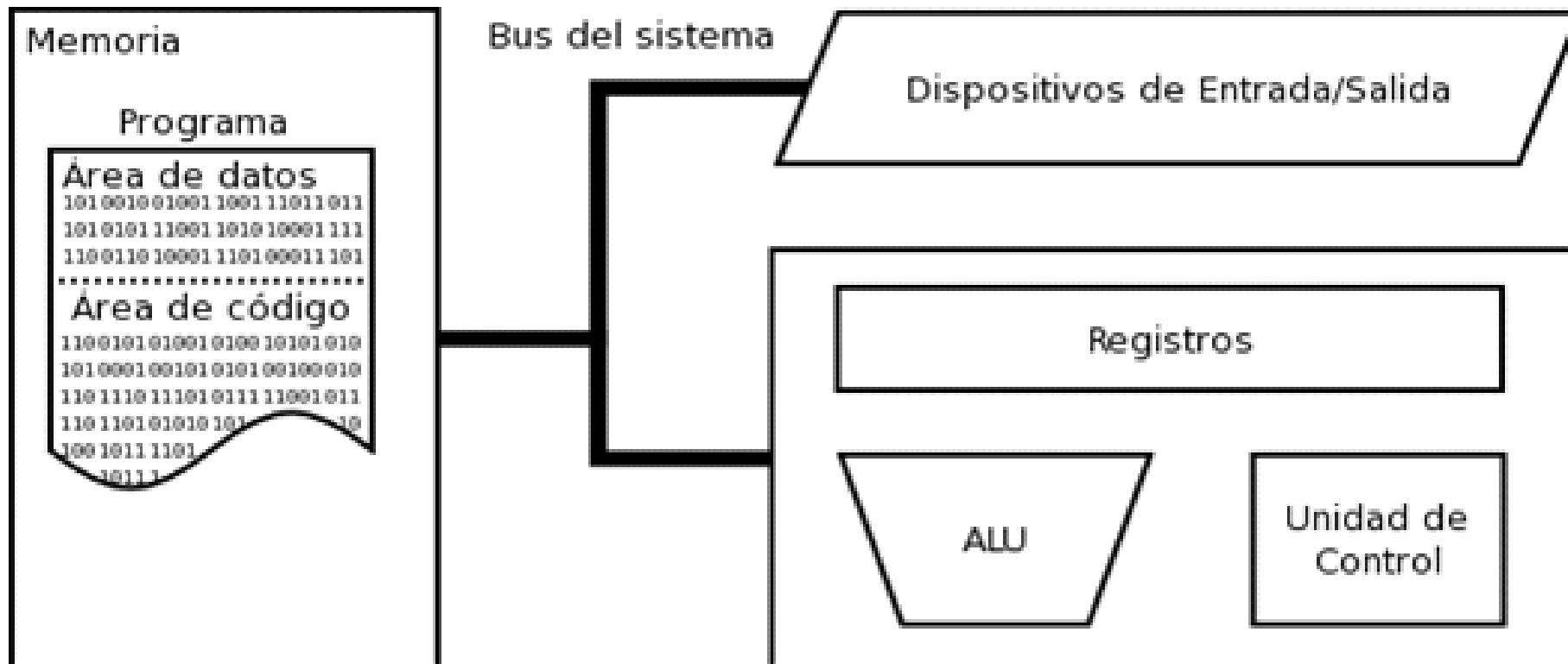
Arquitectura VON-NEUMANN

ARQUITECTURA VON NEUMANN



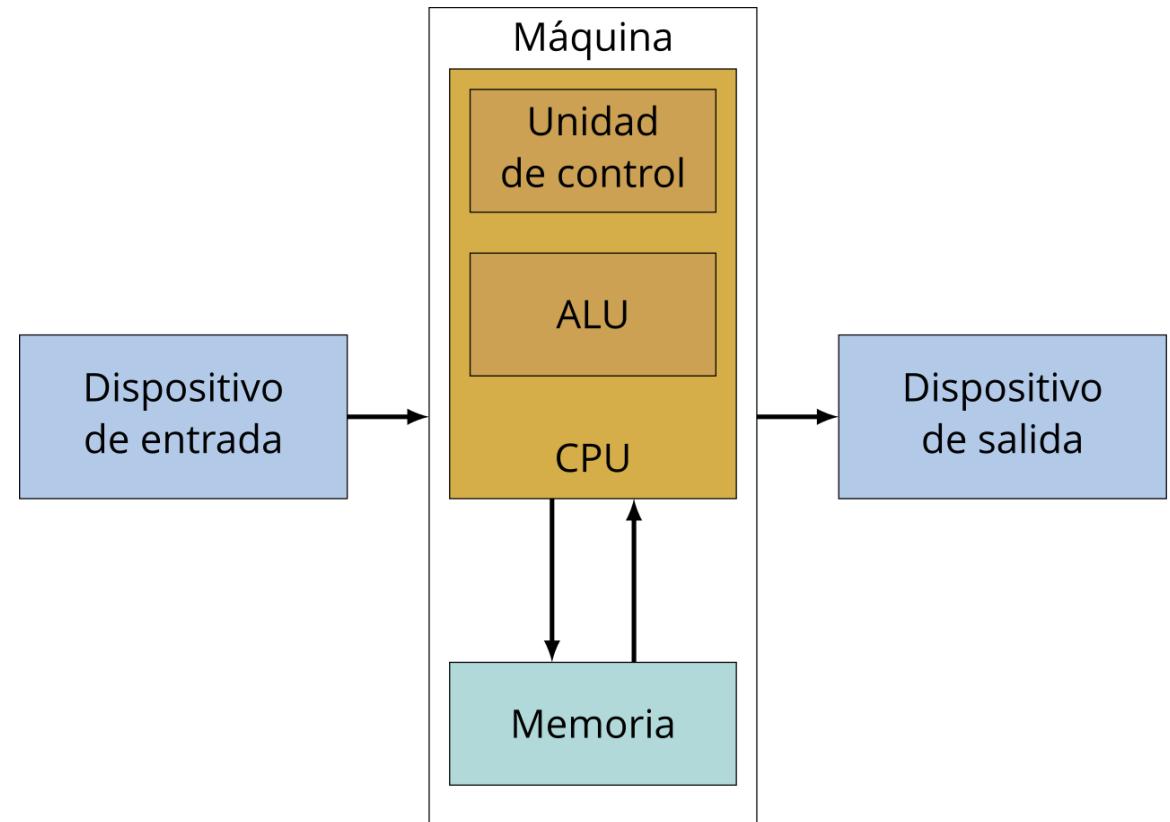
3. Datos y programas.

- Arquitectura Von Neumann. 1945.

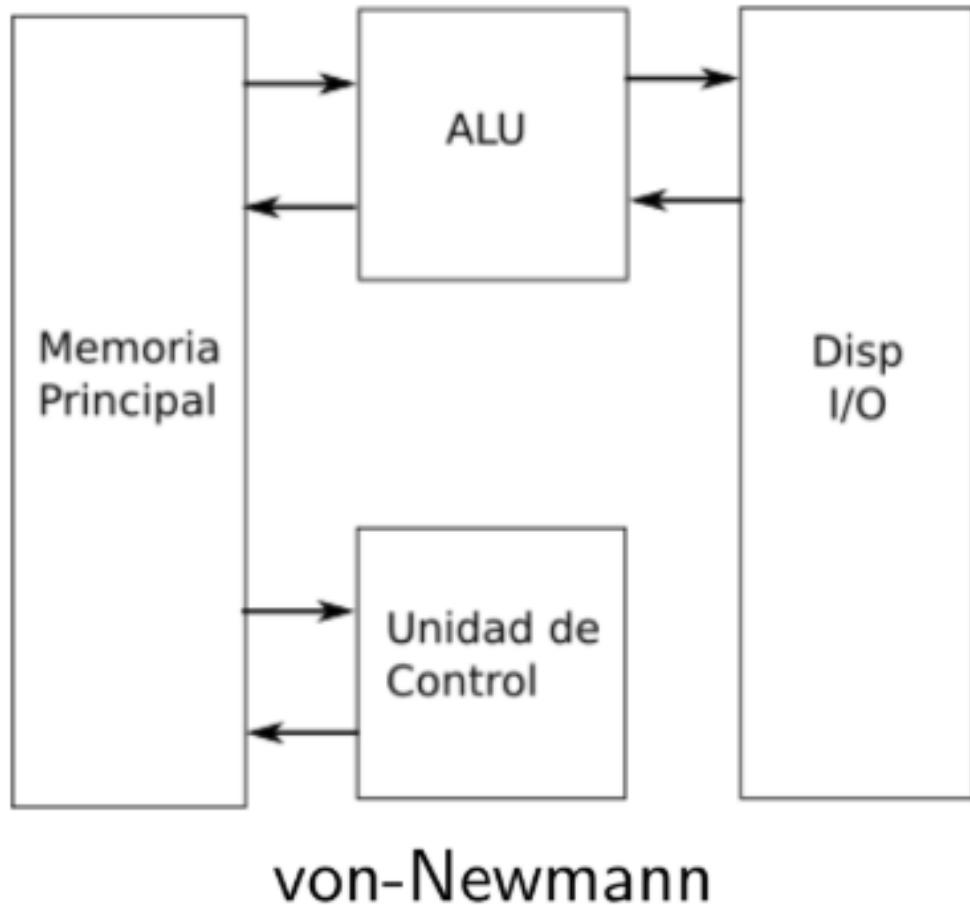


3. Datos y programas.

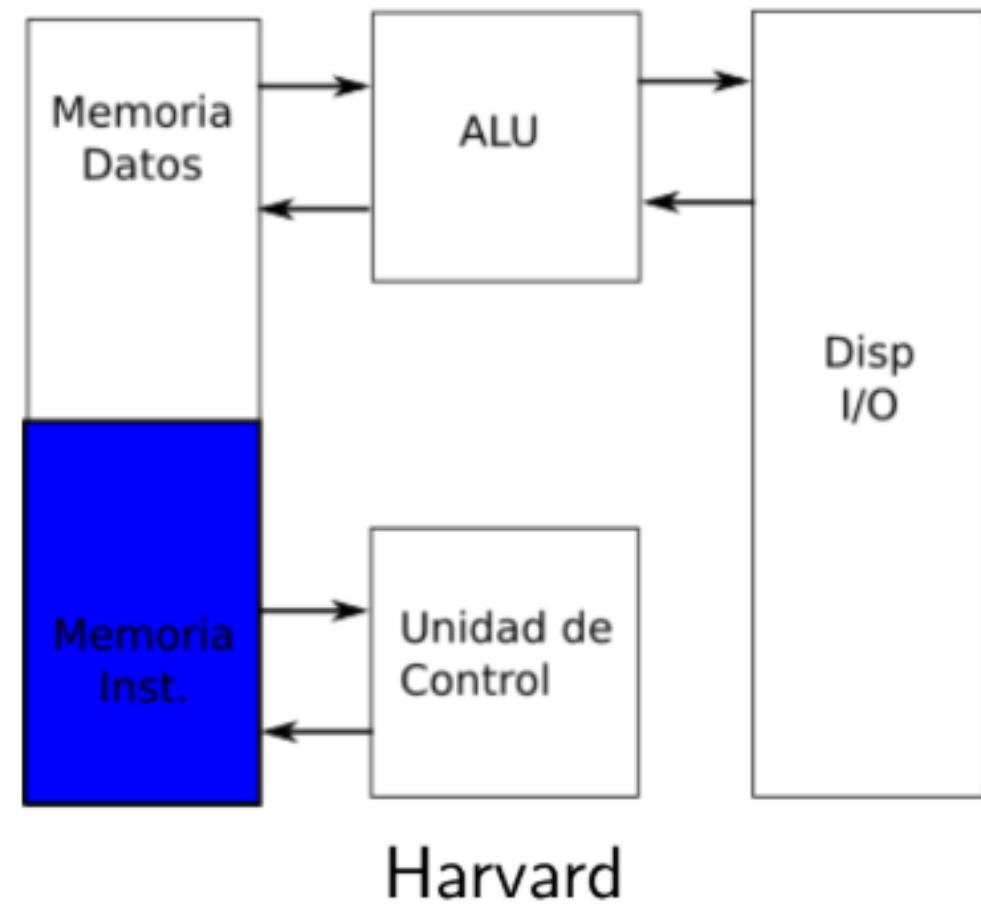
- Arquitectura Von Neumann.
1945.
- Elementos Funcionales:
 - Unidad de Control
 - Unidad Aritmético-lógica
 - Memoria Principal
 - Buses de ENTRADA / SALIDA



Diferencias



von-Newmann



Harvard

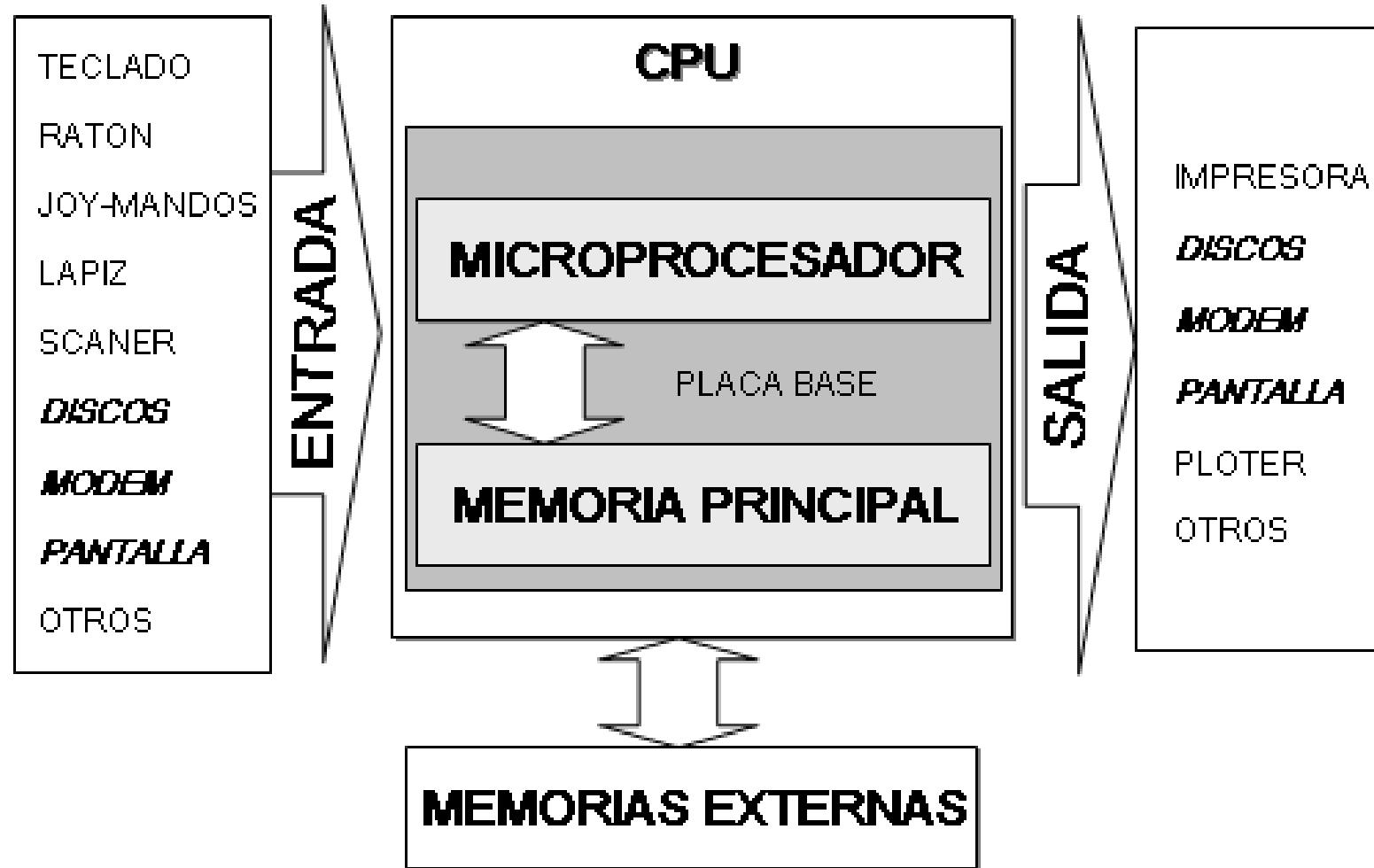
4. Hardware vs. Software.

¿Qué diferencias se os
ocurren?

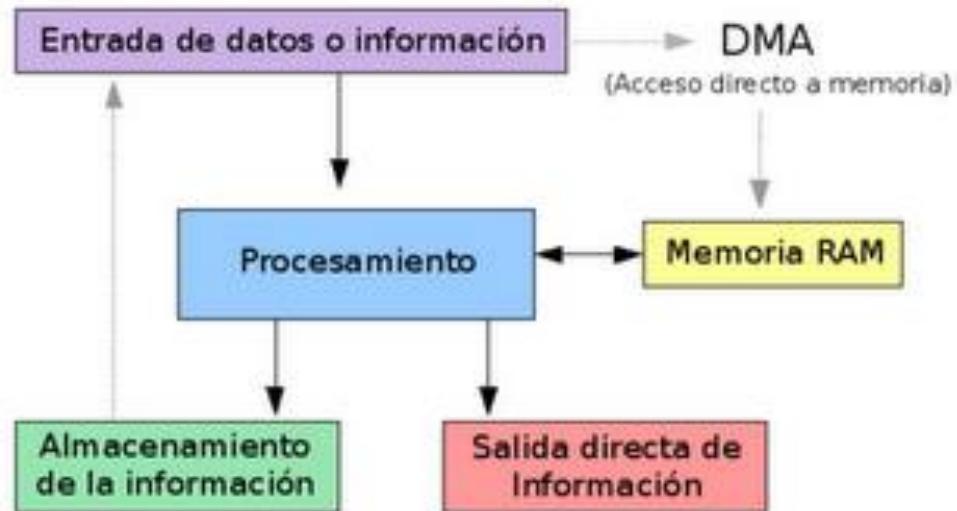
Hardware – Placa Base



5. Estructura funcional de un ordenador: procesador, memoria.



5. Estructura funcional de un ordenador: procesador, memoria.



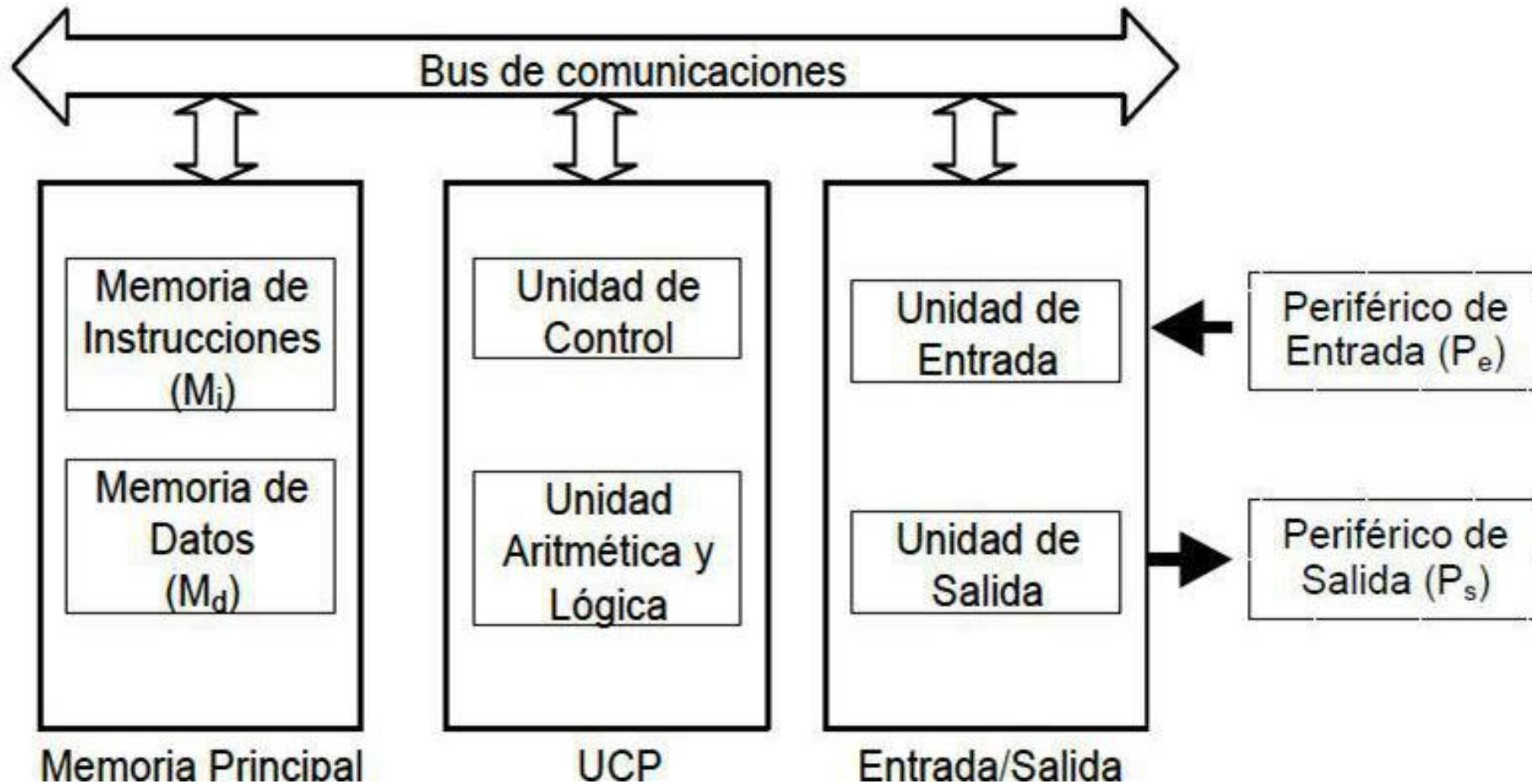
Ejemplos:

Teclado, Ratón (mouse), Micrófono, Pantalla táctil...

Disco Rígido, DVD/CD – R/RW, USB Drive...

Monitor, Altavoces, Impresora...

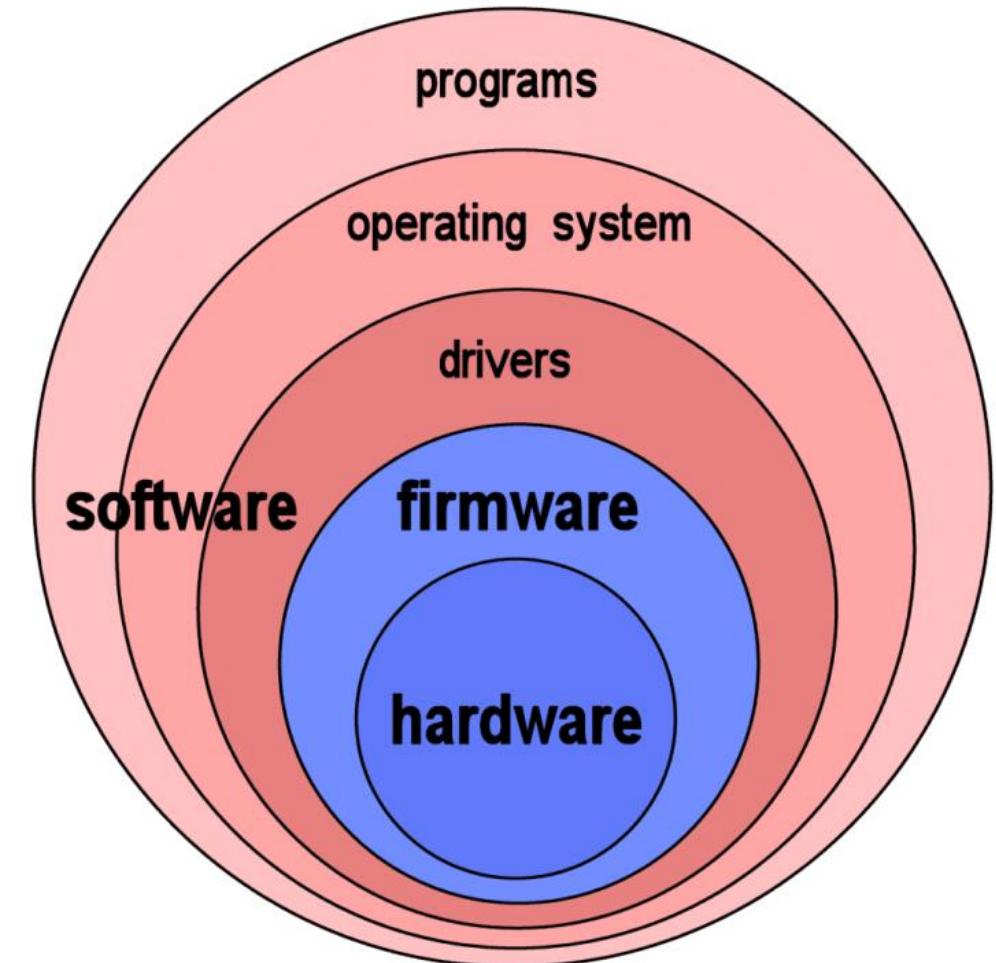
5. Estructura funcional de un ordenador: procesador, memoria.





6. Tipos de software.

- BIOS = Basic Input Output System
- Software de:
 - Sistema
 - De aplicación
 - De programación



Tipos de Software - SISTEMA

- **Software de sistema:** Permite que el hardware funcione. Ej. sistemas operativos, controladores de dispositivo, herramientas de diagnóstico, etc...
- También llamado firmware.
- Ejemplo: BIOS.



BIOS - AWARD

CMOS Setup Utility – Copyright (C) 1984–1999 Award Software

<ul style="list-style-type: none">▶ Standard CMOS Features▶ Advanced BIOS Features▶ Advanced Chipset Features▶ Integrated Peripherals▶ Power Management Setup▶ PnP/PCI Configurations▶ PC Health Status	<ul style="list-style-type: none">▶ Frequency/Voltage ControlLoad Fail-Safe DefaultsLoad Optimized DefaultsSet Supervisor PasswordSet User PasswordSave & Exit SetupExit Without Saving
<p>Esc : Quit ↑ ↓ → ← : Select Item F10 : Save & Exit Setup</p>	
<p>Time, Date, Hard Disk Type...</p>	



Easy Mode

Information

B450M GAMING
BIOS Ver.F1
AMD Ryzen 3 2200G with
Radeon Vega Graphics
Speed: 3514.00MHz
Memory: 8192MB

CPU Temperature

38.0 °C

CPU Vcore

1.344

System Load Optimized Defaults
32

DRAM Status

Frequency: 2409.60MHz
DIMM_1: 4096
DIMM_2: 4096

X.M.P. Profile1

Boot Sequence

- ② Windows Boot Manager (P0: Samsung SSD 860 EVO 250GB)
- ② P0: Samsung SSD 860 EVO 250GB
- ② ubuntu

Alt

Help

Fail-Safe Defaults

System Load Optimized Defaults

32

Set Supervisor Password

Set User Password

Save & Exit Setup

Exit Without Saving

Select Item

F11 : Save CMOS to BIOS

Save & Exit Setup

F12 : Load CMOS from BIOS

Save Data to CMOS

Tipos de Software - APLICACIÓN

- **Software de aplicación:** programas que nos ayudan a realizar tareas específicas en cualquier campo susceptible de ser automatizado o asistido.
- Ej. aplicaciones ofimáticas, software educativo, software médico, diseño asistido.



Tipos de Software - DESARROLLO

- **Software de desarrollo.** → Entornos de desarrollo
- Proporciona al programador herramientas para ayudarle a escribir programas informáticos y a usar diferentes lenguajes de programación de forma práctica.
- Ej. entornos de desarrollo integrados (IDE).



Lenguajes de Programación. Concepto

- 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 pueda comunicarse con los dispositivos de hardware y software existentes.
- 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 0s y 1s de grandes proporciones, labor sumamente tediosa.
- Se tomó la solución de establecer un nombre a las secuencias de programación más frecuentes, estableciéndolas en posiciones de memoria concretas, a cada una de estas secuencias nominadas se las llamó instrucciones, y al conjunto de dichas instrucciones, lenguaje ensamblador.
- Como un modo de facilitar la tarea de programar nace el concepto de lenguaje de alto nivel con Fortran (FORmula TRANslation) como primer debutante.
- Los lenguajes de alto nivel son aquellos que elevan la abstracción del código máquina lo más posible, para que programar sea una tarea más liviana, entendible e intuitiva. El compilador hará que de nuestro código solo lleguen 1s y 0s a la máquina.

Lenguajes de Programación. Clasificación y características

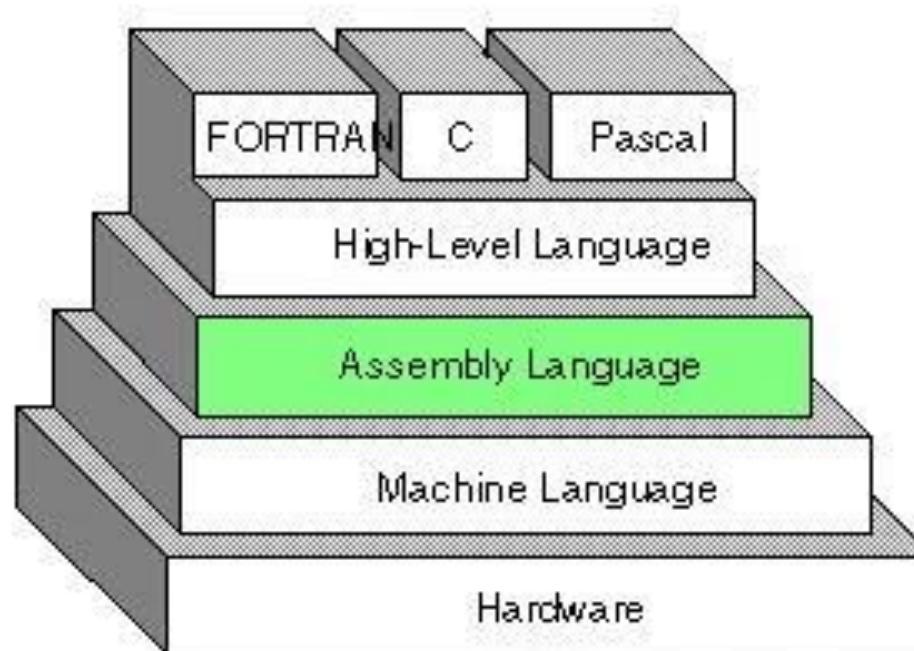
- Los lenguajes de programación se pueden clasificar mediante una gran variedad de criterios. En este capítulo vamos a clasificar los lenguajes de programación siguiendo tres criterios globales y reconocidos:
 - Nivel de abstracción
 - Forma de ejecución
 - Paradigma de programación.

Nivel de abstracción

- Llamamos nivel de abstracción al modo en que los lenguajes se alejan del código máquina y se acercan cada vez más a un lenguaje similar a los que utilizamos diariamente para comunicarnos.
- Cuanto más alejado esté del código máquina, de mayor nivel será el lenguaje.
- **Tipos de abstracción** según el nivel:
 - Lenguajes de **bajo** nivel:
 - Se acercan al funcionamiento de un ordenador
 - Específicos para cada procesador
 - Lenguaje máquina
 - Lenguajes de **medio** nivel:
 - Se utilizan para aplicaciones como la creación de sistemas operativos
 - Lenguaje C
 - Lenguajes de **alto** nivel:
 - Más fáciles de aprender, lenguaje natural
 - Independientes de la máquina
 - Sin conocimientos de código máquina
 - C++, COBOL, Fortran, Java, PHP, ...



- Lenguaje ensamblador



```

;*****  

;Programa E001.asm  

;Este programa suma dos valores inmediatos (7+8) y el resultado  

;lo deposita en la posición Ox10  

;Revisión: 0.0  

;Velocidad de reloj: 4 MHz  

;Perro Guardián: deshabilitado  

;Protección de código: OFF  

;*****  

        LIST      p=16F84 ;Tipo de PIC  

;*****  

RESULTADO    EQU Ox10          ;Define la posición del resultado  

;*****  

        ORG 0              ;Comando que indica al Ensamblador  

; la dirección de la memoria de programa  

; donde situar la siguiente instrucción  

;*****  

INICIO       movlw    Ox07      ;Carga primer sumando en W  

                addlw    Ox08      ;Suma W con segundo sumando  

                movwf    RESULTADO ;Almacena el resultado  

;  

END         ;Fin del programa fuente

```

Cabecera

Tipo de microcontrolador

Definición de constantes

Comentarios

Programa

Lenguaje de nivel medio - C

```
/*
 * power2.c - Print out powers of 2: 1, 2, 4, 8, .. up to 2^N
 * www.cyberciti.biz/faq/
 */

#include <stdio.h>
#define N 10

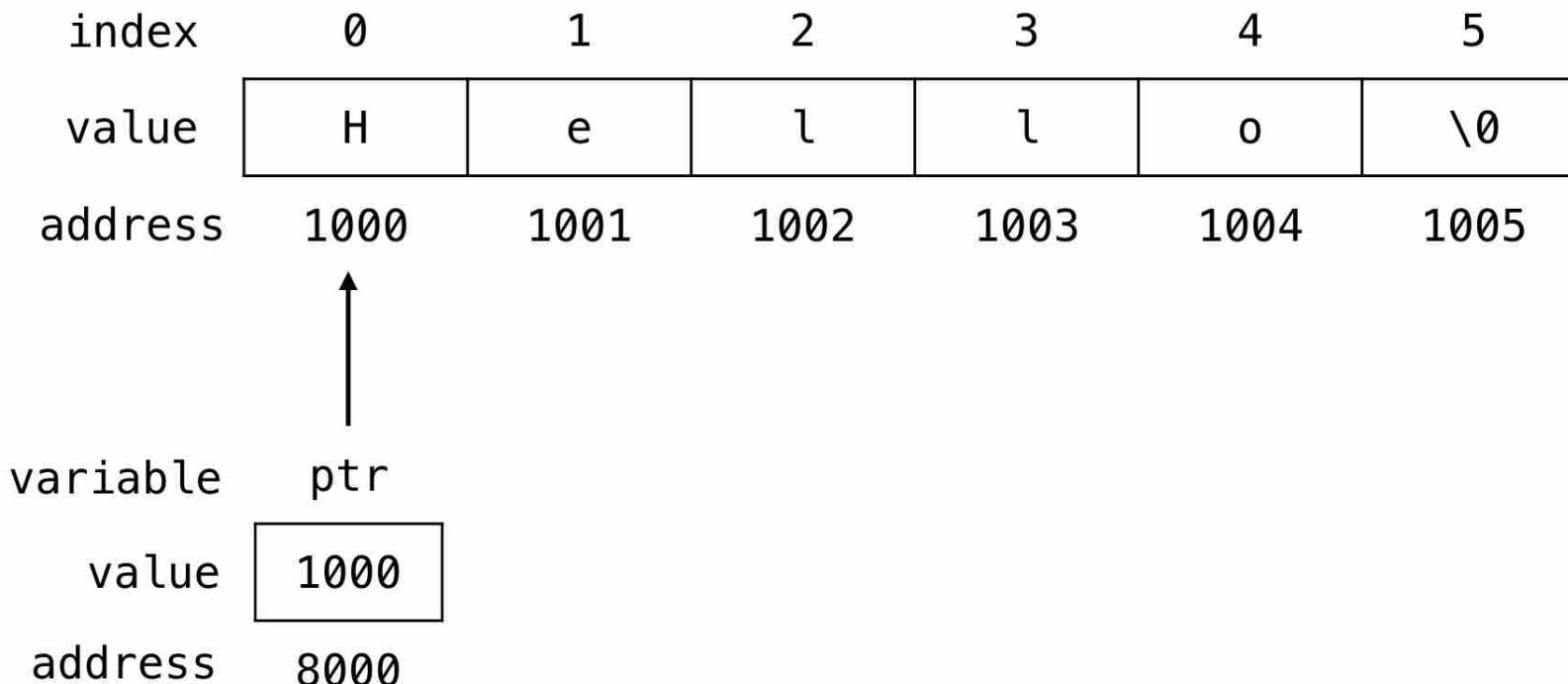
int main(void) {
    int n;          /* The current exponent */
    int val = 1;    /* The current power of 2 */

    printf("\t n \t 2^n\n");
    printf("\t-----\n");
    for (n=0; n<=N; n++) {
        printf("\t%3d \t %6d\n", n, val);
        val = 2*val;
    }
    return 0;
}
```

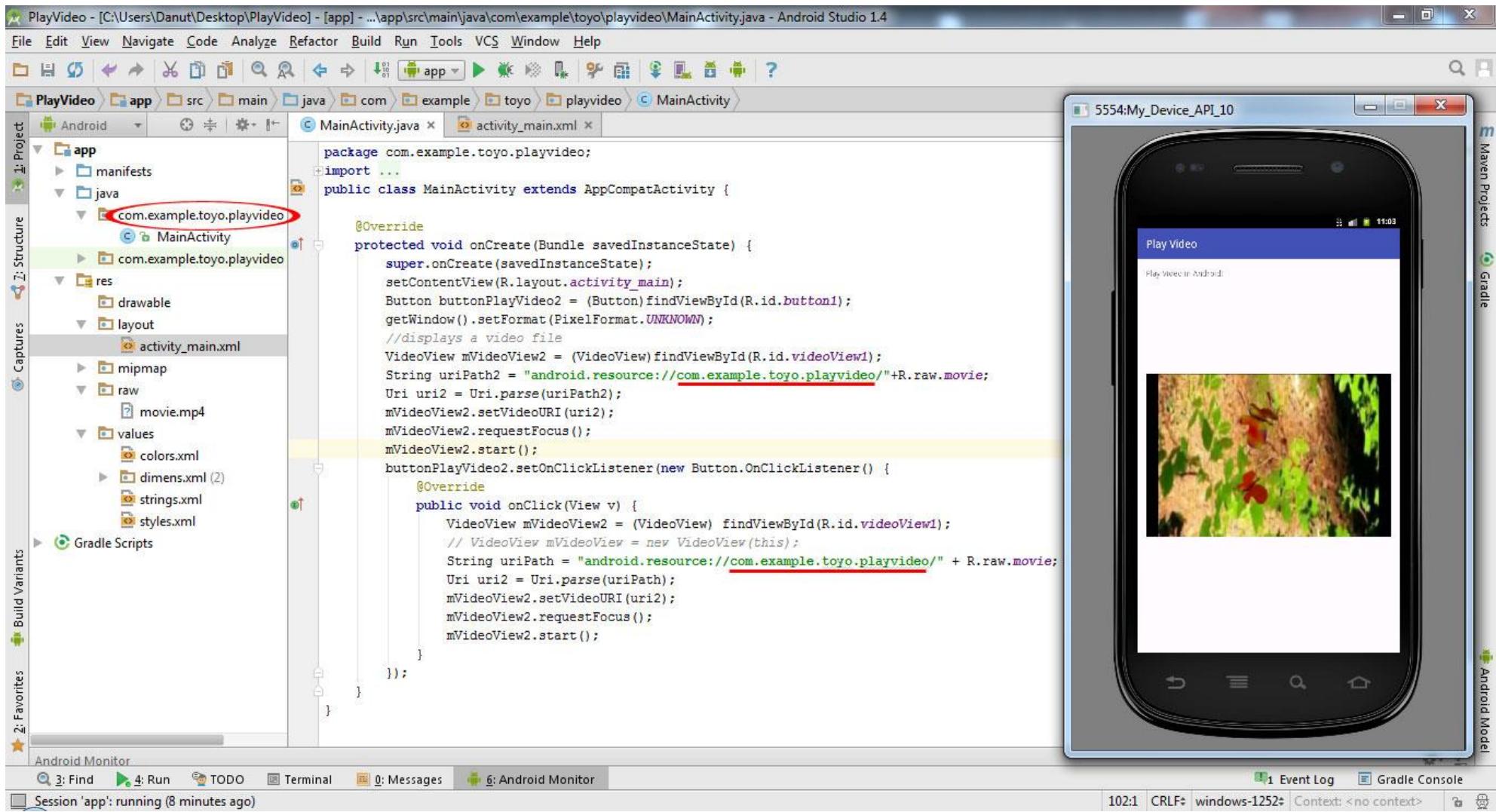
Lenguaje de nivel medio - C

DY CLASSROOM

```
char str[6] = "Hello";
```



Lenguaje de alto nivel



Paradigmas de lenguajes de programación

- Tipos:
 - Imperativo: C
 - Orientado a objetos: Java, C++, Haskell
 - Declarativo:
- Características
- Actividad. Busca en Internet qué tipo de lenguajes existen y cuáles son sus características.

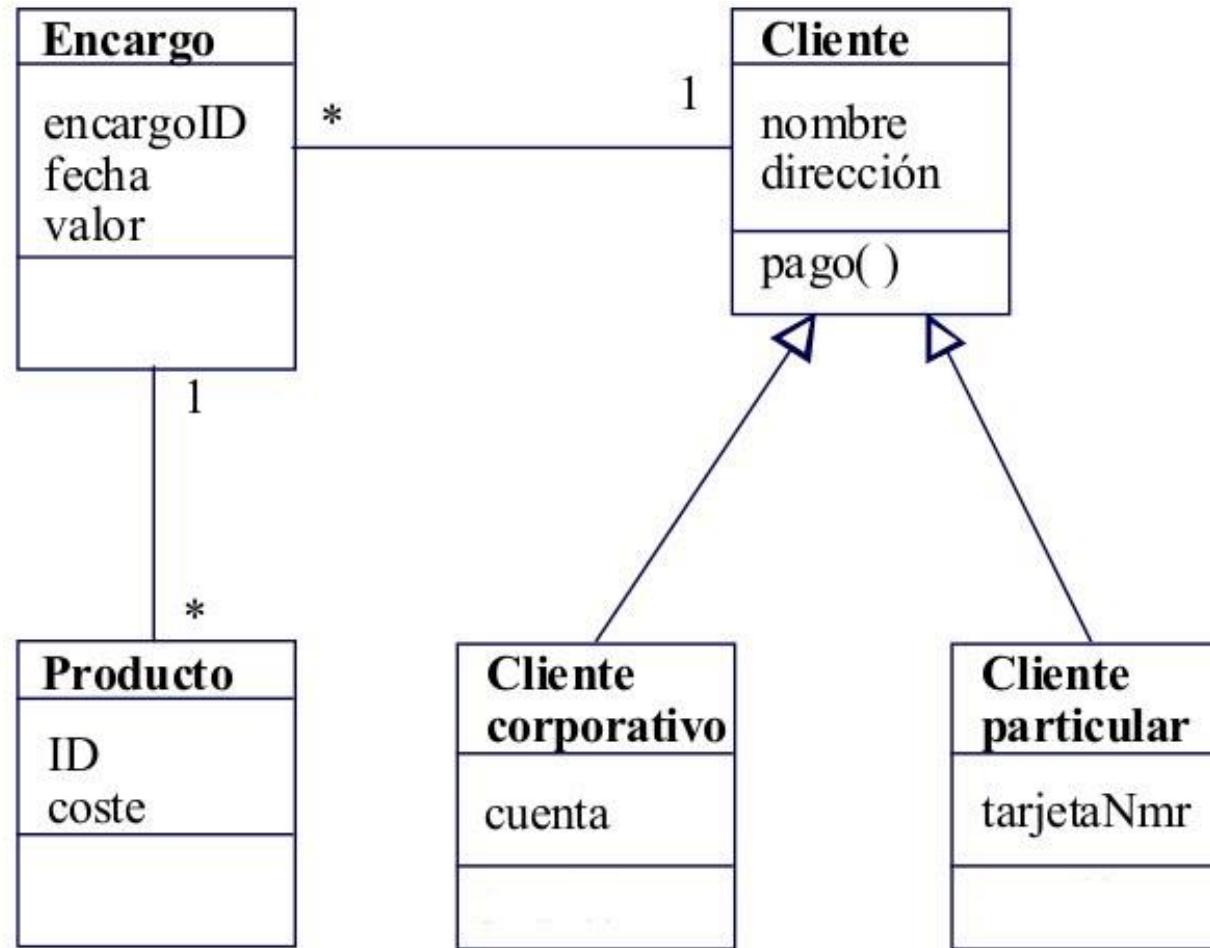
Lenguajes Imperativos

- Un lenguaje imperativo programa mediante una serie de comandos, agrupados en bloques y compuestos de órdenes condicionales que permiten al programa retornar a un bloque de comandos si se cumple la condición.
- Estos fueron los primeros lenguajes de programación en uso y aún hoy muchos lenguajes modernos usan este principio.
- No obstante, los lenguajes imperativos estructurados carecen de flexibilidad debido a la secuencialidad de las instrucciones.

Lenguajes orientados a objetos (OO)

- Características:
 - Herencia: jerarquía de clases preexistente
 - [https://es.wikipedia.org/wiki/Herencia_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Herencia_(inform%C3%A1tica))
 - Cohesión:
 - Abstracción.
 - Polimorfismo: Sobrecarga de métodos / funciones.
 - Acoplamiento.
 - Encapsulamiento: Objetos contenidos dentro de otros.

Lenguajes orientados a objetos (OO)



Lenguajes funcionales/declarativos

- Crea programas mediante funciones, devuelve un nuevo estado de resultado y recibe como entrada el resultado de otras funciones.
- Cuando una función se invoca a sí misma, hablamos de recursividad.

Lenguajes funcionales/declarativos

- Ejemplo función de Fibonacci.
- Función recursiva.

Tipos de lenguajes según la forma de ejecución

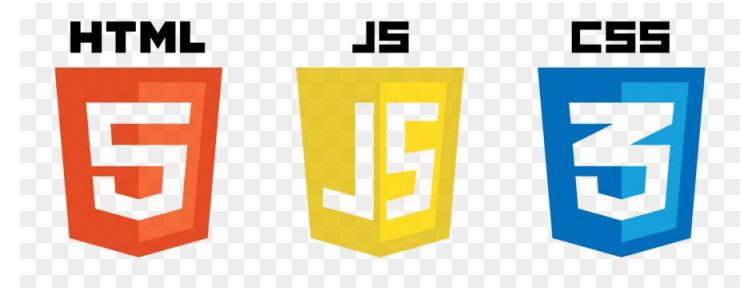
- **Lenguajes compilados:**

- **Compilador:** convierte el código fuente en código objeto.
- **Enlazador:** une el código objeto del programa con el código objeto de las librerías necesarias para producir el programa ejecutable. “programa.exe”



- **Lenguajes interpretados:**

- Ejecutan las instrucciones secuencialmente y en tiempo real, sin que se genere código objeto.
- Programa intérprete.



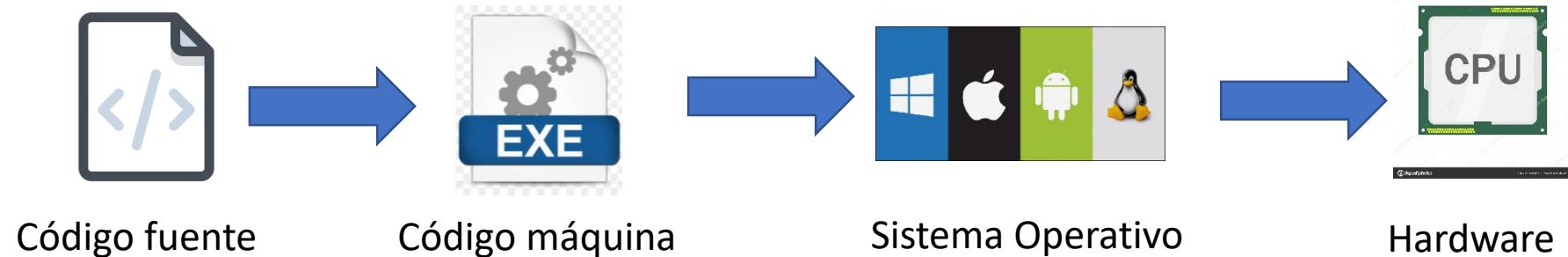
- **Lenguajes virtuales:**

- Multiplataforma
- Se pasa de código fuente a bytecode.
- El bytecode es ejecutado por la máquina virtual
- Más lento

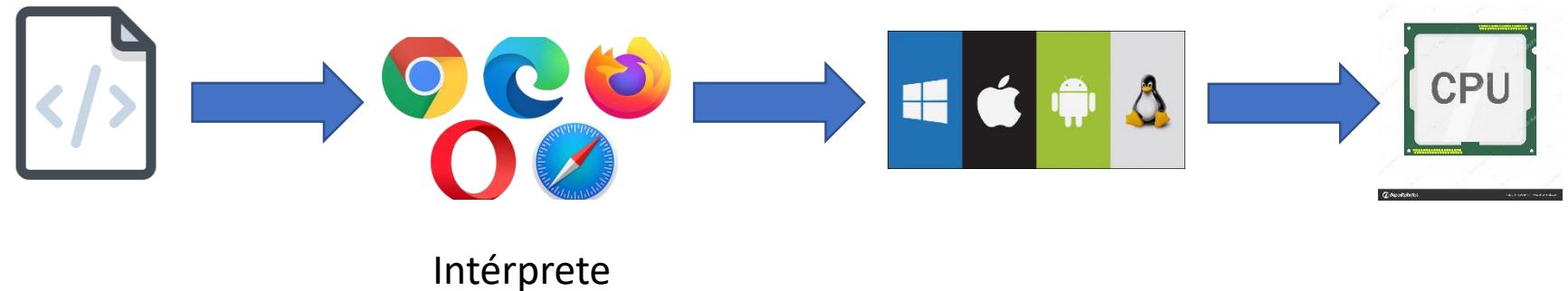


Tipos de lenguajes según la forma de ejecución

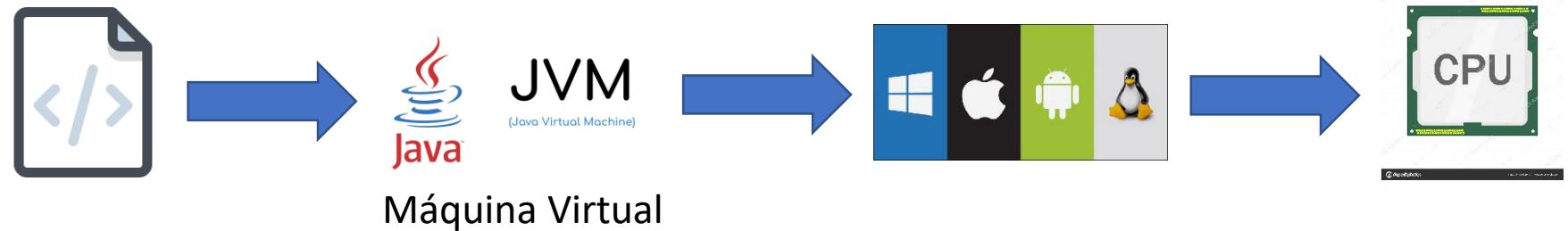
- Compilados



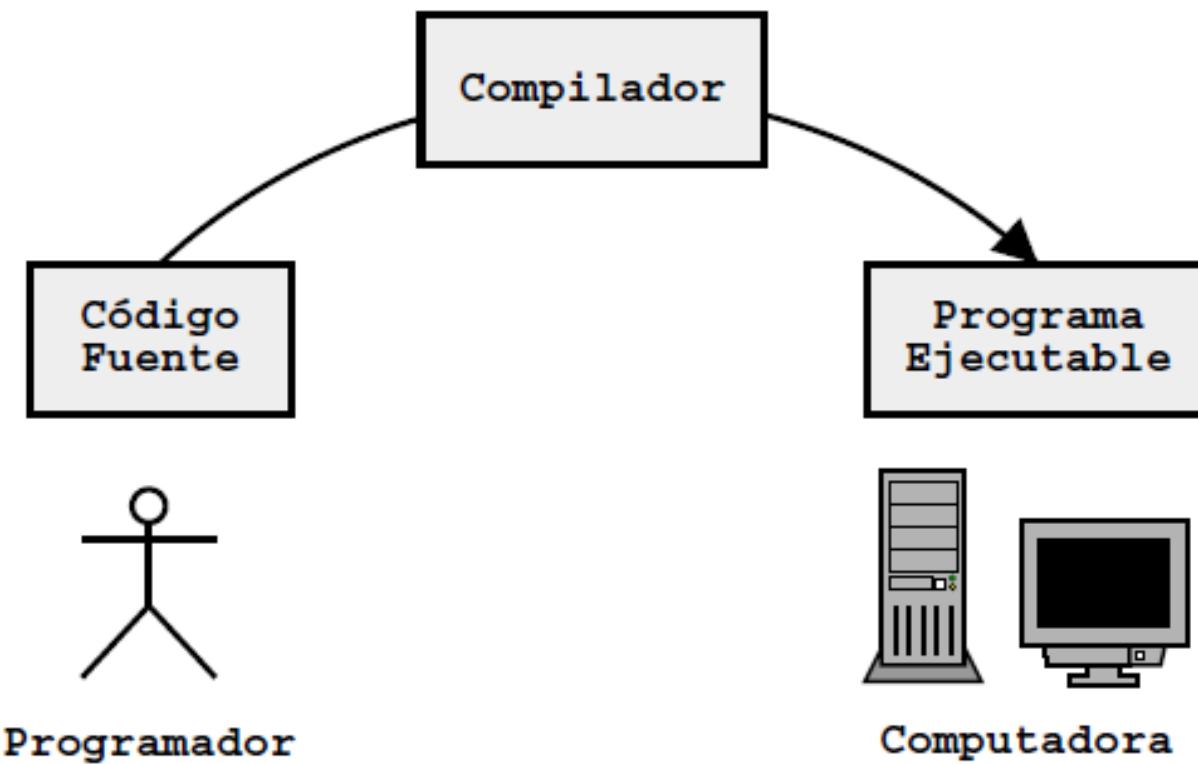
- Interpretados



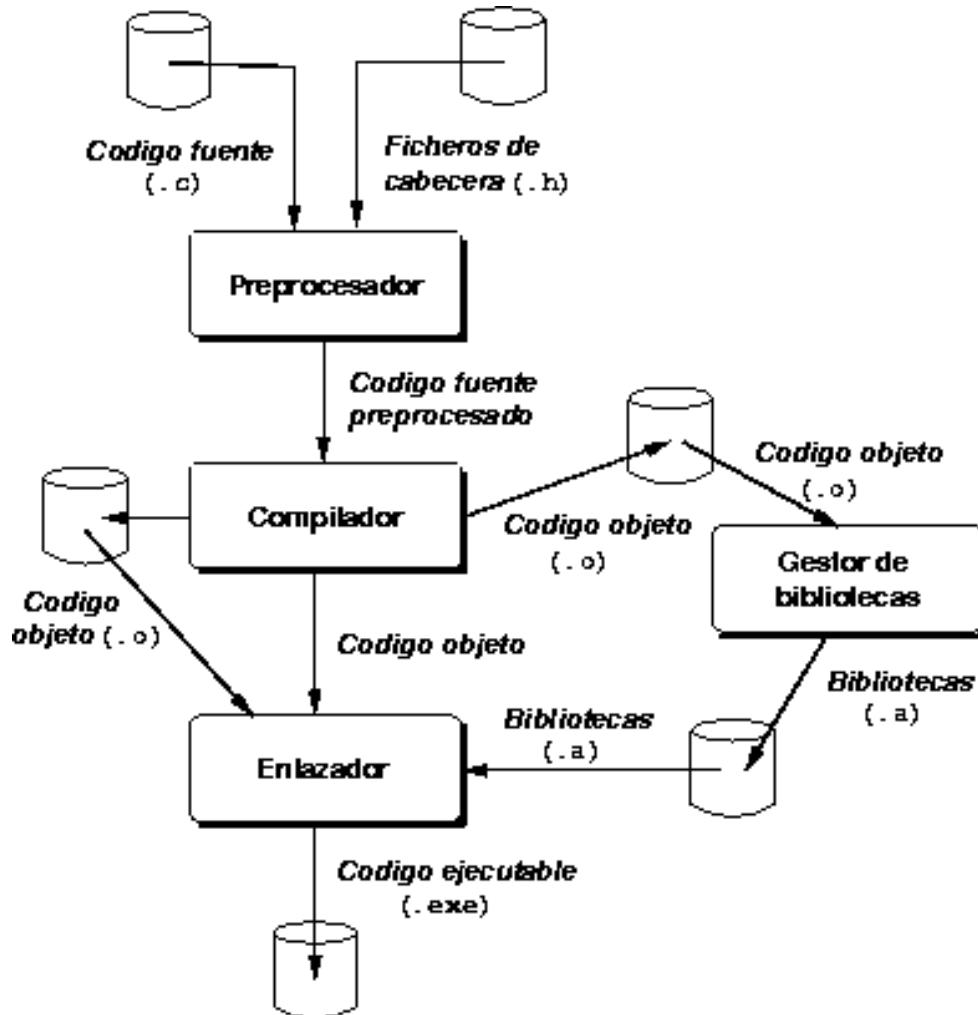
- Virtuales



Proceso de creación de un programa



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

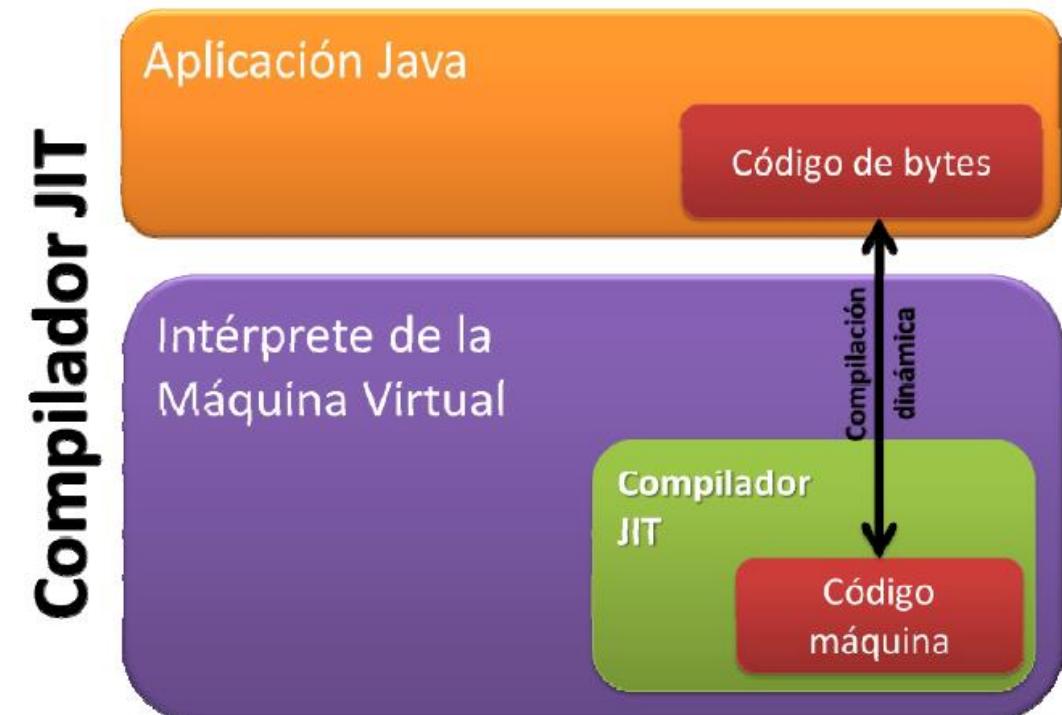


7. Código fuente, código objeto y código ejecutable; máquinas virtuales.

- **Código fuente:** conjunto de instrucciones escritas en un lenguaje de programación determinado. Es el código en el que nosotros escribimos nuestro programa.

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

- **Código objeto:** el código objeto es el código 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**.
- Java → Bytecodes
- Write Once, Run Anywhere...

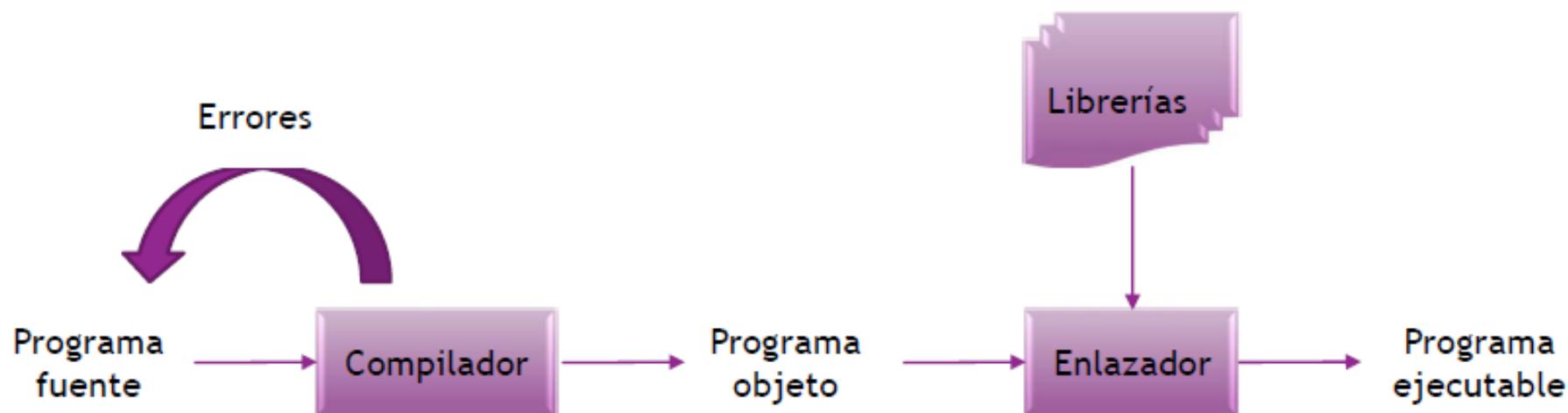


7. Código fuente, código objeto y código ejecutable; máquinas virtuales.

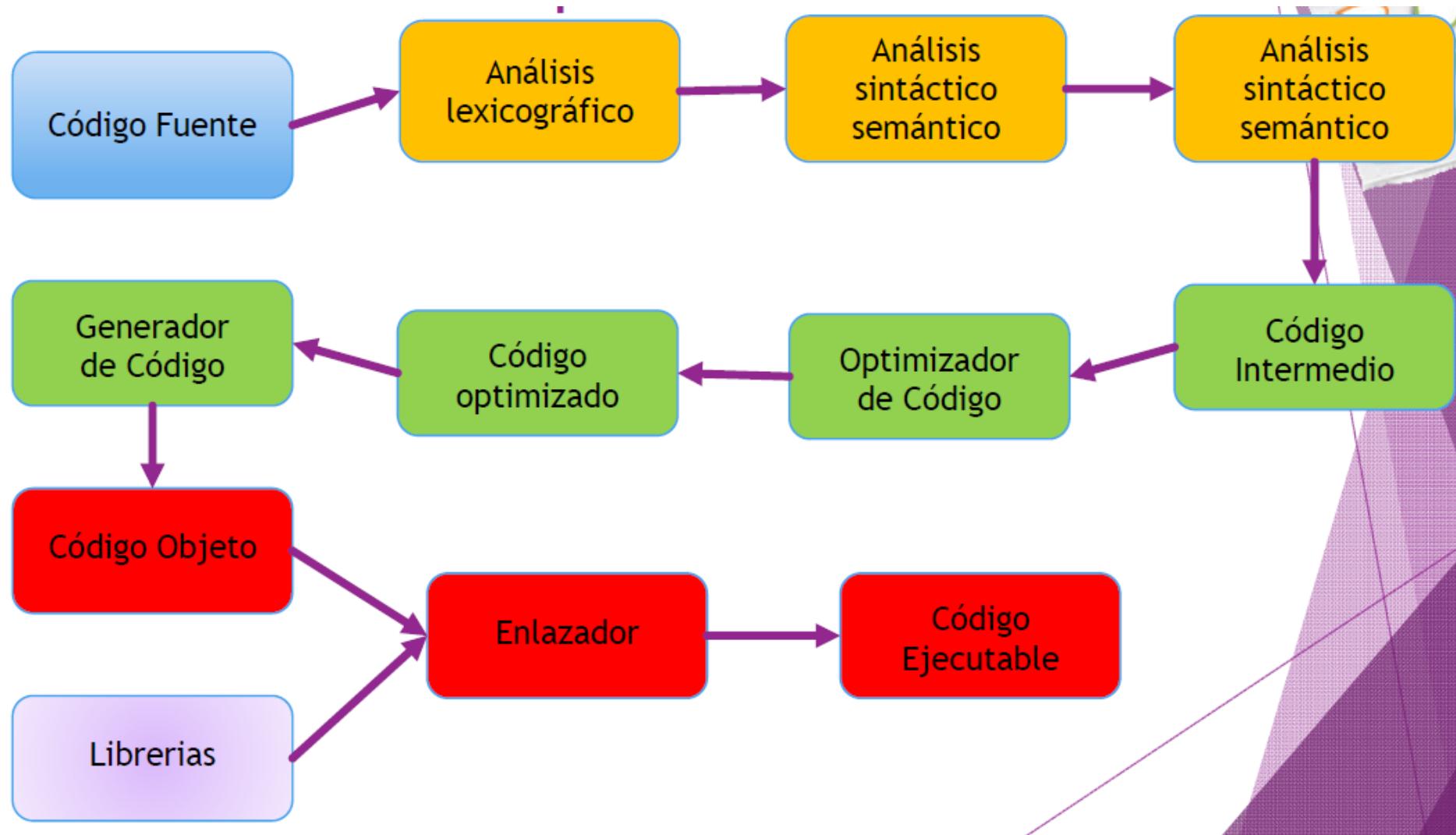
- **Código ejecutable:** resultado obtenido de enlazar nuestro código objeto con las librerías.
- Este código ya es nuestro programa ejecutable, programa que se ejecutará directamente en nuestro sistema o sobre una máquina virtual en el caso de los lenguajes de programación virtuales.

Proceso de compilación

Aunque el proceso de obtener nuestro código ejecutable pase tanto por un **compilador** como por un **enlazador**, se suele llamar al proceso completo “**compilación**”.



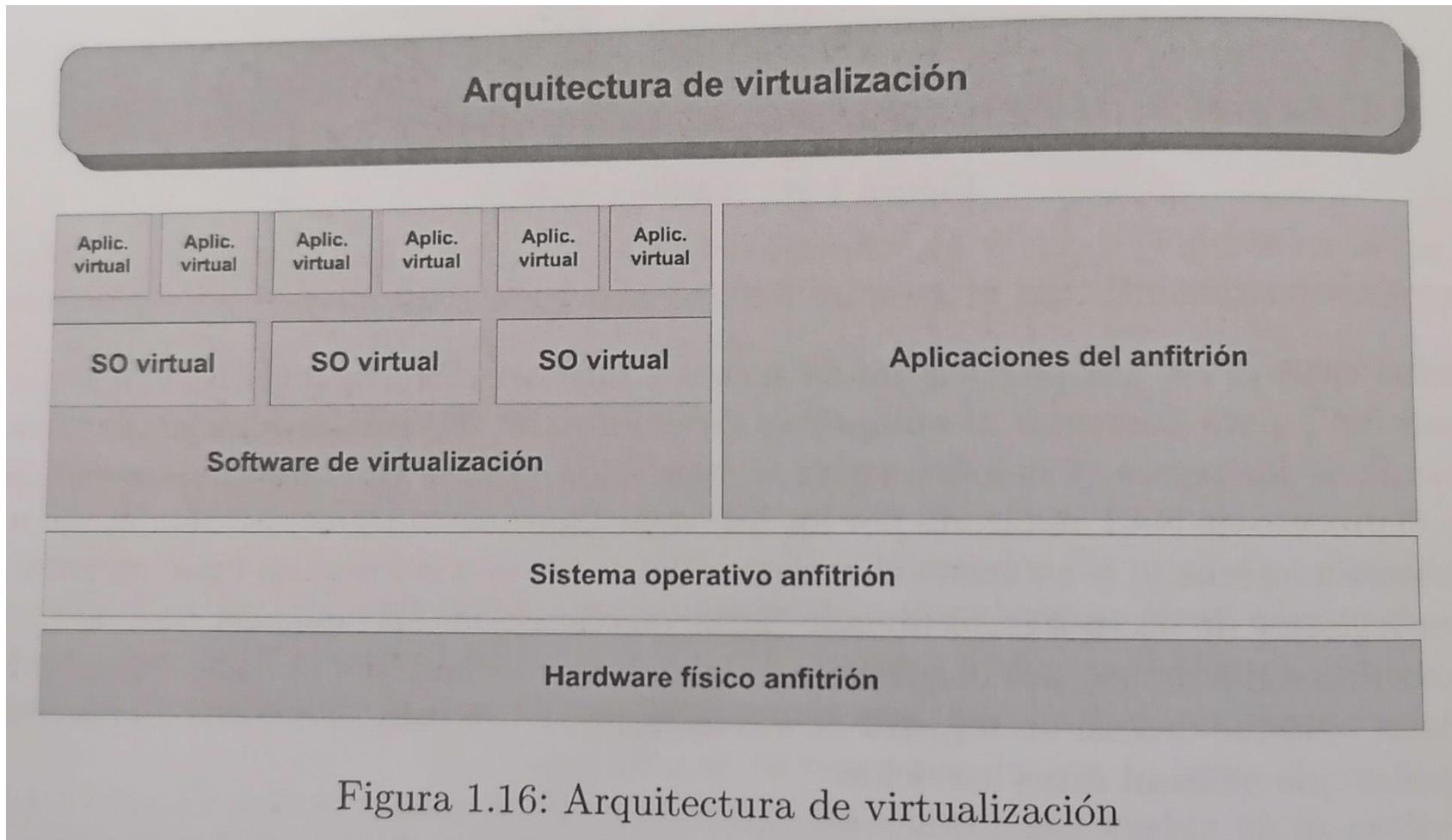
Fases del proceso de compilación



Fases del proceso de compilación

- **Análisis lexicográfico:** se leen de manera secuencial todos los caracteres de nuestro código fuente, buscando palabras reservadas, operaciones, caracteres de puntuación y agrupándolos todos en cadenas de caracteres que se denominan lexemas.
- **Análisis sintáctico-semántico:** agrupa todos los componentes léxicos estudiados en el análisis anterior en forma de frases gramaticales.
 - Con el resultado del proceso del análisis sintáctico, se revisa la coherencia de las frases gramaticales, si su “significado” es correcto, si los tipos de datos son correctos, si los arrays tienen el tamaño y tipo adecuados, y así consecutivamente con todas las reglas semánticas de nuestro lenguaje.
- **Generación de código intermedio:** una vez finalizado el análisis, se genera una representación intermedia a modo de pseudoensamblador con el objetivo de facilitar la tarea de traducir al código objeto.
- **Optimización de código:** revisa el código generado en el paso anterior optimizándolo para que el código resultante sea más fácil y rápido de interpretar por la máquina.
- **Generación de código:** genera el código objeto de nuestro programa.
- **Enlazador de librerías:** como se ha comentado anteriormente, se enlaza nuestro código objeto con las librerías necesarias, produciendo en último término nuestro código final o código ejecutable.

7. Máquinas virtuales.



7. Máquinas virtuales.

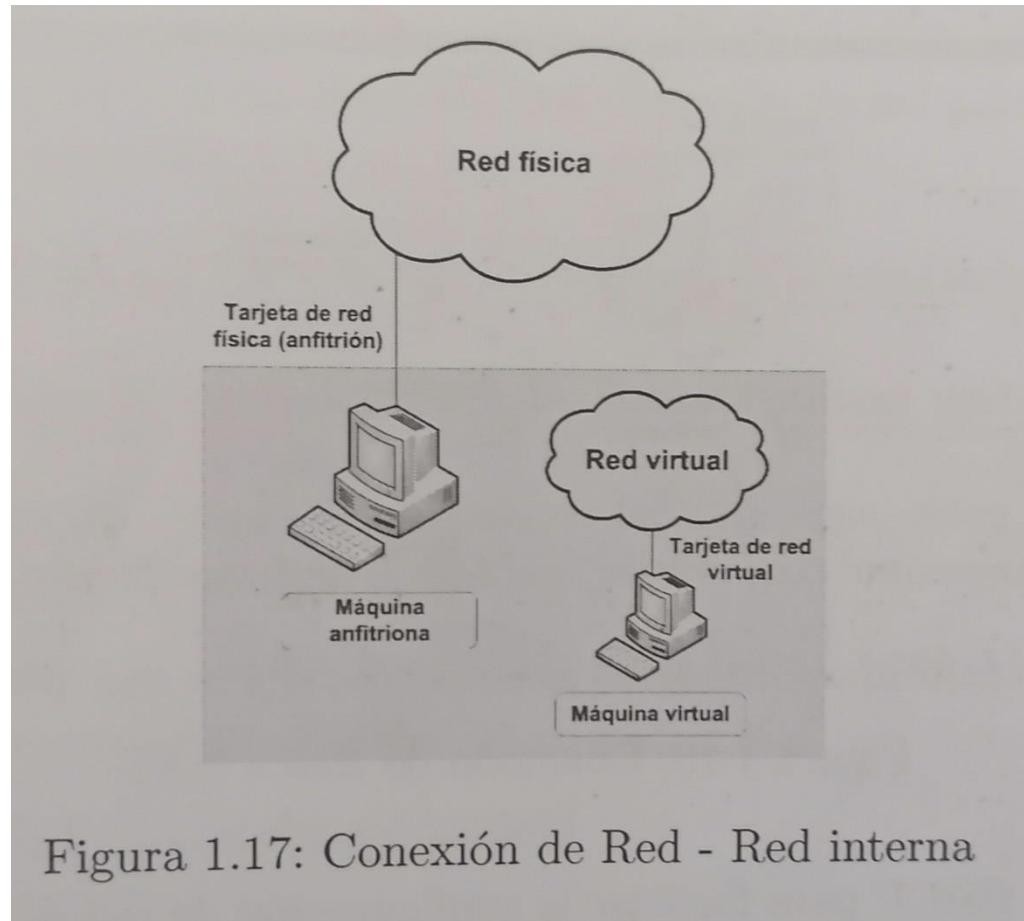
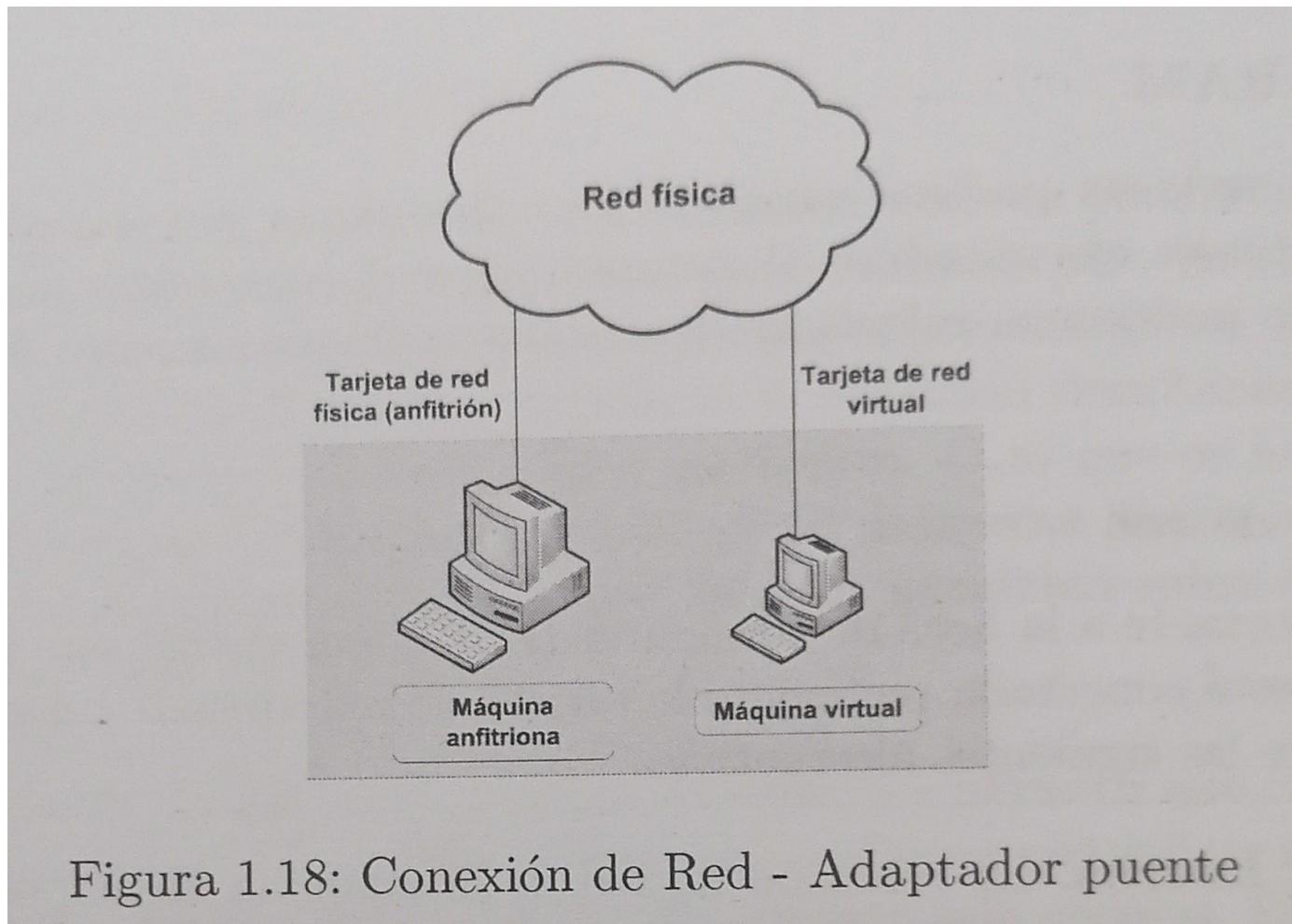
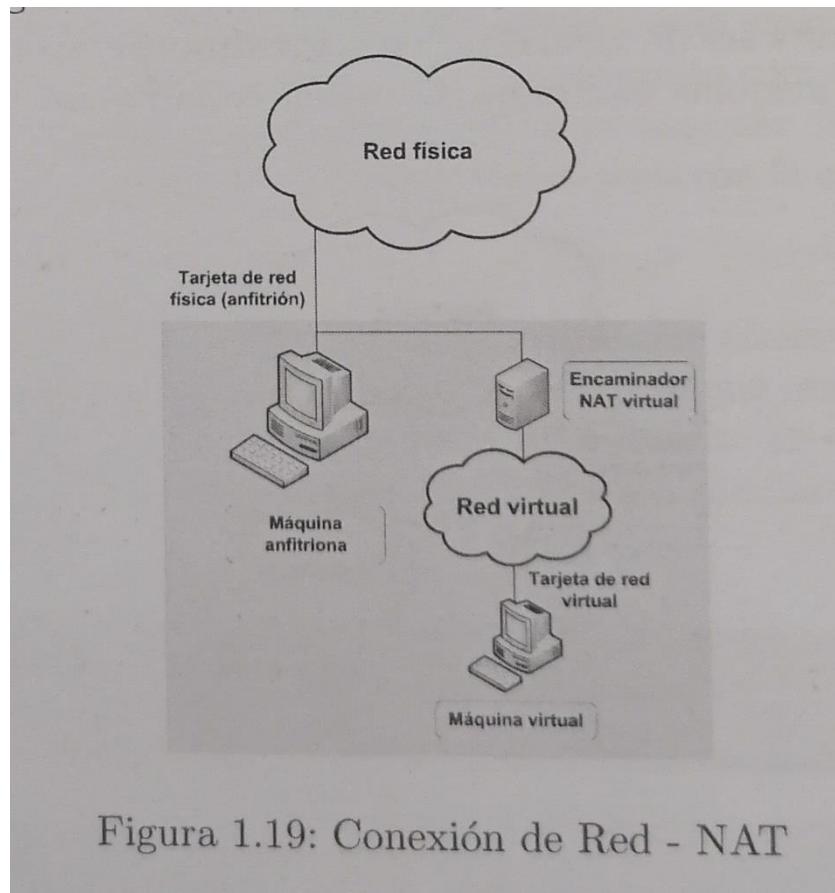


Figura 1.17: Conexión de Red - Red interna

7. Máquinas virtuales.

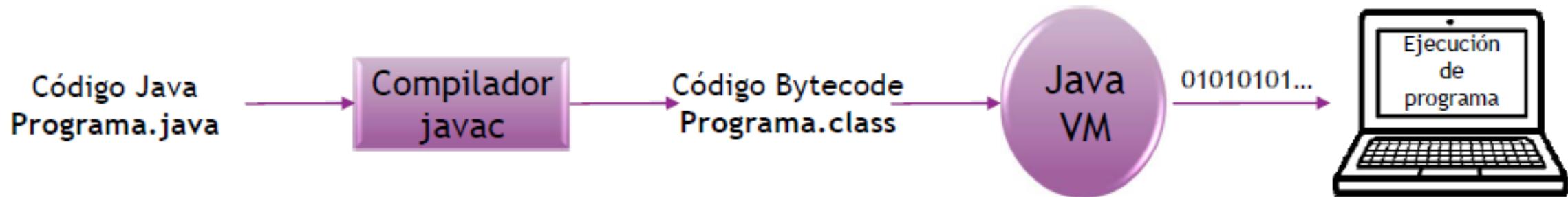


7. Máquinas virtuales.



JAVA Virtual Machine

- Los programas compilados en Java son multiplataforma gracias a que los programas los ejecuta la Máquina Virtual de Java (JVM, Java Virtual Machine).



JVM

- Las tareas principales de la Máquina Virtual de Java son las siguientes:
- Reservar espacio en memoria para los objetos creados y liberar la memoria no utilizada.
- Comunicarse con el sistema huésped (sistema donde se ejecuta la aplicación) para funciones como controlar el acceso a dispositivos hardware.
- Vigilar las normas de seguridad de las aplicaciones Java.
- La principal desventaja de los lenguajes basados en máquina virtual es que son **más lentos** que los completamente compilados, debido a la sobrecarga que genera tener una capa de software intermedia.



Ejemplo de juego hecho en JAVA

Proceso de creación de un videojuego



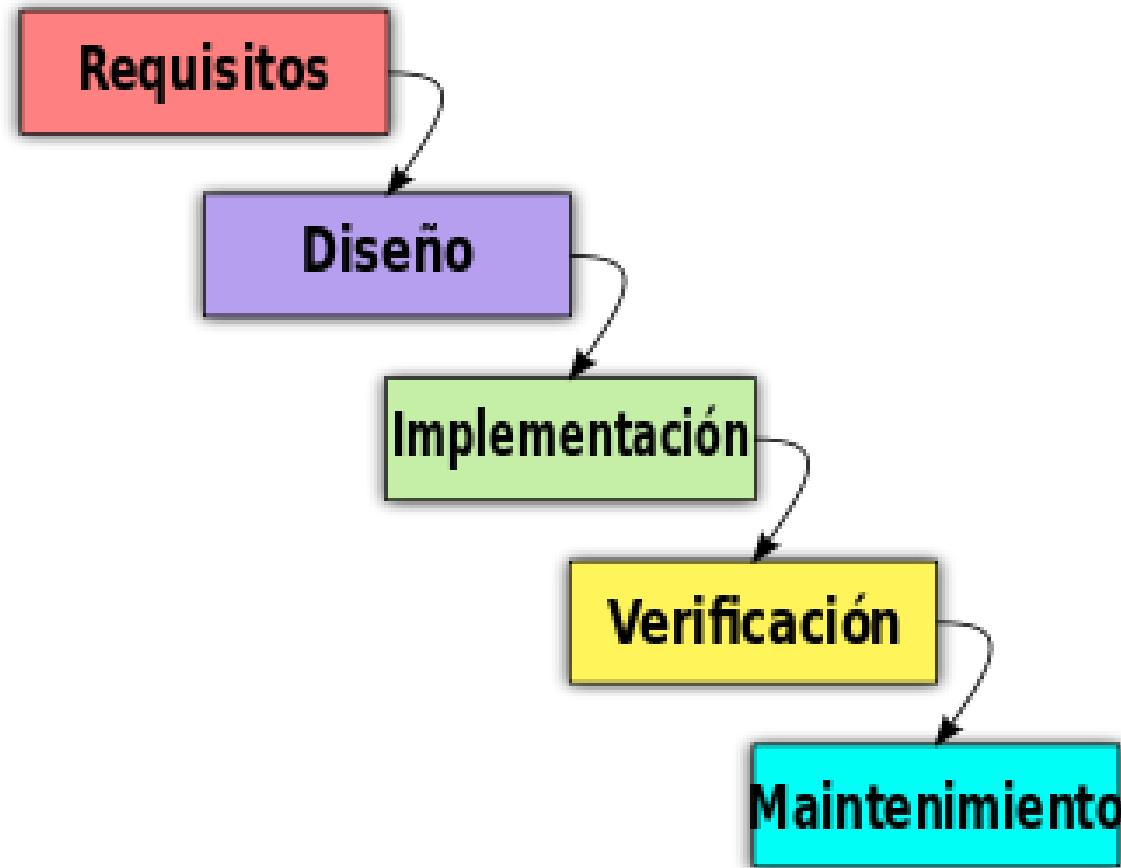
A screenshot from a video game, likely Assassin's Creed, showing a character in a stone wall with scaffolding. The character is seen from behind, wearing a white tunic and brown pants. The background features a large, multi-story stone building under construction or renovation, with wooden scaffolding and ropes visible.

Ejemplo de juego hecho en lenguaje compilado

9. Introducción a la Ingeniería del Software

- Proceso software y ciclo de vida del software.
- Fases del desarrollo de una aplicación: análisis, diseño, codificación, pruebas, documentación, explotación y mantenimiento.
- Roles dentro de un equipo de desarrollo software.

Ciclo tradicional (heredado de la construcción)



TRADICIONAL



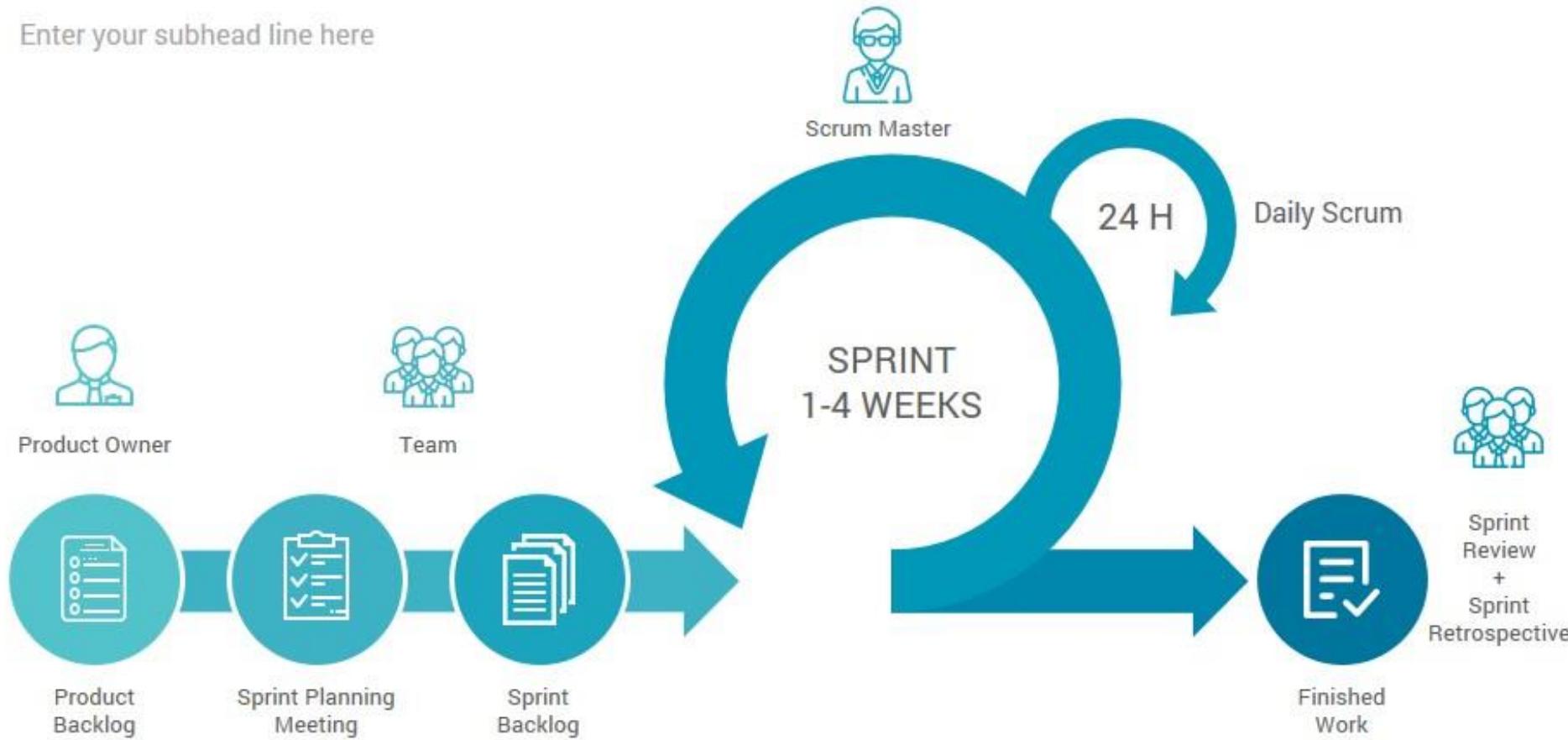
AGILE



Desarrollo con SCRUM – Vigente en la actualidad

Scrum Process

Enter your subhead line here



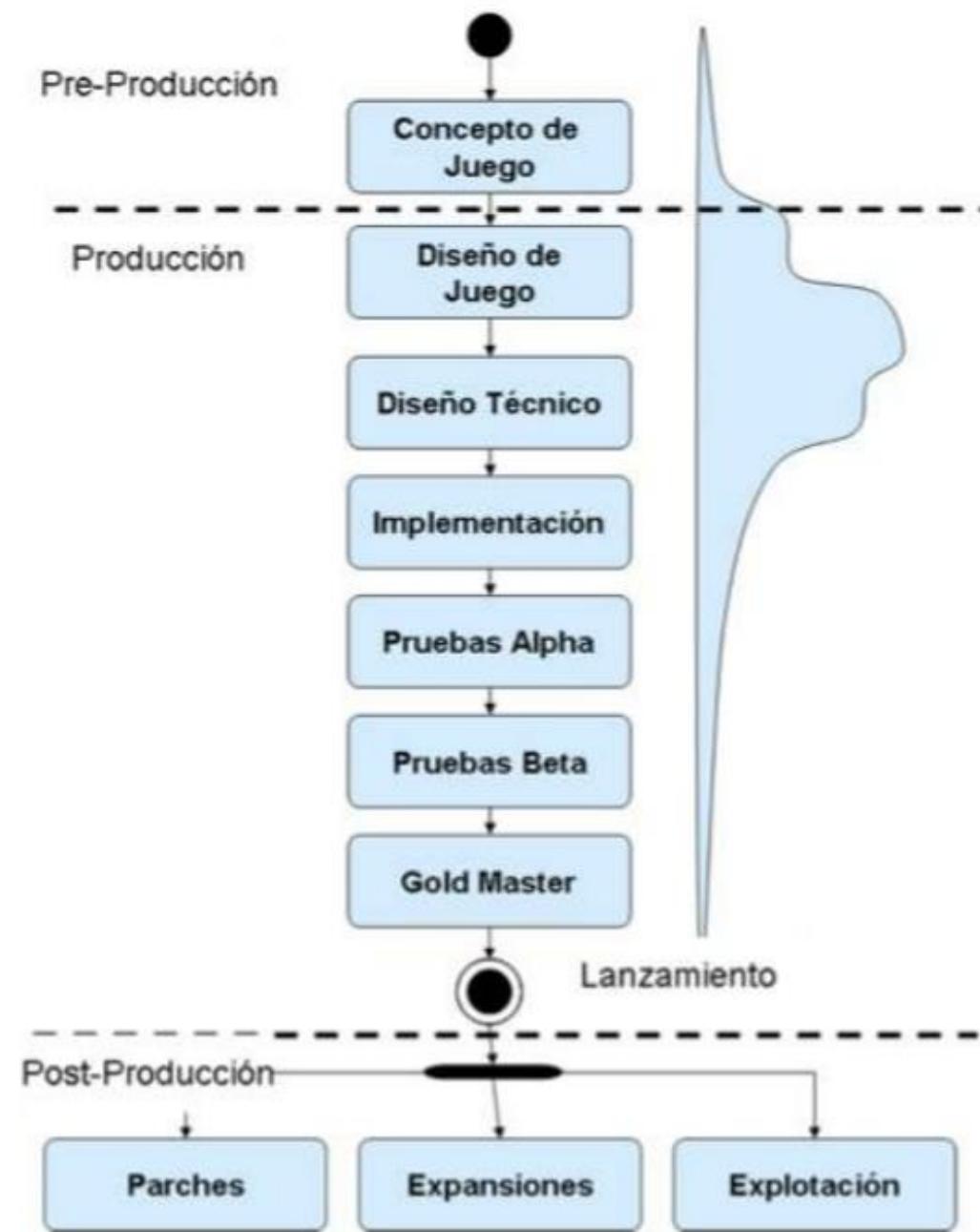
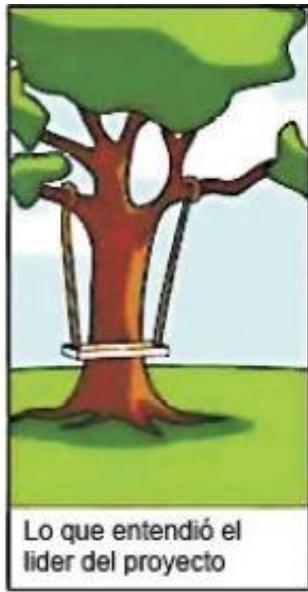


Figura 1.11: Organización de fases y etapas en la producción de un videojuego

La importancia de los requisitos



La solicitud del usuario



Lo que entendió el líder del proyecto



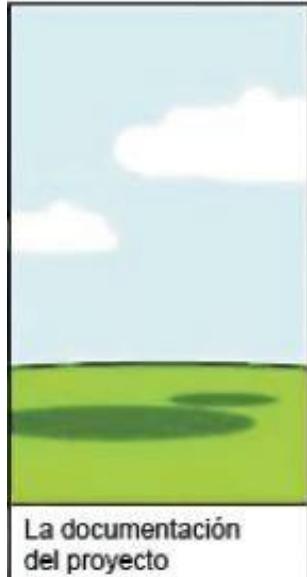
El diseño del analista de sistemas



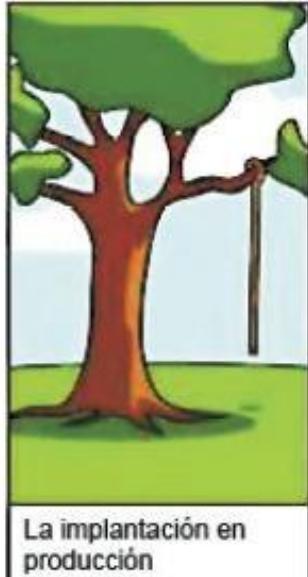
El enfoque del programador



La recomendación del consultor externo



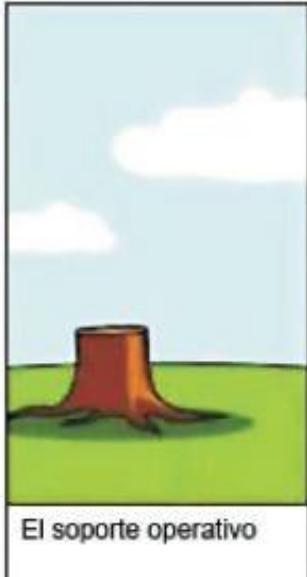
La documentación del proyecto



La implantación en producción



El presupuesto del proyecto



El soporte operativo



Lo que el usuario realmente necesitaba

Proceso de desarrollo Software. Ciclo de vida.

- El desarrollo de un software o de un conjunto de aplicaciones pasa por diferentes etapas desde que se produce la necesidad de crear un software hasta que se finaliza y está listo para ser usado por un usuario. Ese conjunto de etapas en el desarrollo del software responde al concepto de ciclo de vida del programa.
- El estándar **ISO/IEC 12207-1** define ciclo de vida del software como:
- “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 de software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso.”

Modelos de proceso de desarrollo software

- Existen varios modelos de ciclo de vida, entre los más importantes están:
 - Cascada
 - Modelos evolutivos:
 - Iterativo incremental
 - Espiral

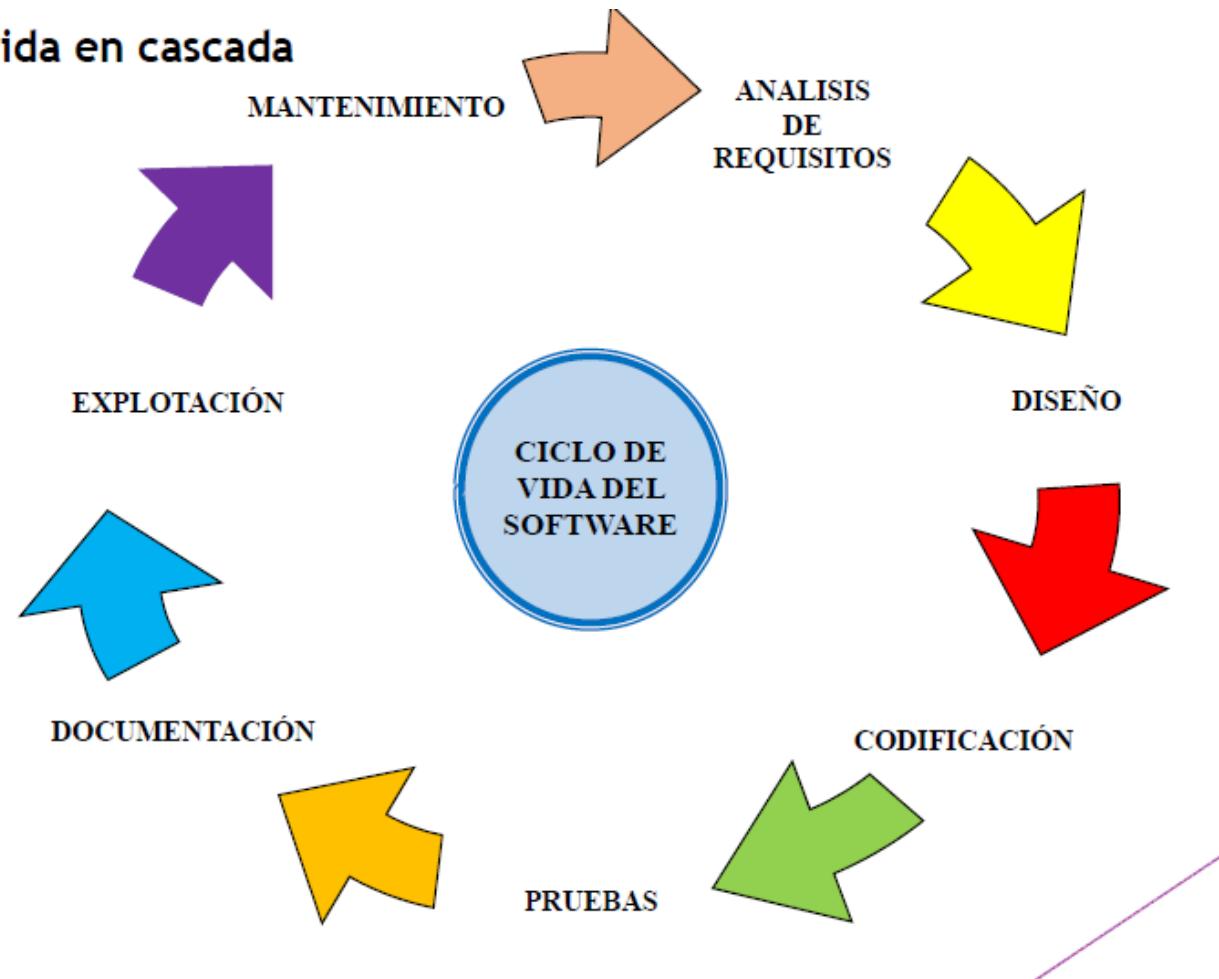
Modelos de proceso de desarrollo software

Ciclo de vida en cascada



Modelos de proceso de desarrollo software

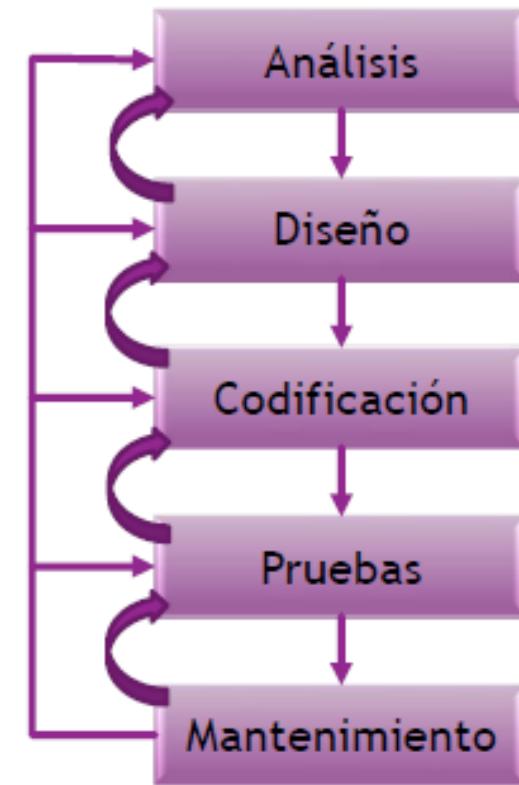
➤ Ciclo de vida en cascada



Modelos de proceso de desarrollo software

Ciclo de vida en cascada con Retroalimentación

Fallos detectados entre
fase y fase



Modelos de proceso de desarrollo software

Ciclo de vida en cascada

Ventajas

- Fácil de comprender, planificar y seguir
- Alta calidad del producto resultante
- Permite trabajar con personal poco cualificado

Inconvenientes

- Necesidad de tener todos los requisitos definidos desde el principio
- Es difícil volver atrás si se comenten errores en una etapa
- El producto no está disponible para su uso hasta que no está completamente terminado

Se recomienda cuando...

- El proyecto es similar a alguno que ya se ha realizado con éxito anteriormente
- Los requisitos son estable y están bien comprendidos
- Los clientes no necesitan versiones intermedias

Modelos de proceso de desarrollo software

Modelos evolutivos

- ▶ El software evoluciona con el tiempo
- ▶ Los requisitos del usuario y del producto pueden cambiar conforme se desarrolla el mismo
- ▶ Las empresas no pueden esperar a tener un producto totalmente terminado para lanzarlo al mercado
- ▶ Los modelos evolutivos permiten desarrollar versiones cada vez más completas hasta llegar al producto final deseado
- ▶ Modelos evolutivos más conocidos:
 - ▶ Iterativo incremental
 - ▶ Espiral



Modelos de proceso de desarrollo software

Modelo Iterativo Incremental

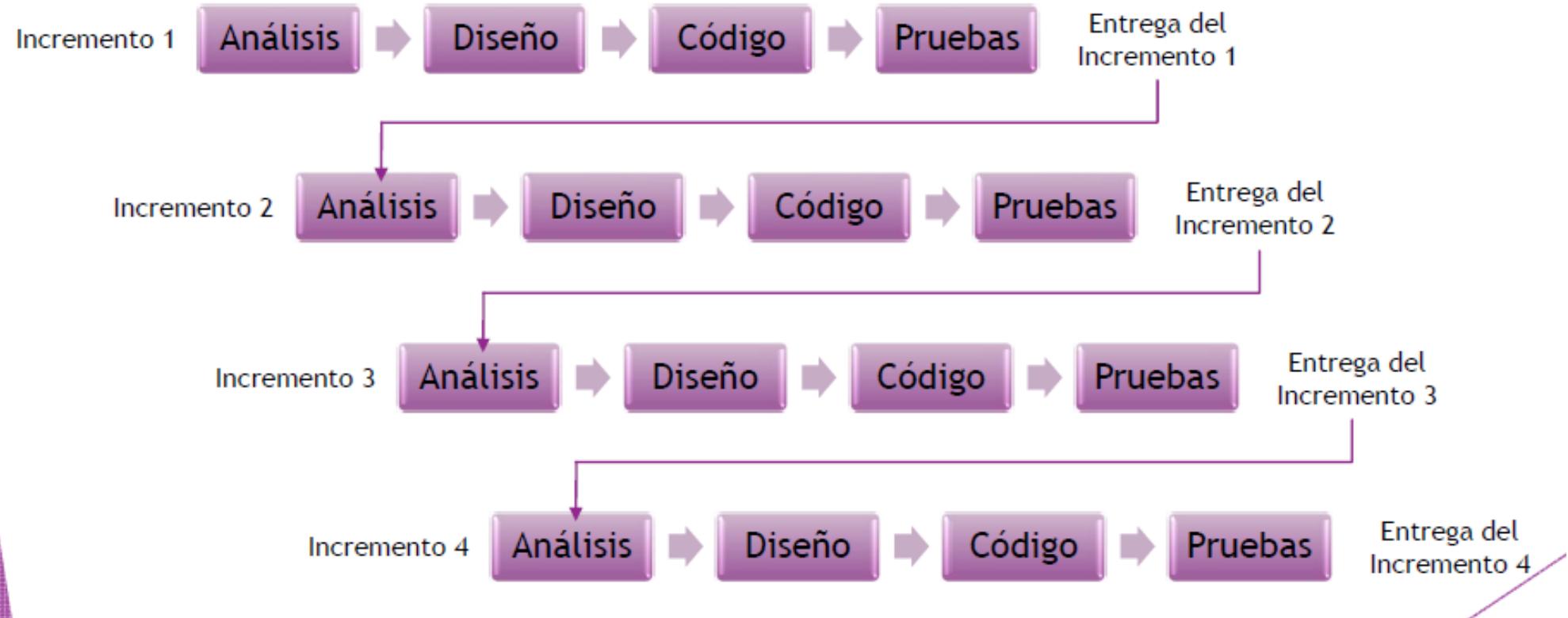
- ▶ Se basa en varios ciclos en cascada realimentados aplicados repetidamente
- ▶ Se entrega el software en partes pequeñas, utilizables: *incrementos*
- ▶ Cada *incremento* se construye sobre aquel que ya ha sido entregado

Ejemplo: un procesador de textos

En un primer incremento se desarrollan funciones básicas de gestión de archivos y de producción de documentos. En el segundo incremento se desarrollan funciones gramaticales y de corrección ortográfica. En el tercer incremento se desarrollan funciones avanzadas de paginación, y así sucesivamente.

Modelos de proceso de desarrollo software

Modelo Iterativo Incremental



Modelos de proceso de desarrollo software

Modelo Iterativo Incremental

Ventajas

- No se necesita conocer todos los requisitos al comienzo
- Permite la entrega temprana al cliente de partes operativas del software
- Las entregas facilitan la realimentación de los próximos entregables

Inconvenientes

- Es difícil estimar el esfuerzo y el coste final necesario
- Se tiene el riesgo de no acabar nunca
- No es recomendable para el desarrollo de ciertos sistemas como aquellos de alto índice de riesgo

Se recomienda cuando...

- Los requisitos o el diseño no están completamente definidos y es posible que haya grandes cambios
- Se están probando o introduciendo nuevas tecnologías.

Modelos de proceso de desarrollo software

Modelo en Espiral



Modelos de proceso de desarrollo software

Modelo en Espiral

Para cada ciclo, se siguen estas fases:

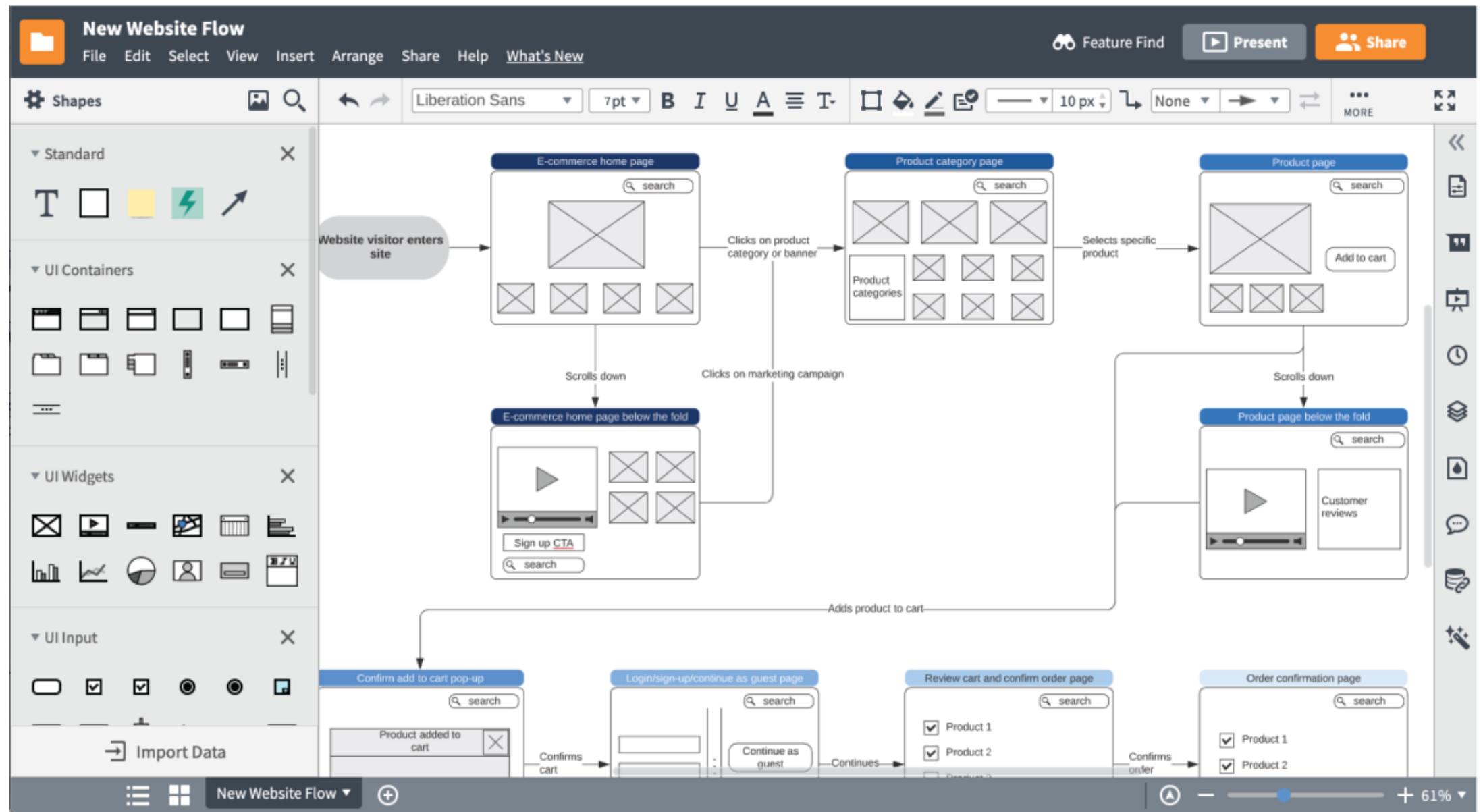
- ▶ **Determinar objetivos:** alternativas de diseños, restricciones impuestas, ...
- ▶ **Análisis de riesgos:** se identifican los riesgos involucrados y la manera de resolverlos (requisitos no comprendidos, mal diseño, errores de implementación)
- ▶ **Desarrollar y probar**
- ▶ **Planificación:** Revisar y evaluar todo lo que se ha hecho y decidir si se continua. Se planifican las fases del ciclo siguiente.

Modelos de proceso de desarrollo software

Modelo en Espiral

- ▶ Combina el *modelo en cascada* con el *modelo iterativo* de construcción de prototipos
- ▶ El proceso de desarrollo se representa como una espiral, donde en cada ciclo se desarrolla una parte del mismo.
- ▶ Se parece al modelo Iterativo Incremental con la diferencia de que en cada ciclo se tiene en cuenta el análisis de riesgos
- ▶ Durante los primeros ciclos la versión podrían ser maquetas en papel o modelos de pantallas (prototipos de interfaz)
- ▶ En el último ciclo se tendría un prototipo operacional que implementa algunas funciones del sistema.

Ejemplo de prototipo



Modelos de proceso de desarrollo software

Modelo en Espiral

Ventajas

- No requiere una definición completa de los requisitos para empezar a funcionar
- Análisis de riesgo en todas las etapas
- Reduce riesgos del proyecto
- Incorpora objetivos de calidad

Inconvenientes

- Es difícil evaluar los riesgos
- El coste del proyecto aumenta a medida que la espiral pasa por sucesivas iteraciones
- El éxito del proyecto depende en gran medida de la fase de análisis de riesgos

Se recomienda cuando...

- Proyectos de gran tamaño y que necesitan constantes cambios
- Proyectos donde sea importante el factor riesgo
- Muy utilizado para el desarrollo de sistemas orientados a objetos

Herramientas CASE

- Las herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadora) son diversas aplicaciones informáticas o programas informáticos destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero...

Herramientas CASE - Objetivos

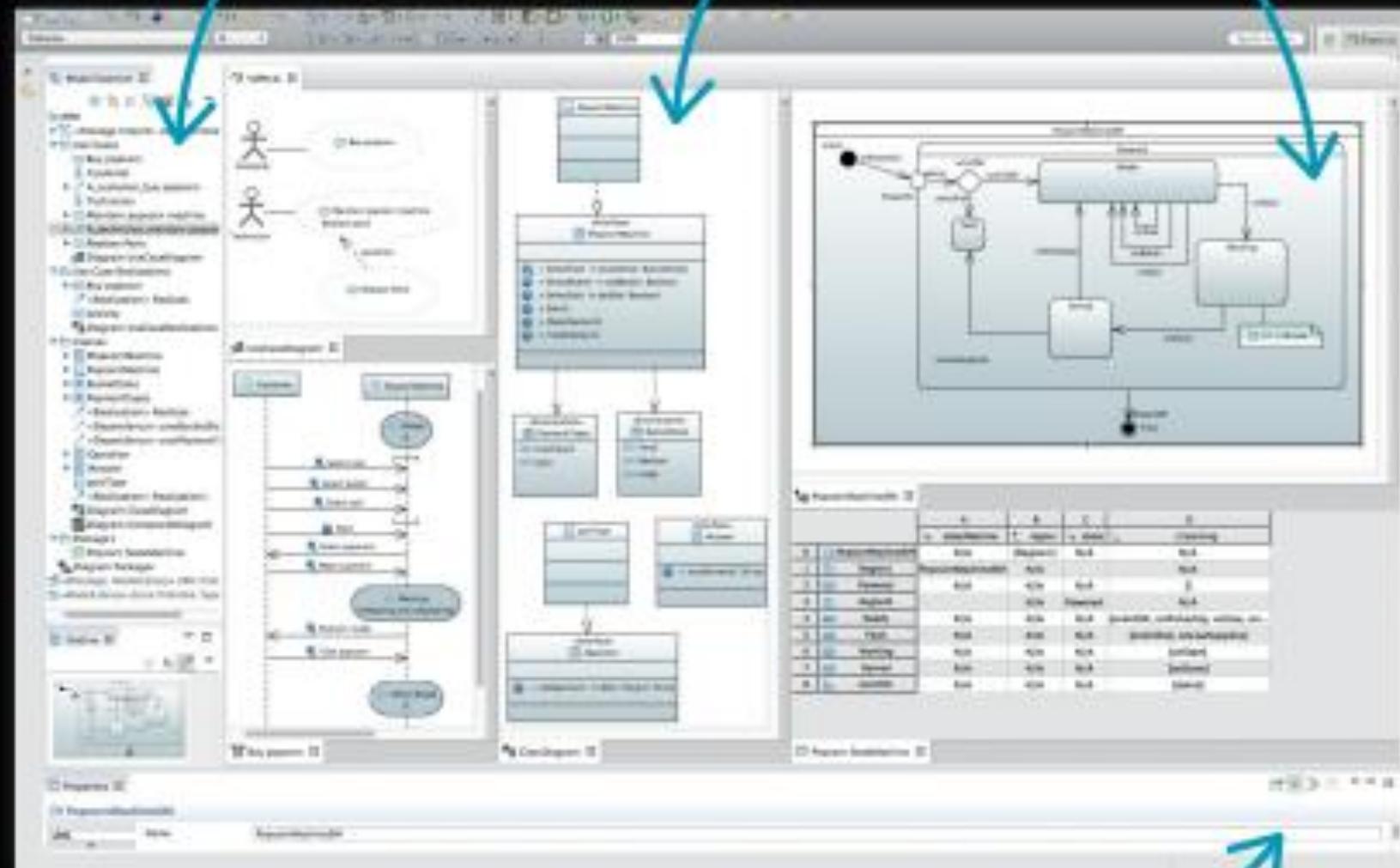
- Mejorar la productividad del software.
- Aumentar la calidad del software.
- Reducir el tiempo y costo de desarrollo y mantenimiento de los sistemas informáticos.
- Mejorar la planificación de un proyecto.
- Aumentar la biblioteca de conocimiento informático de una empresa ayudando a la búsqueda de soluciones para los requisitos.
- Automatizar el desarrollo del software, la documentación, la generación de código, las pruebas de errores y la gestión del proyecto.
- Ayuda a la reutilización del software, portabilidad y estandarización de la documentación.
- Gestión global en todas las fases de desarrollo de software con una misma herramienta.
- Facilitar el uso de las distintas metodologías propias de la ingeniería del software.

Herramientas CASE - Ejemplos

- Eclipse Papyrus
- MICROSOFT VISIO.
- LUCIDA CHART
- SMART DRAW. (ONLINE)
- ClickCharts
- DIA (aplicación gratuita multisistema)
- METRICA 3. Instrumento que da soporte al ciclo de vida del SOFTWARE, del ministerio, promovida por el consejo superior de informática (CSI).
- Visual Paradigm
- GANTT PROYECT

UML model explorer

Editors for all UML diagrams



Form-based properties edition

Fases del desarrollo de una aplicación

- Análisis → ¿QUÉ?
- Diseño → ¿CÓMO?
- Codificación → CONSTRUCCIÓN
- Pruebas → ¿FUNCIONA? ¿HACE LO QUE YO QUERÍA?
- **Documentación.**
- Explotación.
- Mantenimiento.
- Cada etapa tiene como entrada uno o varios documentos procedentes de las etapas anteriores y produce otros documentos de salida, por lo que una tarea importante a realizar en cada etapa es la documentación.

Análisis

- En esta etapa se debe entender y comprender de forma detallada el problema que se va a resolver y, una vez comprendido, darle solución.
- Se analizan y especifican los requisitos o capacidades que el sistema debe tener porque el cliente así lo ha pedido.

Análisis



Análisis

- Para facilitar la comunicación con el cliente, se utilizan varias técnicas:
 - **Entrevistas**
 - **Desarrollo conjunto de aplicaciones (JAD)**: Entrevista muy estructurada apoyada en la dinámica de grupos.
 - **Planificación conjunta de requisitos (JRP)**: subconjunto de JAD dirigida a la alta dirección (requisitos de alto nivel o estratégicos)
 - **Brainstorming**: Reuniones de grupo cuyo objetivo es generar ideas desde diferentes puntos de vista para la resolución de un problema.
 - Uso de **prototipos**.
 - **Casos de uso**: Requisitos funcionales del sistema, describen qué hace el sistema no cómo lo hace

Análisis

- Se especifican dos tipos de requisitos
- **Requisitos funcionales:** Describen con detalle la función que realiza el sistema, cómo reacciona ante determinadas entradas, cómo se comporta en situaciones particulares,...
- **Requisitos no funcionales:** Tratan sobre las características del sistema, como puede ser la fiabilidad, mantenibilidad, sistema operativo, plataforma hardware, restricciones, limitaciones, ...

Análisis

- Para representar los requisitos se utilizan diferentes técnicas:
 - Diagramas de flujo de datos, DFD.
 - Diagramas de flujo de control, DFC.
 - Diagramas de transición de estados, DTE.
 - Diagrama Entidad/Relación, DER.
 - Diccionario de datos, DD.
- Todo lo realizado en esta fase debe quedar reflejado en el Documento de Requisitos del Sistema (DRS).
- La estructura del documento DRS queda reflejada en la propuesta por el IEEE en la versión del estándar 830 [IEEE, 1998]

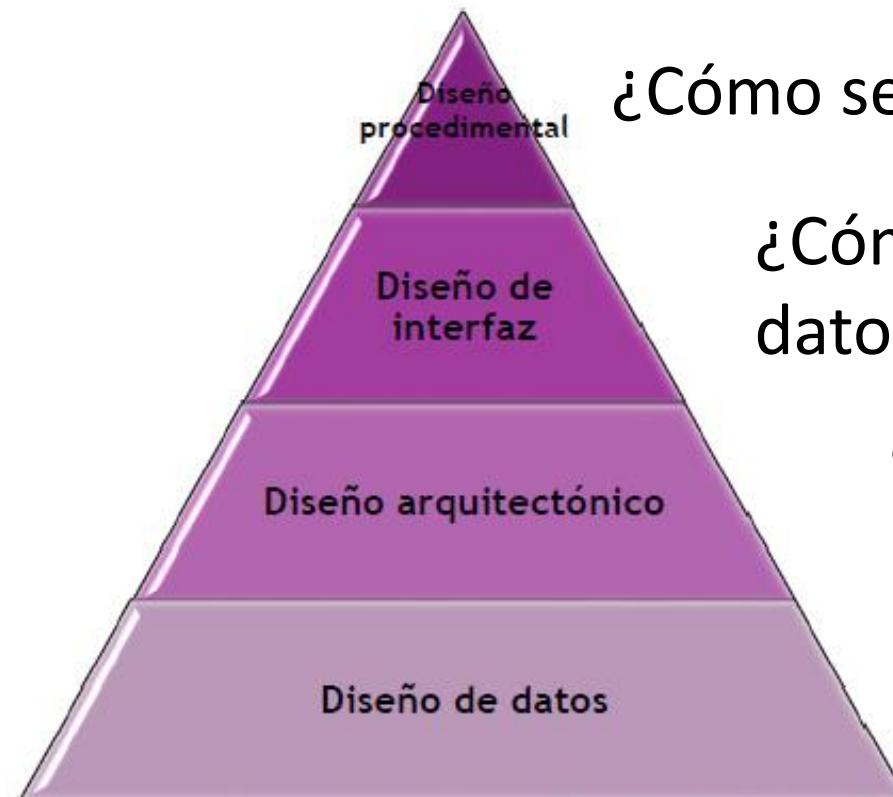
Diseño

- Una vez identificados los requisitos, es necesario componer la forma en que se solucionará el problema.
- Hay dos tipos de diseño:
 - **Diseño estructurado:** basado en el flujo de datos a través del sistema.
 - **Diseño orientado a objetos:** el sistema se entiende como un conjunto de objetos que tienen propiedades y comportamientos, además de eventos que activan operaciones que modifican el estado de los objetos.

Diseño

Diseño estructurado

Produce un modelo de diseño con 4 elementos:



¿Cómo se procesan los datos?

¿Cómo se comunican los datos al usuario?

¿Cómo se comunican entre las partes del sistema?

¿Qué datos almacena la aplicación?

Diseño

Diseño estructurado

- ▶ **Diseño de datos:** está basado en los datos y las relaciones definidos en el diagrama entidad/relación y en la descripción detallada de los datos
- ▶ **Diseño arquitectónico:** se centra en la representación de los componentes del software, sus propiedades e interacciones. Partiendo de los DFD se establece la estructura modular del software que se desarrolla.
- ▶ **Diseño de la interfaz:** Describe como se comunica el software con las personas que lo utilizan.
- ▶ **Diseño procedural (a nivel de componentes):** Transforma los elementos estructurales de la arquitectura del software en una descripción procedural de los componentes del software. Para llevar a cabo este diseño se utilizan representaciones gráficas mediante diagramas de flujo, diagramas de cajas, tablas de decisión, pseudocódigo, etc.

Diseño

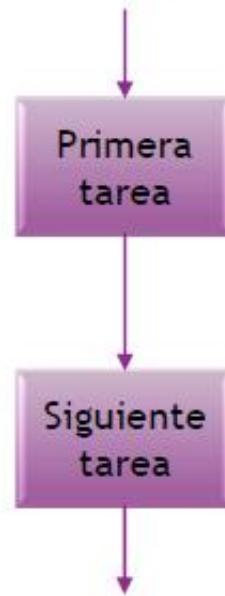
Diseño a nivel de componentes

Los fundamentos del diseño a nivel de componentes se establecieron cuando se propuso el uso de un conjunto de construcciones lógicas con las que podía formarse cualquier programa, fundamentales para la programación estructurada:

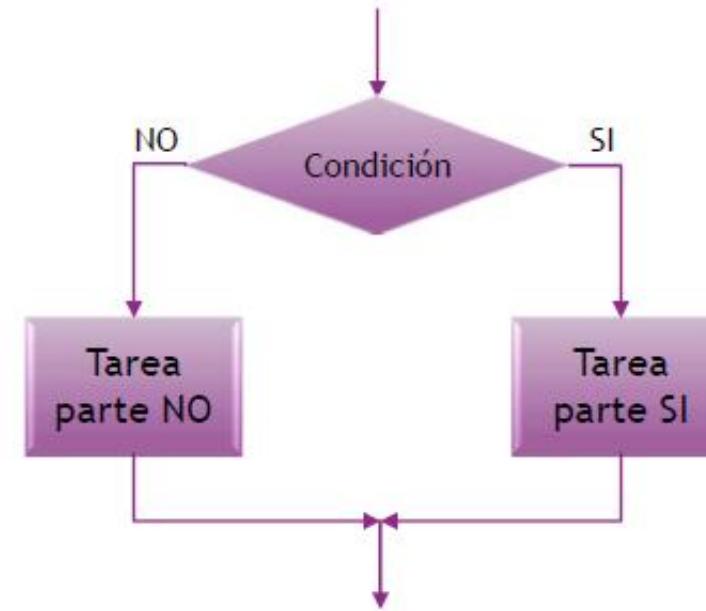
- ▶ Secuencial
- ▶ Condicional
- ▶ Repetitiva

Diseño

Diseño a nivel de componentes



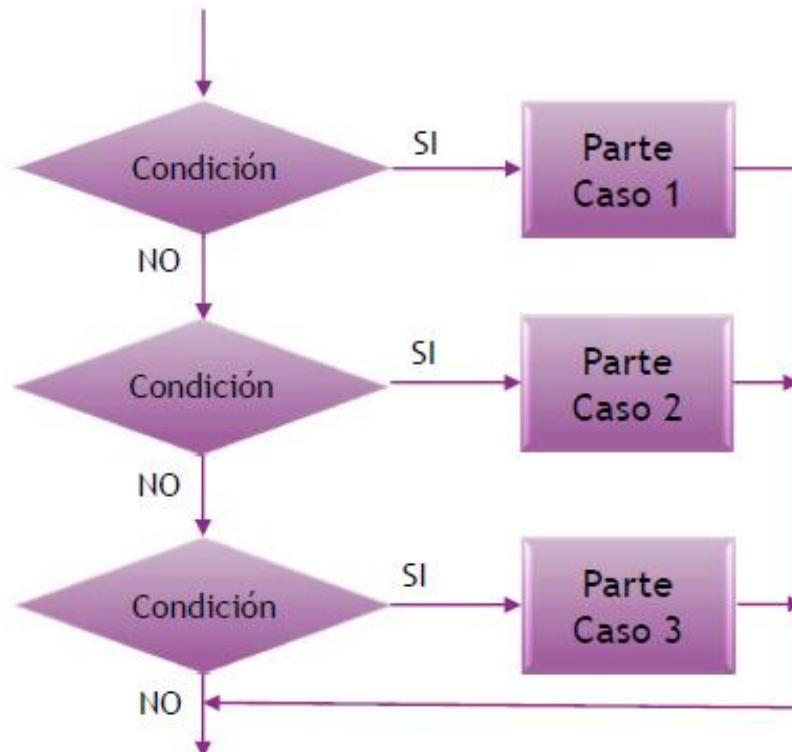
Construcción
secuencial



Construcción
condicional

Diseño

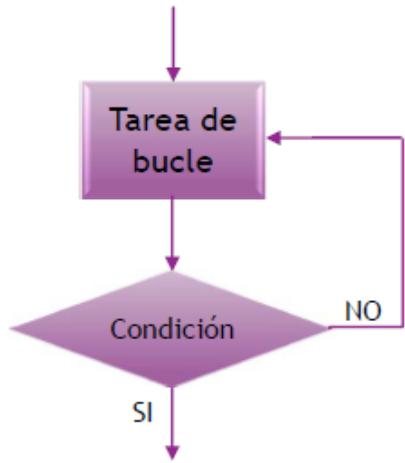
Diseño a nivel de componentes



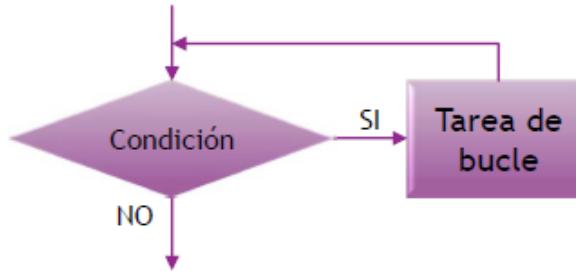
Construcción selección múltiple

Diseño

Diseño a nivel de componentes



Repetir-hasta



Hacer-mientras

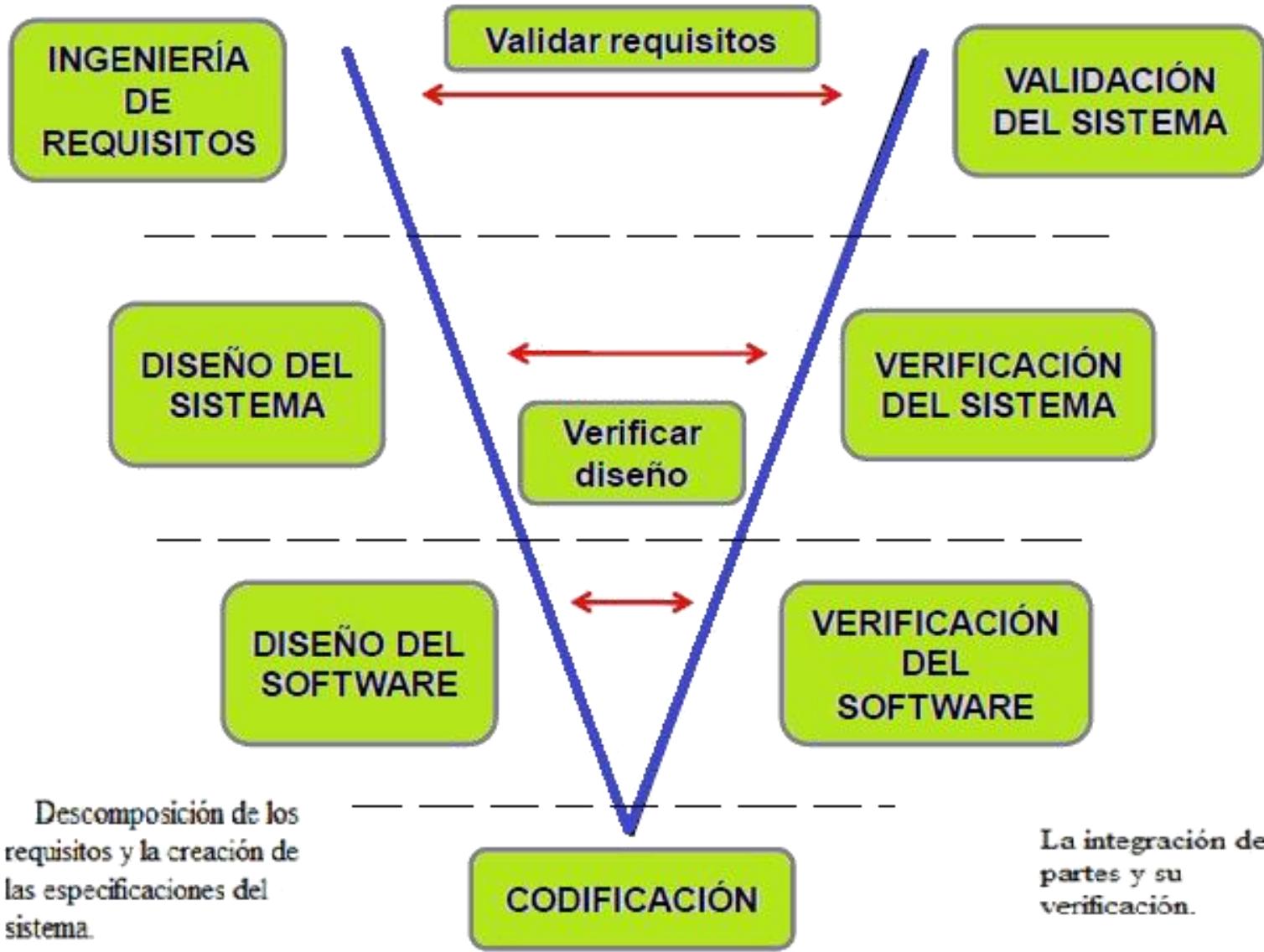
Construcciones
repetitivas

Codificación.

- Proceso de construcción de los algoritmos utilizando lenguajes de programación entornos de desarrollo y la documentación elaborada en las fases anteriores.
- Cuanto más elaborado y exhaustivo hayan sido el proceso de diseño y análisis, más fluida será la fase de codificación ya que el número de cambios y la magnitud de éstos a implementar será de menor impacto para el desarrollo del proyecto.

Pruebas.

- Conjunto de procesos con el fin de comprobar y asegurar que el software que se desarrolla está acorde a su especificación y cumple las necesidades de los clientes
- Cumple dos objetivos:
 - Verificación:
 - ¿Estamos construyendo el producto correctamente?
 - Se comprueba que el software cumple los requisitos funcionales y no funcionales de su especificación.
 - Validación:
 - ¿Estamos construyendo el producto correcto?
 - Comprueba que el software cumple las expectativas que el cliente espera.



Descomposición de los requisitos y la creación de las especificaciones del sistema.

La integración de partes y su verificación.

Documentación

- Principalmente se crean 2 tipos de documentación.
 - **Documentación técnica:** destinada al equipo de desarrollo. Explica todas las decisiones técnicas que se han tomado para que un miembro nuevo del proyecto pueda adaptarse rápidamente al desarrollo/mantenimiento de la aplicación.
 - **Documentación de usuario:** Manual de usuario que debe ofrecer una visión completa de como interactuar con el sistema desarrollado de una manera sencilla e intuitiva.

Explotación.

- Una vez se ha desarrollado y testeado la aplicación debe, se pone en producción.
- Venta y/o distribución.
 - Se puede distribuir en un formato cloud accesible desde internet.
 - Ejemplo: aplicaciones web
 - Soporte físico de almacenamiento: discos ópticos, tarjetas de memoria, etc...

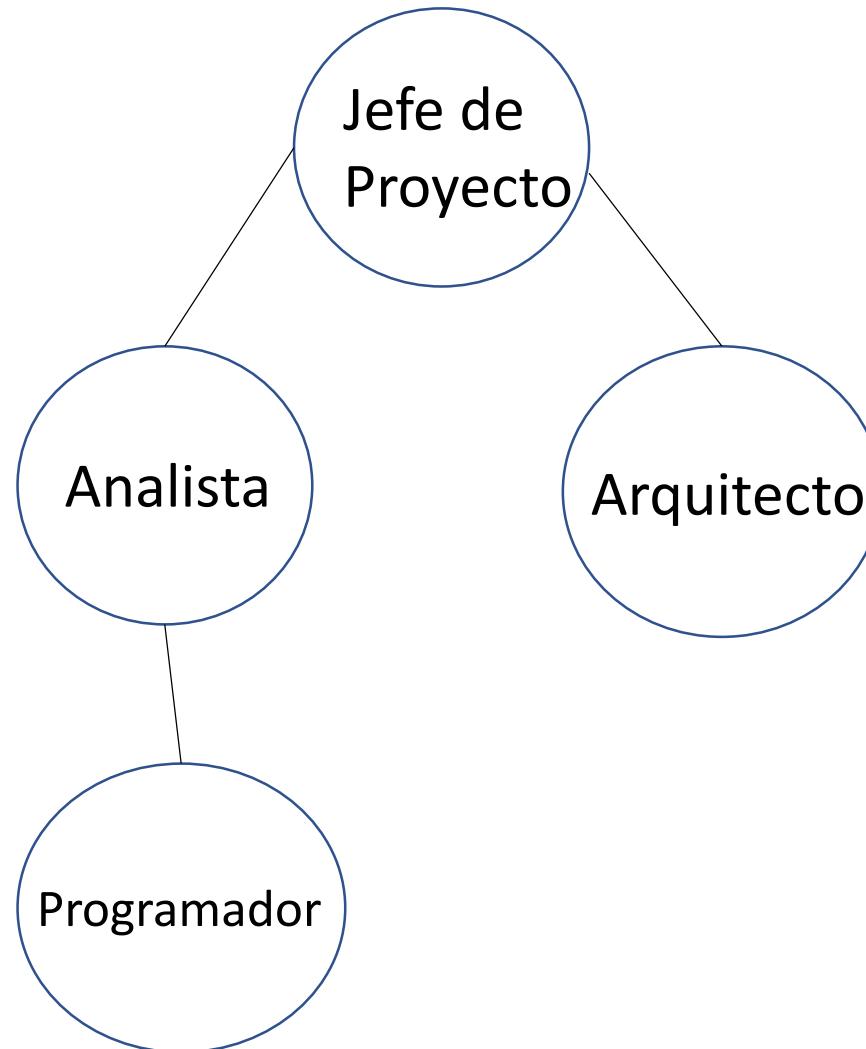
Mantenimiento.

- Deben ofrecerse **actualizaciones** de soporte que corrijan posibles fallos (bugs) y/o amplíen el funcionamiento de la aplicación.
 - Ejemplo: actualizaciones de Windows, DLCs de videojuegos, etc...
- Dentro de un proyecto, en la fase de mantenimiento a las actualizaciones suelen llamarse “evolutivos” pues hacen evolucionar el funcionamiento de la aplicación.
 - Ejemplo: Versiones de Microsoft Office 20010, 2013, 2016, 365, etc...

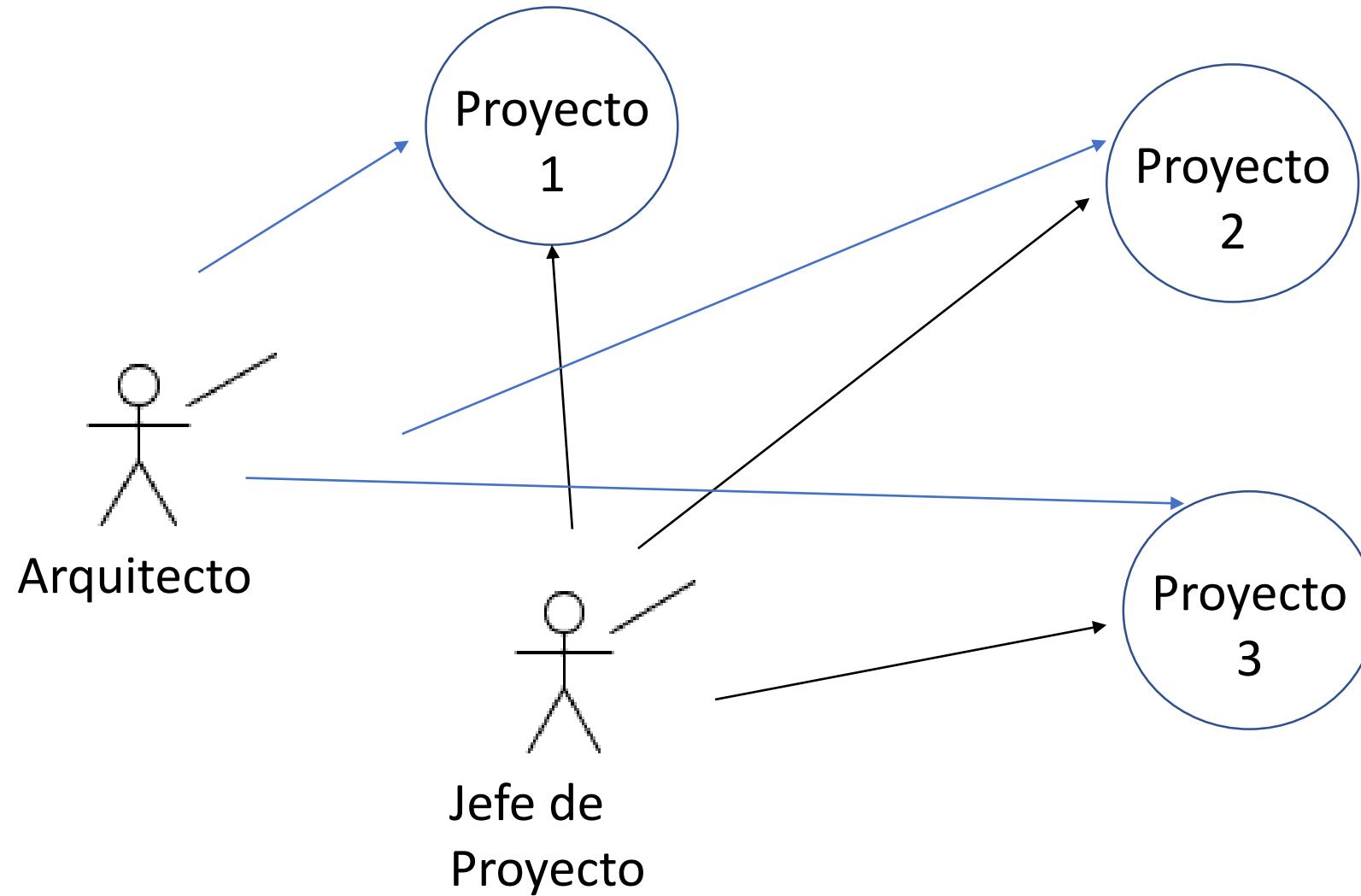
Roles dentro de un proyecto software

- **Programador:** Se encarga exclusivamente de desarrollar el código fuente de la aplicación siguiendo las indicaciones que el analista, diseñador y arquitecto le proporcionan.
- **Analista programador:** Rol intermedio entre el analista y el programador, en la práctica suele ser un programador con amplia experiencia.
- **Analista orgánico:** Responsable de captar los requisitos del sistema, es el intermediario entre el equipo de desarrollo y el cliente.
- **Arquitecto/Diseñador:** Evolución del analista, define las tecnologías que se van a utilizar en el proyecto. Es el encargado de planificar la arquitectura software y hardware en la que se va a implementar la aplicación.
- **Jefe de proyecto:** Encargado de firmar el presupuesto del proyecto y planificar la temporalización de este en forma de fechas de entrega. No tiene porqué tener conocimientos de informática, debe ser capaz de gestionar recursos materiales y humanos.

Roles dentro de un proyecto software



Roles dentro de un proyecto software



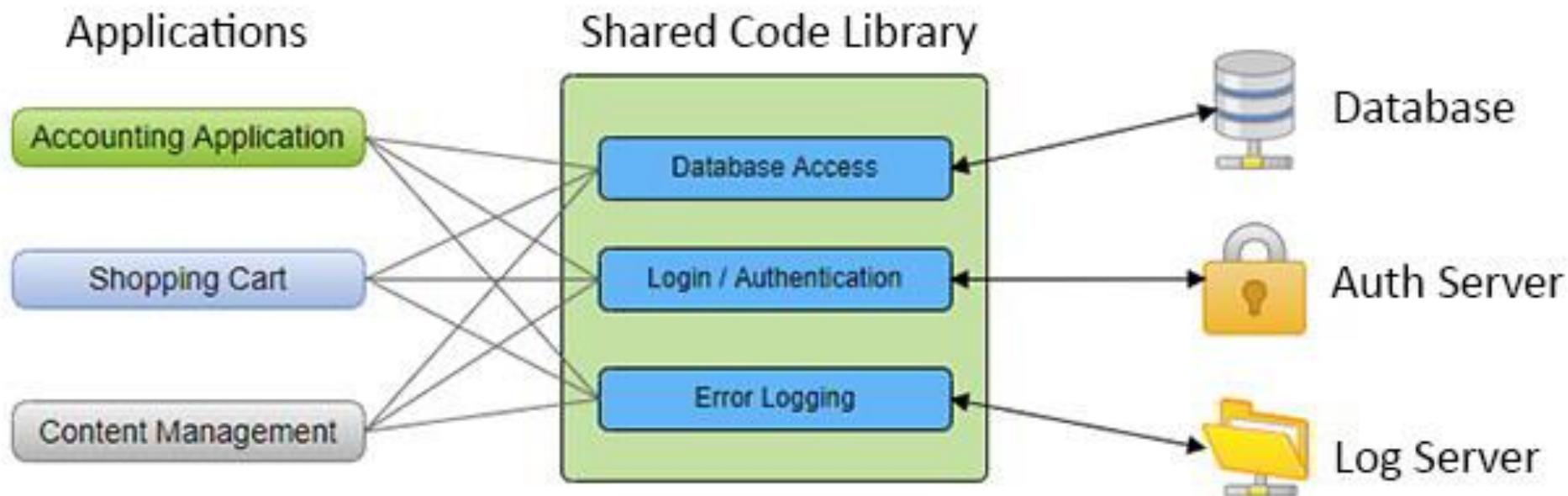
Errores en el proceso de desarrollo software

- Software mal estructurado.
 - Proveniente de una mala captación de requisitos o de una concepción del sistema errónea.
- Trabajo en equipo donde falla la comunicación. No se usan gestores de tareas.
 - La comunicación entre el equipo es fluida cuando cada vez que surge un conflicto en el desarrollo se resuelve en equipo.
- Testing y/o Integración no continuos.
 - Al no probarse la funcionalidad desarrollada con cierta frecuencia es posible que se vayan arrastrando fallos de una parte a otra del sistema conforme éste se va construyendo.

Importancia de la reutilización de código

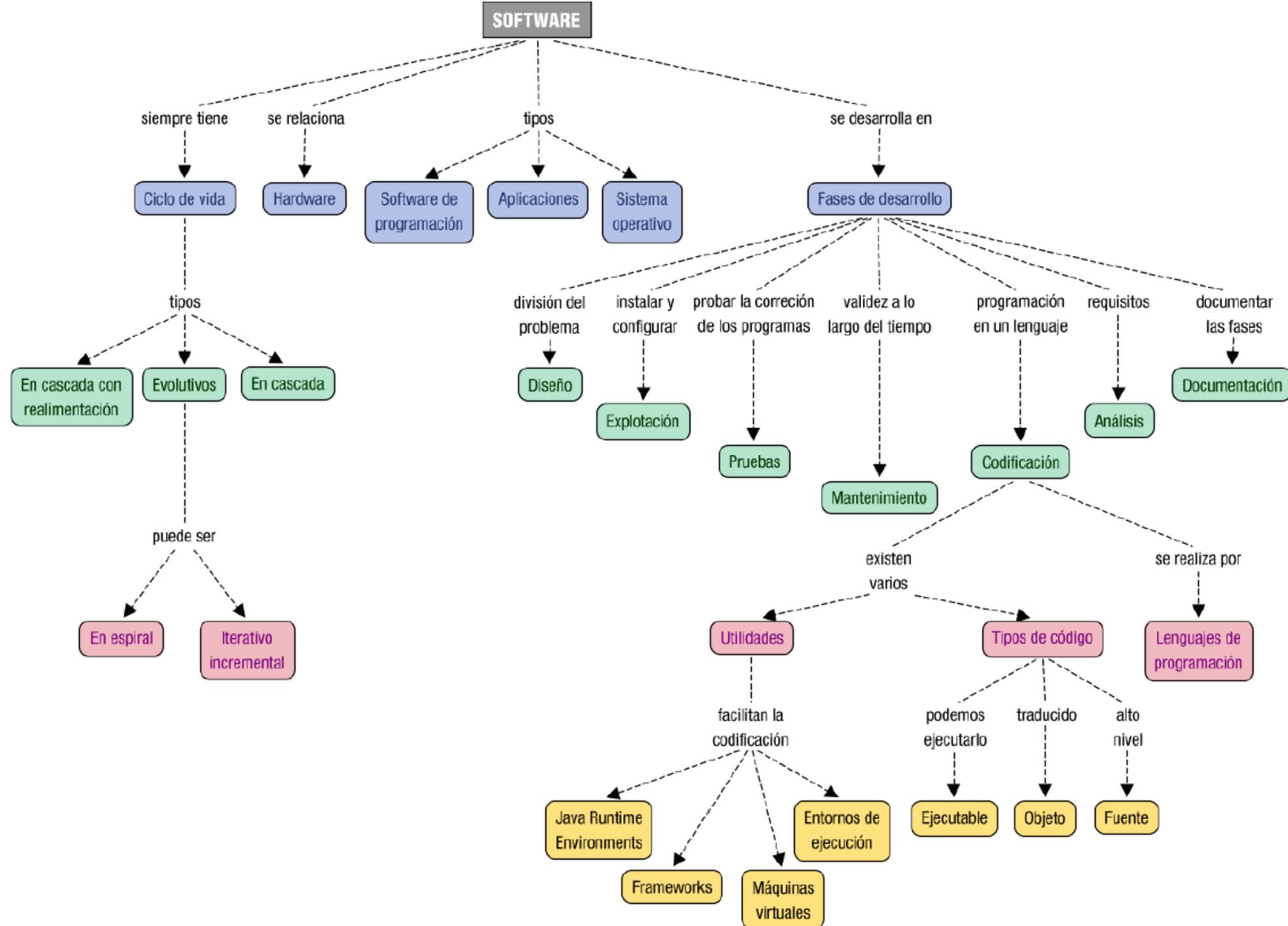
- Se refiere al comportamiento y a las técnicas que garantizan que una parte o la totalidad de un programa informático existente se pueda emplear en la construcción de otro programa.
- De esta forma se aprovecha el trabajo anterior, se economiza tiempo, y se reduce la redundancia.
- La manera más fácil de reutilizar código es copiarlo total o parcialmente desde el programa antiguo al programa en desarrollo.
- Es laborioso mantener múltiples copias del mismo código, por lo que en general se elimina la redundancia dejando el código reusable en un único lugar, y llamándolo desde los diferentes programas.
- Ejemplo: Bibliotecas de software.

Importancia de la reutilización de código



Actividades

- ¿Qué diferencias existen entre el código fuente y el código máquina?
- ¿Qué lenguaje de programación utilizarías para?:
 - Crear una aplicación móvil
 - Crear una aplicación web
 - Crear un sistema operativo
 - Crear un videojuego.



Referencias

- Libro “*Entornos de Desarrollo*”, 2018. Juan Carlos Moreno Pérez. Editorial: Síntesis.
- [https://www ctr unican es asignaturas Ingenieria Software 4 F/ Doc/M7_09_VerificacionValidacion-2011.pdf](https://www ctr unican es asignaturas Ingenieria Software 4 F Doc M7_09_VerificacionValidacion-2011 pdf)