

# **Tema 1: Desarrollo de software**



Desarrollo de Aplicaciones Multiplataforma

# Índice

1. Concepto de software. Tipos
2. Relación hardware-software.
3. Desarrollo de software.
  - 3.1. Ciclos de vida del software.
4. Fases en el desarrollo del software.
  - 4.1. Análisis
  - 4.2. Diseño
  - 4.3. Codificación. Tipos de Código
  - 4.4. Pruebas
  - 4.5. Documentación
  - 4.6. Explotación
  - 4.7. Mantenimiento

# Índice

## 5. Concepto de programa.

### 5.1. Programa y componentes del sistema informático.

## 6. Lenguajes de programación.

### 6.1. Clasificación y características.

## 7. Máquinas virtuales

### 7.1. La máquina virtual de Java.

## 8. Herramientas de programación

# Concepto de software

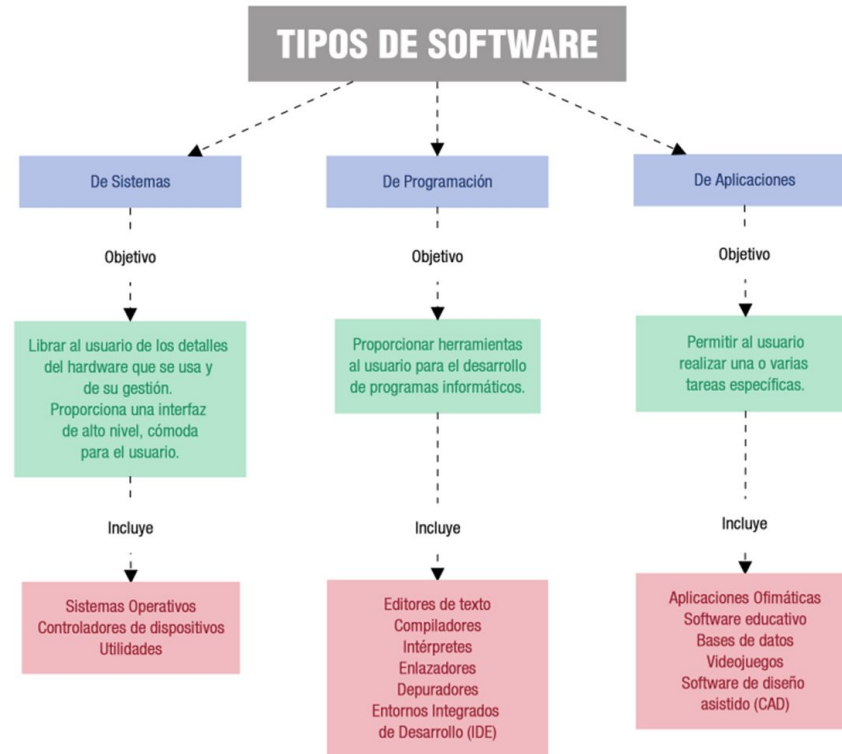
Un ordenador tiene dos partes diferenciadas: **hardware y software**.

Software es el **conjunto de programas informáticos** que actúan sobre el Hardware para ejecutar lo que el usuario desee.

Según su función se distinguen **tres tipos** de software:

- **Sistemas: software base** que ha de estar instalado y configurado en el ordenador para que las aplicaciones puedan ejecutarse y funcionar. Ej. sistemas operativos, controladores, drivers.
- **Software de programación: conjunto de herramientas** que nos permiten desarrollar programas informáticos. Editores, compiladores, depuradores etc. Sin embargo lo más habitual es usar un tipo de software que integre todas estas herramientas -> **IDE (ENTORNOS DE DESARROLLO INTEGRADOS)**
- **Aplicaciones informáticas:** son un **conjunto de programas** que tienen una finalidad más o menos concreta. Son ejemplos de aplicaciones: un procesador de textos, una hoja de cálculo, reproductor de música, un videojuego, etc.

# Concepto de software



# Concepto de software

- En este tema, nuestro interés se centra en las **aplicaciones informáticas**: cómo se desarrollan y cuáles son las fases por las que necesariamente han de pasar.
- A lo largo de esta primera unidad vas a aprender los conceptos fundamentales de software y las fases del llamado **ciclo de vida** de una aplicación informática.
- También aprenderás a distinguir los diferentes lenguajes de programación y los procesos que ocurren hasta que el programa funciona y realiza la acción deseada.



# Relación hardware-software

**Hardware:** conjunto de dispositivos físicos que conforman un ordenador.

Existe una relación indisoluble entre éste y el software, ya que necesitan estar instalados y configurados correctamente para que el equipo funcione. El software se ejecutará sobre los dispositivos físicos.



# Relación hardware-software

Esta relación software-hardware se puede ver desde dos puntos de vista:

- Desde el punto de vista del **sistema operativo (SO)**
  - **El SO**: 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 memoria RAM, tratamiento de interrupciones, 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).



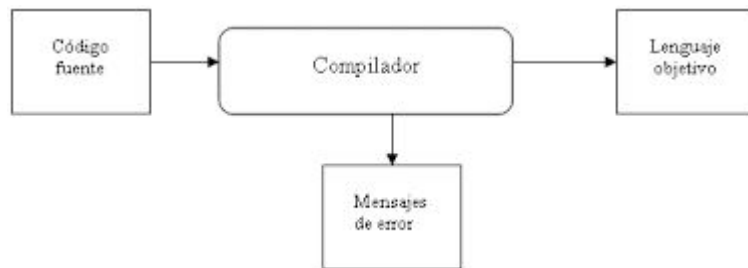
# Relación hardware-software

- Desde el punto de vista de las **aplicaciones**:
  - **Aplicación: conjunto de programas**, escritos en algún lenguaje de programación que el hardware del equipo debe interpretar y ejecutar.
  - Hay multitud de lenguajes de programación diferentes, todos ellos con algo en común, estar escritos con sentencias de un idioma que el ser humano puede aprender y usar fácilmente.
  - Por otra parte, el hardware de un ordenador sólo es capaz de interpretar señales eléctricas que, en informática, se traducen en secuencias de 0 y 1 (**código binario**).

# Relación hardware-software

¿Cómo será capaz el ordenador de "entender" algo escrito en un lenguaje que no es el suyo?

Como veremos a lo largo de esta unidad, tendrá que pasar algo (un proceso de traducción de código) para que el ordenador ejecute las instrucciones escritas en un lenguaje de programación.



# Desarrollo de software

- **Desarrollo del software:** **proceso** que ocurre desde que se concibe una idea hasta que un programa está implementado en el ordenador y funcionando.
- **Ciclo de vida del software:** describe el desarrollo de software desde la fase inicial hasta la fase final, proponiendo etapas que sirven como referencia para realizar este proceso.

Las fases que conforman el ciclo de vida son:

- Análisis
- Diseño
- Desarrollo o codificación
- Pruebas
- Documentación
- Explotación
- Mantenimiento

# Fases de desarrollo del software

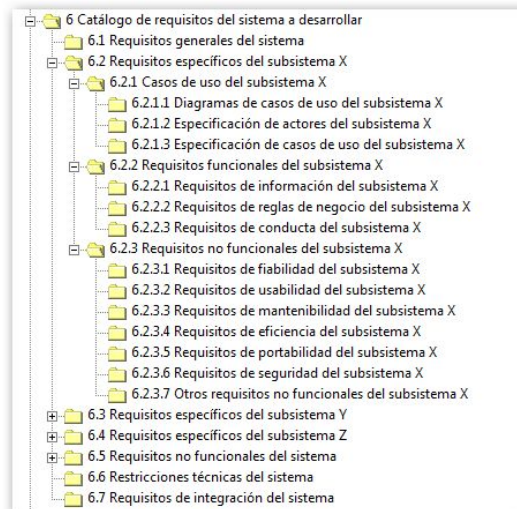
## Análisis

Es la **primera etapa** del proyecto, la que depende del **analista**. En ella se especifican los requisitos funcionales y no funcionales del sistema.

Es fundamental la comunicación entre el analista y el cliente para la aplicación cumpla con sus expectativas. Todo lo recogido en esta fase queda reflejado en el documento **Especificación de Requisitos del Software (ERS)**, que no puede contener ambigüedades, debe ser completo, consistente, fácil de verificar y de utilizar, y fácil de identificar el origen y las consecuencias de estos requisitos.

Existen **varias metodologías** de análisis:

- Entrevistas
- Dinámicas de grupo
- Brainstorming (lluvia de ideas)
- Prototipos (versiones iniciales del sistema)



# Fases de desarrollo del software

## Análisis

- Los **requisitos funcionales (RF)** detallan:
  - qué funciones tendrá que realizar la aplicación
  - qué respuesta dará la aplicación ante cualquier tipo de entrada
  - cómo se comportará la aplicación en situaciones inesperadas
- Los **requisitos no funcionales (RNF)** tratan características del sistema
  - tiempos de respuesta del programa
  - legislación que se aplica
  - fiabilidad
  - mantenibilidad
  - sistema operativo necesario
  - restricciones

Requisitos funcionales	Requisitos no funcionales
El usuario puede agregar un nuevo contacto	La aplicación debe funcionar en sistemas operativos Linux y Windows
El usuario puede ver una lista con todos los contactos	El tiempo de respuesta a consultas, altas, bajas y modificaciones ha de ser inferior a 5 segundos
A partir de la lista de contactos el usuario puede acceder a un contacto	Utilizar un sistema gestor de base de datos para almacenar los datos
El usuario puede eliminar un contacto o varios de la lista	Utilizar un lenguaje multiplataforma para el desarrollo de la aplicación
El usuario puede modificar los datos de un contacto seleccionado de la lista	La interfaz de usuario es a través de ventanas debe ser intuitiva y fácil de manejar
El usuario puede seleccionar determinados contactos	El manejo de la aplicación se realizará con el teclado y el ratón
El usuario puede imprimir la lista de contactos	Espacio libre en disco, mínimo:1GB Mínima cantidad de memoria : 2GB

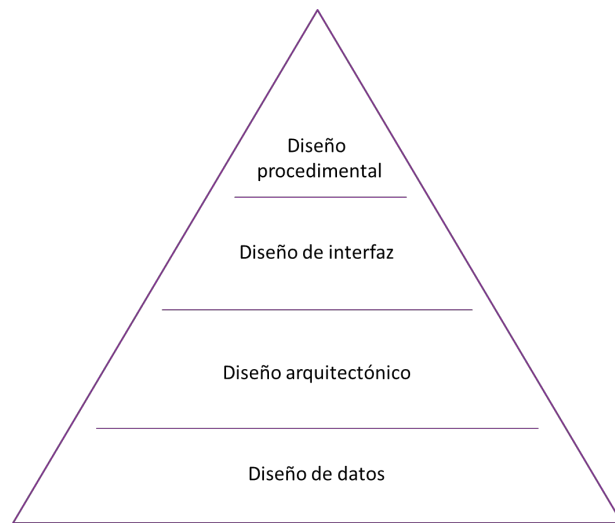
# Fases de desarrollo del software

## Diseño

Ya sabemos que hay que hacer, **¿cómo lo hacemos?** Tenemos que traducir los requisitos en una representación del software a desarrollar.

Tenemos dos opciones de diseño:

- **Estructurado:** basado en el flujo de datos. Produce un modelo con cuatro elementos (figura derecha)
- **Orientado a objetos:** el sistema entendido como un conjunto de objetos que tienen prioridades y comportamientos, existen eventos que activan operaciones, se modifica el estado de dichos objetos...



# Fases de desarrollo del software

## Diseño

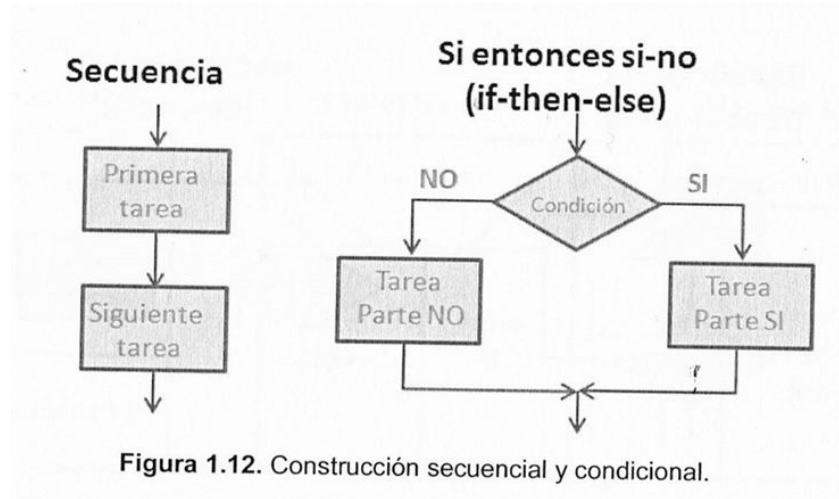
- **Diseño de datos:** Está basado en los datos y las relaciones, definidos en el diagrama entidad-relación y en la descripción detallada de los datos.
- **Diseño arquitectónico:** Representa la estructura de los componentes del software, sus propiedades e interacciones.
- **Diseño de la interfaz:** Describe cómo se comunica el software consigo mismo, con los sistemas que operan con él y con las personas que lo utilizan. Resultado: creación de formatos de pantalla.
- **Diseño a nivel de componentes o diseño procedimental:** El diseño a nivel de componentes se presenta a menudo después que se ha terminado la primera iteración del diseño arquitectónico y el objetivo de esta fase es traducir el diseño en software operacional. El diseño a nivel de componentes define las estructuras de datos, los algoritmos, las características de la interfaz y los mecanismos de comunicación asignados a cada componente de software. Esta fase permite revisar si los detalles de diseño son correctos y consistentes con las representaciones iniciales de diseño.

# Fases de desarrollo del software

## Diseño

### Construcciones lógicas para diseñar cualquier programa:

- **Secuencial:** Implementa los pasos del proceso esenciales para la especificación de cualquier algoritmo.
- **Condicional:** Permite seleccionar un proceso u otro a partir de la evaluación de una condición lógica. Si se cumple, realiza la tarea «Parte Si» y si no se cumple realiza la tarea «Parte No».



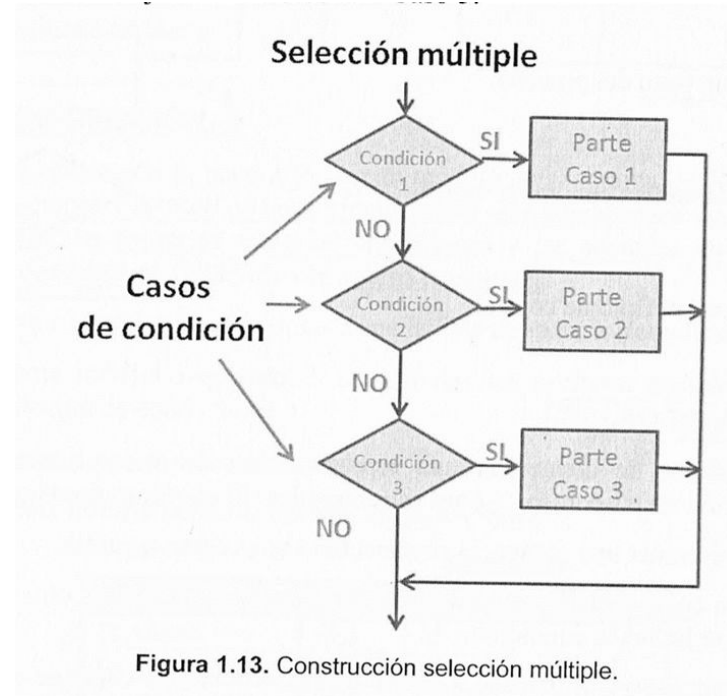


# Fases de desarrollo del software

## Diseño

**Construcciones lógicas para diseñar cualquier programa:**

- **Selección múltiple:** es una extensión de la anterior. Decisiones sucesivas hasta que ocurre una condición verdadera.

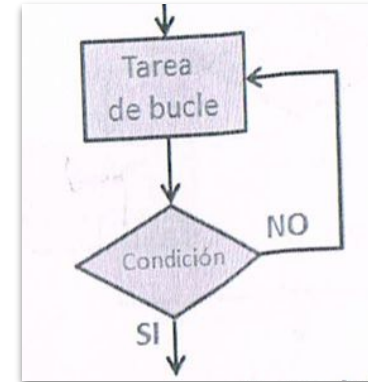
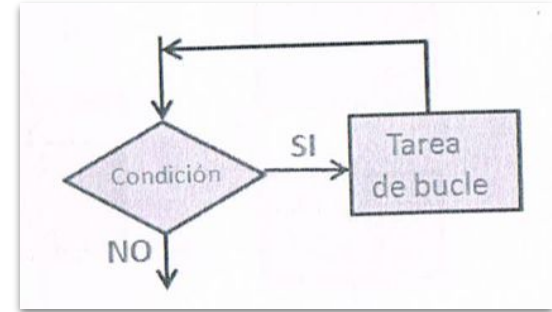


# Fases de desarrollo del software

## Diseño

### Construcciones lógicas para diseñar cualquier programa:

- **Repetitiva:** proporciona los bucles
  - **do while:** primero ejecuta una tarea del bucle y después comprueba la condición, si no cumple se vuelve a realizar la tarea, si cumple, finaliza el bucle.
  - **while:** primero se comprueba la condición y después se realiza la tarea del bucle repetidamente siempre y cuando la condición se cumpla; el bucle finaliza cuando la condición no se cumple.



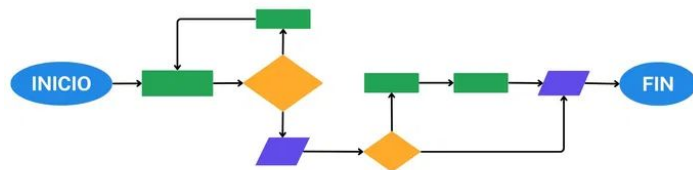
# Fases de desarrollo del software

## Diseño

- Para representar el diseño se emplean algunas **herramienta gráficas** como son: diagramas de flujo, diagramas de cajas, tablas de decisión o pseudocódigo.

Pseudocódigo	Diagrama de Flujo
<ol style="list-style-type: none"><li><b>Inicio</b></li><li>Declaración de variables: <math>R = 0</math>, <math>H = 0</math></li><li><b>Leer</b> el valor de Radio (R) y Altura (H)</li><li><b>Calcular</b> el Volumen aplicando la fórmula</li><li><b>Calcular</b> el valor del área aplicando la fórmula respectiva</li><li><b>Escribir</b> el valor del Área y del Volumen</li><li><b>Fin</b></li></ol>	<pre>graph TD; INICIO([INICIO]) --&gt; Input[/R,H/]; Input --&gt; Vol[VOL= π* R² H]; Vol --&gt; Area[AREA= 2* π R H]; Area --&gt; Output[/AREA,VOL/]; Output --&gt; FIN([FIN]);</pre>

Email (condition1)	T	T	F	F
Password (condition2)	T	F	T	F
Expected Result (Action)	Account Page	Incorrect password	Incorrect email	Incorrect email








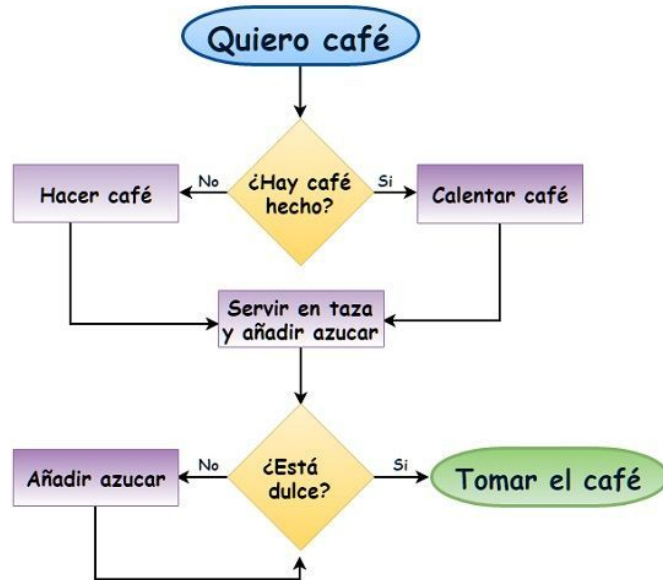
# Fases de desarrollo del software

## Diseño

### Notaciones gráficas:

- **Diagramas de flujo:** es una herramienta muy usada para el diseño procedimental. Esta es la simbología:

Símbolo	Nombre	Función
	Inicio / Final	Representa el inicio y el final de un proceso
	Línea de Flujo	Indica el orden de la ejecución de las operaciones. La flecha indica la siguiente instrucción.
	Entrada / Salida	Representa la lectura de datos en la entrada y la impresión de datos en la salida
	Proceso	Representa cualquier tipo de operación
	Decisión	Nos permite analizar una situación, con base en los valores verdadero y falso



# Fases de desarrollo del software

## Diseño

### Notaciones gráficas:

- **Diagramas de flujo (ejercicios):**
  1. Realiza un diagrama de flujo para mostrar la suma de los 50 primeros números.
  2. Hacer un diagrama de flujo que permita leer 2 números diferentes y nos diga cual es el mayor de los 2 números.
  3. Crear un diagrama de flujo de procesos en el que se almacenen 3 números en 3 variables A, B y C. El diagrama debe decidir cuál es el mayor de los tres.

# Fases de desarrollo del software

## Diseño

### Notaciones gráficas:

- **Tablas de decisión:** permiten representar en tablas una combinación compleja de condiciones y las acciones que llevan a cabo combinando estas condiciones.

### Pasos a seguir:

- Hacer una lista de todas las acciones y condiciones
- Asociar conjuntos de condiciones con acciones específicas
- Determinar las reglas indicando las acciones que ocurren para un determinado número de condiciones.

Pago con tarjeta oro	Sí	No	Sí	No	No
Pago con tarjeta club	No	Sí	No	Sí	No
Modalidad joven de tarjeta	Sí	Sí	No	No	No
Inviabile					
Descuento 15% (oro)	X		X		
Descuento 5% (club)		X		X	
Descuento 5% (joven)	X	X			
Calcular importe	X	X	X	X	X

CONDICIONES	1	2	3	4
Antigüedad empleado	<5 años	5 a <10 años	10 a 15 años	> 15 años
ACCIONES				
Calcular bonificación por antigüedad.				
Sueldo x 1% x años antig.	X			
Sueldo x 1,5% x años antig.		X		
Sueldo x 2% x años antig.			X	
Sueldo x 2,5% x años antig.				X

# Fases de desarrollo del software

## Diseño

### Notaciones gráficas:

- **Pseudocódigo:**

- Utiliza texto descriptivo para realizar el diseño de un algoritmo.
- A primera vista se parece a un lenguaje de programación.
- Depende mucho de quien lo escriba ya que no hay un estándar definido.
- No puede ser compilado.

### Ejemplo:

Inicio

Abrir fichero

Leer Registro del Fichero

Mientras no sea Fin de Fichero Hacer

    Procesar Registro leído

    Leer Registro del Fichero

Fin Mientras

Cerrar Fichero

Fin

# Fases de desarrollo del software

[Video](#)

## Diseño

### Diseño orientado a objetos

- El **diseño de software orientado a objetos** es difícil. Hay que partir de un análisis orientado a objetos en el que se definen todas las clases que son importantes para el problema a resolver, las operaciones y los atributos asociados, las relaciones, los comportamientos y las comunicaciones entre clases.
- Este diseño define 4 capas:
  - **Subsistema:** se centra en el diseño de los subsistemas que implementan las funciones principales del sistema.
  - **Clases y objetos:** especifica la arquitectura de objetos global y jerarquía de clases.
  - **Mensajes:** indica cómo se realiza la colaboración entre los objetos
  - **Responsabilidades:** identifica las operaciones y atributos que caracterizan cada clase.



# Fases de desarrollo del software

## Codificación

- Una vez diseñado, se realiza el proceso de codificación. Esta tarea la realiza el **programador** y tiene que cumplir **exhaustivamente** con todos los datos impuestos en el análisis y en el diseño de la aplicación.
- El programador recibe las **especificaciones** de 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**.
- En el proyecto debe haber unas normas de codificación y estilo, claras y homogéneas. Estas normas facilitan la corrección y mantenimiento de los programas, sobre todo cuando las realiza una persona que no los ha desarrollado.

# Fases de desarrollo del software

## Codificación

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.

Una vez obtenido el código, es necesario **probar** el programa y verificar las especificaciones del diseño.

# Fases de desarrollo del software

## Codificación

- **Código Fuente:** se escribe usando algún lenguaje de programación de alto nivel y contiene el conjunto de instrucciones necesarias.
- **Código Objeto:** es el código binario resultado de compilar el código fuente.

La **compilación** es la traducción de una sola vez del programa, y se realiza utilizando un **compilador**. La **interpretación** es la traducción y ejecución simultánea del programa línea a línea.

*El código objeto no es directamente inteligible por el ser humano, pero tampoco por la computadora. Es un código intermedio entre el código fuente y el ejecutable.*

- **Código Ejecutable:** Es el código binario resultante de enlazar los archivos de código objeto con ciertas rutinas y bibliotecas necesarias. El sistema operativo será el encargado de cargar el código ejecutable en memoria RAM y proceder a ejecutarlo. También es conocido como código máquina y ya sí es directamente inteligible por la computadora.

# Fases de desarrollo del software

## Codificación



# Fases de desarrollo del software

## Pruebas

En esta etapa ya tenemos el software y se trata de **encontrar errores** tanto de **codificación** como de **especificación** o **diseño**.

Se realizan tareas de **Verificación y Validación del software (V&V)**.

- **Verificación:** son las actividades que tratan de comprobar si estamos diseñando el producto correctamente, es decir, si el software implementa correctamente una función específica.
- **Validación:** son las actividades que tratan de comprobar que el producto es correcto, es decir, si se ajusta a los requisitos del cliente.

# Fases de desarrollo del software

## Pruebas

- **Objetivo de esta etapa:** planificar y diseñar pruebas que detecten diferentes clases de errores, con la menor cantidad de tiempo y esfuerzo posible.
- Una prueba tiene **éxito** si descubre un error que no se había detectado anteriormente.
- **Caso de prueba:** documento que especifica los valores de entrada, salida esperada y las condiciones previas para la ejecución de la prueba.

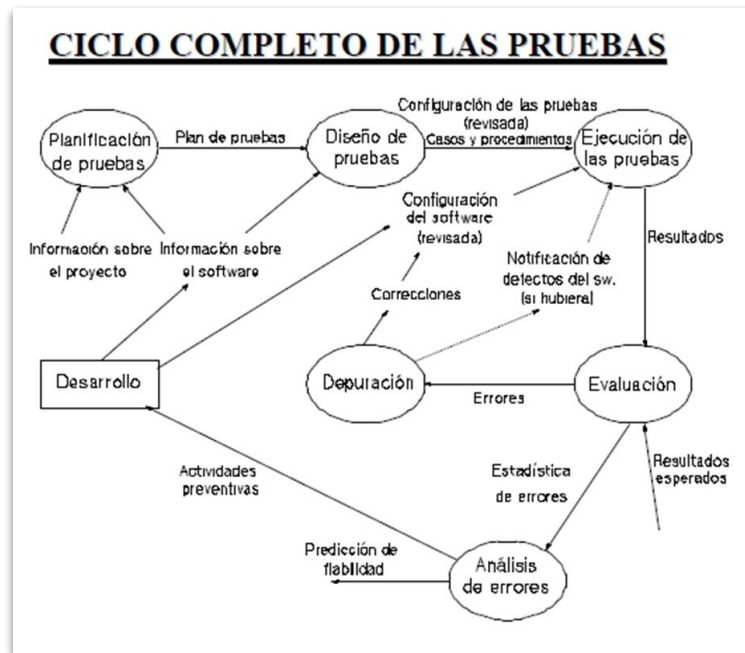
### RECOMENDACIONES

- ✓ Cada caso de prueba debe definir el resultado de salida esperado.
- ✓ El programador debe evitar probar sus propios programas, ya que desea (consciente o inconscientemente) demostrar que funcionan sin problemas.
- ✓ Se debe inspeccionar a conciencia el resultado de cada prueba, así, poder descubrir posibles síntomas de defectos.
- ✓ Al generar casos de prueba, se deben incluir tanto datos de entrada válidos y esperados como no válidos e inesperados.

# Fases de desarrollo del software

## Pruebas. Ciclo completo

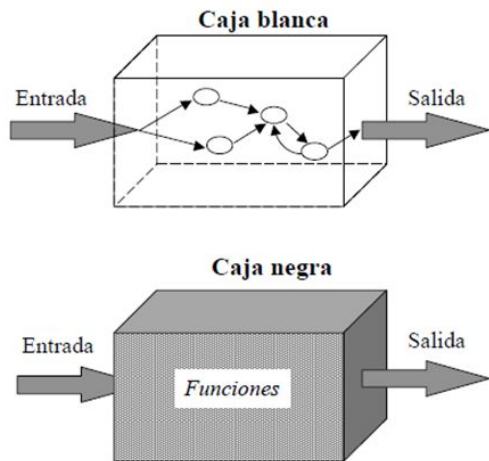
1. Generar un plan de pruebas.
2. Diseñar las pruebas. Ver las técnicas a utilizar.
3. Generar los casos de prueba.
4. Definir los procedimientos de prueba. ¿Cómo, quién, cuándo...?
5. Ejecución de las pruebas
6. Evaluación y realización de informe.
7. Depuración. (Si se corrige un error hay que volver a probar el software para ver que se ha resuelto )
8. Análisis de errores. (Sirve para predecir la fiabilidad del software y mejora los procesos de desarrollo)



# Fases de desarrollo del software

## Pruebas

### LOS ENFOQUES DE DISEÑO DE PRUEBAS DE CAJA BLANCA Y DE CAJA NEGRA



Veremos dos técnicas:

- **Prueba de caja blanca:** se centra en validar la estructura interna del programa (necesita conocer los detalles procedimentales del código)
- **Pruebas de caja negra:** se centran en validar los requisitos funcionales sin fijarse en el funcionamiento interno del programa (lo que se busca es que demuestren que las funciones del software son operativas)

No son excluyentes, **pueden combinarse** ambas.



# Fases de desarrollo del software

## Documentación

Todas las etapas del desarrollo **deben estar documentadas** y es en esta etapa donde tenemos que reunir todos los documentos generados, clasificarlos según el nivel técnico de sus descripciones.

Los documentos:

- Actuarán como medio de comunicación entre los miembros del equipo de desarrollo
- Serán un repositorio de información del sistema (mantenimiento)
- Proporcionarán información que ayude a planificar la gestión del presupuesto y programar el proceso del desarrollo del software
- Algunos documentos deben indicar a los usuarios cómo utilizar y administrar el sistema (manual de usuario)

# Fases de desarrollo del software

## Documentación

Se divide en dos clases:

- **Documentación del proceso:** registra el proceso de desarrollo y mantenimiento. Los documentos indican planes, estimaciones y horarios que predicen y controlan el proceso de software, informan sobre cómo usar los recursos durante el proceso de desarrollo y las normas de cómo se ha de implementar el proceso (se usa para gestionar todo el proceso de desarrollo del software).
- **Documentación del producto:** describe el producto que está siendo desarrollado (se utiliza una vez que el sistema está en funcionamiento).
  - Documentación del **sistema**, punto de vista técnico (mantenimiento).
  - Documentación del **usuario**, orientada a los futuros usuarios del sistema.

# Fases de desarrollo del software

## Documentación del usuario

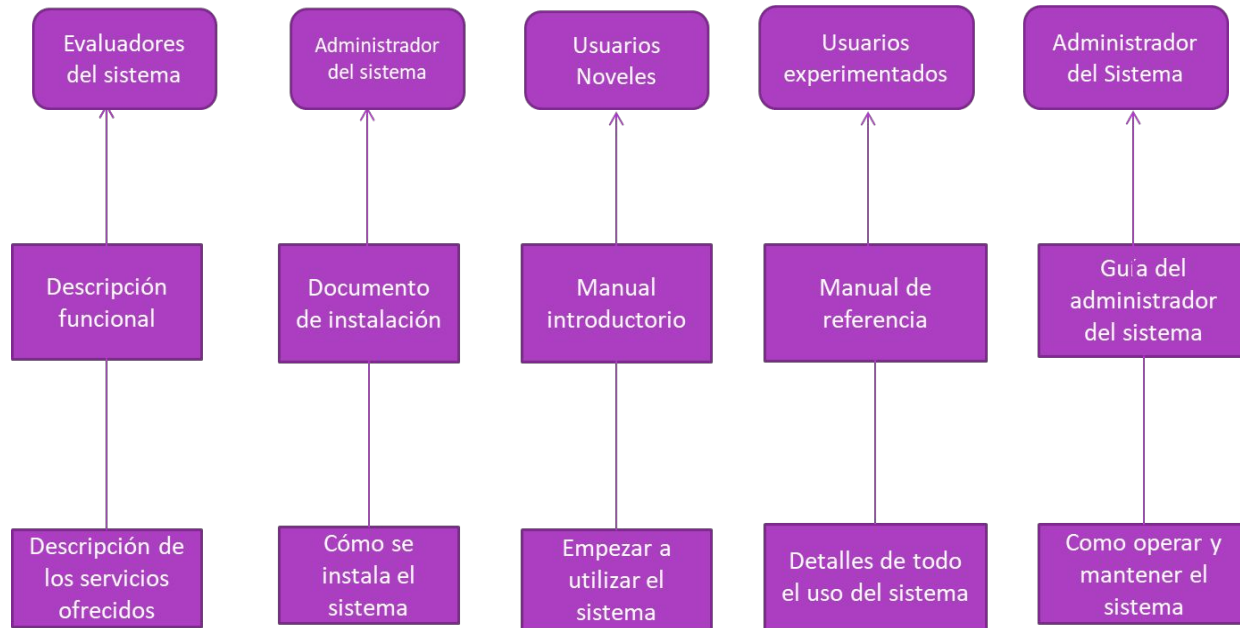
Hay que tener en cuenta que hay diferentes tipos de usuarios:

- **Usuarios finales:** utilizan el software para realizar alguna tarea, necesitan conocer cómo el software les ayuda a realizarla, pero no están interesados en los detalles informáticos ni del programa.
- **Administradores del sistema:** son responsables de la gestión de los programas informáticos utilizados por los usuarios finales (gestores de red, técnicos, etc.)

Para atender a los distintos tipos de usuarios con diferentes niveles de experiencia se definen una serie de documentos que deberán ser entregados con el sistema de software.

# Fases de desarrollo del software

## Documentación



# Fases de desarrollo del software

## Documentación del sistema

Incluye todos los documentos que describen el sistema, desde la especificación de requisitos hasta las pruebas de aceptación. Los documentos que describen el diseño, la implementación y las pruebas del sistema son esenciales para entender y mantener el software.

Debe incluir:

- Fundamentos del sistema (objetivos)
- Análisis y especificación de requisitos (ERS)
- Diseño (describe la arquitectura del sistema)
- Implementación (describe la forma en que se expresa el sistema en el lenguaje de programación usado y las acciones en forma de comentarios dentro del programa)
- Plan de pruebas del sistema (describe cómo las ud. De programa son evaluadas individualmente y todo el sistema se prueba tras la integración)
- Plan de pruebas de aceptación (describe las pruebas que ha de pasar el sistema antes de que los usuarios las acepten)
- Diccionarios de datos (contiene la descripción de todos los términos que se relacionan con el sistema de software en cuestión)

# Fases de desarrollo del software

## Explotación

En esta etapa se lleva a cabo la **instalación** y **puesta en marcha** del producto software en el entorno de trabajo del cliente.

Se ejecutan varias tareas:

- **Definir la estrategia para la implementación del proceso.** Normas y procedimientos para tratar los problemas que surjan e ir probando el programa.
- **Pruebas de operación** para cada *release* (gestión de entregas) del producto software se efectuarán pruebas de funcionamiento.
- **Uso operacional del sistema.** El sistema entrará en acción en el entorno previsto de acuerdo con la documentación de usuario.
- **Soporte al usuario.** Se debe proporcionar asistencia y consultoría a los usuarios cuando la soliciten. Registrada y supervisada.

# Fases de desarrollo del software

## Mantenimiento



# Fases de desarrollo del software

## Mantenimiento

El mantenimiento software consiste en la **modificación** de este producto de software con objeto de corregir fallos mejorar el rendimiento u otros atributos o adaptar el producto a un entorno modificado. Existen cuatro tipos de mantenimiento:

- **Mantenimiento adaptativo:** si se requiere cambiar el entorno de uso de la aplicación (que incluye al sistema operativo, a la plataforma de hardware o, en el caso de las aplicaciones web, al navegador), puede ser indispensable modificarla para mantener su plena funcionalidad en estas nuevas condiciones. Es el más usual debido a los rápidos cambios de la informática.
- **Mantenimiento correctivo:** corrige los defectos encontrados en el software, y que originan un comportamiento distinto al deseado.



# Fases de desarrollo del software

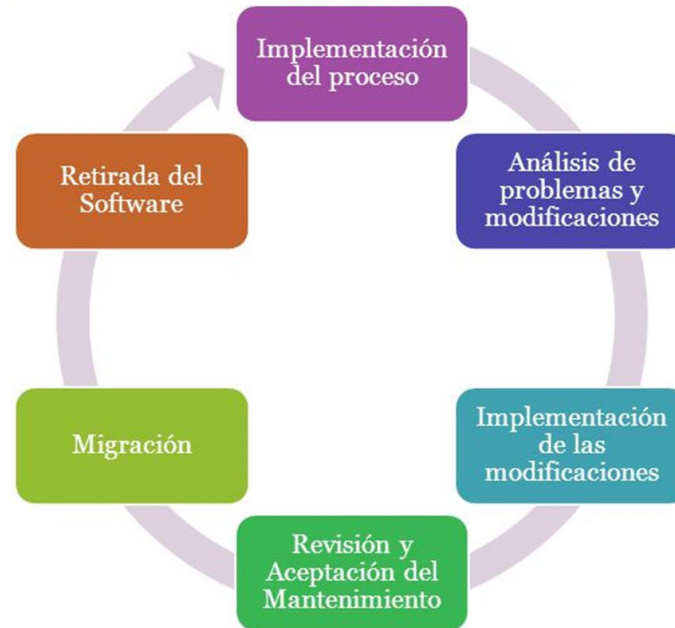
## Mantenimiento

- **Mantenimiento perfectivo:** por distintas razones, el usuario puede solicitar el agregado de nuevas funcionalidades o características no contempladas al momento de la implementación del software. El mantenimiento perfectivo adapta la aplicación a este requerimiento.
- **Mantenimiento preventivo:** modificación del producto de software sin alterar las especificaciones del mismo con el fin de mejorar las tareas de mantenimiento. Ej. reestructuración de programas para mejorar su legibilidad o añadir comentarios para mejorar la comprensión.



# Fases de desarrollo del software

Mantenimiento. Tareas a realizar



# Fases de desarrollo del software

## Mantenimiento. Tareas a realizar

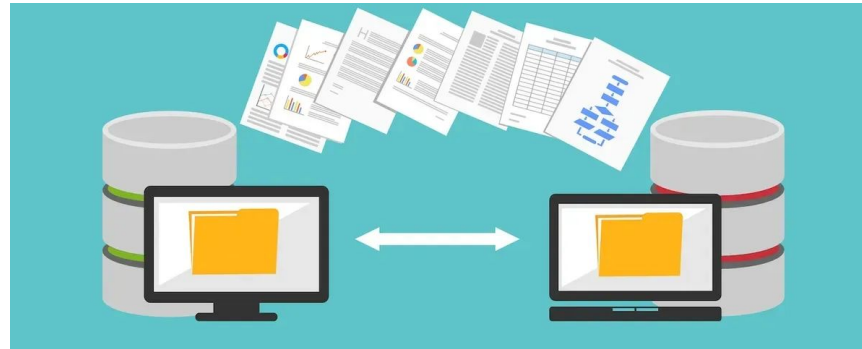
- **Implementación del proceso:** establecer planes y procedimientos, ejecutarlos, registrarlo todo e informar a los usuarios sobre la situación.
- **Análisis de problemas y modificaciones:** realizar el análisis teniendo en cuenta el impacto en la organización, el sistema existente, los sistemas con los que interacciona...
- **Implementación de las modificaciones:** analizar y determinar qué documentación y unidades de software y versiones se deben modificar.
- **Revisión/aceptación del mantenimiento:** revisiones con la organización que autorice las modificaciones para determinar la integridad del sistema modificado.



# Fases de desarrollo del software

## Mantenimiento. Tareas a realizar

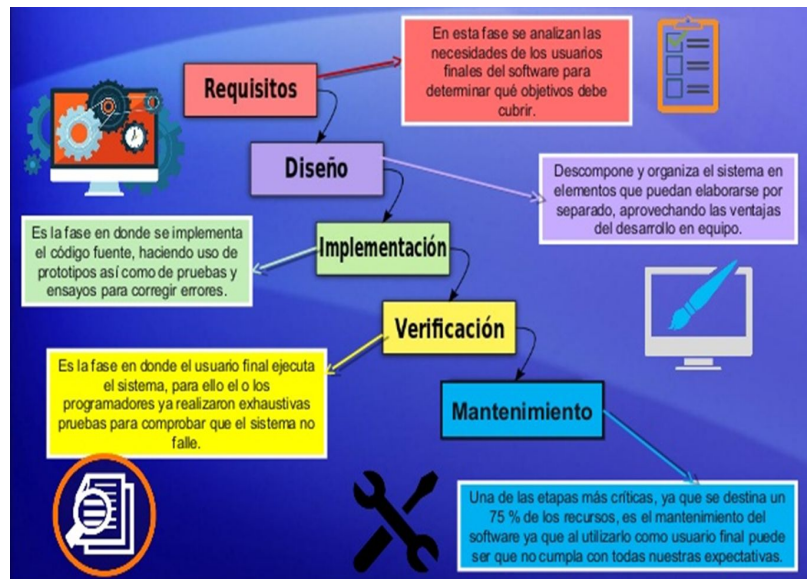
- **Migración:** preparar, documentar y ejecutar un plan de migración (incluir a los usuarios, notificar los planes y las actividades)
- **Retirada del software:** se retirará por petición del propietario. La operación de retirada e implantación del nuevo software deberá hacerse en paralelo y durante este periodo se deberá formar a los usuarios.



# Modelos de ciclos de vida

## Modelo en cascada

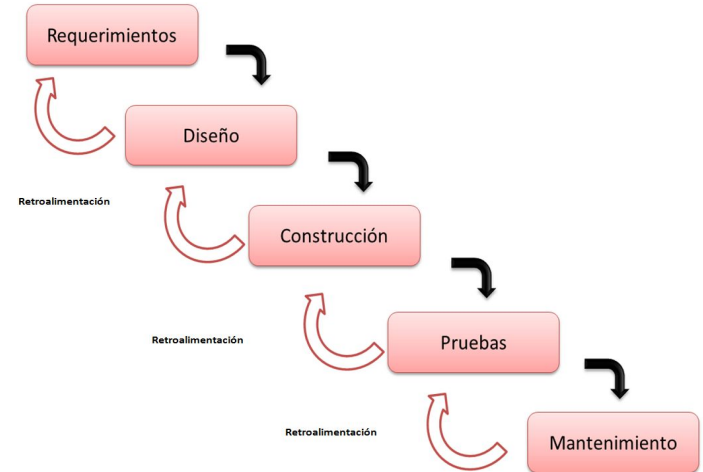
- Es el **modelo clásico** del software.
- Es prácticamente **imposible** de utilizar, ya que requiere conocer de antemano todos los requisitos del sistema.
- Sólo se podría aplicar a **pequeños desarrollo**, ya que las etapas pasan de una a otra sin posibilidad de retorno.



# Modelos de ciclos de vida

## Modelo en cascada con retroalimentación

- Es uno de los modelos **más utilizados**.
- Proviene del modelo anterior, pero se introduce una **realimentación entre etapas**, de forma que podamos volver atrás en cualquier momento para corregir, modificar o depurar algún aspecto. No obstante, si se prevén muchos cambios durante el desarrollo no es el modelo más idóneo.
- Es el modelo perfecto si el proyecto es rígido (pocos cambios, poco evolutivo) y los requisitos están claros.



# Modelos de ciclos de vida

## Modelo en cascada con retroalimentación

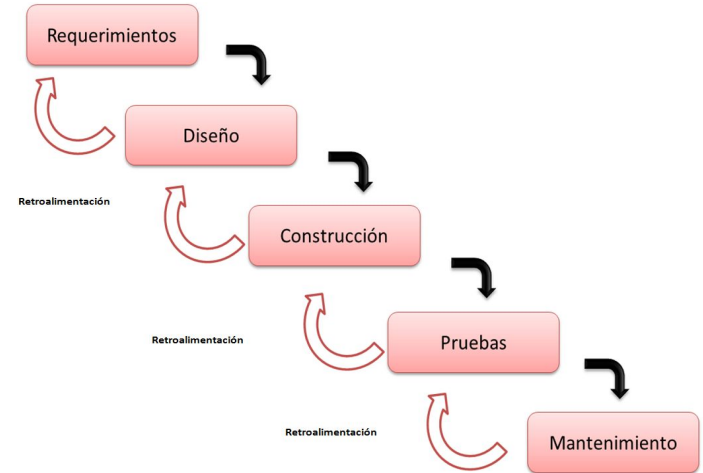
- **Ventaja**

Fácil de comprender, de planificar y de seguir, y existen herramientas que lo soportan.

- **Inconveniente**

Los cambios generan confusiones conforme el proyecto avanza, es difícil para el cliente especificar todos los requisitos al principio, cuanto más tarde se detecte un fallo más costoso será hacer los cambios que implique (rehacer todas las fases).

**Si descubrimos un fallo al final = DESASTRE**



# Modelos de ciclos de vida

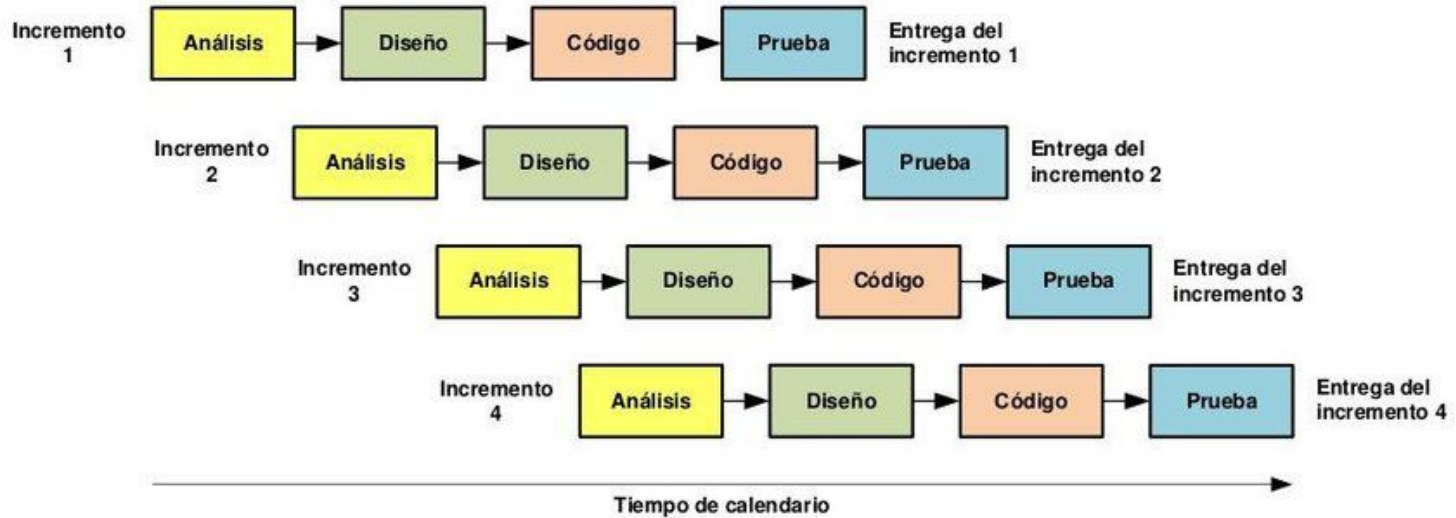
## Modelo iterativo incremental

- Para resolver el problema del tiempo del modelo en cascada, podemos ir haciendo entregas con una **funcionalidad parcial** del software al cliente y aumentarla en entregas posteriores.
- De esta forma se concibe el desarrollo de software como un proceso en el que se van produciendo diversos **incrementos** o **entregables**.
- Es decir se realiza el modelo anterior (cascada) eliminando la retroalimentación y haciendo los cambios necesarios en cada incremento y entrega. Las fases se **repiten y refinan**, y van propagando su mejora a las fases siguientes.
- El modelo incremental aplica **secuencias lineales de forma escalonada** mientras progresa el tiempo en el calendario.



# Modelos de ciclos de vida

## Modelo iterativo incremental



# Modelos de ciclos de vida

## Modelos evolutivos

- Los modelos evolutivos son **más modernos** que los anteriores. Tienen en cuenta la naturaleza cambiante del software.
- Se trata de **modelos iterativos** que van desarrollando versiones más completas del software con el paso del tiempo.
- En concreto nos vamos a centrar en dos de ellos: el modelo de **construcción de prototipos** y el modelo **en espiral**.



# Modelos de ciclos de vida

## Modelos evolutivos

- **Construcción de prototipos**

En la práctica, la construcción de prototipos ayuda al ingeniero de sistemas y al cliente a entender de mejor manera cuál será el resultado de la construcción cuando los requisitos estén satisfechos.

Tiene ciertas desventajas:

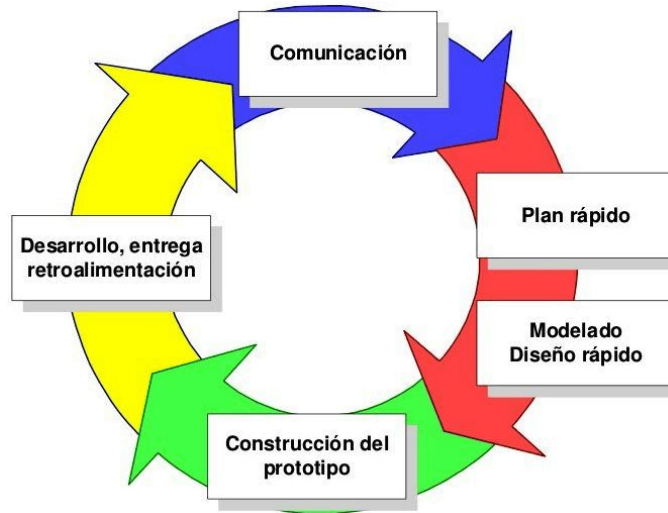
- El cliente considera la mayoría de las veces al prototipo como el producto final.
- La calidad del software o la factibilidad de mantenimiento puede que no se tomen en cuenta.

Es conveniente seguir este modelo cuando el cliente es capaz de definir un conjunto de objetivos generales para el software pero no puede identificar los requerimientos en detalle o cuando los desarrolladores tienen dudas considerables.

# Modelos de ciclos de vida

## Modelos evolutivos

- Construcción de prototipos

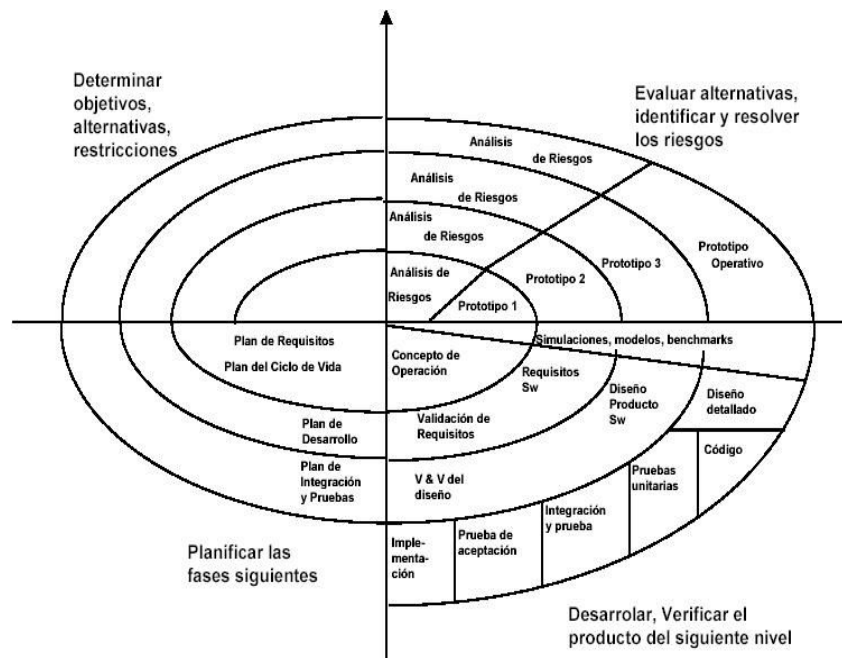


# Modelos de ciclos de vida

## Modelos evolutivos

- **Modelo en espiral**

Es una combinación del modelo anterior con el modelo en cascada añadiendo un nuevo elemento, **el análisis de riesgo**. En él, el software se va construyendo repetidamente en forma de versiones que son cada vez mejores, debido a que incrementan la funcionalidad en cada versión. Es un modelo bastante complejo.



# Modelos de ciclos de vida

Cuadro Comparativo	Modelo en Cascada	Modelo Incremental	Modelo en Espiral
Ventajas	Permite a los administradores, avanzar en el desarrollo, aunque en una escala muy bruta.	El modelo proporciona todas las ventajas del modelo en cascada realimentado, reduciendo sus desventajas sólo al ámbito de cada incremento.	Al ser un modelo de Ciclo de Vida orientado al riesgo se dice que uno de los aspectos fundamentales de su éxito radica en que el equipo que lo aplique sea capaz de detectar y catalogar correctamente dicho riesgo.
Desventajas	Los cambios introducidos durante el desarrollo pueden confundir al equipo profesional en las etapas tempranas del proyecto. Si los cambios se producen en etapa madura (codificación o prueba) pueden ser catastróficos para un proyecto grande.	El modelo Incremental no es recomendable para casos de sistemas de tiempo real, de alto nivel de seguridad, de procesamiento distribuido, y/o de alto índice de riesgos.	Requiere mucha experiencia y habilidad para la evaluación de los riesgos, lo cual es requisito para el éxito del proyecto. Es difícil convencer a los grandes clientes que se podrá controlar este enfoque evolutivo.
Crítica	Este es un modelo en el cual se debe usar cuando todos los requerimientos han sido establecidos claramente de entrada.	En este modelo se debe especificar con precisión todo lo que el sistema va a hacer antes de desarrollarlo. Lo cual lo hace manejable y disminuiría los costos.	Este modelo es útil para grandes proyectos pero no ha sido utilizado tanto como el lineal secuencial o el de prototipos.
Características	Planear un proyecto antes de embarcarse en él. Definir el comportamiento externo deseado del sistema antes de diseñar su arquitectura interna. Documentar los resultados de cada actividad. Diseñar un sistema antes de codificarlo. Testear un sistema después de construirlo.	Construir un sistema pequeño es siempre menos riesgoso. Al ir desarrollando parte de las funcionalidades, es más fácil determinar si los requerimientos para los niveles subsiguientes son correctos. Si un error importante es realizado, sólo la última iteración necesita ser descartada.	Un enfoque cíclico para el crecimiento incremental del grado de definición e implementación de un sistema, mientras que disminuye su grado de riesgo. Un conjunto de puntos de fijación para asegurar el compromiso del usuario con soluciones.

# Concepto de programa

## Programa informático

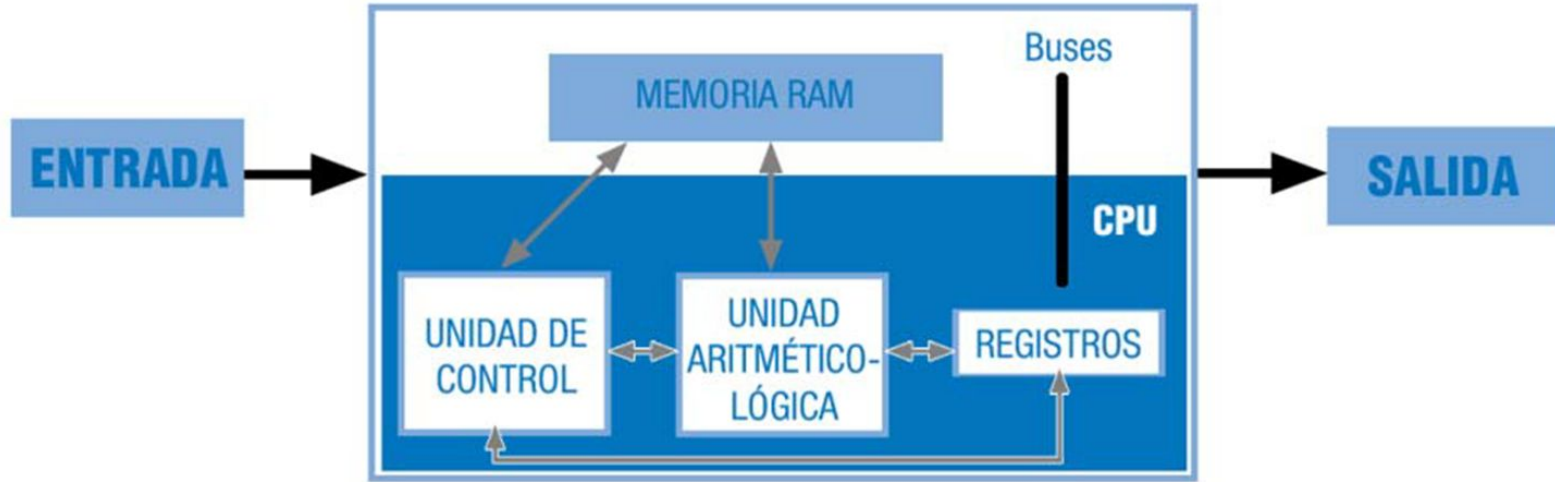
Conjunto de **instrucciones escritas** en un lenguaje de programación que aplicadas sobre un conjunto de datos **resuelven un problema** o parte del mismo.

Para que pueda ser ejecutado es necesario traducirlo a lenguaje máquina (**compilarlo**) después cargarlo en la memoria principal para que el procesador ejecute una a una todas las tareas .

Para ejecutar un programa necesitamos **recursos hardware** del ordenador, CPU, memoria RAM, dispositivos de E/S, etc. Las instrucciones de un programa se cargan en la **RAM** y son ejecutadas por la **CPU**.



# Concepto de programa



*Componentes de la CPU*



# Concepto de programa

## Componentes de la CPU

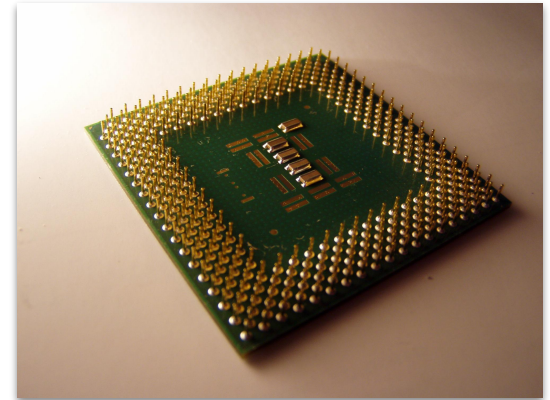
- **Unidad de control (UC):** interpreta y ejecuta las instrucciones máquina almacenadas en la memoria principal y genera las señales de control necesarias para ejecutarlas.
- **Unidad aritmético-lógica (UAL):** recibe los datos sobre los que efectúa operaciones de cálculo, comparaciones y toma decisiones lógicas, determina si una afirmación es cierta o falsa (Álgebra de Boole) y devuelve un resultado todo ello supervisado por la UC.
- **Registros:** de trabajo o de propósito general donde se almacena información temporal, es el almacenamiento interno de la CPU.



# Concepto de programa

## Registros

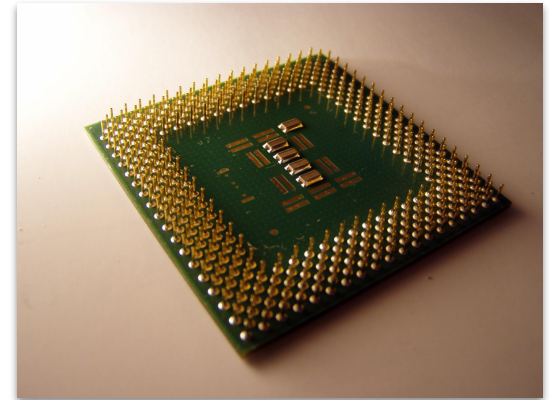
- **Contador de programa:** contiene la dirección de la siguiente instrucción a ejecutar. Se actualiza cada vez que la CPU captura una instrucción.
- **Registro de instrucción:** contiene el código de la instrucción a ejecutar.
- **Registro de dirección de memoria:** Contiene la dirección de una posición de memoria donde se encuentra o se almacenará la información (el intercambio se realiza a través del bus de direcciones)
- **Registro de intercambio de memoria:** recibe, o envía, el dato contenido en la posición apuntada por el registro de dirección de memoria (el intercambio de datos con la memoria se realiza a través del bus de datos)



# Concepto de programa

## Registros

- **Decodificador de instrucción (DI):** extrae y analiza el código de operación de la instrucción en curso contenida en el registro de instrucción y genera las señales de control necesarias para ejecutar correctamente la instrucción.
- **Reloj:** proporciona una sucesión de impulsos eléctricos constantes y marca los tiempos de ejecución de los pasos a realizar para cada instrucción, marca el ritmo de funcionamiento del decodificador de instrucción.
- **El secuenciador:** este dispositivo genera órdenes y micro órdenes elementales que sincronizadas con los impulsos del reloj hace que se ejecute paso a paso y de manera ordenada la instrucción cargada en él.





# Lenguajes de programación

Consta de:

- **Léxico o vocabulario:** formado por el conjunto de símbolos permitidos.
- **Sintaxis:** reglas que indican cómo realizar las construcciones con dichos símbolos.
- **Semántica:** reglas que determinan el significado de cualquier construcción del lenguaje.

```
1 // class declaration
2 public class ProgrammingExample {
3
4     // method declaration
5     public void sayHello() {
6
7         // method output
8         System.out.println("Hello World!");
9     }
10 }
```

# Lenguajes de programación

## Clasificación

Podemos clasificar los distintos tipos de lenguajes de programación en base a distintas características:

*Según su nivel de abstracción (lo cerca que esté del lenguaje humano)*

- **Lenguajes de programación de alto nivel:** por su esencia, están más próximos al razonamiento humano.
- **Lenguajes de programación de nivel medio**
- **Lenguajes de programación de bajo nivel:** están más próximos al funcionamiento interno de la computadora:
  - Lenguaje ensamblador.
  - Lenguaje máquina.

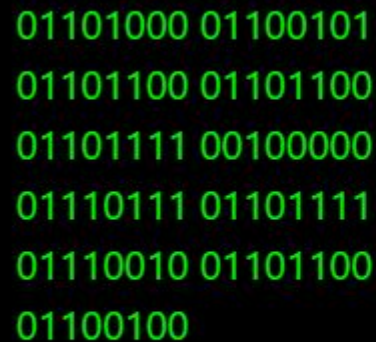
```
-u 100 1a
OCFD:0100 BAOB01      MOV    DX,010B
OCFD:0103 B409      MOV    AH,09
OCFD:0105 CD21      INT     21
OCFD:0107 B400      MOV    AH,00
OCFD:0109 CD21      INT     21
-d 10b 13f
OCFD:0100                48 6F 6C 61 2C      Hola,
OCFD:0110                20 65 73 74 65 20 65 73-20 75 6E 20 70 72 6F 67      este es un prog
OCFD:0120                72 61 60 61 20 68 65 63-68 6F 20 65 6E 20 61 73      rama hecho en as
OCFD:0130                73 65 60 62 6C 65 72 20-70 61 72 61 20 6C 61 20      sembler para la
OCFD:0140                57 69 68 69 70 65 64 69-61 24      Wikipedias
```

# Lenguajes de programación

## Lenguajes de bajo nivel

### Lenguaje máquina

- Es por excelencia el lenguaje de más bajo nivel.
- Sus instrucciones son combinaciones de unos y ceros (alfabeto binario)
- 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 de un equipo a otro).
- Hoy día nadie programa en este lenguaje.



```
01101000 01100101
01101100 01101100
01101111 00100000
01110111 01101111
01110010 01101100
01100100
```

Lenguaje Máquina

# Lenguajes de programación

## Lenguajes de bajo nivel

### Lenguaje ensamblador

- Sustituyó al lenguaje máquina para facilitar la labor de programación.
- En lugar de unos y ceros se programa usando mnemotécnicos (instrucciones complejas).
- Necesita traducción al lenguaje máquina para poder ejecutarse.
- Sus instrucciones son sentencias que hacen referencia a la ubicación física de los archivos en el equipo (trabaja con registros de memoria física)
- Es difícil de utilizar.

```
> pd
0x00000000 90      nop
0x00000001 90      nop
0x00000002 6800009c00 push 0x9c0000 ; 0x009c0000
0x00000007 e8c7ace37b call 0x7be3acd3
           0x7be3acd3(unk)
0x0000000c bb04009c00 mov ebx, 0x9c0004
0x00000011 8903     mov [ebx], eax
0x00000013 e81903f47b call 0x7bf40331
           0x7bf40331()
0x00000018 bb08009c00 mov ebx, 0x9c0008
0x0000001d 8903     mov [ebx], eax
0x0000001f bb00009c00 mov ebx, 0x9c0000
0x00000024 c60300   mov byte [ebx], 0x0
0x00000027 68e8030000 push 0x3e8 ; 0x000003e8
0x0000002c e81124e37b call 0x7be32442
           0x7be32442(unk)
0x00000031 ebf4     jmp 0x100000027
0x00000033 90      nop
0x00000034 ff      invalid
0x00000035 ff      invalid
0x00000036 ff      invalid
0x00000037 ff      invalid
```



# Lenguajes de programación

## Lenguajes de nivel medio

Tiene características que los acercan al nivel bajo y otras al alto nivel. Un ejemplo de este lenguaje es **C**. Suele utilizarse para aplicaciones como la creación de un sistema operativo.

```
/* Suma de n números */  
  
#include <stdio.h>  
int main() {  
    int num=0, suma=0;  
  
    do {  
        suma=suma+num;  
        printf("un número: ");  
        scanf("%d",&num);  
    } while(num>=0);  
    printf("suma es: %d", suma);  
    return 0;  
}
```

# Lenguajes de programación

[Video](#)

## Lenguajes de alto nivel

- Sustituyeron al lenguaje ensamblador para facilitar más la labor de programación. Son más cercanos al razonamiento humano
- En lugar de mnemotécnicos, se utilizan **sentencias y órdenes** derivadas del idioma inglés (necesita traducción al lenguaje máquina).
- 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, no dependen del hardware del ordenador y no requieren conocimientos de lenguaje máquina.

**Ejemplos:** ALGOL, Basic, C++, C#, Clipper, COBOL, Fortran, Java, Logo, Pascal, Object Pascal, Perl, PHP, PL/SQL, Python, etc.

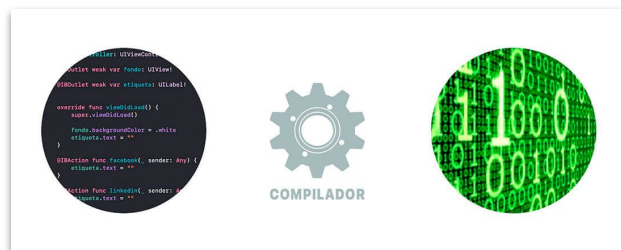
# Lenguajes de programación

## Según la forma de ejecución: Compilados o interpretados

El **código fuente** es el conjunto de instrucciones que la computadora deberá realizar, escritas por los programadores en algún lenguaje de alto nivel.

Este conjunto de instrucciones no es directamente ejecutable por la máquina, sino que deberá ser **traducido** al lenguaje máquina, que la computadora será capaz de entender y ejecutar.

Los programas traductores que realizan esta operación se llaman **compiladores** o **intérpretes**.



# Lenguajes de programación

## Según la forma de ejecución: Compilados o interpretados

Un **compilador** lee el programa escrito en código fuente y lo traduce a otro lenguaje llamado **código objeto**, que tras enlazar con las librerías lo convierte en **lenguaje máquina**, que ya el ordenador entiende.

A estos lenguajes se les llama **compilados**.

Otra alternativa para traducir estos programas son los **intérpretes**, que lo que hacen es leer una instrucción traducirla al lenguaje máquina y ejecutarla. A los que son así traducidos se les llama **lenguajes interpretados**.



**Compilados traduce todo a la vez**

**Interpretados traduce línea a línea**

# Lenguajes de programación

## JAVA

Los procesadores del lenguaje Java combinan la compilación y la interpretación. Un programa fuente en Java puede compilarse primero en un formato intermedio, llamado **bytecode** y después una máquina virtual los interpreta.



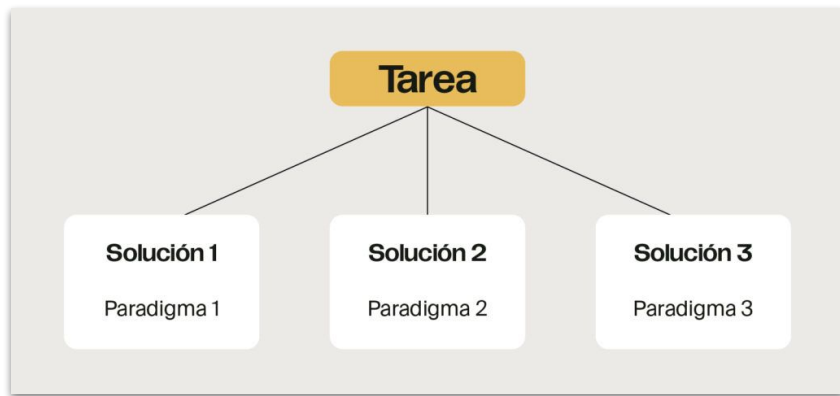
# Lenguajes de programación

## Según el paradigma de programación

Un **paradigma** define un conjunto de reglas, patrones y estilos de programación que son usados por un grupo de lenguajes de programación. Dependiendo del problema a resolver resultará más apropiado un paradigma u otro.

Existen las siguientes categorías:

- Paradigma imperativo
- Paradigma funcional o aplicativo
- Paradigma lógico o declarativo



# Lenguajes de programación

## Según el paradigma de programación

### Lenguajes imperativos

- Los **programas imperativos** son un conjunto de instrucciones que le indican al computador **cómo** realizar una tarea, de la misma manera que el modo imperativo en los lenguajes naturales humanos le dice qué hacer al interlocutor.
- La sentencia principal es la **asignación**.
- La mayoría de los lenguajes usados para desarrollo de software comercial son imperativos.
- Dentro de esta categoría se engloba la **programación estructurada**, la **programación modular** y la **programación orientada a objetos**.
- **Ejemplos:** Basic, Fortran, Algol, Pascal, C, Ada, C++, Java, C#.

# Lenguajes de programación

## Según el paradigma de programación

### Lenguajes imperativos: programación estructurada

- Cuando hablamos del diseño hablamos de las **tres construcciones lógicas** que son el fundamento de la programación estructurada: la estructura secuencial, la condicional y la repetitiva.
- Son lenguajes fáciles de leer, pueden ser leídos secuencialmente desde el comienzo hasta el final sin perder la continuidad de lo que hace.
- Su problema es que todo el código se encuentra en un **único bloque** y si el programa es muy grande o el problema muy complejo es difícil su lectura y manejo.



# Lenguajes de programación

## Según el paradigma de programación

### Lenguajes imperativos: programación modular

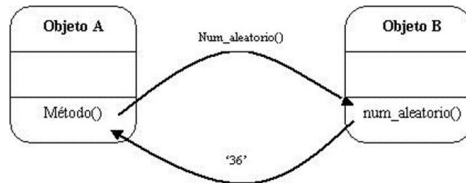
- Este tipo de programación es la **evolución** de la programación estructurada.
- Al dividir el programa en varios módulos, **varios programadores** pueden trabajar **simultáneamente**.
- Los módulos se pueden reutilizar en otras aplicaciones.
- Resolver pequeños problemas de forma aislada es **menos costoso** que abordar el problema de forma global.
- **Ejemplos:** Pascal, C, Fortran, Modula-2, etc

# Lenguajes de programación

## Según el paradigma de programación

### Lenguajes imperativos: programación orientada a objetos

- En este tipo de programación, el programa está compuesto por **una serie de objetos** ( no por instrucciones, ni módulos, como anteriormente hemos visto).
- Un objeto tiene una estructura de datos y una colección de métodos u operaciones que manipulan esos datos. A estos datos se les llama **atributos** y las operaciones definen el comportamiento de los objeto y cambian el valor de los mismos.
- Para comunicarse un objeto con otro lo hace a través del paso de **mensajes**.



# Lenguajes de programación

## Según el paradigma de programación

Lenguajes imperativos: programación orientada a objetos



# Lenguajes de programación

## Según el paradigma de programación

### Lenguajes imperativos: programación orientada a objetos

Declaración de una clase, llamada Persona, con dos atributos y cuatro métodos.

```
Public class Persona {  
    //Atributos  
    private string nombre;  
    private int edad;  
    //Constructor  
    public Persona (String nombre, int edad){  
        this.nombre=nombre;  
        this.edad= edad;  
    }  
    //Métodos  
    public void setNombre (String nom) { nombre=nom; }  
    public void setEdad (int ed) { edad=ed; }  
    public string getNombre () {return nombre;} //devuelve nombre  
    public int getEdad() {return edad;} //devuelve edad  
}
```

La creación de un objeto de esa clase sería:  
Persona persona =new Persona ("Manuel", 20);

set = modificar  
get = Mostrar



# Lenguajes de programación

## Según el paradigma de programación

### Lenguajes funcionales

- Este paradigma está basado en el concepto matemático de **función**.
- Están constituidos por un conjunto de definiciones de funciones.
- No existe la operación de asignación
- Las variables almacenan definiciones o referencias a expresiones
- La operación fundamental es la **aplicación de una función a una serie de argumentos**.
- La computación se realiza mediante la evaluación de **expresiones**.
- **Ejemplos:** Lisp, Scheme, ML, Miranda o Haskell.

# Lenguajes de programación

## Según el paradigma de programación

### Lenguajes lógicos

- Los **programas escritos** en estos lenguajes se pueden ver como una **base de datos** formada por listas de **declaraciones lógicas** (reglas) que se pueden consultar. La ejecución consistirá en realizar preguntas de forma interactiva.
- El lenguaje lógico por excelencia es **Prolog**.
- Está indicado para aplicaciones muy específicas como: sistemas expertos, demostración de teoremas, consulta de bases de datos relacionales, procesamiento del lenguaje natural.

# Máquinas virtuales

**Máquina virtual:** aplicación software de una máquina que ejecuta los programas como si fuese una máquina real.

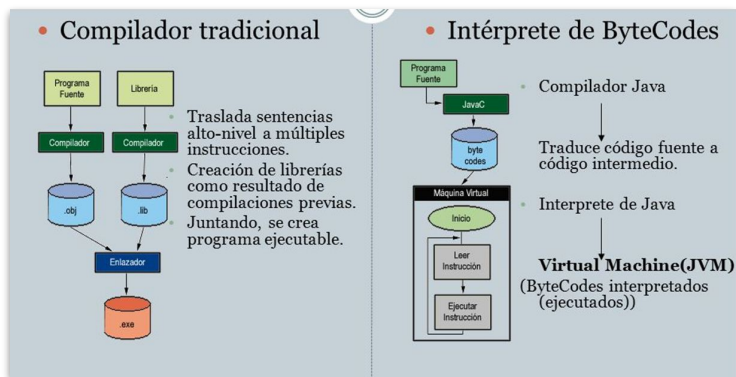
Existen los siguientes tipos:

- **Máquinas virtuales de sistema:** permite ejecutar en la misma máquina física varias máquinas virtuales, cada una con su sistema operativo. Coexistencia. Ejemplo: Virtual Box.
- **Máquinas virtuales de proceso:** 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. La máquina se inicia automáticamente cuando se lanza el proceso que se desea ejecutar y se detiene para cuando éste finaliza. Su objetivo es el de proporcionar un entorno de ejecución independiente de la plataforma de hardware y del sistema operativo, que oculte los detalles de la plataforma subyacente y permita que un programa se ejecute siempre de la misma forma sobre cualquier plataforma. Ej. Máquina virtual de Java

# Máquinas virtuales

## Máquina virtual de Java

- Los programas compilados en lenguaje Java, pueden ejecutarse en cualquier plataforma, es decir, 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).





# Máquinas virtuales

## Máquina virtual de Java

- Extensión archivo código fuente: **.java** (texto plano)
- Nombre del compilador: **javac**
- Extensión archivo compilado: **.class**
- Este archivo contiene un lenguaje intermedio llamado **bytecode**.
- La máquina virtual de Java traduce el **bytecode a binario**.
- Los ficheros .class pueden ejecutarse en distintas plataformas: Windows, Solaris, Linux o Mac OS.



# Herramientas de programación

Para llevar a cabo la codificación y prueba de los programas solemos utilizar entornos de programación, llamados **entornos de desarrollo integrado, o IDE**, que nos permiten:

- 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).

Están diseñados para maximizar la **productividad** del programador.

Son un **programa informático** formado por un conjunto de herramientas de programación que facilitan las tareas descritas anteriormente.

La mayoría proporcionan un entorno de trabajo **visual**.