



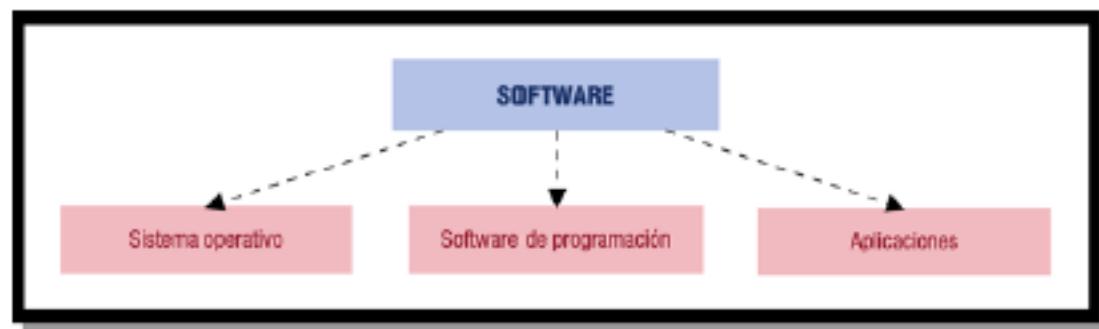
# Tema I Desarrollo De Software

## ENTORNOS DE DESARROLLO

Marina Moreno Martínez  
IES Pedro Mercedes  
Cuenca 2019

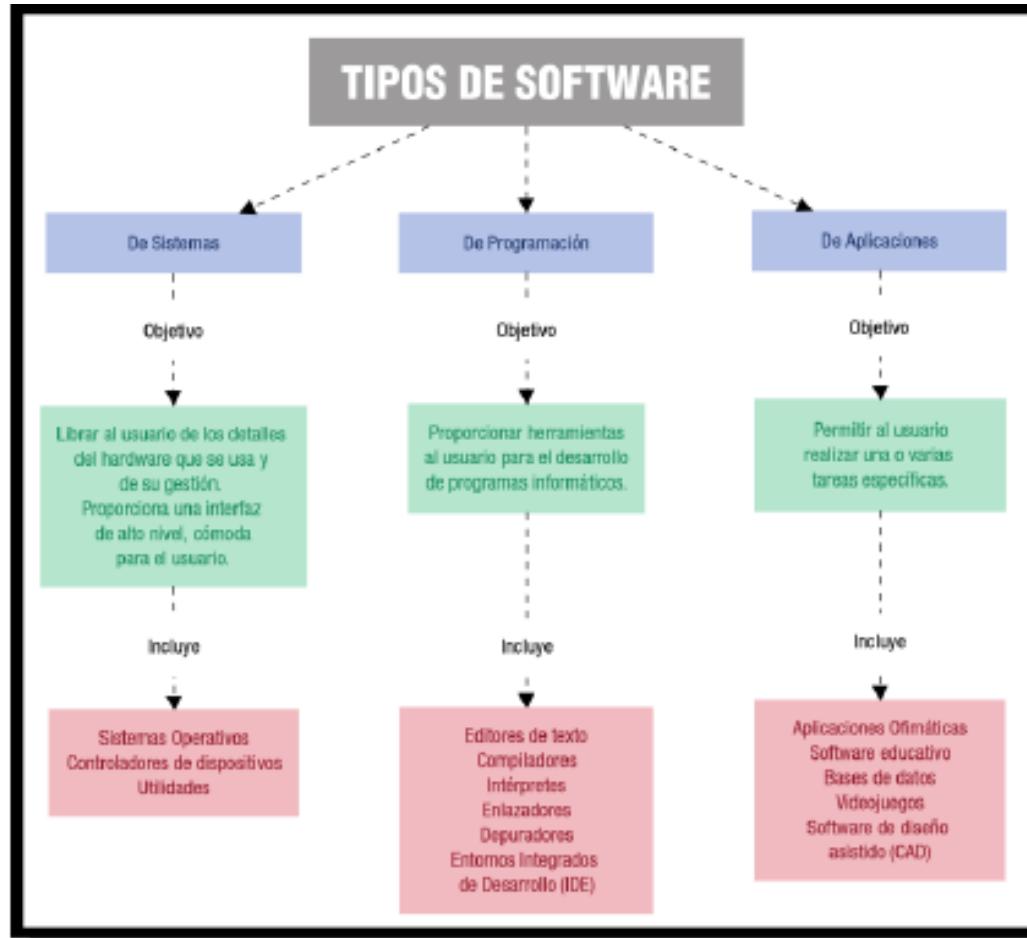
# I.- Software y programa.Tipos de software.

- ❑ Ordenador = Software +Hardware
- ❑ El software es el conjunto de programas informáticos que actúan sobre el hardware para ejecutar lo que el usuario desee.



# I.- Software y programa.

## Software basado en el tipo de tarea que realiza.



# I.- Software y programa.

## Software basado en el método de distribución.

### Shareware

- Evaluación del producto durante un tiempo limitado.
- Shareware de precio cero o pago modesto

### Freeware

- Distribución sin cargo
- A veces con código fuente
- Redistribución con restricciones.

### Adware

- Programas Shareware que descargan publicidad

### Software multimedia

### Software de uso específico

## I.- Licencias Software

# Licencias de software

### **Software libre**

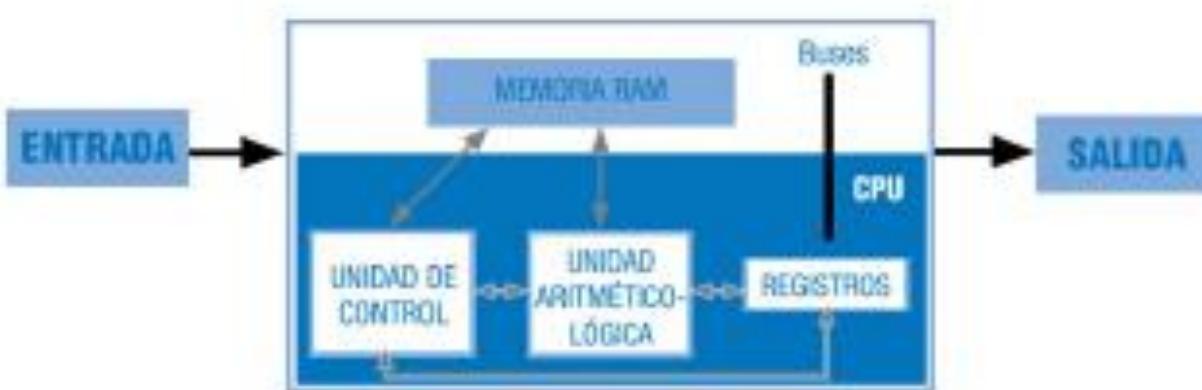
- Libertad para utilizar, estudiar, distribuir y mejorar un programa.
- Licencia GPL

### **Software propietario**

- Redistribución en formato binario.
- El autor prohíbe alguna o todas las libertades del software libre.

## 2.- Relación hardware-software.

- ❑ La primera arquitectura hardware con programa almacenado se estableció en 1946 por John Von Neumann
- ❑ Esta relación software-hardware la podemos poner de manifiesto desde dos puntos de vista:
  - a) Desde el punto de vista del sistema operativo
  - b) Desde el punto de vista de las aplicaciones



### 3.- Desarrollo del software.

- **Desarrollo del software** es el proceso desde que se concibe una idea hasta que un programa es implementado y puesto en funcionamiento.



## **3.- Desarrollo del software.**

### **3.1 ciclo de vida del software**

- ❑ El ciclo de vida de un producto software comprende el periodo que transcurre desde que el producto es concebido hasta que deja de estar disponible o es retirado.
- ❑ Tipos:
  - Modelo en Cascada.
  - Modelo en Cascada con Realimentación.
  - Modelos Evolutivos:
    1. Modelo Iterativo Incremental
    2. Modelo en Espiral

# 3.- Desarrollo del software.

## 3.1 ciclo de vida del software

### □ **Modelo en Cascada.**

- Es el modelo de vida clásico del software.
- Es prácticamente imposible que se pueda utilizar, ya que requiere conocer de antemano todos los requisitos del sistema.
- Sólo es aplicable a pequeños desarrollos, ya que las etapas pasan de una a otra sin retorno posible. (se presupone que no habrá errores ni variaciones del software).

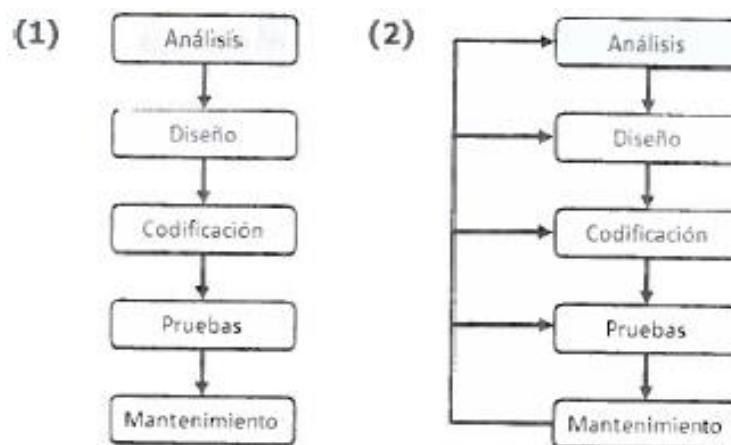


Figura 1.4. Modelo en Cascada.

## 3.- Desarrollo del software.

### 3.1 ciclo de vida del software

#### □ **Modelo en Cascada con Realimentación**

- Realimentación entre etapas.
- Es el modelo perfecto si el proyecto es rígido (pocos cambios, poco evolutivo) y los requisitos están claros.

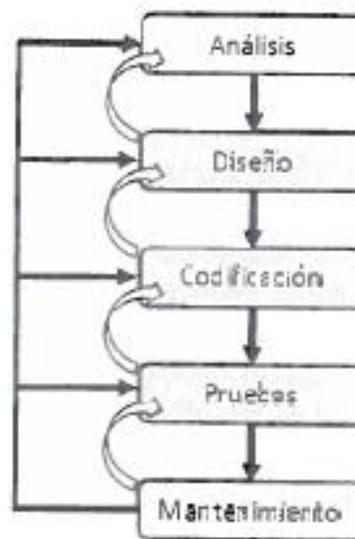


Figura 1.5. Modelo en Cascada con Realimentación.

## **3.- Desarrollo del software.**

### **3.1 ciclo de vida del software**

#### **Ventajas:**

- Fácil de comprender, planificar y seguir.
- La calidad del producto resultante es alta.
- Permite trabajar con personal poco cualificado.

#### **Inconvenientes:**

- La necesidad de tener todos los requisitos definidos desde el principio (algo que no siempre ocurre ya que pueden surgir necesidades imprevistas).
- Es difícil volver atrás si se cometen 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 haya realizado con éxito anteriormente.
- Los requisitos son estables y están bien comprendidos.
- Los clientes no necesitan versiones intermedias.

## 3.- Desarrollo del software.

### 3.1 ciclo de vida del software

- **Modelos Evolutivos** Tienen en cuenta la naturaleza cambiante y evolutiva del software. Distinguimos dos variantes:
  - **Modelo Iterativo Incremental** Está basado en el modelo en cascada con realimentación, donde las fases se repiten y refinan, y van propagando su mejora a las fases siguientes.

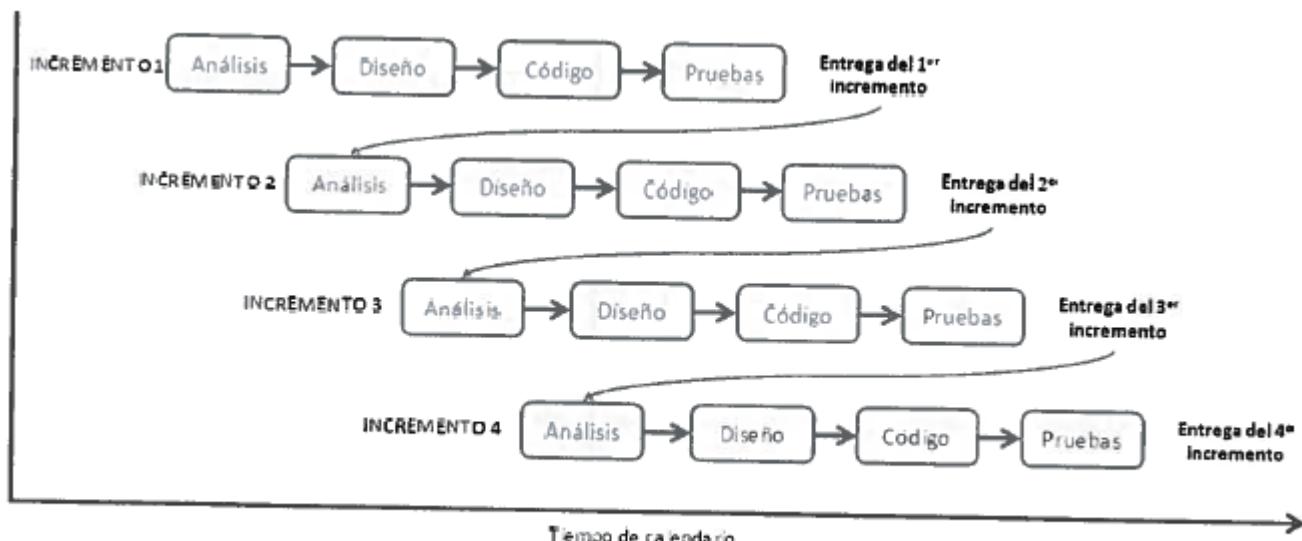


Figura 1.6. Modelo Iterativo Incremental.

## **3.- Desarrollo del software.**

### **3.1 ciclo de vida del software**

#### **Ventajas:**

- No se necesitan 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 recomendable para desarrollo de sistemas de tiempo real, de alto nivel de seguridad, de procesamiento distribuido, y/o de alto índice de riesgos.

#### **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.

# 3.- Desarrollo del software.

## 3.I ciclo de vida del software

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

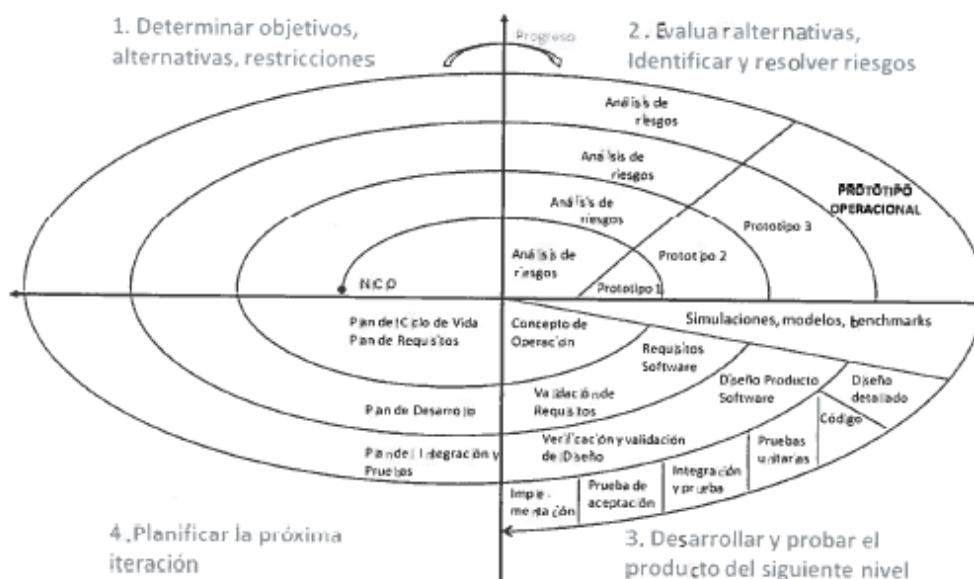


Figura 1.8. Ejemplo de Modelo en espiral.

### **3.- Desarrollo del software.**

#### **3.I ciclo de vida del software**

##### **Ventajas:**

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

##### **Inconvenientes:**

- Es difícil evaluar los riesgos.
- El costo 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 para:**

- Proyectos de gran tamaño y que necesitan constantes cambios.
- Proyectos donde sea importante el factor riesgo.

### **3.- Desarrollo del software.**

#### **3.2.- Herramientas de apoyo al desarrollo del software.**

- ❑ Las herramientas **CASE** son un conjunto de aplicaciones que se utilizan en el desarrollo de software con el objetivo de reducir costes y tiempo del proceso, mejorando por tanto la productividad del proceso.
  
- ❑ Estas herramientas permiten:
  - Mejorar la planificación del proyecto.
  - Darle agilidad al proceso.
  - Poder reutilizar partes del software en proyectos futuros.
  - Hacer que las aplicaciones respondan a estándares.
  - Mejorar la tarea del mantenimiento de los programas.
  - Mejorar el proceso de desarrollo, al permitir visualizar las fases de forma gráfica.

### **3.- Desarrollo del software.**

#### **3.2.- Herramientas de apoyo al desarrollo del software.**

##### **Tipos de herramientas CASE:**

- **U-CASE**: ofrece ayuda en las fases de planificación y análisis de requisitos.
- **M-CASE**: ofrece ayuda en análisis y diseño.
- **L-CASE**: ayuda en la programación del software, detección de errores del código, depuración de programas y pruebas y en la generación de la documentación del proyecto.

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

## **4.- Fases en el desarrollo y ejecución del software.**

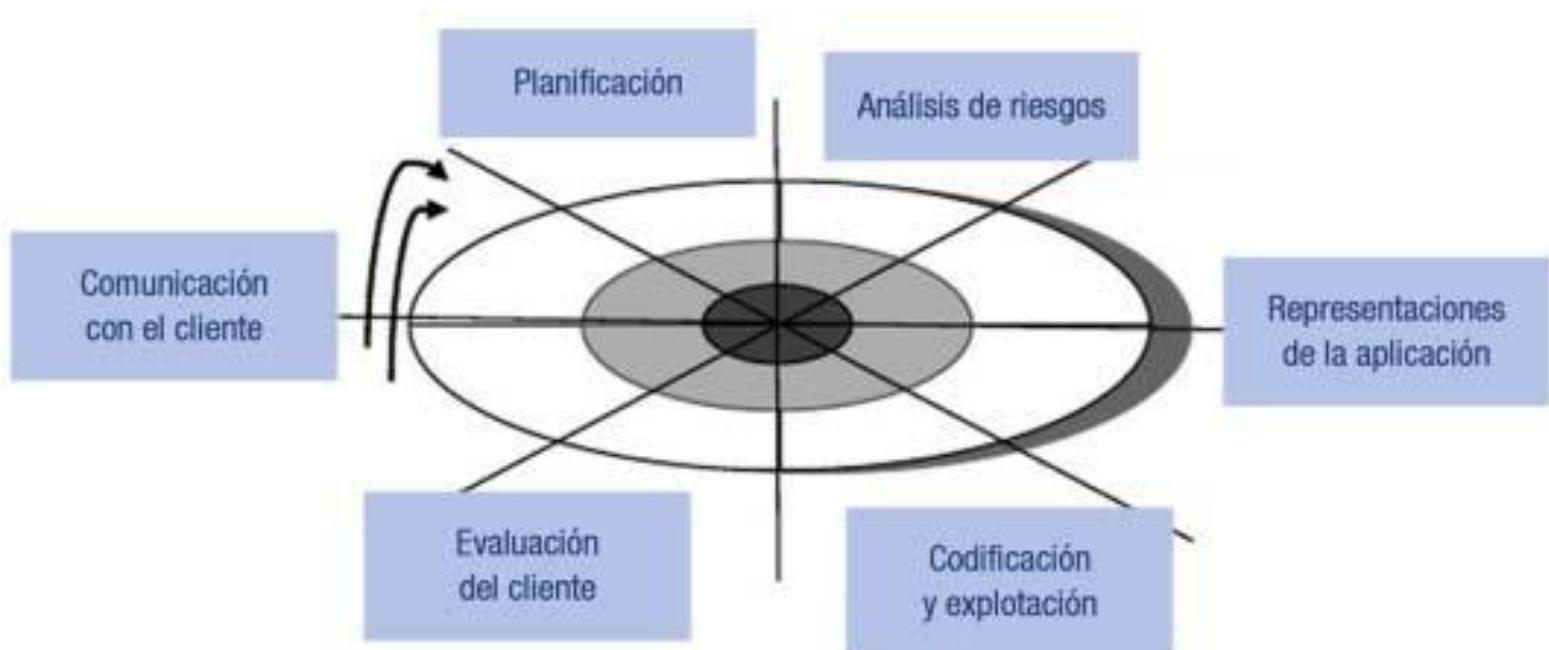
### **4.I.- Análisis.**

- Es la primera etapa del proyecto, la más complicada y la que más depende de la capacidad del analista.
  
- Métodos de comunicación con el cliente:
  - Entrevistas
  - Desarrollo conjunto de aplicaciones(JAD Joint Application Development)
  - Planificación conjunta de requisitos (JRP Joint Requirements Planning)
  - Brainstorming
  - Prototipos
  - Casos de uso. UML(Unified Modeling Language)

## 4.- Fases en el desarrollo y ejecución del software.

### 4.I.- Análisis.

- ❑ Lo fundamental es la buena comunicación entre el analista y el cliente para que la aplicación que se va a desarrollar cumpla con sus expectativas.



## 4.- Fases en el desarrollo y ejecución del software.

### 4.I.- Análisis.

- ❑ Se especifican y analizan los requisitos funcionales y no funcionales del sistema.
  - **Requisitos Funcionales:** Qué funciones tendrá que realizar la aplicación. Qué respuesta dará la aplicación ante todas las entradas. Cómo se comportará la aplicación en situaciones inesperadas.
  - **Requisitos No funcionales:** Tiempos de respuesta del programa, legislación aplicable, tratamiento ante la simultaneidad de peticiones, etc.

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

## 4.- Fases en el desarrollo y ejecución del software.

### 4.I.- 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.

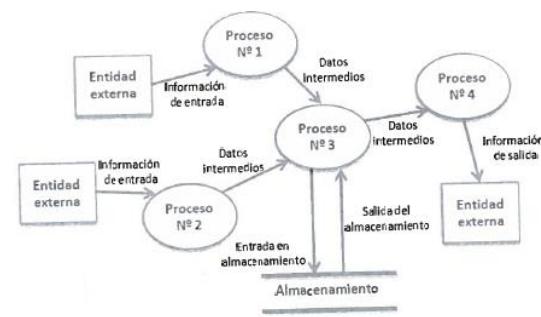


Figura 1.9. Diagrama de flujo de datos o DFD.

- Ejemplo de requisitos funcionales, en la aplicación para nuestros clientes de las tiendas de cosmética, habría que considerar:

- Si desean que la lectura de los productos se realice mediante códigos de barras.
- Si van a detallar las facturas de compra y de qué manera la desean.
- Si los trabajadores de las tiendas trabajan a comisión, tener información de las ventas de cada uno.
- Si van a operar con tarjetas de crédito.
- Si desean un control del stock en almacén.
- Etc.

## 4.- Fases en el desarrollo y ejecución del software.

### 4.1.- Análisis.

#### □ Diagrama de flujo de datos. DFD

Flujos de datos	
Procesos	
Almacenes de datos	
Entidades externas	

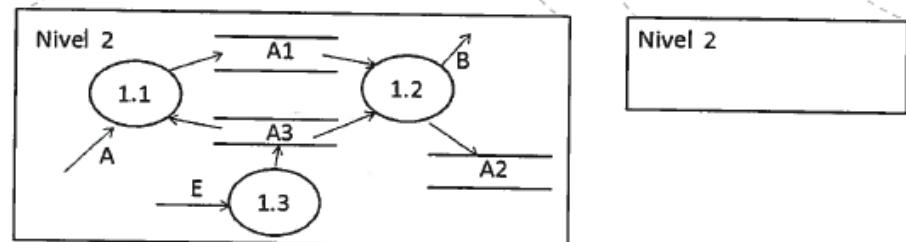
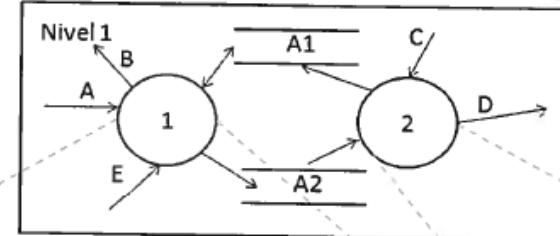
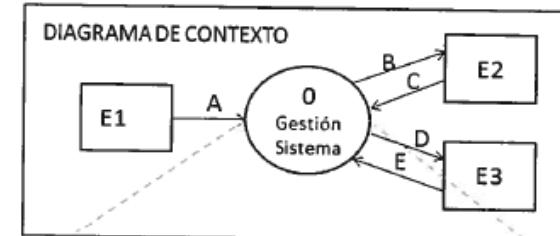


Figura 1.37. Descomposición en niveles de un DFD.

## 4.- Fases en el desarrollo y ejecución del software.

### 4.I.- Análisis.

- ❑ La culminación de esta fase es el documento **ERS (Especificación de Requisitos Software)**.
- ❑ En este documento quedan especificados:
  - La planificación de las reuniones que van a tener lugar.
  - Relación de los objetivos del usuario cliente y del sistema
  - Relación de los requisitos funcionales y no funcionales del sistema.
  - Relación de objetivos prioritarios y temporización.
  - Reconocimiento de requisitos mal planteados o que llevan contradicciones, etc.

#### 1. Introducción.

- 1.1 Propósito.
- 1.2 Ámbito del Sistema.
- 1.3 Definiciones, Acrónimos y Abreviaturas.
- 1.4 Referencias.
- 1.5 Visión general del documento.

#### 2. Descripción General.

- 2.1 Perspectiva del Producto.
- 2.2 Funciones del Producto.
- 2.3 Características de los usuarios.
- 2.4 Restricciones.
- 2.5 Suposiciones y Dependencias.
- 2.6 Requisitos Futuros.

#### 3. Requisitos Específicos.

- 3.1 Interfaces Externas.
- 3.2 Funciones.
- 3.3 Requisitos de Rendimiento.
- 3.4 Restricciones de Diseño.
- 3.5 Atributos del Sistema.
- 3.6 Otros Requisitos.

#### 4 Apéndices.

## 4.- Fases en el desarrollo y ejecución del software.

### 4.2.- Diseño

- ❑ Durante esta fase, donde ya sabemos lo que hay que hacer, el siguiente paso es **¿Cómo hacerlo?**
- ❑ Se debe dividir el sistema en partes y establecer qué relaciones habrá entre ellas.
- ❑ Decidir qué hará exactamente cada parte, por ejemplo:
  - Entidades y relaciones de la BD.
  - Selección del lenguaje de programación que se utilizará.
  - • Selección del Sistema Gestor de Base de Datos.
  - • Etc.
- ❑ Hay dos tipos de diseño:
  - Estructurado
  - Orientado a Objetos

## 4.- Fases en el desarrollo y ejecución del software.

### 4.2.- Diseño estructurado

- Los fundamentos del diseño a nivel de componentes son las siguientes construcciones lógicas: secuencial, condicional y repetitiva.

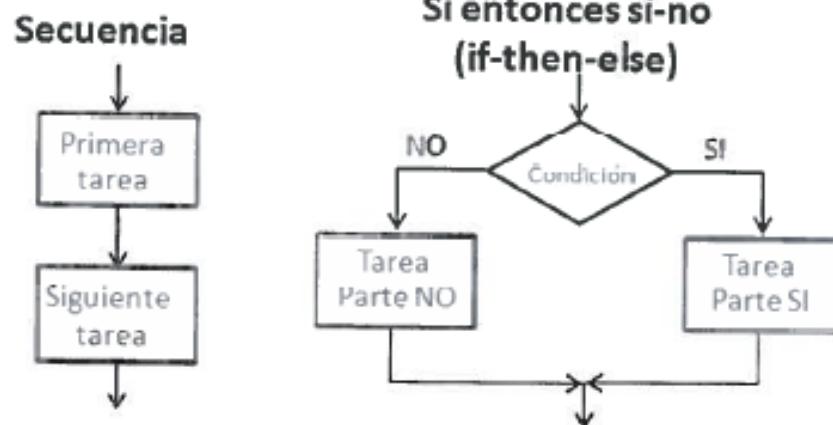


Figura 1.12. Construcción secuencial y condicional.

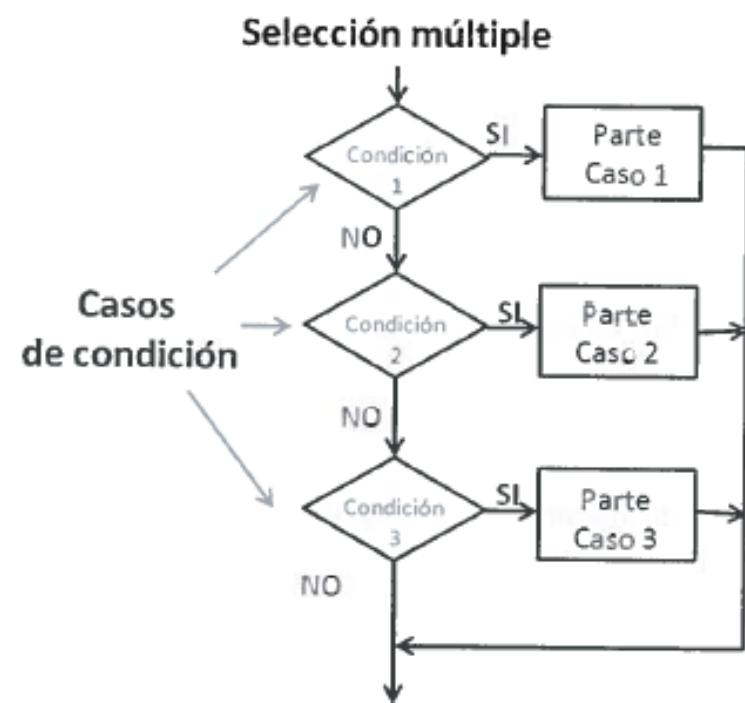


Figura 1.13. Construcción selección múltiple.

## 4.- Fases en el desarrollo y ejecución del software.

### 4.2.- Diseño estructurado

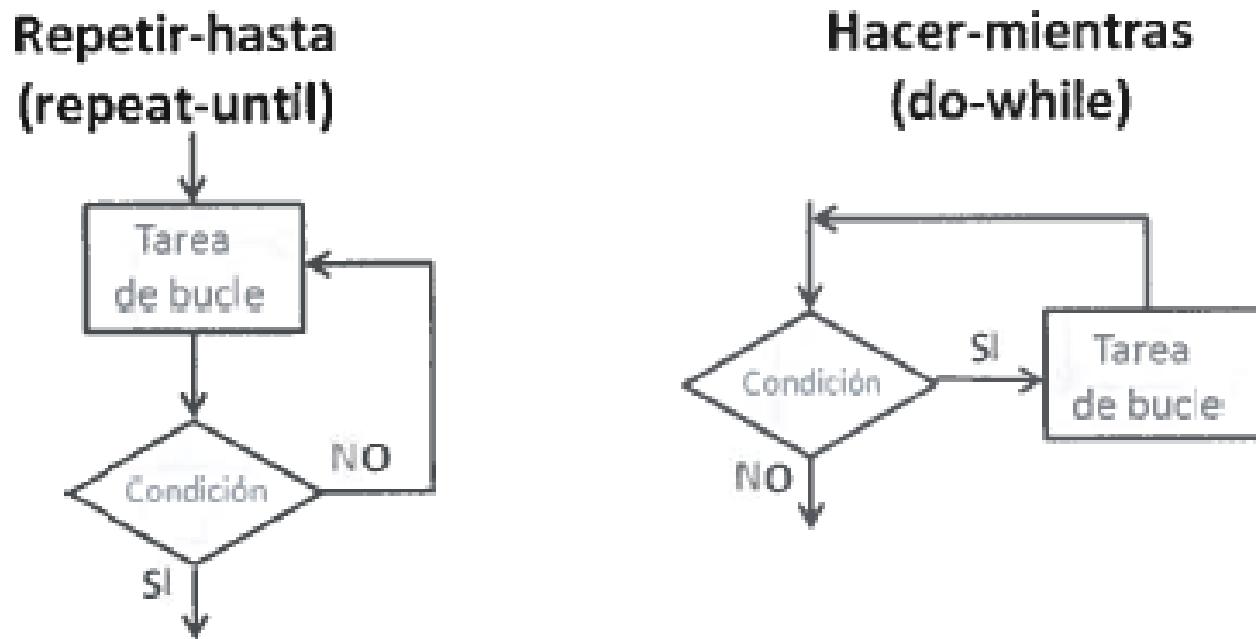


Figura 1.14. Construcciones repetitivas.

## 4.- Fases en el desarrollo y ejecución del software.

### 4.2.- Diseño estructurado

- ❑ El **diseño estructurado** que está basado en el flujo de los datos a través del sistema.
- ❑ Produce un modelo de diseño en cuatro capas.
- ❑ Notaciones graficas para el diseño:
  - Diagrama de flujo de datos.
  - Diagrama de cajas
  - Tablas de decisión
  - Pseudocódigo



Figura 1.11. Modelo de diseño.

## **4.- Fases en el desarrollo y ejecución del software.**

### **4.2.- Diseño estructurado**

#### **□ Diagrama de Flujo**

Es una herramienta muy usada para el diseño procedural. Se utilizan los símbolos vistos anteriormente:

Una caja indica un paso del proceso.



Un rombo representa una condición lógica.



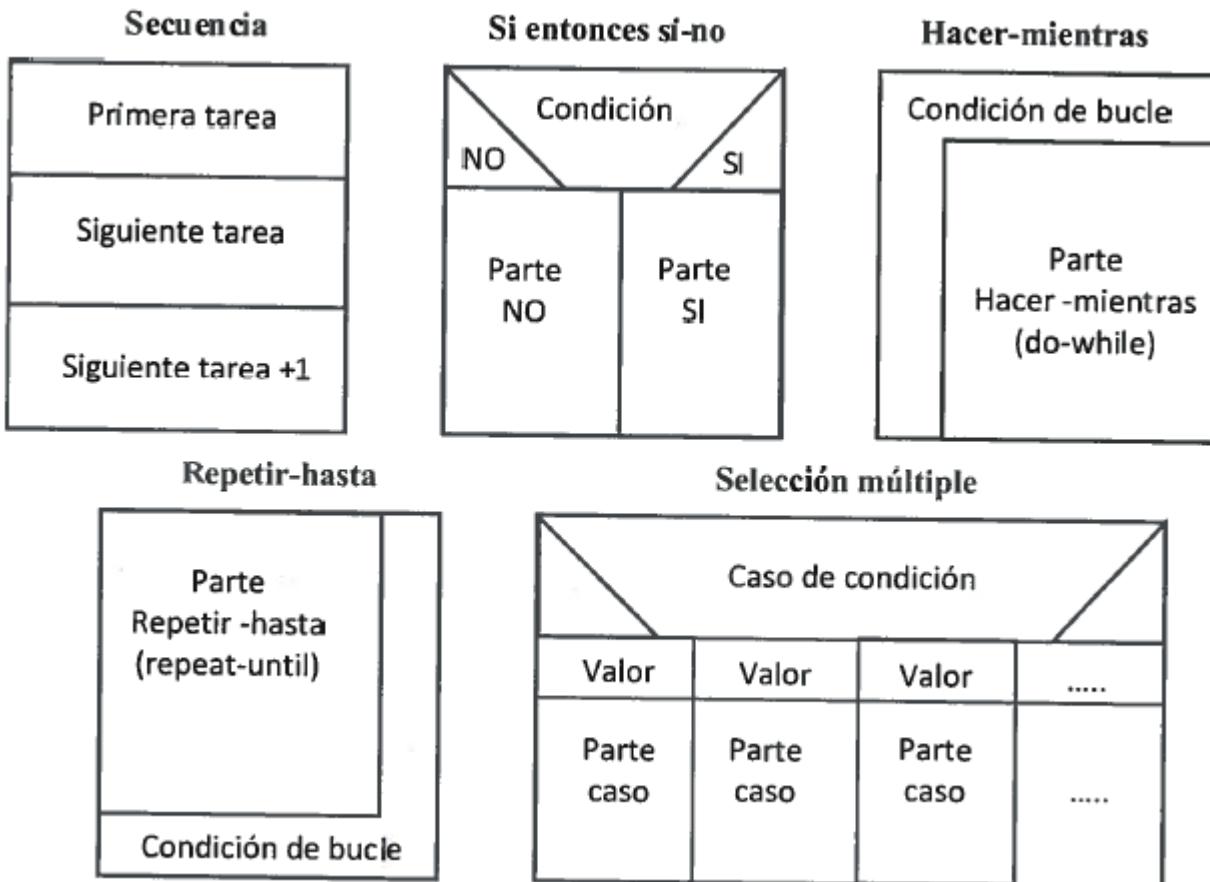
Las flechas indican el flujo de control.



## 4.- Fases en el desarrollo y ejecución del software.

### 4.2.- Diseño estructurado

#### □ Diagrama de cajas



## 4.- Fases en el desarrollo y ejecución del software.

### 4.2.- Diseño estructurado

#### □ Tablas de decisión

Condiciones	1	2	3	4	n
Condición nº 1	✓		✓		
Condición nº 2		✓	✓		
Condición nº 3	✓			✓	
Acciones					
Acción nº 1	✓			✓	
Acción nº 2	✓	✓			
Acción nº 3		✓	✓	✓	
Acción nº 4			✓	✓	

#### □ Ejemplo

Condiciones	1	2	3	4
Cliente registrado	SI	SI	NO	NO
Importe compra > 800 €	SI	NO	SI	NO
Acciones				
Aplicar 1% Bonificación sobre el importe compra	✓	✓		
Aplicar 3% Descuento sobre el importe compra	✓		✓	
Calcular Factura	✓	✓	✓	✓

## 4.- Fases en el desarrollo y ejecución del software.

### 4.2.- Diseño estructurado

#### □ Pseudocódigo

Secuencial	Instrucción 1 Instrucción 2 ..... Instrucción n
Condicional	<b>Si</b> <condición> <b>Entonces</b> <instrucciones> <b>Si no</b> <instrucciones> <b>Fin si</b>
Condicional múltiple.	<b>Según</b> sea <variable> <b>Hacer</b> <b>Caso</b> valor 1: <instrucciones> <b>Caso</b> valor 2: <instrucciones> <b>Caso</b> valor 3: <instrucciones> <b>Otro caso:</b> <instrucciones> <b>Fin según</b>
Repetir-hasta	<b>Repetir</b> <instrucciones> <b>Hasta que</b> <condición >
Hacer-mientras	<b>Mientras</b> <condición> <b>Hacer</b> <instrucciones> <b>Fin mientras</b>

#### □ Ejemplo

##### **Inicio**

```
Abrir Fichero
Leer Registro del Fichero
Mientras no sea Fin de Fichero Hacer
    Procesar Registro leido
    Leer Registro del Fichero
Fin mientras
Cerrar Fichero
Fin.
```

## 4.- Fases en el desarrollo y ejecución del software.

### 4.2.- Diseño Orientado a Objetos

- ❑ **El diseño orientado a objetos (OO):** 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; los objetos interactúan de manera formal con otros objetos.
- ❑ Es difícil
- ❑ Partimos del Análisis Orientado a Objetos(AOO).
- ❑ Define 4 capas de diseño:
  - ❑ 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 la jerarquía de clases requerida para implementar un sistema.
  - ❑ Mensajes. Indica cómo se realiza la colaboración entre los objetos.
  - ❑ Responsabilidades. Identifica las operaciones y atributos que caracterizan cada clase.

## 4.- Fases en el desarrollo y ejecución del software.

### 4.2.- Diseño Orientado a Objetos

- Para el análisis y diseño orientado a objetos se **utiliza UML (Unified Modeling Language -Lenguaje de Modelado Unificado)**. Es un lenguaje de modelado basado en diagramas que sirve para expresar modelos.

```
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  
}
```

## **4.- Fases en el desarrollo y ejecución del software.**

### **4.3.- Codificación.Tipos de código.**

- Durante la fase de codificación se realiza el proceso de programación.
- 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.
- 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.

## **4.- Fases en el desarrollo y ejecución del software.**

### **4.3.- Codificación.Tipos de código.**

- Durante esta fase, el código pasa por diferentes estados:
  - **Código Fuente:** es el escrito por los programadores.
  - **Código Objeto:** es el código binario resultado de compilar el código fuente.
  - **Código Ejecutable:** Es el código binario resultante de enlazar los archivos de código objeto con ciertas rutinas y bibliotecas necesarias.

## **4.- Fases en el desarrollo y ejecución del software.**

### **4.4.- Fases en la obtención de código.**

#### **□ Código Fuente**

- Conjunto de instrucciones que la computadora deberá realizar, escritas por los programadores en algún lenguaje de alto nivel.
- **Pasos:**
  1. Partir del análisis y diseño.
  2. Diseñar un algoritmo que simbolice los pasos a seguir para la resolución del problema.
  3. Elegir una Lenguajes de Programación de alto nivel.
  4. Codificar el algoritmo antes diseñado.
- **Tipos de código fuente:**
  - Abierto. El usuario pueda estudiarlo, modificarlo o reutilizarlo.
  - Cerrado. Es aquel que no tenemos permiso para editarlo.

## 4.- Fases en el desarrollo y ejecución del software.

### 4.4.- Fases en la obtención de código.

#### □ Código Objeto

- Código que se obtiene al traducir el código fuente a binario. Este todavía no es ejecutable. Se llama Bytecode.

#### □ Hay 2 formas de obtenerlo:

a) **Compilación:** El proceso de traducción se realiza sobre todo el código fuente, en un solo paso. Se crea código objeto que habrá que enlazar. El software responsable se llama compilado

b) **Interpretación:** El proceso de traducción del código fuente se realiza línea a línea y se ejecuta simultáneamente. No existe código objeto intermedio. El software responsable se llama intérprete

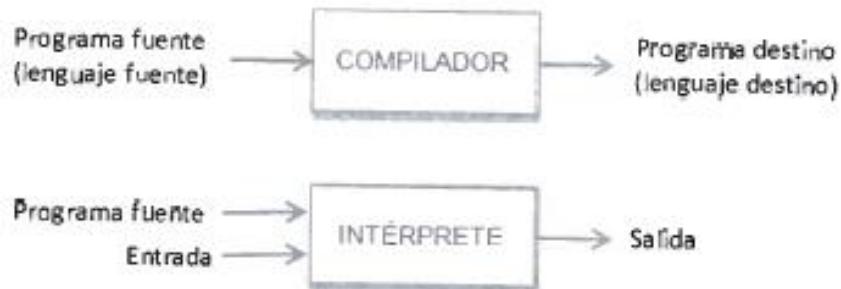


Figura 1.28. Compilador e Intérprete.

## 4.- Fases en el desarrollo y ejecución del software.

### 4.4.- Fases en la obtención de código.

#### □ Código Ejecutable

- Es el resultado de enlazar los archivos de código objeto.
- Consta de un único archivo que puede ser directamente ejecutado por la computadora.
- No necesita ninguna aplicación externa.
- Este archivo es ejecutado y controlado por el sistema operativo.
- El software encargado de obtenerlo se llamará linker (Enlazador. Pequeño software encargado de unir archivos para generar un programa ejecutable.)

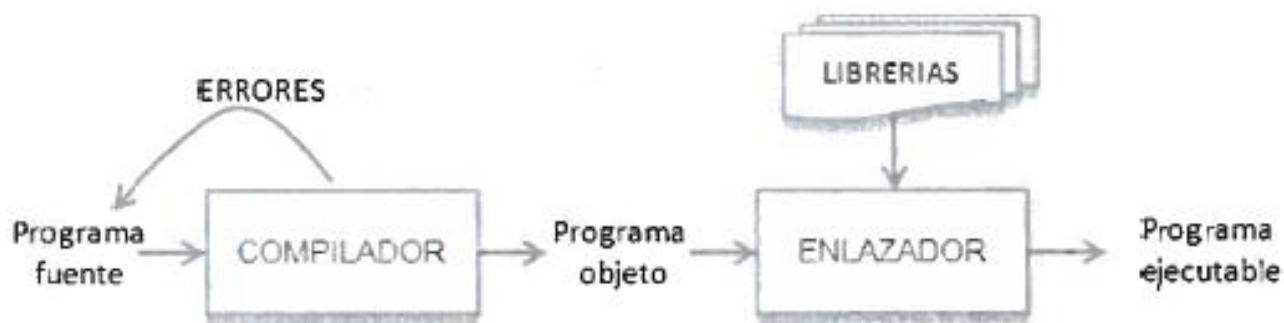
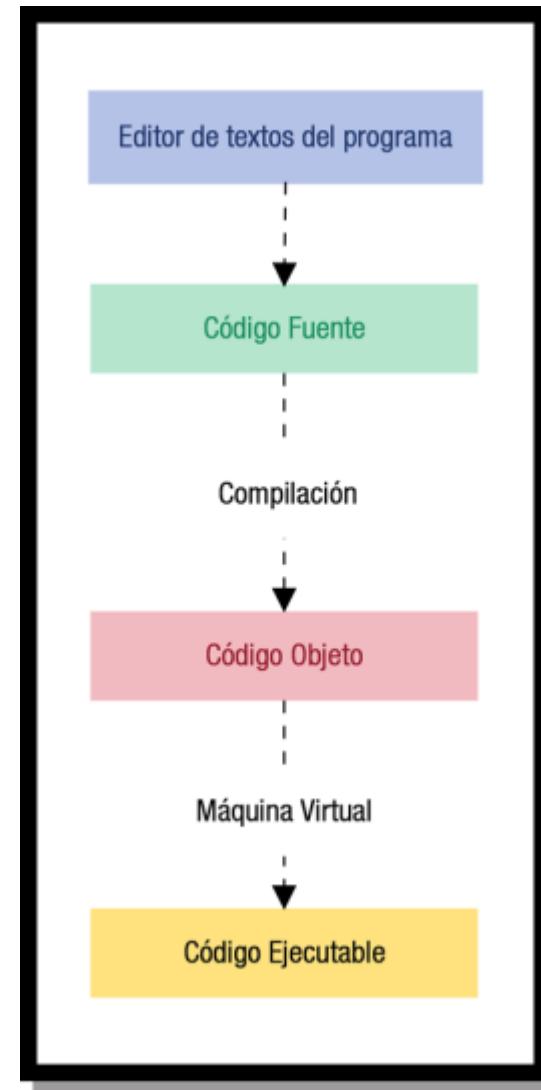


Figura 1.32. Proceso de compilación.

## 4.- Fases en el desarrollo y ejecución del software.

### 4.4.- Fases en la obtención de código.

Esquema de generación de código ejecutable



## **4.- Fases en el desarrollo y ejecución del software.**

### **4.5.- Máquinas virtuales.**

- Tipo especial de software cuya misión es separar el funcionamiento del ordenador de los componentes hardware instalados.**
- Funciones principales:**
  - Conseguir aplicaciones portables.
  - Reservar y liberar la memoria.
  - Comunicarse con el sistema para el control de los dispositivos hardware implicados.
  - Cumplimiento de las normas de seguridad de las aplicaciones.

## 4.- Fases en el desarrollo y ejecución del software.

### 4.5.- Máquinas virtuales.

- ❑ Características de la máquina virtual
  - Aísla la aplicación de los detalles físicos del equipo en cuestión.
  - Verifica todo el bytecode antes de ejecutarlo.
  - Protege las direcciones de memoria.



Figura 1.34. Compilación y ejecución de un programa Java.

## 4.- Fases en el desarrollo y ejecución del software.

### 4.5.- Máquinas virtuales.

- ❑ Las tareas principales de la Máquina Virtual 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 ciertas funciones, como controlar el acceso a dispositivos hardware.
  - Vigilar el cumplimiento de las normas de seguridad de las aplicaciones Java.

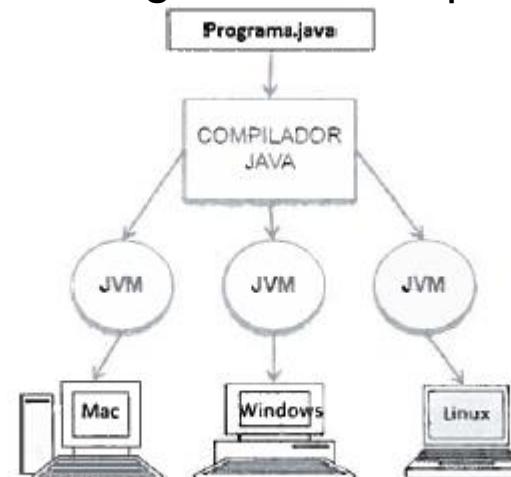


Figura 1.35. Ejecución de un programa Java en distintas plataformas.

## **4.- Fases en el desarrollo y ejecución del software.**

### **4.5.1.- Frameworks.**

- ❑ Un **frameworks** o **entorno de desarrollo** es una plataforma software que ayuda al programador a desarrollar programas.
  
- ❑ Ventajas de utilizar un framework:
  - Desarrollo rápido de software.
  - Reutilización de partes de código
  - Diseño uniforme del software
  - Portabilidad de aplicaciones
  
- ❑ Inconvenientes:
  - Gran dependencia del código respecto al framework utilizado.
  - Consumen muchos recursos de la máquina.

## **4.- Fases en el desarrollo y ejecución del software.**

### **4.5.1.- Frameworks.**

- ❑ Un **frameworks** o **entorno de desarrollo** es una plataforma software que ayuda al programador a desarrollar programas.
- ❑ Ventajas de utilizar un framework:
  - Desarrollo rápido de software.
  - Reutilización de partes de código
  - Diseño uniforme del software
  - Portabilidad de aplicaciones
- ❑ Inconvenientes:
  - Gran dependencia del código respecto al framework utilizado.
  - Consumen muchos recursos de la máquina.
- ❑ Ejemplos: .NET, Spring de Java

## 4.- Fases en el desarrollo y ejecución del software.

### 4.5.2.- Entornos de ejecución.

- ❑ Un **entorno de ejecución** es un servicio de máquina virtual que sirve como base software para la ejecución de programas. A veces pertenece al SO, pero también se puede instalar como software independiente.
- ❑ Durante la ejecución, los entornos se encargarán de:
  - Configurar la memoria principal disponible en el sistema.
  - Enlazar los archivos del programa con las bibliotecas existentes y con los subprogramas creados.
  - Depurar los programas: comprobar la existencia (o no existencia) de errores semánticos del lenguaje.
- ❑ Ejemplo: Java runtime environment.



## 4.6.- Pruebas.

- ❑ La realización de pruebas es imprescindible para asegurar la validación y verificación del software construido.
- ❑ Entre todas las pruebas que se efectúan sobre el software podemos distinguir básicamente:
  - **Pruebas unitarias:** Junit en Java
  - **Pruebas de integración**

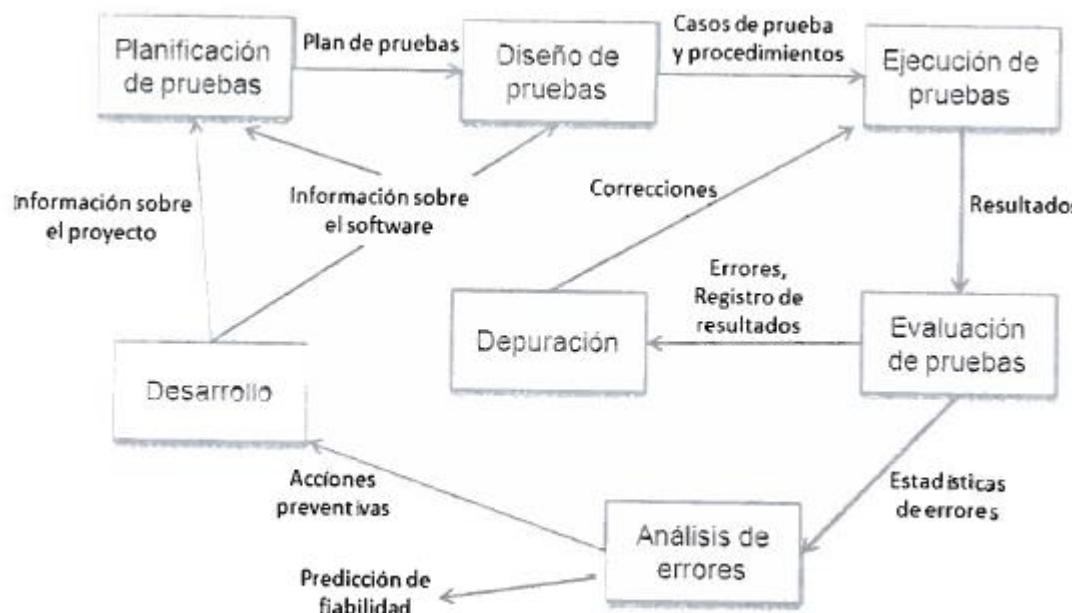


Figura 1.16. Flujo de proceso para probar el software.

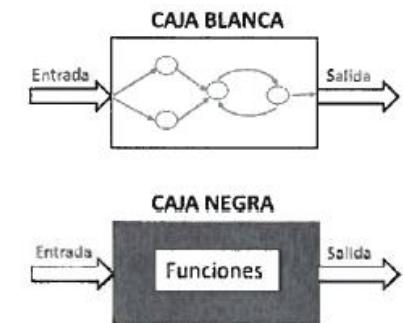


Figura 1.17. Pruebas de caja blanca y negra.

## 4.7.- Documentación.

Documentos a elaborar en el proceso de desarrollo de software			
	GUÍA TÉCNICA	GUÍA DE USO	GUÍA DE INSTALACIÓN
Quedan reflejados:	<ul style="list-style-type: none"> <li>• El diseño de la aplicación.</li> <li>• La codificación de los programas.</li> <li>• Las pruebas realizadas.</li> </ul>	<ul style="list-style-type: none"> <li>• Descripción de la funcionalidad de la aplicación.</li> <li>• Forma de comenzar a ejecutar la aplicación.</li> <li>• Ejemplos de uso del programa.</li> <li>• Requerimientos software de la aplicación.</li> <li>• Solución de los posibles problemas que se pueden presentar.</li> </ul>	Toda la información necesaria para: <ul style="list-style-type: none"> <li>• Puesta en marcha.</li> <li>• Explotación.</li> <li>• Seguridad del sistema.</li> </ul>
¿A quién va dirigido?	Al personal técnico en informática (analistas y programadores).	A los usuarios que van a usar la aplicación (clientes).	Al personal informático responsable de la instalación, en colaboración con los usuarios que van a usar la aplicación (clientes).
¿Cuál es su objetivo?	Facilitar un correcto desarrollo, realizar correcciones en los programas y permitir un mantenimiento futuro.	Dar a los usuarios finales toda la información necesaria para utilizar la aplicación.	Dar toda la información necesaria para garantizar que la implantación de la aplicación se realice de forma segura, confiable y precisa.

## 4.8.- Explotación.

- ❑ **La explotación** es la fase en que los usuarios finales conocen la aplicación y comienzan a utilizarla.
- ❑ La explotación es la instalación, puesta a punto y funcionamiento de la aplicación en el equipo final del cliente.
- ❑ En esta etapa se llevan a cabo las siguientes tareas:
  - Se define la estrategia para la implementación del proceso.
  - Instalación del software
  - Pruebas de operación, **Beta Test**.
  - Configuración del software.
  - Uso operacional del sistema.
  - Soporte al usuario.

## 4.9.- Mantenimiento.

- ❑ Esta etapa es la más larga de todo el ciclo de vida del software.
- ❑ Motivos:
  - Por su naturaleza, el software es cambiante y deberá actualizarse y evolucionar con el tiempo.
  - Corregir errores.
  - Realizar nuevas versiones que lleven a cabo mejoras de la aplicación.

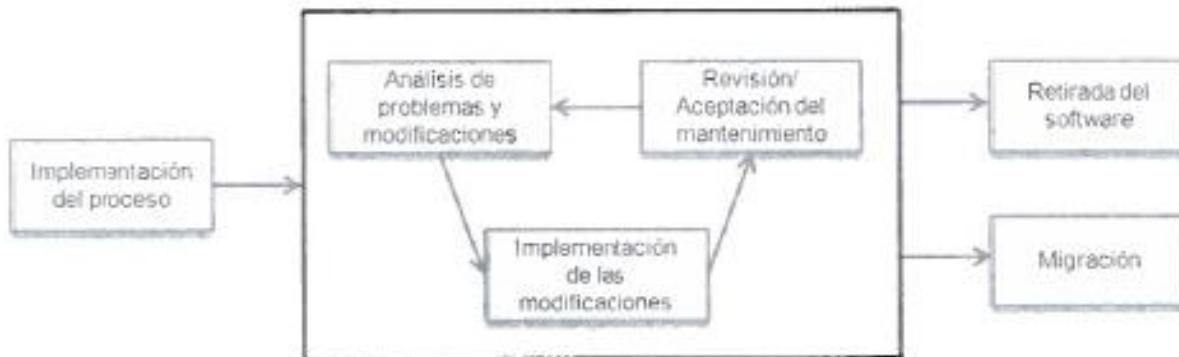


Figura 1.20. Tareas en el mantenimiento del software.

## 4.9.- Mantenimiento.

- ❑ Los tipos de cambios que hacen necesario el mantenimiento del software son los siguientes:
  - **Perfectivos:** Para mejorar la funcionalidad del software.
  - **Evolutivos:** El cliente tendrá en el futuro nuevas necesidades
  - **Adaptativos:** Modificaciones, actualizaciones... para adaptarse a las nuevas tendencias del mercado, a nuevos componentes hardware, etc.
  - **Correctivos:** La aplicación tendrá errores en el futuro.



Figura 1.19. Tipos de mantenimiento del software.

## **6.- LENGUAJES DE PROGRAMACIÓN.**

### **6.1.- Concepto y características.**

- La elección del lenguaje de programación** para codificar un programa dependerá de las características del problema a resolver.
  
- Un lenguaje de programación** es el conjunto de:
  - Alfabeto: conjunto de símbolos permitidos.
  - Sintaxis: normas de construcción permitidas de los símbolos del lenguaje.
  - Semántica: significado de las construcciones para hacer acciones válidas.

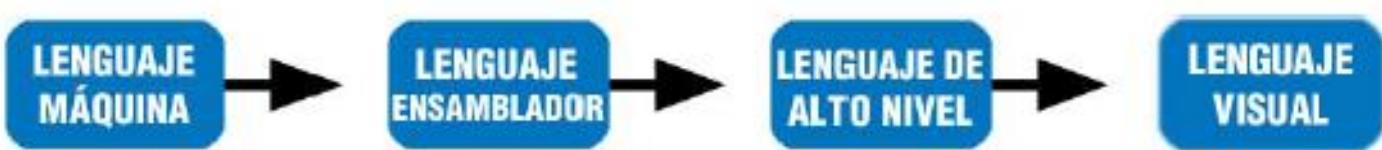
## **6.- LENGUAJES DE PROGRAMACIÓN.**

**I. Los lenguajes de programación se pueden clasificar según varios criterios:**

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

# 6.- LENGUAJES DE PROGRAMACIÓN.

## I. Según su nivel de abstracción:



## 2. Características de los Lenguajes de Programación

### **Lenguaje de bajo nivel:**

#### **Lenguaje de máquina:**

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

# 6.- LENGUAJES DE PROGRAMACIÓN.

## ○ **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.
- Es difícil de utilizar.

# **6.- LENGUAJES DE PROGRAMACIÓN.**

## **□ Lenguaje de nivel medio:**

- Sustituyeron al lenguaje ensamblador para facilitar más la labor de programación.
- Tienen características de bajo y alto nivel
- Ejemplo C en SO

## **□ Lenguaje de alto nivel basados en código:**

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

## **6.- LENGUAJES DE PROGRAMACIÓN.**

### **□ Lenguajes visuales:**

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

# 6.- LENGUAJES DE PROGRAMACIÓN.

## I. Según su forma de ejecución:

### Compilados:



### Interpretados:



# 6.- LENGUAJES DE PROGRAMACIÓN.

## I. Según el paradigma de Programación:

### Imperativos:

- Instrucciones simples
- En esta categoría están los de programación estructurada, modular y OO
- Ejemplos C, C++, Java, Pascal

### Funcionales:

- Basados en el concepto de función.
- 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 comparación se realiza mediante al evaluación de expresiones.

### Logicos:

- Son una lista de declaraciones lógicas
- Se ejecutan haciendo preguntas de forma interactiva
- Ejemplo: Prolog
- Sistemas expertos, procesamiento de lenguaje natural

# 6.- LENGUAJES DE PROGRAMACIÓN.

## I. Según el paradigma de Programación:

- De programación estructurada



Figura 1.30. Programa dividido en módulos.

- De programación OO

```
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  
}
```

# **6.- LENGUAJES DE PROGRAMACIÓN.**

## **6.2.- Lenguajes de programación estructurados.**

□ La **programación estructurada** se define como una técnica para escribir lenguajes de programación que permite sólo el uso de tres tipos de sentencias o estructuras de control:

- Sentencias secuenciales.
- Sentencias selectivas (condicionales).
- Sentencias repetitivas (iteraciones o bucles).

### **VENTAJAS DE LA PROGRAMACIÓN ESTRUCTURADA**

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

### **INCONVENIENTES**

- Todo el programa se concentra en un único bloque.
- No permite reutilización eficaz de código, ya que todo va "en uno". Es por esto que a la programación estructurada le sustituyó la programación modular.
- Ejemplos de lenguajes estructurados: *Pascal, C, Fortran*

## **6.- LENGUAJES DE PROGRAMACIÓN.**

### **6.3.- Lenguajes de programación orientados a objetos.**

- ❑ La P.O.O. los programas se componen de objetos independientes entre sí que colaboran para realizar acciones.
- ❑ Características:
  - Los objetos del programa tendrán una serie de atributos que los diferencian unos de otros.
  - Se define clase como una colección de objetos con características similares.
  - Mediante los llamados métodos, los objetos se comunican con otros produciéndose un cambio de estado de los mismos.
  - Los objetos son, pues, como unidades

# **6.- LENGUAJES DE PROGRAMACIÓN.**

## **6.3.- Lenguajes de programación orientados a objetos.**

## VENTAJAS

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

## INCONVENIENTES

- No es una programación tan intuitiva como la estructurada.

□ Ejemplos de lenguajes OO: *Ada, C++, VB.NET, Delphi, Java, PowerBuilder, etc.*