



UD1: DESARROLLO DE SOFTWARE

ENTORNOS DE DESARROLLO



Arangoya
1974

Ander González Ortiz
C.E. Arangoya
ander.gonzalez@arangoya.net

CONTENIDOS (I)

1. Introducción

- Mapa conceptual
- Software y hardware

2. El software de ordenador

- Basado en el tipo de tarea que realiza
- Basado en el método de distribución
- Licencias de software. Software libre y propietario

3. Ciclo de vida del software

- Definición
- Modelos de ciclo de vida

4. Fases del desarrollo de una aplicación

- Análisis
- Diseño
- Codificación
- Pruebas
- Documentación
- Explotación
- Mantenimiento

5. Concepto de programa

- Componentes del sistema informático

CONTENIDOS (II)

6. Lenguajes de programación

- Clasificación y características

7. Obtención de código ejecutable

- Tipos de código
- Compilación

8. Máquinas virtuales

- La máquina virtual de Java

9. Herramientas utilizadas en programación

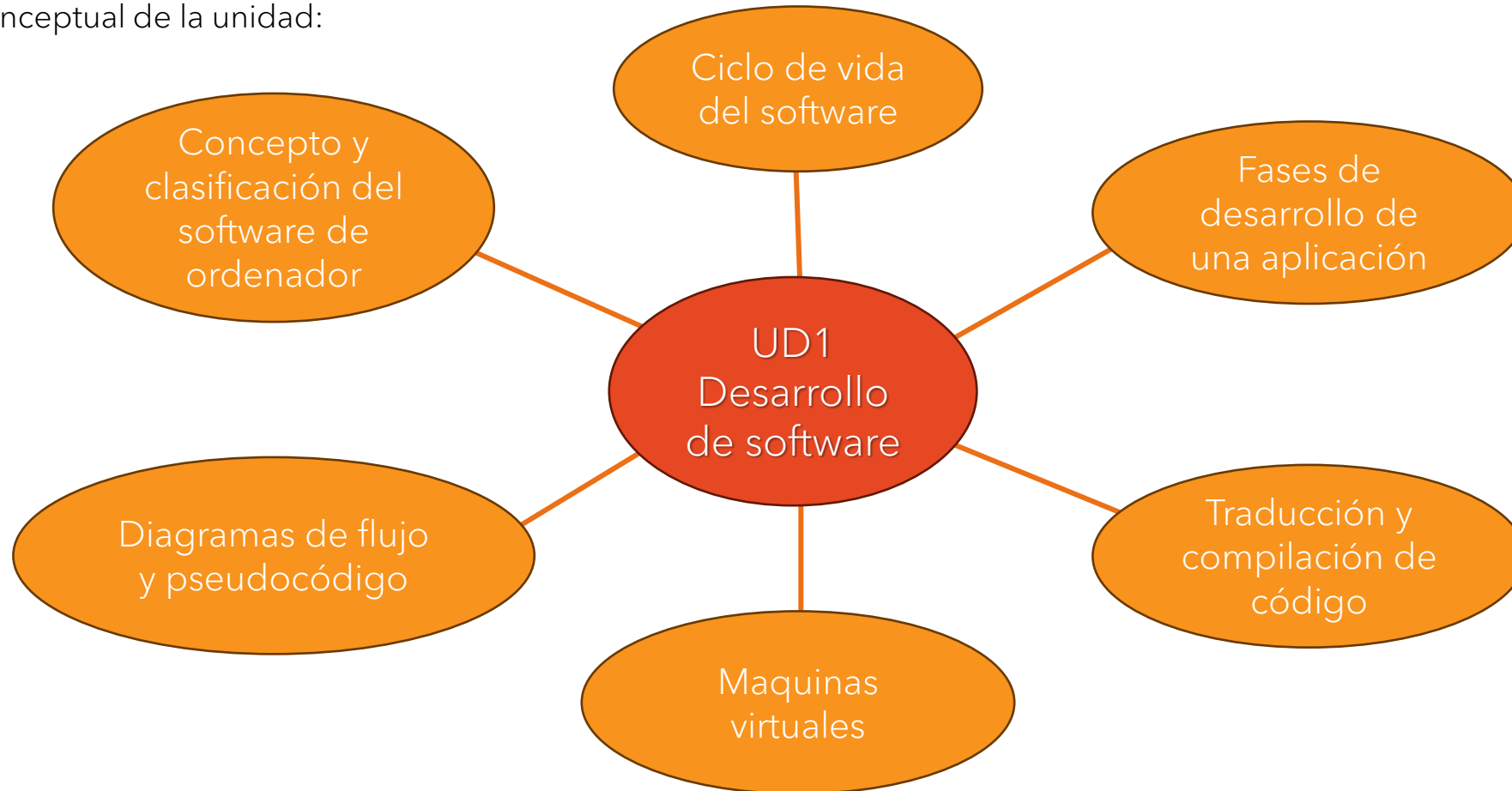
- IDE

10. Diseño de software

- Diagrama de flujo
- Pseudocódigo

1. INTRODUCCIÓN

Mapa conceptual de la unidad:



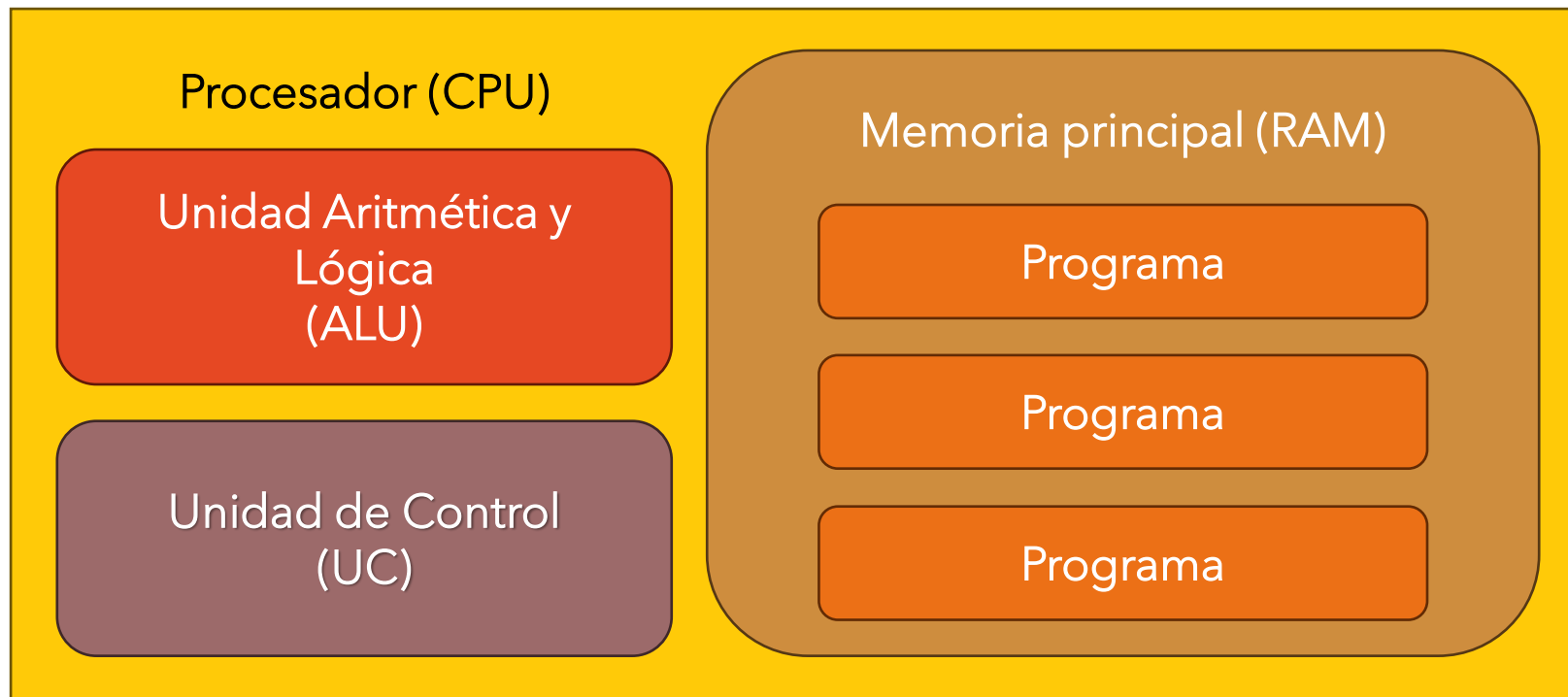
1. INTRODUCCIÓN

¿Qué es un ordenador?

- Los primeros ordenadores aparecen a partir de la década de los 40
- Un ordenador acepta **datos de entrada**, los **procesa** y produce unas **salidas** (resultados)
- Se componen de **elementos físicos** (hardware) y **lógicos** (programas, software)
- Las órdenes de los usuarios deben ser **traducidas** para que las entienda el ordenador

1. INTRODUCCIÓN

¿Qué es un ordenador?



2. EL SOFTWARE DEL ORDENADOR

Podemos definir la palabra software como:

“el conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora”

El software son los programas encargados de dar instrucciones para realizar tareas con el hardware.

El software se puede clasificar según:

- El tipo de tarea que realiza
- El método de distribución
- La licencia de distribución

2. EL SOFTWARE DEL ORDENADOR

Si clasificamos el software según el **tipo de tarea realiza**, se distinguen **cinco tipos**:

- **Software de sistema:** es aquel que permite que el hardware funcione. Administran la parte física o los recursos del ordenador y es el que interactúa entre los usuarios y los componentes hardware del ordenador
 - Sistemas operativos, controladores de dispositivos, herramientas de diagnóstico...
- **Software de aplicación:** ayudan a realizar tareas específicas en cualquier campo susceptible de ser automatizado o asistido. Hacen que el ordenador sea una herramienta útil para el usuario
 - Aplicaciones ofimáticas, automatización industrial, aplicaciones de contabilidad, diseño asistido (CAD)...
- **Software de programación o desarrollo:** proporciona al programador herramientas para ayudarle a desarrollar programas informáticos y hacer uso de los lenguajes de programación de forma práctica
 - Entornos de desarrollo integrado (IDE)

2. EL SOFTWARE DEL ORDENADOR

- **Software multimedia:** se distribuye con el objetivo de presentar de una forma integrada textos, gráficos, sonidos y animaciones.
 - MovieMaker, Adobe Premier Pro...
- **Software de uso específico:** es aquel que se desarrolla especialmente para resolver un problema determinado de alguna organización o persona.
 - Aplicación para registro de venta de vehículos, utilizado por un proveedor de vehículos autorizado
 - Software para control de libros, utilizado por una biblioteca.

2. EL SOFTWARE DEL ORDENADOR

Si clasificamos el software según el **método de distribución**, se distinguen **tres tipos**:

- **Shareware**: es aquel que se puede evaluar de forma gratuita por un tiempo específico. Para poder hacer uso del software de manera completa se requiere adquirir una licencia mediante un pago.
 - WinRAR, ZoneAlarm...
- **Freeware**: es distribuye sin cargo. Incluye una licencia de uso que permite su redistribución sin modificación, ni venta y dando cuenta del autor.
 - Mozilla Firefox, Google Chrome, VLC y OpenOffice
- **Adware**: suelen ser programas Shareware que de forma automática descargan publicidad en nuestro ordenador cuando los ejecutamos o instalamos, a veces es evitable y la publicidad se elimina al comprar el programa.
 - Barras para navegadores que añaden publicidad

2. EL SOFTWARE DEL ORDENADOR

Una **licencia de software** es un contrato que se establece entre el desarrollador de un software sometido a propiedad intelectual y a derechos de autor y el usuario, en el cual se definen con precisión los derechos y deberes de ambas partes.

El **software libre** es aquél en el cual el autor cede una serie de **libertades básicas** al usuario:

1. Libertad de utilizar el programa con cualquier fin en cuantos ordenadores se desee.
2. Libertad de estudiar cómo funciona el programa y de adaptar su código a necesidades específicas
3. Libertad de distribuir copias a otros usuarios (con o sin modificaciones).
4. Libertad de mejorar el programa (ampliarlo, añadir funciones) y de hacer públicas y distribuir al público las modificaciones.

2. EL SOFTWARE DEL ORDENADOR

El **software propietario** es aquél que se distribuye en formato binario, sin posibilidad de acceso al código fuente y, por regla general, prohíbe alguna o todas las siguientes posibilidades:

- la redistribución, modificación, copia, uso en varias máquinas simultáneamente
- transferencia de titularidad,
- difusión de fallos y errores que se pudiesen descubrir en el programa

Un **software de dominio público** es aquél que carece de licencia o no hay forma de determinarla pues se desconoce al autor. Esta situación se produce bien cuando su propietario abandona los derechos que le acreditan como titular o se produce el fin del plazo de protección de los derechos de autor. No pertenece a una persona concreta, sino que todo el mundo lo puede utilizar.

La licencia más utilizada en los productos y desarrollos de software libre y de fuentes abiertas es la licencia **GPL** (GNU General Public License - Licencia Pública General) que da derecho al usuario a usar y modificar el programa con la obligación de hacer públicas las versiones modificadas de éste.

3. CICLO DE VIDA DEL SOFTWARE

Podemos definir el 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.”

Normalmente se divide en etapas y en cada etapa se realizarán una serie de tareas:

1. **Análisis.** Construye un modelo de los requisitos. En esta etapa se debe entender y comprender de forma detallada el problema que se va a resolver. Es muy importante producir en esta etapa una documentación entendible, completa y fácil de verificar y modificar.
2. **Diseño.** Se define cómo se va a resolver el problema. Se deducen las estructuras de datos, la arquitectura de software, la interfaz de usuario y los procedimientos. Se selecciona el lenguaje de programación, el Sistema Gestor de Bases de Datos, etc.

3. CICLO DE VIDA DEL SOFTWARE

3. **Codificación.** En esta etapa se traduce lo descrito en el diseño a una forma legible por la máquina. La salida de esta fase es código ejecutable.
4. **Pruebas.** Se comprueba que se cumplen criterios de corrección y calidad. Las pruebas deben garantizar el correcto funcionamiento del sistema.
5. **Mantenimiento.** Tiene lugar después de la entrega del software al cliente. En ella hay que asegurar que el sistema pueda adaptarse a los cambios. Se producen cambios porque se han encontrado errores, es necesario adaptarse al entorno (por ejemplo, se ha cambiado de sistema operativo) o porque el cliente requiera mejoras funcionales.

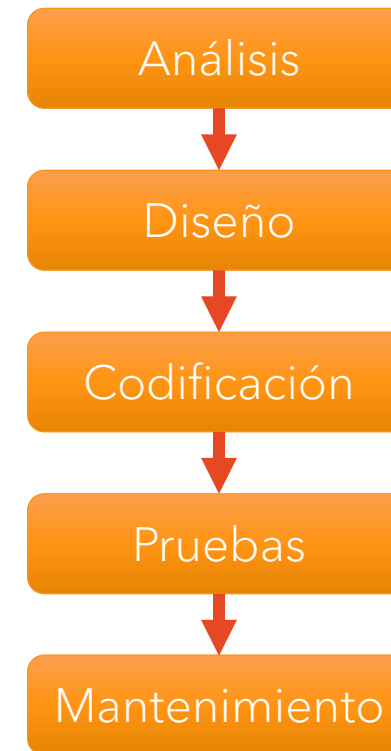
Cada etapa tiene como entrada uno o varios documentos procedentes de las etapas anteriores y produce otros documentos de salida, por ello una tarea importante a realizar en cada etapa es la documentación.

3. CICLO DE VIDA DEL SOFTWARE

Existen distintos modelos de ciclo de vida, es importante tener en cuenta las características del proyecto software para elegir un modelo u otro:

Ciclo de vida en cascada

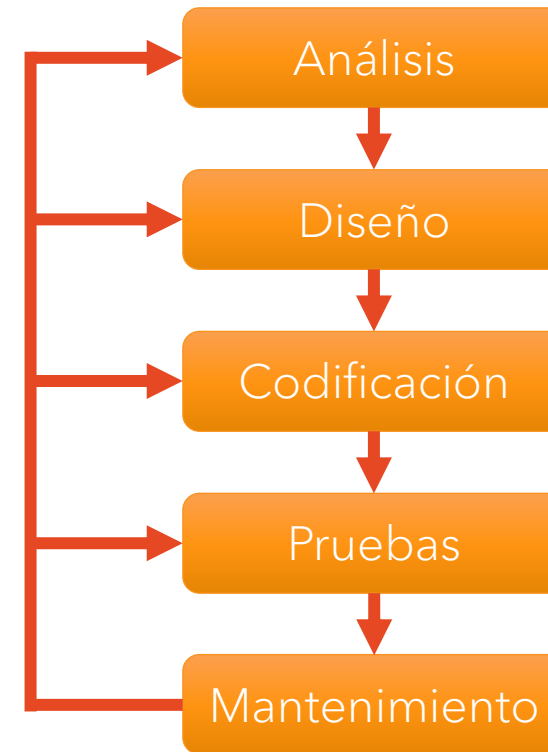
Las etapas del desarrollo de software tienen un orden, de tal forma que **para empezar una etapa es necesario finalizar la etapa anterior**; después de cada etapa se realiza una **revisión** para comprobar si se puede pasar a la siguiente.



3. CICLO DE VIDA DEL SOFTWARE

Este modelo permite hacer iteraciones.

Si durante la etapa de mantenimiento del producto el cliente requiere una mejora, esto implica que hay que modificar algo en el **diseño**, lo cual significa que habrá que hacer cambios en la **codificación** y se tendrán que realizar de nuevo las pruebas, es decir, si se tiene que volver a una de las etapas anteriores hay que **recorrer de nuevo** el resto de las etapas.



3. CICLO DE VIDA DEL SOFTWARE

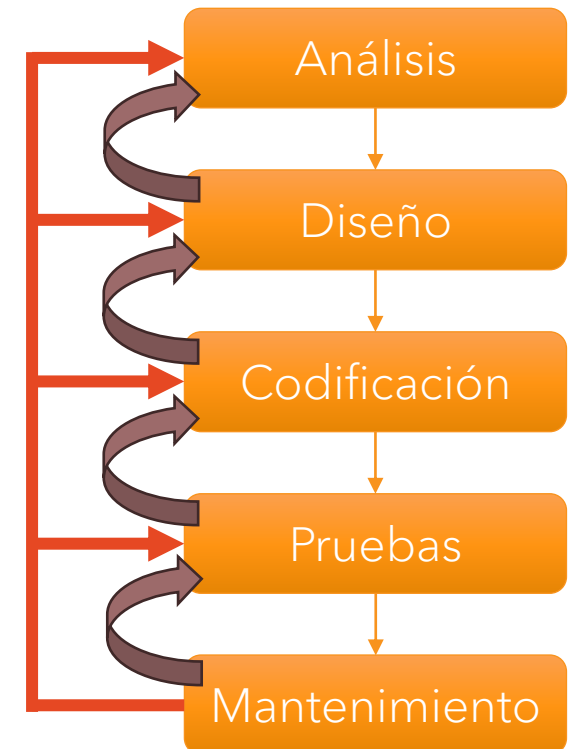
Este modelo tiene varias variantes, una de las más utilizadas es la que produce una realimentación entre etapas, se la conoce como **Modelo en Cascada con Realimentación**.

Ejemplo

Supongamos que la etapa de Análisis (captura de requisitos) ha finalizado y se puede pasar a la de Diseño.

Durante el desarrollo de esta etapa se detectan fallos (los requisitos han cambiado, han evolucionado...) entonces será necesario volver a la etapa anterior, realizar los ajustes pertinentes y continuar de nuevo con el Diseño.

A esto se le conoce como realimentación, pudiendo volver de una etapa a la anterior o incluso varias atrás.



3. CICLO DE VIDA DEL SOFTWARE

MODELO EN CASCADA

Ventajas:	Desventajas:	Se recomienda cuando:
<ul style="list-style-type: none">• Fácil de comprender, planificar y seguir.• La calidad del producto resultante es alta.• Permite trabajar con personal poco cualificado.	<ul style="list-style-type: none">• 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.• Producto no disponible para su uso hasta que no está completamente terminado.	<ul style="list-style-type: none">• 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. CICLO DE VIDA DEL SOFTWARE

MODELOS EVOLUTIVOS

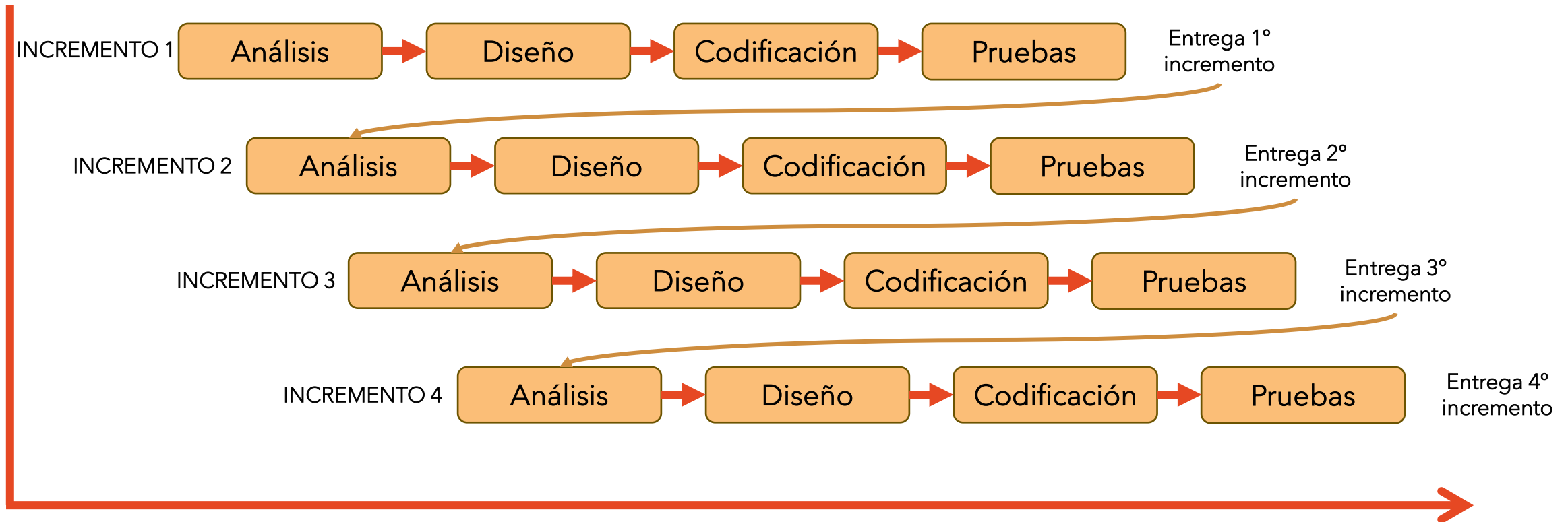
El modelo en cascada asume que se va a entregar un producto completo, en cambio los modelos evolutivos permiten desarrollar versiones cada vez más completas hasta llegar al producto final deseado.

Modelo iterativo incremental

- Basado en varios ciclos cascada realimentados aplicados repetidamente.
- El modelo incremental entrega el software en partes pequeñas, pero utilizables, llamadas «incrementos».
- Cada incremento se construye sobre aquél que ya ha sido entregado.

3. CICLO DE VIDA DEL SOFTWARE

Diagrama del modelo iterativo incremental lineal secuencial:



3. CICLO DE VIDA DEL SOFTWARE

MODELO ITERATIVO INCREMENTAL

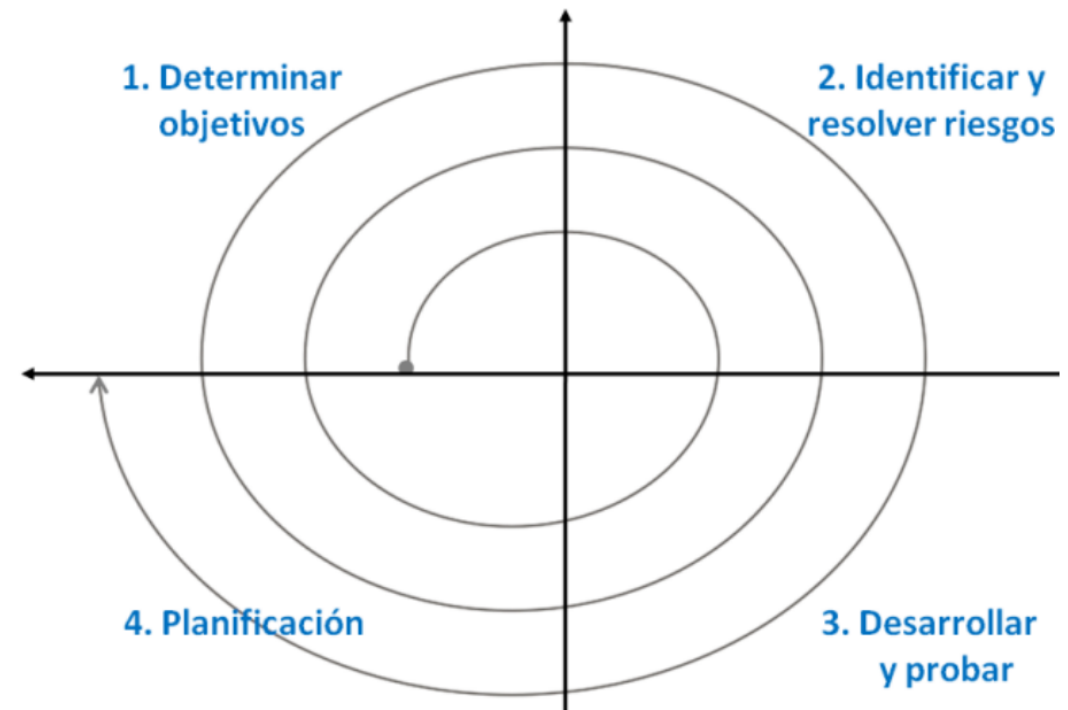
Ventajas:	Desventajas:	Se recomienda cuando:
<ul style="list-style-type: none">• No se necesitan conocer todos los requisitos al comienzo.• Permite la entrega temprana al cliente de partes operativas del software.• Las entregas facilitan la realimentación de los próximos entregables.	<ul style="list-style-type: none">• 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.	<ul style="list-style-type: none">• Los requisitos o el diseño no están completamente definidos y es posible que haya grandes cambios.• Se están probando o introduciendo nuevas tecnologías.

3. CICLO DE VIDA DEL SOFTWARE

Modelo en espiral

El proceso de desarrollo del software se representa como una **espiral**, donde en cada ciclo se desarrolla una parte del mismo. Cada ciclo está formado por **cuatro fases** y cuando termina **produce una versión incremental** del software con respecto al ciclo anterior.

En este aspecto se parece al Modelo Iterativo Incremental con la diferencia que en cada ciclo se tiene en cuenta el **análisis de riesgos**.



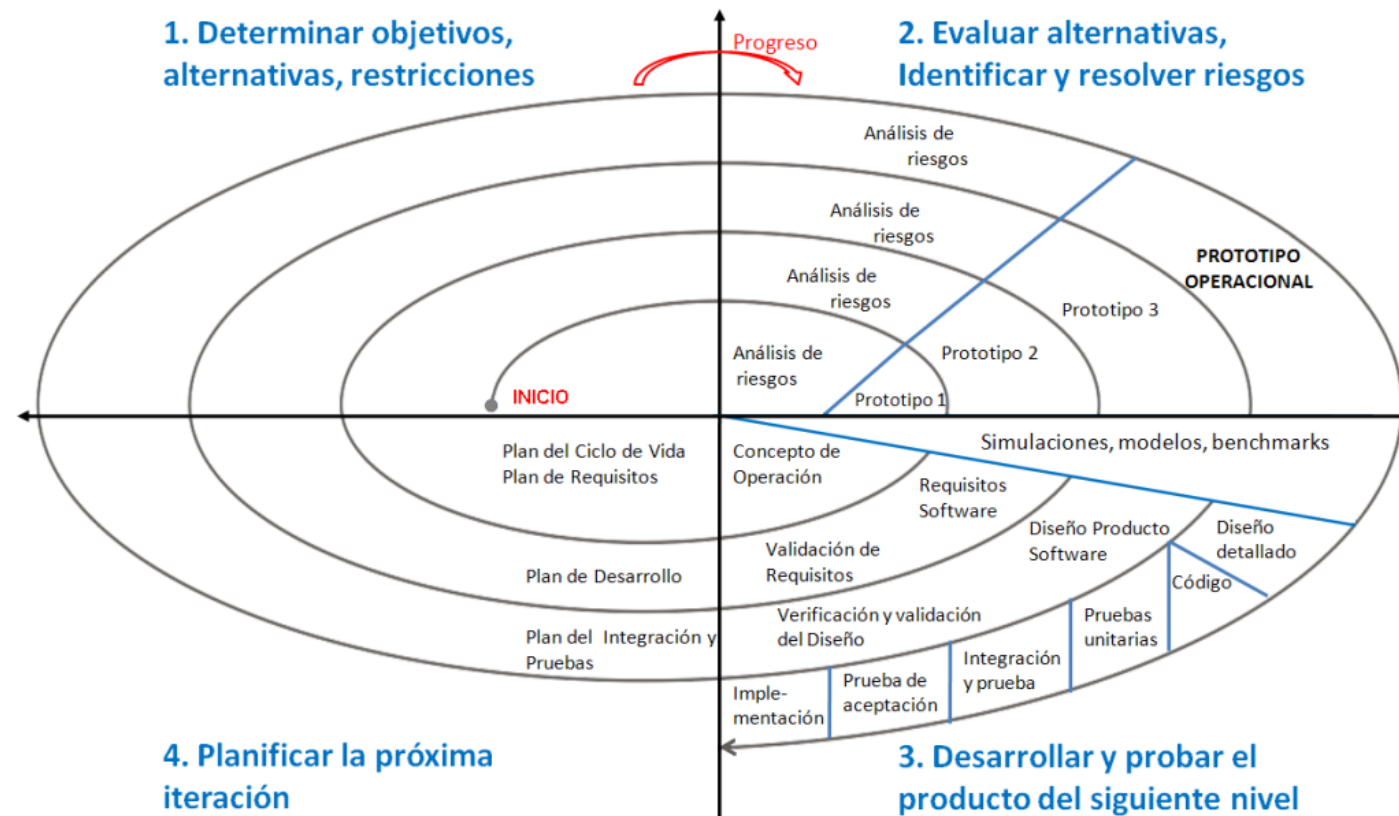
3. CICLO DE VIDA DEL SOFTWARE

Para cada ciclo, los desarrolladores siguen estas fases:

1. **Determinar objetivos.** Cada ciclo de la espiral comienza con la identificación de los objetivos, las alternativas para alcanzar los objetivos (diseño A, diseño B, reutilización, compra...), y las restricciones impuestas a la aplicación de las alternativas (costos, plazos, interfaz...).
2. **Análisis del riesgo.** Se evalúan las alternativas en relación con los objetivos y limitaciones. Con frecuencia, en este proceso se identifican los riesgos involucrados y (si es posible) la manera de resolverlos. Un riesgo puede ser cualquier cosa: requisitos no comprendidos, mal diseño, errores en la implementación, etc.
Utiliza la **construcción de prototipos** (representación limitada de un producto) como mecanismo de reducción de riesgos.
3. **Desarrollar y probar.** Desarrollar la solución al problema en este ciclo, y verificar que es aceptable.
4. **Planificación.** Revisar y evaluar todo lo que se ha hecho, y con ello decidir si se continúa, entonces hay que planificar las fases del ciclo siguiente.

3. CICLO DE VIDA DEL SOFTWARE

Ejemplo del modelo evolutivo en espiral:



3. CICLO DE VIDA DEL SOFTWARE

MODELO EN ESPIRAL

Ventajas:	Desventajas:	Se recomienda cuando:
<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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.	<ul style="list-style-type: none">• Proyectos de gran tamaño y que necesitan constantes cambios.• Proyectos donde sea importante el factor riesgo.

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

Antes de desarrollar un proyecto software hay que elegir un modelo de ciclo de vida, en función de las características del mismo. Independientemente del modelo elegido hay una serie de etapas que se deben seguir para construir un proyecto de calidad:

ANÁLISIS

- En esta fase se analizan y especifican los requisitos o capacidades que el sistema debe tener porque el cliente así lo ha pedido.
- Pueden surgir nuevos requisitos y pueden cambiar lo especificados.
- Para realizar un proyecto satisfactorio es necesario obtener unos buenos requisitos y para ello es esencial una buena comunicación entre el cliente y los desarrolladores.

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

Para facilitar esta comunicación entre el cliente y el desarrollador se pueden utilizar varias técnicas:

- **Entrevistas.** Es la técnica más tradicional que consiste en hablar con el cliente. Hay que tener sobre todo conocimientos de psicología.
- **Desarrollo conjunto de aplicaciones (JAD, Joint Application Development).** Es un tipo de entrevista muy estructurada aplicable a grupos de personas (usuarios, administradores, analistas, desarrolladores, etc). Cada persona juega un rol concreto y todo lo que se hace está reglamentado.
- **Planificación conjunta de requisitos (JRP Joint Requirements Planning).** Se caracterizan por estar dirigidas a la alta dirección y en consecuencia los productos resultantes son los requisitos de alto nivel o estratégicos.
- **Brainstorming.** Es un tipo de reuniones en grupo cuyo objetivo es generar ideas desde diferentes puntos de vista para la resolución de un problema. Su utilización es adecuada al principio del proyecto, pues puede explorar un problema desde muchos puntos de vista.

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

Para facilitar esta comunicación entre el cliente y el desarrollador se pueden utilizar varias técnicas:

- **Prototipos.** Es una versión inicial del sistema, se utiliza para clarificar algunos puntos, demostrar los conceptos. Después se tira o bien se usa como base para añadir más cosas.
- **Casos de uso.** Es la técnica definida en UML (Unified Modeling Language), se basa en escenarios que describen como se usa el software en una determinada situación.

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, etc.
- **Requisitos no funcionales.** Tratan sobre las características del sistema, como puede ser la fiabilidad, mantenibilidad, sistema operativo, plataforma hardware, restricciones, limitaciones, etc.

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

La siguiente tabla muestra un ejemplo de requisitos funcionales y no funcionales para una aplicación de gestión de una agenda de contactos:

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.
El usuario puede eliminar un contacto o varios de la lista.	Utilizar un sistema gestor de base de datos para almacenar los datos.
El usuario puede seleccionar determinados contactos.	Utilizar un lenguaje multiplataforma para el desarrollo de la aplicación.
El usuario puede imprimir la lista de contactos.	Espacio libre en disco, mínimo: 1GB. Mínima cantidad de memoria 2GB.

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

Todo lo realizado en esta fase debe quedar reflejado en el documento de **Especificación de Requisitos del Software (ERS)**, este documento no debe tener ambigüedades, debe ser completo, consistente, fácil de verificar y modificar, fácil de utilizar en la fase de explotación y mantenimiento, y fácil de identificar el origen y las consecuencias de los requisitos.

Sirve como entrada para la siguiente fase en el desarrollo de la aplicación.

El estándar 830 [IEEE, 1998] estructura y especifica su contenido.

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 DEL DESARROLLO DE UNA APLICACIÓN

DISEÑO

Una vez identificados los requisitos es necesario componer la forma en que se solucionará el problema. En esta etapa se traducen los requisitos funcionales y no funcionales en una representación de software.

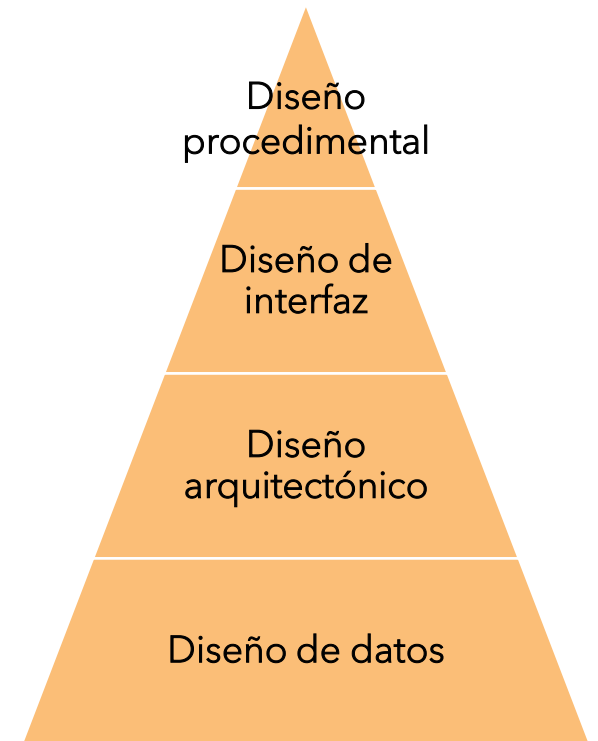
Principalmente hay dos tipos de diseño:

- **Diseño estructurado:** está basado en el flujo de los datos a través del sistema.
- **Diseño orientado a objetos:** el sistema se entiende como un conjunto de objetos que tienen propiedades y comportamientos, y eventos que activan operaciones que modifican el estado de los objetos; los objetos interactúan de manera formal con otros objetos.

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

El diseño estructurado (diseño clásico) produce un modelo de diseño con 4 elementos:






- **Diseño de datos.** Transforma el modelo de dominio de la información creado durante el análisis, en las estructuras de datos que se utilizarán para implementar el software.
- **Diseño arquitectónico.** Se centra en la representación de 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.
- **Diseño a nivel de componentes (diseño procedimental).** Transforma los elementos estructurales de la arquitectura del software en una descripción procedimental de los componentes del software. (Diagramas de flujo, pseudocódigo...)



4. FASES DEL DESARROLLO DE UNA APLICACIÓN

Para representar el diseño se emplean algunas herramientas gráficas como son: diagramas de flujo y pseudocódigo.

Diagramas de flujo

Símbolo	Nombre	Función
	Inicio / Final	Representa el inicio y final de un proceso.
	Línea de flujo	Indica el orden de ejecución de las operaciones.
	Entrada / Salida	Presenta la lectura (entrada) e impresión (salida) de los datos.
	Proceso	Representa un proceso general (asignación, operación aritmética...).
	Decisión	Permite analizar una situación y decidir si es verdadera o falsa.

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

Pseudocódigo

El pseudocódigo utiliza **texto descriptivo** para realizar el diseño de un algoritmo.

Se parece a un lenguaje de programación, ya que mezcla frases en lenguaje natural con estructuras sintácticas que incluyen palabras clave.

Permiten construir las estructuras básicas de la programación estructurada, declarar datos, definir subprogramas y establecer características de modularidad.

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

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

El **diseño de software orientado a objetos (DOO)** se definen todas las clases que son importantes para el problema que se trata de resolver, las operaciones y los atributos asociados, las relaciones y comportamientos y las comunicaciones entre clases.

El diseño orientado a objetos 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 como se realiza la colaboración entre los objetos.
- **Responsabilidades.** Identifica las operaciones y atributos que caracterizan cada clase.

Para el análisis y diseño orientado a objetos se utiliza UML (Unified Modeling Language - Lenguaje de Modelado Unificado).

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

CODIFICACIÓN

En esta etapa, el programador recibe las especificaciones del diseño y las transforma en un conjunto de instrucciones escritas en un lenguaje de programación, almacenadas dentro de un programa, al que se denomina **código fuente**.

A continuación, se describen una serie de **normas de escritura de código fuente** en Java que facilita la lectura de los programas haciendo que sea más sencillo mantenerlos y encontrar errores:

Nombres de ficheros

La extensión para los ficheros de código fuente es .java y para los ficheros compilados es .class.

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

Organización de ficheros

Las secciones en las que se divide el fichero son:

- Comentarios. Todos los ficheros fuente deben comenzar con un comentario que muestra el nombre de la clase, información de la versión, la fecha, y el aviso de derechos de autor.
- Sentencias del tipo package e import. Van después de los comentarios, la sentencia package va delante de import.

Comentarios

Los comentarios deben contener sólo la información que es relevante para la lectura y la comprensión del programa.

```
/*  
 * Nombre de clase  
 *  
 * Información de la versión  
 *  
 * Fecha  
 *  
 * Aviso de Copyright  
 */  
  
package paquete.ejemplo;  
import java.io.*;
```

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

Existen dos tipos de comentarios:

- Los **comentarios de documentación** están destinados a describir la especificación del código. Se utilizan para describir las clases Java, los interfaces, los constructores, los métodos y los campos. Debe aparecer justo antes de la declaración.

```
/**  
 * La clase Ejemplo proporciona ...  
 */  
public class Ejemplo {...
```

- Los **comentarios de implementación** son para comentar algo acerca de la aplicación particular. Pueden ser de 3 tipos:

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

- Comentarios de bloque:

```
/*  
 * Esto es un comentario de bloque  
 */
```

- Comentarios de línea: `/* Esto es un comentario de línea */`

- Comentario corto: `// Esto es un comentario corto`

Declaraciones

- Se recomienda declarar una variable por línea.
- Inicializar las variables locales donde están declaradas y colocarlas al comienzo del bloque.

```
public void miMetodo() {  
    int var1 = 0; //comienza el bloque de miMetodo  
    int var2 = 10;  
    ....  
}
```

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

Secuencias

- Cada línea debe contener una sentencia.
- Si hay un bloque de sentencias, éste debe ser sangrado con respecto a la sentencia que lo genera y debe estar entre llaves, aunque sólo tenga una sentencia.
- Sentencias if-else, if else-if else. Definen bloques a los que se aplican las normas anteriores. Todos los bloques tienen el mismo nivel de sangrado.
- Bucles. Definen un bloque, que sigue las normas anteriores. Si el bucle está vacío no se abren ni se cierran llaves.
- Las sentencias return no deben usar paréntesis.

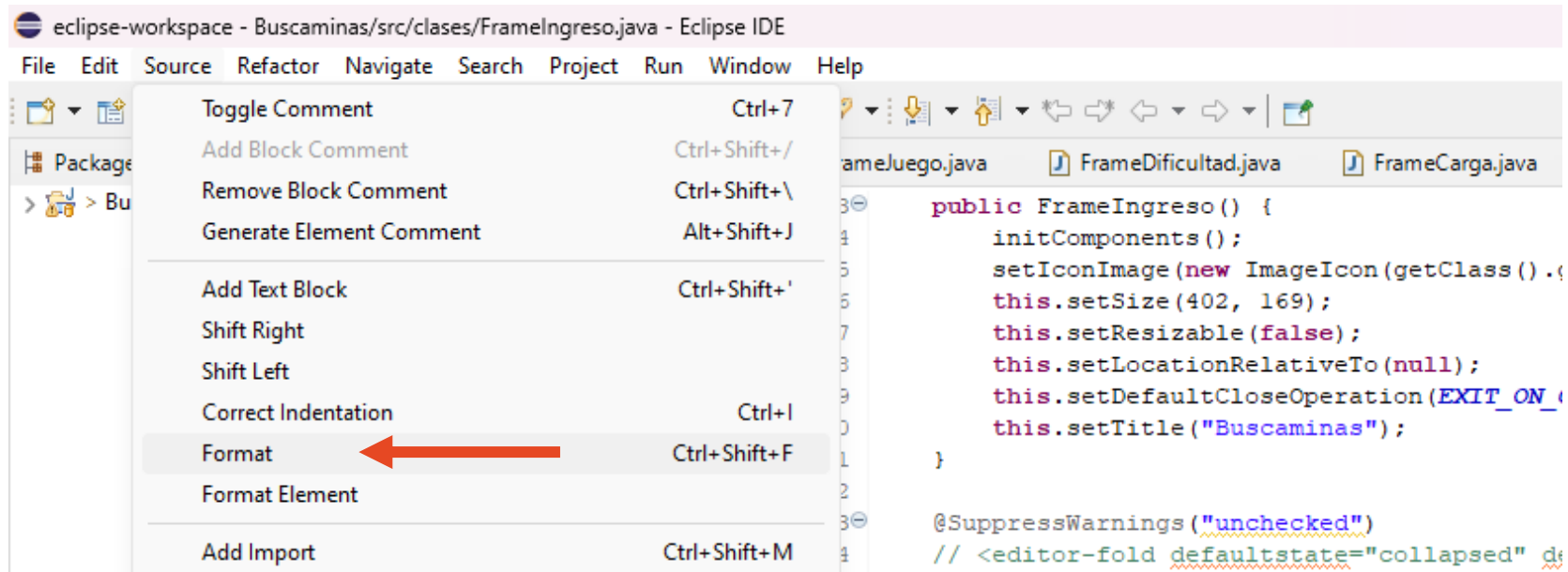
4. FASES DEL DESARROLLO DE UNA APLICACIÓN

Nombres

- **Paquetes:** el nombre se escribe en minúscula, se pueden utilizar puntos para reflejar algún tipo de organización jerárquica. Ejemplo: java.io.*
- **Clases e interfaces:** los nombres deben ser sustantivos. Se deben utilizar nombres descriptivos. Si el nombre está formado por varias palabras la primera letra de cada palabra debe estar en mayúscula: Ejemplo: HiloServidor.
- **Métodos:** se deben usar verbos en infinitivo. Si esta formado por varias palabras el verbo debe estar en minúscula y la siguiente palabra debe empezar en mayúscula. Ejemplo: ejecutar(), verDestino().
- **Variables:** deben ser cortas (pueden ser de una letra) y significativas. Si son varias palabras la primera debe estar en minúscula. Ejemplos: i, j, sumaTotal.
- **Constantes:** el nombre debe ser descriptivo. Se escriben en mayúsculas y si son varias palabras van unidas por un carácter de subrayado. Ejemplo: MAX_VALOR.

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

Las herramientas utilizadas para el desarrollo de los programas suelen ayudar a formatear correctamente el código. Por ejemplo, desde el entorno Eclipse con el fichero Java abierto se puede pulsar en la opción de menú **Source>>Format** para formatear el código.



4. FASES DEL DESARROLLO DE UNA APLICACIÓN

PRUEBAS

En esta etapa ya se dispone del software y se trata de encontrar errores, no solo de codificación sino también relativos a la especificación o el diseño. Durante la prueba del software se realizarán tareas de verificación y validación del software (V&V):

- **La verificación:** se refiere al conjunto de actividades que tratan de comprobar si se está construyendo el producto correctamente, es decir, si el software implementa correctamente una función específica.
- **La validación:** se refiere al conjunto de actividades que tratan de comprobar si es el producto es correcto, es decir, si el software construido se ajusta a los requisitos del cliente.

El objetivo en esta etapa es planificar y diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y de esfuerzo.

Un **caso de prueba** es un documento que especifica los valores de entrada, salida esperada y las condiciones previas para la ejecución de la prueba.

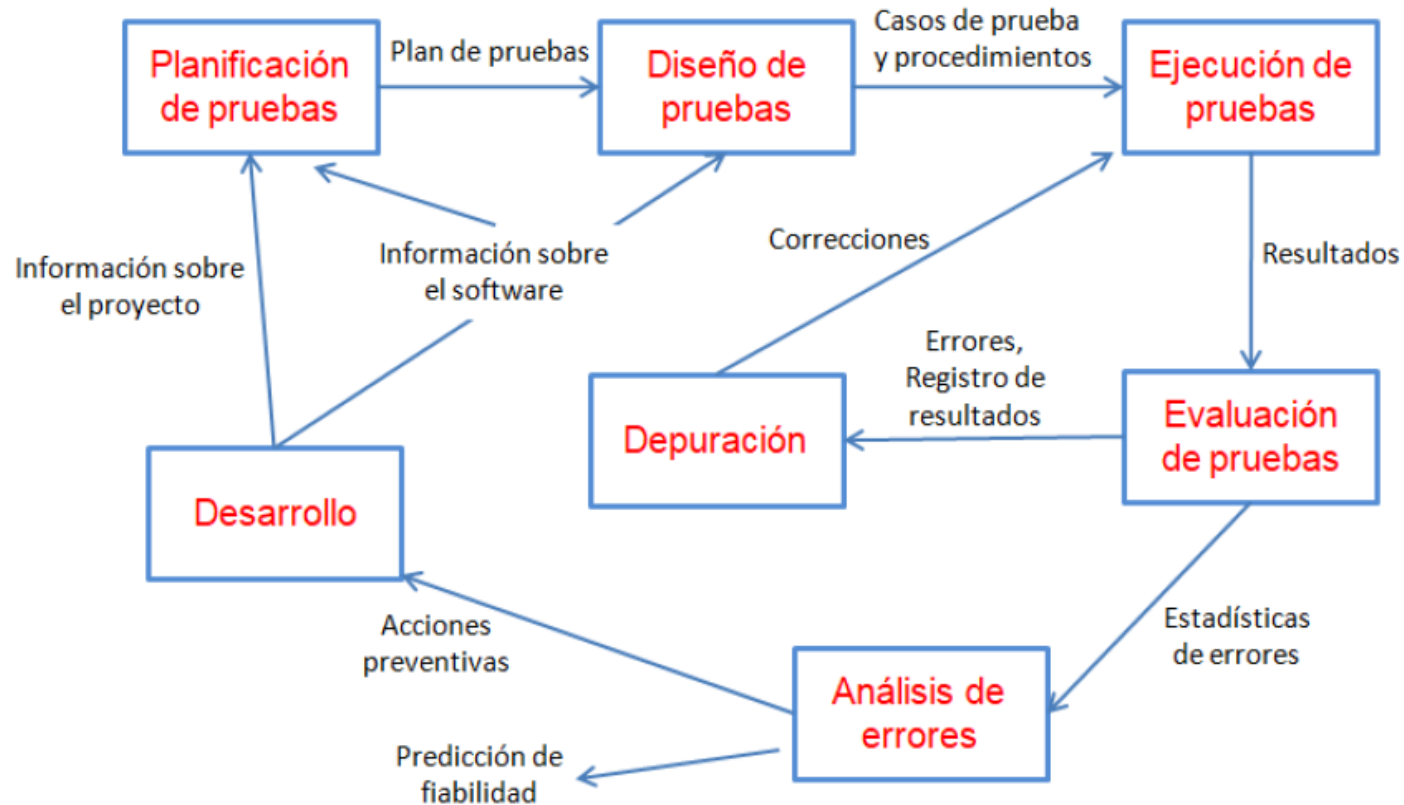
4. FASES DEL DESARROLLO DE UNA APLICACIÓN

Las recomendaciones para las pruebas son las siguientes:

- Cada prueba debe definir los resultados de salida esperados.
- Un programador o una organización debe evitar probar sus propios programas.
- Es necesario revisar los resultados de cada prueba en profundidad.
- Las pruebas deben incluir datos de entrada válidos y esperados, así como no válidos e inesperados.
- Centrar las pruebas en dos objetivos: 1) comprobar si el software no hace lo que debe hacer y 2) comprobar si el software hace lo que no debe hacer.
- Evitar hacer pruebas que no estén documentadas ni diseñadas con cuidado.
- No planear pruebas asumiendo que no se encontrarán errores.
- La probabilidad de encontrar errores en una parte del software es proporcional al número de errores ya encontrados.
- Las pruebas son una tarea creativa.

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

El flujo de proceso para probar el software se muestra a continuación:



4. FASES DEL DESARROLLO DE UNA APLICACIÓN

1. Se **genera un plan de pruebas** partiendo de la documentación del proyecto.
2. A partir del plan se **diseñan las pruebas**. Se identifican las técnicas a utilizar para probar el software.
3. **Generación de los casos de prueba**. Se han de confeccionar los distintos casos de prueba según la técnica o técnicas identificadas previamente.
4. **Definición de los procedimientos de la prueba**. Hay que especificar cómo se va a llevar a cabo el proceso, quién lo va a realizar, cuándo, etc.
5. **Ejecución de las pruebas** aplicando los casos de prueba generados previamente.
6. **Evaluación**. Se identifican los posibles errores producidos al comparar los resultados obtenidos en la ejecución con los esperados. Se elabora un informe con el resultado de ejecución de las pruebas.
7. **Depuración**. Trata de localizar y corregir los errores. Si se corrige un error se debe volver a probar el software para ver que se ha resuelto el problema.
8. El **análisis de errores** puede servir para predecir la fiabilidad del software y mejorar los procesos de desarrollo.

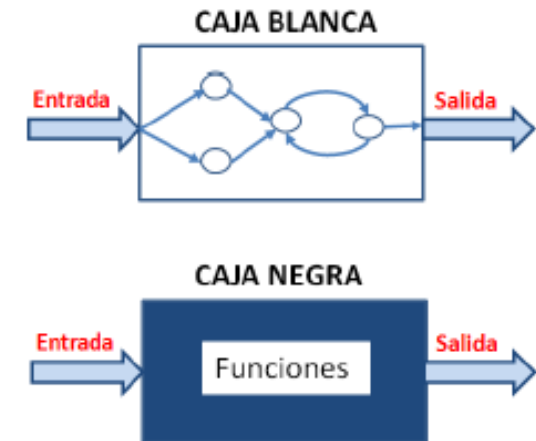
4. FASES DEL DESARROLLO DE UNA APLICACIÓN

Para llevar a cabo el diseño de casos de prueba se utilizan dos técnicas:

- **Prueba de caja blanca:** se centran en validar la estructura interna del programa (necesitan 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 (necesitan saber la funcionalidad que el código ha de proporcionar)

Estas pruebas no son excluyentes y se pueden combinar para descubrir diferentes tipos de errores.

En el Capítulo 3 se tratarán más ampliamente el diseño y la realización de las pruebas.



4. FASES DEL DESARROLLO DE UNA APLICACIÓN

DOCUMENTACIÓN

Todas las etapas del desarrollo deben quedar perfectamente documentadas. En esta etapa será necesario reunir todos los documentos generados y clasificarlos según el nivel técnico de sus descripciones.

Los documentos relacionados con un proyecto de software...

- Deben actuar como un medio de comunicación entre los miembros del equipo de desarrollo.
- Deben ser un repositorio de información del sistema para ser utilizado por el personal de mantenimiento.
- Deben proporcionar información para ayudar a planificar la gestión del presupuesto y programar el proceso del desarrollo del software.
- Algunos de los documentos deben indicar a los usuarios cómo utilizar y administrar el sistema. (manuales de usuario)

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

Por lo general se puede decir que la documentación presentada se divide en dos clases:

- La **documentación del proceso**. Registra el proceso de desarrollo y mantenimiento. Se indican planes, estimaciones y horarios que se utilizan para predecir y controlar el proceso de software, que informan sobre como usar los recursos durante el proceso de desarrollo, sobre normas de cómo se ha de implementar el proceso.
- La **documentación del producto**. Describe el producto que está siendo desarrollado. Define dos tipos de documentación: la que describe el producto desde un punto de vista técnico, orientado al desarrollo y mantenimiento del mismo; y la dirigida al usuario que ofrece una descripción del producto orientada a los usuarios que utilizarán el sistema.

La **documentación del proceso** se utiliza para gestionar todo el proceso de desarrollo del software. La **documentación del producto** se utiliza después de que el sistema está en funcionamiento, aunque también es esencial para la gestión del desarrollo del sistema.

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

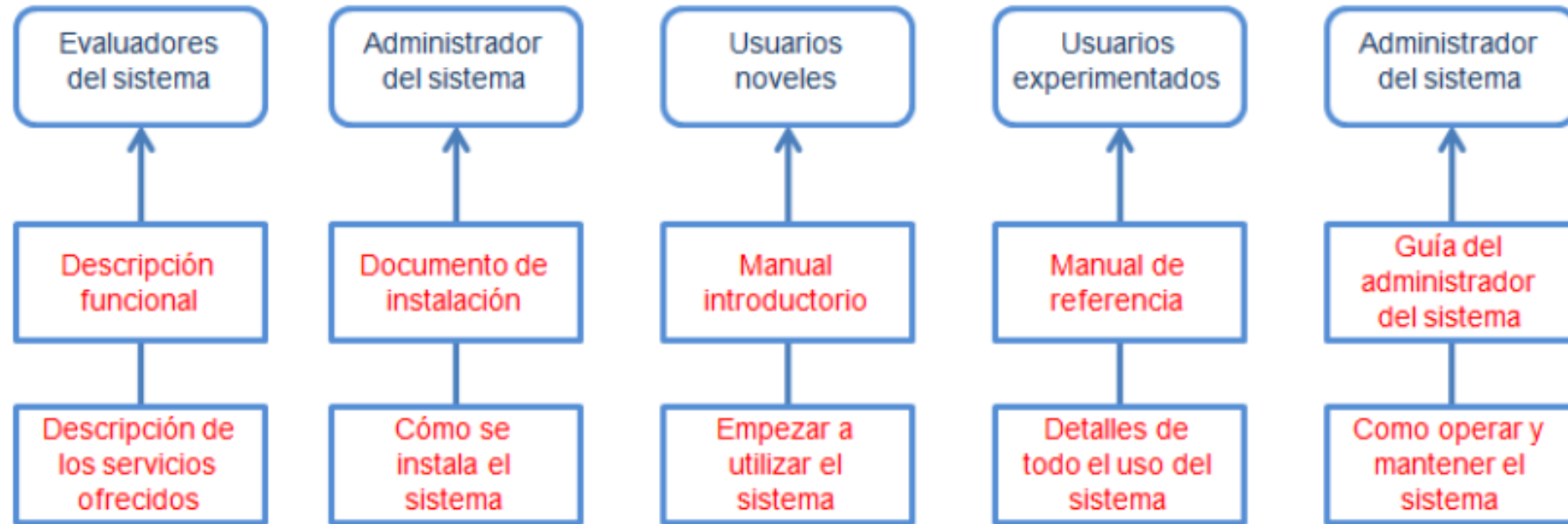
Documentación de usuario

Los usuarios de un sistema no son todos iguales. Hay que distinguir entre los usuarios finales y los usuarios administradores del sistema.

- Los **usuarios finales** buscan utilizar el software para sus tareas sin preocuparse por detalles técnicos.
- Los **administradores del sistema** se encargan de gestionar y mantener el software para los usuarios, actuando como gestores de red o técnicos de soporte.

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

4. FASES DEL DESARROLLO DE UNA APLICACIÓN



4. FASES DEL DESARROLLO DE UNA APLICACIÓN

Documentación del sistema

La 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.

En los grandes sistemas software la documentación debe incluir:

- El análisis y especificación de requisitos.
- Diseño
- Implementación
- Plan de pruebas del sistema
- Plan de pruebas de aceptación
- Los diccionarios de datos

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

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 llevan a cabo las siguientes tareas:

- Se define la **estrategia para la implementación del proceso**. Se desarrolla un plan donde se establecen las normas, los procedimientos para recibir, registrar, solucionar, hacer un seguimiento de los problemas y para probar el producto software en el entorno de trabajo.
- **Pruebas de operación**. Para cada *release* del producto software, se llevarán a cabo pruebas de funcionamiento y tras satisfacerse los criterios especificados, se libera el software para uso operativo.
- **Uso operacional del sistema**. El sistema entrará en acción en el entorno previsto de acuerdo con la documentación de usuario.
- **Soporte al usuario**. Se deberá proporcionar asistencia y consultoría a los usuarios cuando la soliciten. Estas peticiones y las acciones subsecuentes se deberán registrar y supervisar.

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

MANTENIMIENTO

El mantenimiento del software se define como la modificación de un producto de software después de la entrega para corregir los fallos, para mejorar el rendimiento u otros atributos, o para adaptar el producto a un entorno modificado.

Existen cuatro tipos de mantenimiento del software que dependen de las demandas de los usuarios del producto software a mantener:

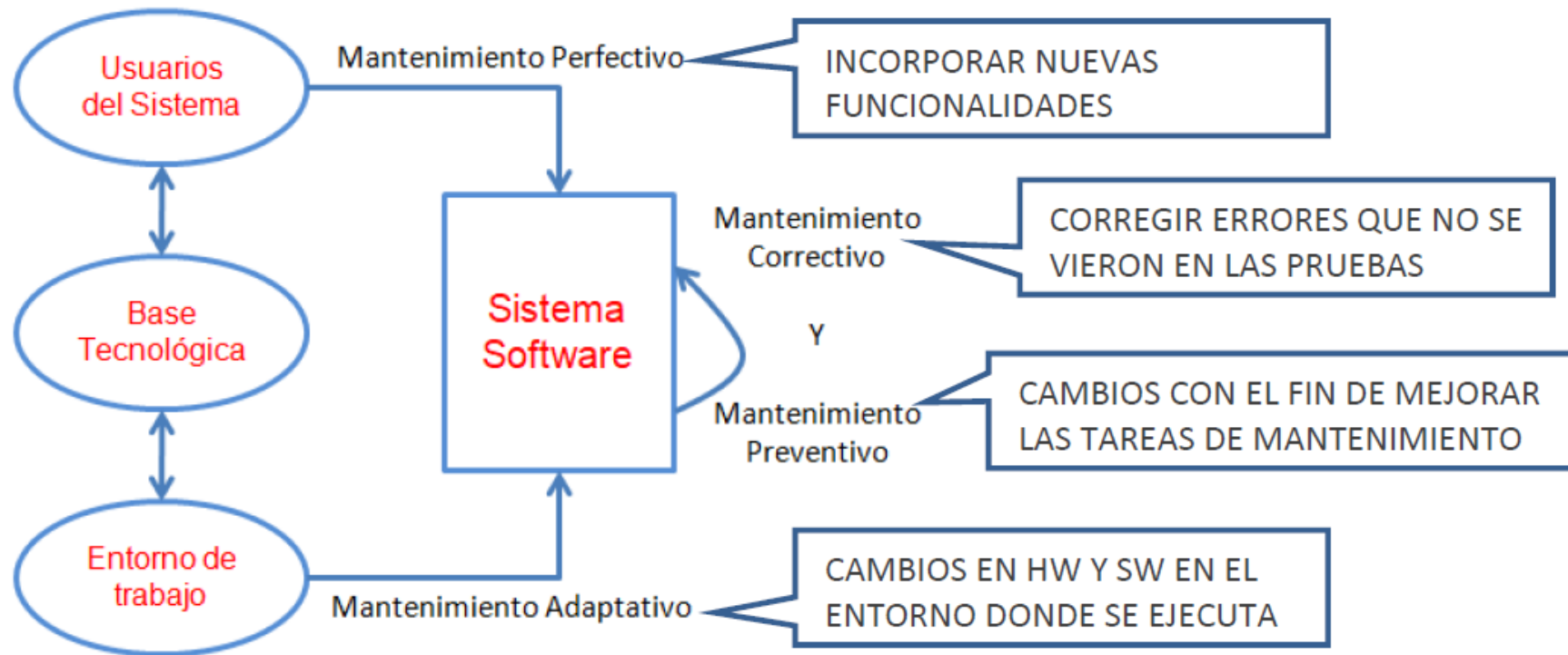
- El **mantenimiento adaptativo** se centra en ajustar un producto software a los cambios en el entorno original, como modificaciones en hardware, sistema operativo y reglas empresariales. Este tipo de mantenimiento es común debido a los rápidos avances tecnológicos que a menudo vuelven obsoletos los productos software desarrollados originalmente. Su objetivo es asegurar que el software funcione de manera óptima en el entorno cambiante.

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

- El **mantenimiento correctivo** se enfoca en corregir errores o defectos que el cliente pueda encontrar después de la entrega del producto, a pesar de las pruebas previas. Su objetivo principal es resolver los fallos identificados en el software.
- El **mantenimiento perfectivo** implica la mejora del software a medida que el cliente lo utiliza, incorporando nuevas funcionalidades y mejoras de rendimiento o mantenibilidad. Se enfoca en añadir características más allá de los requisitos iniciales para brindar beneficios adicionales al cliente.
- El **mantenimiento preventivo** se basa en modificar el software para mejorar y facilitar futuras tareas de mantenimiento, sin alterar sus especificaciones. Implica cambios que permiten una corrección, adaptación y mejora más sencilla. Puede abarcar reestructuración de programas para mayor legibilidad y la adición de comentarios para comprensión. Este tipo de mantenimiento también se conoce como reingeniería del software.

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

A continuación, se presenta de forma esquemática:



4. FASES DEL DESARROLLO DE UNA APLICACIÓN

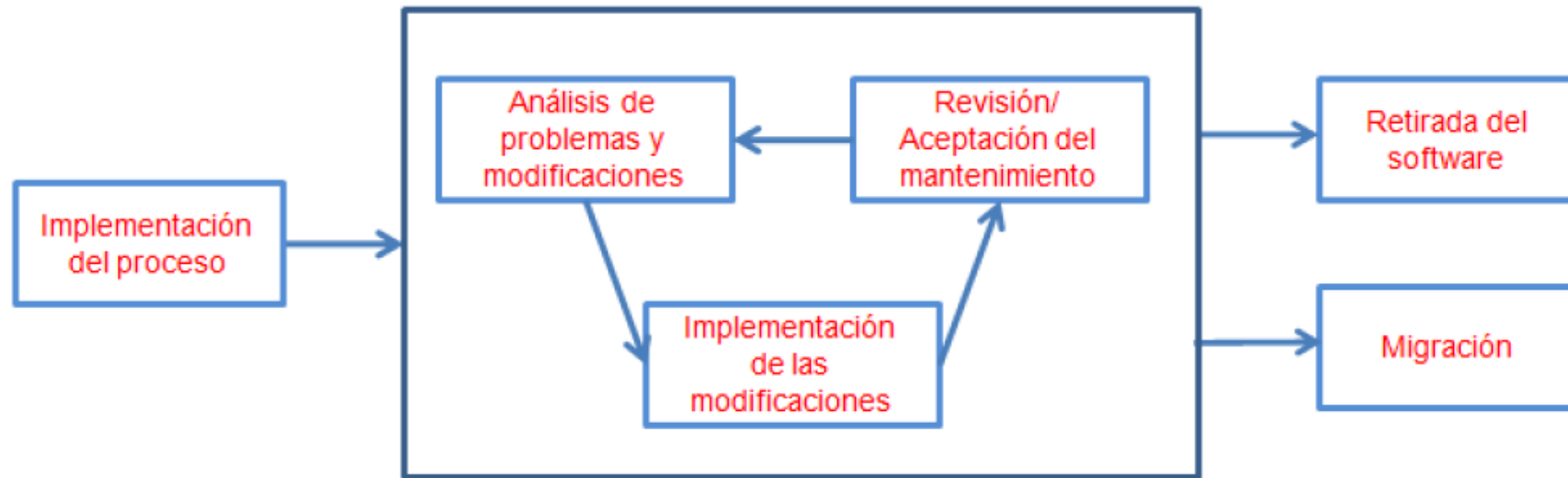
Las tareas que se llevan a cabo en esta etapa son:

- **Implementación del proceso.** El responsable de mantenimiento debe planificar, documentar y ejecutar actividades de mantenimiento, incluyendo la recepción y seguimiento de informes de problemas y solicitudes de usuarios, así como proporcionar actualizaciones a los usuarios.
- **Análisis de problemas y modificaciones.** El responsable de mantenimiento analiza informes de problemas y solicitudes de modificación considerando su impacto en la organización y sistemas interconectados. Luego, desarrolla opciones y documenta el análisis, la solicitud y las posibles soluciones.
- **Implementación de las modificaciones.** El responsable de mantenimiento analiza la documentación y software a modificar, registrando este análisis. Luego, ejecuta el proceso de desarrollo para implementar las modificaciones identificadas.

4. FASES DEL DESARROLLO DE UNA APLICACIÓN

- **Revisión/aceptación del mantenimiento.** El responsable de mantenimiento realiza revisiones con la entidad autorizante para evaluar la integridad del sistema modificado y obtiene aprobación para la finalización exitosa de la modificación.
- **Migración.** Se deberá preparar, documentar y ejecutar un plan de migración. Las actividades de planificación deberán incluir a los usuarios a los que se les dará notificación de los planes y actividades de migración.
- **Retirada del software.** El software será retirado a solicitud del propietario. Se creará y documentará un plan de retirada para cesar el soporte activo por parte de las entidades de operación y mantenimiento. Se informará a los usuarios sobre los planes de retirada. Para una transición suave al nuevo sistema, se recomienda operar en paralelo con el antiguo y el nuevo software, brindando formación a los usuarios durante este periodo.

4. FASES DEL DESARROLLO DE UNA APLICACIÓN



5. CONCEPTO DE PROGRAMA

Un **programa informático** consiste en un conjunto de instrucciones escritas en un lenguaje de programación que resuelven problemas mediante el procesamiento de datos. La traducción al lenguaje máquina es necesaria para la ejecución, realizada por un **compilador**. Luego, el programa se carga en la memoria principal para que el procesador ejecute las instrucciones una por una.

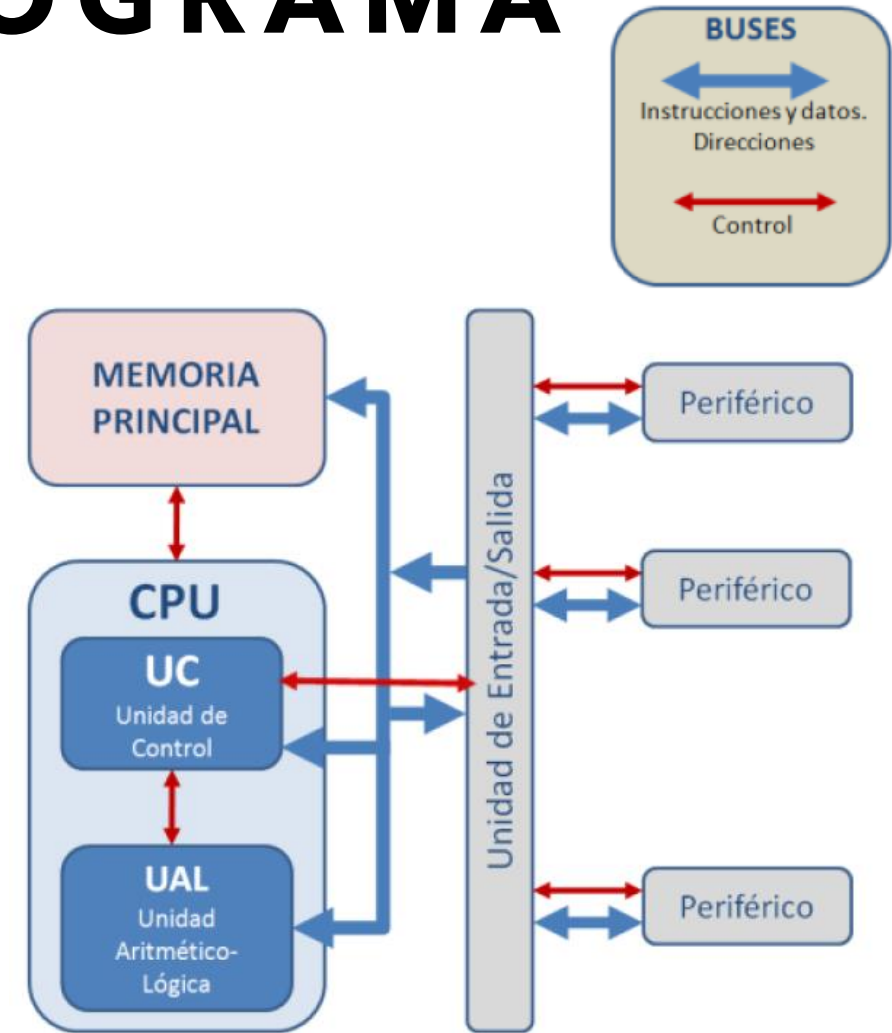
PROGRAMA Y COMPONENTES DEL SISTEMA INFORMÁTICO

- Para ejecutar un programa se necesitan los recursos hardware del ordenador: el **procesador** (también conocido como UCP Unidad Central de Proceso o CPU), la **memoria RAM** (o memoria principal), los **dispositivos de E/S**, etc.
- Las **instrucciones de un programa** se cargan en la memoria principal y la CPU será la encargada de ejecutarlas.

5. CONCEPTO DE PROGRAMA

En una arquitectura Von Neumann, la CPU está formada por varios componentes:

- La **Unidad de Control (UC)** que interpreta y ejecuta las instrucciones máquina almacenadas en la memoria principal y genera las señales de control necesarias para ejecutarlas.
- La **Unidad Aritmético-Lógica (UAL)**, recibe los datos sobre los que efectúa operaciones de cálculo y comparaciones, y toma decisiones lógicas (determina si una afirmación es cierta o falsa mediante las reglas del álgebra de Boole), y devuelve el resultado, todo ello bajo la supervisión de la Unidad de Control.
- **Los registros** de trabajo o de propósito general donde se almacena información temporal, es el almacenamiento interno de la CPU.



5. CONCEPTO DE PROGRAMA

La UC consta de una serie de registros que intervienen en la ejecución de las instrucciones, un decodificador y un reloj:

- **Contador de Programa o de instrucciones** (CP, Program Counter). Contiene la dirección de la siguiente instrucción a ejecutar, su valor es actualizado por la CPU después de capturar una instrucción.
- **Registro de Instrucción** (RI, Instruction Register). Contiene el código de la instrucción a ejecutar, aquí se analiza el código de operación. Está dividido en dos zonas: el código de operación y la dirección de memoria donde se encuentra el operando.
- **Registro de dirección de memoria** (RDM, Memory Address Register). Contiene la dirección de una posición de memoria, donde se encuentra o va a ser almacenada la información, este intercambio se realiza a través del bus de direcciones. Apunta a una instrucción o un dato.
- **Registro de intercambio de memoria** (RIM Memory Buffer Register). Recibe o envía, dependiendo de si es una operación de lectura o escritura, la información o el dato contenido en la posición apuntada por el RDM, el intercambio de datos con la memoria se realiza a través al bus de datos.

5. CONCEPTO DE PROGRAMA

- **Decodificador de instrucción (DI).** Se encarga de extraer y analizar el código de operación de la instrucción en curso contenida en el RI y genera las señales de control necesarias para ejecutar correctamente la instrucción.
- **El Reloj.** Proporciona una sucesión de impulsos eléctricos a intervalos constantes, va marcando 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, que sincronizadas con los impulsos de reloj hace que se ejecute paso a paso y de manera ordenada la instrucción cargada en él.

A la hora de ejecutar una instrucción se distinguen dos fases:

- **Fase de búsqueda.** Consiste en localizar la instrucción a ejecutar dentro de la memoria principal y llevarla a la Unidad de Control o UC para procesarla.
- **Fase de ejecución.** Es la realización de las acciones que llevan asociadas las instrucciones. Por ejemplo, una suma, una resta, una carga de datos.

5. CONCEPTO DE PROGRAMA

Vamos a ver con un ejemplo como se llevarían a cabo las fases de búsqueda y ejecución para un programa almacenado en memoria que lee dos números, los suma y visualiza la suma:

MEMORIA			
PROGRAMA		DATOS	
1	9	17	25
2	10	18	26
3	11	19	²⁷ Dato A
⁴ Leer A	12	20	²⁸ Dato B
⁵ Leer B	13	21	²⁹ Dato C
⁶ Calcular C=A+B	14	22	30
⁷ Visualizar C	15	23	31
8	16	24	32

Podemos imaginarnos la memoria como una serie de casillas con su dirección asociada.

5. CONCEPTO DE PROGRAMA

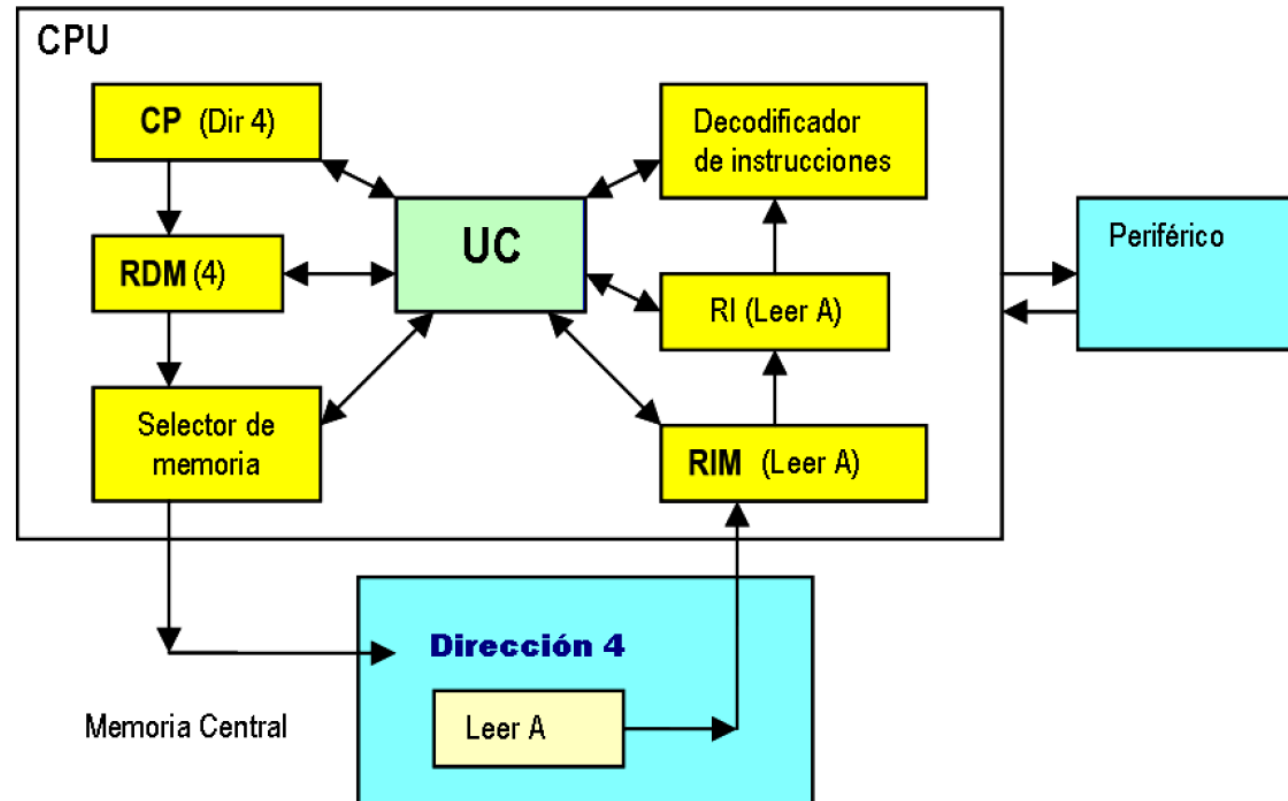
Cada instrucción tiene su fase de búsqueda y su fase de ejecución. A continuación veremos la fase de búsqueda de la instrucción Leer A que se producirá al momento de su ejecución.

Fase de búsqueda:

1. En el CP se almacena la dirección de memoria de comienzo del programa, la 4.2. La UC envía una orden para que el contenido del CP se transfiera al RDM (registro de dirección de memoria).
2. El selector de memoria localiza la posición 4, y transfiere su contenido al RIM (registro de intercambio). El RIM contiene: Leer A.
3. La UC da la orden de transferir el contenido del RIM al registro de instrucción RI en el que deposita el código de la instrucción a ejecutar.
4. Seguidamente el decodificador de instrucción (DI) analiza el código contenido en el RI (en el ejemplo la operación es Leer A) y genera las señales de control para ejecutar correctamente la instrucción.
5. El CP se incrementa en uno y apuntará a la siguiente instrucción, en este caso es la 5 Leer B.

5. CONCEPTO DE PROGRAMA

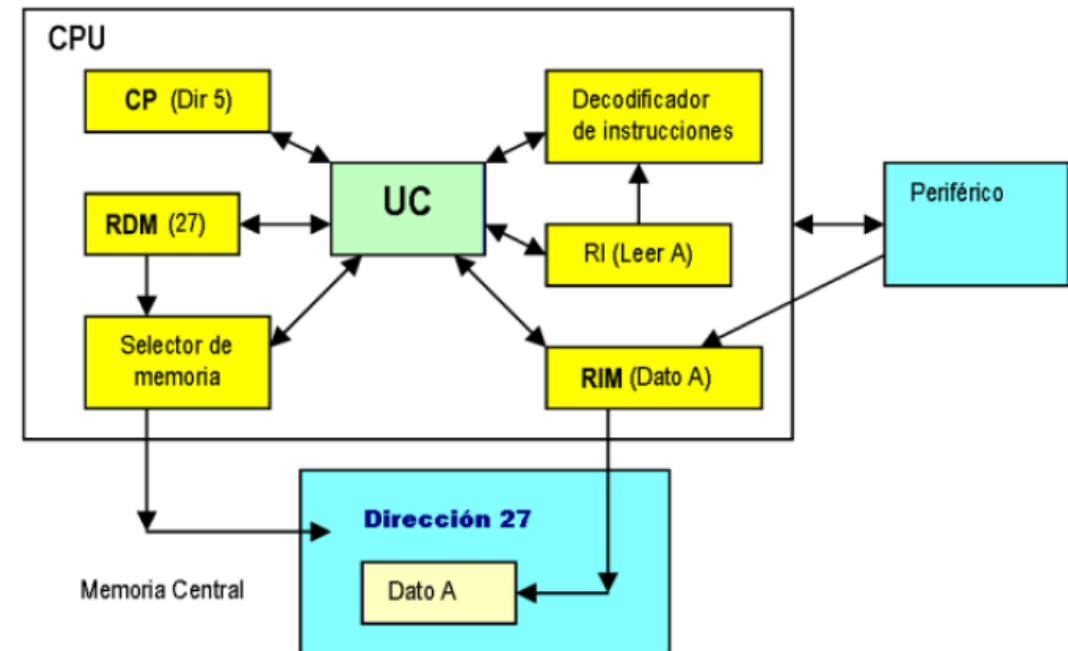
Fase de búsqueda de la instrucción "Leer A":



5. CONCEPTO DE PROGRAMA

Fase de ejecución:

7. Una vez conocido el código de operación, la U.C. establece las conexiones con el dispositivo de entrada para aceptar el dato A.
8. La UC da la orden de que el dato leído se cargue en el RIM y que en el RDM se cargue la dirección de memoria donde se va a almacenar el dato leído, en el ejemplo esta dirección es la 27.
9. El selector de memoria es el encargado de guardar en la dirección indicada por RDM, la 27, el contenido del RIM.



Nota: mas ejemplos en el libro, pág. 46

6. LENGUAJES DE PROGRAMACIÓN

Podemos definir un lenguaje de programación como un conjunto de caracteres, las reglas para la combinación de esos caracteres y las reglas que definen sus efectos cuando son ejecutadas por un ordenador. Consta de los siguientes elementos:

- Un **alfabeto o vocabulario** (léxico): formado por el conjunto de símbolos permitidos.
- Una **sintaxis**: son las reglas que indican cómo realizar las construcciones con los símbolos del lenguaje.
- Una **semántica**: son las reglas que determinan el significado de cualquier construcción del lenguaje.

CLASIFICACIÓN Y CARACTERÍSTICAS

Según su nivel de abstracción:

- **Lenguajes de bajo nivel**: Los lenguajes de programación de bajo nivel se asemejan al funcionamiento interno de las computadoras. El más básico es el lenguaje máquina, compuesto de ceros y unos entendibles por la máquina y específico para cada procesador. Le sigue el lenguaje ensamblador, también específico y complejo, que requiere traducción a lenguaje máquina para ejecutarse. Se emplean nombres nemotécnicos y las instrucciones operan directamente en registros de memoria física.

6. LENGUAJES DE PROGRAMACIÓN

```
-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
OCFD:0110 20 65 73 74 65 20 65 73-20 75 6E 20 70 72 6F 67
OCFD:0120 72 61 6D 61 20 68 65 63-68 6F 20 65 6E 20 61 73
OCFD:0130 73 65 6D 62 6C 65 72 20-70 61 72 61 20 6C 61 20
OCFD:0140 57 69 6B 69 70 65 64 69-61 24

                                Hola,
                                este es un prog
                                rama hecho en as
                                sembler para la
                                Wikipedia$
```

- **Lenguajes de nivel medio:** Este tipo de lenguajes tienen ciertas características que los acercan a los lenguajes de bajo nivel, pero a la vez también tienen características de los de alto nivel. Un lenguaje de programación de este tipo es el lenguaje C, muy utilizado en la creación de sistemas operativos.
- **Lenguajes de alto nivel:** Los lenguajes de programación de alto nivel son más accesibles debido a su uso de palabras del lenguaje natural y no requieren conocimiento del código máquina. Para ejecutarlos se requiere un intérprete o compilador que los traduzca al lenguaje máquina. Estos lenguajes son independientes del hardware y se centran en resolver problemas específicos: C++, C#, Java, Python...

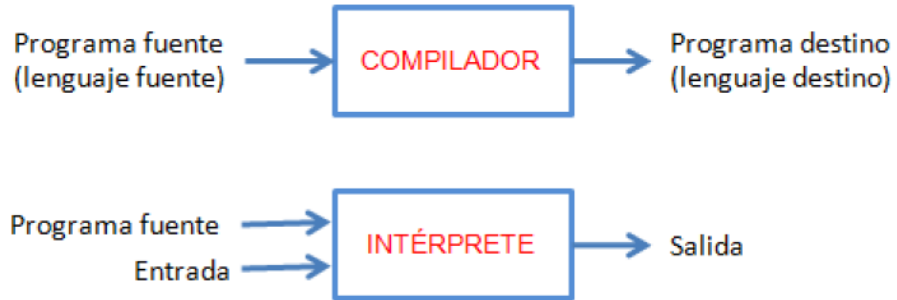
6. LENGUAJES DE PROGRAMACIÓN

Según la forma de ejecución:

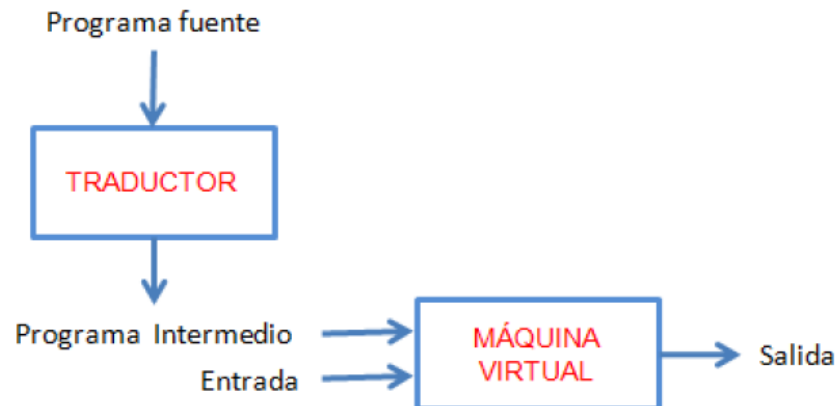
- **Lenguajes compilados:** Un programa que se escribe en un lenguaje de alto nivel tiene que traducirse a un código que pueda utilizar la máquina. Los programas traductores que pueden realizar esta operación se llaman compiladores o intérpretes. Un **compilador** es un programa que puede leer un programa escrito en un determinado lenguaje y traducirlo en un programa equivalente en otro lenguaje. Hay que tener en cuenta que el compilador devolverá errores si el programa en el lenguaje fuente no está bien escrito. Ejemplos: C#, C++...
- **Lenguajes interpretados:** En este caso, en vez de producir un programa destino como resultado del proceso de traducción, el intérprete nos da la apariencia de ejecutar directamente las operaciones especificadas en el programa fuente con las entradas proporcionadas por el usuario. Cada vez que se ejecuta una instrucción se debe interpretar y traducir a lenguaje máquina. Ejemplos: PHP, JavaScript...

El programa destino en el lenguaje máquina que produce un compilador es por lo general más rápido que un intérprete al momento de asignar las entradas a las salidas.

6. LENGUAJES DE PROGRAMACIÓN



Los procesadores del lenguaje Java combinan la compilación y la interpretación. Un programa fuente en Java (por ejemplo *miprograma.java*) puede compilarse primero en un formato intermedio, llamado *bytecodes* (*miprograma.class*), después una máquina virtual los interpreta (*java miprograma.java*).



6. LENGUAJES DE PROGRAMACIÓN

Según el paradigma de programación:

- **Lenguajes imperativos:** Los lenguajes imperativos surgieron inicialmente como lenguajes máquina en computadoras, con instrucciones simples, evolucionando luego hacia lenguajes ensambladores. Estos lenguajes implican cálculos mediante sentencias que definen cómo manipular la información en memoria y comunicarse con dispositivos.

La asignación es la sentencia principal, mientras que las estructuras de control determinan el orden y flujo del programa según los resultados de acciones básicas.

Ejemplos: C, Ada, C++, Java, C#.

```
int main() {  
    int x;  
    int fact = 1;  
    scanf("%i", &x);  
    for(int i = 2; i <= x; i++) {  
        fact = fact * i;  
    }  
    printf ("Factorial = %i", fact);  
}
```


6. LENGUAJES DE PROGRAMACIÓN

Según el paradigma de programación:

- **Lenguajes funcionales:** El paradigma funcional está basado en el concepto matemático de función. Los programas escritos en lenguajes funcionales estarán constituidos por un conjunto de definiciones de funciones matemáticas junto con los argumentos sobre los que se aplican.

En los lenguajes funcionales:

- 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, Miranda...

```
(defun factorial (N)
  "Calcula el factorial de N."
  (if (= N 1)
      1
      (* N (factorial (- N 1)))))
```

6. LENGUAJES DE PROGRAMACIÓN

- **Lenguajes lógicos:** En este tipo de lenguajes un cálculo es el proceso de encontrar qué elementos de un dominio cumplen determinada relación definida sobre dicho dominio o el proceso de determinar si un determinado elemento cumple o no dicha relación.

Los programas escritos en estos lenguajes se pueden ver como una base de datos formada por listas de declaraciones lógicas (reglas) que pueden ser consultadas. La ejecución consistirá en realizar preguntas de forma interactiva.

Ejemplo: Prolog

```
padrede(pedro,maria). % pedro es padre de maria
padrede(juan,luis). % juan es padre de luis
padrede(juan,marta).
padrede(manuel,ana).
```

```
?- padrede(maria,juan).
no.
?- padrede(manuel,ana).
yes.
?- padrede(pedro,marta).
no.
```

6. LENGUAJES DE PROGRAMACIÓN

- **Lenguajes de programación estructurados:** tres construcciones lógicas son el fundamento de la programación estructurada: la estructura secuencial, la condicional y la repetitiva. Los lenguajes de programación basados en la programación estructurada son los llamados lenguajes de programación estructurados.

Un programa estructurado resulta fácil de leer, es decir, puede ser leído secuencialmente desde el comienzo hasta el final, pero como todo su código se concentra en un único bloque, si el programa es demasiado grande o el problema a resolver es complejo resulta difícil su lectura y manejo.

Actualmente cuando se habla de programación estructurada, nos solemos referir a la división de un programa en partes más manejables conocidas como **módulos**. A esta evolución de la programación estructurada se le llama **programación modular**.

Ejemplos: Pascal, C, Fortran, Modula-2...

6. LENGUAJES DE PROGRAMACIÓN

- **Lenguajes de programación orientados a objetos:** en la programación orientada a objetos un programa está compuesto por un conjunto de objetos no por un conjunto de instrucciones o un conjunto de módulos, como en la programación estructurada.

Un **objeto** consta de una estructura de datos y de una colección de métodos u operaciones que manipulan esos datos. Los datos definidos dentro de un objeto son sus atributos.

```
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;}
```

Ejemplos: C++, Java, Ada...

7. OBTENCIÓN DE CÓDIGO EJECUTABLE

La generación del código fuente se lleva a cabo en la etapa de codificación. En esta etapa el código pasa por diferentes estados.

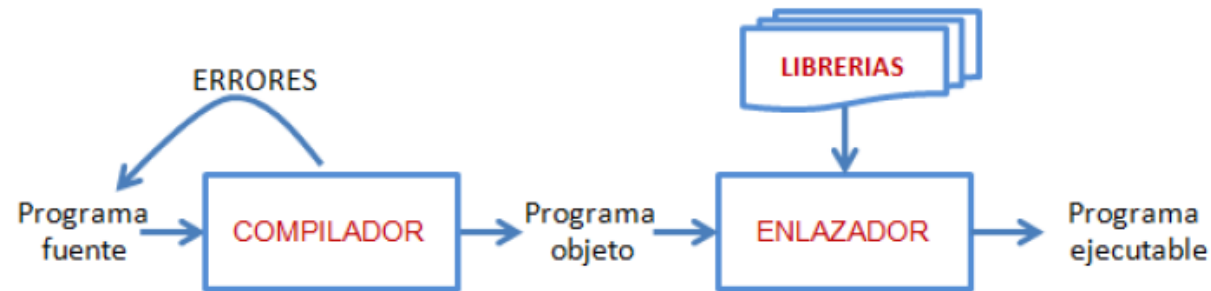
TIPOS DE CÓDIGO

- **Código fuente:** Es el código escrito por los programadores utilizando algún editor de texto o alguna herramienta de programación. Se utiliza un lenguaje de programación de alto nivel apropiado para el problema que se trata de resolver. Para escribir el código se parte de los diagramas de flujo o pseudocódigos diseñados en la etapa de diseño. Este código no es directamente ejecutable por el ordenador.
- **Código objeto:** Es el código resultante de compilar el código fuente. No es directamente ejecutable por el ordenador ni entendido por el ser humano. Es un código o representación intermedia de bajo nivel.
- **Código ejecutable:** Es el resultado de enlazar el código objeto con una serie de rutinas y librerías, obteniendo así el código que es directamente ejecutable por la máquina.

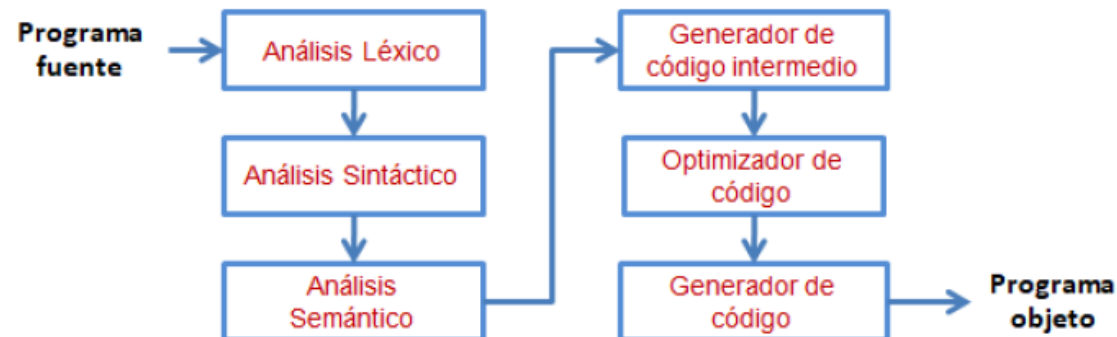
7. OBTENCIÓN DE CÓDIGO EJECUTABLE

COMPILACIÓN

El proceso de compilación de un programa se lleva a cabo mediante dos programas, el compilador y el enlazador.



El compilador se compone internamente de varias etapas o fases que realizan distintas operaciones:



7. OBTENCIÓN DE CÓDIGO EJECUTABLE

- **Análisis léxico:** se lee secuencialmente todo el código fuente obteniendo unidades significativas de caracteres denominadas tokens. Por ejemplo, la instrucción: `suma = x + 2`, generaría 5 tokens: `suma`, `=`, `x`, `+`, `2`.
- **Análisis sintáctico:** recibe el código fuente en forma de tokens y realiza el análisis sintáctico que determina la estructura del programa; es decir, se comprueba si las construcciones de tokens cumplen las reglas de sintaxis definidas en el lenguaje de programación correspondiente.
- **Análisis semántico:** se comprueban que las declaraciones son correctas, se verifican los tipos de todas las expresiones, si las operaciones se pueden realizar sobre esos tipos, si los arrays tienen el tamaño y tipo adecuados, etc.
- **Generación de código intermedio:** después del análisis se genera una representación intermedia similar al código máquina con el fin de facilitar la tarea de traducir al código objeto.

7. OBTENCIÓN DE CÓDIGO EJECUTABLE

- **Optimización de código:** trata de mejorar el código intermedio generado en la fase anterior de tal forma que el código resultante sea más fácil y rápido de interpretar por la máquina.
- **Generación de código:** genera el código objeto de nuestro programa.
- **Optimización de código:** trata de mejorar el código intermedio generado en la fase anterior de tal forma que el código resultante sea más fácil y rápido de interpretar por la máquina.
- **Generación de código:** genera el código objeto de nuestro programa.

El programa enlazador inserta en el código objeto las funciones de librería necesarias para producir el programa ejecutable. Por ejemplo, en un programa escrito en C si el fichero fuente hace referencia a funciones de una biblioteca o a funciones que están definidas en otros ficheros fuentes, entonces el ligador combina estas funciones con el programa principal para crear un fichero ejecutable.

8. MÁQUINAS VIRTUALES

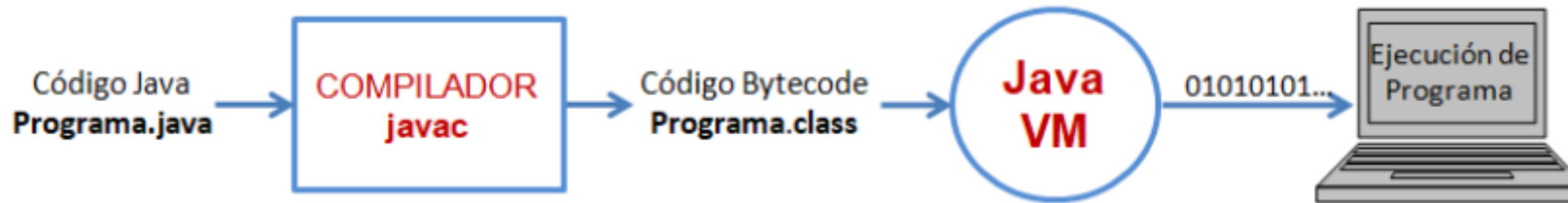
Una máquina virtual es una aplicación software de una máquina (por ejemplo, un ordenador) que ejecuta los programas como si fuese una máquina real. Se pueden clasificar en dos categorías:

- **Máquinas virtuales de sistema:** Permiten ejecutar en la misma máquina física varias máquinas virtuales cada una con un sistema operativo, de esta manera pueden coexistir diferentes sistemas operativos sobre una misma máquina. Un ejemplo de este software es VMware Workstation y Virtual Box; que se pueden utilizar para evaluar y probar nuevos sistemas operativos, para ejecutar aplicaciones sobre distintos sistemas operativos, etc.
- **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. 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. El ejemplo más conocido actualmente de este tipo de máquina virtual es la máquina virtual de Java8.

8. MÁQUINAS VIRTUALES

LA MAQUINA VIRTUAL DE JAVA

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



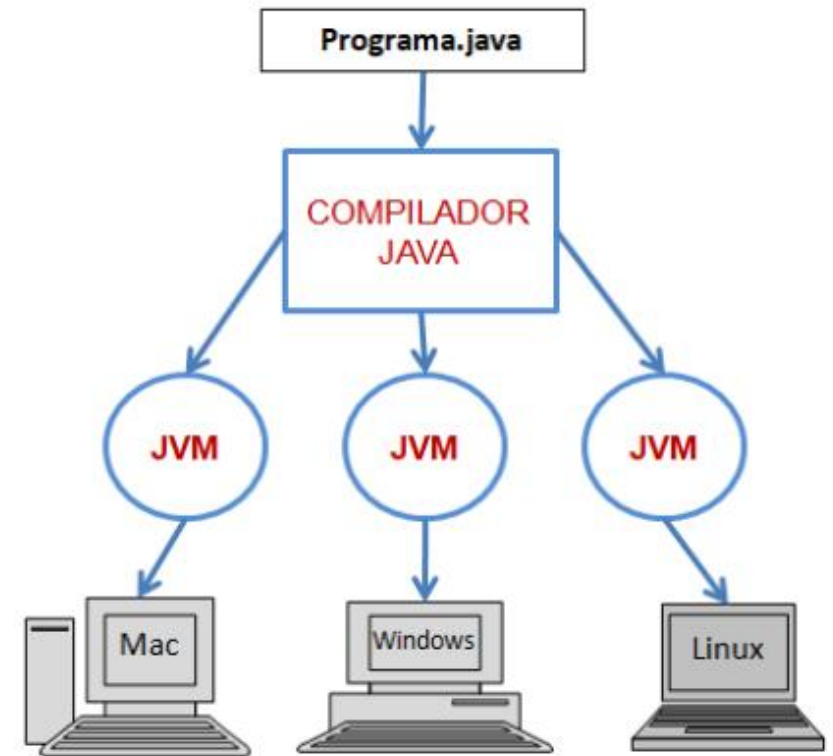
El código fuente del programa está escrito en ficheros de texto plano que tienen la extensión .java. La compilación del fichero, mediante el compilador Java javac, genera un fichero (o varios) con la extensión .class (siempre y cuando no haya errores).

8. 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.

La principal desventaja de los lenguajes basados en máquina virtual es que son más lentos que los lenguajes completamente compilados.



8. 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.

La principal desventaja de los lenguajes basados en máquina virtual es que son más lentos que los lenguajes completamente compilados.

