ENTORNOS DE DESARROLLO

UD1 - DESARRROLLO DE SOFTWARE

1. Software y programa. Tipos de software.

2. Relación hardware-software.

3. Desarrollo de software
    - Ciclos de vida del software
    - Herramientas de apoyo para el desarrollo

4. Lenguajes de programación
    - Concepto y características
    - Lenguajes de programación estructurados
    - Lenguaje de programación orientado a objetos

5. Fases en el desarrollo y ejecución del software
    - Análisis
    - Diseño
    - Codificación. Tipos de código
    - Fases en la obtención de código
    - Maquinas virtuales
    - Pruebas
    - Documentación
    - Explotación
    - Mantenimiento

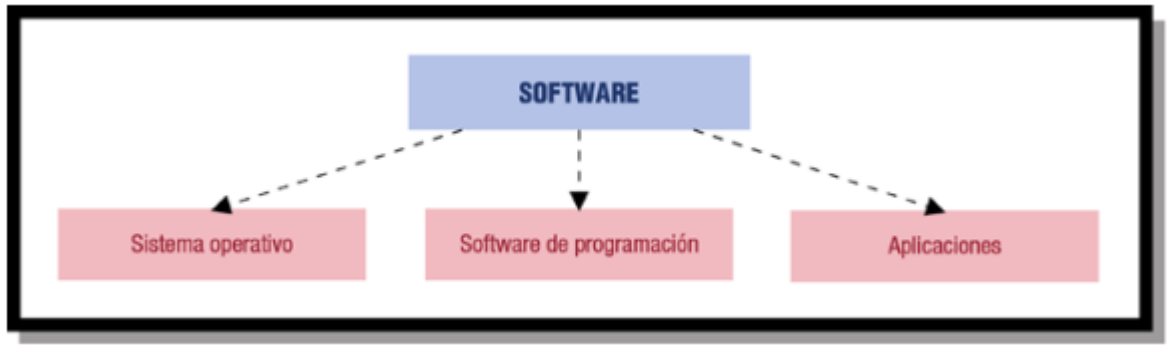
1. Software y programa. Tipos de software.

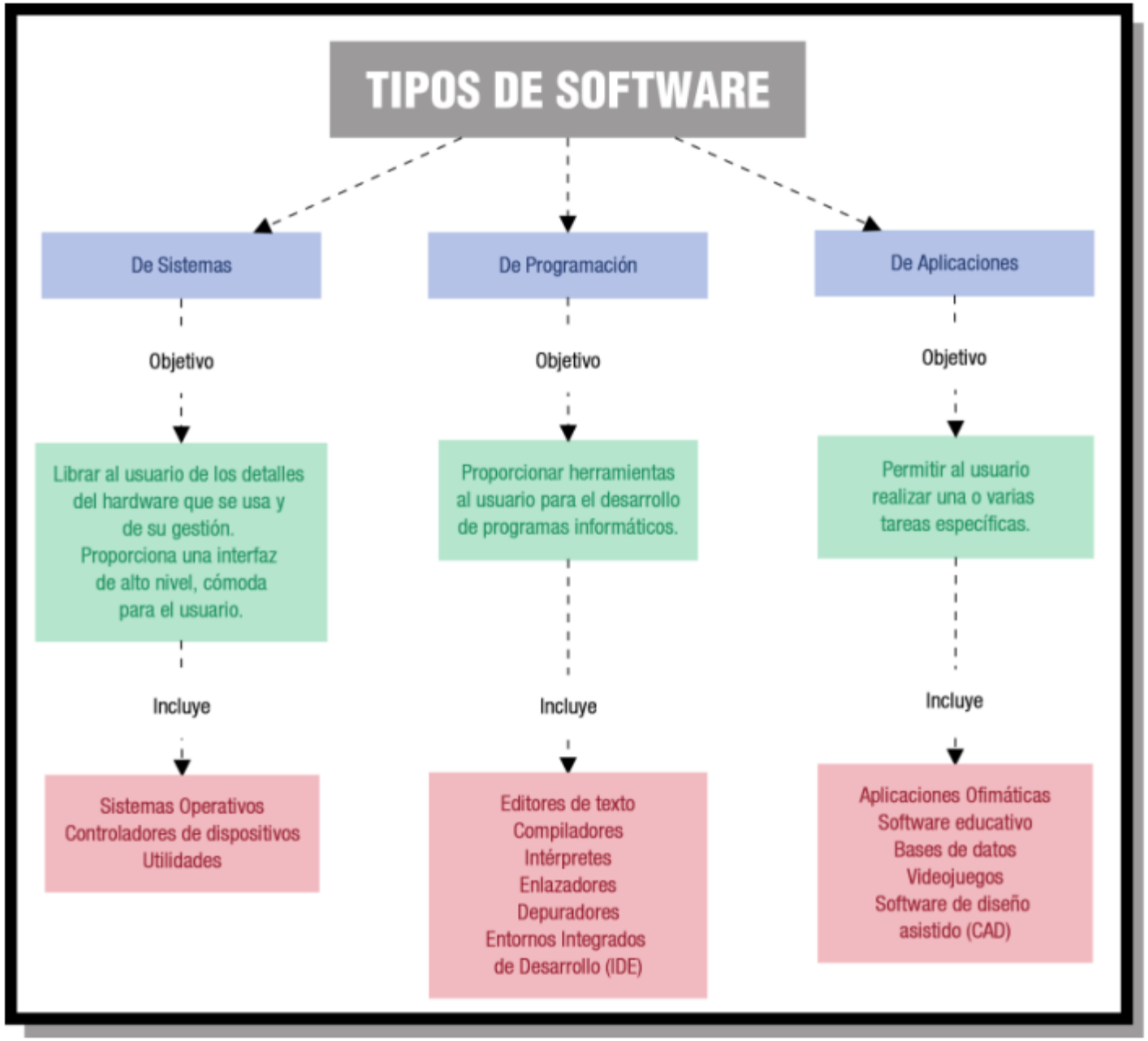
EL PROGRAMA INFORMÁTICO

Un **programa informático** es un conjunto de instrucciones que se ejecutan de manera secuencial con el objetivo de realizar una o varias tareas en un sistema.

Un programa informático interactúa con el sistema ejecutando las diferentes instrucciones del programa en la ALU, mediante instrucciones que sea capaz de entender.

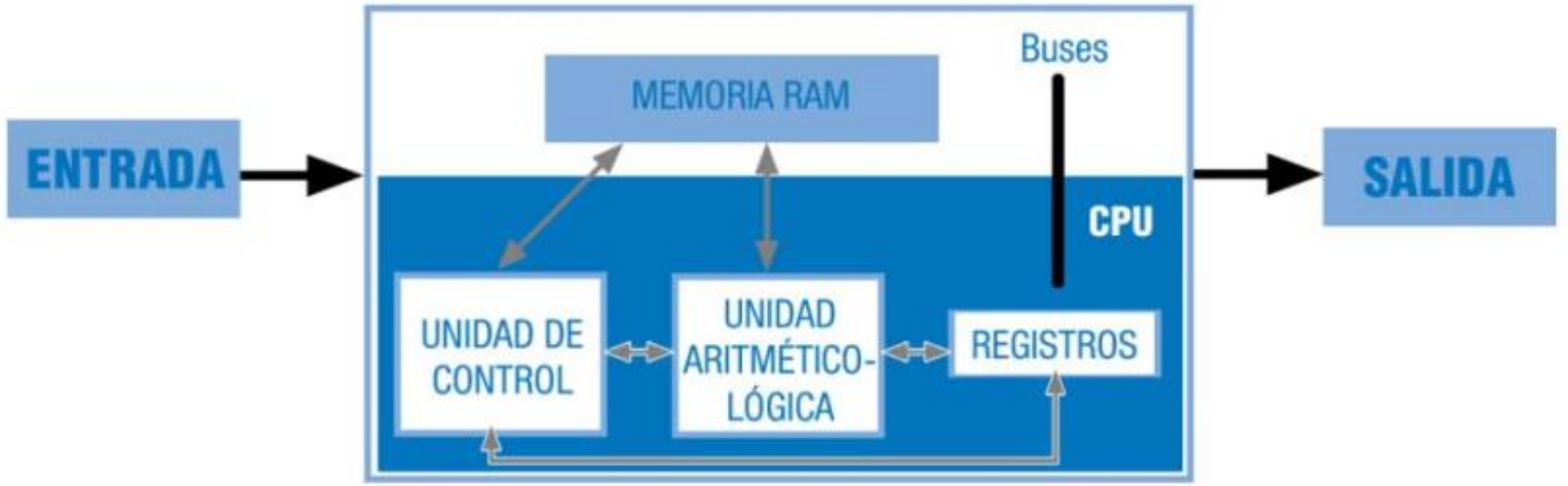
Cada instrucción se divide en instrucciones más pequeñas que se ejecutarán individual y secuencialmente en cada ciclo del procesador. A esas instrucciones más pequeñas se las conoce por el nombre de microinstrucciones.




TIPOS DE SOFTWARE
De Sistemas
De Programación
De Aplicaciones
Objetivo
Objetivo
Objetivo
Librar al usuario de los detalles del hardware que se usa y de su gestión. Proporciona una interfaz de alto nivel, cómoda para el usuario.
Proporcionar herramientas al usuario para el desarrollo de programas informáticos.
Permitir al usuario realizar una o varias tareas específicas.
Incluye
Incluye
Incluye
Sistemas Operativos
Controladores de dispositivos
Utilidades
Editores de texto
Compiladores
Intérpretes
Enlazadores
Depuradores
Entornos Integrados de Desarrollo (IDE)
Aplicaciones Ofimáticas
Software educativo
Bases de datos
Videojuegos
Software de diseño asistido (CAD)

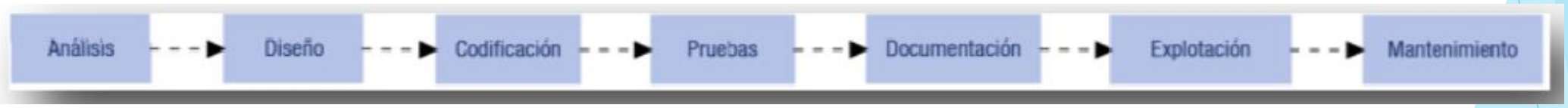
2. Relación hardware-software.

Arquitectura de Von Neumann (von Neumann architecture)



3. Desarrollo de software

Entendemos por Desarrollo de Software todo el proceso que ocurre desde que se concibe una idea hasta que un programa esta implementado en el Ordenador y funcionando.



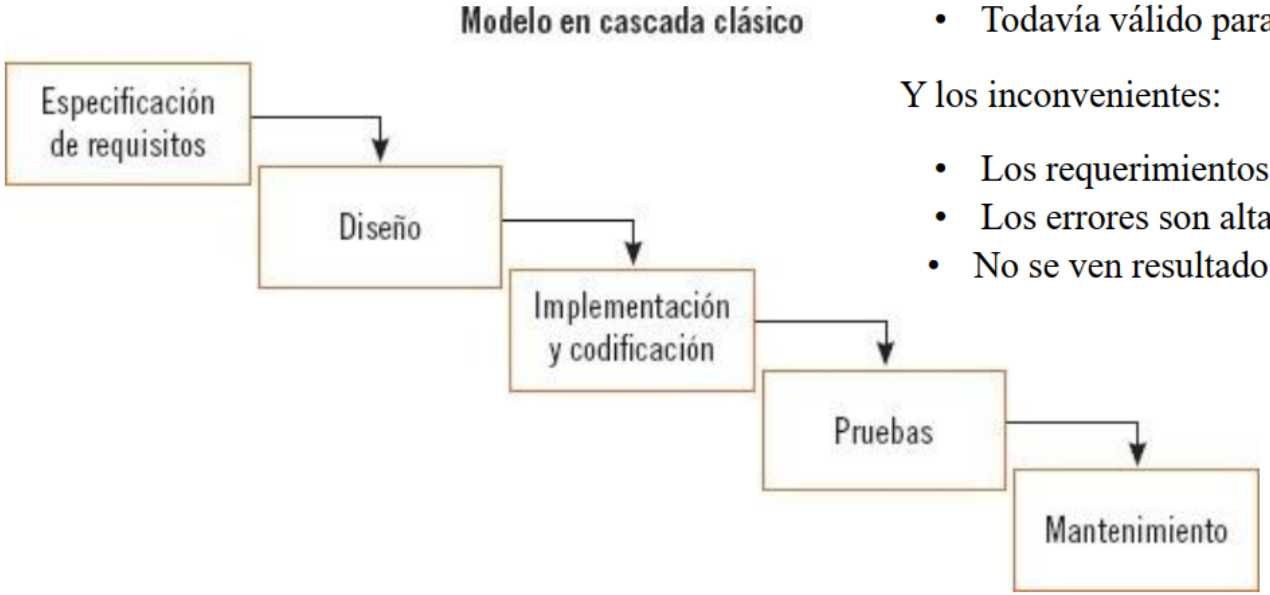
En cascada (waterfall)

Las ventajas son las siguientes:

- Planificación sencilla.
- Gran calidad del producto final.
- Todavía válido para pequeños/medios desarrollos.

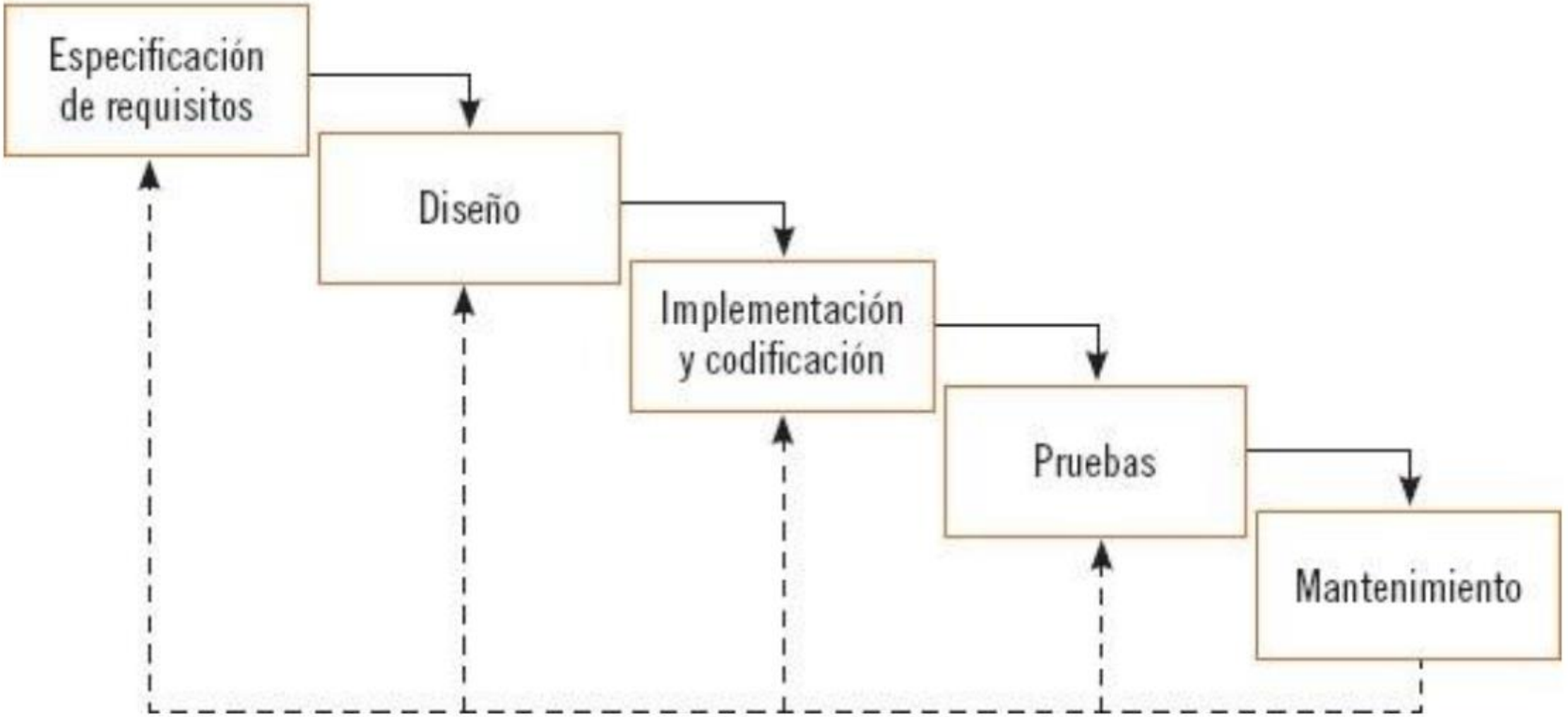
Y los inconvenientes:

- Los requerimientos son necesarios al comienzo del proyecto.
- Los errores son altamente penalizados.
- No se ven resultados hasta una fase avanzada del ciclo

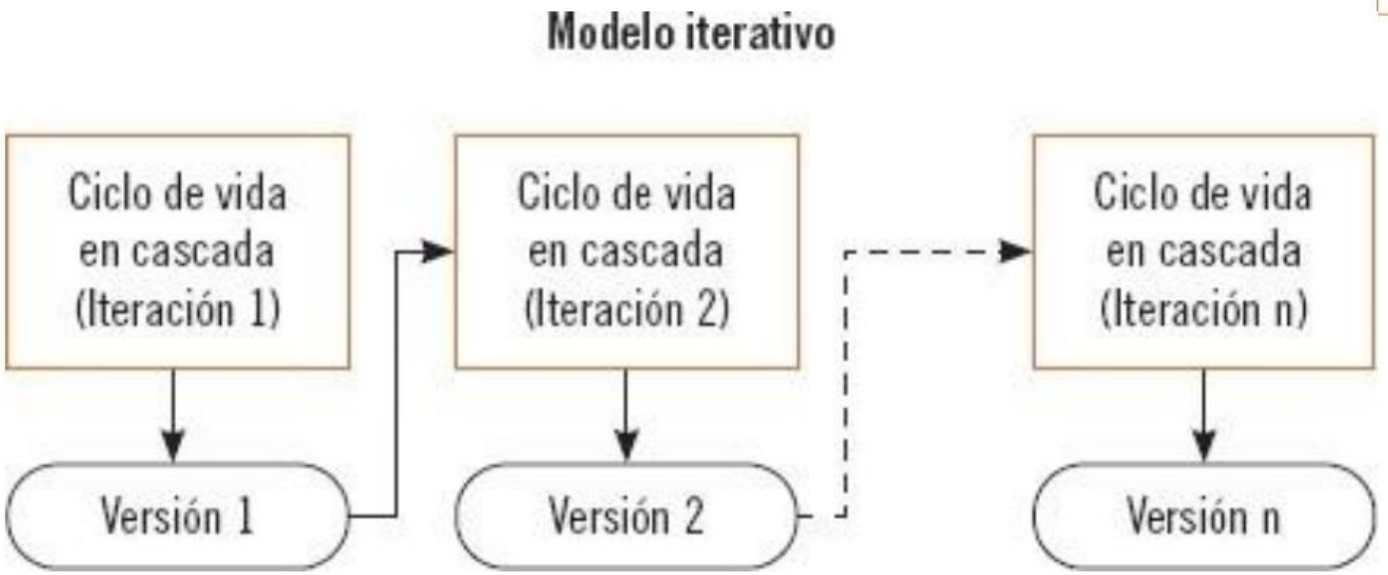


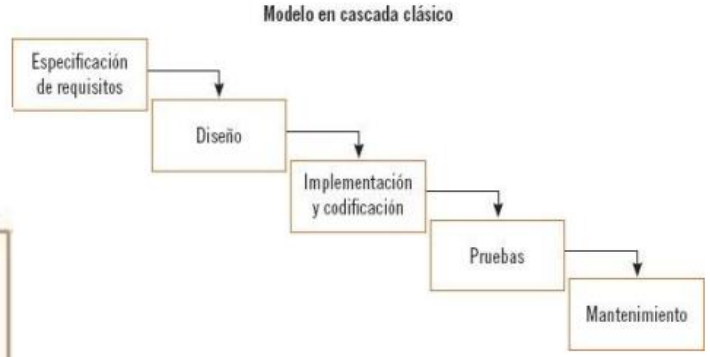
Modelo en cascada: desarrollos no excesivamente complejos, sin entregas parciales y requisitos claramente definidos

Modelo en cascada realimentado



Iterativo





Las ventajas son las siguientes:

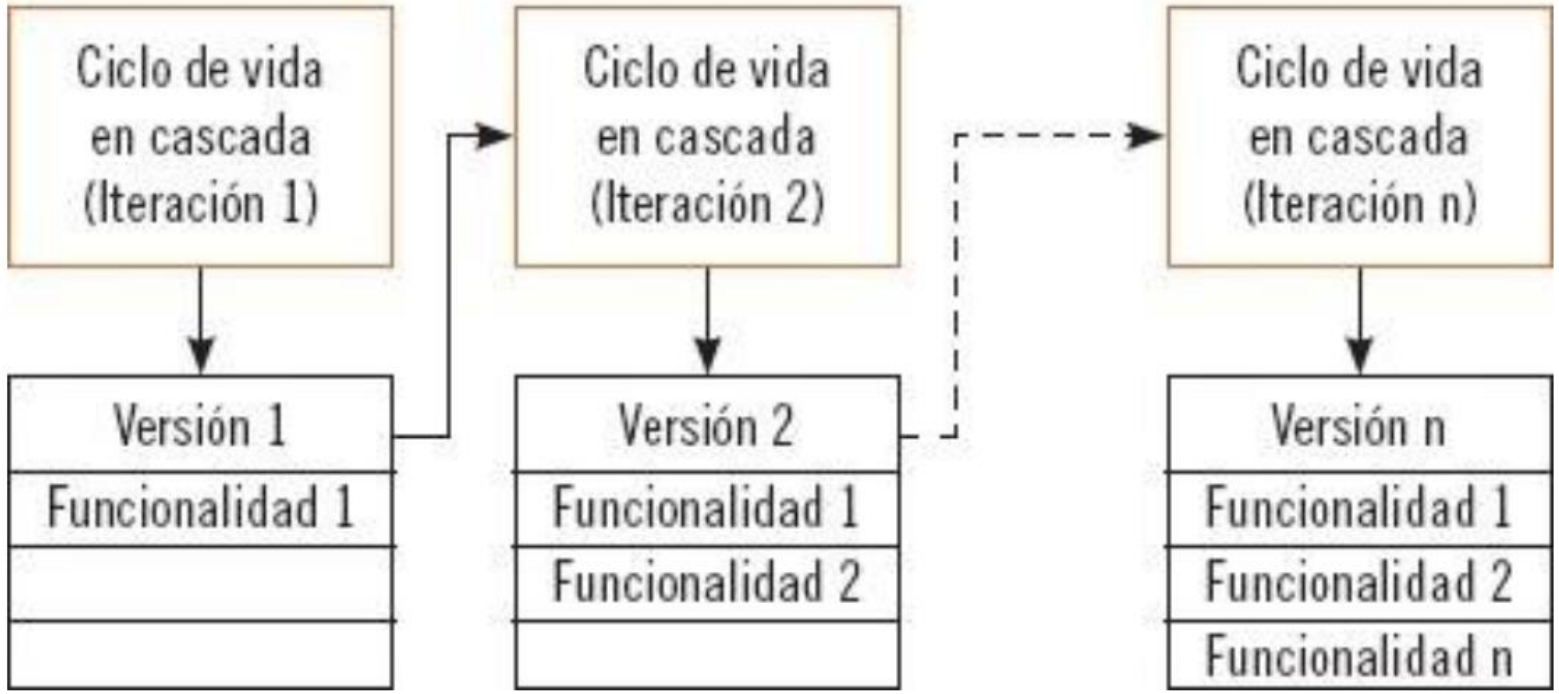
- Se ofrecen versiones intermedias, evaluables por el cliente. No requiere una alta especificación de requisitos.

Y las desventajas:

Si el cliente se involucra excesivamente puede cambiar su idea, modificando drásticamente en un desarrollo avanzado. Es difícil establecer un tiempo total de desarrollo.

Modelo iterativo: desarrollos para los cuales no se tienen suficientemente claros los requisitos o bien estos son cambiantes. Este modelo favorece la creación de prototipos e involucra al cliente desde el principio.

Modelo incremental



Las ventajas son las siguientes:

Entregas rápidas con funcionalidad creciente.

Y las desventajas:

No es válido para cualquier *software (software* con alta funcionalidad inicial, trabajo en sistemas distribuidos, trabajo en tiempo real, altos requerimientos de seguridad, etc.).

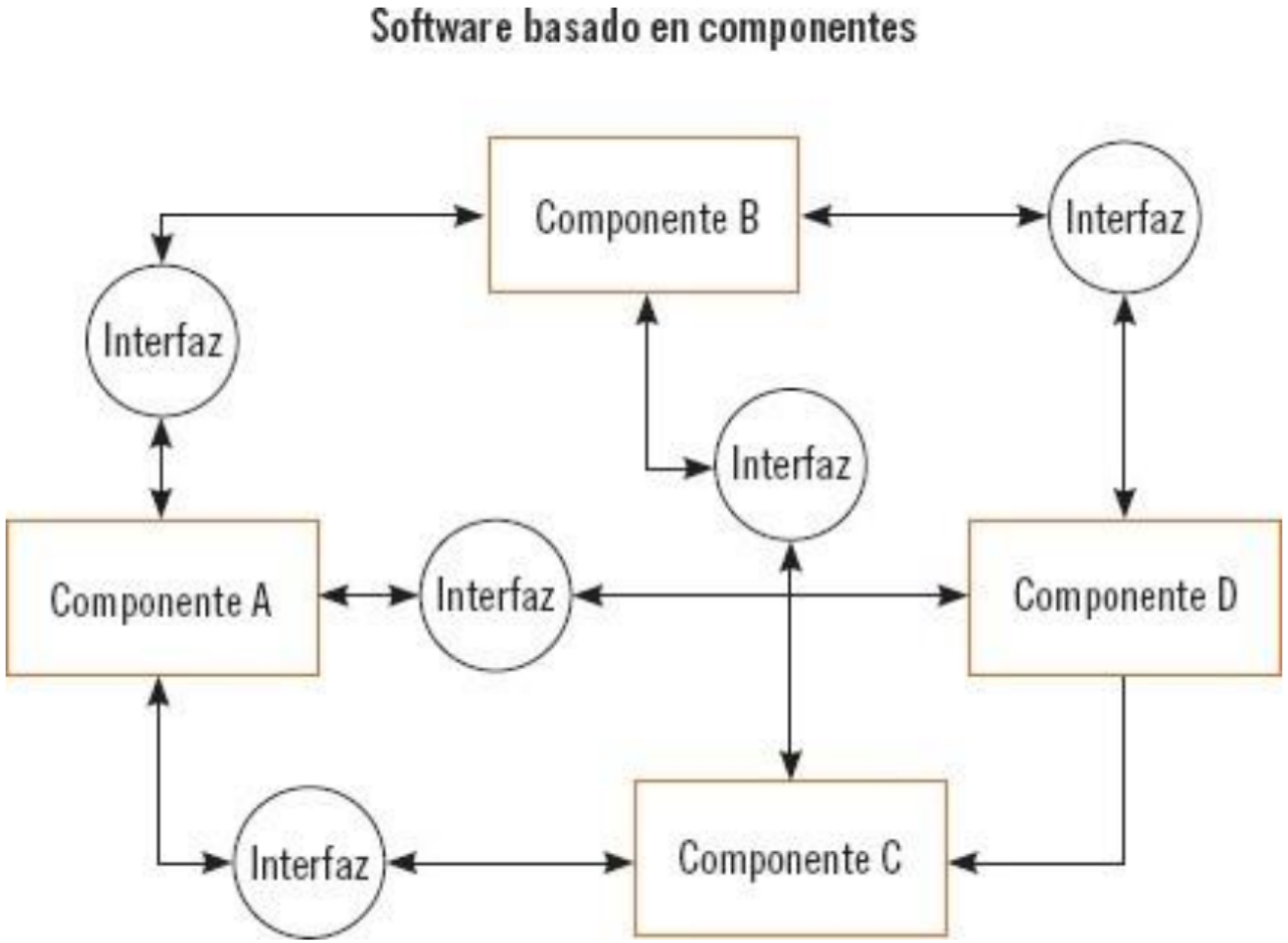
Basado en componentes (CBSE)

Las ventajas son las siguientes:

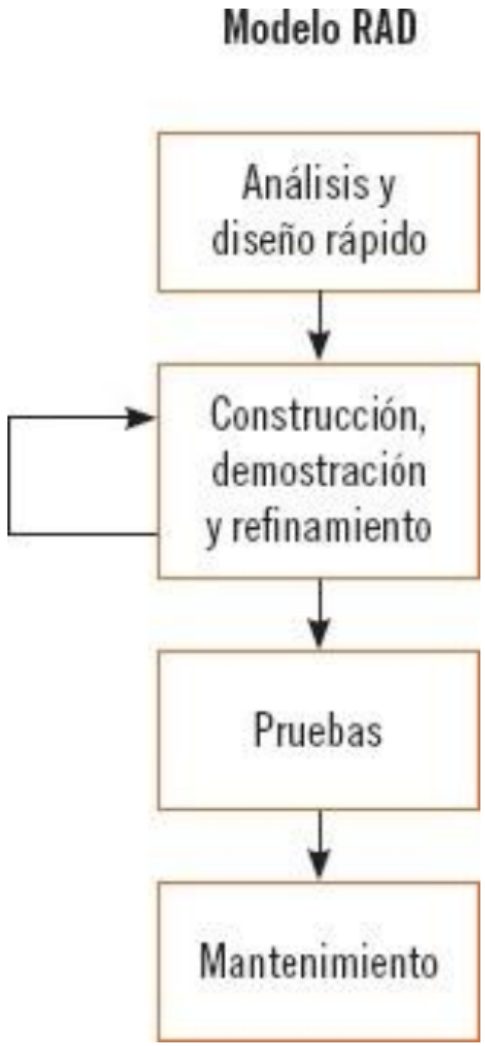
Reutilización de *software.*
Se puede reducir el tiempo de desarrollo.

Y los inconvenientes:

Los componentes pueden ser caros.
El componente, al ser externo, puede no ser actualizado durante la vida del *software.*
Hay que realizar pruebas exhaustivas.



Desarrollo rápido (RAD)



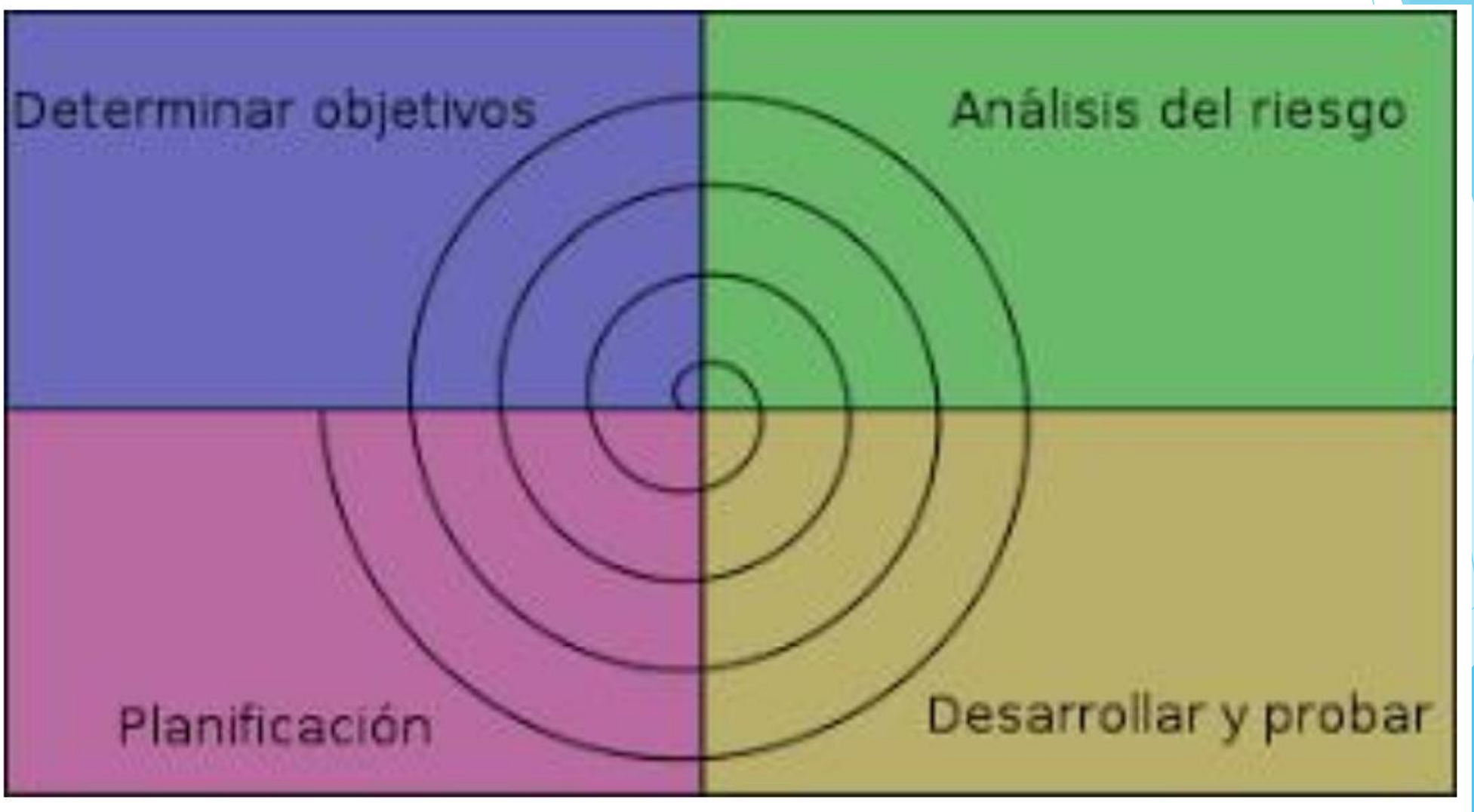
Las ventajas son las siguientes:

- Resultados rápidamente visibles.
- Permite la reusabilidad de código.
  Participación del usuario.

Y los inconvenientes:

- Exige bastante disciplina y compromiso por todas las partes.
- Gran coste en herramientas y entornos de desarrollo.
- Si el proyecto es grande, son necesarios varios equipos de trabajo.

Espiral


Determinar objetivos
Análisis del riesgo
Planificación
Desarrollar y probar

Resumen ciclos de vida software

Modelo en cascada: desarrollos no excesivamente complejos, sin entregas parciales y requisitos claramente definidos

Modelo iterativo: desarrollos para los cuales no se tienen suficientemente claros los requisitos o bien estos son cambiantes. Este modelo favorece la creación de prototipos e involucra al cliente desde el principio.

Modelo incremental: recomendable para *software* que no requiera toda su funcionalidad de manera inicial y cuando se precisen entregas rápidas.

Modelo en V: desarrollos que requieran gran robustez y confiabilidad. Por ejemplo, *software* que precise de operaciones constantes sobre una base de datos.

Modelo basado en componentes (CBSE): aplicable a desarrollos que han contemplado en su diseño la incorporación de componentes comerciales, siempre que se pueda afrontar su coste

Desarrollo rápido de aplicaciones (RAD): desarrollos rápidos, siempre que se disponga de suficiente equipo para hacer frente y cumplir los plazos

Requisitos

Especificación de requisitos

- Introducción.
- Descripción del problema
- Requisitos funcionales.
- Requisitos no funcionales.

Introducción

Una necesidad /problema.

Al igual que ponemos los caballos delante del carro, pongamos primero un problema y, después, una solución.

- Busquemos un problema sencillo (en su concepción).
- Por ejemplo una tienda de música online.

Descripción del problema

- Quiero vender música a través de Internet.
- Los usuarios comprarán créditos para adquirir canciones.
- Los usuarios buscarán las canciones que deseen y las pagarán con créditos.
- Los usuarios tendrán algunos días para descargar en su ordenador las canciones que hayan adquirido.
- Quiero hacer ofertas generales (afectan a todos los usuarios) y particulares (afectan a usuarios concretos).
- La solución es un sistema software.
- ¿Qué características debe tener este sistema para satisfacer las necesidades de nuestro cliente?.
- Esto es ingeniería de requisitos.

Requisitos funcionales

Los usuarios comprarán créditos para adquirir canciones.

Por lo tanto:

- El sistema debe registrar la información de los usuarios y los créditos que poseen.
- El sistema debe permitir que los usuarios registrados compren créditos y proporcionar las herramientas para que las usuarios paguen.

Los usuarios buscarán las canciones que deseen y las pagarán con créditos.

Por lo tanto:

- El sistema debe almacenar información sobre las canciones que se pueden adquirir y su precio en créditos.
- El sistema debe permitir a los usuarios buscar y consultar la información sobre las canciones.
- El sistema debe permitir a un usuario adquirir una canción a cambio de una cantidad de crédito.

Los usuarios tendrán algunos días para descargar en su ordenador las canciones que hayan adquirido.

Por lo tanto:

- El sistema debe almacenar las canciones adquiridas por un usuario y la fecha, para saber durante cuanto tiempo puede descargar dichas canciones.
- El sistema debe permitir descargar las canciones que un usuario ha adquirido mientras tenga tiempo.

Requisitos NO funcionales

¿Se os ocurren requisitos
(algo que la aplicación deba tener)
que no sea funcional?.

El sistema debe visualizarse y funcionar correctamente en cualquier navegador, especialmente en Edge, FireFox, Chrome

El sistema no debe tardar más de cinco segundos en mostrar los resultados de una búsqueda. Si se supera este plazo, el sistema detiene la búsqueda y muestra los resultados encontrados

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.

- **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: MagicDraw, Papyrus UML, Modellom ArgoUML, StartUML

LENGUAJES DE PROGRAMACIÓN

Un lenguaje de programación es un conjunto de instrucciones, operadores y reglas de sintaxis y semánticas, que se ponen a disposición del programador para que éste pueda comunicarse con los dispositivos de hardware y software existentes.

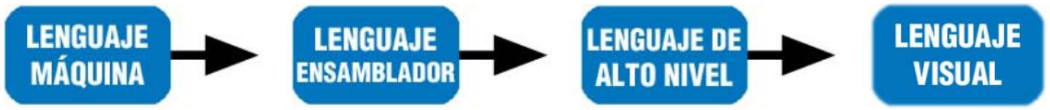
El sistema sólo es capaz de entender código escrito en código máquina (1s y 0s), programar directamente en código máquina es una tarea ardua y tediosa.

El uso de un lenguaje de programación tiene el objetivo de facilitar la tarea a los programadores permitiendo escribir programas utilizando un mayor nivel de abstracción en el código.

4. Lenguajes de programación
Concepto y características

LENGUAJES DE PROGRAMACIÓN

NIVEL DE ABSTRACCIÓN



El nivel de abstracción de un lenguaje implica cuan alejado está del código máquina, cuanto más parecido sea a nuestro lenguaje y menos al código máquina, de mayor nivel será el lenguaje.

Bajo nivel: Sólo hay un lenguaje de primera generación, el código máquina.

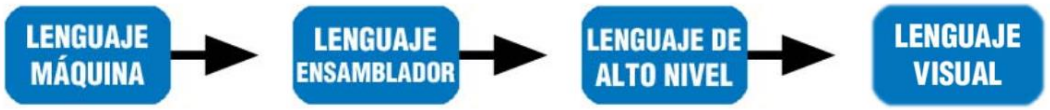
Medio nivel: Tienen definidas una serie de instrucciones sencillas para trabajar con datos simples y posiciones de memoria. El lenguaje clave de éste nivel es el lenguaje ensamblador.

Alto nivel: La mayoría de los lenguajes que se utilizan hoy en día para programar aplicaciones son de alto nivel. Son muy cercanos a nuestro propio lenguaje y permiten la realización de patrones de diseño complejos como Java o C#. También nos encontramos en éste nivel de abstracción a los lenguajes destinados a un propósito específico, como podría ser el SQL.

4. Lenguajes de programación
Concepto y características

LENGUAJES DE PROGRAMACIÓN

NIVEL DE ABSTRACCIÓN



El nivel de abstracción de un lenguaje implica cuan alejado está del código máquina, cuanto más parecido sea a nuestro lenguaje y menos al código máquina, de mayor nivel será el lenguaje.

Bajo nivel: Sólo hay un lenguaje de primera generación, el código máquina.

Medio nivel: Tienen definidas una serie de instrucciones sencillas para trabajar con datos simples y posiciones de memoria. El lenguaje clave de éste nivel es el lenguaje ensamblador.

Alto nivel: La mayoría de los lenguajes que se utilizan hoy en día para programar aplicaciones son de alto nivel. Son muy cercanos a nuestro propio lenguaje y permiten la realización de patrones de diseño complejos como Java o C#. También nos encontramos en éste nivel de abstracción a los lenguajes destinados a un propósito específico, como podría ser el SQL.

4. Lenguajes de programación

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)

Buscar ejemplos

4. Lenguajes de programación

Lenguajes de programación estructurados

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 (si se hace demasiado grande es difícil manejarlo).
- 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, donde los programas se codifican por módulos y bloques, permitiendo mayor funcionalidad.

4. Lenguajes de programación

Lenguaje de programación orientado a objetos

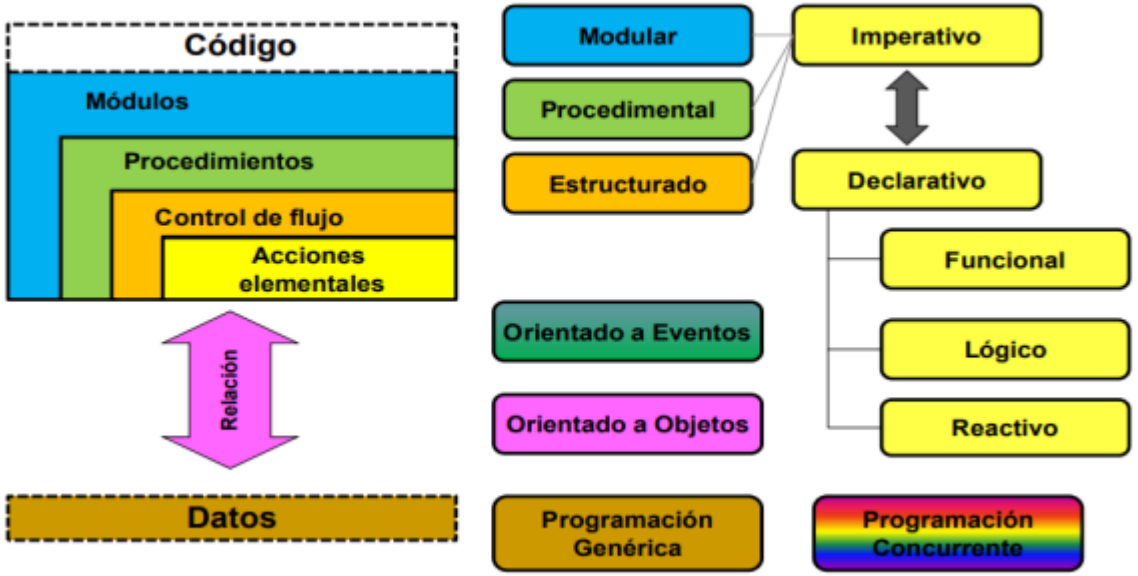
En la P.O.O. los programas se componen de objetos independientes entre sí que colaboran para realizar acciones.

Los objetos son reutilizables para proyectos futuros.

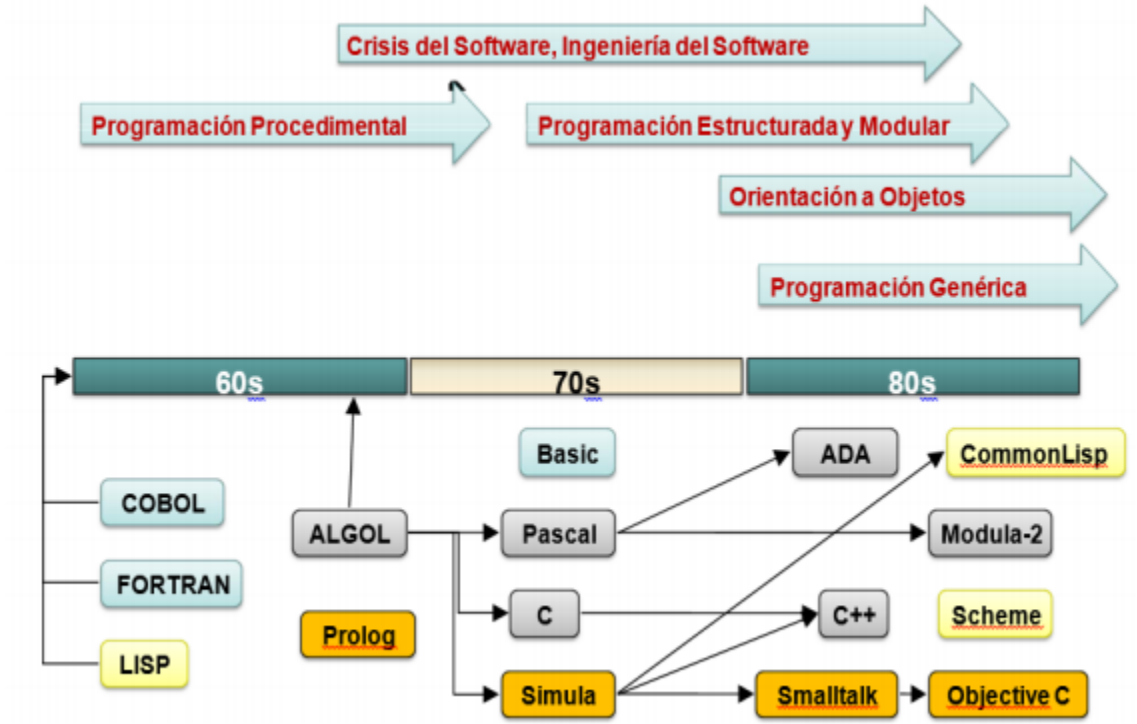
Características:

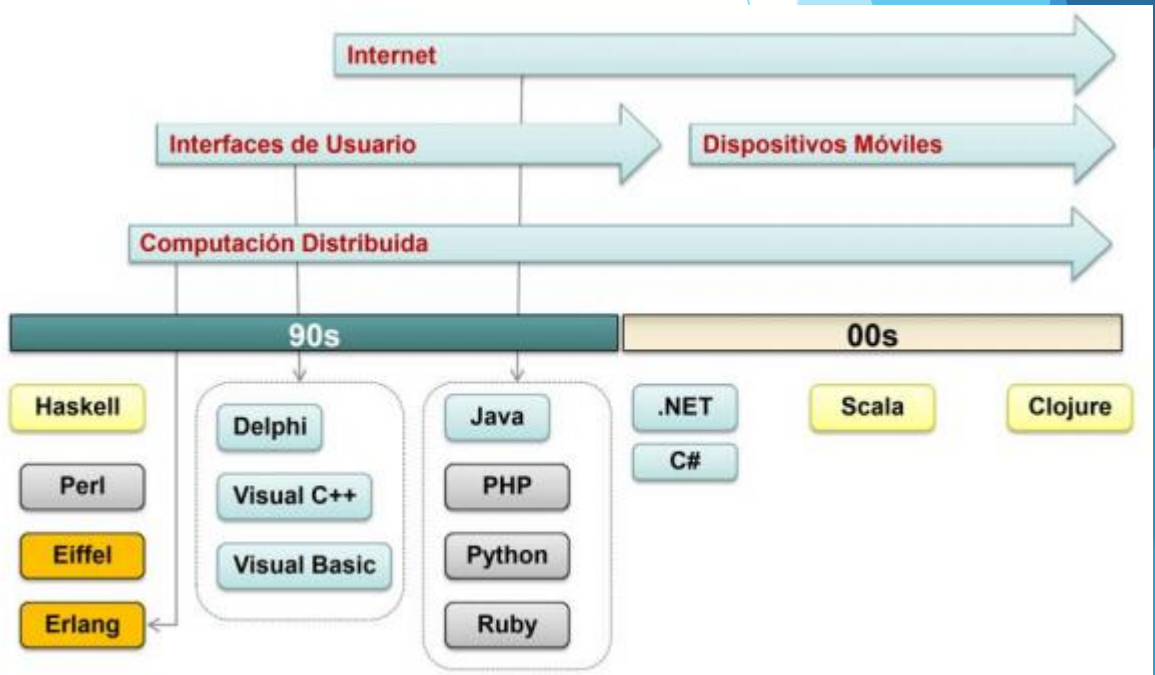
- Los objetos del programa tendrán una serie de atributos que los diferencian unos de otros.
- Se define clase como una colección de objetos con características similares.
- Mediante los llamados métodos, los objetos se comunican con otros produciéndose un cambio de estado de los mismos.
- Los objetos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

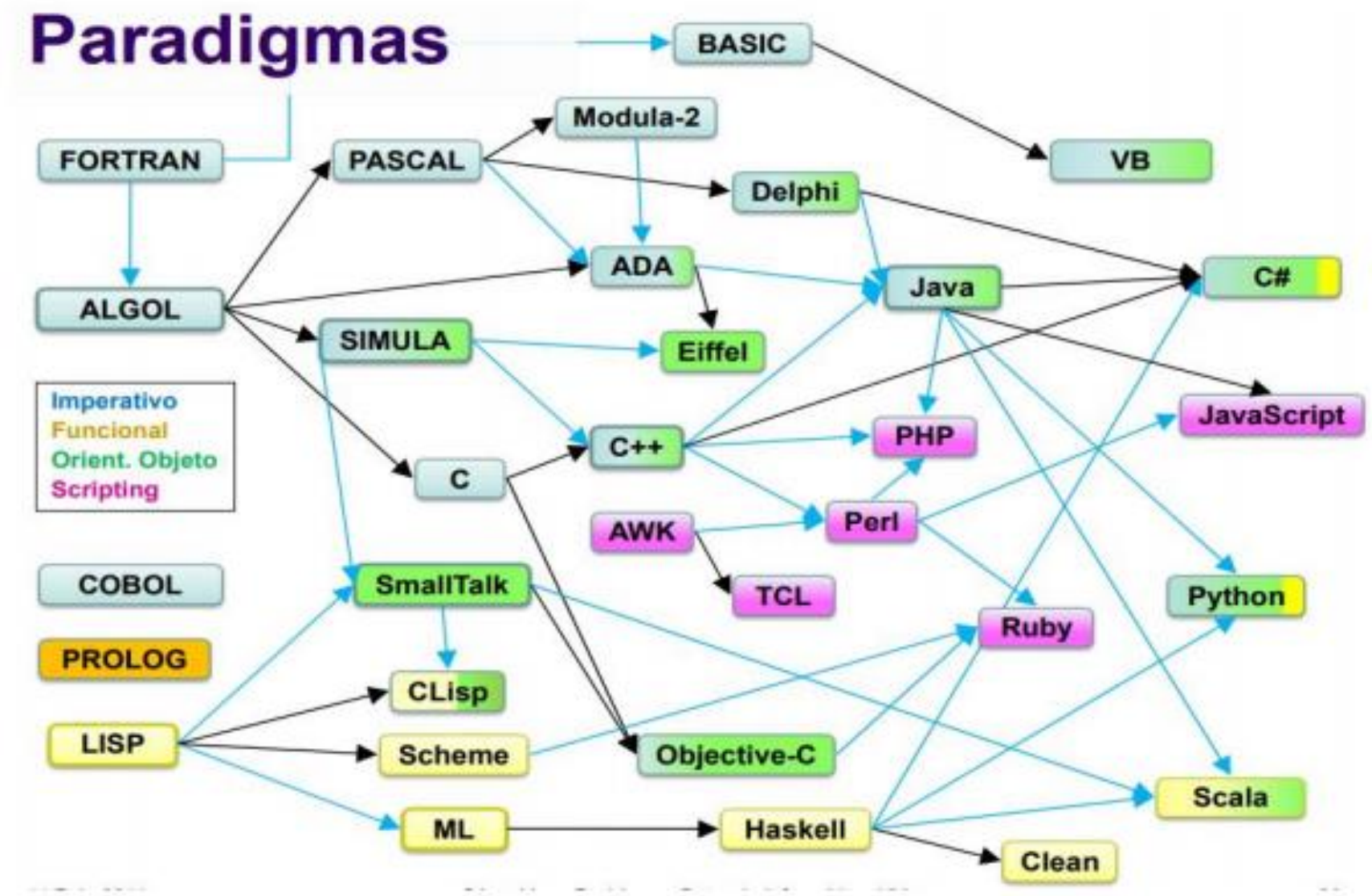
Buscar ejemplos



Generación	Lenguajes	Hardware	Movimientos
Primera (1945-55)	Código Máquina	Relés, Válvulas de vacío	
Segunda (1955-68)	FORTRAN COBOL LISP	Transistores, Memorias de ferrita	Prog. Estructurada y Modular Proceso por Lotes
Tercera (1968-1980)	ALGOL PASCAL C BASIC ADA	Circuitos integrados, Memorias de transistores	Ingeniería de Software Orientación a Objetos Bases de Datos
Cuarta (1980-)	C++ JAVA HASKELL PYTHON	VLSI MultiCore Flash	Comp. Distribuida Interfaces Gráficas Multimedia Internet






Paradigmas
BASIC
FORTRAN
PASCAL
Modula-2
VB
Delphi
ADA
Java
C#
ALGOL
SIMULA
Eiffel
Imperativo
Funcional
Orient. Objeto
Scripting
JavaScript
C++
PHP
C
AWK
Perl
COBOL
SmallTalk
TCL
Python
PROLOG
Ruby
CLisp
LISP
Scheme
Objective-C
Scala
ML
Haskell
Clean

FORMA DE EJECUCIÓN

Compilados: Un programa traductor (compilador) convierte el código fuente en código objeto para crear un programa ejecutable.

Interpretados: Ejecutan las instrucciones directamente, sin generar código objeto. La propia máquina se encargará de interpretarlo instrucción a instrucción.

Virtuales: Con un comportamiento similar al de los lenguajes compilados con una peculiaridad, el compilador no genera código objeto sino código bytecode. Una máquina virtual se encargará de interpretar el bytecode para ejecutarlo en el sistema.

LENGUAJES DE PROGRAMACIÓN

Imperativo: describe la programación como una secuencia de instrucciones que cambian el estado del programa, indicando cómo realizar una tarea.

Declarativo: especifica o declara un conjunto de premisas y condiciones para indicar qué es lo que hay que hacer y no necesariamente cómo hay que hacerlo.

Procedimental: el programa se divide en partes más pequeñas, llamadas funciones y procedimientos, que pueden comunicarse entre sí. Permite reutilizar código ya programado y solventa el problema de la programación spaghetti.

Orientado a objetos: encapsula el estado y las operaciones en objetos, creando una estructura de clases y objetos que emula un modelo del mundo real, donde los objetos realizan acciones e interactúan con otros objetos.

Funcional: evalúa el problema realizando funciones de manera recursiva, evita declarar datos haciendo hincapié en la composición de las funciones y en las interacciones entre ellas.

Lógico: define un conjunto de reglas lógicas para ser interpretadas mediante inferencias lógicas. Permite responder preguntas planteadas al sistema para resolver problemas.

5. Fases en el desarrollo y ejecución del software

Análisis

Diseño

Codificación. Tipos de código

Fases en la obtención de código

Maquinas virtuales

Pruebas

Documentación

Explotación

Mantenimiento

5. Fases en el desarrollo y ejecución del software

Análisis: La fase de análisis define los requisitos del software que hay que desarrollar.

Diseño: En esta etapa se pretende determinar el funcionamiento de una forma global y general, sin entrar en detalles.

Codificación: La fase más obvia en el proceso de desarrollo de software es sin duda la codificación. Es más que evidente que una vez definido el software que hay que crear haya que programarlo.

Maquinas virtuales:

Pruebas: Con una doble funcionalidad, las pruebas buscan confirmar que la codificación ha sido exitosa y que el software no contiene errores, a la vez que se comprueba que el software hace lo que debe hacer, que no necesariamente es lo mismo.

Documentación: Debe mostrar una información completa y de calidad que ilustre mediante los recursos más adecuados cómo manejar la aplicación. También se debe realizar una documentación técnica destinada a ser leída por los demás desarrolladores que trabajen en la aplicación.

Explotación: Una vez que tenemos nuestro software, hay que prepararlo para su distribución. Para ello se implementa el software en el sistema elegido o se prepara para que se implemente por sí solo de manera automática.

Mantenimiento: En esta fase del desarrollo de un software se arreglan los fallos o errores que suceden cuando el programa ya ha sido implementado en un sistema y se realizan las ampliaciones necesitadas o requeridas.

5. Fases en el desarrollo y ejecución del software

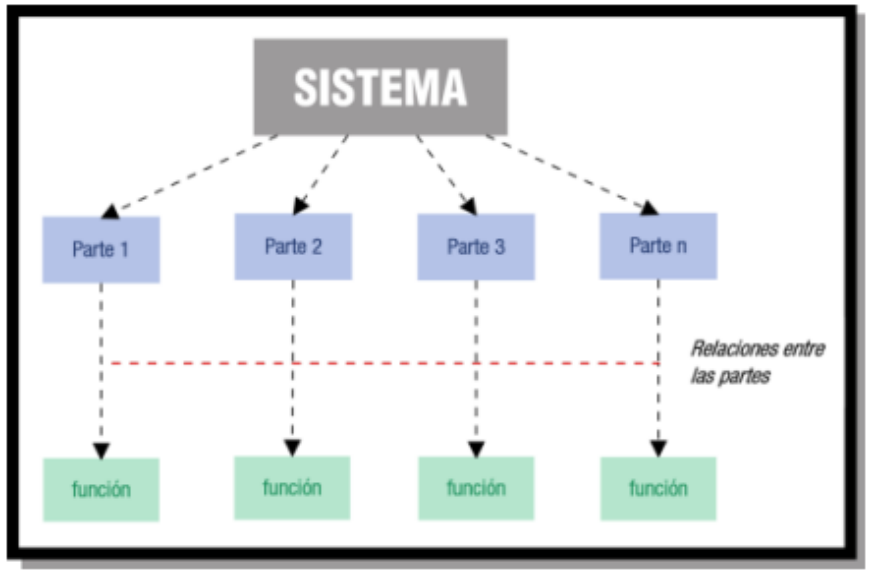
Análisis: La fase de análisis define los requisitos del software que hay que desarrollar.

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.
- **No funcionales**: Tiempos de respuesta del programa, legislación aplicable, tratamiento ante la simultaneidad de peticiones, etc.

5. Fases en el desarrollo y ejecución del software

Diseño: En esta etapa se pretende determinar el funcionamiento de una forma global y general, sin entrar en detalles.



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

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

5. Fases en el desarrollo y ejecución del software

Codificación: La fase más obvia en el proceso de desarrollo de software es sin duda la codificación.

Es más que evidente que una vez definido el software que hay que crear haya que programarlo.

Las características deseables de todo código son:

1. Modularidad: que esté dividido en trozos más pequeños.
2. Corrección: que haga lo que se le pide realmente.
3. Fácil de leer: para facilitar su desarrollo y mantenimiento futuro.
4. Eficiencia: que haga un buen uso de los recursos.
5. Portabilidad: que se pueda implementar en cualquier equipo.

Codificación –Tipos de código

Código fuente: el código fuente de un programa informático es un conjunto de instrucciones escritas en un lenguaje de programación determinado. Es decir, es el código en el que nosotros escribimos nuestro programa.

Código objeto: el código objeto es el código resultante de compilar el código fuente. Si se trata de un lenguaje de programación compilado, el código objeto será código máquina, mientras que si se trata de un lenguaje de programación virtual, será código bytecode.

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

Codificación –Fases de compilación - OBTENCIÓN DE CÓDIGO EJECUTABLE

Práctica

5. Fases en el desarrollo y ejecución del software

Maquinas virtuales

Una máquina virtual es un tipo especial de software cuya misión es separar el funcionamiento del ordenador de los componentes hardware instalados.

Vagrant es una herramienta gratuita de línea de comandos, disponible **para** Windows, MacOS X y GNU/Linux, que permite generar **entornos de desarrollo** reproducibles y compartibles de forma muy sencilla.
Para ello, **Vagrant** crea y configura máquinas virtuales a partir de simples ficheros de configuración.

5. Fases en el desarrollo y ejecución del software

Maquinas virtuales .- Frameworks

Un framework (*Plataforma, entorno, marco de trabajo del desarrollo rápido de aplicaciones*) es una estructura de ayuda al programador, en base a la cual podemos desarrollar proyectos sin partir desde cero.

- Ventajas de utilizar un framework:
  - **Desarrollo rápido** de software.
  - **Reutilización** de partes de código para otras aplicaciones
  - **Diseño** uniforme del software
  - **Portabilidad** de aplicaciones de un ordenador a otro
- Inconvenientes:
  - Gran dependencia del código respecto al framework utilizado (sin cambiamos de framework, habrá que reescribir gran parte de la aplicación).
  - La instalación e implementación del frameworl en nuestro equipo consume recursos del sistema

5. Fases en el desarrollo y ejecución del software

Pruebas: Con una doble funcionalidad, las pruebas buscan confirmar que la codificación ha sido exitosa y que el software no contiene errores, a la vez que se comprueba que el software hace lo que debe hacer, que no necesariamente es lo mismo.

PRUEBAS UNITARIAS

Consisten en probar, una a una, las diferentes partes de software y comprobar su funcionamiento (por separado, de manera independiente). Junit es el entorno de pruebas para Java.

PRUEBAS DE INTEGRACIÓN

Se realizan una vez que se han realizado con éxito las pruebas unitarias y consistirán en comprobar el funcionamiento del sistema completo: con todas sus partes interrelacionadas.

5. Fases en el desarrollo y ejecución del software

Documentación: Debe mostrar una información completa y de calidad que ilustre mediante los recursos más adecuados cómo manejar la aplicación. También se debe realizar una documentación técnica destinada a ser leída por los demás desarrolladores que trabajen en la aplicación.

Documentos a elaborar en el proceso de desarrollo de software			
	Guía Técnica	Guía de Uso	Guía de Instalación
Quedan reflejados	El diseño de la app La codificación de los programas Las pruebas realizadas	• Descripción de la funcionalidad de la aplicación. • Forma de comenzar a ejecutar la aplicación. • Ejemplos de uso del programa. • Requerimientos software de la aplicación. • Solución de los posibles problemas que se pueden presentar.	Toda la información necesaria para: • Puesta en marcha. • Explotación. • Seguridad del sistema.
¿A Quien 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.

5. Fases en el desarrollo y ejecución del software

Explotación: Una vez que tenemos nuestro software, hay que prepararlo para su distribución. Para ello se implementa el software en el sistema elegido o se prepara para que se implemente por sí solo de manera automática.

La explotación

es la fase en que los usuarios finales conocen la

aplicación y comienzan a utilizarla.

5. Fases en el desarrollo y ejecución del software

Mantenimiento: En esta fase del desarrollo de un software se arreglan los fallos o errores que suceden cuando el programa ya ha sido implementado en un sistema y se realizan las ampliaciones necesitadas o requeridas.

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. Por tanto, serán necesarias modificaciones, expansiones o eliminaciones de código.
- **Adaptativos**: Modificaciones, actualizaciones... para adaptarse a las nuevas tendencias del mercado, a nuevos componentes hardware, etc.
- **Correctivos**: La aplicación tendrá errores en el futuro (sería utópico pensar lo contrario).

Ejemplo

SENTENCIAS SECUENCIALES

Las sentencias secuenciales son aquellas que se ejecutan una detrás de la otra, según el orden en que hayan sido escritas.

Ejemplo en lenguaje C:

```
printf ("declaración de variables");
int numero_entero;
espacio=espacio_inicio + veloc*tiempo;
```

SENTENCIAS SELECTIVAS (CONDICIONALES)

Ejemplo en lenguaje C:

```
if (a >= b)
c= a-b;
else
c=a+b;
```

SENTENCIAS REPETITIVAS (ITERACIONES O BUCLES)

Ejemplo en lenguaje C:

```
int num;
num = 0;
while (num<=10) {
printf("Repetición numero %d\n", num);
num = num + 1;
}
```

ROLES QUE INTERACTÚAN EN EL DESARROLLO

Durante el proceso de desarrollo de un software interviene un personal con diferentes roles.

Analista de sistemas: Su objetivo consiste en realizar un estudio del sistema para dirigir el proyecto en una dirección que garantice las expectativas del cliente determinando el comportamiento del software. Participa en la etapa de análisis.

Diseñador de software: Nace como una evolución del analista y realiza, en función del análisis de un software, el diseño de la solución que hay que desarrollar.

Participa en la etapa de diseño.

Analista programador: Aporta una visión general del proyecto más detallada diseñando una solución más amigable para la codificación y participando activamente en ella.

Participa en las etapas de diseño y codificación.

Programador: Escribe el código fuente en función al estudio realizado por analistas y diseñadores. Participa en la etapa de codificación.

Arquitecto de software: Conoce e investiga los frameworks y tecnologías revisando que todo el procedimiento se lleva a cabo de la mejor forma y con los recursos más apropiados.

Participa en las etapas de análisis, diseño, documentación y explotación.

UD1 - DESARRROLLO DE SOFTWARE

6. ARQUITECTURA DE SOFTWARE

- DESARROLLO EN TRES CAPAS

El desarrollo en capas nace de la necesidad de separar la lógica de la aplicación del diseño, separando a su vez los datos de la presentación al usuario

El desarrollo por capas no solo nos mejora y facilita la estructura de nuestro propio software, sino que nos aporta la posibilidad de interoperar con otros sistemas ajenos a nuestra aplicación.

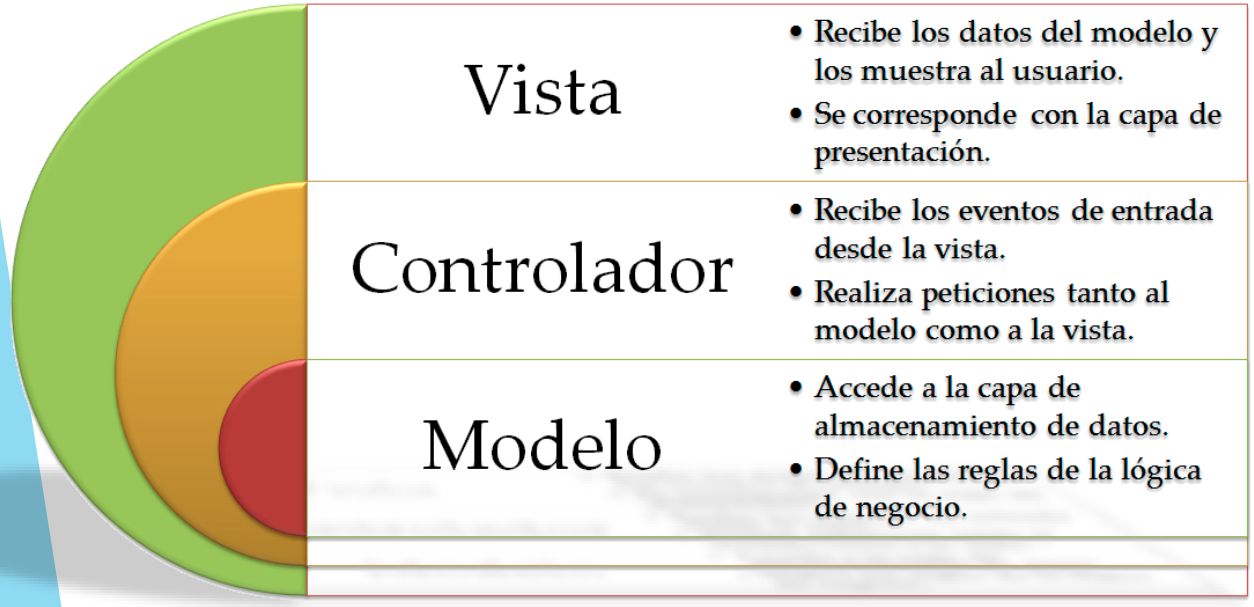
UD1 - DESARRROLLO DE SOFTWARE

ARQUITECTURA DE SOFTWARE

DESARROLLO EN TRES CAPAS

Modelo Vista Controlador

El MVC define tres componentes para las capas del desarrollo del software, organiza el código mediante unas directrices específicas utilizando un criterio basado en la funcionalidad y no en las características del componente en
sí mismo.

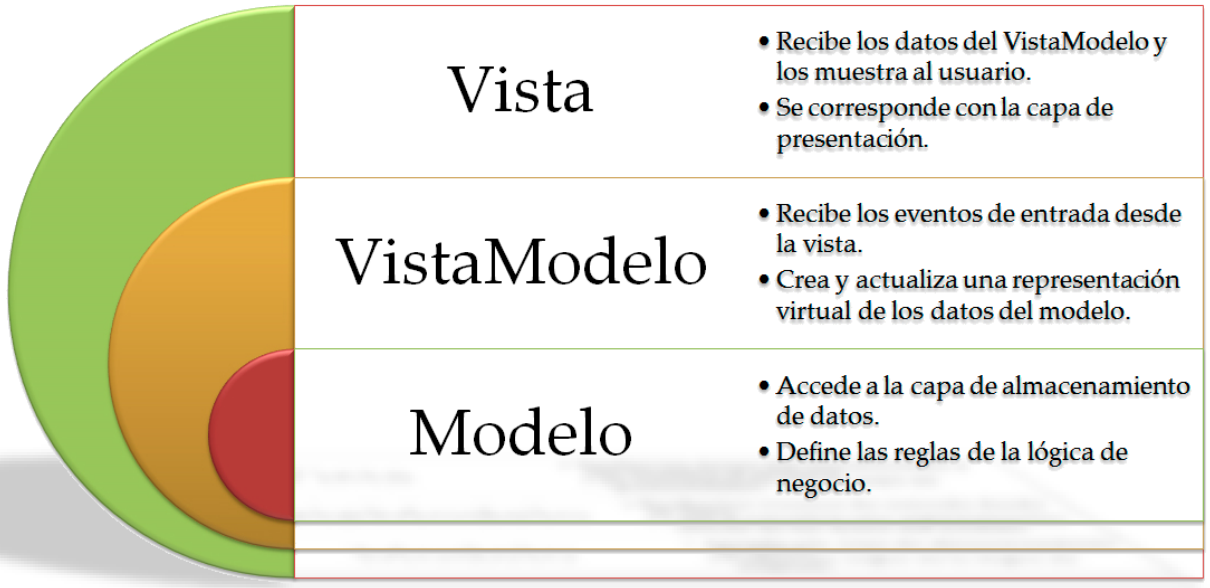


Se suele establecer una serie de "bindings" que enlacen diferentes componentes de la vista a propiedades y campos de las entidades.

UD1 - DESARRROLLO DE SOFTWARE

ARQUITECTURA DE SOFTWARE

DESARROLLO EN TRES CAPAS



Modelo Vista VistaModelo

EL MVVM parte de un concepto muy similar al modelo MVC. A diferencia del MVC, la vista del MVVM es un observador que se actualiza cuando cambia la información contenida en el VistaModelo.

Los eventos de la vista son recogidos por el controlador, pero a diferencia del MVC los datos de la vista son obtenidos y actualizados a través del Vista Modelo, dejando al modelo como una mera representación de las entidades.