Tema 1: Desarrollo de software

1. *Introducción. Conocimientos previos.*

Fuente: www.jorgesanchez.net

computadora y sistema operativo

computadora

Según la RAE (Real Academia de la lengua española), una computadora es una *máquina electrónica, analógica o digital, dotada de una memoria de gran capacidad y de métodos de tratamiento de la información, capaz de resolver problemas matemáticos y lógicos mediante la utilización automática de programas informáticos*.

Sin duda esta máquina es la responsable de toda una revolución que está cambiando el panorama económico, social e incluso cultural. Debido a la importancia y al difícil manejo de estas máquinas, aparece la **informática** como la ciencia orientada al proceso de información mediante el uso de computadoras.

Una computadora consta de diversos componentes entre los que sobresale el procesador, el componente que es capaz de realizar las tareas que se requieren al ordenador o computadora.

En realidad un procesador sólo es capaz de realizar tareas sencillas como:

- **Operaciones aritméticas simples**: suma, resta, multiplicación y división
- **Operaciones de comparación entre valores**
- **Almacenamiento de datos**

Algunos de los componentes destacables de un ordenador son:

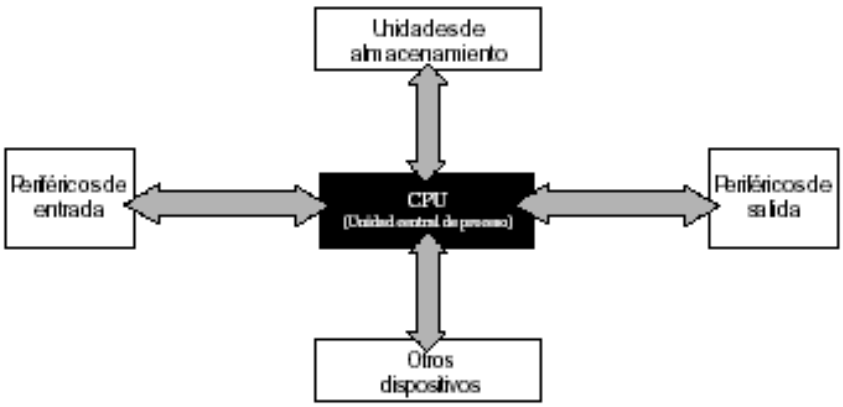


Ilustración 1, componentes de un ordenador desde un punto de vista lógico

Este desglose de los componentes del ordenador es el que interesa a los programadores. Pero desde un punto de vista más físico, hay otros componentes a señalar:

- **Procesador.** Núcleo digital en el que reside la CPU del ordenador. Es la parte fundamental del ordenador, la encargada de realizar todas las tareas.
- **Placa base.** Circuito interno al que se conectan todos los componentes del ordenador, incluido el procesador.

- **Memoria RAM.** Memoria interna formada por un circuito digital que está conectado mediante tarjetas a la placa base. Su contenido se evapora cuando se desconecta al ordenador. Lo que se almacena no es permanente.
- **Memoria caché.** Memoria ultrarrápida de características similares a la RAM, pero de velocidad mucho más elevada por lo que se utiliza para almacenar los últimos datos utilizados.
- **Periféricos.** Aparatos conectados al ordenador mediante tarjetas o ranuras de expansión (también llamados puertos). Los hay de **entrada** (introducen datos en el ordenador: teclado, ratón, escáner,...), de **salida** (muestran datos desde el ordenador: pantalla, impresora, altavoces,...) e incluso de **entrada/salida** (módem, tarjeta de red).
- **Unidades de almacenamiento.** En realidad son periféricos, pero que sirven para almacenar de forma permanente los datos que se deseen del ordenador. Los principales son el **disco duro** (unidad de gran tamaño interna al ordenador), la **disquetera** (unidad de baja capacidad y muy lenta, ya en desuso), el **CD-ROM** y el **DVD**.

[1.1.2] hardware y software

hardware

Se trata de todos los componentes físicos que forman parte de un ordenador: procesador, RAM, impresora, teclado, ratón,...

software

Se trata de la parte conceptual del ordenador. Es decir los datos y aplicaciones que maneja y que permiten un grado de abstracción mayor. Cualquier cosa que se pueda almacenar en una unidad de almacenamiento es software (la propia unidad sería hardware).

[1.1.3] Sistema Operativo

Se trata del software (programa) encargado de gestionar el ordenador. Es la aplicación que oculta la física real del ordenador para mostrarnos un interfaz que permita al usuario un mejor y más fácil manejo de la computadora.

funciones del Sistema Operativo

Las principales funciones que desempeña un Sistema Operativo son:

- Permitir al usuario comunicarse con el ordenador. A través de comandos o a través de una interfaz gráfica.
- Coordinar y manipular el hardware de la computadora: memoria, impresoras, unidades de disco, el teclado,...
- Proporcionar herramientas para organizar los datos de manera lógica (carpetas, archivos,...)
- Proporcionar herramientas para organizar las aplicaciones instaladas.
- Gestionar el acceso a redes
- Gestionar los errores de hardware y la pérdida de datos.

- Servir de base para la creación de aplicaciones, proporcionando funciones que faciliten la tarea a los programadores.
- Administrar la configuración de los usuarios.
- Proporcionar herramientas para controlar la seguridad del sistema.

algunos sistemas operativos

- **Windows.** A día de hoy el Sistema Operativo más popular (instalado en el 95% de computadoras del mundo). Es un software propiedad de Microsoft por el que hay que pagar por cada licencia de uso.

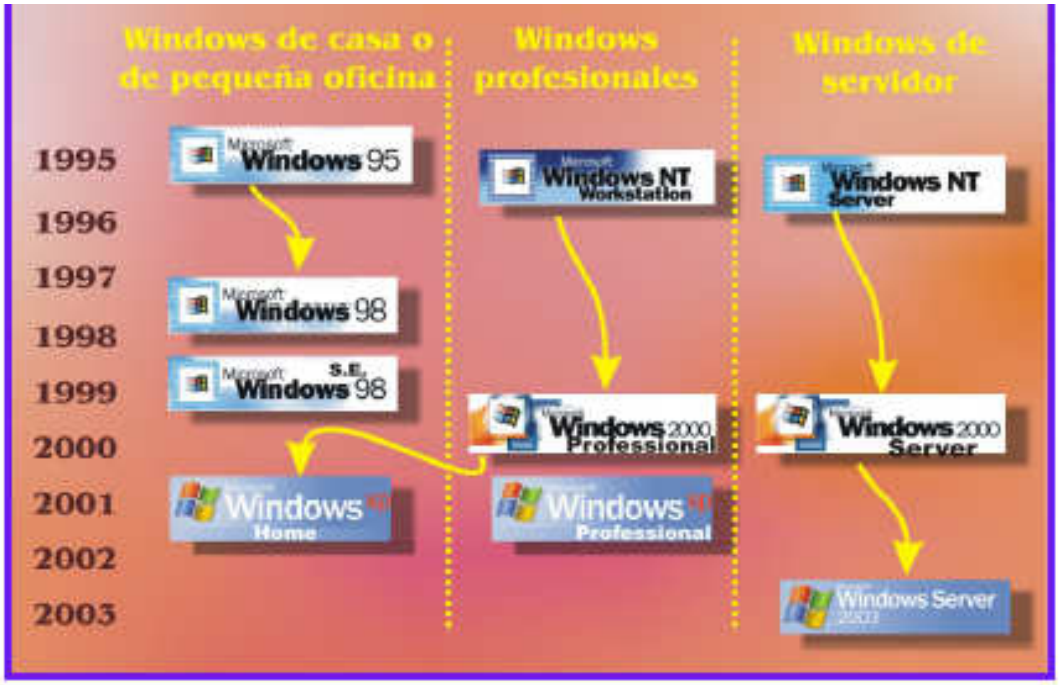


Ilustración 2, Versiones actuales de Windows

- **Linux.** Sistema operativo de código abierto. Posee numerosas distribuciones (muchas de ellas gratuitas) y software adaptado para él (aunque sólo el 15% de ordenadores posee Linux).
- **MacOs.** Sistema operativo de los ordenadores MacIntosh.
- **Unix.** Sistema operativo muy robusto para gestionar redes de todos los tamaños. Actualmente en desuso debido al uso de Linux (que está basado en Unix).
- **Solaris.** Versión de Unix para sistemas Sun.

2. Concepto de programa informático. Lenguajes de programación. Código fuente, código objeto y código ejecutable.

La definición de la **RAE** es: Conjunto unitario de instrucciones que permite a un ordenador realizar funciones diversas, como el tratamiento de textos, el diseño de gráficos, la resolución de problemas matemáticos, el manejo de bancos de datos, etc. Pero normalmente se entiende por programa un conjunto de instrucciones ejecutables por un ordenador.

Un programa estructurado es un programa que cumple las condiciones de un algoritmo (finitud, precisión, repetición, resolución del problema,...).

Aplicación. Software formado por uno o más programas, la documentación de los mismos y los archivos necesarios para su funcionamiento, de modo que el conjunto completo de archivos forman una herramienta de trabajo en un ordenador.
Normalmente en el lenguaje cotidiano no se distingue entre aplicación y programa; en nuestro caso entenderemos que la aplicación es un software completo que cumple la función completa para la que fue diseñado, mientras que un programa es el resultado de ejecutar un cierto código entendible por el ordenador.

Para el desarrollo o construcción de un Programa se utiliza uno o varios Lenguajes de Programación, podemos decir que un **lenguaje de programación (Ver anexo I al final del tema)** es una notación o conjunto de símbolos y caracteres combinados entre sí de acuerdo con una sintaxis ya definida para posibilitar la transmisión de instrucciones a la CPU, dichos símbolos son traducidos a un conjunto de señales eléctricas representadas en código binario (0 y 1) que es el único código comprensible por el microprocesador.

Los lenguajes de programación se clasifican en dos grandes grupos:

- Lenguajes de bajo nivel
  - Lenguaje máquina. Prácticamente no utilizado.
  - Lenguaje Ensamblador.
- Lenguajes de alto nivel.

  Pueden establecerse actualmente varias subdivisiones en los lenguajes de alto nivel pero dependen de los autores y no están estandarizadas.

Bajo Nivel. LENGUAJES MÁQUINA

Son los escritos en lenguaje directamente entendible por la computadora, dado que sus instrucciones son cadenas binarias. El código máquina es el código binario. El conjunto de instrucciones depende de la CPU del ordenador.
Ventajas:

- Velocidad, se transfiere el código a la memoria directamente.

Inconvenientes:

- Lentitud y dificultad en la codificación.
- Poca fiabilidad.
- Dificultad en la depuración y puesta a punto.
- Los programas sólo son ejecutables en el mismo procesador.

Bajo Nivel. LENGUAJE ENSAMBLADOR

Son algo más fáciles de utilizar que los de bajo nivel, pero también dependen de la máquina en la que se utilice.

El lenguaje por excelencia es el ensamblador (assembly languaje).

Las instrucciones se denominan mnemotécnicos ADD, SUB, MOV, etc.

El lenguaje escrito no puede ser ejecutado directamente por la máquina, requiere un traductor. El original se denomina programa fuente y el traducido u objeto con extensión. COM ya es ejecutable por el ordenador. El traductor se denomina ensamblador (assembler) no confundir con el lenguaje de programación ensamblador que tiene su

estructura y gramática definida. Presentan la ventaja frente al lenguaje máquina de la facilidad de codificación.
Inconvenientes:

- Dependencia de la máquina, microprocesador.
- La creación de los programas es mucho más compleja que los de alto nivel. especialmente debido a la necesidad del conocimiento del interior del ordenador.

Se utilizan de forma reducida en programas de aplicaciones en tiempo real, control de procesos y de dispositivos electrónicos.

LENGUAJES DE ALTO NIVEL

Son los más utilizados al estar diseñados para que los programas se entiendan de forma más clara y fácil que los lenguajes máquina y ensambladores. Pueden ser ejecutados en cualquier ordenador de forma independiente al microprocesador utilizado. Esta característica los hace portables o transportables.

Presentan las ventajas:

- Tiempo de formación relativamente corto para los programadores.
- Las reglas de escritura se parecen a las utilizadas en el lenguaje humano, while, until, declare, etc.
- Las modificaciones y puesta a punto de programas son más fáciles.
- Reducción del coste de desarrollo.
- Portabilidad.

Los inconvenientes:

- Se explotan peor los recursos de la máquina.
- La conversión a programa ejecutable es más compleja.
- Aumento de la ocupación de memoria.
- Su ejecución es más lenta.

Los programas fuente deben de ser traducidos para ser ejecutados, los traductores son de dos tipos compiladores e interpretes, aunque para ser convertidos a ejecutables desde el sistema todos deben ser compilados.

Los lenguajes de alto nivel son muy numerosos aunque el uso más frecuente se centra en:

C/C++, COBOL, Pascal/Delphi, Quick/Visual Basic, Java.

están muy extendidos:

Ada-95, Módula-2, Prolog, LISP, Smalltack, Eiffel.

en el mundo profesional:

Delphi, Visual Basic, C++Builder, Power Builder.

en el mundo Internet:

Java, HTML, XML, Java Script, Visual J, Visual Basic, VBscript.

Alto Nivel. TRADUCTORES DE LENGUAJE

Son programas que traducen los programas fuente escritos en lenguajes de alto nivel a código ejecutable, se dividen en intérpretes y compiladores.

- Intérpretes

Es un traductor que tomando un programa fuente lo traduce en memoria y a continuación lo ejecuta. Los traductores clásicos fueron las versiones del BASIC del cual queda en vigor la versión QuickBASIC, también una versión interpretada del lenguaje Smalltak, que es el orientado a objetos por excelencia, y especialmente Java.

Programa fuente → Interprete → Traducción y ejecución en línea

- Compiladores

Es un programa que traduce los programas escritos en lenguajes de alto nivel a lenguaje máquina. Los programas obtenidos de la compilación de un programa fuente se denominan programas objeto o código objeto. Éste programa obtenido no es directamente ejecutable pues es simplemente la traducción línea a línea del fuente.

Programa fuente → Compilador → Programa objeto

La compilación y sus fases

Una vez obtenido un código objeto mediante la compilación, se hace necesario el montaje de sus módulos mediante un programa montador o enlazador (Linker). Este proceso genera un programa en código máquina directamente ejecutable.

Las fases del proceso de compilación:

Programa fuente → Compilador → Programa objeto → Montador → Ejecutable

Los pasos para convertir un programa escrito en programa ejecutable son:

- Escritura del programa fuente mediante un editor y almacenar en disco.
- Carga del programa fuente en memoria.
- Compilación
- Verificar errores de compilación. Listado de errores según programa.
- Obtención del programa objeto.
- Montaje mediante un enlazador y obtención de programa ejecutable.
- Ejecución del programa, si no existen errores se tendrá la salida adecuada.

En la actualidad casi todas las herramientas de programación tienen un IDE o Entorno Integrado de Desarrollo, éste consta de un editor con las funciones necesarias para la compilación y una serie de funciones que facilitan la depuración de los programas.

Cuando un programa obtiene una salida que no es la esperada, se dice que posee errores. Los errores son uno de los caballos de batalla de los programadores ya que a veces son muy difíciles de encontrar (de ahí que hoy en día en muchas aplicaciones se distribuyan parches para subsanar errores no encontrados en la creación de la aplicación).

Tipos de errores:

Error del usuario. Errores que se producen cuando el usuario realiza algo inesperado y el programa no reacciona apropiadamente.

Error del programador. Son errores que ha cometido el programador al generar el código. La mayoría de errores son de este tipo.

Errores de documentación. Ocurren cuando la documentación del programa no es correcta y provoca fallos en el manejo.

Error de interfaz. Ocurre si la interfaz de usuario de la aplicación es enrevesada para el usuario impidiendo su manejo normal. También se llaman así los errores de protocolo entre dispositivos.
Error de entrada / salida o de comunicaciones. Ocurre cuando falla la comunicación entre el programa y un dispositivo (se desea imprimir y no hay papel, falla el teclado,...)
Error fatal. Ocurre cuando el hardware produce una situación inesperado que el software no puede controlar (el ordenador se cuelga, errores en la grabación de datos,...)
Error de ejecución. Ocurren cuando la ejecución del programa es más lenta de lo previsto.
La labor del programador es predecir, encontrar y subsanar (si es posible) o al menos controlar los errores. Una mala gestión de errores causa experiencias poco gratas al usuario de la aplicación.

3. *Generaciones, lenguajes y ordenadores (lectura recomendada).*

En el siguiente artículo podemos comprobar que todo está relacionado, los conceptos no se pueden aislar en le mundo de la informática, esta situación ha provocado que se esté utilizando un concepto más amplio para hablar de informática: las TICS, Tecnologías de la Información y la Comunicación.

Fuente: www.jorgesanchez.net

Charles Babbage definió a mediados del siglo XIX lo que él llamó la **máquina analítica**. Se considera a esta máquina el diseño del primer ordenador. La realidad es que no se pudo construir hasta el siglo siguiente. El caso es que su colaboradora **Ada Lovelace** escribió en tarjetas perforadas una serie de instrucciones que la máquina iba a ser capaz de ejecutar. Se dice que eso significó el inicio de la ciencia de la programación de ordenadores.
En la segunda guerra mundial debido a las necesidades militares, la ciencia de la computación prospera y con ella aparece el famoso **ENIAC** (***Electronic Numerical Integrator And Calculator***), que se programaba cambiando su circuitería. Esa es la primera forma de programar (que aún se usa en numerosas máquinas) que sólo vale para **máquinas de único propósito**. Si se cambia el propósito, hay que modificar la máquina.

Código máquina. Primera generación de lenguajes (1GL)

No mucho más tarde apareció la idea de que las máquinas fueran capaces de realizar más de una aplicación. Para lo cual se ideó el hecho de que hubiera una memoria donde se almacenaban esas instrucciones. Esa memoria se podía rellenar con datos procedentes del exterior. Inicialmente se utilizaron tarjetas perforadas para introducir las instrucciones.
Durante mucho tiempo esa fue la forma de programar, que teniendo en cuenta que las máquinas ya entendían sólo código binario, consistía en introducir la programación de la máquina mediante unos y ceros. El llamado **código maquina.** Todavía los ordenadores es el único código que entienden, por lo que cualquier forma de programar debe de ser convertida a código máquina.

Sólo se ha utilizado por los programadores en los inicios de la informática. Su incomodidad de trabajo hace que sea impensable para ser utilizado hoy en día. Pero cualquier programa de ordenador debe, finalmente, ser convertido a este código para que un ordenador puede ejecutar las instrucciones de dicho programa.

Un detalle a tener en cuenta es que el código máquina es distinto para cada tipo de procesador. Lo que hace que los programas en código máquina no sean portables entre distintas máquinas.

Lenguaje ensamblador. Segunda generación de lenguajes (2GL)

En los años 40 se intentó concebir un lenguaje más simbólico que permitiera no tener que programar utilizando código máquina. Poco más tarde se ideó el **lenguaje ensamblador**, que es la traducción del código máquina a una forma más textual. Cada tipo de instrucción se asocia a una palabra mnemotécnica (como SUM para sumar por ejemplo), de forma que cada palabra tiene traducción directa en el código máquina.
Tras escribir el programa en código ensamblador, un programa (llamado también ensamblador) se encargará de traducir el código ensamblador a código máquina. Esta traducción es rápida puesto que cada línea en ensamblador tiene equivalente directo en código máquina (en los lenguajes modernos no ocurre esto).
La idea es la siguiente: si en el código máquina, el número binario 0000 significa sumar, y el número 0001 significa restar. Una instrucción máquina que sumara el número 8 (00001000 en binario) al número 16 (00010000 en binario) sería: 0000 00001000 00010000
Realmente no habría espacios en blanco, el ordenador entendería que los primeros cuatro BITS representan la instrucción y los 8 siguientes el primer número y los ocho siguientes el segundo número (suponiendo que los números ocupan 8 bits). Lógicamente trabajar de esta forma es muy complicado. Por eso se podría utilizar la siguiente traducción en ensamblador: SUM 8 16
Que ya se entiende mucho mejor.

Ejemplo1 (programa que saca el texto “Hola mundo” por pantalla):

```
DATOS SEGMENT
saludo db "Hola mundo!!!","$"
DATOS ENDS
CODE SEGMENT
assume cs:code,ds:datos
START PROC
mov ax,datos
mov ds,ax
mov dx,offset saludo
mov ah,9
int 21h
mov ax,4C00h
int 21h
START ENDP
```

```
CODE ENDS
END START
```

Puesto que el ensamblador es una representación textual pero exacta del código máquina; cada programa sólo funcionará para la máquina en la que fue concebido el programa; es decir, no es **portable**.
La ventaja de este lenguaje es que se puede controlar absolutamente el funcionamiento de la máquina, lo que permite crear programas muy eficientes.
Lo malo es precisamente que hay que conocer muy bien el funcionamiento de la computadora para crear programas con esta técnica. Además las líneas requeridas para realizar una tarea se disparan ya que las instrucciones de la máquina son excesivamente simples.

Lenguajes de alto nivel. Lenguajes de tercera generación (3GL)

Aunque el ensamblador significó una notable mejora sobre el código máquina, seguía siendo excesivamente críptico. De hecho para hacer un programa sencillo requiere miles y miles de líneas de código.
Para evitar los problemas del ensamblador apareció la tercera generación de lenguajes de programación, la de los lenguajes de alto nivel. En este caso el código vale para cualquier máquina pero deberá ser traducido mediante software especial que adaptará el código de alto nivel al código máquina correspondiente. Esta traducción es necesaria ya que el código en un lenguaje de alto nivel no se parece en absoluto al código máquina.
Tras varios intentos de representar lenguajes, en 1957 aparece el que se considera el primer lenguaje de alto nivel, el **FORTRAN** (***FORmula TRANslation)***, lenguaje orientado a resolver fórmulas matemáticos. Por ejemplo la forma en FORTRAN de escribir el texto ***Hola mundo*** por pantalla es:

```
PROGRAMHOLA
PRINT *, '¡Hola, mundo!'
END
```

1 Ejemplo tomado de la página **http://www.victorsanchez2.net**

Poco a poco fueron evolucionando los lenguajes formando lenguajes cada vez mejores (ver). Así en 1958 se crea **LISP** como lenguaje declarativo para expresiones matemáticas.

Programa que escribe ***Hola mundo*** en lenguaje LISP:

```
(format t "¡Hola, mundo!")
```

En 1960 la conferencia **CODASYL** se creo el **COBOL** como lenguaje de gestión en 1960. En 1963 se creo **PL/I** el primer lenguaje que admitía la multitarea y la programación modular. En COBOL el programa ***Hola mundo*** sería éste (como se ve es un lenguaje más declarativo):

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
MAIN SECTION.
DISPLAY "Hola mundo"
```

```
STOP RUN.
```

BASIC se creo en el año 1964 como lenguaje de programación sencillo de aprender en 1964 y ha sido, y es, uno de los lenguajes más populares. En 1968 se crea **LOGO** para enseñar a programar a los niños. **Pascal** se creo con la misma idea académica pero siendo ejemplo de lenguaje estructurado para programadores avanzados. El creador del Pascal (**Niklaus Wirdth**) creo **Modula** en 1977 siendo un lenguaje estructurado para la programación de sistemas (intentando sustituir al **C**).
Programa que escribe por pantalla ***Hola mundo*** en lenguaje Pascal):

```
PROGRAM HolaMundo;
BEGIN
Writeln('¡Hola, mundo!');
END.
```

Lenguajes de cuarta generación (4GL)

En los años 70 se empezó a utilizar éste término para hablar de lenguajes en los que apenas hay código y en su lugar aparecen indicaciones sobre qué es lo que el programa debe de obtener. Se consideraba que el lenguaje SQL (muy utilizado en las bases de datos) y sus derivados eran de cuarta generación. Los lenguajes de consulta de datos, creación de formularios, informes,... son lenguajes de cuarto nivel. Aparecieron con los sistemas de base de datos Actualmente se consideran lenguajes de éste tipo a aquellos lenguajes que se programan sin escribir casi código (lenguajes visuales), mientras que también se propone que éste nombre se reserve a los lenguajes orientados a objetos.

Lenguajes orientados a objetos

En los 80 llegan los lenguajes preparados para la programación orientada a objetos todos procedentes de **Simula** (1964) considerado el primer lenguaje con facilidades de uso de objetos. De estos destacó inmediatamente **C++**.
A partir de **C++** aparecieron numerosos lenguajes que convirtieron los lenguajes clásicos en lenguajes orientados a objetos (y además con mejoras en el entorno de programación, son los llamados lenguajes **visuales**): **Visual Basic**, **Delphi** (versión orientada a objetos de Pascal), **Visual C++,...**
En 1995 aparece Java como lenguaje totalmente orientado a objetos y en el año 2000 aparece C# un lenguaje que toma la forma de trabajar de **C++** y del propio Java.
El programa ***Hola mundo*** en C# sería:

```
using System;
class MainClass
{
public static void Main()
{
Console.WriteLine("¡Hola, mundo!");
}
}
```

Lenguajes para la Web

La popularidad de Internet ha producido lenguajes híbridos que se mezclan con el código **HTML** con el que se crean las páginas web. HTML no es un lenguaje en sí sino un formato de texto pensado para crear páginas web. Éstos lenguajes se usan para poder realizar páginas web más potentes.
Son lenguajes interpretados como **JavaScript** o **VB Script**, o lenguajes especiales para uso en servidores como **ASP**, **JSP** o **PHP**. Todos ellos permiten crear páginas web usando código mezcla de página web y lenguajes de programación sencillos.
Ejemplo, página web escrita en lenguaje HTML y JavaScript que escribe en pantalla "Hola mundo" (de color rojo aparece el código en JavaScript):

```
<html>
<head>
<title>Hola Mundo</title>
</head>
<body>
<script type="text/javascript">
document.write("¡Hola mundo!");
</script>
</body>
</html>
```

4. *Tipos de programación.*

Ya sabemos que son los programas, los leguajes, su evolución… pero a la hora de programar hay que distinguir distintos tipos de programación, estos tipos no siempre están condicionado al tipo de lenguaje,**(Ver Anexo II al final del tema)** en muchas ocasiones dependen directamente más del programador y de su metodología de trabajo que del propio lenguaje, vamos a aclarar este concepto.
La programación consiste en pasar algoritmos a algún lenguaje de ordenador a fin de que pueda ser entendido por el ordenador. La programación de ordenadores comienza en los años 50 y su evolución a pasado por diversos pasos.
La programación se puede realizar empleando diversas **técnicas** o métodos. Esas técnicas definen los distintos tipos de programaciones.

Programación desordenada

Se llama así a la programación que se realizaba en los albores de la informática (aunque desgraciadamente en la actualidad muchos programadores siguen empleándola). En este estilo de programación, predomina el ***instinto*** del programador por encima del uso de cualquier método lo que provoca que la corrección y entendimiento de este tipo de programas sea casi ininteligible.
Ejemplo de uso de esta programación (listado en Basic clásico):

```
10 X=RANDOM()*100+1;
20 PRINT "escribe el número que crees que guardo"
30 INPUT N
40 IF N>X THEN PRINT "mi numero es menor" GOTO 20
50 IF N<X THEN PRINT "mi numero es mayor" GOTO 20
```

60 PRINT "¡Acertaste!"
El código anterior crea un pequeño juego que permite intentar adivinar un número del 1 al 100.

Programación estructurada

En esta programación se utiliza una técnica que genera programas que sólo permiten utilizar tres estructuras de control:
Secuencias (instrucciones que se generan secuencialmente)
Alternativas (sentencias ***if***)
Iterativas (bucles condicionales)

El listado anterior en un lenguaje estructurado sería (listado en Pascal):

```
PROGRAM ADIVINANUM;
USES CRT;
VAR x,n:INTEGER;
BEGIN
X=RANDOM()*100+1;
REPEAT
WRITE("Escribe el número que crees que guardo");
READ(n);
IF (n>x) THEN WRITE("Mi número es menor");
IF (n>x) THEN WRITE("Mi número es mayor");
UNTIL n=x;
WRITE("Acertaste");
```

La ventaja de esta programación está en que es más legible (aunque en este caso el código es casi más sencillo en su versión desordenada). **Todo programador debería escribir código de forma estructurada.**

Programación modular

Completa la programación anterior permitiendo la definición de módulos independientes cada uno de los cuales se encargará de una tarea del programa.
De esta forma el programador se concentra en la codificación de cada módulo haciendo más sencilla esta tarea. Al final se deben integrar los módulos para dar lugar a la aplicación final.
El código de los módulos puede ser invocado en cualquier parte del código. Realmente cada módulo se comporta como un subprograma que, partir de unas determinadas entradas obtienen unas salidas concretas. Su funcionamiento no depende del resto del programa por lo que es más fácil encontrar los errores y realizar el mantenimiento.

Programación orientada a objetos

Es la más novedosa, se basa en intentar que el código de los programas se parezca lo más posible a la forma de pensar de las personas. Las aplicaciones se representan en esta programación como una serie de objetos independientes que se comunican entre sí.
Cada objeto posee datos y métodos propios, por lo que los programadores se concentran en programar independientemente cada objeto y luego generar el código que inicia la comunicación entre ellos.

Es la programación que ha revolucionado las técnicas últimas de programación ya que han resultado un importante éxito gracias a la facilidad que poseen de encontrar fallos, de reutilizar el código y de documentar fácilmente el código.

Ejemplo (código Java):

```
/**
*Calcula los primos del 1 al 1000
*/
public class primos {
/** Función principal */
public static void main(String args[]){
int nPrimos=10000;
boolean primo[]=new boolean[nPrimos+1];
short i;
for (i=1;i<=nPrimos;i++) primo[i]=true;
for (i=2;i<=nPrimos;i++){
if (primo[i]){
for (int j=2*i;j<=nPrimos;j+=i){
primo[j]=false; }
}
}
for (i=1;i<=nPrimos;i++) {
System.out.print(" "+i);
}
}
}
```

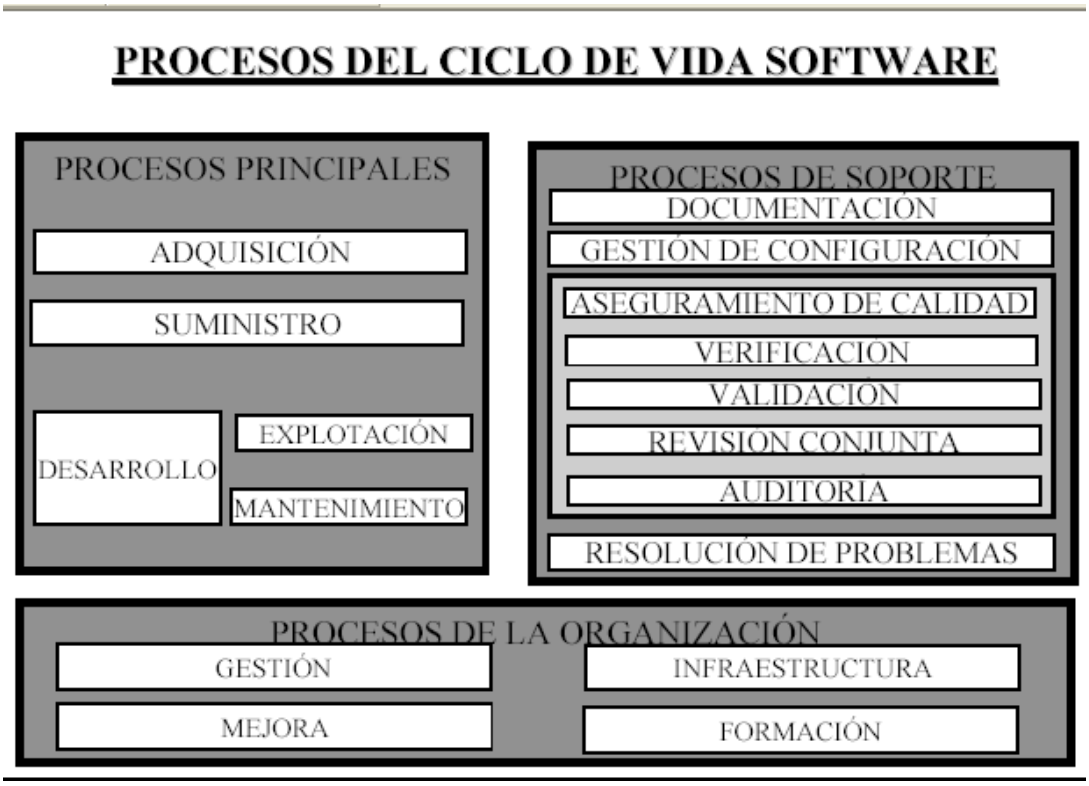
5. Fases del desarrollo de una aplicación: análisis, diseño, codificación, pruebas, documentación, explotación y mantenimiento, entre otras.

En esta última pregunta vamos a centrarnos en tres apartados:

1. El proceso del ciclo de vida (un programa = un producto más del mercado).
2. Fases del ciclo de vida en general (común a casi todas las metodologías). Distintos modelos.
3. Distintas metodologías de desarrollo. Aparecerá nuestra metodología Métrica (aplicada por la administración pública española)

5.1. El proceso del ciclo de vida del software.

Los profesionales del sector informático y los organismos internacionales proponen una serie de procesos para el mejor desarrollo del software. El IEEE (Instituto de Ingenieros, Electrónicos y Eléctricos) junto con la ISO (Organización de Estandarización Internacional) en su norma 12207-1, dejando al margen y sin fomentar ninguna metodología en particular propone 3 procesos dentro del ciclo de vida:



1. **PROCESOS PRINCIPALES.** Afectan a todas las personas; clientes-usuarios, analistas, programadores, compradores, vendedores… Comprende:

   1.1. **Adquisición:** Desde la preparación de la oferta-catálogo del producto, la selección del suministrador, condiciones de suministro, de pago, hasta entrega del producto.
   1.2. **Suministro:** Comprende las actividades del vendedor o suministrador, va desde la propuesta al cliente hasta la firma del contrato, debe llevar un proyecto de plazos de entrega para garantizar la ejecución del proyecto.
   1.3. **Desarrollo: En** este proceso se hace referencia las fases propias del desarrollo del software, varia según la metodología utilizada.
   1.4. **Explotación:** Incluye la explotación por parte del usuario y el soporte operativo que se le proporcione.
   1.5. **Mantenimiento:** El objetivo es mantener el software actualizado, útil. Los cambios se producen por un mal funcionamiento, una nueva necesidad o un cambio en un proceso (ley cambia o convenio).

2. **PROCESOS DE SOPORTE.** Sirven de apoyo a los demás procesos. Comprende:

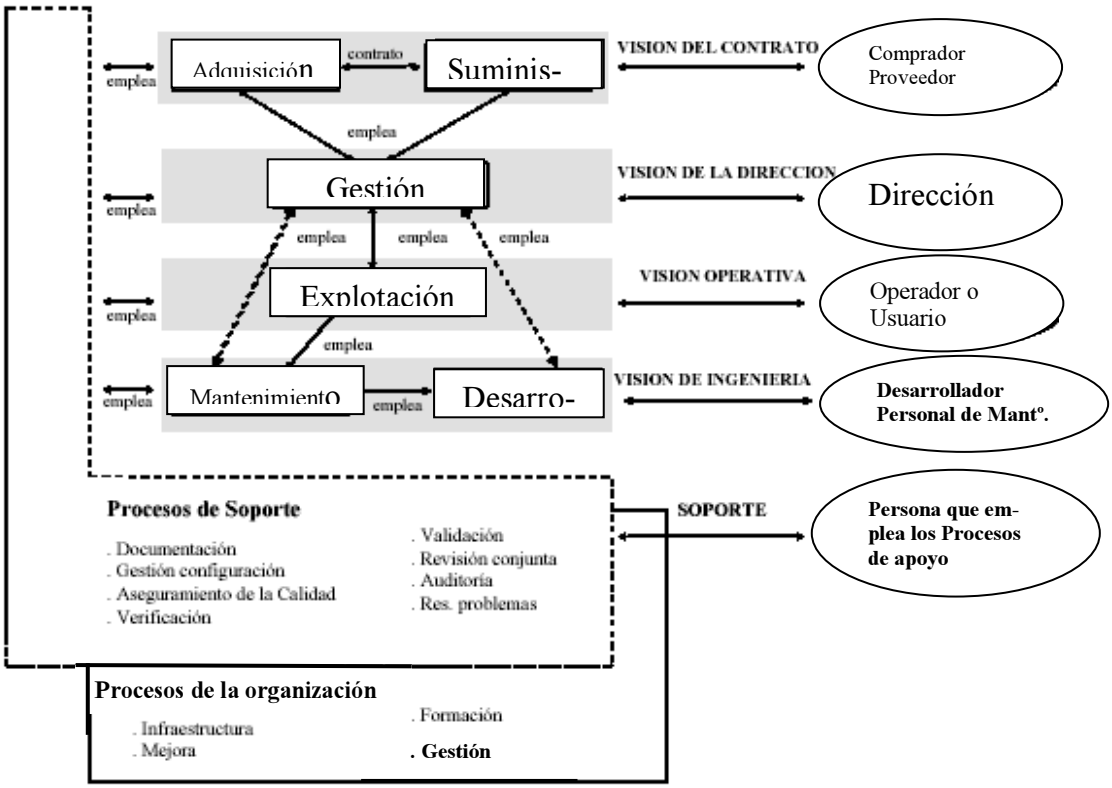
   2.1. **Documentación.** Todo está escrito para poder acudir al papel.
   2.2. **Gestión de configuración.** Establece procedimientos administrativos y técnicos en todo el ciclo.
   2.3. **Auditoria.** Puede ser interna o externa.
   2.4. **Calidad.** Debemos ajustarnos a los planes.
   2.5. **Verificación.** Todo completo y correcto.
   2.6. **Validación.** Comparar resultados y objetivos iniciales.
   2.7. **Revisión conjunta.** Evaluación del ciclo completo.
   2.8. **Resolución de problemas.** Eliminar problemas durante el ciclo.

3. PROCESOS DE LA ORGANIZACIÓN: Son propios de la empresa:

3.1. Gestión. Planificar, controlar, revisar y evaluar.
3.2. Mejoras. A todos los niveles.
3.3. Infraestructuras. Según necesidades actuales y futuras.
3.4. Formación. Plan de formación del personal.

CICLO DE VIDA DEL SOFTWARE

110



Ejercicios:

1. Cuando se paga una aplicación. (cobrar-pagar).
2. Ayudas a la explotación de una aplicación.
3. ¿Qué es una auditoria?
4. Inventa una norma de calidad para la fabricación de sillas de ordenador.
5. Como detectar necesidades de software en una empresa.
6. Que se puede hacer si no se utiliza una parte del software.
7. Analogías entre el ciclo de vida y hacer un traje o construir un coche.
8. Relaciona ISO 12207-1 y las personas que intervienen (P. Principales).

5.2. Fases del ciclo de vida del software.

Algunas ideas básicas:

Software: (1) **instrucciones** de ordenador que cuando se ejecutan cumplen una función y tienen un comportamiento deseado, (2) **estructuras de datos** que facilitan a los programadores la adecuada manipulación de la información, y (3) **documentos** que describen la operación y el uso de los programas.

Características del software:

- El software se desarrolla, no se fabrica en sentido estricto.
- El software no se estropea.
- La mayoría del software se construye a medida.

Ingeniería del software: Establecimiento y uso de principios de ingeniería robustos, orientados a garantizar la obtención de software económico, fiable y eficiente sobre máquinas reales.

En general cada metodología aplicada al desarrollo del software tiene sus fases y sus técnicas, actividades, tareas o como se deseen llamar, sin embargo en general coinciden en los siguientes apartados:

1. Identificación de necesidades (Situación inicial, estudio previo…=Enunciado).
2. Análisis de Requisitos (Qué hay y qué se quiere, soluciones, alternativas, prioridades… Arquitectura Lógica de datos, eventos y procedimientos = Diccionario de Datos).
3. Diseño (Arquitectura física. Viene condicionada por el Análisis.).
4. Construcción, Codificación o Programación.
5. Pruebas.
6. Implantación o Implementación.
7. Explotación
8. Mantenimiento.

Visión genérica de la Ingeniería del Software.

- **Definición.** **¿Qué?**
  - Análisis del sistema.
    - *Establecer el ámbito del software.*
  - Análisis de requisitos del sistema software.
    - *Definición detallada de la función del software.*
  - Planificación.
    - *Análisis de riesgos.*
    - *Asignación de recursos.*
    - *Definición de tareas.*
    - *Estimación de costes.*
- Desarrollo. ¿Cómo?
  - **Diseño.**
    - ***Arquitectura de la aplicación.***
    - ***Estructura de los datos.***
    - ***Estructura interna de los programas.***
    - ***Diseño de las interfaces.***
  - **Codificación.**
  - **Pruebas.**
- Mantenimiento. El cambio.
  - **Corrección de errores.**
  - **Cambios en el entorno.**
  - **Cambios en los requisitos.**

Ejercicio:

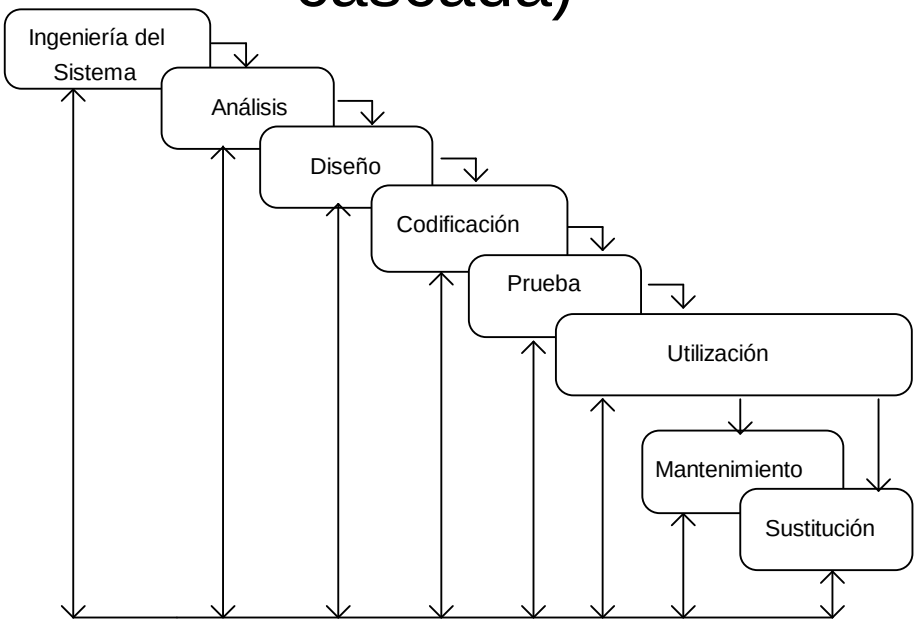
1. Explica las fases de ciclo de vida de un pantalón a medida.
2. Explica la utilización de la palabra código en la fase de ciclo de vida.
3. Arquitectura lógica de datos.¿Qué será?.

A partir de aquí se puede hablar de distintos modelos de ciclos de vida, veamos algunos:

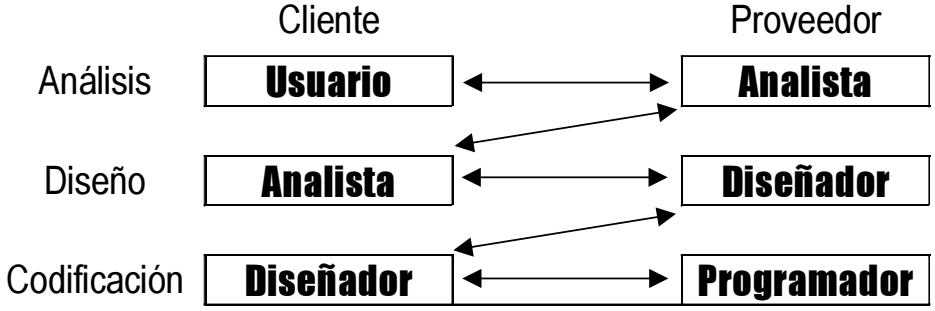
1. Modelo Lineal, Secuencial o en Cascada. Las Fases se generan una tras otra de forma que esta que no termine una no comenzará la siguiente.
   Ventajas:
   a. Es el modelo más antiguo y continua hoy vigente, por algo será.
   b. Permite tener muy claro cual es el punto de partida, el final de la fase anterior.
   c. Se pueden controlar o revisar los plazos u objetivos a final de cada fase.
   Inconvenientes:
   a. En ocasiones el cliente puede alterar los requisitos iniciales, esto puede suceder y de hecho sucede en cualquier momento.
   b. Un cambio en una fase genera un efecto cascada en las siguientes.

Diapositiva 1 y 2.

Ciclos de vida: clásico (en cascada)



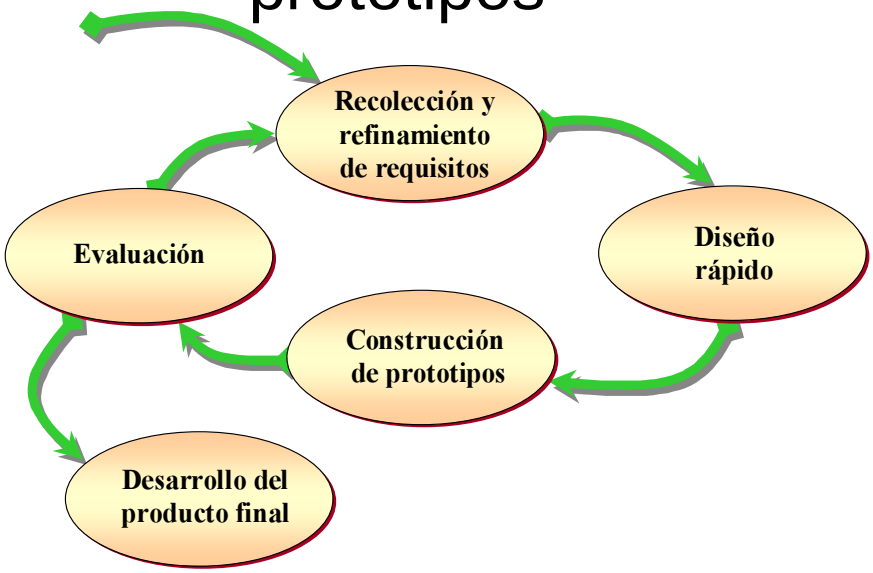
Ciclos de vida: contractual



2. Modelo de construcción de prototipos. Se utilizan cuando el entorno presenta cierto grado de desconocimiento o incertidumbre, exige una estrecha colaboración entre analista y usuario, ambos definen los requisitos y se produce un diseño rápido que permite construir el prototipo. Difícilmente será el definitivo, sobre él se continúa trabajando hasta llegar a algo real.
   Ventajas:
   a. El cliente colabora estrechamente. Si algo va mal se comparten responsabilidades.
   b. Consume poco tiempo, al cliente esto le gusta y también al desarrollador de aplicaciones.
   Inconvenientes:
   a. No genera documentación alguna.
   b. Baja calidad y dificultad de mantenimiento.

Diapositiva 3.

Ciclos de vida: construcción de prototipos



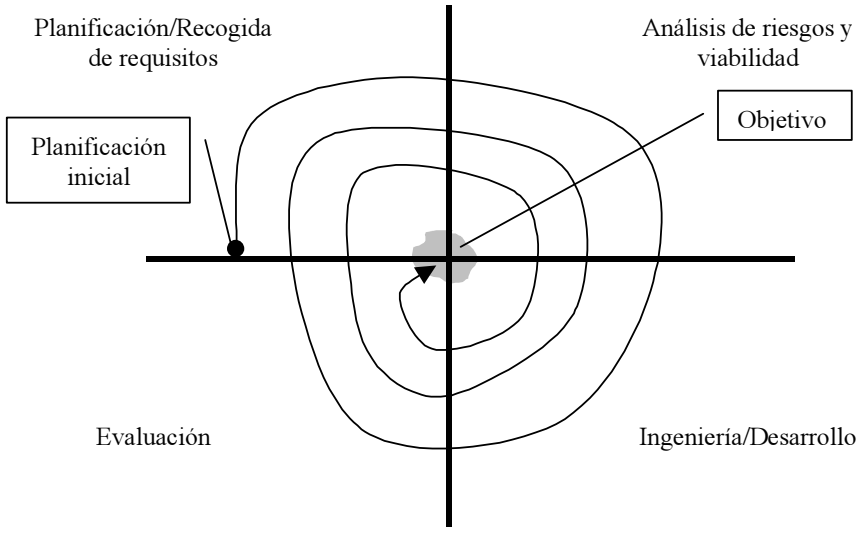
3. Modelo DRA (Desarrollo Rápido de Aplicaciones). Es un resumen del Modelo Lineal, todo queda reducido a:
   a. Modelado de gestión.
   b. Modelado de datos.
   c. Modelado de Procesos.
   d. Generación de aplicaciones: Lenguajes de 4G.
   Ventaja: Simple de aplicar, siempre que se tenga claro el entorno, las herramientas de diseño (CASE) y el lenguaje a utilizar.
   Inconveniente: Consume mucho personal que, además, debe estar perfectamente coordinado y requiere que el programa esté perfectamente modulado. Requiere una alta estabilidad técnica.

4. Modelo en espiral. Combina el modelo secuencial con el prototipo. Cada vuelta en el sentido de las agujas del reloj genera un prototipo, para llegar a él se sigue de

forma secuencial una serie de fases (Comunicación con cliente, Análisis, Diseño, Construcción, Evaluación del cliente y vuelta a empezar hasta satisfacer al cliente).
Ventajas:
a. Control del riesgo. Se evalúa cada prototipo antes de empezar con el siguiente.
b. Enfoque realista. Todo cambio origina una vuelta más = un nuevo prototipo.
Inconveniente:
a. El cliente no puede estar siempre con el mismo tema, pendiente del modelo.
b. Un error no descubierto a tiempo originaría problemas incontrolables.

Diapositiva 4.

Ciclos de vida: en espiral



Ejercicios:
1. Comentar una actividad donde se pueda aplicar el modelo D.R.A..
2. Situación ideal para aplicar un prototipo.

5.3. Distintas metodologías de desarrollo

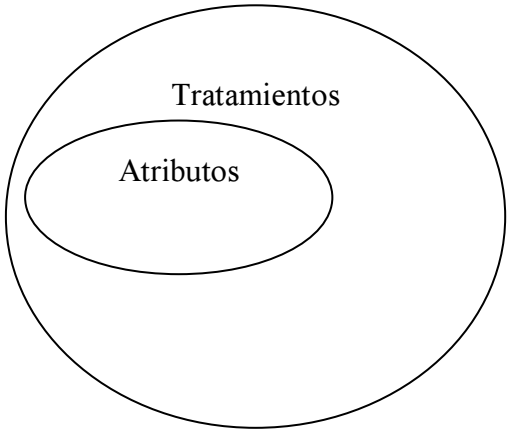
Existe un desarrollo paralelo entre la programación y las metodologías de desarrollo, por este motivo podemos hablar de tres tipos: convencional o clásica, estructurada y orientada a objetos (tendencias actuales):

1. Análisis Clásico; dividía su trabajo en Previo, Funcional y Orgánico. En muchos casos primero se compraba el ordenador y luego se planteaba el programa. Se estudiaba el sistema actual (Previo), se planteaban posibilidades, se establecían las soluciones (funcional) y se indicaban los medios físicos (orgánico). Se planteaba todo

pensando en una duración del programa que no coincide con los conceptos de hoy día, nada se podría aprovechar en el futuro.

2. Análisis Estructurado. Surge a raíz de la programación estructurada (Dijkstra 1968).
   - Se basa en el llamado análisis descendente o top-down.
   - Se establecen nuevas técnicas de modelado de datos, estudio de eventos, etc.
   - La idea es utilizar: Gráficos como referencia para las especificaciones, particiones para poder tratar de forma independiente cualquier parte y mínimamente redundante (evitar repetir algo que ya está hecho. Todo ello bajo en concepto de reingeniería.**Herramientas CASE.**
   - Algunos autores han llamado a todos estos conceptos el intento de informatizar a los programadores.

3. Análisis Orientado a Objeto. Surge con los lenguajes Smalltalk, C++, Objetive C.. Empiezan a surgir Metodologías O.O. como O.M.T. o U.M.L.. Incluso la siguiente versión de Métrica Ver. 3. Se prevé que la incorpore. Utiliza objetos que poseen internamente datos y procesos. Se crea un modelo objetos que incluye tratamiento.



Objeto

Existe un concepto superior llamado Clase, se supone que los objetos son instanciaciones u ocurrencias de ella.

1968……..Conceptos de P.Estructurada. Dijsktra.
1975……..Primeros conceptos de diseño estructurado. Yourdon.(EEUU).
1978……..Nace la Metodología francesa de Merise (Francia).
1981……..Versión inicial de SSADM (Inglaterra).Metod.Dise.Anali.Sistem.Estru.
1986……..Ver. 3 de SSADM.
1989……..Métrica. (España)
Hoy en día…Métrica (Ver. 3 visitar www.map.es).

Principales metodologías que han influido en Métrica:

YOURDON: (se dice que es estructurada orientada a procesos. De Constantine)

1. Realizar el D.F.D. del sistema (lo toma Métrica de aquí).

2. Diagramas de estructuras de datos (DED) a partir de DFD.
3. Diseño midiendo acoplamiento y cohesión.
4. División en unidades llamadas cuadernos de carga (se toma del clásico).

MERISE: (es estructurada orientada a los datos. Es utilizada por la Administración en Francia).

1. Estudio Preliminar (modelado de datos lo toma Métrica).
2. Estudio detallado (coincide con el diseño en otras metodologías).
3. Implementación.
4. Realización y puesta en marcha.

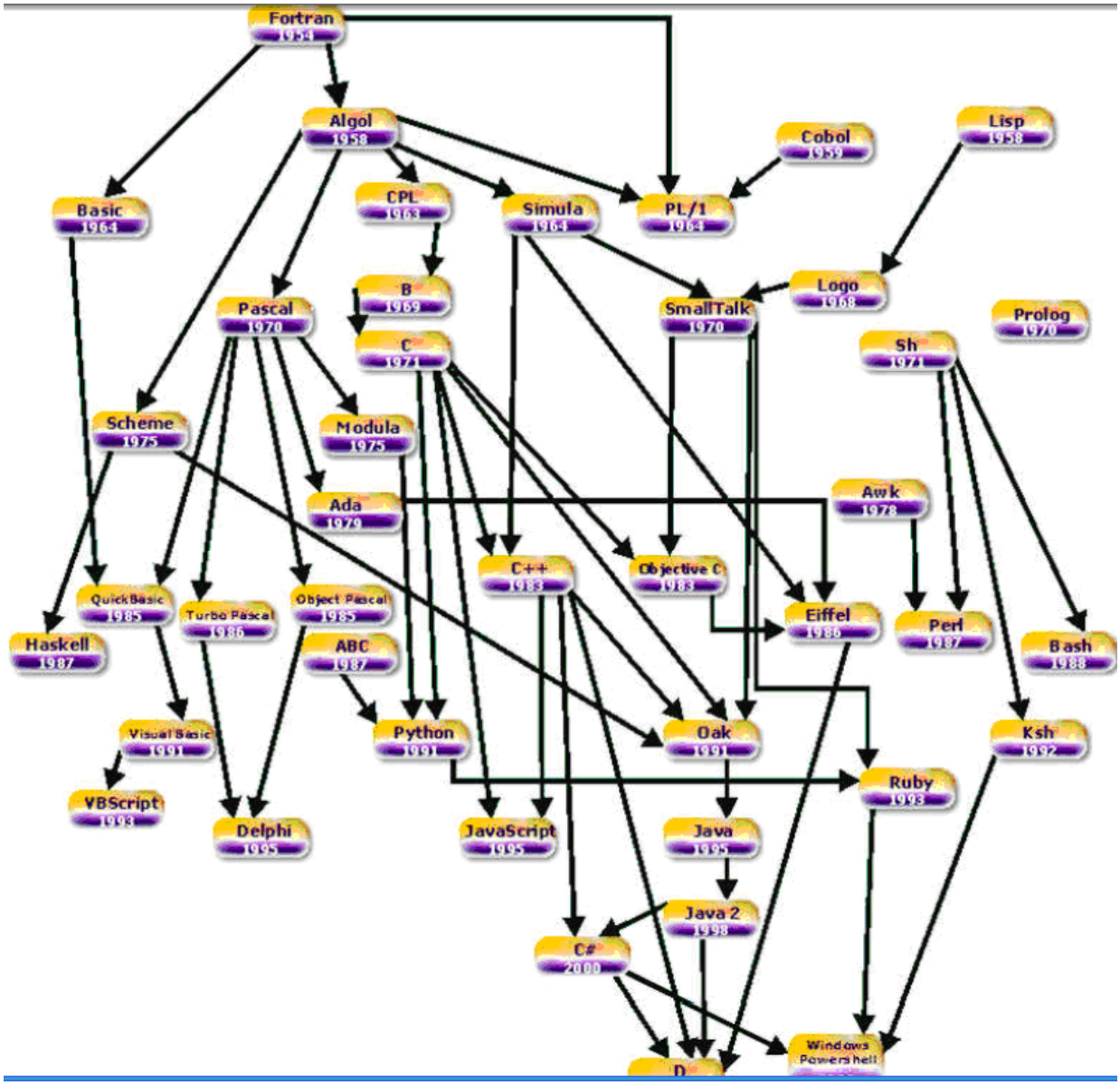
SSADM: (utilizada por la administración inglesa, de ella toma la mayor parte Métrica). No hablamos de sus fases ya que coinciden en parte con Métrica, hablaremos de sus características:

1. Especial participación al usuario.
2. Qué hacer, cuando y como.
3. Tres puntos de vista: datos, eventos y procesos.
4. Máxima flexibilidad en las técnicas y herramientas.

Ejercicios:

1. Comenta los procesos que intervienen en una actividad; ir de compras, matricularse en el Instituto, jugar un partido.
2. Idem los datos.
3. Inventa unos símbolos para enseñar a bailar.
4. Sustituye los símbolos por palabras.
5. Actividades donde se puedan dar procesos; lineales, selectivos y repetitivos.
6. Diferencia entre técnica y herramienta.
7. Diferencia entre las metodologías estructuradas y la O.O..

ANEXO I. HISTORIA DE LOS LENGUAJES DE PROGRAMACIÓN:



ANEXO II: Lenguajes y formas de programar (metodologías).