UT 1: Desarrollo del software y entornos de desarrollo

BIBLIOGRAFÍA:

UNIDAD 1: (TEMA DISPONIBLES EN MOODLE)

•Cabrera, G,Montoya, G : Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión. Mc Graw-Hill.
•Piattini, Calvo, Cervera, Fernández. Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión. Ra-Ma
•Pressman R.S. Ingeniería del Software. Un Enfoque Práctico. McGraw-Hill
•Sommerville,I. Ingeniería de Software.(6ª Edición),Addison Wesley.
•Larman C. UML y Patrones. (2ª Edición), Pearson Prentice Hall.
•Carlos Casado Iglesias: Entornos de Desarrollo. RA-MA
•Juan Carlos Moreno Pérez: Entornos de Desarrollo. Editorial SINTESIS
•RAMOS MARTÍN, Alicia (2014). Entornos de desarrollo. Madrid: Editorial Garceta.
•Alicia Ramos Martín, Mª Jesús Ramos Martín: Entornos de Desarrollo. Garceta grupo editorial
•Juan Carlos Moreno Pérez. Entornos de Desarrollo. Editorial SINTESIS
•Carlos Casado Iglesias. Entornos de Desarrollo. Ra-Ma

Profesora: Elena Fernández Chirino

DEPARTAMENTO DE INFORMÁTICA

http://www.iestrassierra.com

RESULTADOS DE APRENDIZAJE

Al finalizar esta unidad valoraremos que el alumno:

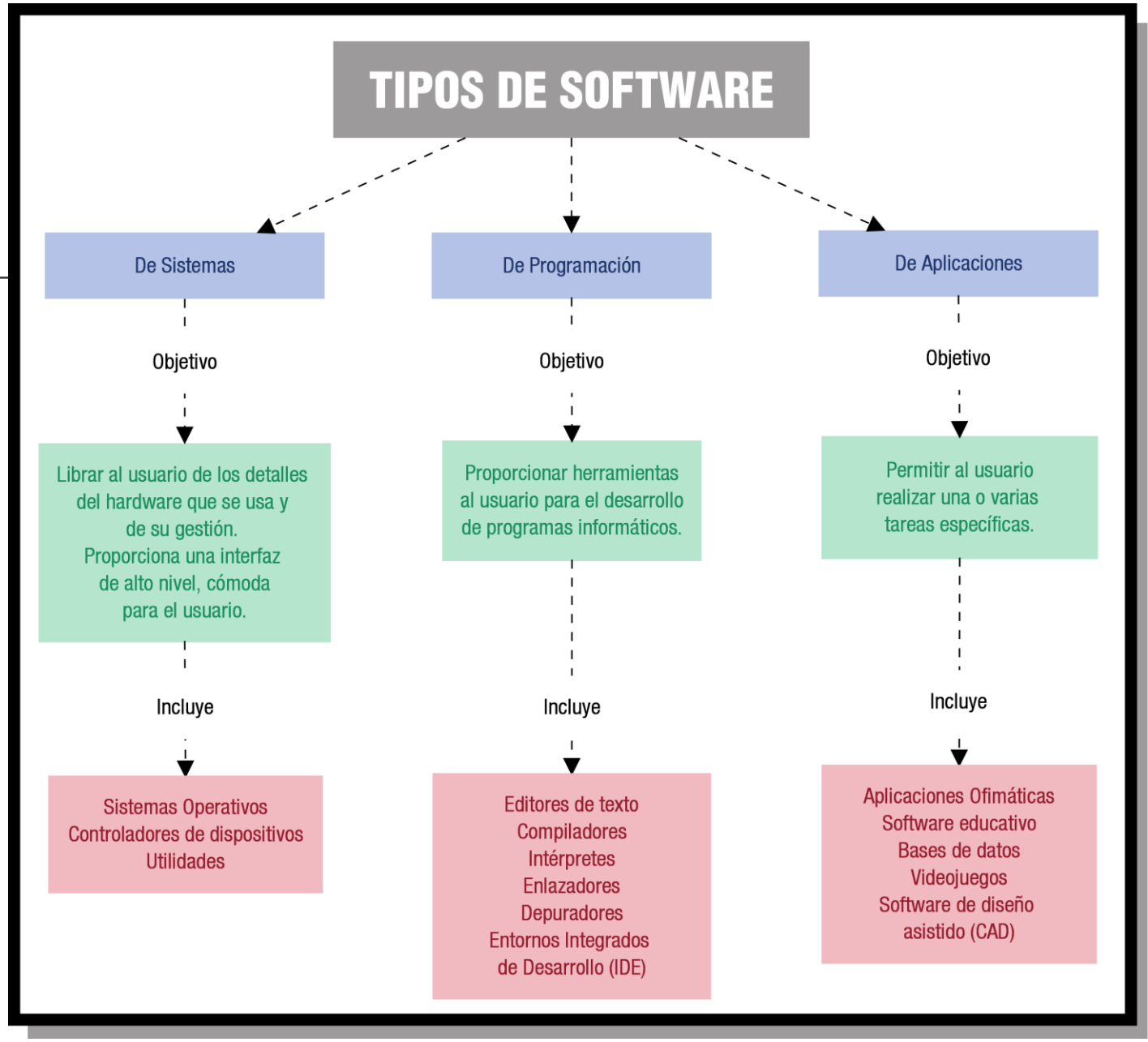
- R.A.1: Reconocer los elementos y herramientas que intervienen en el desarrollo de un programa informático, analizando sus características y las fases en las que actúan hasta llegar a su puesta en funcionamiento.

1.- SOFTWARE Y PROGRAMA. TIPOS DE SOFTWARES

El software es el conjunto de programas informáticos que actúan sobre el hardware para ejecutar lo que el usuario desee.

Según su función se distinguen tres tipos de software:

- sistema operativo,
- software de programación
- y aplicaciones.

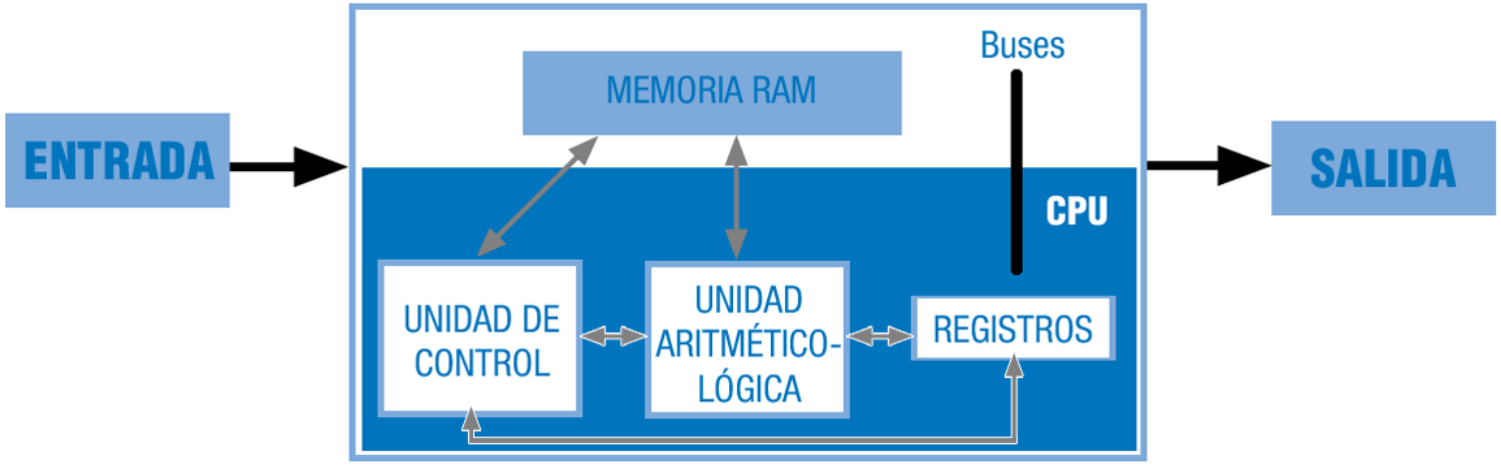


2.- RELACIÓN HARDWARE-SOFTWARE

Existe una relación indisoluble entre éste y el software, ya que necesitan estar instalados y configurados correctamente para que el equipo funcione. El software se ejecutará sobre los dispositivos físicos. Esta relación software-hardware la podemos poner de manifiesto **desde dos puntos de vista**:

Desde el punto de vista del sistema operativo: coordina el hardware durante el funcionamiento del ordenador, actuando como intermediario entre éste y las aplicaciones.

Desde el punto de vista de las aplicaciones: conjunto de programas escritos en algún lenguaje de programación que el hardware del equipo debe interpretar y ejecutar.



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

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

3.- DESARROLLO DE SOFTWARE. 3.1. Ciclos de vida

Desarrollo de Software

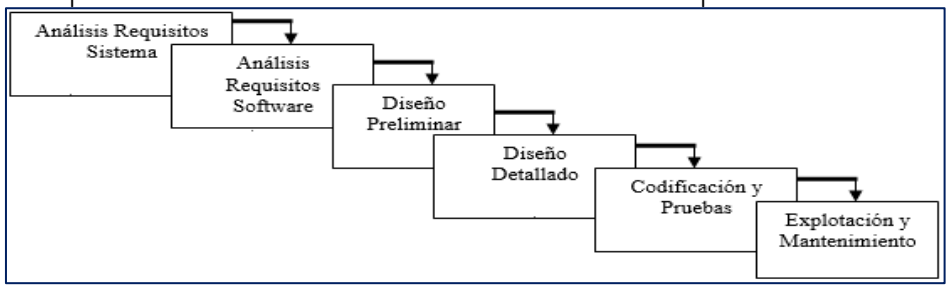
Conjunto de procesos desde que nace una idea hasta que se convierte en software.

Conlleva una serie de **Etapas** en el desarrollo de software:

Según el orden y la forma en que se lleven a cabo las etapas hablaremos de diferentes **ciclos de vida del software**.

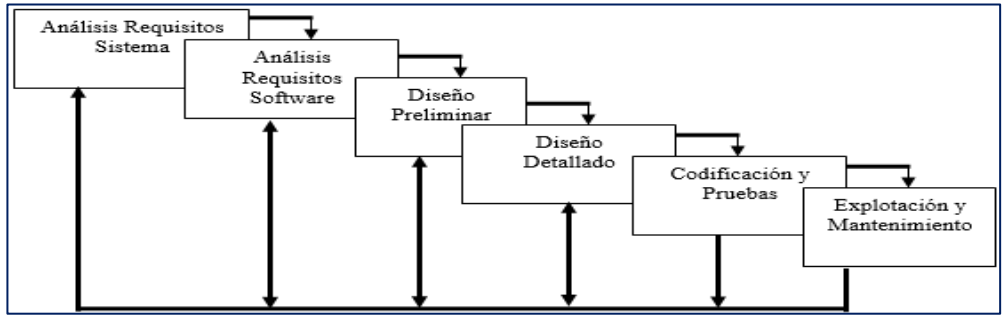
Modelo en Cascada:

Sólo es aplicable a pequeños desarrollos, ya que las etapas pasan de una a otra sin retorno posible.



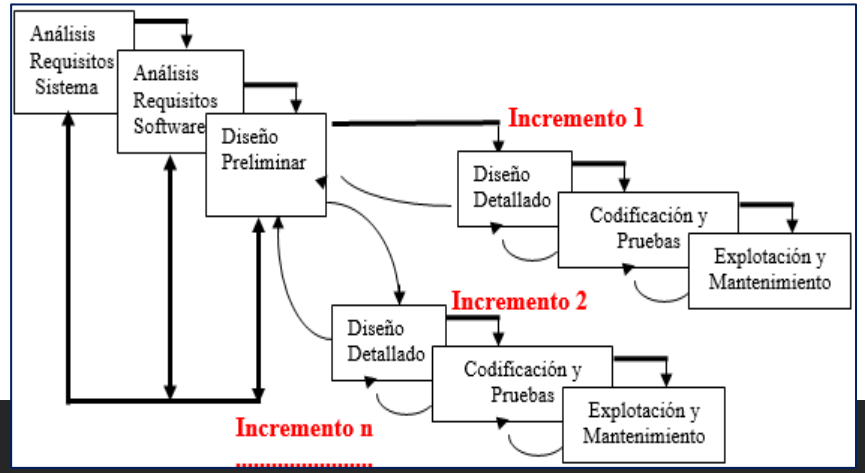
Modelo en Cascada con Realimentación:

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

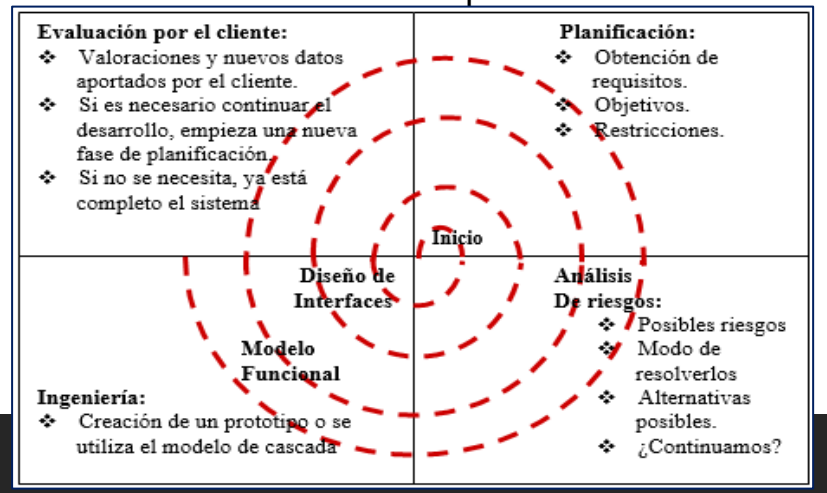


Modelos Evolutivos: Son más modernos que los anteriores. Tienen en cuenta la naturaleza cambiante y evolutiva del software. Distinguimos dos variantes:

Modelo Iterativo Incremental



Modelo es espiral



3.2.- Herramientas de apoyo al desarrollo del software.

Las **herramientas CASE** son un conjunto de aplicaciones que se utilizan en el desarrollo de software con el objetivo de reducir costes y tiempo del proceso, mejorando por tanto la productividad del proceso.

CLASIFICACIÓN en función de las fases del ciclo de vida del software en la que ofrecen ayuda:

- U-CASE: ofrece ayuda en las fases de planificación y análisis de requisitos.
- M-CASE: ofrece ayuda en análisis y diseño.
- L-CASE: ofrece 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.

El **desarrollo rápido de aplicaciones o RAD** es un proceso de desarrollo de software que comprende el desarrollo iterativo, la construcción de prototipos y el uso de utilidades

4.- LENGUAJES DE PROGRAMACIÓN

Son los que nos permiten comunicarnos con el hardware del ordenador.

Los lenguajes de bajo nivel son dependientes del hardware

EVOLUCIÓN →

- ***Lenguaje máquina***
  - Sus instrucciones son combinaciones de unos y ceros
  - Es el único lenguaje que entiende directamente el ordenador
  - Fue el primer lenguaje utilizado
  - Es único para cada procesador (no es portable de un equipo a otro)
  - Hoy día nadie programa en este lenguaje
- ***Lenguaje ensamblador***
  - Sustituyó al lenguaje máquina para facilitar la labor de programación
  - Se programa usando mnemotécnicos (instrucciones complejas)
  - Necesita traducción al lenguaje máquina para poder ejecutarse
  - Sus instrucciones que hacen referencia a la ubicación física de los archivos
  - Es difícil de utilizar

Lenguajes de bajo nivel

- ***Lenguaje de alto nivel***
  - Sustituyeron al lenguaje ensamblador
  - Utilizan sentencias y órdenes derivadas del idioma inglés.
    - *Estruturados*
    - *Orientado a objetc*
  - Necesita traducción al lenguaje máquina
  - Son más cercanos al razonamiento humano
  - Son utilizados hoy día

Lenguajes de alto nivel

- ***Lenguajes visuales***
  - Están sustituyendo a los lenguajes de alto nivel basados en código
  - Se programa gráficamente usando el ratón
  - Su correspondiente código se genera automáticamente
  - Necesitan traducción al lenguaje máquina
  - Son completamente portables de un equipo a otro

Los lenguajes de alto nivel son independientes de la arquitectura del ordenador

4.1.- Concepto y características.

La elección del lenguaje de programación para codificar un programa dependerá de las características del problema a resolver.

4.2.- Lenguajes estructurados.

Permite sólo el uso de tres tipos de sentencias:

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

VENTAJAS

Programas fáciles de leer, sencillos y rápidos.
El mantenimiento es sencillo.
La estructura es sencilla y clara.

INCONVENIENTES

Todo el programa se concentra en **un único bloque.**
No permite reutilización eficaz de código,
Se sustituyó la programación modular, donde los programas se codifican por módulos y bloques, permitiendo mayor funcionalidad.

4.3.- Lenguajes orientados 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.

VENTAJAS

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

INCONVENIENTES

No es una programación tan intuitiva como la estructurada.

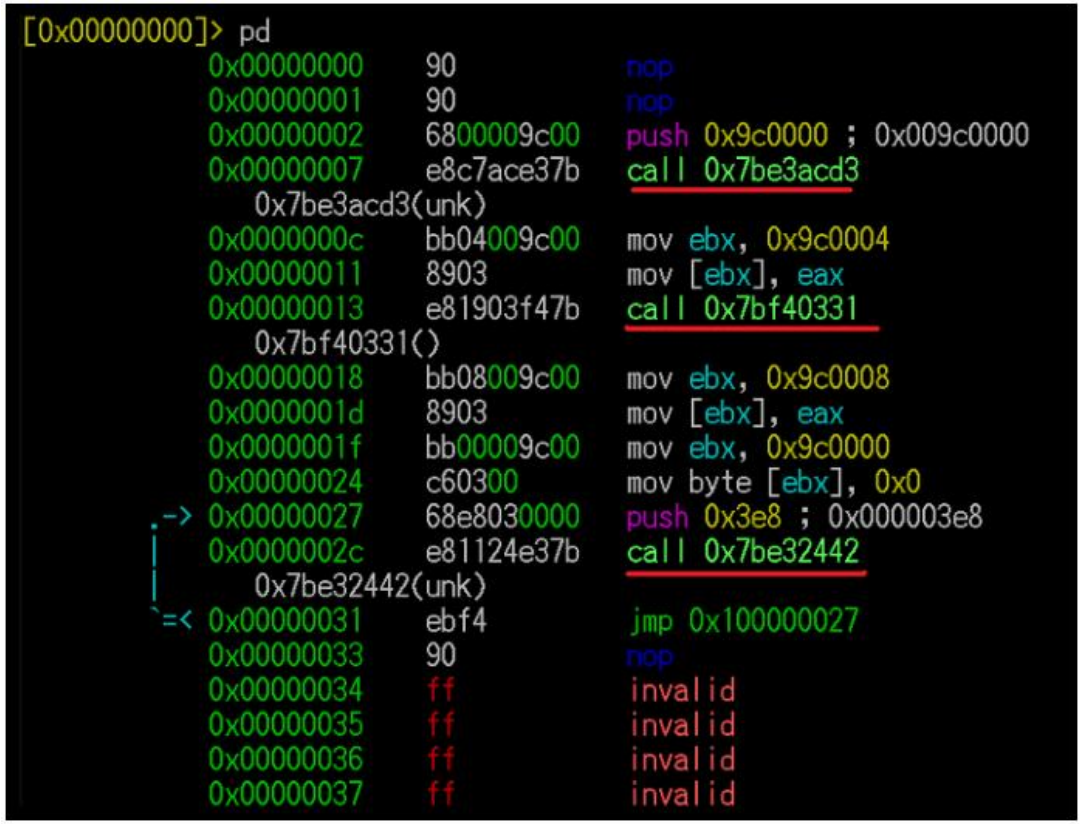
4.- LENGUAJES DE PROGRAMACIÓN

Lenguaje máquina

Lenguaje de máquina - tabla binaria.

```
11001010 00010111 11110101 00101011
00010111 11110101 00101011 00101011
11001010 00010111 11110101 00101011
00010111 11110101 00101011 00101011
11001010 11110101 00101011 00101011
11001010 11001010 11110101 00101011
11001010 11110101 00101011 00101011
11001010 00010111 11110101 00101011
00010111 11110101 00101011 00101011
11001010 11110101 00101011 00101011
```

Lenguaje ensamblador



4.- LENGUAJES DE PROGRAMACIÓN

4.2.- Lenguajes estructurados. (C)

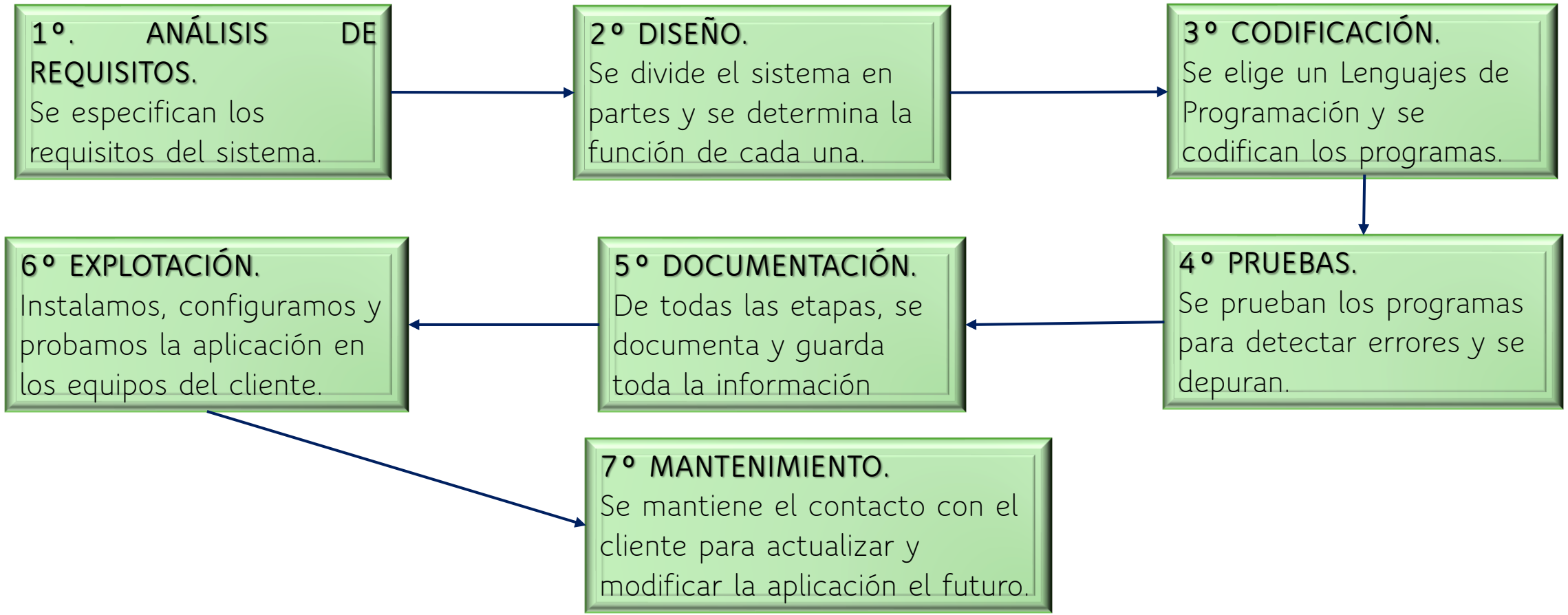
```c
/* Suma de n números */

#include <stdio.h>
int main() {
    int num=0,suma=0;

    do {
        suma=suma+num;
        printf("un número: ");
        scanf("%d",&num);
    } while(num>=0);
    printf("suma es: %d",suma);
    return 0;
}
```

4.3.- Lenguajes orientados a objetos.

```
84  METHOD nuevo() CLASS ConceptoCtr
85  // ALTA
86
87     LOCAL concepto
88
89     WITH OBJECT ::formulario := ConceptoFrm():new()
90        :cText   := "Nuevo concepto"
91        :cargo   := "A"
92        // asignamos este controlador al formulario
93        :setControlador( self )
94
95        IF :ShowModal() == mrOk
96           // creamos un nuevo objeto concepto, le damos valores y grabamos
97           WITH OBJECT concepto := Concepto():new()
98              :setDescripcion( ::formulario:EdDescripcion:Value )
99              :grabar()
100          END WITH
101          // añadimos el concepto a la lista
102          ::listaConceptos:add( concepto )
103          // añadimos al browse
104          ::brw:BrConceptos:AddRow( concepto:toArray() )
105       END IF
106    END WITH
107
108 RETURN nil
109
```

5.- FASES EN EL DESARROLLO Y EJECUCIÓN DEL SOFTWARE

Para el desarrollo de nuestro software debemos elegir un modelo de **ciclo de vida**.
Independientemente del modelo elegido, siempre hay una serie de **etapas** que debemos seguir para construir software fiable y de calidad..



5.1.- Análisis.

Es la primera etapa del proyecto, la más complicada y la que más depende de la capacidad del analista.

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

¿Qué se hace en esta fase?

Se **especifican y analizan los requisitos** funcionales y no funcionales del sistema

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, tratamiento ante la simultaneidad de peticiones, etc.

PRODUCTO RESULTANTE:
documento ERS (Especificación de Requisitos Software). En este documento quedan especificados:

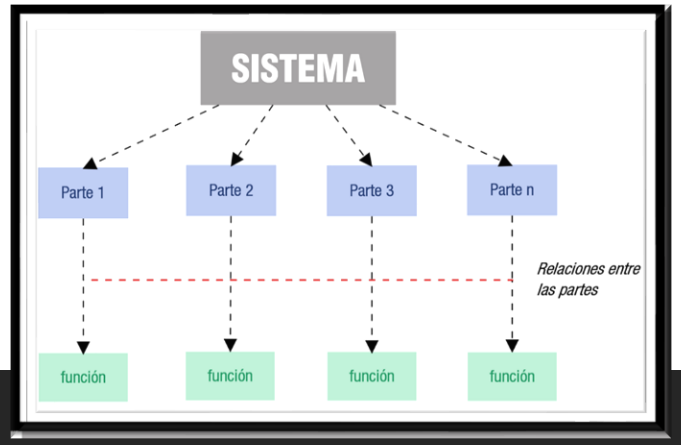
- La planificación de las reuniones que van a tener lugar.
- Relación de los objetivos del usuario cliente y del sistema.
- Relación de los requisitos funcionales y no funcionales del sistema.
- Relación de objetivos prioritarios y temporización.
- Reconocimiento de requisitos mal planteados o que conllevan contradicciones, etc.

5.2.- Diseño.

Durante esta fase, donde ya sabemos lo que hay que hacer, el siguiente paso es: ¿Cómo hacerlo?

Se debe dividir el sistema en partes y establecer qué relaciones habrá entre ellas. Decidir qué hará exactamente cada parte.

En definitiva, debemos crear un modelo funcional-estructural de los requerimientos del sistema global, para poder dividirlo y afrontar las partes por separado.



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 SGBD.
- Definición de diagrama de clases.
- Definición de diagrama de colaboración.
- Definición de diagrama de paso de mensajes.
- Definición de diagrama de casos de uso.

Etc.

5.3.- Codificación. Tipos de código.

Durante la fase de codificación se realiza el proceso de programación. Consiste en elegir un determinado lenguaje de programación, codificar toda la información anterior y llevarlo a código fuente.

Esta tarea la realiza el programador y tiene que cumplir exhaustivamente con todos los datos impuestos en el análisis y en el diseño de la aplicación.

Las características deseables de todo código son:

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

FASES

Código Fuente: es el escrito por los programadores en un algún lenguaje de programación y contiene el conjunto de instrucciones necesarias.

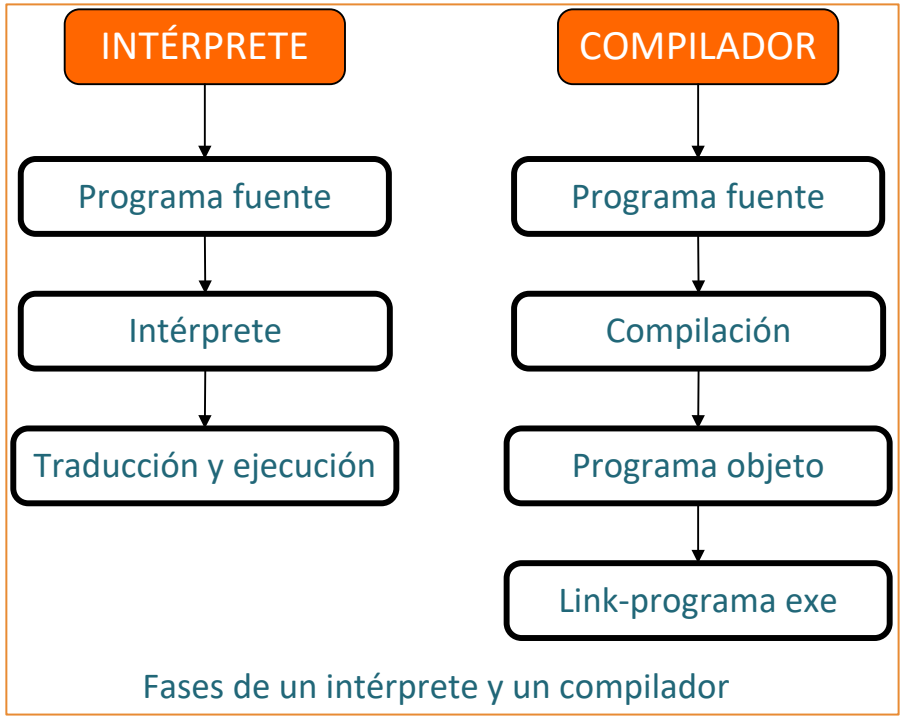
Código Objeto o bytecode: es el código binario resultado de compilar el código fuente. Es intermedio entre el código fuente y el ejecutable y sólo existe si el programa se compila, ya que si se interpreta (traducción línea a línea del código) se traduce y se ejecuta en un solo paso.

Código Ejecutable: Es el código binario resultante de enlazar los archivos de código objeto con ciertas rutinas y bibliotecas necesarias. El sistema operativo será el encargado de cargar el código ejecutable en memoria RAM y proceder a ejecutarlo. Es el que entiende la computadora

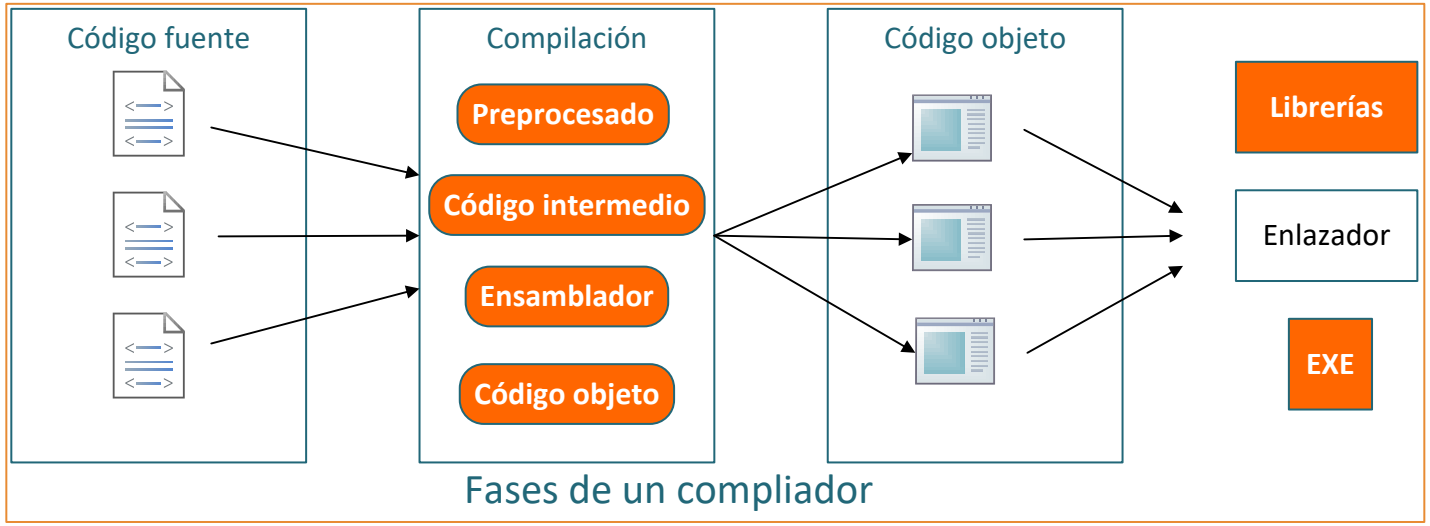
5.3.- Codificación. Tipos de código.

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

Los programas interpretados no producen código objeto. El paso de fuente a ejecutable es directo.



Fases de un intérprete y un compilador



Fases de un compliador

El programa enlazador inserta en el código objeto las funciones de librería necesarias para producir el programa ejecutable.

CLASE 10-10-2022

¿QUÉ HEMOS VISTO HASTA EL MOMENTO?

Hemos comenzado nuestro primer Resultado de Aprendizaje (R.A.1)

Cuando lo terminemos debemos se capaces de:

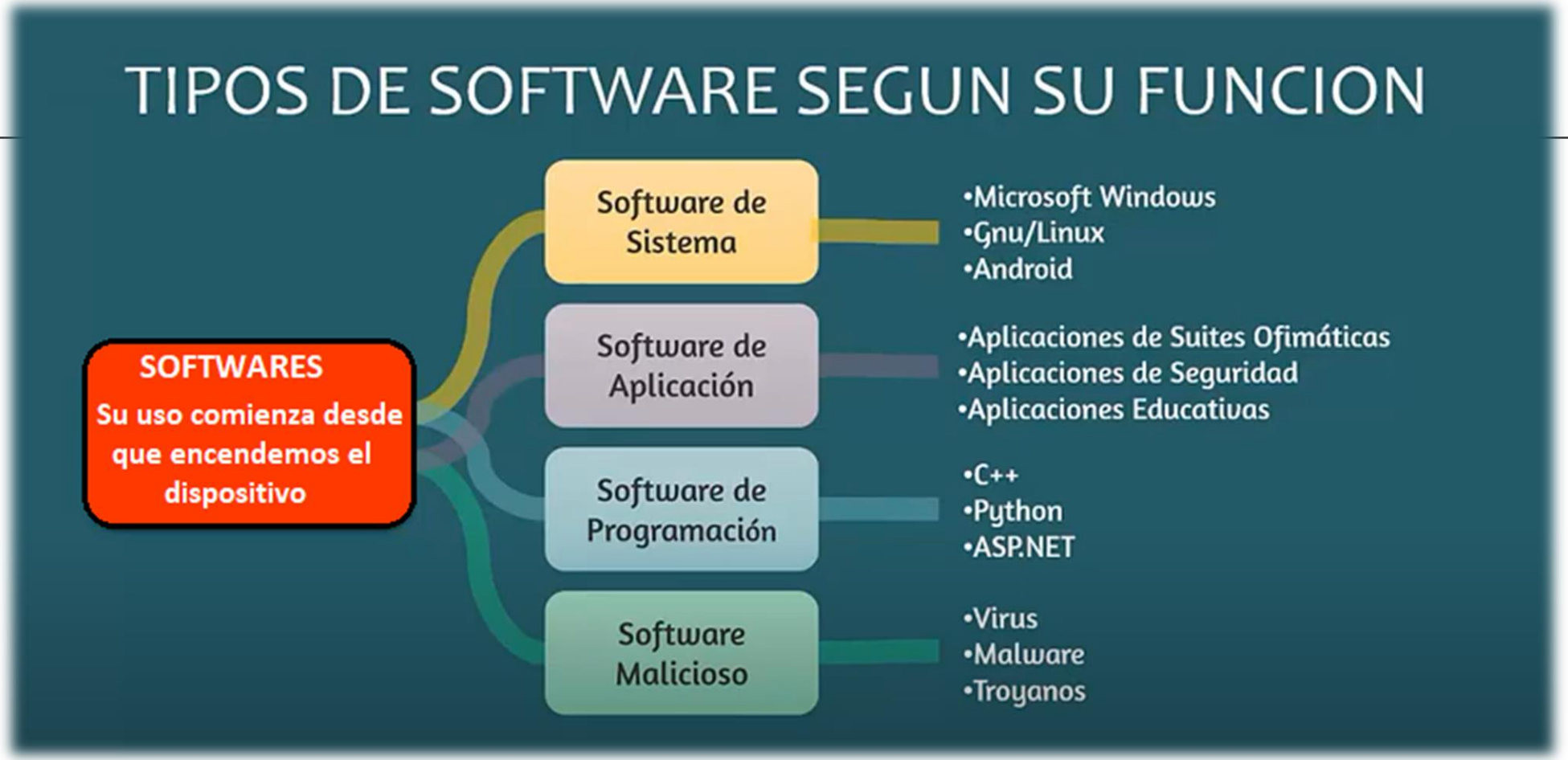
Reconocer los elementos y herramientas que intervienen en el desarrollo de un programa informático

Analizando su características

Y Fases

HASTA SU PUESTA EN FUNCIONAMIENTO

1.- SOFTWARE SEGÚN SU FUNCIÓN



https://www.youtube.com/watch?v=6Dfd-W0KXf4&ab_channel=NancyRosmeryTorrezOrtega

2.- RELACIÓN INDISOLUBLE HARDWARE-SOFTWARE

Desde el punto de vista del sistema operativo: coordina el hardware durante el funcionamiento del ordenador, actuando como intermediario entre éste y las aplicaciones.

Desde el punto de vista de las aplicaciones: conjunto de programas escritos en algún lenguaje de programación que el hardware del equipo debe interpretar y ejecutar.

¿CREEIS QUE SON INDEPENDIENTES ENTRE SÍ?

3.- DESARROLLO DE SOFTWARE. 3.1. Ciclos de vida

https://www.youtube.com/watch?v=4D1nhygWOKw&ab_channel=GersonStick

3.2.- Herramientas de apoyo al desarrollo del software.

Las **herramientas CASE** son un conjunto de aplicaciones que se utilizan en el desarrollo de software con el objetivo de reducir costes y tiempo del proceso, mejorando por tanto la productividad del proceso.

https://www.youtube.com/watch?v=lQhpa4gr7cU&ab_channel=AngiePacheco

4.- LENGUAJES DE PROGRAMACIÓN

Son los que nos permiten comunicarnos con el hardware del ordenador.

Los lenguajes de bajo nivel son dependientes del hardware

EVOLUCIÓN →

- ***Lenguaje máquina***
  - Sus instrucciones son combinaciones de unos y ceros
  - Es el único lenguaje que entiende directamente el ordenador
  - Fue el primer lenguaje utilizado
  - Es único para cada procesador (no es portable de un equipo a otro)
  - Hoy día nadie programa en este lenguaje
- ***Lenguaje ensamblador***
  - Sustituyó al lenguaje máquina para facilitar la labor de programación
  - Se programa usando mnemotécnicos (instrucciones complejas)
  - Necesita traducción al lenguaje máquina para poder ejecutarse
  - Sus instrucciones que hacen referencia a la ubicación física de los archivos
  - Es difícil de utilizar

Lenguajes de bajo nivel

- ***Lenguaje de alto nivel***
  - Sustituyeron al lenguaje ensamblador
  - Utilizan sentencias y órdenes derivadas del idioma inglés.
    - *Estruturados*
    - *Orientado a objetc*
  - Necesita traducción al lenguaje máquina
  - Son más cercanos al razonamiento humano
  - Son utilizados hoy día

Lenguajes de alto nivel

- ***Lenguajes visuales***
  - Están sustituyendo a los lenguajes de alto nivel basados en código
  - Se programa gráficamente usando el ratón
  - Su correspondiente código se genera automáticamente
  - Necesitan traducción al lenguaje máquina
  - Son completamente portables de un equipo a otro

Los lenguajes de alto nivel son independientes de la arquitectura del ordenador

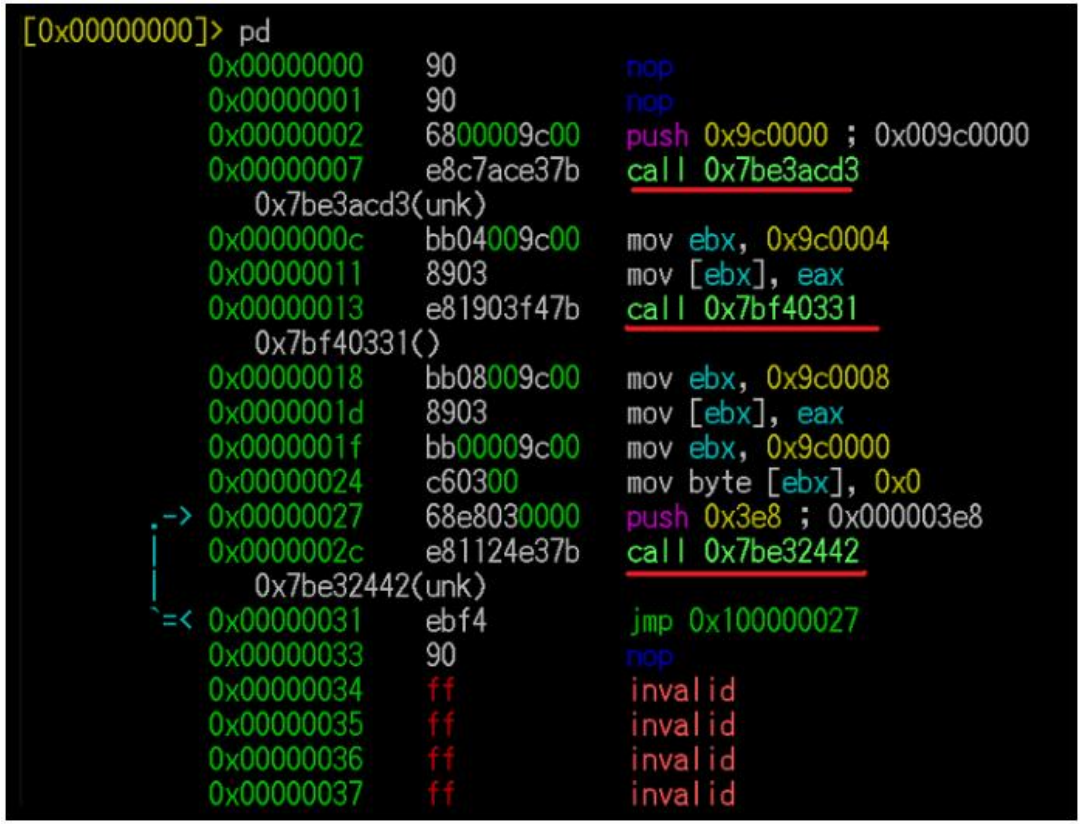
4.- LENGUAJES DE PROGRAMACIÓN

Lenguaje máquina

Lenguaje de máquina - tabla binaria.

```
11001010 00010111 11110101 00101011
00010111 11110101 00101011 00101011
11001010 00010111 11110101 00101011
00010111 11110101 00101011 00101011
11001010 11110101 00101011 00101011
11001010 11001010 11110101 00101011
11001010 11110101 00101011 00101011
11001010 00010111 11110101 00101011
00010111 11110101 00101011 00101011
11001010 11110101 00101011 00101011
```

Lenguaje ensamblador



4.- LENGUAJES DE PROGRAMACIÓN

4.2.- Lenguajes estructurados. (C)

```c
/* Suma de n números */

#include <stdio.h>
int main() {
    int num=0,suma=0;

    do {
        suma=suma+num;
        printf("un número: ");
        scanf("%d",&num);
    } while(num>=0);
    printf("suma es: %d",suma);
    return 0;
}
```

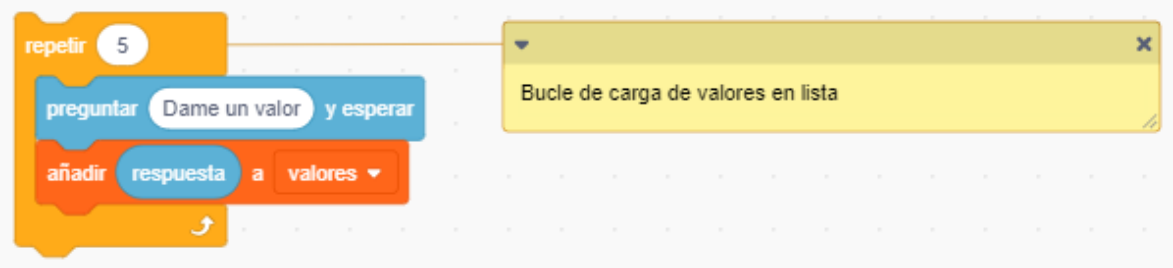
4.3.- Lenguajes orientados a objetos.

```
84  METHOD nuevo() CLASS ConceptoCtr
85  // ALTA
86
87     LOCAL concepto
88
89     WITH OBJECT ::formulario := ConceptoFrm():new()
90        :cText   := "Nuevo concepto"
91        :cargo   := "A"
92        // asignamos este controlador al formulario
93        :setControlador( self )
94
95        IF :ShowModal() == mrOk
96           // creamos un nuevo objeto concepto, le damos valores y grabamos
97           WITH OBJECT concepto := Concepto():new()
98              :setDescripcion( ::formulario:EdDescripcion:Value )
99              :grabar()
100          END WITH
101          // añadimos el concepto a la lista
102          ::listaConceptos:add( concepto )
103          // añadimos al browse
104          ::brw:BrConceptos:AddRow( concepto:toArray() )
105       END IF
106    END WITH
107
108 RETURN nil
109
```

4.- LENGUAJES DE PROGRAMACIÓN

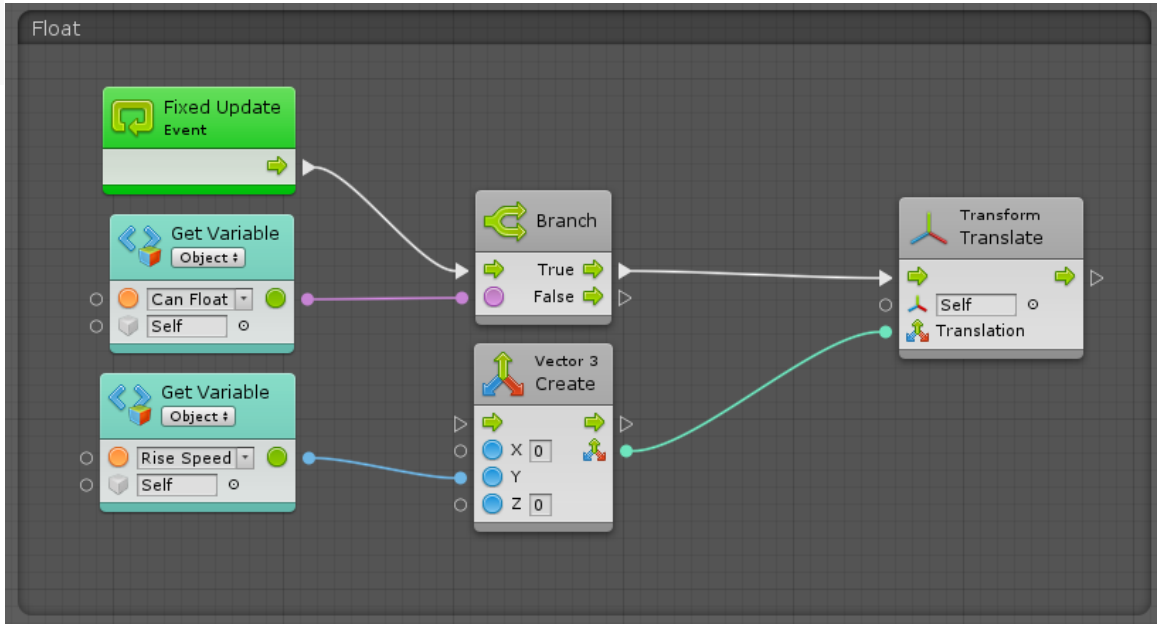
4.4.- Lenguajes visuales

En la programación visual se emplean bloques gráficos con diferentes colores y formas geométricas que pueden acoplarse o enlazarse intuitivamente del mismo modo que un puzle o un juego de construcción.



Ejemplo de programación visual mediante acoplamiento en Scratch

Ejemplo de programación visual mediante enlaces en Bolt para Unity



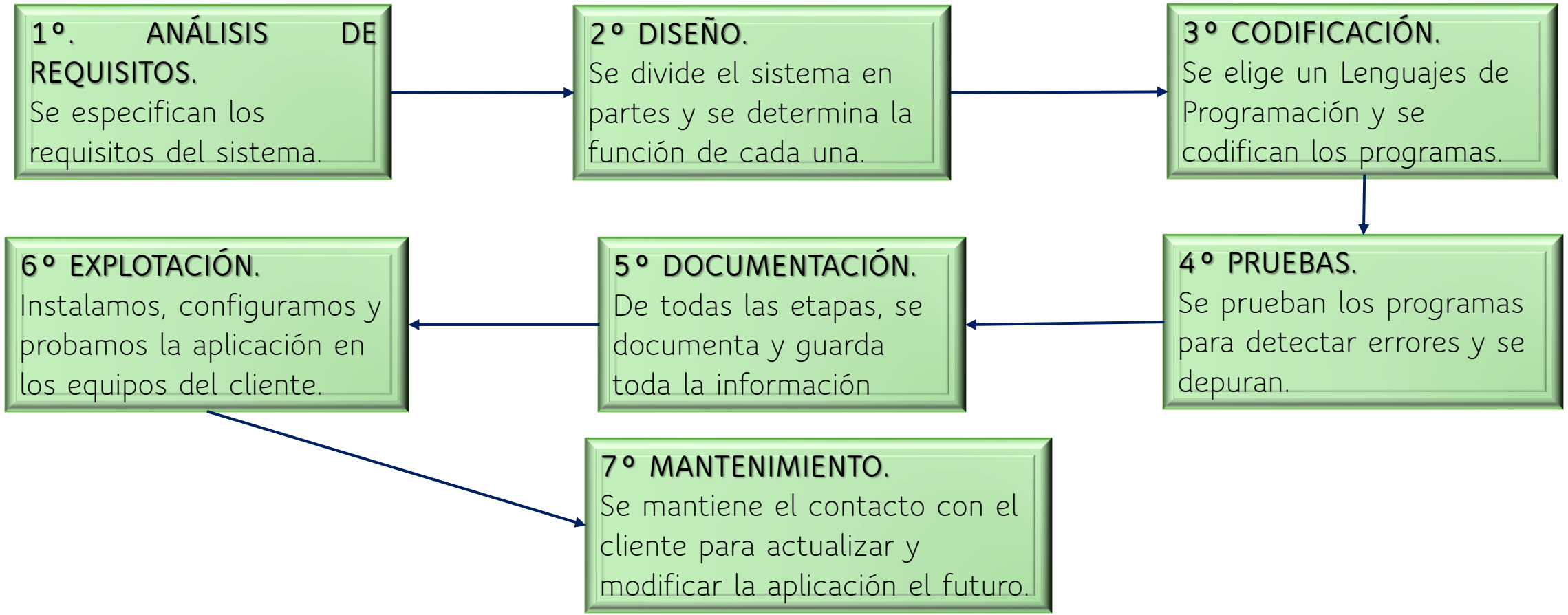
Enlace interesante

https://es.slideshare.net/yungeovanny/p-estructurada-vs-programacin-orientada-a-objetos

https://cipsa.net/programacion-visual/

5.- FASES EN EL DESARROLLO Y EJECUCIÓN DEL SOFTWARE

Para el desarrollo de nuestro software debemos elegir un modelo de **ciclo de vida**.
Independientemente del modelo elegido, siempre hay una serie de **etapas** que debemos seguir para construir software fiable y de calidad..



5.1.- Análisis.

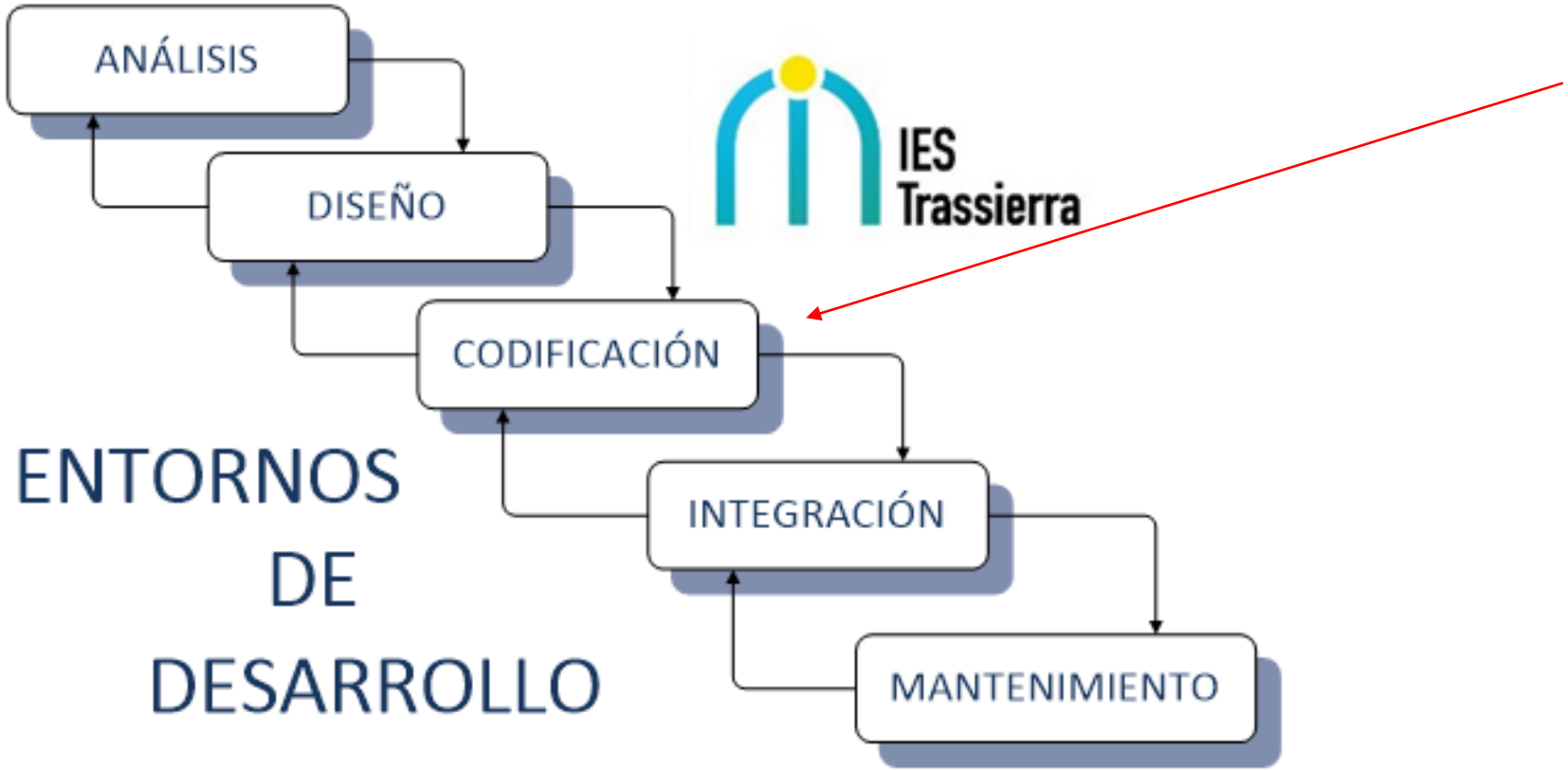
https://www.youtube.com/watch?v=50aSAUFrALk&ab_channel=SebastianGilParga

5.2.- Diseño.

Durante esta fase, donde ya sabemos lo que hay que hacer, el siguiente paso es: ¿Cómo hacerlo?

https://www.youtube.com/watch?v=tKJeEXwZ1QA&ab_channel=CatchExceptionCanal

Realizado el análisis y el diseño habría que pasar a la codificación



5.3.- Codificación. Tipos de código.

Durante la fase de codificación se realiza el proceso de programación. Consiste en elegir un determinado lenguaje de programación, codificar toda la información anterior y llevarlo a código fuente.

Esta tarea la realiza el programador y tiene que cumplir exhaustivamente con todos los datos impuestos en el análisis y en el diseño de la aplicación.

Las características deseables de todo código son:

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

FASES

Código Fuente: es el escrito por los programadores en un algún lenguaje de programación y contiene el conjunto de instrucciones necesarias.

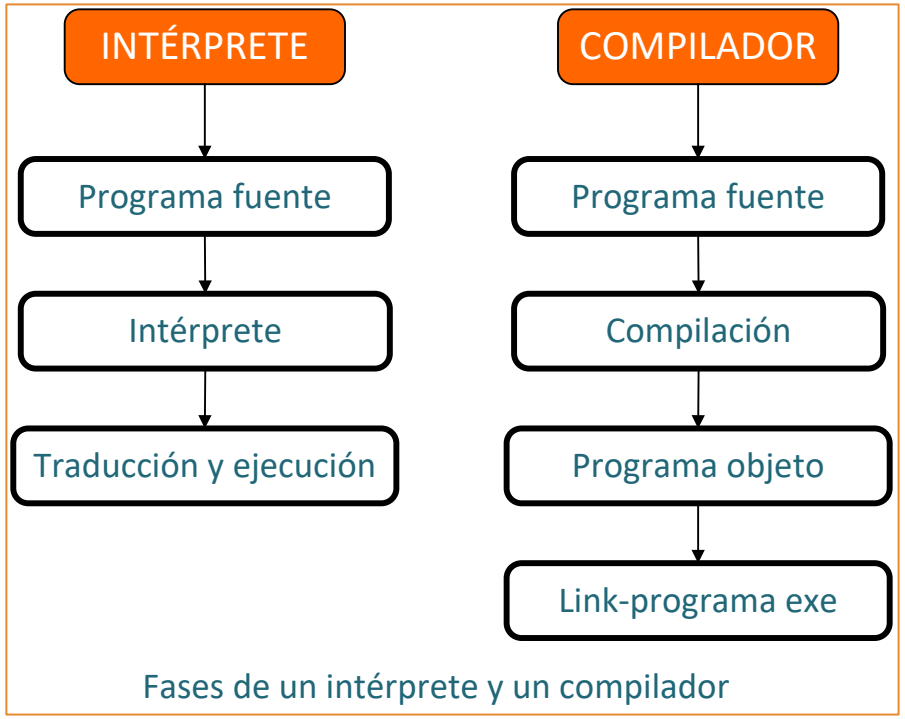
Código Objeto o bytecode: es el código binario resultado de compilar el código fuente. Es intermedio entre el código fuente y el ejecutable y sólo existe si el programa se compila, ya que si se interpreta (traducción línea a línea del código) se traduce y se ejecuta en un solo paso.

Código Ejecutable: Es el código binario resultante de enlazar los archivos de código objeto con ciertas rutinas y bibliotecas necesarias. El sistema operativo será el encargado de cargar el código ejecutable en memoria RAM y proceder a ejecutarlo. Es el que entiende la computadora

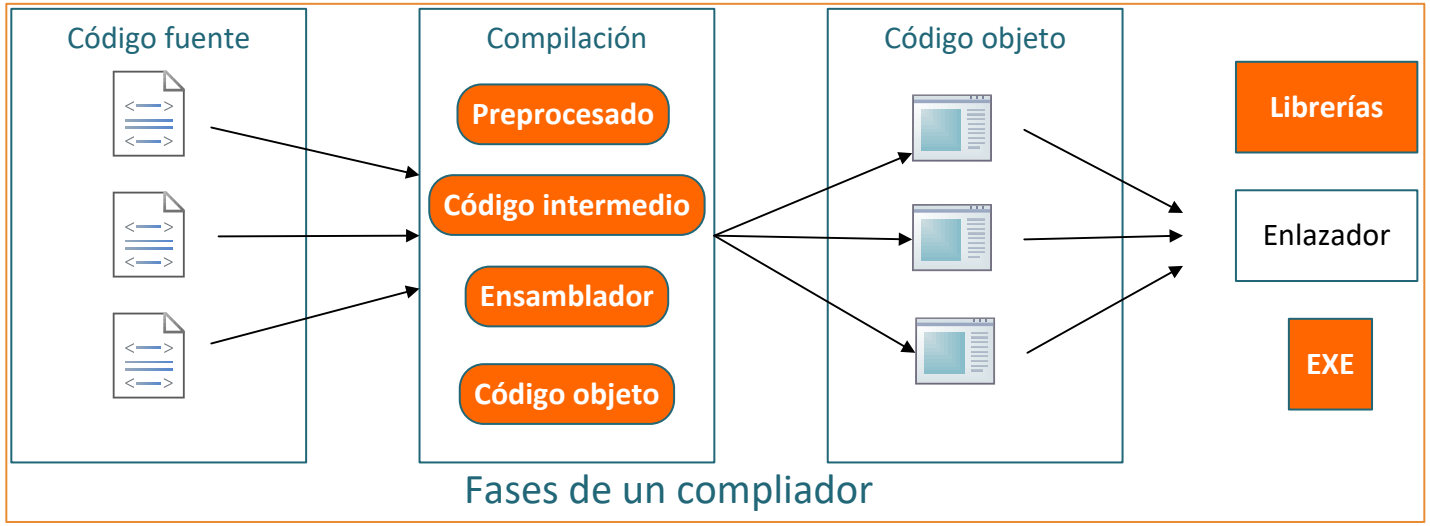
5.3.- Codificación. Tipos de código.

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

Los programas interpretados no producen código objeto. El paso de fuente a ejecutable es directo.



Fases de un intérprete y un compilador



Fases de un compliador

El programa enlazador inserta en el código objeto las funciones de librería necesarias para producir el programa ejecutable.

5.5.- Máquinas virtuales.

Una máquina virtual es un tipo especial de software cuya misión es **separar el funcionamiento del ordenador de los componentes hardware instalados**. Esta capa de software desempeña un papel muy importante en el funcionamiento de los lenguajes de programación, tanto compilados como interpretados.

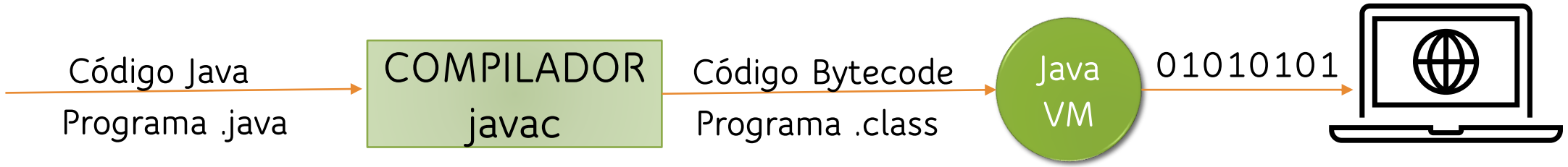
Las funciones principales de una máquina virtual son las siguientes:

- Conseguir que las aplicaciones sean portables.
- Reservar memoria para los objetos que se crean y liberar la memoria no utilizada.
- Comunicarse con el sistema donde se instala la aplicación (huésped), para el control de los dispositivos hardware implicados en los procesos
- Cumplimiento de las normas de seguridad de las aplicaciones.

5.5.- Máquinas virtuales.

El ejemplo más conocido de máquina virtual es la máquina virtual de Java.

Los programas compilados en lenguajes 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 Java. El proceso de compilación y ejecución de un programa Java se muestra en le siguiente esquema:



5.3.- Codificación. Tipos de código.

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

Los programas interpretados no producen código objeto. El paso de fuente a ejecutable es directo.

https://www.youtube.com/watch?v=ojqusTBq4A0&ab_channel=EDteam

Introductorio próximas apartados

https://www.daypo.com/entornos-desarrollo-preguntas-uf1.html#test

CLASE 17-10-2022

¿QUÉ HEMOS VISTO HASTA EL MOMENTO?

Hemos comenzado nuestro primer Resultado de Aprendizaje (R.A.1)

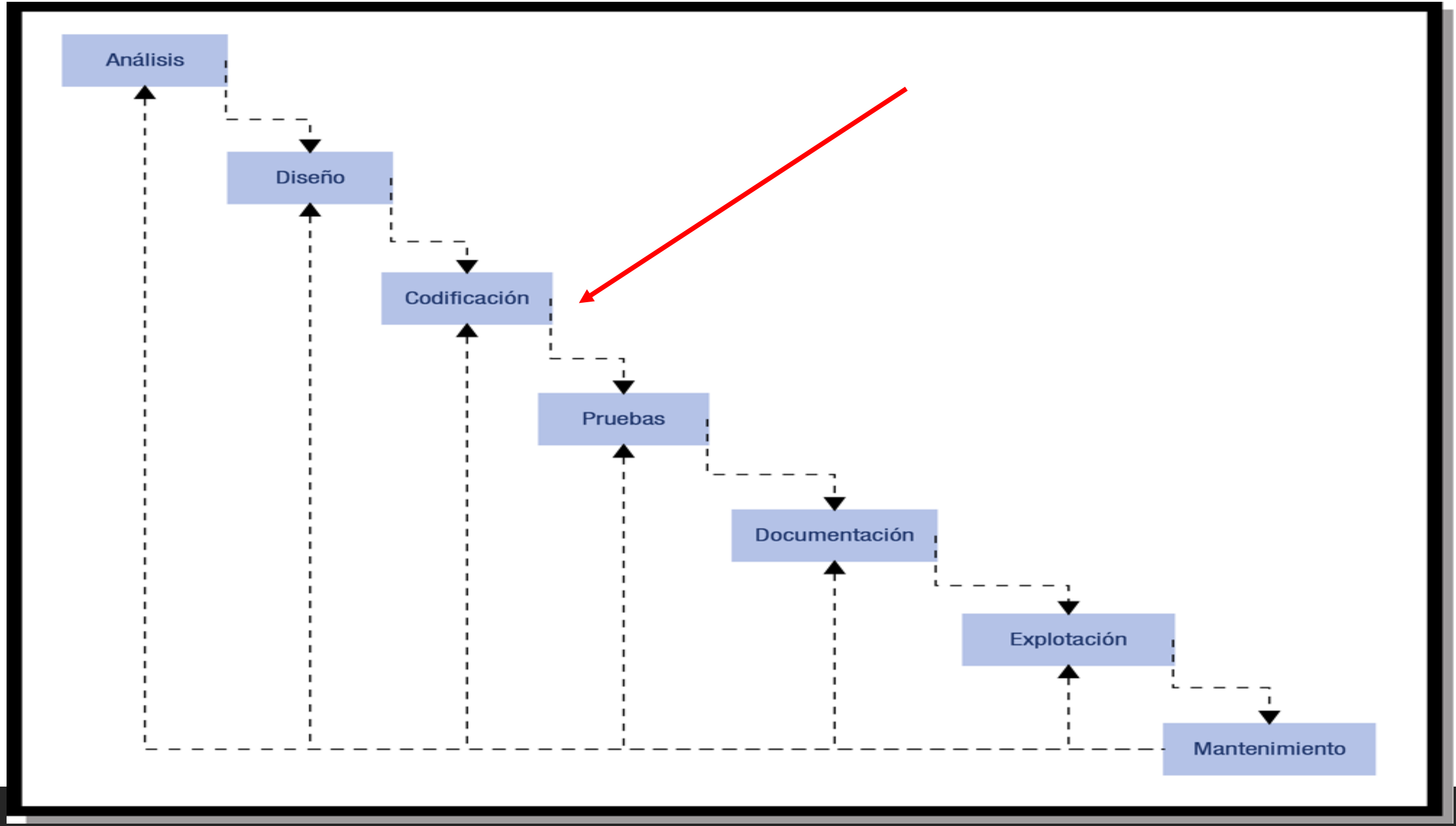
Cuando lo terminemos debemos se capaces de:

Reconocer los elementos y herramientas que intervienen en el desarrollo de un programa informático

Analizando su características

Y Fases

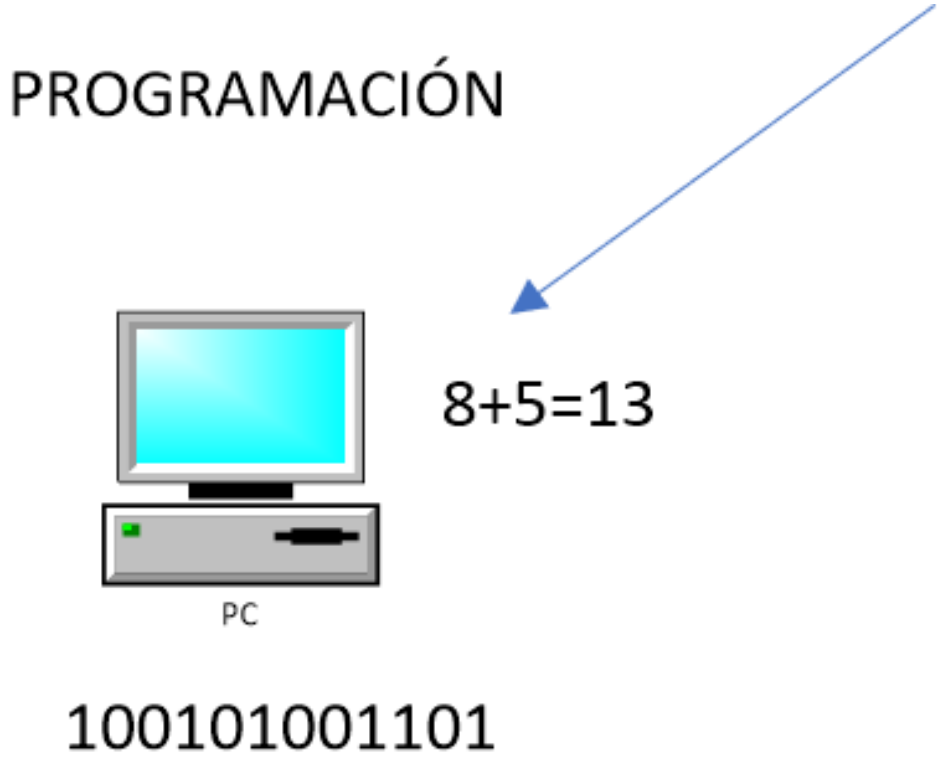
HASTA SU PUESTA EN FUNCIONAMIENTO


Análisis
Diseño
Codificación
Pruebas
Documentación
Explotación
Mantenimiento

5.3.- Codificación. Tipos de código.

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

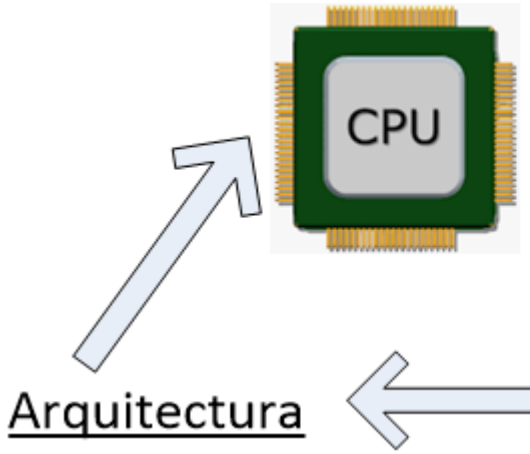
PROGRAMAR ES DARLE INSTRUCCIONES A LA COMPUTADORA.



NECESITAMOS UNA MANERA DE INDICARLE A LA COMPUTADORA QUE DEBE REALIZAR EL CÁLCULO 8+5, SIN NECESIDAD DE HACERLO EN BINARIO

5.3.- Codificación. Tipos de código.

El ordenador tiene un chip, llamado CPU y esa CPU tiene una ARQUITECTURA (cómo están construidos los componentes y cómo la CPU recibe las instrucciones) → Para poder dar instrucciones a esa arquitectura de procesador se usa algo llamado ENSAMBLADOR



El ensamblador se utiliza para poder dar instrucciones a la CPU

ENSAMBLADOR (Básicamente son instrucciones que no usan código binario, sino que usan unas instrucciones muy específicas de cada ARQUITECTURA

EJEMPLO: Decir "abre la puerta" en lenguaje ensamblador sería dar instrucciones "paso a paso": gira a la izquierda→ da tres pasos → extiende la mano → coge el pomo → gíralo en media vuelta → tira hacia ti….

AUNQUE NO PROGRAMAMOS EN BINARIO SIGUE SIENDO "MUY BAJO NIVEL"

En 1952 → Sacó el primer compilador llamado A-0

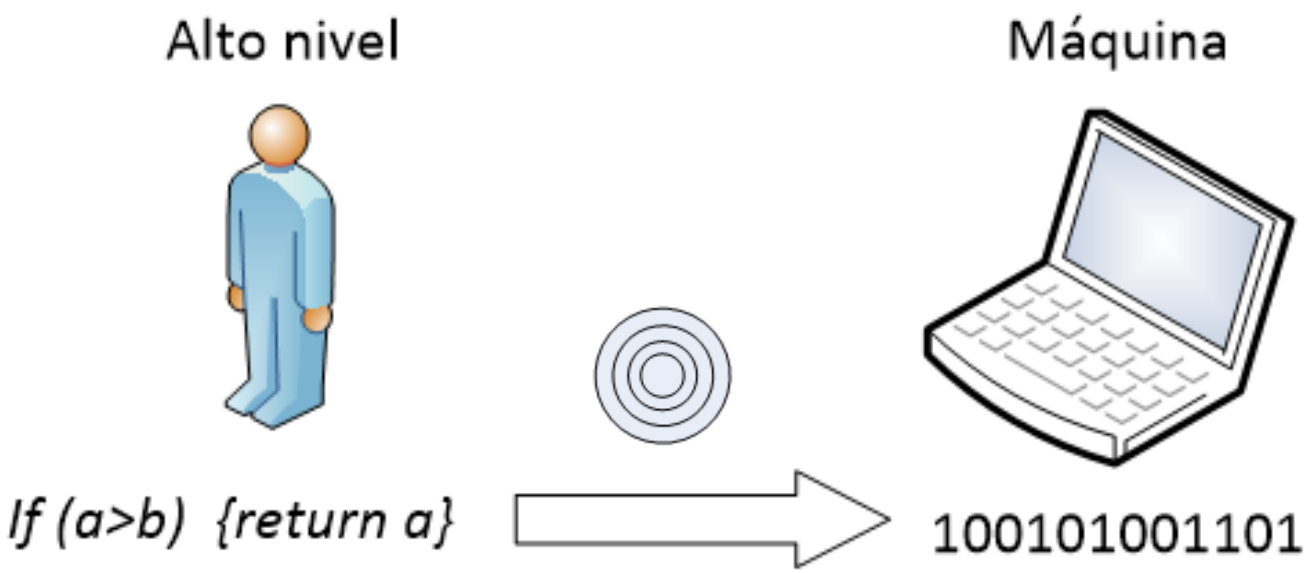
1949 Grace Hooper ¿Por qué no programamos en inglés?

En 1953 → El 2º compilador llamado B-0

5.3.- Codificación. ¿QUÉ ES UN COMPILADOR?

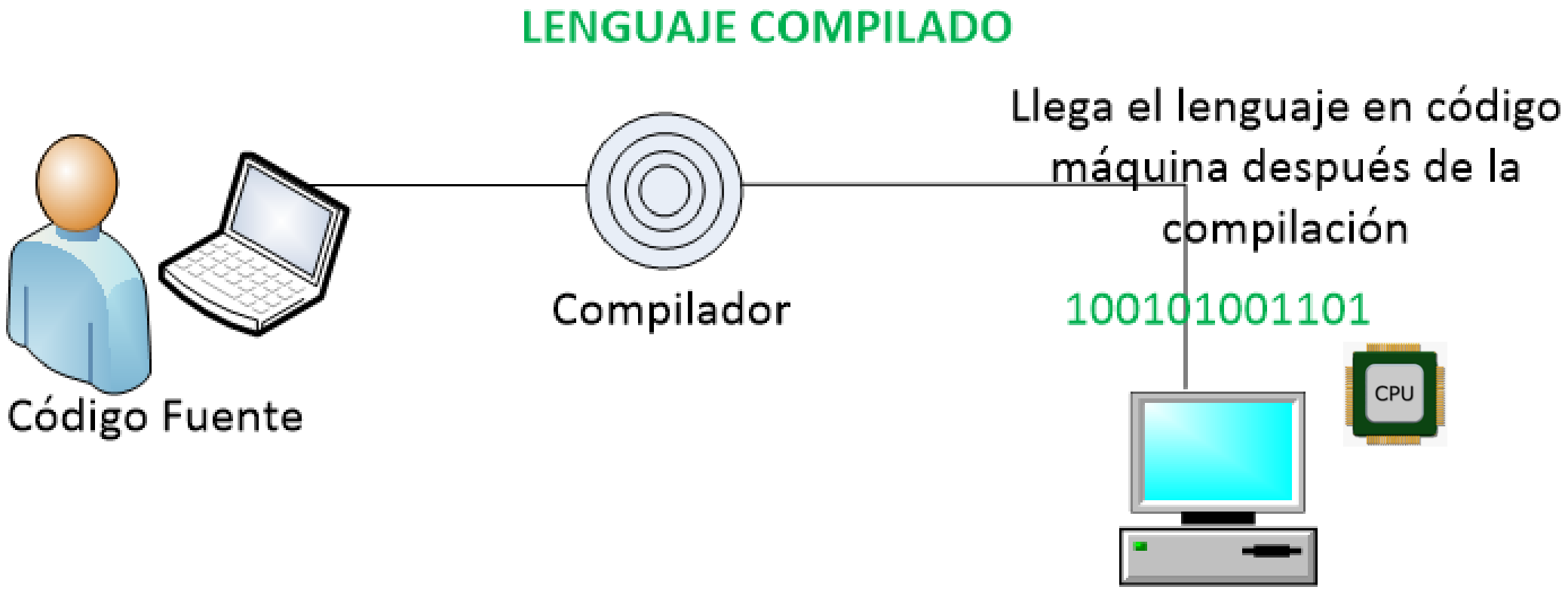
COMPILADOR:

Traductor de lenguaje “alto nivel” a lenguaje máquina



LOS PRIMEROS LENGUAJES EN USAR ESTOS COMPILADORES FUERON FORTRAN Y COBOL

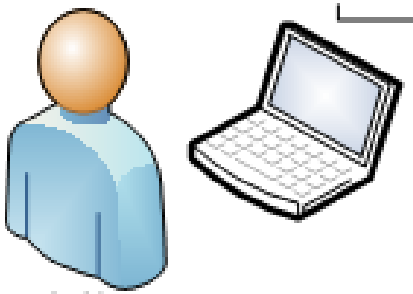
5.3.- Codificación. ¿QUÉ ES UN LENGUAJE COMPILADO “ORIGINAL”?



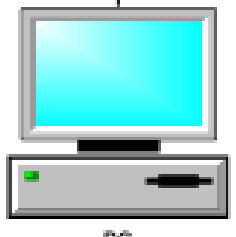
5.3.- Codificación. ¿QUÉ ES UN LENGUAJE INTERPRETADO?

LENGUAJE INTERPRETADO

A la computadora llega el lenguaje en código fuente



Código Fuente



PC

intérprete

runtime

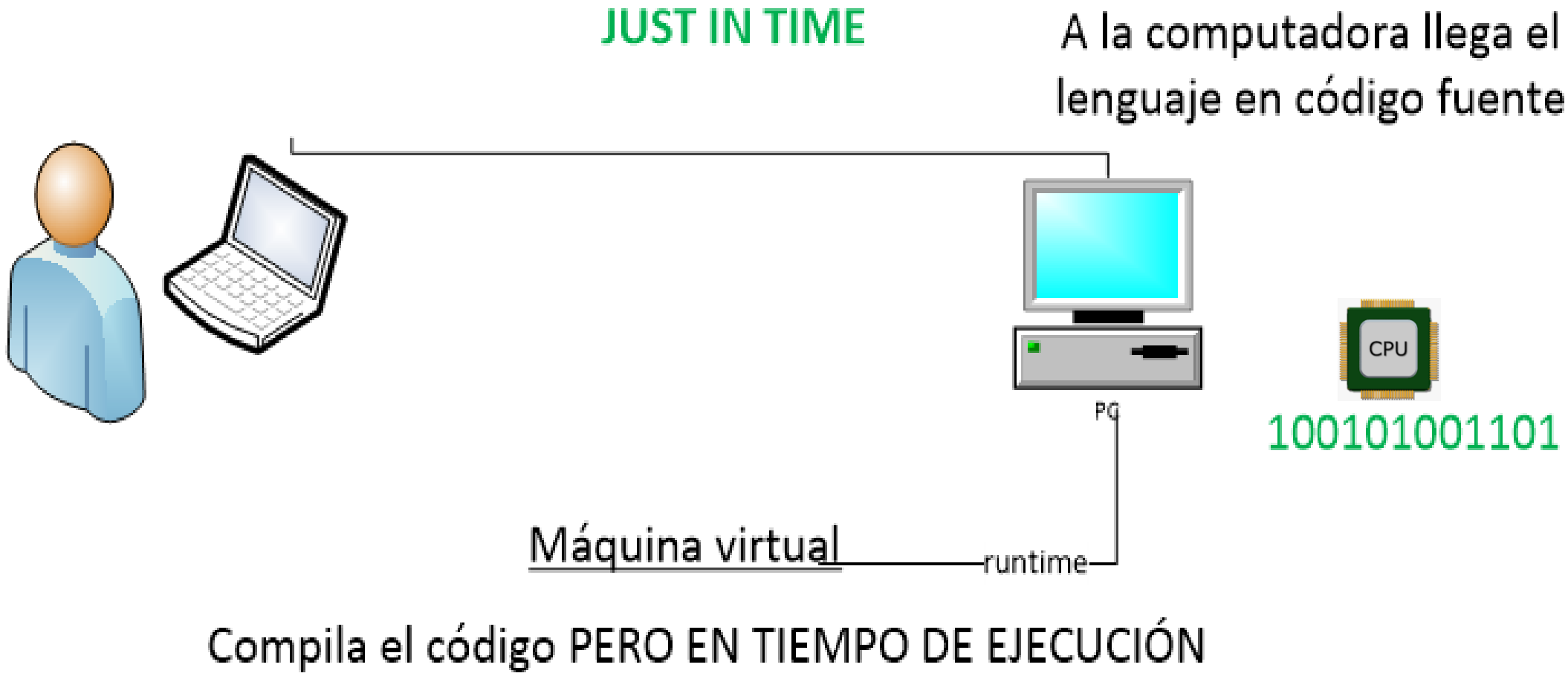
Y dentro de la computadora hay un intérprete que es otro software que se encarga de traducir a código máquina, pero en tiempo de ejecución

100101001101

Por ejemplo, si fuese PHP el tiempo de ejecución sería cuando un usuario entra en la página web. En ese mismo momento, el intérprete, traduce línea a línea

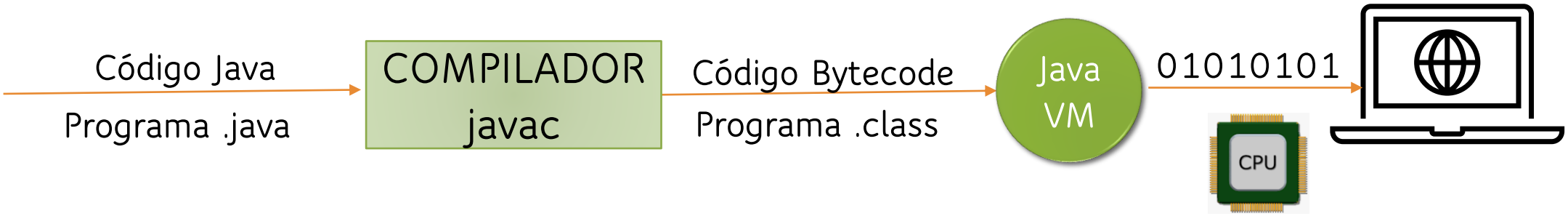
ESTO ES MÁS LENTO QUE LA COMPILACIÓN

5.3.- Codificación. JUST IN TIME (JIT) Mezcla entre compilado e interpretado



5.3.- Codificación. Comparativa

Lenguajes compilador originales	JUST IN TIME (O actuales compilados)	Lenguajes interpretados
C, C++, COBOL, Fortran Son traducidos directamente a ´código máquina" antes desean ejecutados en la computadora	INTERMEDIOS Se compilan + Se interpretan	PHP, JavaScript, Python Llegan en código fuente a la computadora, dónde hay un intérprete que traduce línea a línea en el momento de ejecución (MÁS LENTO)
	JAVA	
	Cuando se compila JAVA, no lo hace a lenguaje máquina, sino a algo llamado "Bytecode" o "código objeto" que tiene que pasar luego por una JVM (máquina virtual Java) que interpreta el "código objeto" y lo manda a la CPU	



5.3.- Codificación. Tipos de código.

Durante la fase de codificación se realiza el proceso de programación. Consiste en elegir un determinado lenguaje de programación, codificar toda la información anterior y llevarlo a código fuente.

Esta tarea la realiza el programador y tiene que cumplir exhaustivamente con todos los datos impuestos en el análisis y en el diseño de la aplicación.

Las características deseables de todo código son:

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

FASES

Código Fuente: es el escrito por los programadores en un algún lenguaje de programación y contiene el conjunto de instrucciones necesarias.

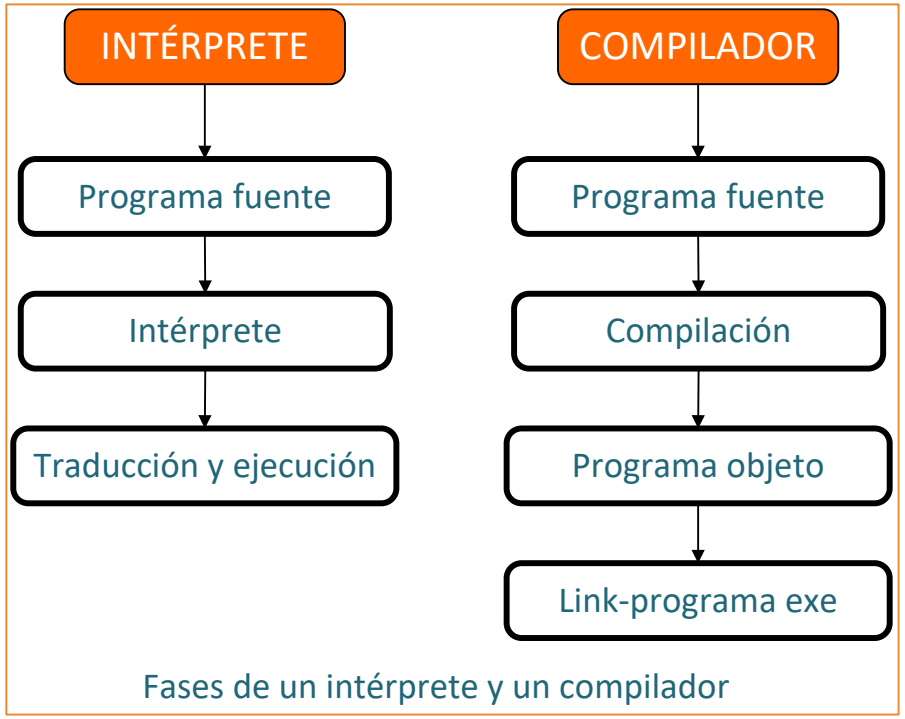
Código Objeto o bytecode: es el código binario resultado de compilar el código fuente. Es intermedio entre el código fuente y el ejecutable y sólo existe si el programa se compila, ya que si se interpreta (traducción línea a línea del código) se traduce y se ejecuta en un solo paso.

Código Ejecutable: Es el código binario resultante de enlazar los archivos de código objeto con ciertas rutinas y bibliotecas necesarias. El sistema operativo será el encargado de cargar el código ejecutable en memoria RAM y proceder a ejecutarlo. Es el que entiende la computadora

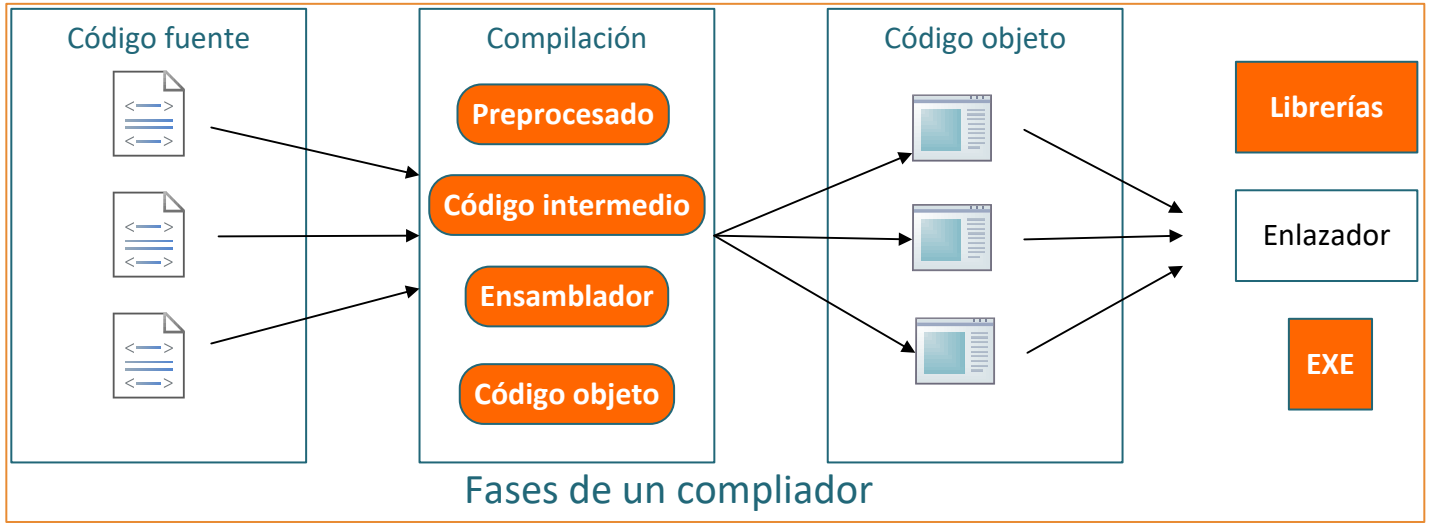
5.3.- Codificación. Tipos de código.

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

Los programas interpretados no producen código objeto. El paso de fuente a ejecutable es directo.



Fases de un intérprete y un compilador



Fases de un compliador

El programa enlazador inserta en el código objeto las funciones de librería necesarias para producir el programa ejecutable.

5.5.- Máquinas virtuales.

Una máquina virtual es un tipo especial de software cuya misión es **separar el funcionamiento del ordenador de los componentes hardware instalados**. Esta capa de software desempeña un papel muy importante en el funcionamiento de los lenguajes de programación, tanto compilados como interpretados.

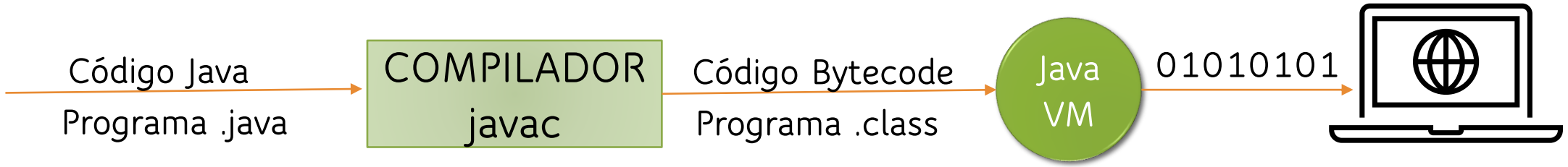
Las funciones principales de una máquina virtual son las siguientes:

- Conseguir que las aplicaciones sean portables.
- Reservar memoria para los objetos que se crean y liberar la memoria no utilizada.
- Comunicarse con el sistema donde se instala la aplicación (huésped), para el control de los dispositivos hardware implicados en los procesos
- Cumplimiento de las normas de seguridad de las aplicaciones.

5.5.- Máquinas virtuales.

El ejemplo más conocido de máquina virtual es la máquina virtual de Java.

Los programas compilados en lenguajes 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 Java. El proceso de compilación y ejecución de un programa Java se muestra en le siguiente esquema:



5.5.1.- Framework.

Un framework es una estructura de ayuda al programador, en base a la cual podemos desarrollar proyectos sin partir desde cero. Se trata de una plataforma software donde están definidos programas soporte, bibliotecas, lenguaje interpretado, etc., que ayuda a desarrollar y unir los diferentes módulos o partes de un proyecto.

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 computador a otro, ya que los bytecodes que se generan a partir del lenguaje fuente podrán ser ejecutados sobre cualquier máquina virtual

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 framework en nuestro equipo consume bastantes recursos del sistema.

Ejemplos de Frameworks:

.NET es un framework para desarrollar aplicaciones sobre Windows.

Spring de Java

5.3.- ¿Qué es Framework.?

https://www.youtube.com/watch?v=xmHwYc-6W3E&ab_channel=KikoPalomares

5.5.2.- ¿Qué es un entorno de ejecución?.

«Entorno de ejecución» es uno de esos muchos conceptos de la informática que pertenece a la categoría de «difíciles de definir con precisión pero fáciles de entender en la práctica».

Si queremos ser prácticos, basta con que entendamos que un lenguaje de programación sirve de poco si no hay «algo» capaz de transformar el código en un conjunto de instrucciones que ponga a trabajar a la computadora como el programador haya dispuesto en dicho código.

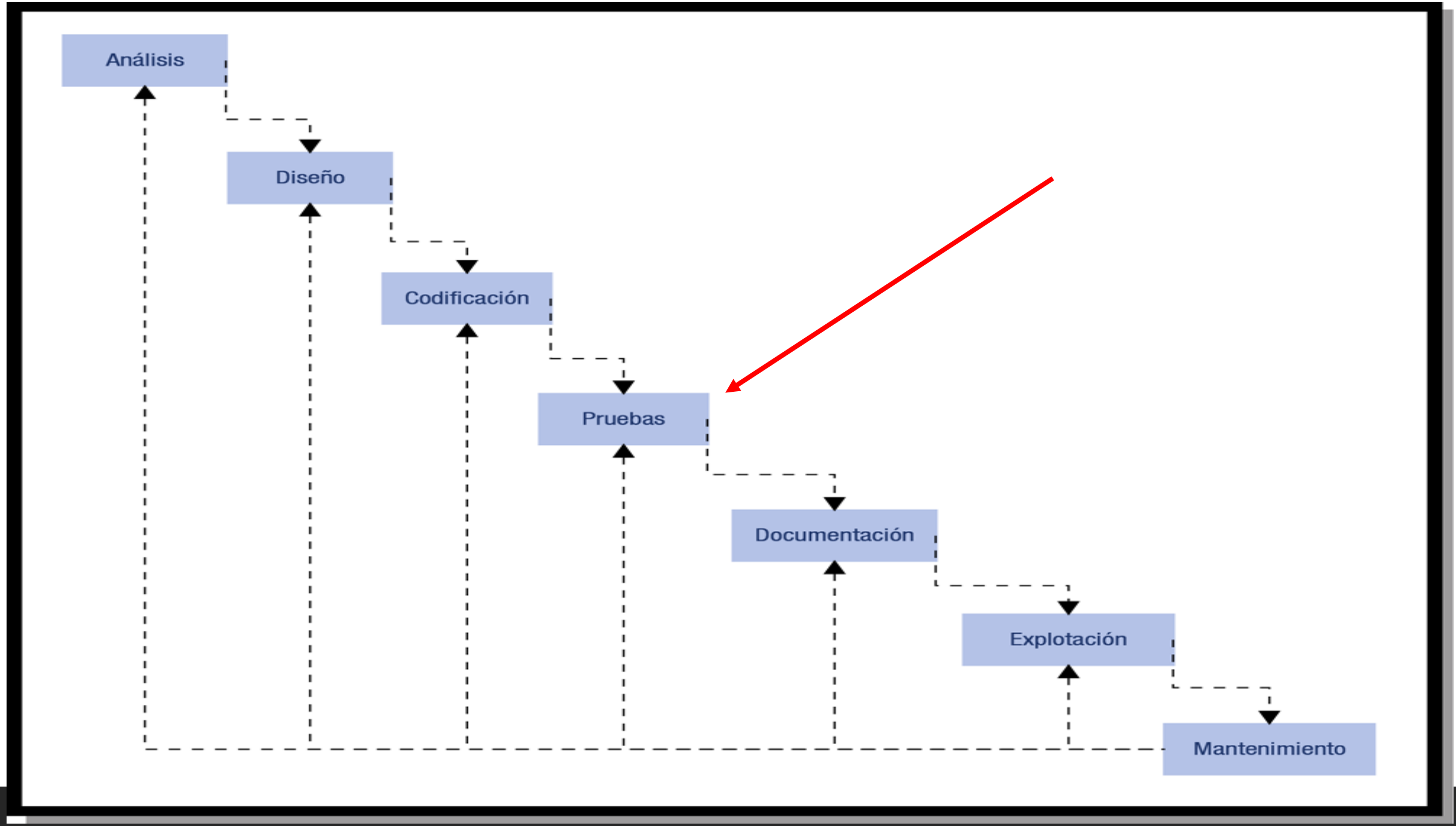
Por lo general siempre es necesario un compilador, intérprete o máquina virtual y un conjunto de librerías propias del lenguaje, pero puede constar de más elementos como un depurador o un bucle de eventos.

Cada vez que leas o escuches la expresión «run time environment» o entorno de ejecución, piensa en todo aquello que transforma el código fuente de un lenguaje de programación en operaciones reales de una computadora.

5.5.2.- Entornos de ejecución.

Funcionamiento:

- El entorno de ejecución está formado por la máquina virtual y los API's (bibliotecas de clases estándar, necesarias para que la aplicación, escrita en algún Lenguaje de Programación, pueda ser ejecutada).
- Intermediario entre el lenguaje fuente y el sistema operativo, y consigue ejecutar aplicaciones.
- Si lo que queremos es desarrollar nuevas aplicaciones, no es suficiente con el entorno de ejecución, necesitaremos un entorno de desarrollo.


Análisis
Diseño
Codificación
Pruebas
Documentación
Explotación
Mantenimiento

5.6.- Pruebas.

Una vez obtenido el software, la siguiente fase del ciclo de vida es la realización de pruebas.

La realización de pruebas es imprescindible para asegurar la validación y verificación del software construido.

Verificación

Proceso por el que se comprueba que el software cumple los requisitos especificados.

Validación

Proceso que comprueba si el software hace lo que el usuario deseaba. Tiene que estar verificado.

Los tipos de pruebas básicos son:

- 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..

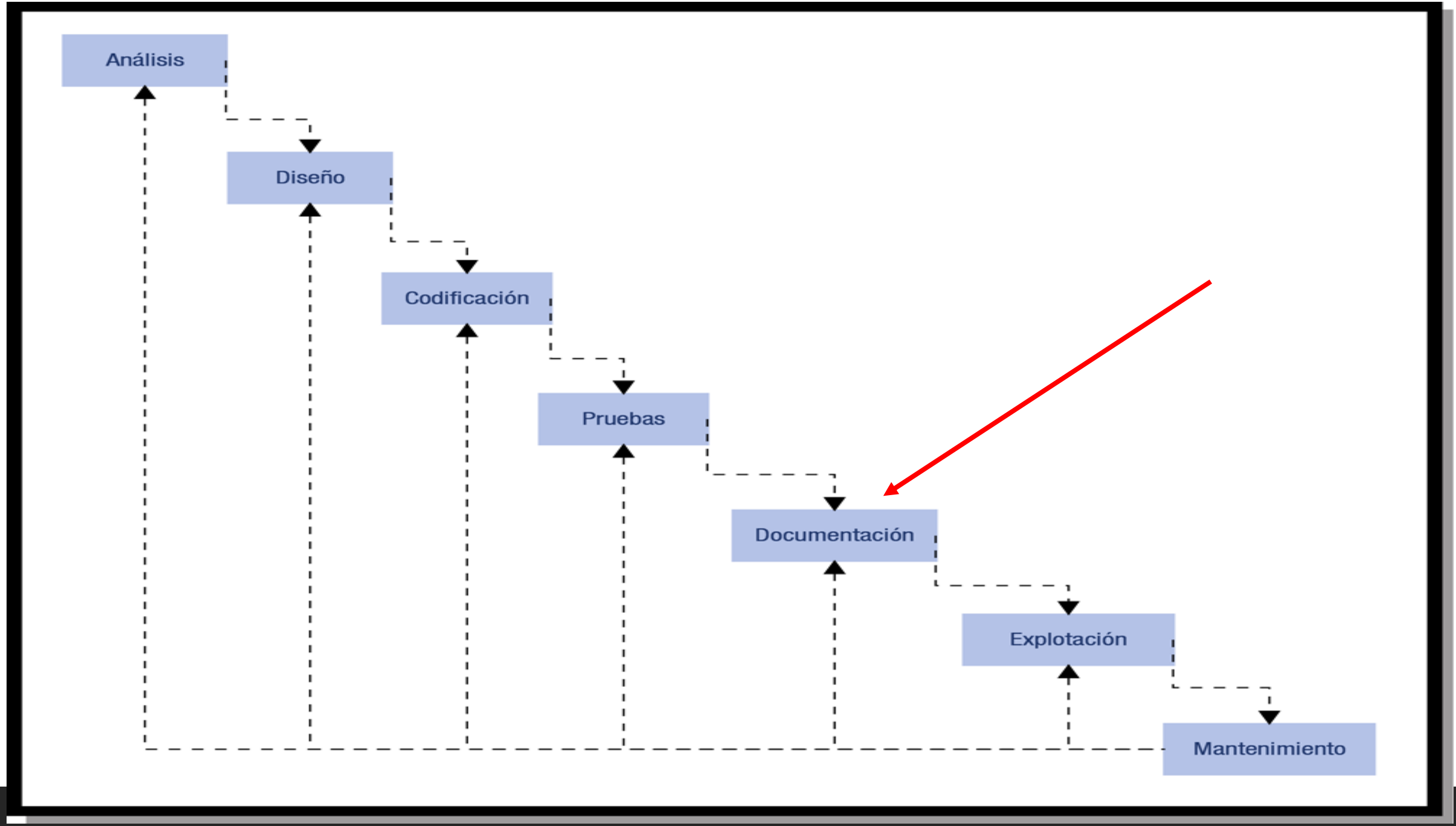
- PRUEBAS DE INTEGRACIÓN

  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.

- La **prueba final** se denomina comúnmente Beta Test, ésta se realiza sobre el entorno de producción donde el software va a ser utilizado por el cliente (a ser posible, en los equipos del cliente y bajo un funcionamiento normal de su empresa).

5.6.- Pruebas.

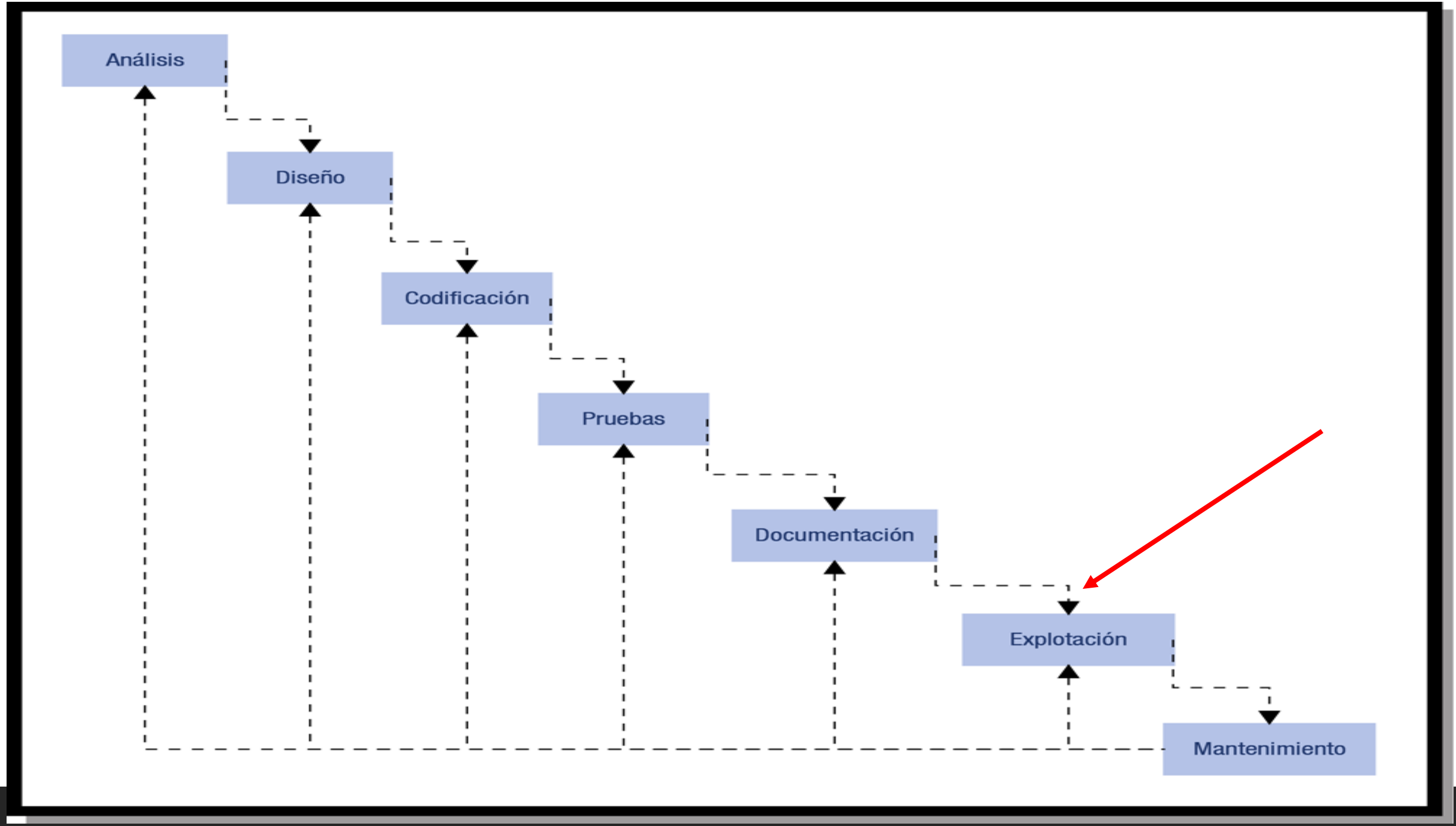
Examen de la unidad	Pruebas de integración
Se lleva a cabo en la fase inicial de prueba y luego se puede realizar en cualquier momento	Debe realizarse después de la prueba unitaria y antes de la prueba del sistema.
Prueba el componente único de todo el sistema, es decir, prueba una unidad de forma aislada.	Prueba los componentes del sistema trabajando juntos, es decir, prueba la colaboración de múltiples unidades.
Más rápido de ejecutar	Puede ser lento
Sin dependencia externa. Cualquier dependencia externa es eliminada.	Requiere interacción con dependencias externas (por ejemplo, base de datos, hardware, etc.)
Simple	Complejo
Realizado por desarrollador	Realizado por probador
Es un tipo de prueba de caja blanca (código y estructuras)	Es un tipo de prueba de caja negra (entradas y salidas)
Mantenimiento económico	Mantenimiento costoso
Comienza a partir de la especificación del módulo	Comienza desde la especificación de la interfaz
Las pruebas unitarias tienen un alcance limitado, ya que solo verifican si cada pequeño fragmento de código está haciendo lo que se pretende que haga.	Tiene un alcance más amplio ya que cubre toda la aplicación.
El resultado de las pruebas unitarias es una visibilidad detallada del código.	El resultado de las pruebas de integración es la visibilidad detallada de la estructura de integración.
Descubre los problemas dentro de la funcionalidad de módulos individuales únicamente..	Descubre los errores que surgen cuando diferentes módulos interactúan entre sí para formar el sistema general


Análisis
Diseño
Codificación
Pruebas
Documentación
Explotación
Mantenimiento

5.7.- Documentación

Todas las etapas en el desarrollo de software deben quedar perfectamente documentadas.

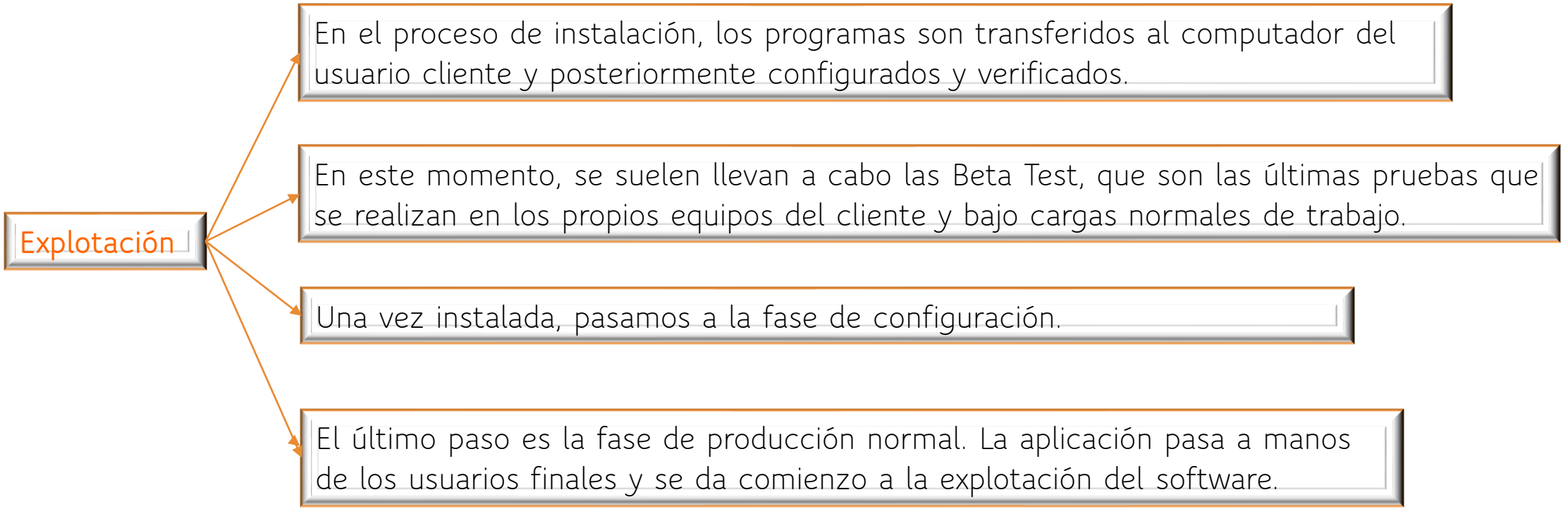
Documentos a elaborar en el proceso de desarrollo de software			
	GUÍA TÉCNICA	GUÍA DE USO	GUÍA DE INSTALACIÓN
Quedan reflejados:	•El diseño de la aplicación. •La codificación de los programas. •Las pruebas realizadas.	•Descripción de la funcionalidad de la aplicación. •Forma de comenzar a ejecutar la 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 quién va dirigido?	Al personal técnico en informática (analistas y programadores).	A los usuarios que van a usar la aplicación (clientes).	Al personal informático responsable de la instalación, en colaboración con los usuarios que van a usar la aplicación (clientes).
¿Cuál es su objetivo?	Facilitar un correcto desarrollo, realizar correcciones en los programas y permitir un mantenimiento futuro.	Dar a los usuarios finales toda la información necesaria para utilizar la aplicación.	Dar toda la información necesaria para garantizar que la implantación de la aplicación se realice de forma segura, confiable y precisa.

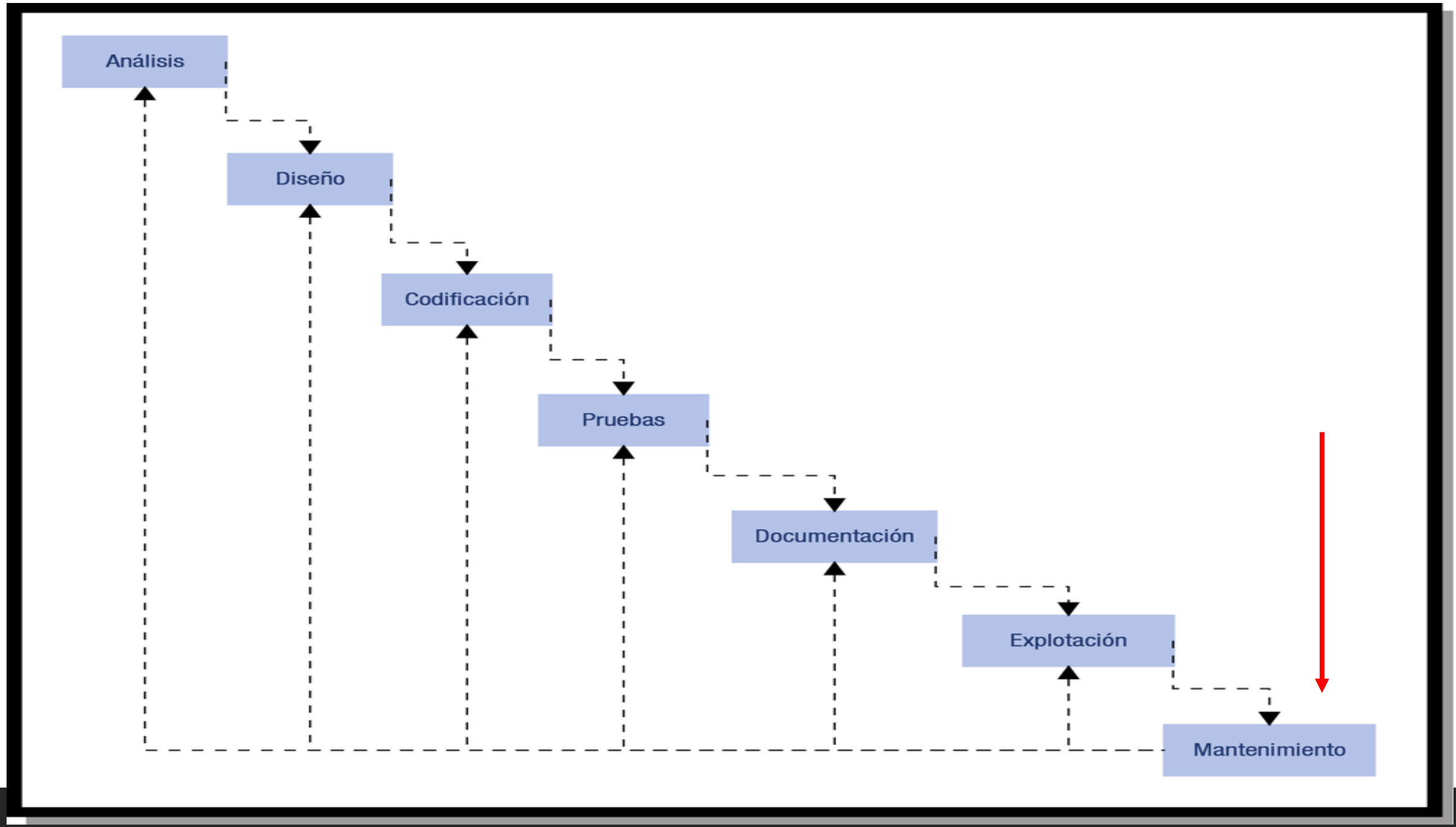

Análisis
Diseño
Codificación
Pruebas
Documentación
Explotación
Mantenimiento

5.8.- Explotación.

La explotación es la fase en que los usuarios finales conocen la aplicación y comienzan a utilizarla

La explotación es la instalación, puesta a punto y funcionamiento de la aplicación en el equipo final del cliente.




Análisis
Diseño
Codificación
Pruebas
Documentación
Explotación
Mantenimiento

5.9.- Mantenimiento.

El mantenimiento se define como el proceso de control, mejora y optimización del software. Su duración es la mayor en todo el ciclo de vida del software, ya que también comprende las actualizaciones y evoluciones futuras del mismo. Es la etapa más larga de todo el ciclo de vida del software.

Tipos de cambios que hacen necesario el mantenimiento del software

- 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).

6.- ENTORNOS DE DESARROLLO

6.1. Evolución Histórica.

- En las décadas de utilización de la **tarjeta perforada** como sistema de almacenamiento el concepto de Entorno de Desarrollo Integrado sencillamente no tenía sentido.

- Los programas estaban escritos con diagramas de flujo y entraban al sistema a través de las tarjetas perforadas. Posteriormente, eran compilados.

- El primer lenguaje de programación que utilizó un IDE fue el **BASIC** (que fue el primero en abandonar también las tarjetas perforadas o las cintas de papel).

- Éste primer IDE estaba basado en consola de comandos exclusivamente (normal por otro lado, si tenemos en cuenta que hasta la década de los 90 no entran en el mercado los sistemas operativos con interfaz gráfica). Sin embargo, el uso que hace de la gestión de archivos, compilación y depuración; es perfectamente compatible con los IDE actuales.

- A nivel popular, el primer IDE puede considerarse que fue el IDE llamado Maestro. Nació a principios de los 70 y fue instalado por unos 22.000 programadores en todo el mundo. Lideró este campo durante los años 70 y 80.

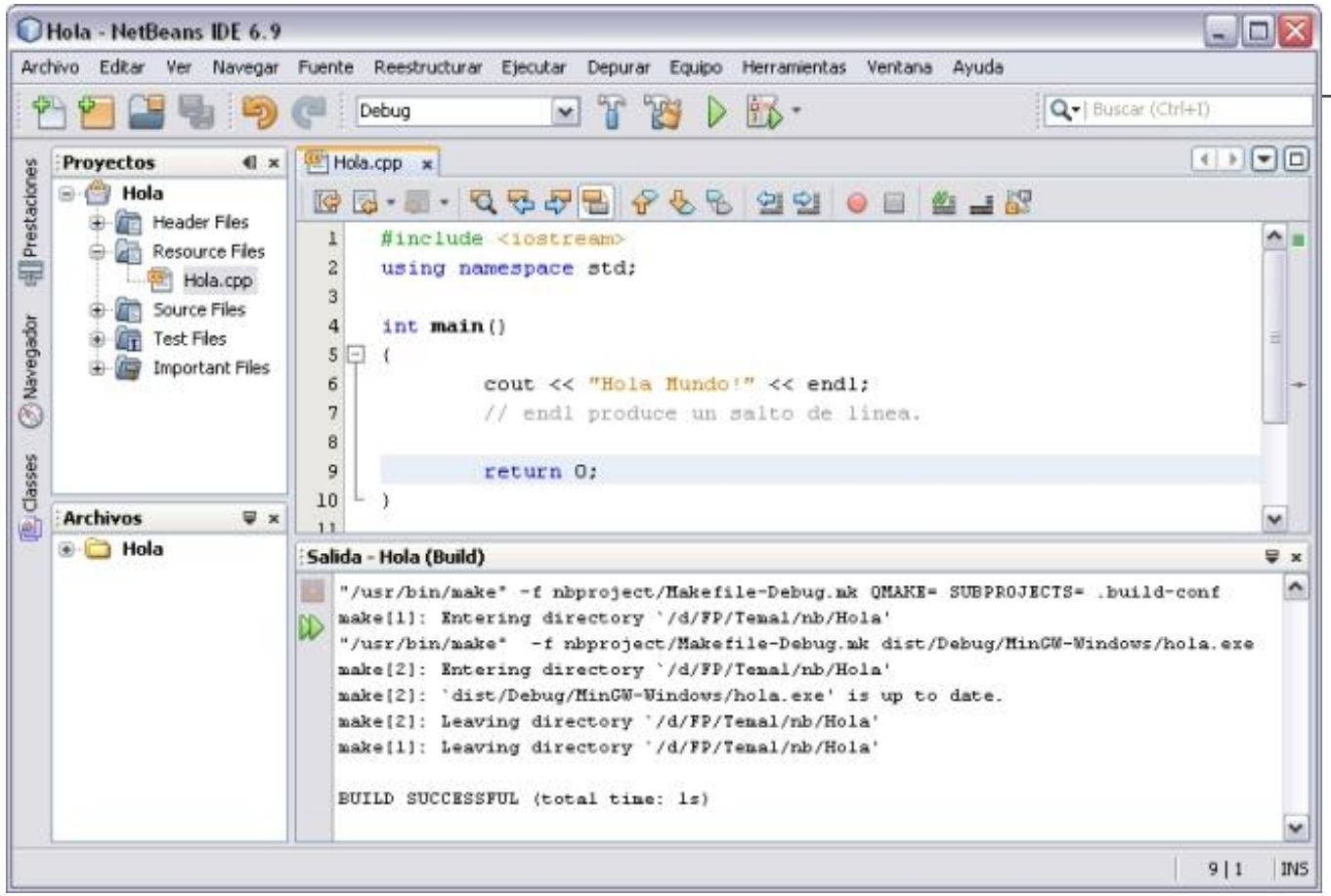
- El uso de los entornos integrados de desarrollo se ratifica y afianza en los 90 y hoy en día contamos con infinidad de IDE, tanto de licencia libre como no.

Tipos de entornos de desarrollo más relevantes en la actualidad

Entorno de desarrollo	Lenguajes que soporta	Tipo de licencia
NetBeans.	C/C++, Java, JavaScript, PHP, Python.	De uso público.
Eclipse.	Ada, C/C++, Java, JavaScript, PHP.	De uso público.
Microsoft Visual Studio.	Basic, C/C++, C#.	Propietario.
C++ Builder.	C/C++.	Propietario.
JBuilder.	Java.	Propietario.

No hay unos entornos de desarrollo más importantes que otros. La elección del IDE más adecuado dependerá del lenguaje de programación que vayamos a utilizar para la codificación de las aplicaciones y el tipo de licencia con la que queramos trabajar.

6.2.- Funciones de un entorno de desarrollo



FUNCIONES DE LOS ENTORNOS DE DESARROLLO

- Escribir código
- Compilar y depurar código
- Ensamblar componentes
- Desplegar aplicaciones
- Dar soporte a varios lenguajes

Entornos de desarrollo libres y propietarios

Tipos de entornos de desarrollo libres más relevantes en la actualidad.

IDE	Lenguajes que soporta	Sistema Operativo
NetBeans.	C/C++, Java, JavaScript, PHP, Python.	Windows, Linux, Mac OS X.
Eclipse.	Ada, C/C++, Java, JavaScript, PHP.	Windows, Linux, Mac OS X.
Gambas.	Basic.	Linux.
Anjuta.	C/C++, Python, Javascript.	Linux.
Geany.	C/C++, Java.	Windows, Linux, Mac OS X.
GNAT Studio.	Fortran.	Windows, Linux, Mac OS X.

Tipos de entornos de desarrollo propietarios más relevantes en la actualidad.

IDE	Lenguajes que soporta	Sistema Operativo
Microsoft Visual Studio.	Basic, C/C++, C#.	Windows.
FlashBuilder.	ActionScript.	Windows, Mac OS X.
C++ Builder.	C/C++.	Windows.
Turbo C++ profesional.	C/C++.	Windows.
JBuilder.	Java.	Windows, Linux, Mac OS X.
JCreator.	Java.	Windows.
Xcode.	C/C++, Java.	Mac OS X.

6.- ENTORNOS DE DESARROLLO

6.3. Estructura de un Entorno de Desarrollo.

En los inicios, para programar podía ser necesario un editor de textos, un compilador y un depurador. Todas estas herramientas se instalaban de forma independiente. **En un entorno se integran todas estas cosas y muchas más.**

Un IDE (Entorno Integrado de Desarrollo) es una aplicación informática que estará formada por un conjunto de herramientas de programación que simplifican la tarea al programador y agilizan el desarrollo de programas. Puede usarse con uno o varios lenguajes.

Editor de textos: Parte en la que escribimos el código fuente.

Compilador: Se encarga de traducir el código fuente escrito en lenguaje de alto nivel a un lenguaje de bajo nivel en el que la máquina sea capaz de interpretarlo y ejecutarlo.

Intérprete: Realiza la traducción a medida que se ejecuta la instrucción. Son más lentos que los compiladores, pero no dependen de la máquina sino del propio intérprete.

Depurador: Depura y limpia los errores en el código fuente. Permite detener el programa en cualquier punto de ruptura para examinar la ejecución.

Generador automático de herramientas: Para la visualización, creación y manipulación de componentes visuales y todo un arsenal de asistentes y utilidades de gestión y generación código.

Interfaz gráfica: Nos brinda la oportunidad de programar en varios lenguajes con un mismo IDE.

Control de versiones. Controla los cambios realizados sobre las aplicaciones,