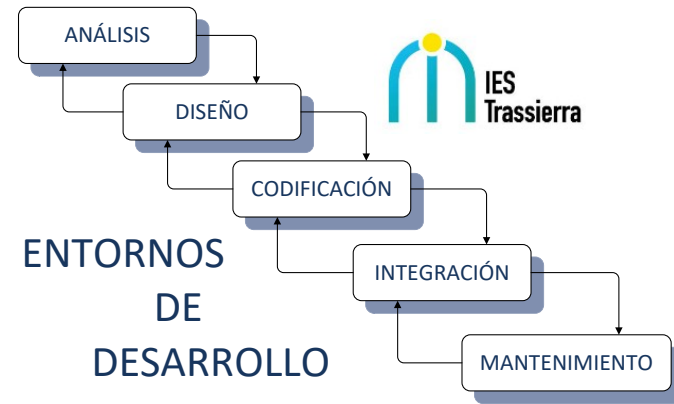




BIBLIOGRAFÍA:

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



UT 1: Desarrollo del software y entornos de desarrollo

Profesora: Elena Fernández Chirino
DEPARTAMENTO DE INFORMÁTICA

<http://www.iestrassierra.com>



This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).

CLASE 17-10-2022

¿QUÉ HEMOS VISTO HASTA EL MOMENTO?

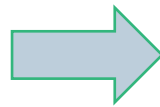
Hemos comenzado nuestro primer
Resultado de Aprendizaje (R.A.1)

Cuando lo terminemos debemos ser capaces de:

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

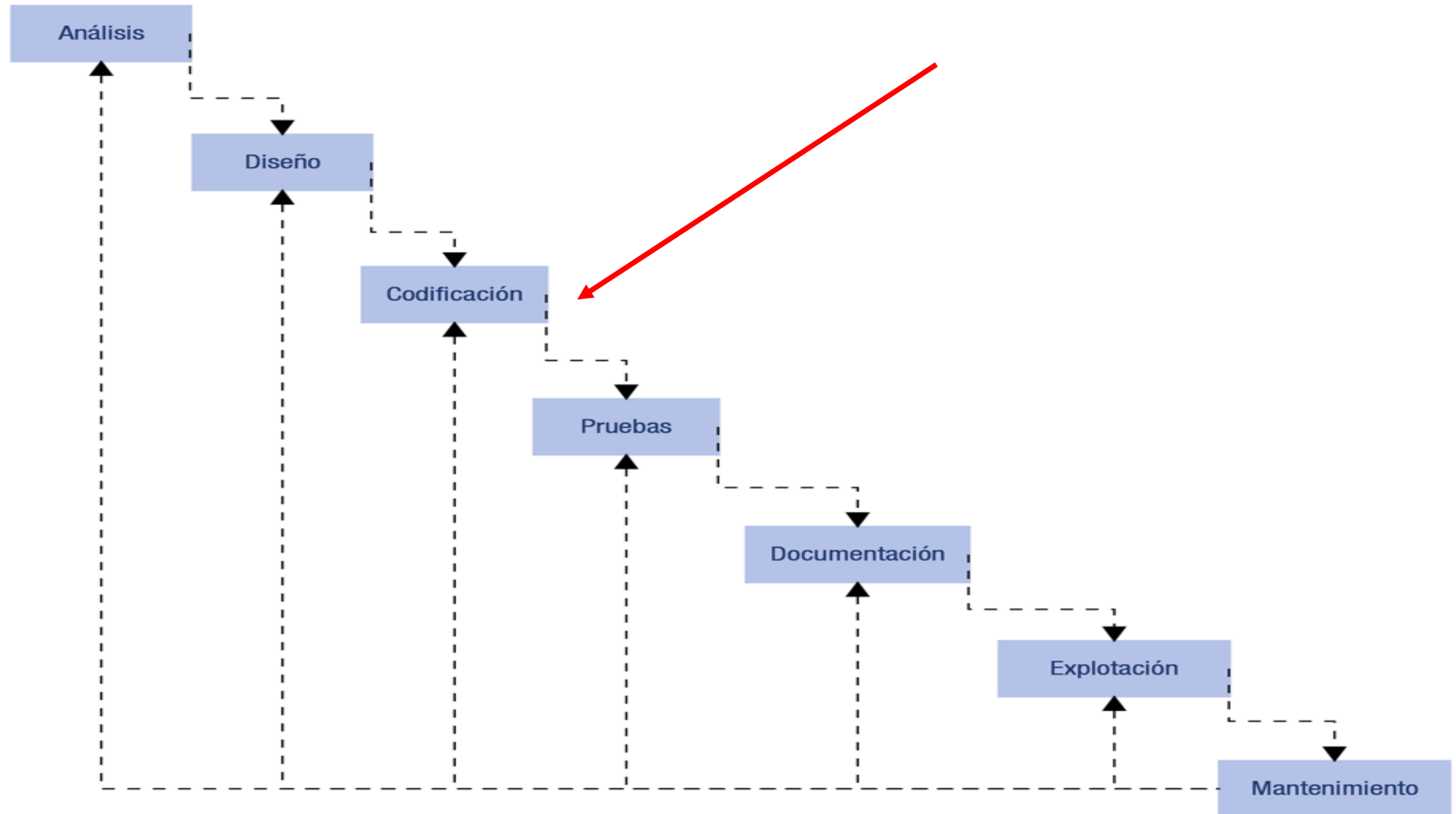
Analizando su características

Y Fases



HASTA SU PUESTA EN FUNCIONAMIENTO





5.3.- Codificación. Tipos de código.

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

PROGRAMAR ES DARLE INSTRUCCIONES A LA COMPUTADORA.

PROGRAMACIÓN



PC

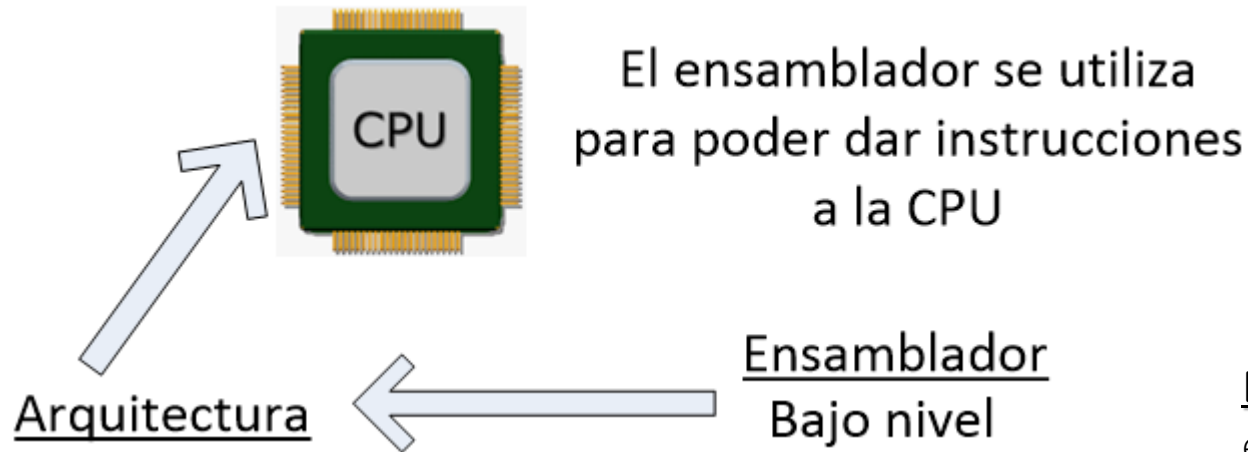
$8+5=13$

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

100101001101

5.3.- Codificación. Tipos de código.

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



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

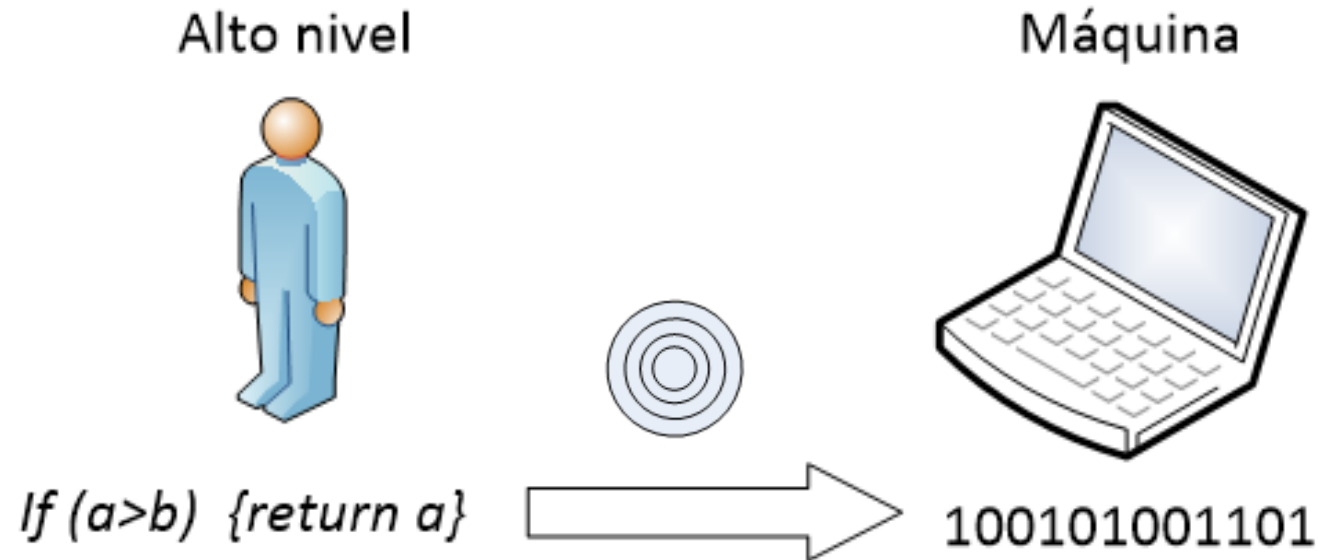
1949 Grace Hooper ¿Por qué no programamos en inglés?

En 1953 → El 2º compilador llamado B-0

5.3.- Codificación. ¿QUÉ ES UN COMPILADOR?

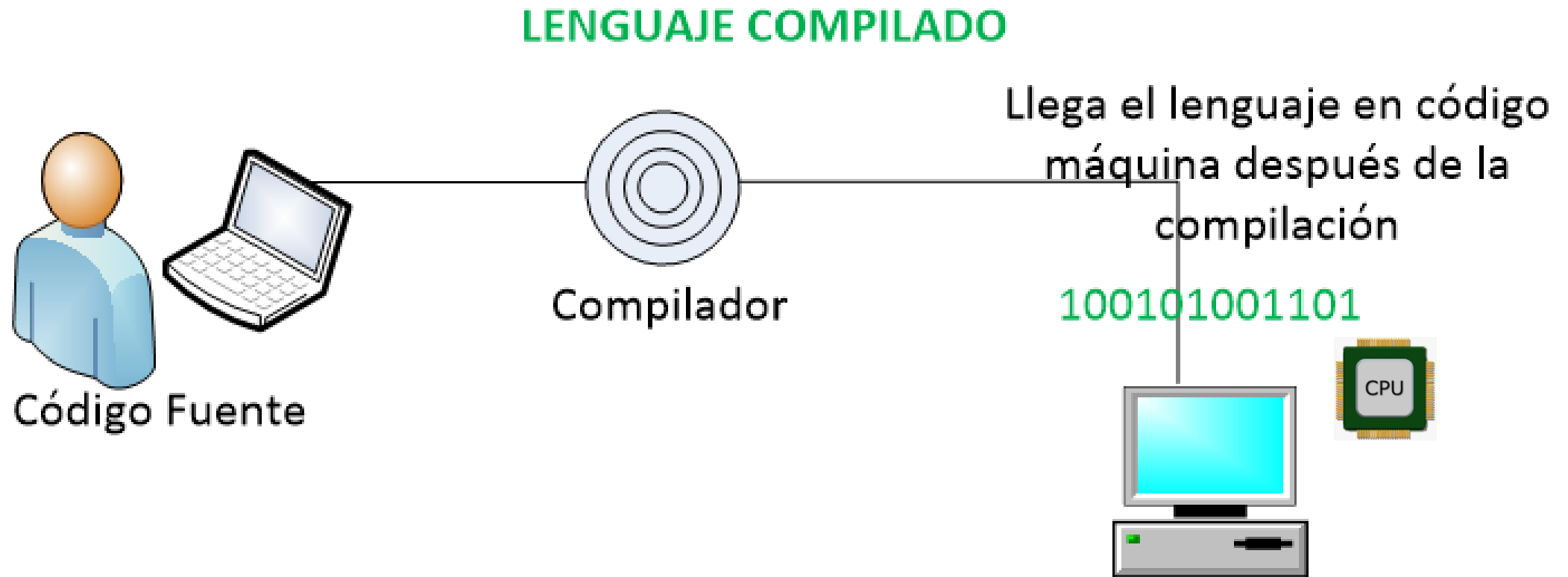
COMPILADOR:

Traductor de
lenguaje “alto nivel”
a lenguaje máquina

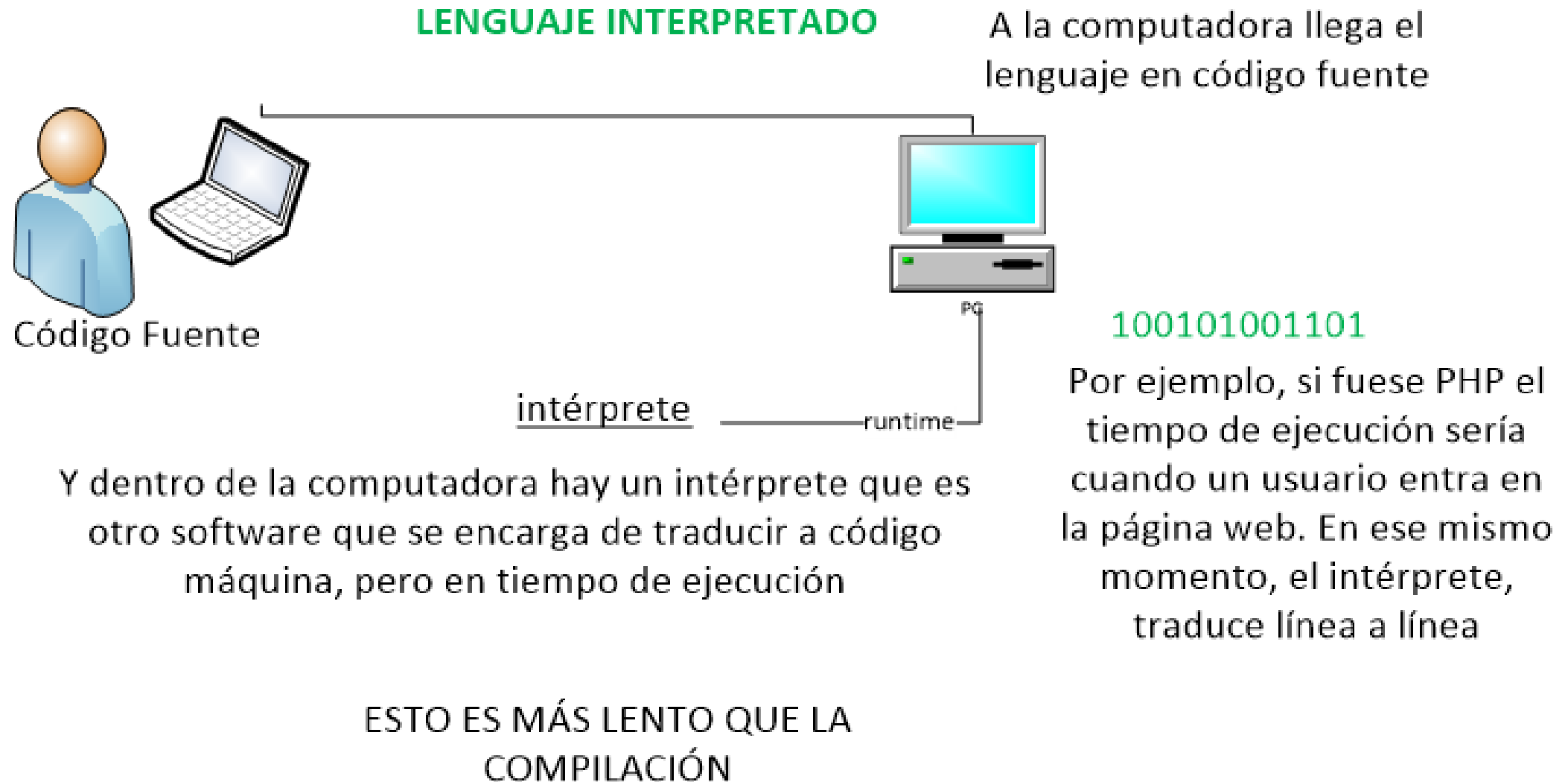


LOS PRIMEROS LENGUAJES
EN USAR ESTOS COMPILADORES
FUERON FORTRAN Y COBOL

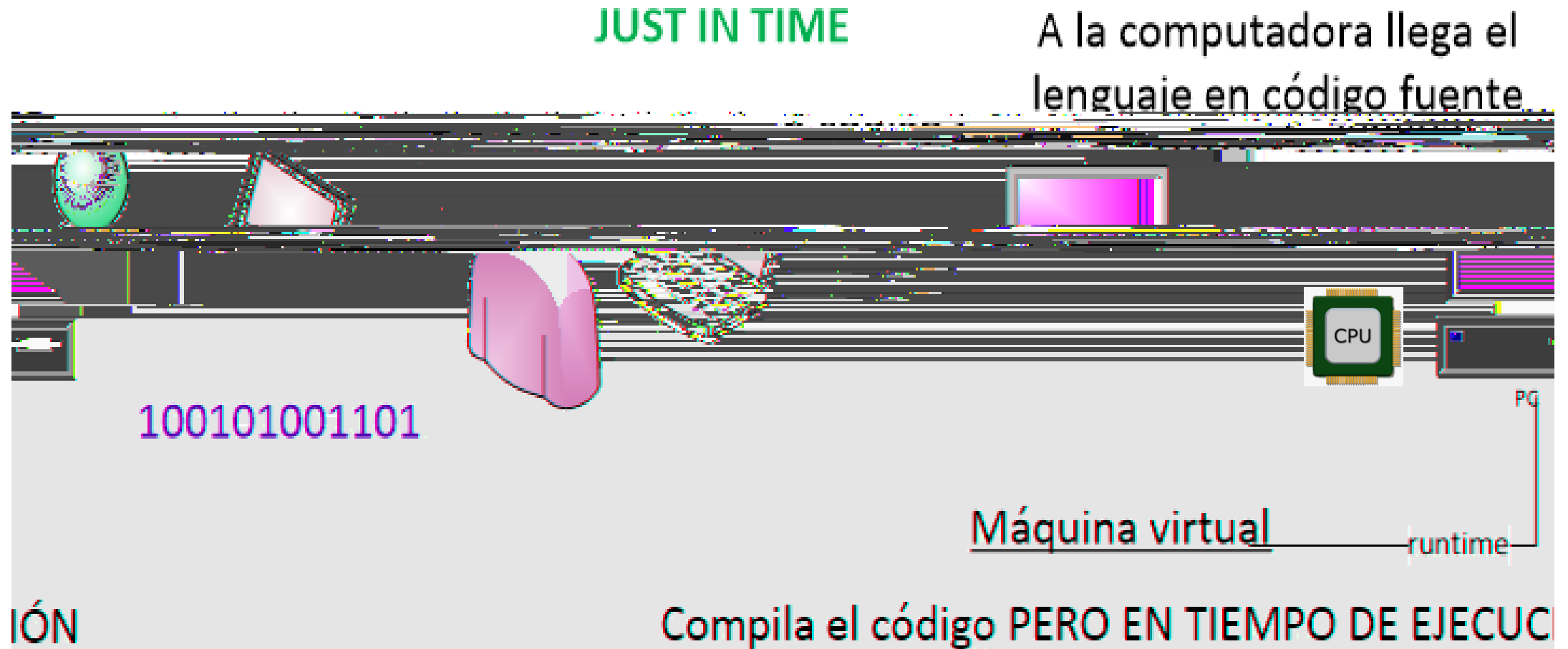
5.3.- Codificación. ¿QUÉ ES UN LENGUAJE COMPILADO “ORIGINAL”?



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

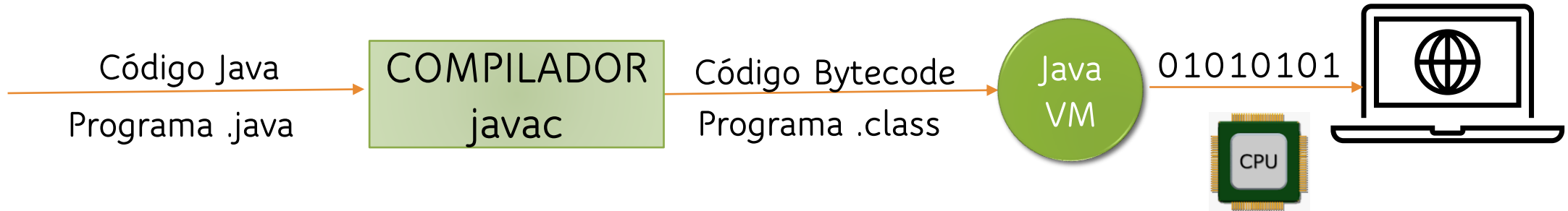


5.3.- Codificación. JUST IN TIME (JIT) Mezcla entre compilado e interpretado



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 de ser ejecutados en la computadora	INTERMEDIOS Se compilan + Se interpretan	PHP, JavaScript, Python Llegan en código fuente a la computadora, donde 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.

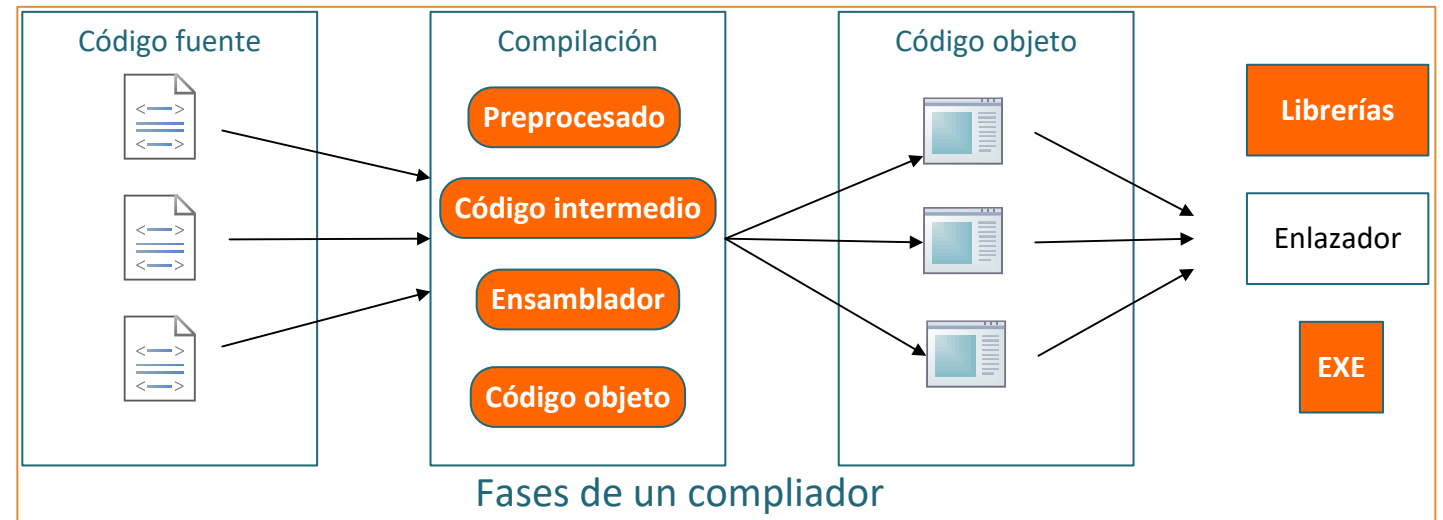
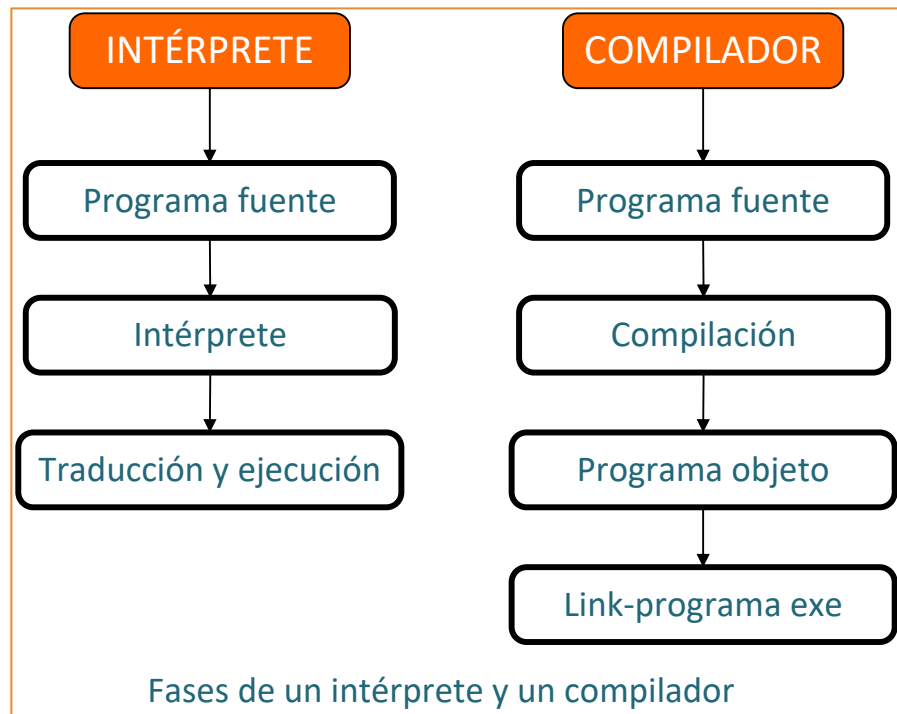
Código Ejecutable: Es el código binario resultante de enlazar los archivos de código objeto con ciertas rutinas y bibliotecas necesarias. El sistema operativo será el encargado de cargar el código ejecutable en memoria RAM y proceder a ejecutarlo. Es el que entiende la computadora

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.



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.

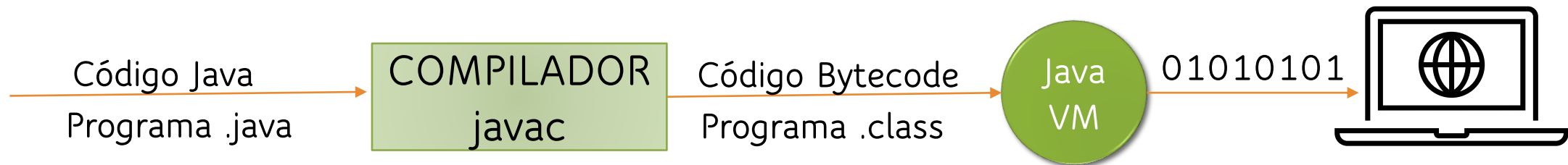
Comunicarse con el sistema donde se instala la aplicación (huésped), para el control de los dispositivos hardware implicados en los procesos

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 cambios 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.?



5.5.2.- ¿Qué es un entorno de ejecución?

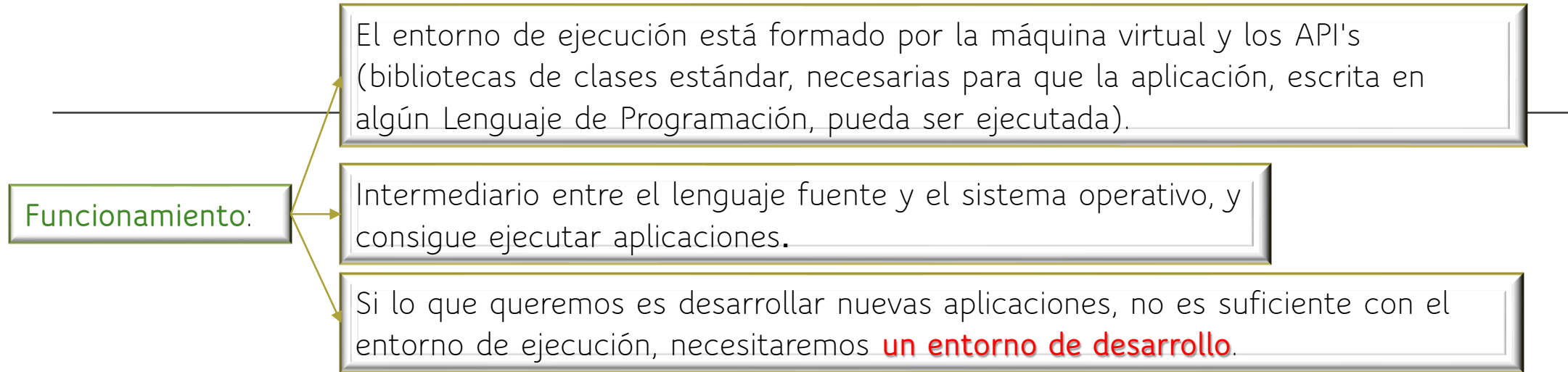
«Entorno de ejecución» es uno de esos muchos conceptos de la informática que pertenece a la categoría de «difíciles de definir con precisión pero fáciles de entender en la práctica».

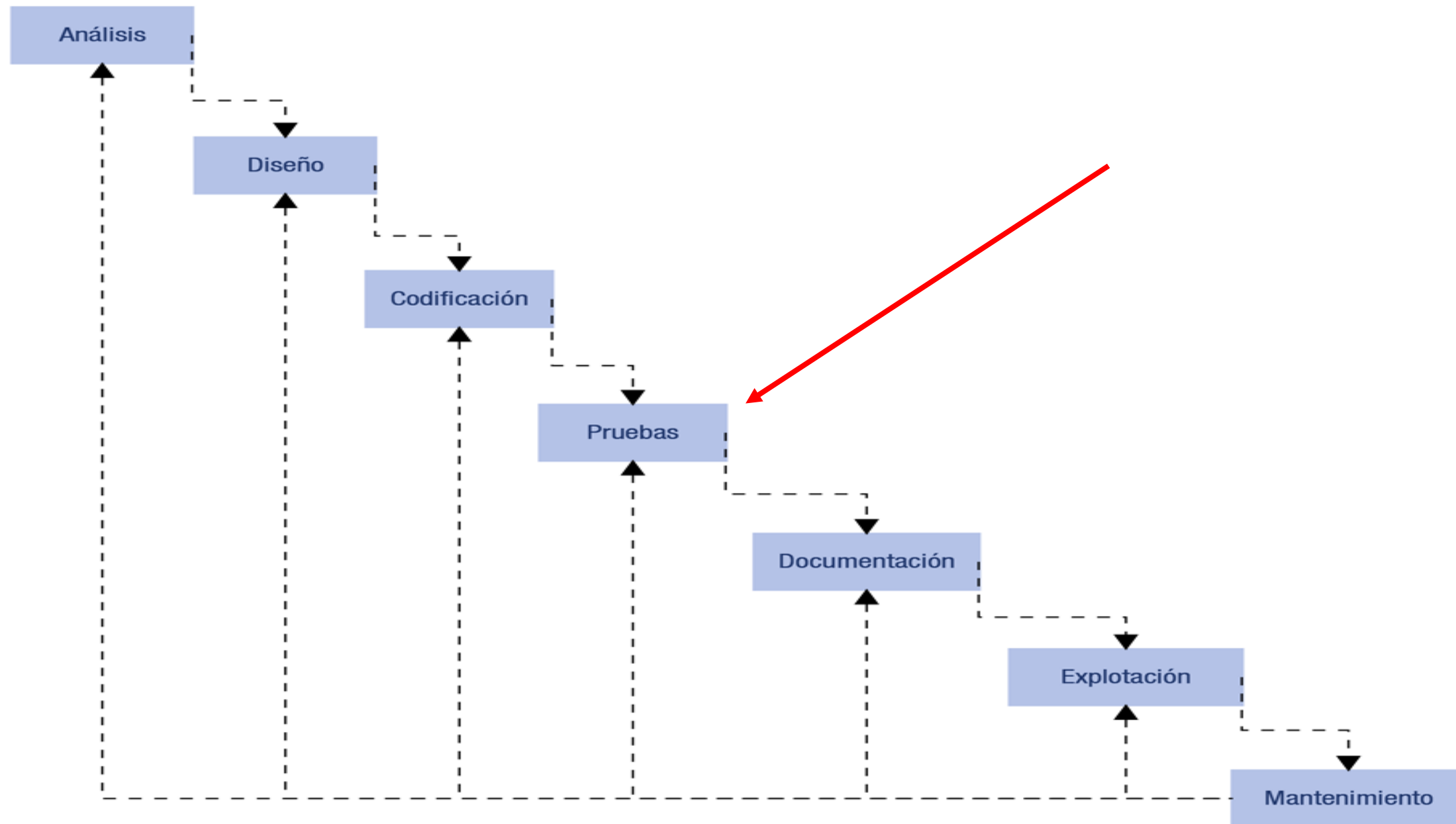
Si queremos ser prácticos, basta con que entendamos que un lenguaje de programación sirve de poco si no hay «algo» capaz de transformar el código en un conjunto de instrucciones que ponga a trabajar a la computadora como el programador haya dispuesto en dicho código.

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.

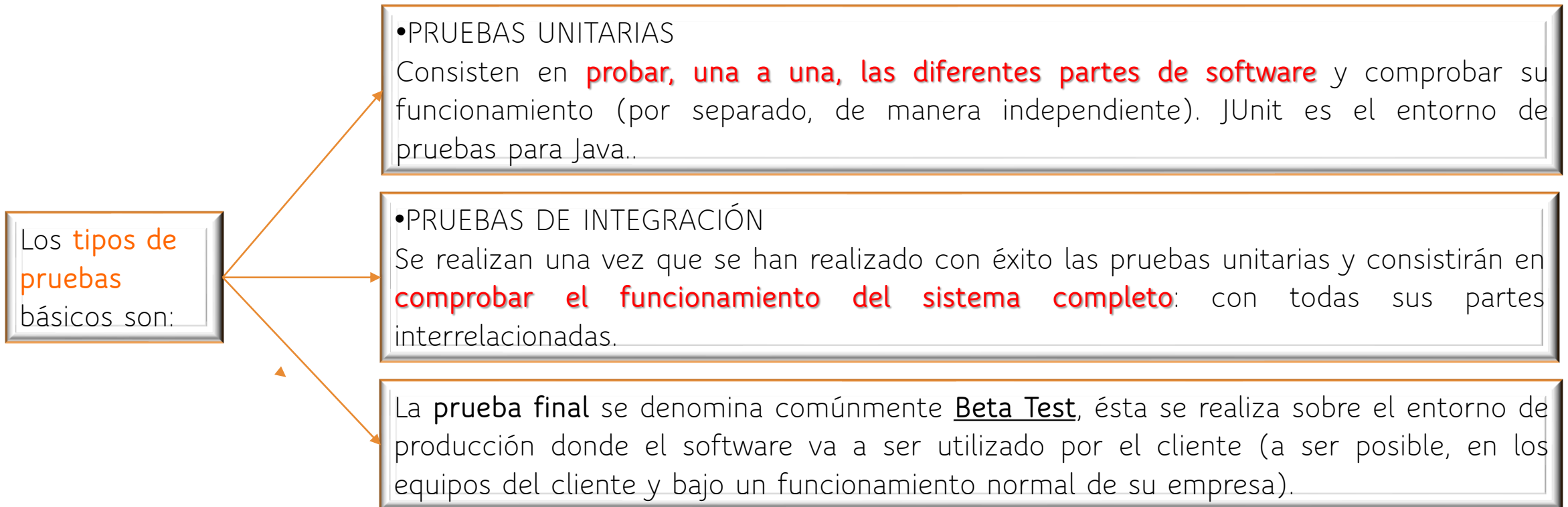
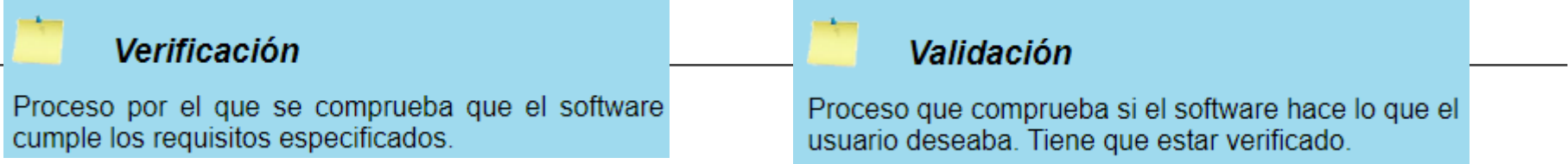




5.6.- Pruebas.

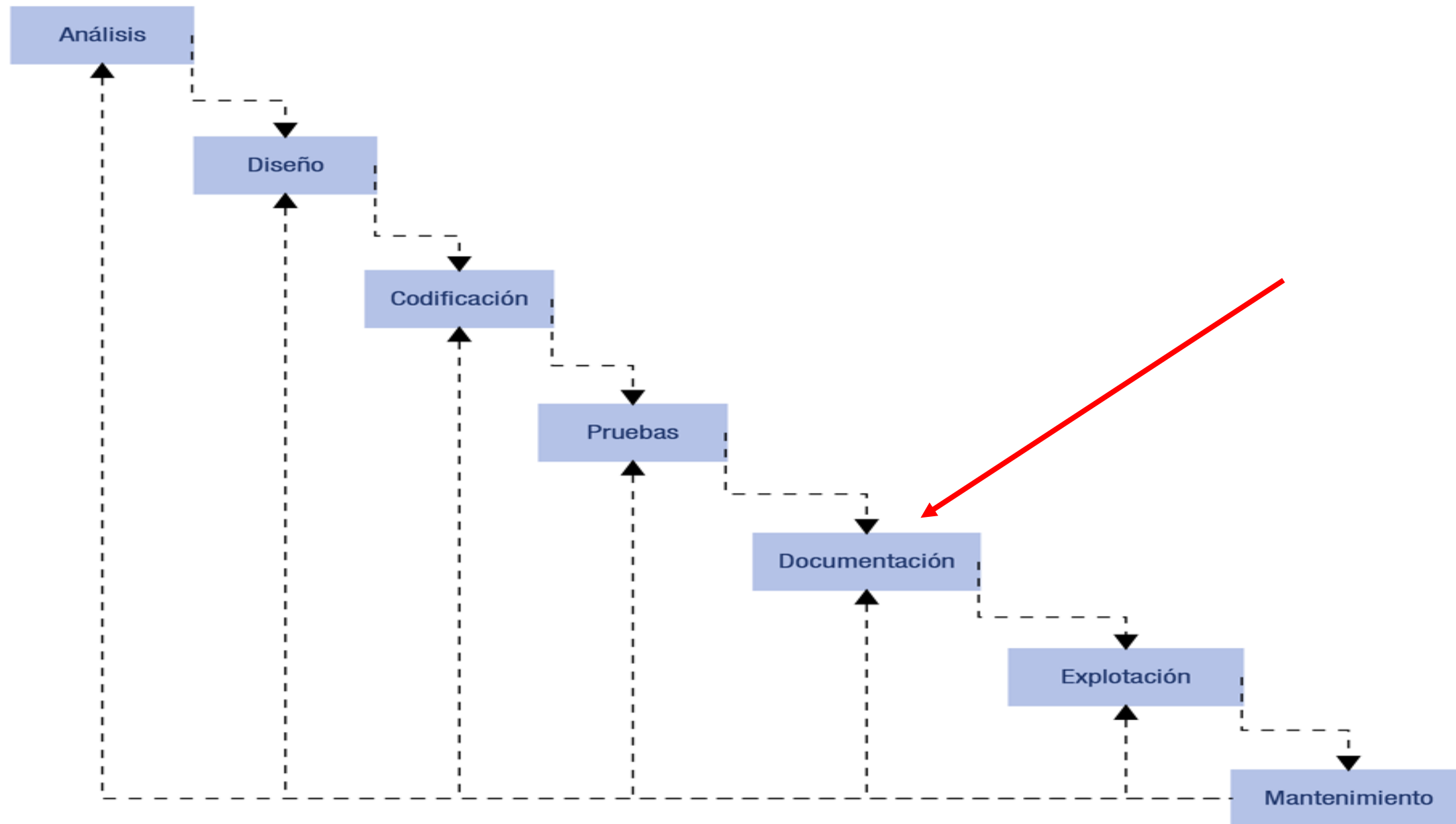
Una vez obtenido el software, la siguiente fase del ciclo de vida es la realización de pruebas.

La realización de pruebas es imprescindible para asegurar la [validación](#) y [verificación](#) del software construido.



5.6.- Pruebas.

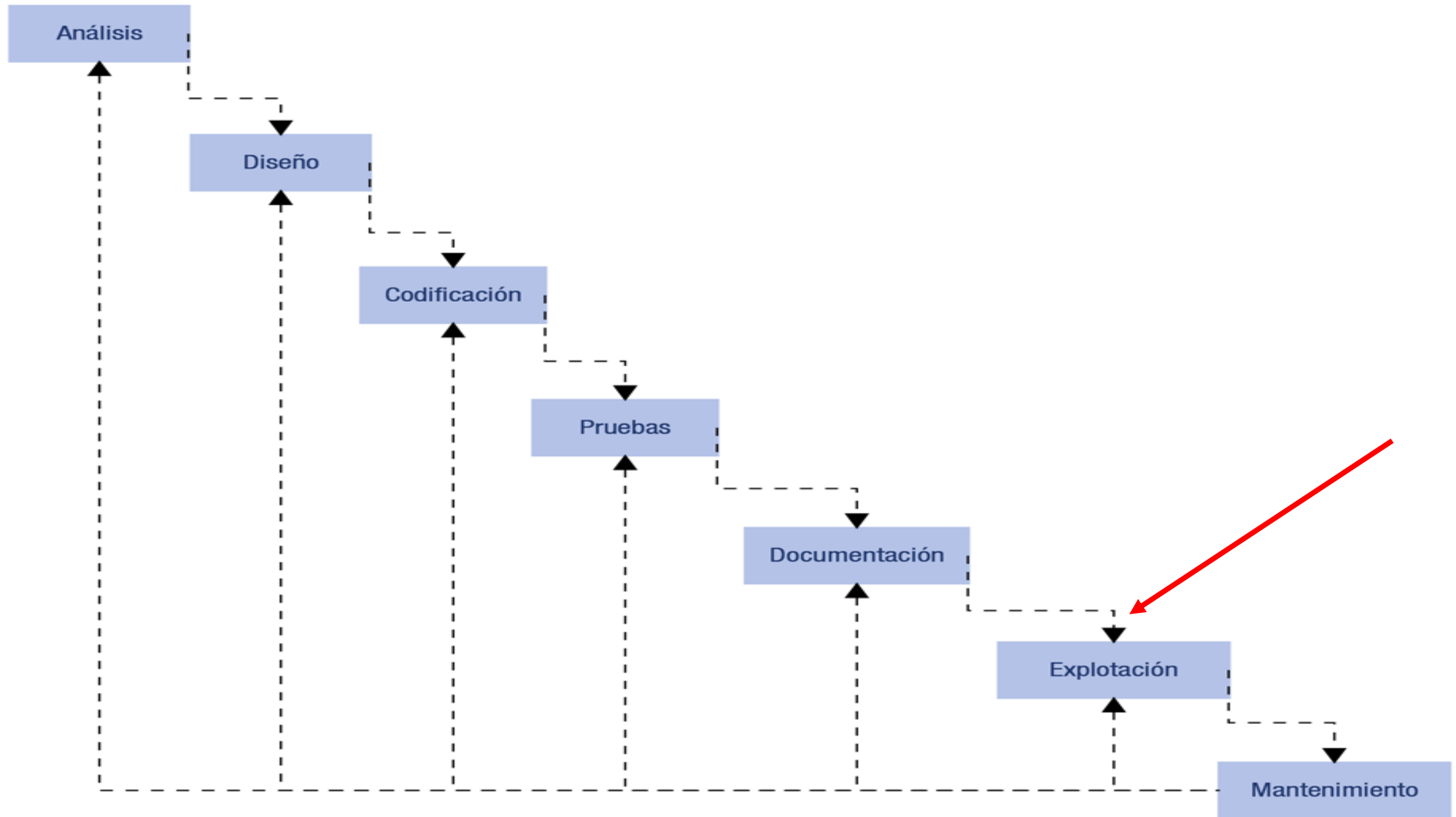
Examen de la unidad	Pruebas de integración
Se lleva a cabo en la fase inicial de prueba y luego se puede realizar en cualquier momento	Debe realizarse después de la prueba unitaria y antes de la prueba del sistema.
Prueba el componente único de todo el sistema, es decir, prueba una unidad de forma aislada.	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



5.7.- Documentación

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

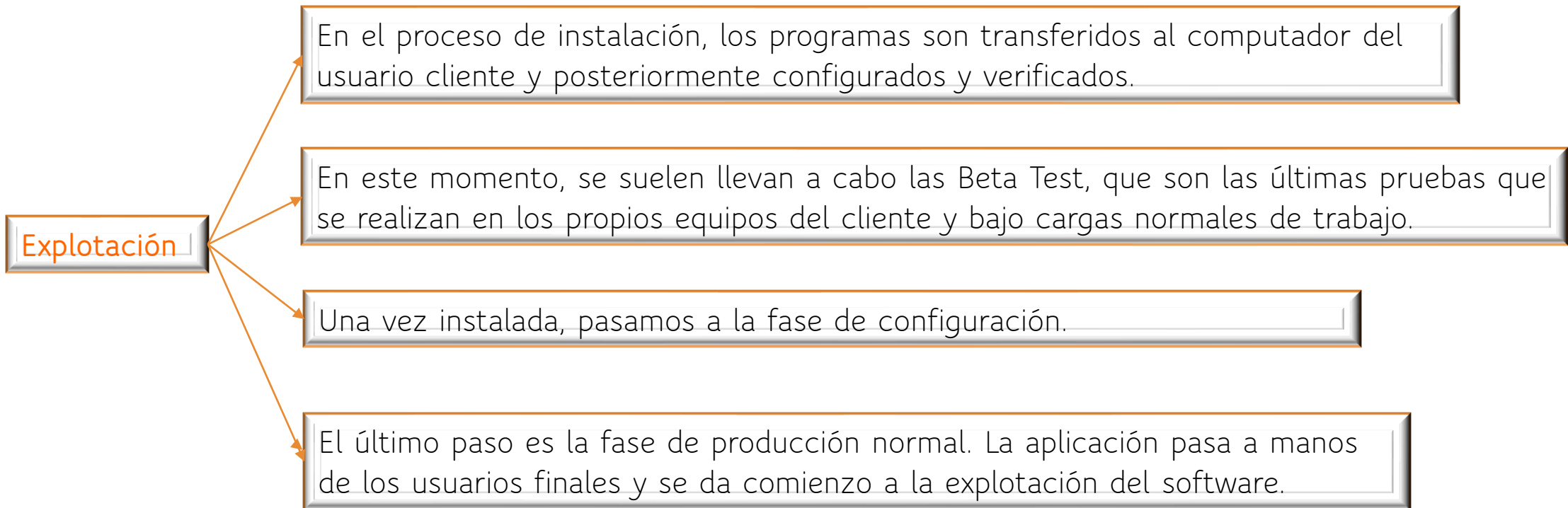
Documentos a elaborar en el proceso de desarrollo de software			
	GUÍA TÉCNICA	GUÍA DE USO	GUÍA DE INSTALACIÓN
Quedan reflejados:	<ul style="list-style-type: none">•El diseño de la aplicación.•La codificación de los programas.•Las pruebas realizadas.	<ul style="list-style-type: none">•Descripción de 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: <ul style="list-style-type: none">•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.

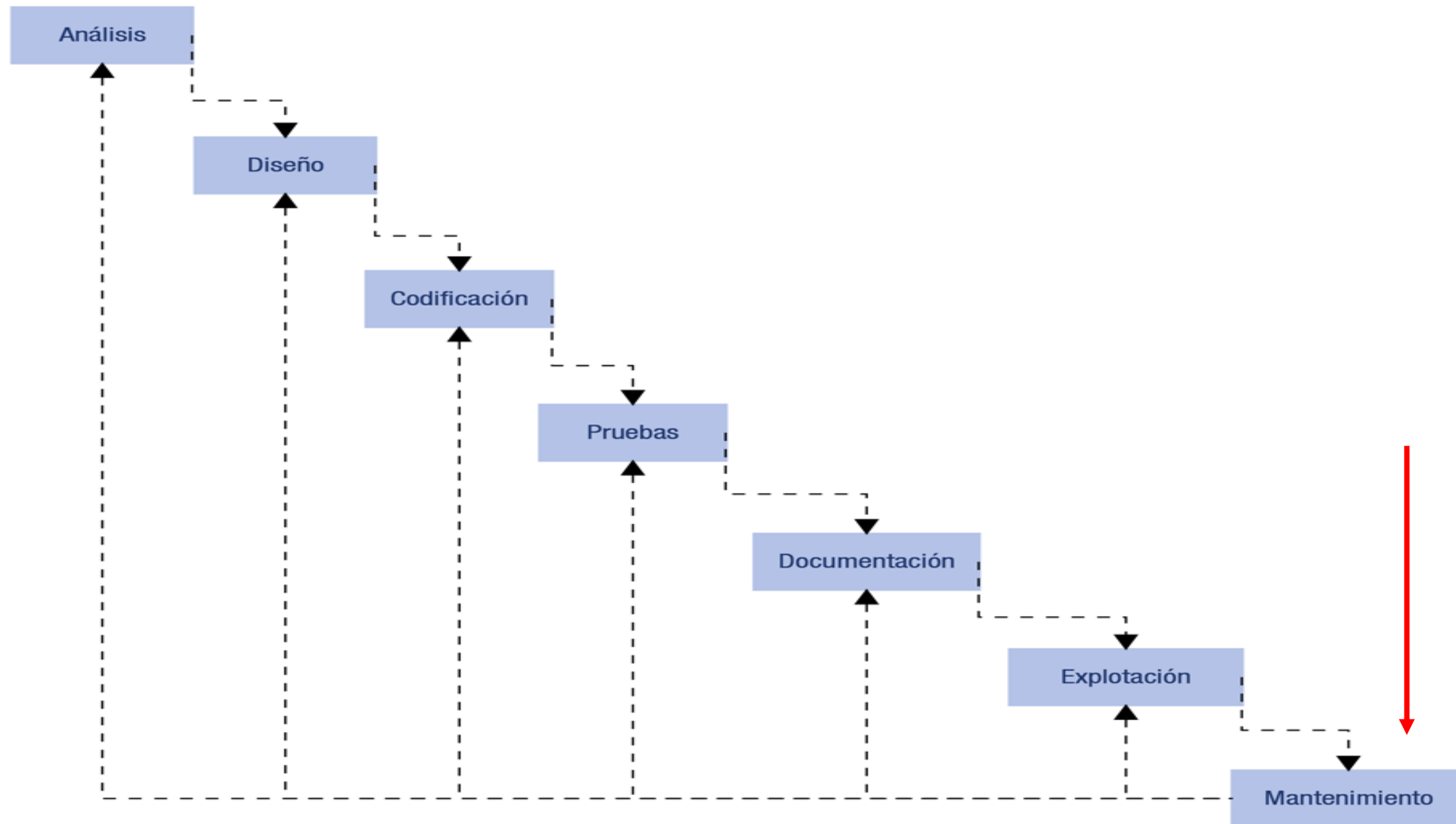


5.8.- Explotación.

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.





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.

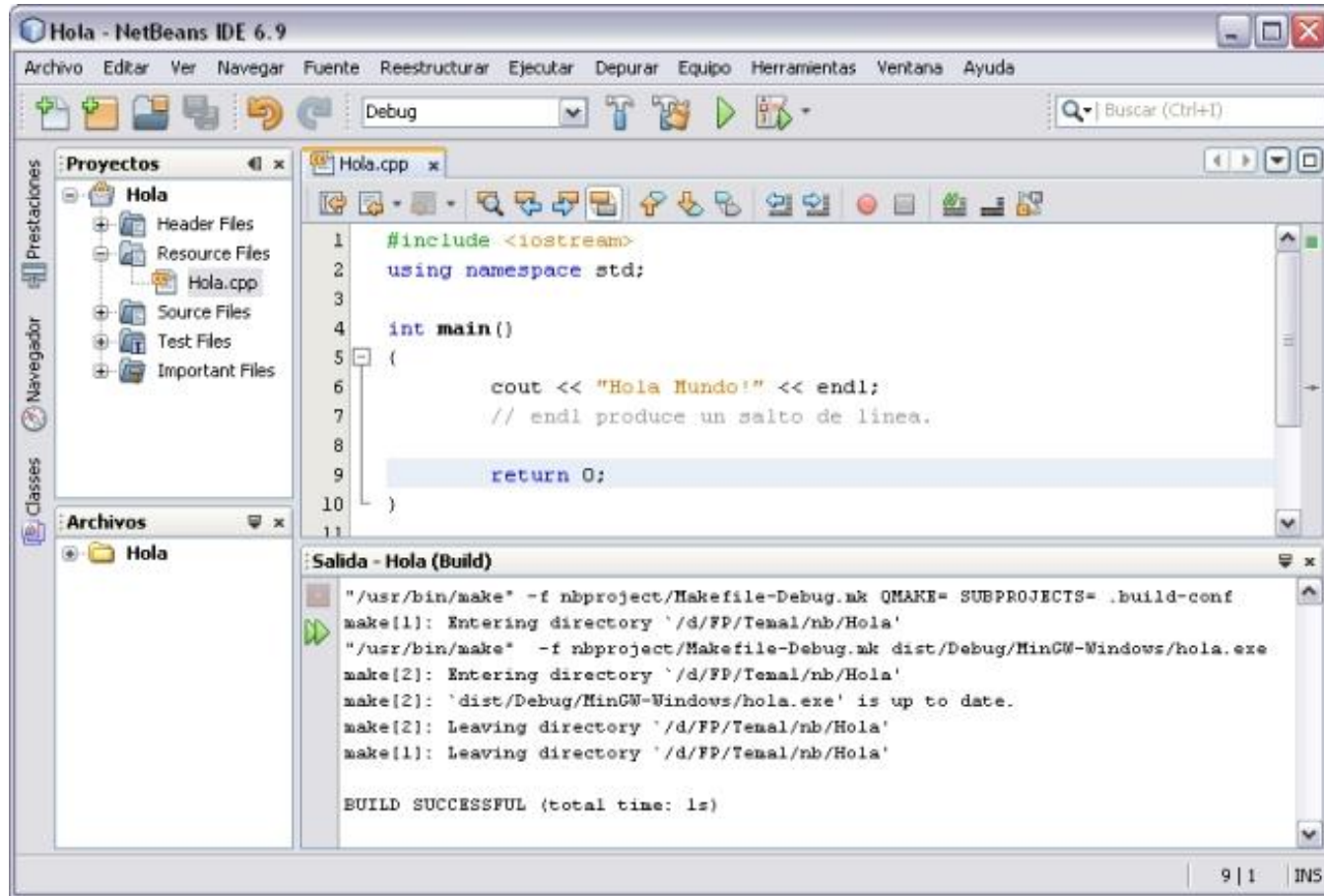
- En las décadas de utilización de la **tarjeta perforada** como sistema de almacenamiento el concepto de Entorno de Desarrollo Integrado sencillamente no tenía sentido.
-
- Los programas estaban escritos con diagramas de flujo y entraban al sistema a través de las tarjetas perforadas. Posteriormente, 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 Studio.	Basic, C/C++, C#.	Windows.
FlashBuilder.	ActionScript.	Windows, Mac OS X.
C++ Builder.	C/C++.	Windows.
Turbo 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,