

---

# UT1. Introducción al desarrollo del software

---

Entornos de desarrollo

# ÍNDICE DE CONTENIDOS

---

- Hardware.
- Software.
- Lenguajes de programación.
- Codificación.
- Ingeniería del software.
- Desarrollo software.

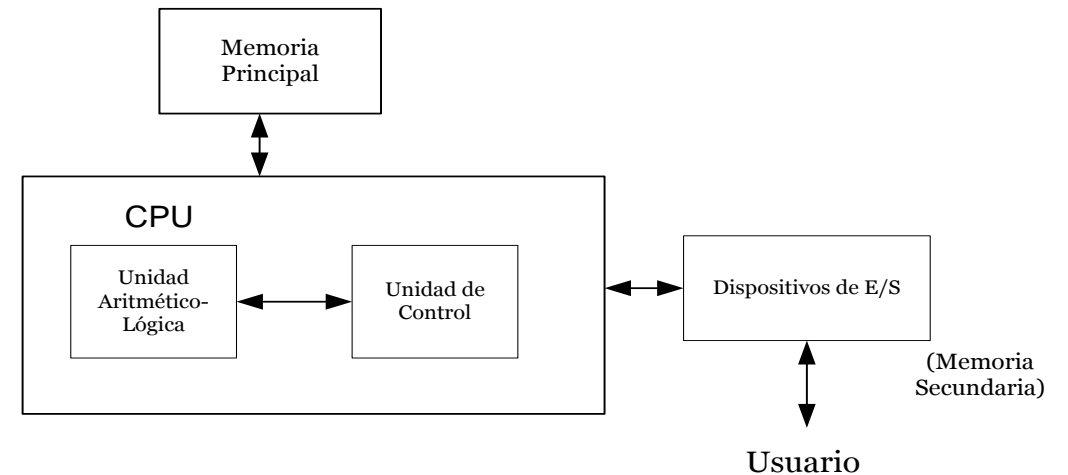
# Hardware.

---

El hardware es la parte **tangible** de un sistema informático, equipamiento físico.

Generalmente, un ordenador se basa en la arquitectura de Bon Newmann, la cuál divide un ordenador en las siguientes partes:

- Unidad de Control.
- Unidad Aritmético-Lógica.
- Memoria principal
- Dispositivos de Entrada/Salida



# Hardware.

---

**Unidad de control** - es el componente básico de un ordenador ya que controla la ejecución de las operaciones y dirige el funcionamiento de todos los demás componentes de tal forma que el trabajo conjunto de todos conduzca a la consecución de las tareas específicas programadas en el sistema.

**Unidad Aritmético-Lógica** – es la parte encargada de realizar las operaciones aritméticas (suma, resta, ....) y las operaciones lógicas (comparación, ....). También realiza otras funciones más complejas (raíces, funciones trigonométricas, ..).

Al conjunto Unidad de Control y Unidad Aritmético-Lógica se le conoce como **CPU (Central Process Unit – Unidad Central de proceso)**. Así tenemos los procesadores Intel, Motorola, AMD.

**Memoria principal** – es la memoria de almacenamiento interno. Opera a gran velocidad. Aquí se ubican los programas: las instrucciones junto con los datos sobre los que actúan.

La memoria principal está formada por una serie de *celdas* o posiciones identificadas por una *dirección* de memoria.

Por otro lado está la *memoria secundaria* o *memoria externa* que permite resolver los problemas de volatilidad y capacidad de la memoria principal (discos duros, CD, ...).

**Dispositivos de Entrada / Salida** – son los que facilitan la interacción del usuario (el mundo exterior) con la máquina (teclado, monitor, impresora, ....).

# Hardware.

El ciclo de una instrucción es:

- 1) La Unidad de Control indica a la Memoria Principal la siguiente instrucción que quiere ejecutar.
- 2) La U.C. recibe la instrucción, procediendo a su análisis para determinar los operandos sobre los que actúa y su localización.
- 3) Bajo las directrices de la U.C. la U.A.L realiza la operación y se guarda el resultado en su destino.
- 4) Una vez ejecutada la instrucción se incrementa el contador de programa y se pasa a realizar la ejecución de la siguiente instrucción.



# Hardware.

---

Los ordenadores digitales utilizan internamente el **código binario**. La mínima información manipulable es un dígito binario, 0 y 1, el **bit**. Tanto los datos como las instrucciones que ejecuta la CPU han de expresarse en este código para poder ser almacenados en memoria. Al conjunto de instrucciones codificadas en binario se le conoce como **lenguaje máquina** y es el lenguaje más básico y el único que entiende el ordenador.

1 Byte = 8 bits

# Software.

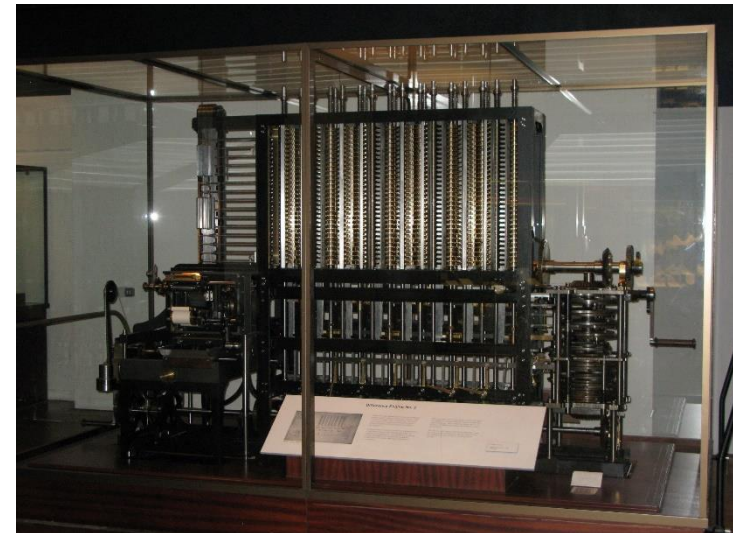
---

El software es la parte **intangible** de un sistema informático, equipamiento lógico.

El software es el encargado de comunicarse con el hardware, es decir, se encarga de traducir las órdenes dadas por el usuario en órdenes comprensibles por el hardware.

➤ El concepto de software fue utilizado por primera vez por Charles Babbage (1822), máquina diferencial.

➤ Ada Lovelace primera mujer programadora, 1843.



# Software.

---

- Alan Turing (1912-1954) profundizó en el concepto de software y destacó como criptógrafo gracias a la creación de la máquina que lleva su mismo nombre.

Actualmente es considerado como el padre de la computación, diseñó el sistema de programación del primer ordenador comercial y sentó las bases de la [inteligencia artificial](#).

- Por último, el término software no se concibe como en la actualidad hasta 1958, gracias al artículo American Mathematical Monthly de John Wilder Turkey,



# Software: Características.

---

- 1) El software es **lógico**, intangible.
- 2) El software **se desarrolla**, no se fabrica.
- 3) El software **no se estropea**, una copia suya da lugar a un clon con las mismas características.
- 4) Se puede construir **a medida**, puede construirse a medida o enlatado.

# Software: Tipos.

---

Existen tres tipos de software: software de sistema, de programación y de aplicaciones.

- 1) El **software de sistema** es el software base que ha de estar instalado y configurado en nuestro ordenador para que las aplicaciones puedan ejecutarse y funcionar. Ej: Windows, Linux,...
- 2) El **software de programación** es el conjunto de herramientas que nos permiten desarrollar programas informáticos.
- 3) El **software de aplicación** son aquellos programas que tienen una finalidad más o menos concreta.

# TIPOS DE SOFTWARE

## De Sistemas

Objetivo

Librar al usuario de los detalles del hardware que se usa y de su gestión.  
Proporciona una interfaz de alto nivel, cómoda para el usuario.

Incluye

Sistemas Operativos  
Controladores de dispositivos  
Utilidades

## De Programación

Objetivo

Proporcionar herramientas al usuario para el desarrollo de programas informáticos.

Incluye

Editores de texto  
Compiladores  
Intérpretes  
Enlazadores  
Depuradores  
Entornos Integrados de Desarrollo (IDE)

## De Aplicaciones

Objetivo

Permitir al usuario realizar una o varias tareas específicas.

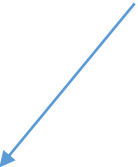
Incluye


Aplicaciones Ofimáticas  
Software educativo  
Bases de datos  
Videojuegos  
Software de diseño asistido (CAD)


# Lenguajes de Programación.

---

- Todos los programas están escritos en algún lenguaje de programación.
- Los lenguajes de programación permiten comunicarnos con el hardware del ordenador.
- Todos los lenguajes de programación tienen una **sintaxis** y un **conjunto de normas y palabras reservadas**.

- 
- Ensamblador: MOV, ADD, END, SIZE,...
  - Java: break, abstract, continue,...

- 
- caseSensitive
  - //Comentario
  - tipado

- 
- indentado
  - #include terminan en .h

# Lenguajes de Programación. Tipos

---

## 1) Lenguaje máquina:

- Son instrucciones **complejas** compuestas de combinaciones de unos y ceros.
- **No necesita ser traducido**, es el único lenguaje que entiende el ordenador.
- Fue el primer lenguaje utilizado.
- **Difiere de cada computador**, es decir, el conjunto de instrucciones no es portable de un equipo a otro.
- Salvo excepciones, nadie programa en este lenguaje.

# Lenguajes de Programación. Tipos

---

## 2) Lenguaje de nivel medio o ensamblador:

- Sustituyó al lenguaje máquina para **facilitar** la labor de programación.
- Utiliza **mnemotécnicos**, es decir, instrucciones complejas que en lugar de ceros y de unos.
- Necesita ser traducido al lenguaje máquina para poder ejecutarse.
- Trabaja con los **registros** del procesador y con direcciones físicas.
- Es difícil de comprender y programar.

# Lenguajes de Programación. Tipos

---

## 3) Lenguaje de alto nivel:

- La mayoría de los lenguajes actuales pertenecen a esta categoría.
- Utilizan **sentencias** y **órdenes** derivadas de otros idiomas, inglés.
- Fortran fue el primer lenguaje de alto nivel 1954.
- Tienen **librerías** y **funciones** predefinidas que facilitan la tarea del programador.
- Suelen trabajar con la **abstracción** y la **orientación a objetos**, haciendo más fácil la reutilización y encapsulación de los componentes.

# Lenguajes de Programación. Tipos

---

Según sean ejecutados, podemos clasificarlos como:

a) Lenguajes compilados:

- Necesitan de un “compilador” para pasar el código fuente a máquina.
- Son los lenguajes con *mejor tiempo de ejecución*.
- Necesitan de un *linker* para unir el código objeto del programa con el de las librerías.

b) Lenguajes interpretados:

- Requieren de un **intérprete** (programa) para interpretar y ejecutar las instrucciones.
- Solamente se traducen aquellas instrucciones que se van ejecutando.

c) Lenguajes virtuales:

- Son los más portables, ya que generan código intermedio o bytecode.
- Su código puede ser interpretado por una máquina virtual instalada en un equipo.
- Son los lenguajes con *mayor tiempo de ejecución*, pero con mayor versatilidad.



# Lenguajes de Programación. Tipos

---

**Ejercicio:** Haz un breve resumen de los siguientes lenguajes de programación indicando su uso principal en nuestro día a día y sus principales características:

- a) Java
- b) Python
- c) C/C++
- d) JavaScript
- e) PHP

# Preguntas:

---

- 1) Lugar donde se llevan a cabo operaciones como suma o resta.  
a) UCP   b) Unidad Aritmética   c) Unidad Aritmética y lógica
- 2) Se clasifica Dentro del software de aplicación.  
a) Visual Basic   b) Microsoft Word   c) Prolog
- 3) Se definen por un conjunto de símbolos a utilizar, bajo reglas de sintaxis y semántica.  
a) Programa   b) Lenguaje de programación   c) Sistema Operativo
- 4) Su función es administrar sus recursos además de fungir (ejecutar) como un intérprete a las acciones de los usuarios.  
a) Programa   b) Lenguaje de programación   c) Sistema Operativo

# Preguntas:

---

- 5) En este tipo de lenguaje las instrucciones se escriben en un lenguaje natural al ser humano, generalmente en el idioma inglés.
- a) Máquina    b) Bajo nivel    c) Alto nivel
- 6) En este tipo de lenguaje las instrucciones se escriben en códigos nemotécnicos
- a) Máquina    b) Bajo nivel    c) Alto nivel

# Preguntas:

---

Complemente las frases siguientes.

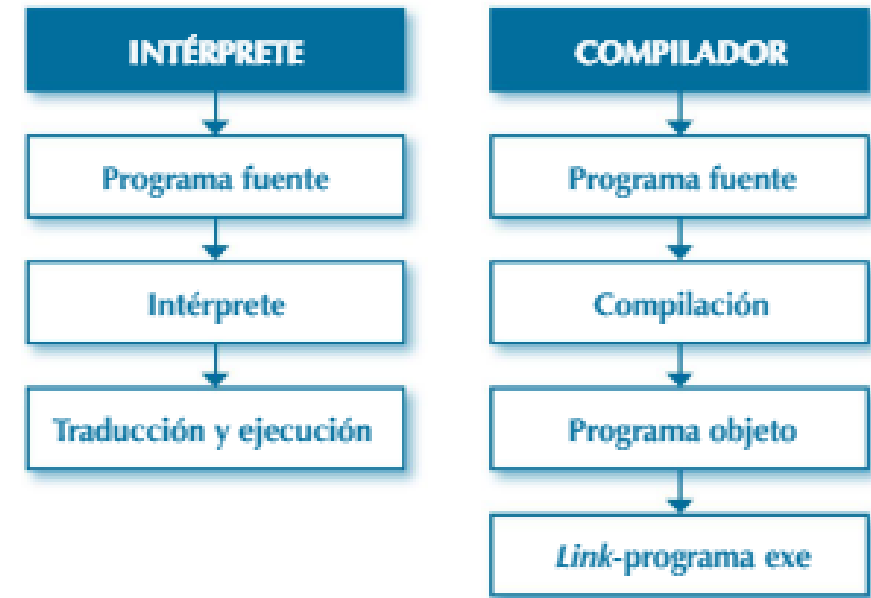
- 1) La programación\_\_\_\_\_se basa en el uso exclusivo de órdenes de control secuenciales, alternativas e iteración para el control del flujo de ejecución del código.
- 2) Los datos \_\_\_\_\_ ocupan un grupo de casillas de memoria
- 3) Los datos \_\_\_\_\_ ocupan una casilla simple de memoria
- 4) El Lenguaje \_\_\_\_\_ es aquel cuyas instrucciones son entendibles directamente por la computadora, ya que se expresan en términos de dígito binario.

# Codificación

Los **traductores** son programas cuya finalidad es traducir lenguajes de alto nivel a lenguajes de bajo nivel.

Un **intérprete** traduce el código fuente línea a línea, tiene que estar en memoria para poder realizar este proceso.

Un **compilador** traduce el código fuente a código máquina. El compilador está en la máquina de desarrollo. El código generado solo funcionará en una máquina con un hardware y software determinado. Si se cambian de hardware o software, habría que recompilarlo



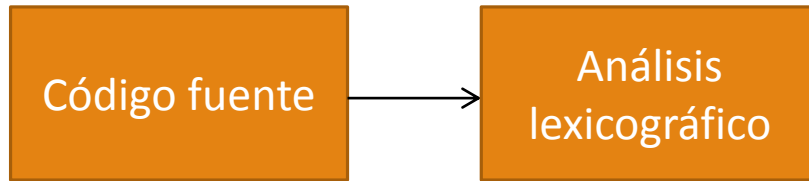
# Codificación: Fases de un compilador

- 1) **Edición:** aquí el programa fuente debe ser tecleado a la computadora a través de un **EDITOR**, de tal manera que se convierta en un archivo de programa almacenado → **PROGRAMA FUENTE (CÓDIGO FUENTE)**
- 2) **Compilación:** el **COMPILADOR** se encarga de traducir la edición del programa fuente a lenguaje máquina. De ser necesario la compilación se repite hasta no producir errores, dando como resultado el programa objeto → **PROGRAMA OBJETO (CÓDIGO OBJETO Ó BYTECODE)**
- 3) **Enlace:** el sistema operativo es instruido a tomar el programa objeto y ligarlo con las librerías del programa compilador, dando como resultado un programa ejecutable → **PROGRAMA EJECUTABLE (CÓDIGO EJECUTABLE)**
- 4) **Ejecución:** Una vez que contamos con el ejecutable, este puede “correrse” en el sistema operativo obteniendo por salida los resultados del programa.



# Codificación: Compilación

---



Se busca dentro del código fuente palabras reservadas, operaciones, caracteres de puntuación y se agrupan en cadenas denominadas **lexemas**.

# Codificación: Compilación

---

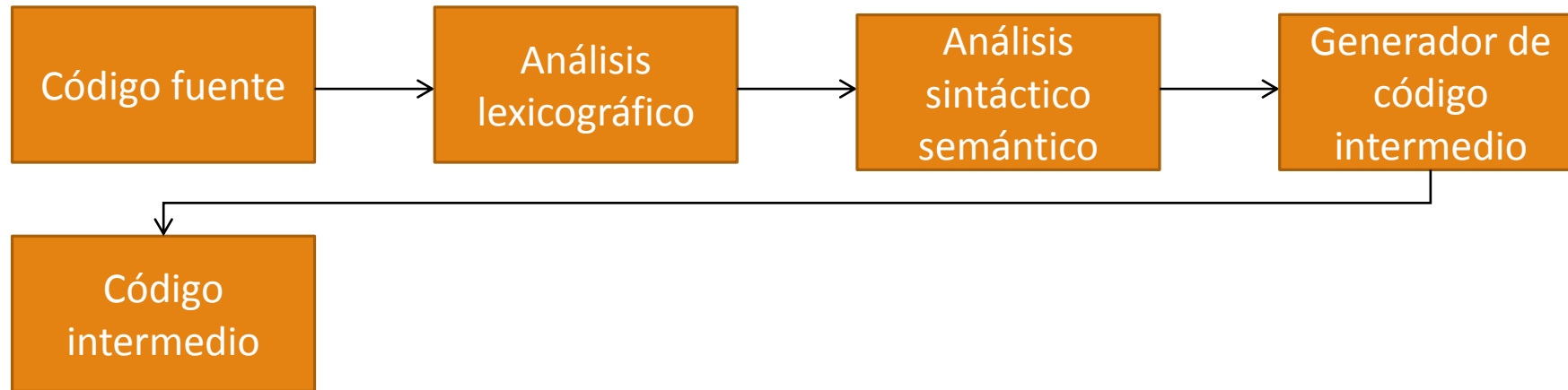


Agrupar los resultados obtenidos en la fase anterior como frases gramaticales. Tras el análisis sintáctico, verifica la coherencia de las frases gramaticales, es decir: si los tipos de datos son correctos, si los *arrays* tienen el tamaño y tipo adecuados, y así consecutivamente con todas las reglas semánticas de nuestro lenguaje.



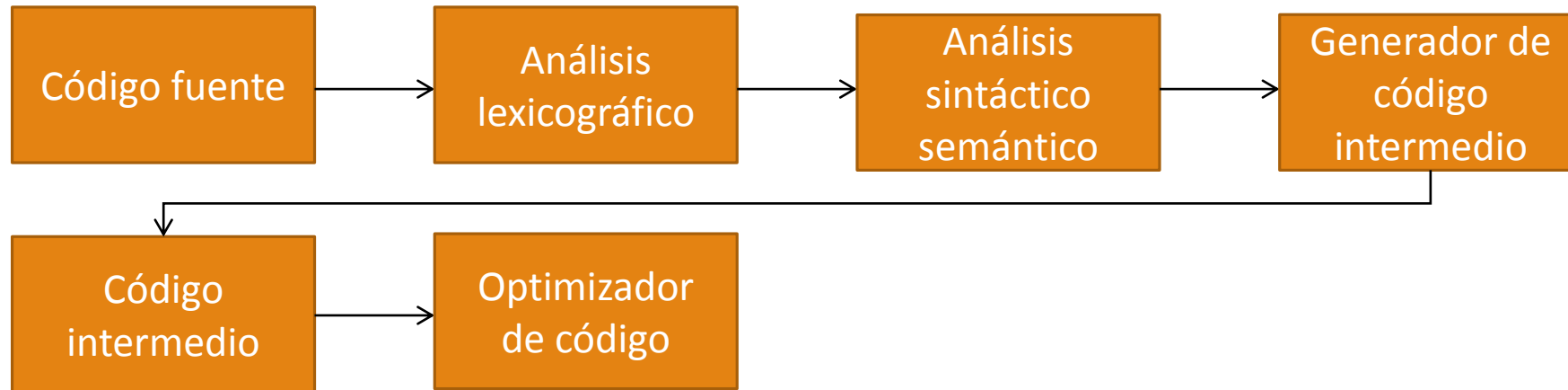
# Codificación: Compilación

---



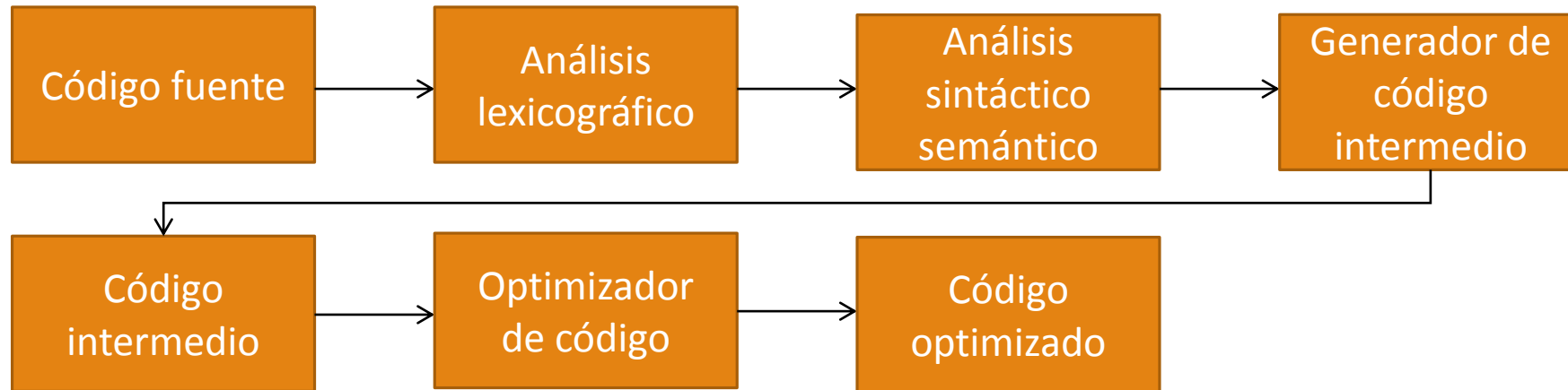
Tras el análisis anterior, se genera una representación intermedia en pseudoensamblador, facilita las tareas de traducción restante.

# Codificación: Compilación



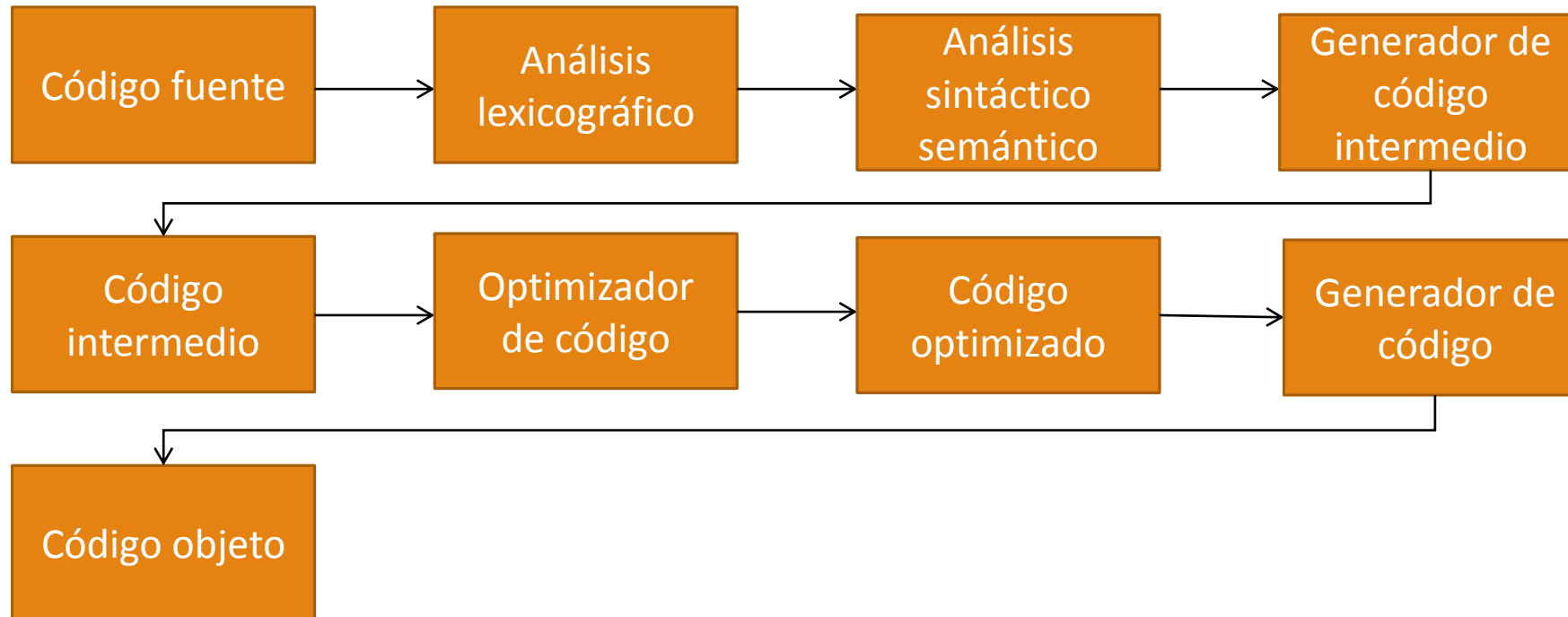
Optimiza el código anterior para que sea más eficiente en la máquina. Ejemplo: Crear variables de intercambio de valores en decisiones.

# Codificación: Compilación



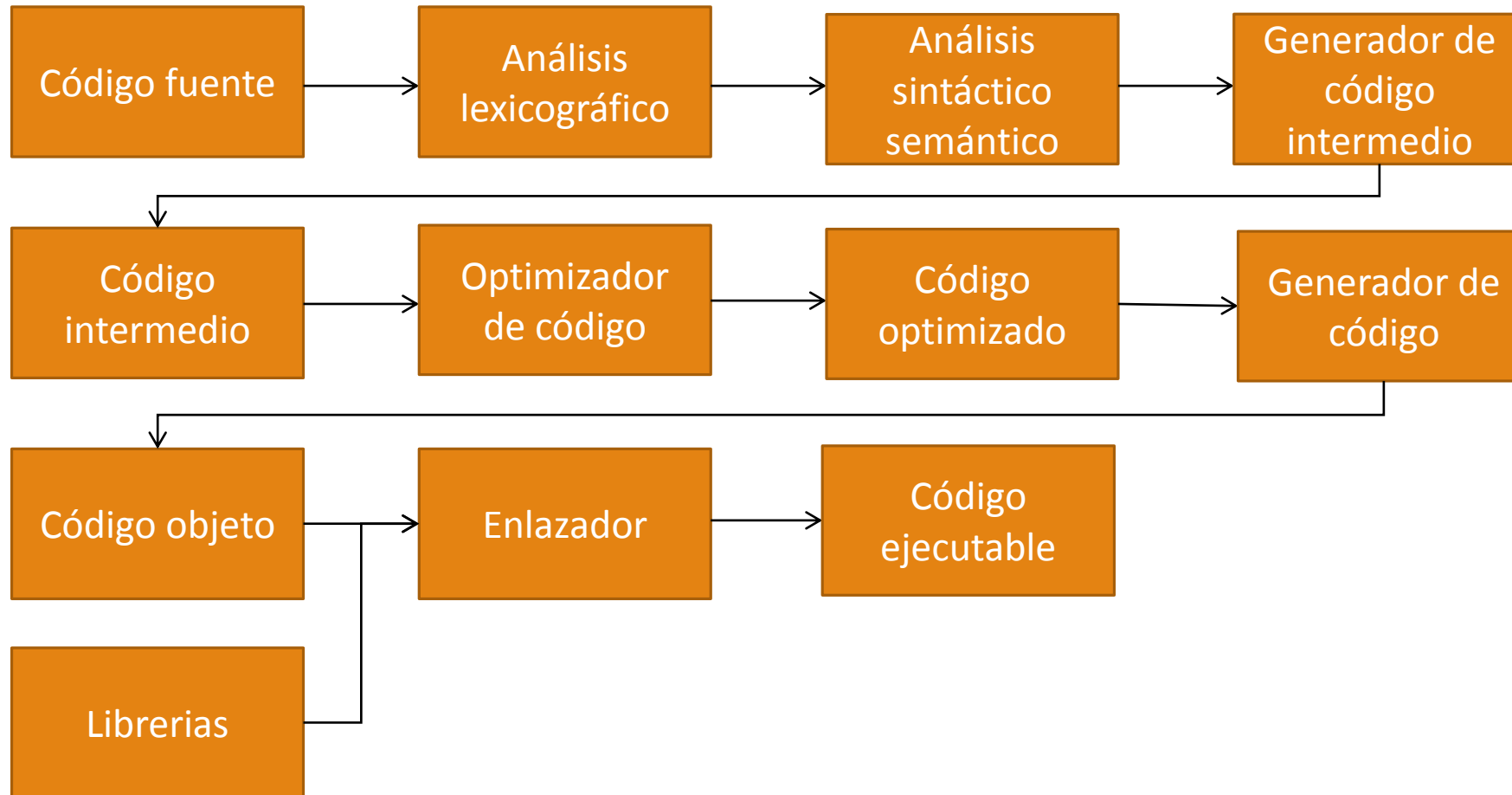
Tras esta optimización de los recursos, genera un código optimizado.

# Codificación: Compilación



Genera el código objeto con posiciones de memoria sin establecer, en otras palabras, no sabemos en que zona de la memoria se ejecutará nuestro programa.

# Codificación: Compilación



# Codificación: Depuración

---

Aunque un programa ejecutable en primera instancia no marque errores, es fundamental aplicarle una serie de **pruebas** (otra fase, en la que: Es el proceso de ejecución del programa con una amplia variedad de datos que determinan si el programa tiene errores) con la finalidad de cerciorar su funcionamiento y evitar **errores en su operación (ERRORES DE EJECUCIÓN)**.

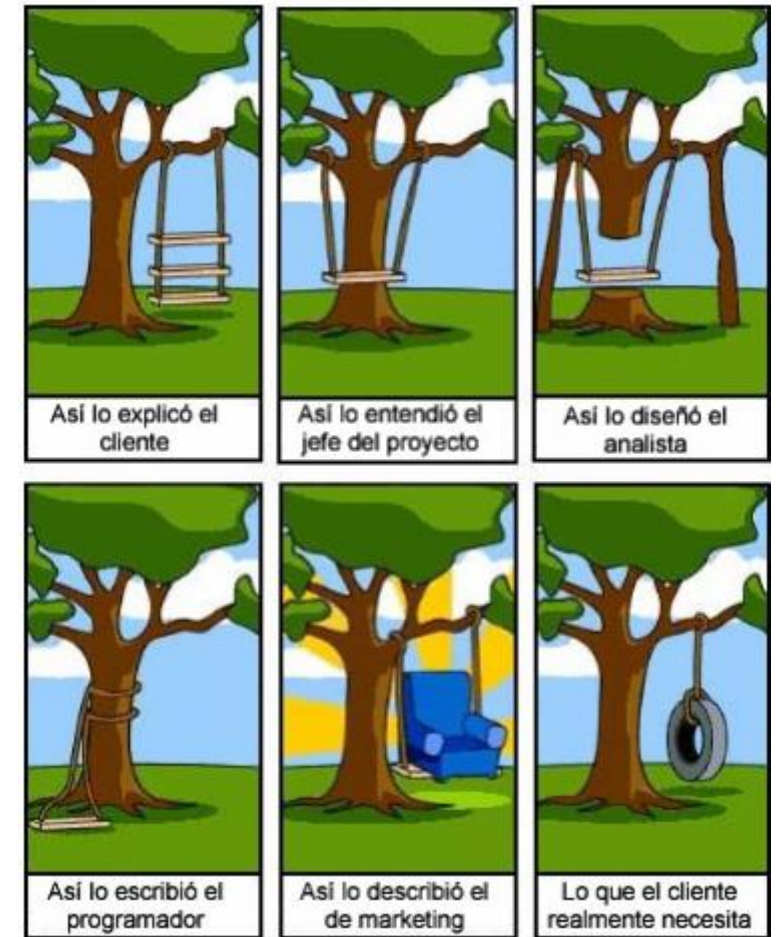
En cuanto a las pruebas a llevar a cabo, está la de ingresar valores extremos de entrada con la intención llevar al límite al programa y validar su comportamiento, el realizar las operaciones manualmente y compararlas con los resultados del programa, otra forma es ingresar datos erróneos que determinen su fiabilidad, es decir, su capacidad para recuperarse o, en su caso, el nivel de control frente al uso inadecuado del programa.

Los posibles errores que no detecta la compilación del sistema son los de **ejecución y lógicos**. En cuanto a los primeros se dice que la computadora los puede “entender”, pero no ejecutar, tales como división entre cero, la raíz cuadrada de un número negativo, exceder un rango de valores no permitidos, realizar una operación con un dato leído como carácter en vez de numérico, etc. Por otra parte, los errores lógicos se dice que son más difíciles de corregir, ya que generalmente son producto del mal análisis y diseño del problema.

# Ingeniería del software.

- **Software**
  - => Instrucciones + Estructuras de Datos + Documentación.
- **Crisis del software**
  1. Aumento de la necesidad de **creación** de SW.
  2. Se desarrolla software de manera desorganizada.
  3. Software de mala calidad, funcionamiento incorrecto.

Según estimaciones, el 26% de los grandes proyectos de software fracasan, el 48% deben modificarse drásticamente y sólo el 26% tienen rotundo éxito. La principal causa del fracaso de un proyecto es la falta de una buena planificación de las etapas y mala gestión de los pasos a seguir. ¿Por qué el porcentaje de fracaso es tan grande? ¿Por qué piensas que estas causas son tan determinantes?



# Ingeniería del software.

El desarrollo de software es un proceso complejo que exige gran coordinación y disciplina por parte del grupo de trabajo que lo desarrolle.

Para facilitar el desarrollo del software, lo dividimos en **etapas**.

## Ciclo de vida:

“Todas las etapas por las que pasa un proyecto de desarrollo de software desde que se inicia hasta que se finaliza y se considera una solución completa, correcta y estable”.





# Ingeniería del software.

---

- QUÉ

- Se va a desarrollar

1. Estudio del Sistema  
2. Planificación del Sistema  
3. Especificación de Requisitos  
4. Análisis del Sistema

---

- CÓMO

- Se va a desarrollar

1. Diseño (arquitectura, estructura de datos, ...)  
2. Codificación e implementación  
3. Pruebas

---

- CAMBIO

Adaptación o mejora del sistema

- Que se puede producir en el sistema

# Ingeniería del software: Fases de desarrollo.

---

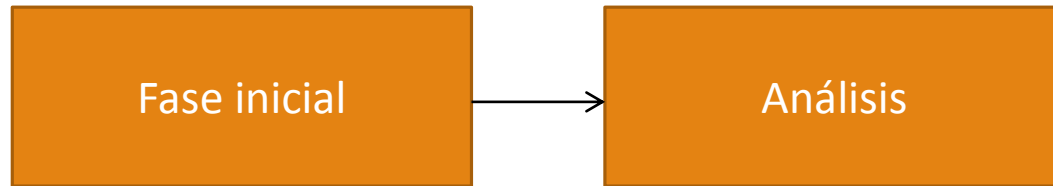


Fase inicial

En esta fase se hacen estimaciones de **viabilidad** del proyecto, es decir, si es rentable o no y se establecen las bases de las fases del proyecto.

# Ingeniería del software: Fases de desarrollo.

---



Se analiza el problema, es decir, se recopila, examina y formulan los requisitos del cliente.

Se realizan varias reuniones con el cliente, generando documentación en la que se recopilan los requisitos .

Al finalizar estas reuniones, se genera un acuerdo en el que el cliente se compromete a no variar los requisitos hasta terminar una primera **release**.

# Ingeniería del software: Fases de desarrollo.

---



Consiste en dividir el problema en partes y se determina la función de cada una de estas.

Se elabora una documentación técnica y detallada de cada módulo del sistema.

La documentación es realizada por el *analista junto al jefe de proyecto*.

# Ingeniería del software: Fases de desarrollo.

---



Se codificará el software en el lenguaje de programación seleccionado durante la etapa de diseño.

Se genera documentación del código que se desarrolla: entradas, salidas, módulos o librerías utilizadas , etc.

# Ingeniería del software: Fases de desarrollo.



Se realizan pruebas para verificar que el programa se ha realizado acuerdo las especificaciones originales. Existen varios tipos de pruebas:

- 1) Funcionales: Se validan las especificaciones establecidas por el cliente y, posteriormente, será validada por este.
- 2) Estructurales: Se realizan pruebas técnicas sin necesidad del consentimiento del cliente. Se realizarán cargas reales de datos.

# Ingeniería del software: Fases de desarrollo.

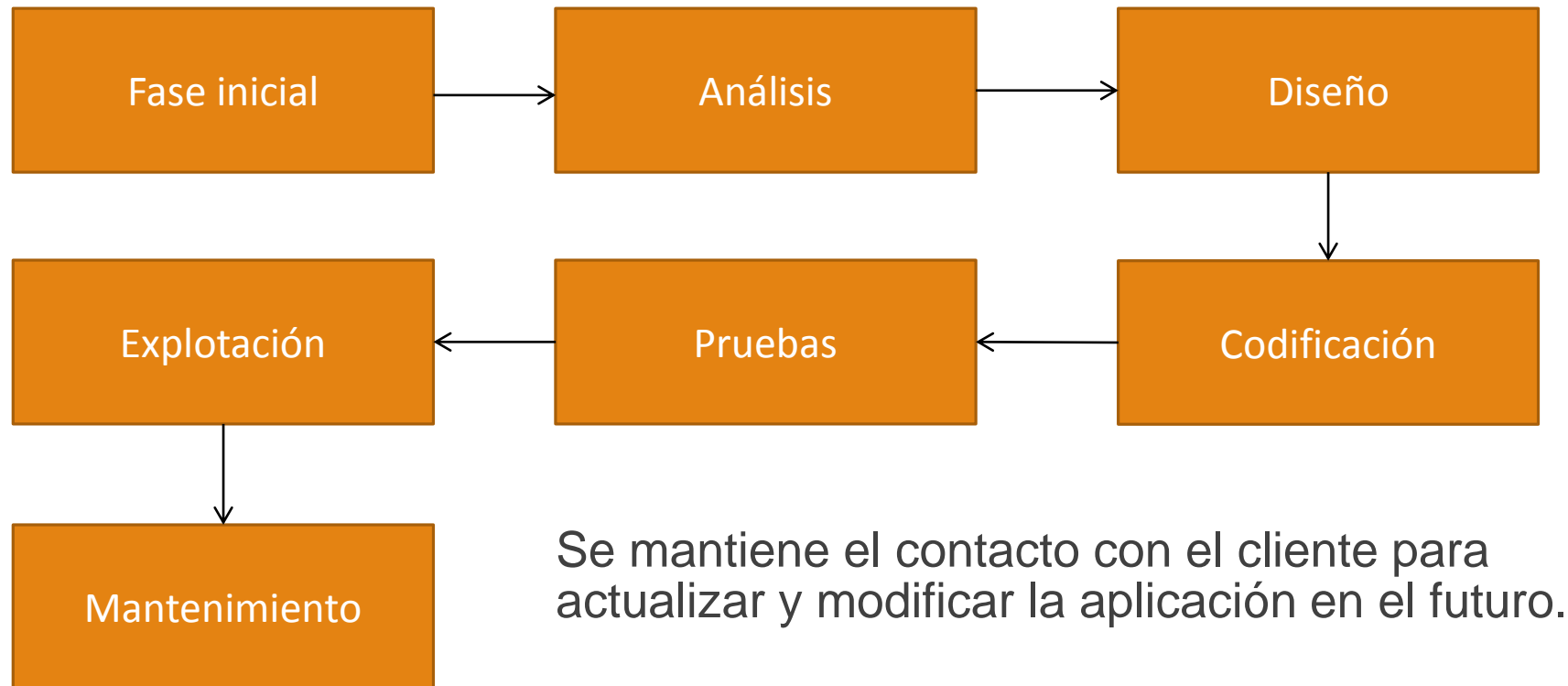
---



El software es instalado y utilizado en un entorno real, uso cotidiano.

Es la fase más larga, ya que suele conllevar múltiples incidencias (**bugs**) y nuevas necesidades.

# Ingeniería del software: Fases de desarrollo.





# Ingeniería del software: Desarrollo de software.

Según el ciclo de vida utilizado, nos encontramos con los siguientes modelos de desarrollo:

a) Modelo en cascada:

- Es el modelo de vida clásico del software.
- Solo es aplicable a pequeños desarrollos, ya que las etapas pasan de unas a otras sin retorno posible.

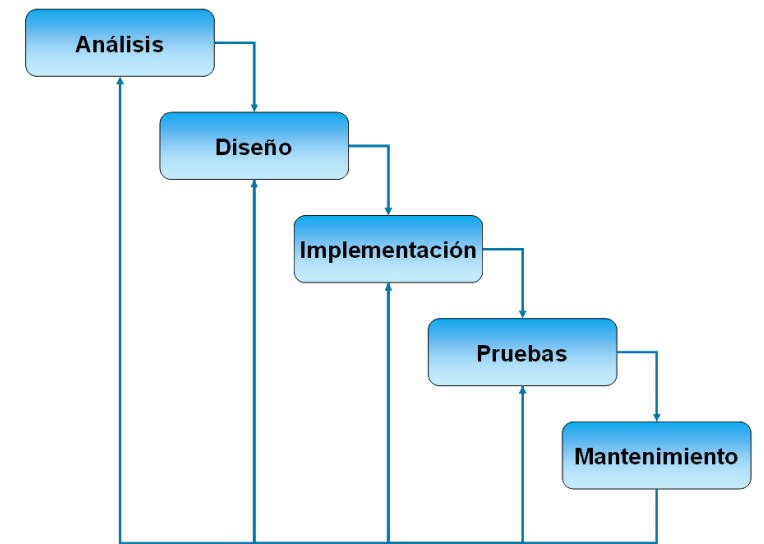
b) Modelo en cascada con realimentación:

- Parte del modelo anterior, añade retroalimentación entre etapas.
- No es el idóneo si se prevén muchos cambios.

## Problemas:

- Difícil de ajustar a proyectos reales, porque en los reales se producen **realimentaciones**.
- Difícil que el usuario exponga sus **requisitos** al principio.
- No está disponible versión operativa del proyecto hasta las últimas etapas.
- Se **retrasa** la localización y corrección de errores.

Estos modelos son útiles en proyectos cortos.



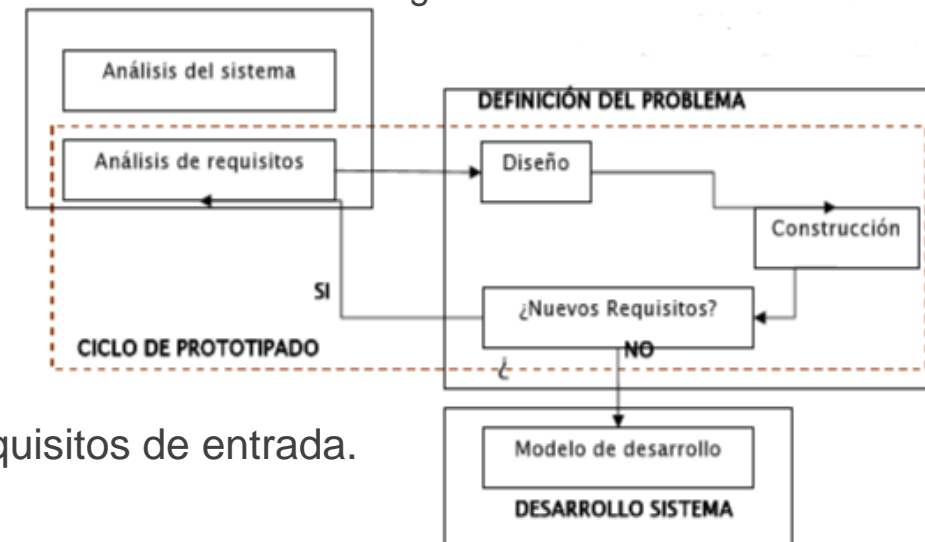
# Ingeniería del software: Desarrollo de software.

## c) Modelo orientado a prototipos:

- Prototipo es una primera propuesta que se irá refinando para la solución final.
- Se va realizando una solución provisional sin tener que llegar al final de todos los procesos.

## Problemas:

- La **gestión** de desarrollo del software es **muy lenta**. El cliente, ve que ya funciona algo ya piensa que se está cerca del producto final, con unos “pequeños ajustes”.
- Los primeros prototipos es posible que no se reutilicen y **no presentan calidad o fiabilidad**. Se ha hecho algo para que el cliente lo “pueda tocar” pero está “cogido con hilos”.
- No está disponible versión operativa del proyecto hasta las últimas etapas.
- Cuando el prototipo está preparado, es necesario **comunicación desarrolladores con el cliente** => produce datos para refinar el diseño.



Útil cuando se tienen los objetivos generales del sistema, pero falta definir los requisitos de entrada.

# Ingeniería del software: Desarrollo de software.

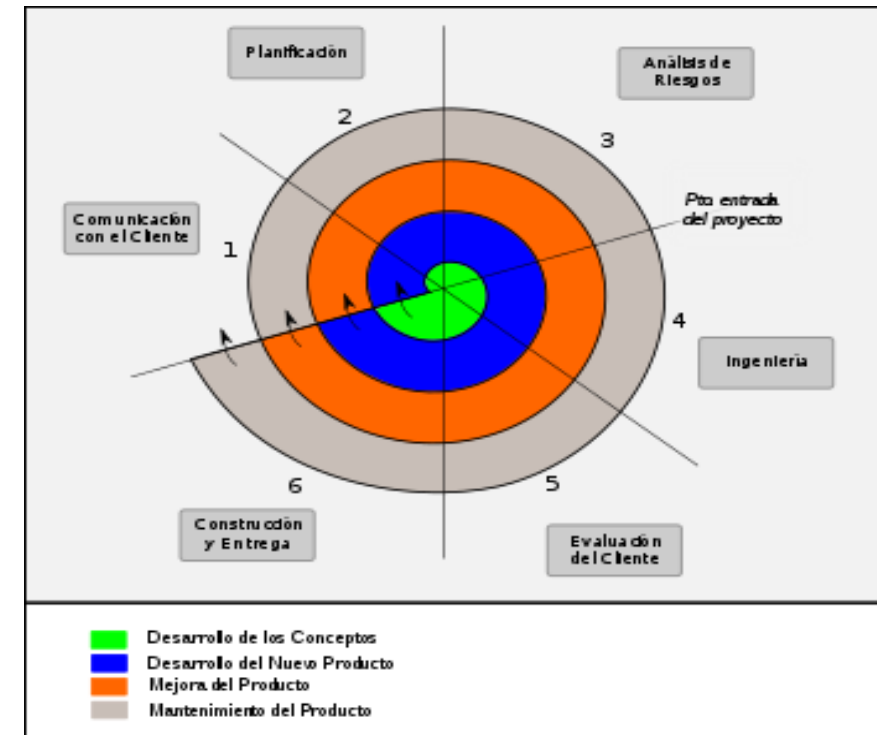
## d) Modelo en espiral:

- Es de los más recientes.
- Tienen en cuenta la naturaleza cambiante y evolutiva del software, mezcla de los dos modelos anteriores.
- El software se va construyendo repetidamente, añadiendo mejoras en cada versión.
- En cada bucle o iteración representa un conjunto de actividades

## PROBLEMAS:

- Genera **mucho tiempo** en el **desarrollo** del sistema.
- Modelo **costoso**.
- Planificar un proyecto con esta metodología genera cierta incertidumbre, ya que desconocemos en el número de iteraciones que serán necesarias. En este contexto la **evaluación de riesgos** es de la mayor importancia y, para grandes proyectos, dicha evaluación requiere la intervención de **profesionales de gran experiencia**.

Útil cuando se tienen los objetivos generales del sistema, pero falta definir los requisitos de entrada.



# Ingeniería del software: Desarrollo de software.

---

**Ejercicio:** Investiga y realiza una breve introducción de la metodología Agile y Scrum, deberás explicar sus principales conceptos y su ciclo de vida.

# Herramientas de apoyo al desarrollo de software

Las **herramientas CASE** (**C**omputer **A**ided **S**oftware **E**ngineering) son diversas aplicaciones informáticas destinadas a aumentar la productividad y calidad en el desarrollo de software.

## ➤ Objetivos:

### ✓ Incrementar

- Productividad del equipo.
- Calidad del software
- Reusabilidad del software

### ✓ Reducir

- Costes de desarrollo y mantenimiento

### ✓ Amortizar

- Gestión del proyecto
- Desarrollo del software
- Mantenimiento del software.

# Herramientas CASE

---

Clasificación según la fase del ciclo de vida que abordan:

- CASE frontales (front-end) o **Upper CASE**: Herramientas de apoyo a las primeras fases:
  - Planificación
  - Análisis de requisitos.
- CASE intermedias o **Middle CASE**: Herramientas de apoyo en las fases intermedias:
  - Análisis
  - Diseño.
- CASE dorsales (back-end) o **Lower CASE**: Herramientas de apoyo a las últimas fases:
  - Implementación (generación de código).
  - Pruebas.
  - Mantenimiento.

Ejemplos de herramientas CASE libres son: ArgoUML, Use Case Maker, ObjectBuilder...