# UT01. DESARROLLO DEL SOFTWARE.

Entornos de Desarrollo – 1ºDAW

# Índice

# 1. Sistema informático.

# Sistema Informático.

- **Sistema informático = Hardware + Software**
  - Conjunto de **dispositivos** físicos, **programas** y **datos** que, en conjunto, permiten el tratamiento automático de la información.
  - A los dispositivos físicos se les conoce como "**hardware**", y por oposición, los programas y datos se engloban en el "**software**".
- **Ordenador o computador:** compone de dos partes bien diferenciadas: hardware y software.
  - Máquina electrónica capaz de recibir información de entrada y procesarla para proporcionar información resultante bajo el control de un programa almacenado.

# 2. Relación Hardware – Software

# Software.

- *"Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora"* (RAE)
- *"Conjunto de programas de cómputo de, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación" (estándar 729 IEEE)*

# Software.

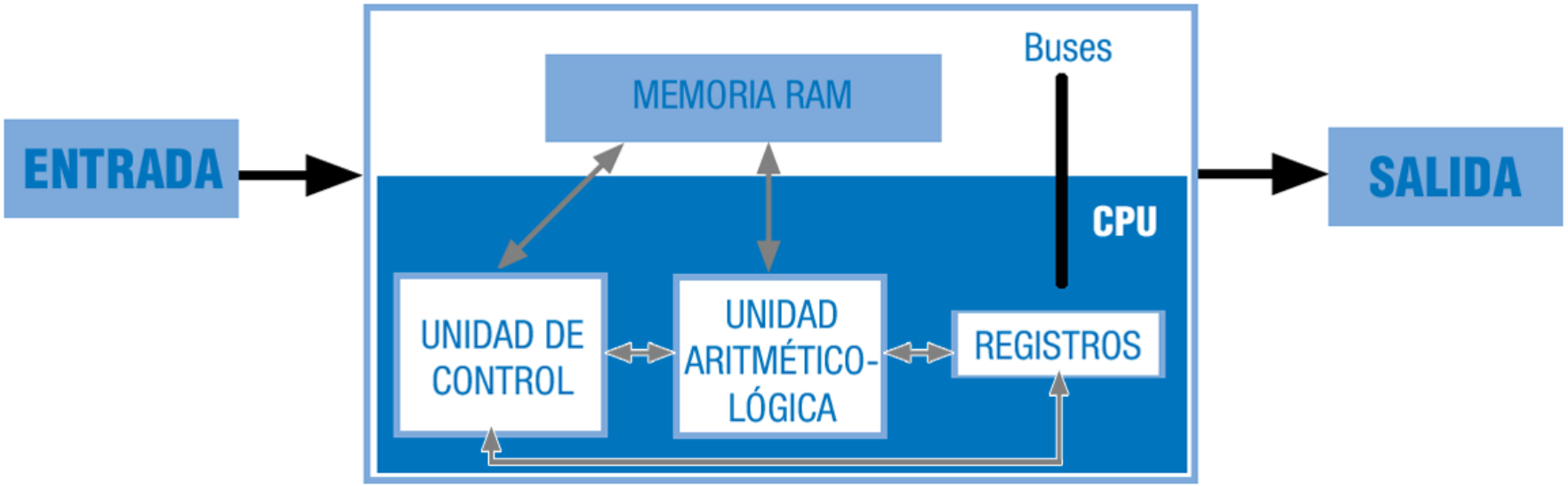
- El software es la parte intangible de un sistema informático, equivale al soporte lógico.
- El software es el encargado de comunicarse con el hardware, es decir, traduce las órdenes del usuario en órdenes comprensibles para el hardware.
- El software se desarrolla, no se fabrica.
- El software no se estropea.
- El software puede construirse a medida.

# Relación Hardware – Software

- Para entender mejor el concepto de software es necesario hacer referencia a las diferentes partes del hardware de un ordenador:
  - **CPU** (Unidad central de proceso): ejecuta las instrucciones de los programas. Toda instrucción por compleja que sea se traduce a operaciones sencillas aritméticas y lógicas que se realizan en binario. La CPU está formada por:
    - ALU (Unidad aritmético-Lógica): ejecuta operaciones aritmético lógicas encomendadas por la UC con los datos que recibe, y devuelve el resultado, siguiendo las órdenes de la UC.
    - UC (Unidad de control): recoge las instrucciones que están en la memoria principal y ordena su ejecución mediante el envío de señales a la ALU y los registros.
      - Los registros forman el almacenamiento interno de la CPU.
  - **Memoria principal** (RAM): contiene las instrucciones del programa que hay que ejecutar y los datos. Es una memoria volátil.
  - **Unidad E/S**: permite la comunicación del ordenador con el exterior transfiriendo información a través de los periféricos.

# Estructura funcional de un ordenador.

- La primera arquitectura hardware con programa almacenado se estableció en 1946 por John Von Neumann: (**Arquitectura Von Neumann**)

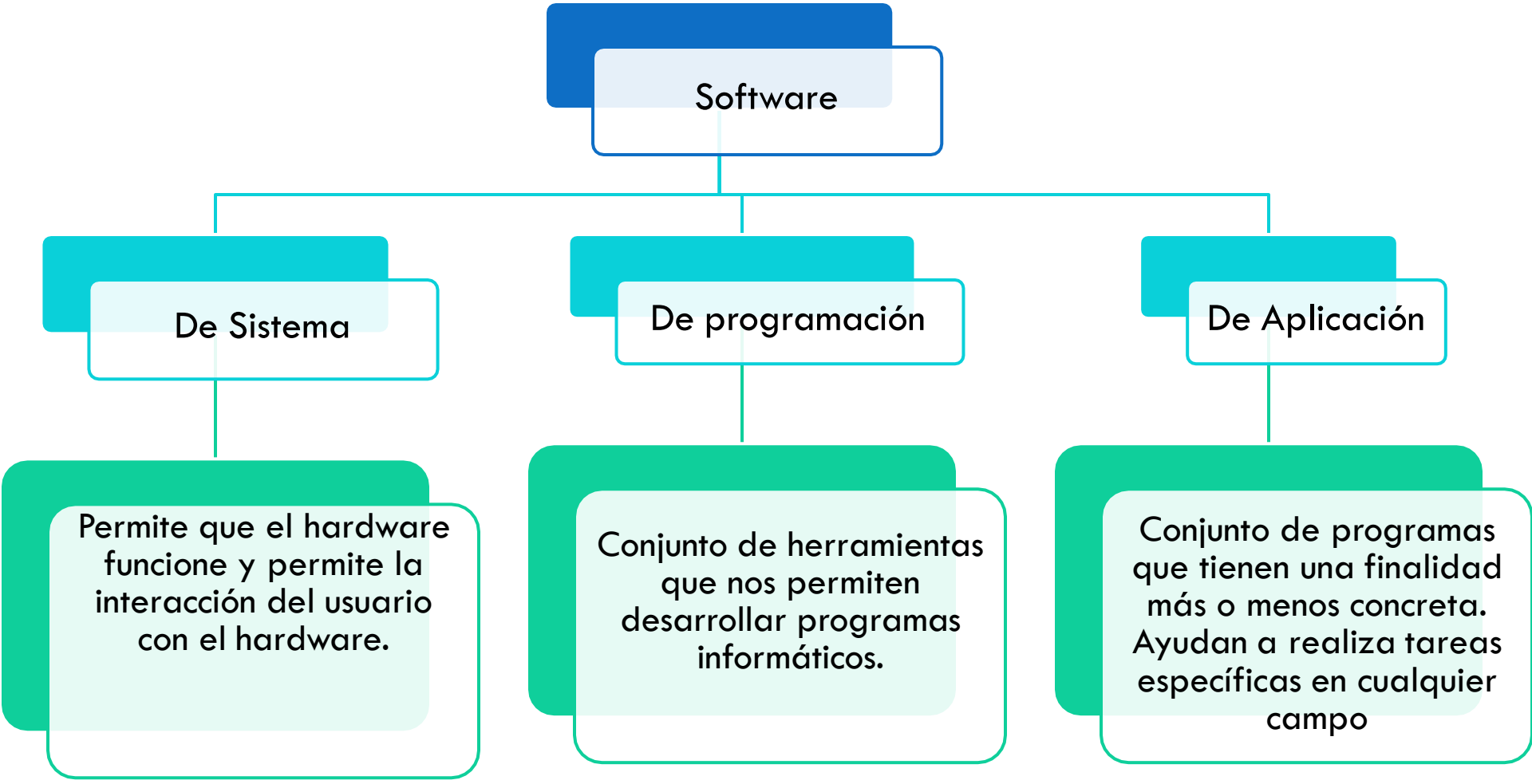


# Programas, datos e instrucciones

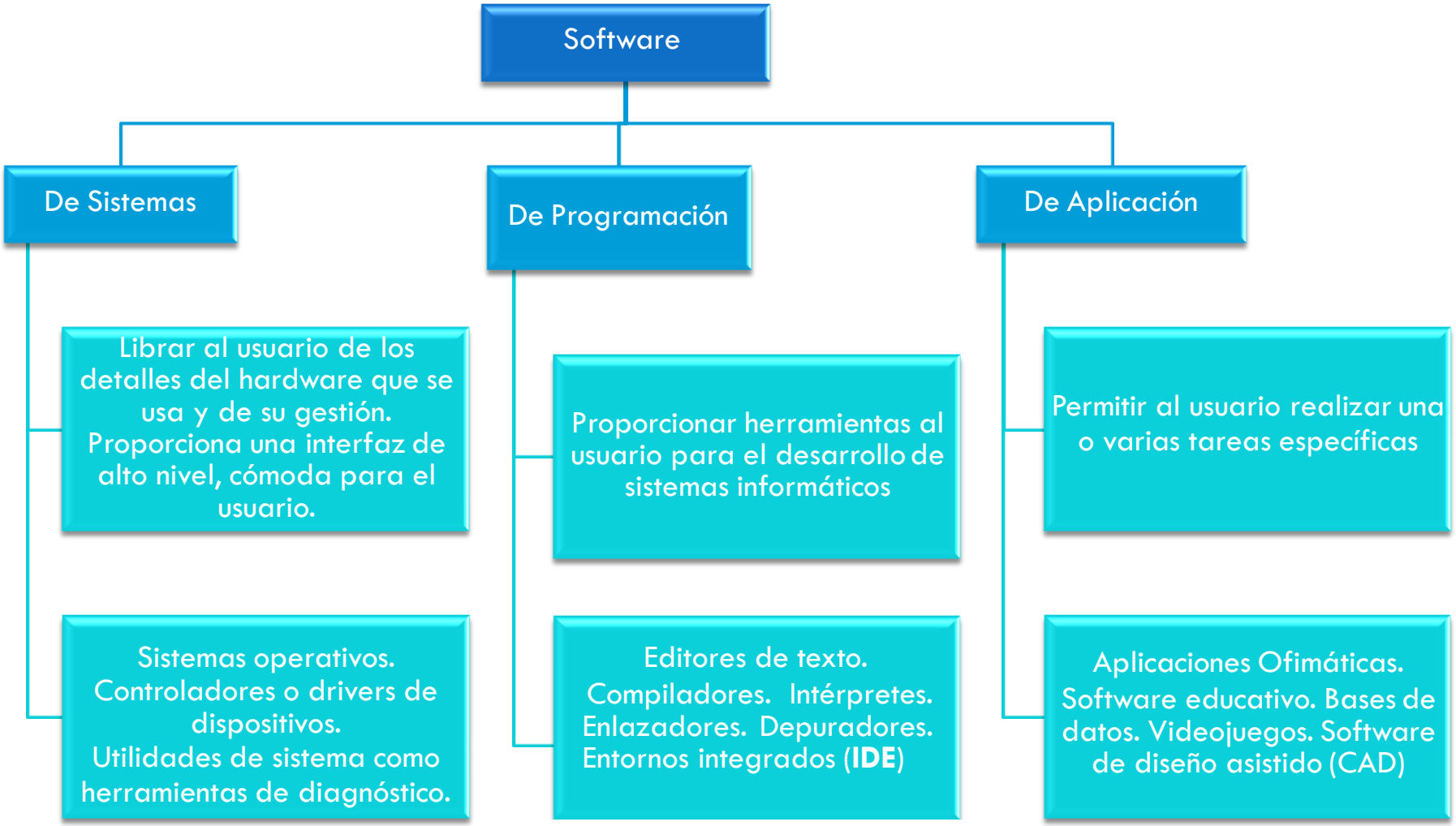
- **Programa:** Secuencia lógica de **instrucciones,** que se le dan al dispositivo, que manipulan **datos** para obtener unos **resultados** que serán la solución del problema que resuelve el programa.
- **Datos:** símbolos que representan un valor numérico, un hecho, un objeto o una idea.
- **Instrucciones:** operaciones bien definidas, específicas de cada máquina y almacenadas de forma permanente en la misma, que establecen las tareas que debe realizar.

# 3. Tipos de software

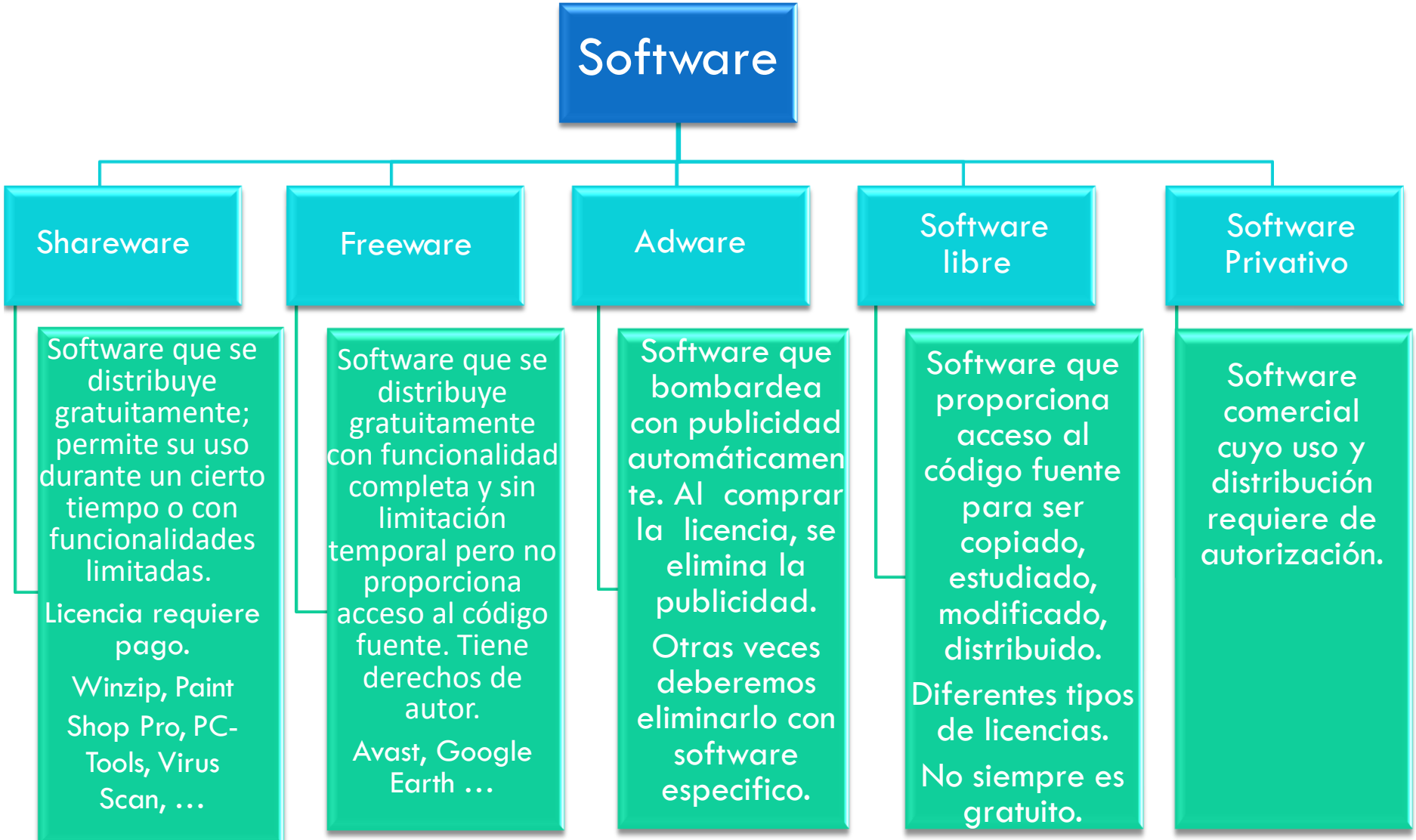
# Tipos de software. Según el tipo de tarea



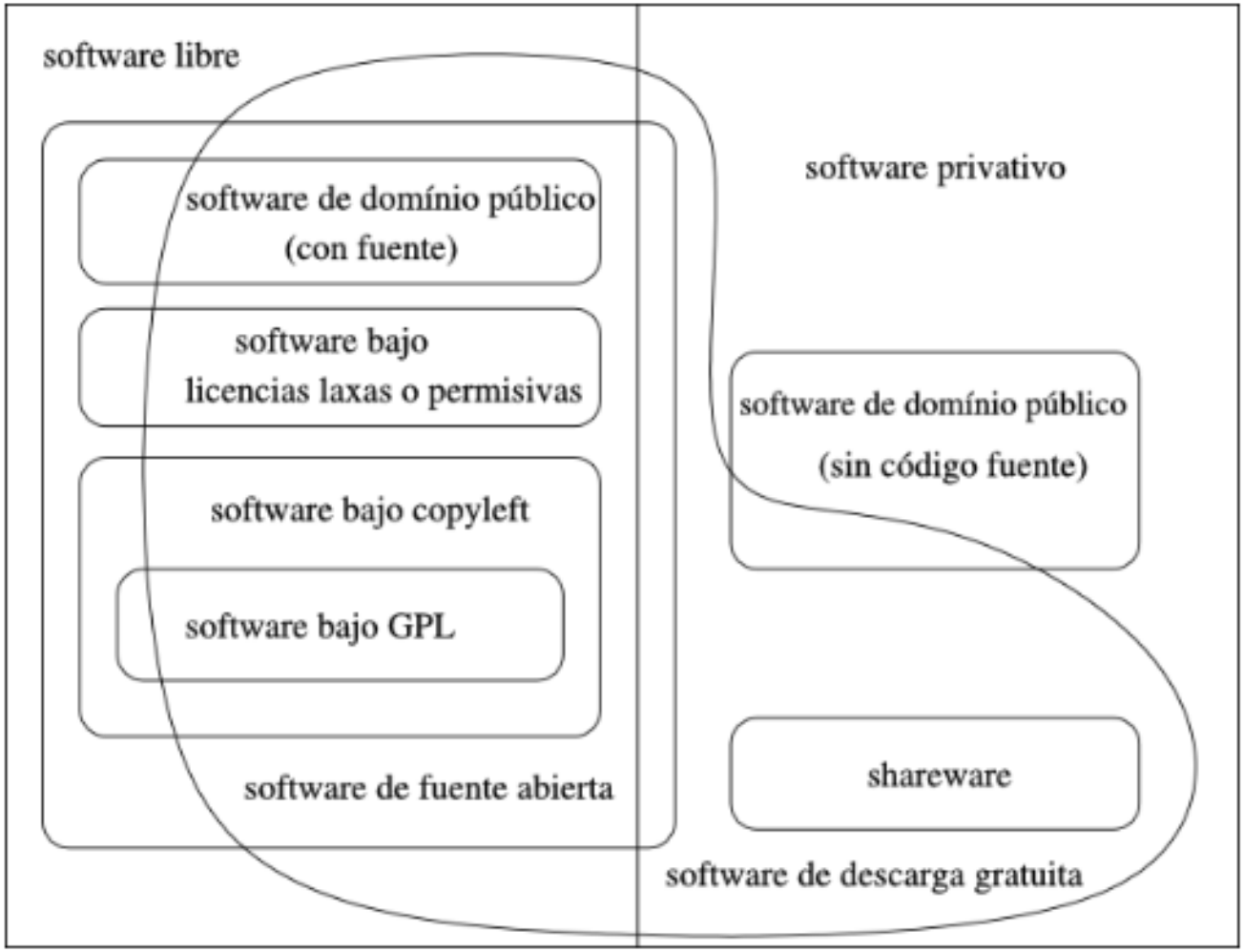
# Tipos de software. Según el tipo de tarea.



# Tipos de sw. según el método de distribución



# Tipos de licencias software.



# Tipos de licencias software.

- Las **licencias software** son un contrato entre el autor y los usuarios y establecen los términos, condiciones y cláusulas que se deben cumplir.
- Hoy se clasifican en dos grandes grupos: software libre y software propietario.
  - El **software libre** no tiene copyright pero no tiene por qué ser gratuito. El usuario tiene la libertad de usar, copiar, modificar, distribuir el software a su gusto. Su fin es expandir el conocimiento.
  - El **software propietario** está limitado por su autor, que prohíbe su copia, distribución o modificación sin permiso o pago previo (copyright).
  - Un tercer grupo, es el **software gratuito o de dominio público** (aunque el autor puede imponer restricciones).

# Tipos de licencias software.

- Dentro **del software libre** distinguimos las siguientes licencias:
  - **Licencias sin copyleft:** el autor autoriza redistribuir, modificar, añadir restricciones adicionales, que pueden ser no libres completamente y requieren un software de pago para su completo funcionamiento.
  - **Licencias con copyleft:** en contraposición al copyright (derechos de autor) es software libre que no permite a quienes lo cambian o redistribuyen agregar ninguna restricción adicional, de forma que debe ser software libre siempre. **Copyleft** es una práctica legal que consiste en el ejercicio del derecho de autor para propiciar el libre uso y distribución de una obra, y exige las mismas libertades al distribuir sus copias y derivados.

# Tipos de licencias software.

- Dentro del software libre distinguimos las siguientes licencias:
  - **Licencias GPL (GNU Licencia Pública General)**: muy usada en el mundo del software libre y open source. Garantiza a los usuarios finales la libertad de usar, estudiar, copiar, modificar el software pero bajo la misma licencia GPL.
  - **Licencias Open Source (código abierto):** permite la libre distribución , modificación del código fuente en las mismas condiciones que el software original.
  - **Licencias Freeware**: autoriza el uso de forma libre y gratuita para particulares, pero no para empresas y organismos oficiales.
  - **Licencias Shareware**: son licencias para evaluación del código o demos.

# Actividad 1

- Investiga sobre Licencias de Software. Software libre y propietario y de dominio público.

http://www.gnu.org/licenses/license-list.es.html#SoftwareLicenses

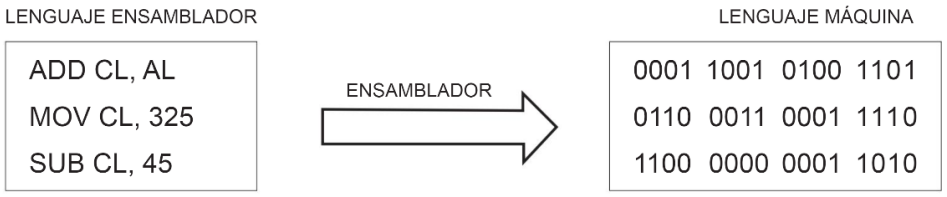
# 4. Lenguajes de programación

# Lenguajes de programación

- Los programas han de ser ejecutados por el ordenador y por ello las instrucciones del programa han de estar escritas en un lenguaje que el ordenador entienda, es decir, lenguaje binario.
- El lenguaje binario es conocido como **lenguaje máquina**.
- La tarea de programar en binario es muy compleja para el ser humano, es por ello que las personas utilizamos lenguajes de programación más cercanos al lenguaje natural, denominados **lenguajes de alto nivel**.
- Un lenguaje de programación es una notación para escribir programas a través de los cuales establecer comunicación con el hardware para la realización de una tarea.

# Lenguajes de programación

- Clasificación de los lenguajes de programación en función de la cercanía al lenguaje natural:
  - **Lenguajes de alto nivel:** facilitan la tarea al programador y evitan que éste necesite conocer las características de la máquina. Como inconveniente presentan que deben ser traducidos a lenguaje máquina.
  - **Lenguajes de nivel intermedio**: lenguaje ensamblador. Necesita ser traducido a lenguaje máquina por un traductor llamado **ensamblador.**




  - **Lenguajes de bajo nivel**: lenguaje binario o lenguaje máquina.

# 5. Código fuente, código objeto y código ejecutable

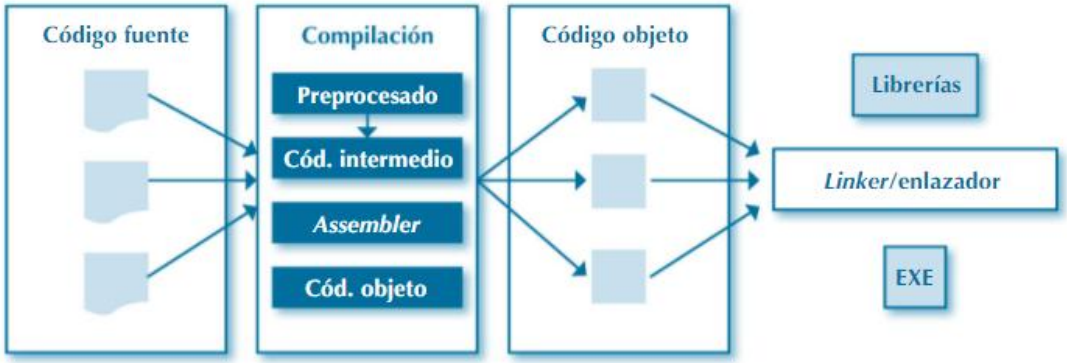
# Código fuente y código objeto

- **Código fuente:** son las instrucciones escritas por el programador en un lenguaje de programación de alto nivel concreto siguiendo las reglas y sintaxis de dicho lenguaje.
  - El código fuente ha de ser traducido al lenguaje que entiende el ordenador o lenguaje máquina (en binario) dando lugar al **código objeto**.
- Este proceso de traducción de **código fuente** a **código objeto o lenguaje máquina** depende del lenguaje de programación:
  - Compilación.
  - Interpretación.

# Compiladores

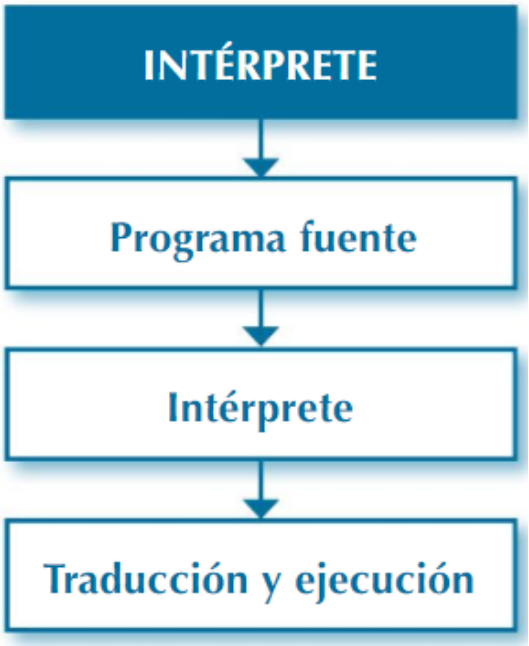
- **Compilación**
  - El **compilador** traduce el código fuente a código objeto.
  - El compilador debe ser especifico del S.O donde se vaya a ejecutar el programa.


- El compilador realiza la traducción e informa de los **errores de sintaxis**.
- El código objeto generado puede ser ejecutado por la máquina directamente o puede necesitar otros pasos como ensamblado, enlazado y carga.
- Una vez finalizado obtenemos el **código ejecutable.**
- Una vez se haya obtenido el programa ejecutable no se necesita software especifico para su ejecución.



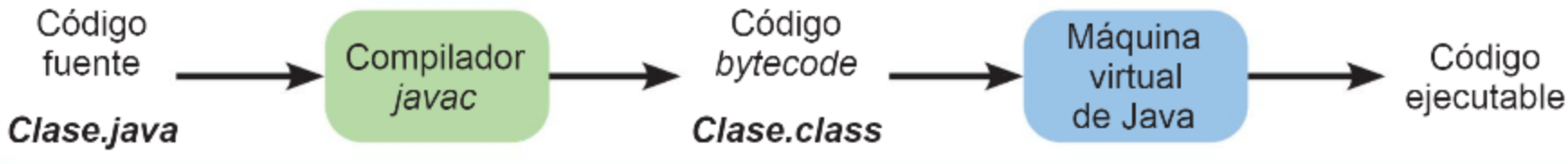
# Intérpretes

- **Interpretación**
  - El intérprete es un programa que se encarga de **analizar** y **ejecutar** el programa escrito en un L.P.
  - El intérprete **lee** una a una las instrucciones, las **analiza, traduce** a código máquina directamente (si no hay error) y **ejecuta** instrucción a instrucción del código fuente.
  - **NO se genera código objeto ni programa ejecutable.**
  - Generan programas de **menor tamaño** pero son **más lentos**.
  - Los lenguajes interpretados necesitan disponer en máquina donde se van a ejecutar de un programa intérprete, mientras que los compilados no.
  - Ejemplos: PHP, JavaScript, Phyton, …



# Bytecodes

- Hay lenguajes como Java que son **pseudo-compilados** o **pseudo-interpretados.**
- En Java el código fuente es compilado a **bytecodes** (estructura similar a instrucciones máquina) que son **independientes del hardware.**
- Java requiere de una **máquina virtual** que hace de intérprete que traduce los bytecodes a código de la máquina en cuestión.

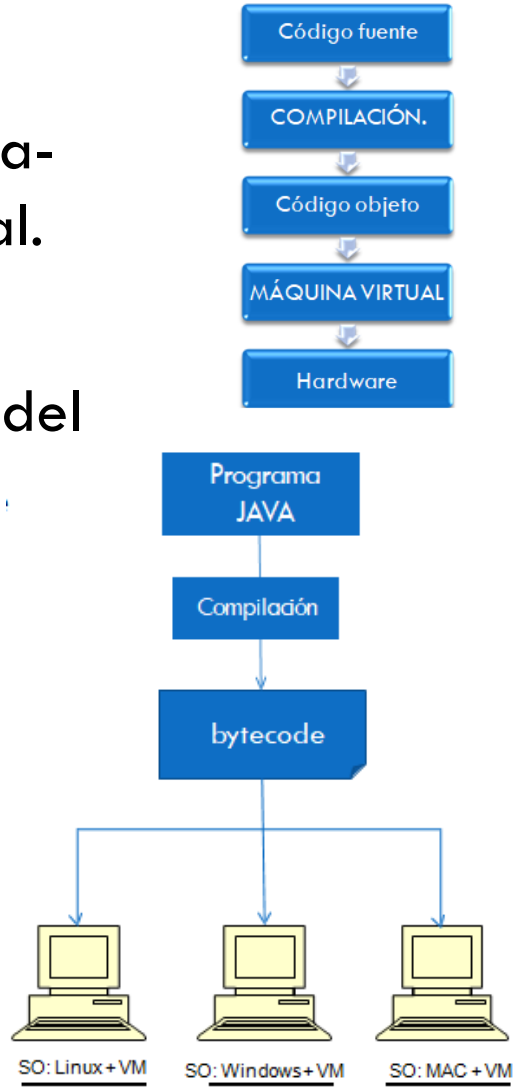


# Entornos de ejecución

- Un **entorno de ejecución (JRE o Java Runtime Environment)** es un servicio de **máquina virtual de proceso** que sirve como base software para la ejecución de programas sobre cualquier plataforma, proporcionando un entorno de ejecución independiente del hardware y sistema operativo subyacente.
- Actividades del entorno durante la ejecución:
  - Configurar la memoria principal
  - Enlazar los archivos del programa con las bibliotecas existentes y con los subprogramas creados.
  - Depurar los programas: comprobar la existencia (o no existencia) de errores semánticos del lenguaje (los sintácticos ya se detectaron en la compilación).
- Componentes:
  - **JMV o Máquina virtual Java** → programa que interpreta el código de la aplicación escrito en Java.
  - **Bibliotecas** de clase estándar que implementan el API de Java.

# Máquina virtual de Java

- **Máquina virtual de Java (JVM)** = máquina ficticia que traduce las instrucciones máquina-ficticia en instrucciones para la máquina real.
  - El programa escrito en Java no es ejecutado realmente por el procesador del ordenador, sino por la JVM.
  - **Garantiza la portabilidad de las aplicaciones.**




# 6. Desarrollo de software

# Ingeniería del software

- La Ingeniería del software es la rama de las ciencias de la computación formada por un conjunto de métodos, herramientas y técnicas que se utilizan en el desarrollo de los programas informáticos.
- "Ingeniería de software es el estudio de los principios y metodologías para el desarrollo y mantenimiento de sistemas software (Zelkovitz, 1978)."
- En un principio el desarrollo del software consistía en escribir código directamente. En la década de los 70 se produce la crisis del software. Se cuestionó entonces cómo desarrollar el software, cómo mantenerlo y cómo satisfacer la demanda creciente de software ➔ Nace la ingeniería del software.

# Ciclo de vida del software.

- El desarrollo del software requiere completar una serie de fases o tareas que en conjunto se denominan "**Ciclo de vida del software**".
- **Ciclo de vida del software**
  - **IEEE 1074**: "una aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del software".
  - **ISO 12207-1**: "actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso.
- **Ciclo de desarrollo**
  - Subconjunto del anterior que empieza en el análisis y termina con la entrega del sistema al usuario

# Ciclo de vida del software.

- **Etapas** comunes a casi todos los modelos de **ciclo de vida:**
  1. **Análisis**. Se trata de analizar las necesidades del usuario del software. Construye un modelo de requisitos. Documentación entendible, completa y fácil de verificar y modificar. **Qué hacer.**
  2. **Diseño**. **Cómo hay que hacerlo**. Se traducen los requisitos en componentes software (tablas de la B.D, programas, funciones, clases con atributos y métodos, interfaces de usuario, etc)
  3. **Codificación**. Traducir el diseño realizado al lenguaje de programación escogido. Se obtiene **código ejecutable.**
  4. **Pruebas**. Comprobar que se cumplen criterios de corrección y calidad. Garantizan el correcto funcionamiento del sistema.
  5. **Explotación.** Instalación y puesta en marcha en el entorno del cliente.
  6. **Mantenimiento**. Después de la entrega del software. Adaptarse a los cambios. Cambios por errores. Adaptación al entorno (p.ej. cambio de SO), mejoras funcionales….

# Ciclo de vida del software

7. **Documentación**: en cada etapa se generan documentos. Cada etapa tiene como entrada los documentos de la etapa anterior y obtiene nuevos documentos.


- Existen diferentes modelos de ciclo de vida, adaptados a los distintos paradigmas de programación.
- Modelos de ciclo de vida:

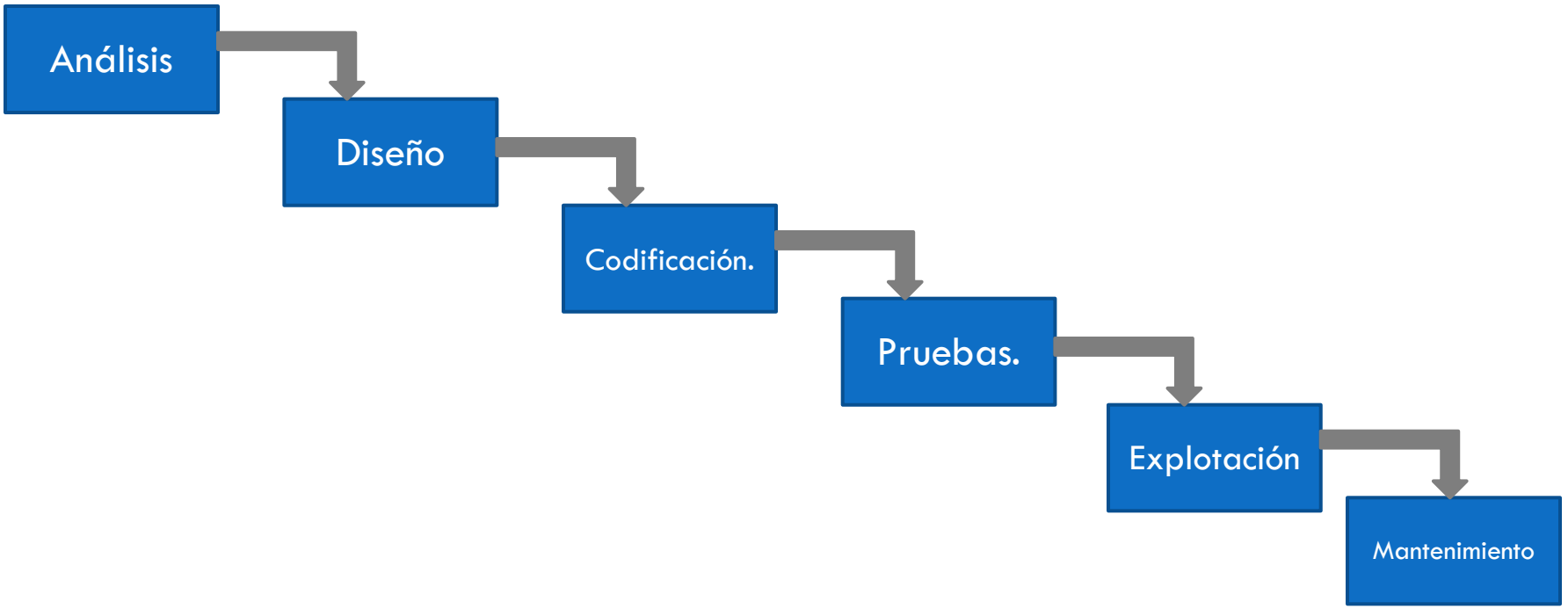
https://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software

# Ciclo de vida del software

- Además de todas estas tareas, es preciso realizar otras adicionales, menos técnicas, relacionadas con la gestión del proyecto, por ejemplo:
  - **Seguimiento y control del proyecto**: evaluar periódicamente el proyecto y compararlo con el plan establecido con el fin de tomar decisiones en caso de desviaciones.
  - **Administración del riesgo**: consiste en evaluar los riesgos (problemas) que pueden afectar al resultado del proyecto o a la calidad del producto.

# Modelos de ciclo de vida I

- **Modelo en cascada clásico**
  - Propuesta por Royce [ROYCE, 1970] con variaciones a lo largo del tiempo, Sommerville, Boehm, …



# Modelos de ciclo de vida I

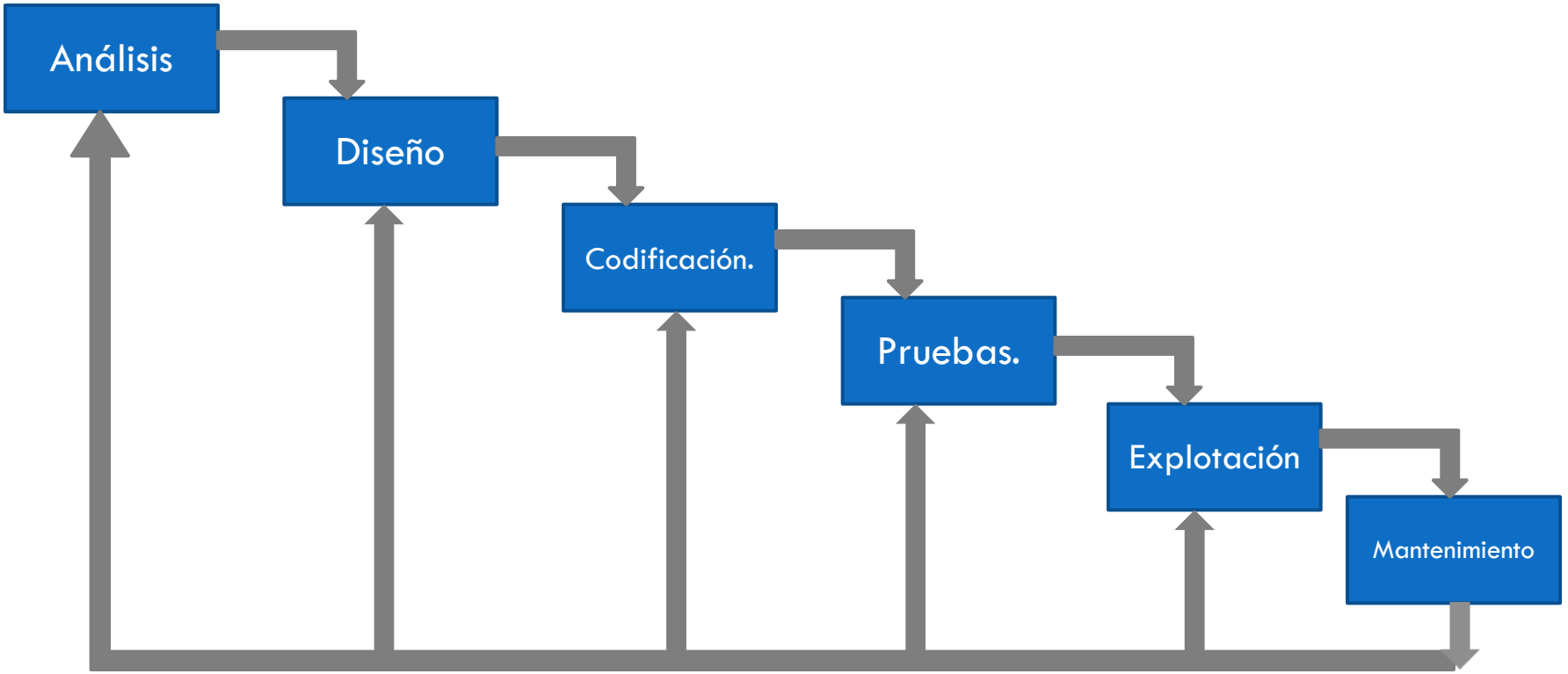
- **Modelo en cascada clásico**
  - **Características**
    - Tiene que finalizar una fase para pasar a la siguiente.
    - Difícil de utilizar.
    - Requiere conocer de antemano todos los requisitos del Sistema.
    - Para desarrollos pequeños.
    - No permite retroalimentación (no refleja el mundo real)
    - Se tarda mucho en pasar por todo el ciclo puesto que no se puede pasar a una etapa sin terminar la anterior.

# Modelos de ciclo de vida I

- **Modelo en cascada clásico**
  - **Ventajas**
    - Fácil de comprender, planificar y seguir.
    - La calidad del producto resultante es alta.
    - Permite trabajar con personal poco cualificado.
  - **Inconvenientes**
    - La necesidad de tener todos los requisitos definidos desde el principio
    - Difícil volver atrás si se comenten errores en una etapa
    - Producto no disponible para su uso hasta que no está terminado
  - **Se recomienda su uso cuando:**
    - El proyecto es similar a alguno que se haya realizado con éxito anteriormente
    - Los requisitos son estables y están bien comprendidos

# Modelos de ciclo de vida II

- **Modelo en cascada con retroalimentación**



# Modelos de ciclo de vida II

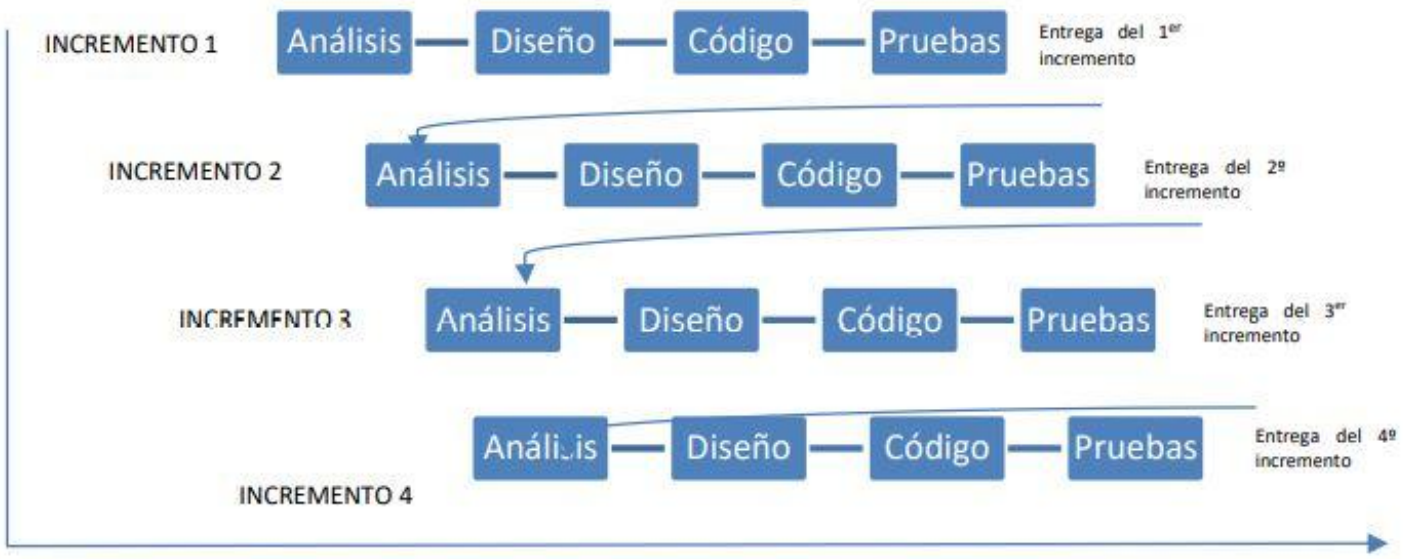
- **Modelo en cascada con retroalimentación**
  - **Características**
    - Uno de los modelos más utilizados.
    - Proviene del anterior con retroalimentación entre etapas.
    - Vuelta atrás para corregir, modificar distintos aspectos.
    - Modelo adecuado para proyectos con pocos cambios y requisitos claros.
- Hoy en día el desarrollo del software está sometido a continuos cambios, por lo que estos ciclos de vida no son adecuados.

# Modelos de ciclo de vida III

- **Modelos evolutivos.**
  - Software que evoluciona con el tiempo.
  - Diferentes versiones cada vez más completas hasta llegar al producto final.
  - Los más conocidos:
    - Modelo iterativo o incremental
    - Modelo en espiral
    - Modelo de prototipos

# Modelos de ciclo de vida III

- **Modelo iterativo incremental.**
  - Se proporcionan funcionalidades parciales al cliente y se aumentan en entregas posterirores.
  - Varios ciclos en cascada que se repiten y refinan en cada ***incremento***.
  - Las versiones son cada vez más completas hasta llegar al producto final.
  - **Ejemplo: un procesador de textos:** 1 inc: funciones básicas de gestión de archivos y producción de documentos. 2 inc: desarrollo de funciones gramaticales y de corrección ortográfica. 3 inc: desarrollo de funciones avanzadas de paginación….



# Modelos de ciclo de vida III

- **Modelo iterativo incremental**
  - **Ventajas**
    - No se necesitan conocer todos los requisitos al comienzo
    - Permite la entrega temprana al cliente de partes operativas del software
    - Las entregas facilitan la realimentación de los siguientes entregables
    - Reduce los riesgos de retrasos, cambios de requisitos y problemas de aceptación
  - **Inconvenientes**
    - Difícil estimación del esfuerzo y el coste final necesario
    - Riesgo de no acabar nunca
    - No recomendable para desarrollo de sistemas de tiempo real, de alto nivel de seguridad, de procesamiento distribuido, y/o de alto índice de riesgos
  - **Se recomienda su uso cuando:**
    - El proyecto es similar a alguno que se haya realizado con éxito anteriormente.
    - Los requisitos o el diseño no están completamente definidos y es posible que haya grandes cambios.

# Modelos de ciclo de vida IV

- **Modelo de prototipos**
  - Se centra en representar aspectos que serán visibles para el cliente.
  - Permite que todo el sistema o una parte se construya rápidamente.
  - Permite que desarrollador, usuario y cliente tengan una comprensión unificada de lo que se necesita y lo que se propone como solución.


ESQUEMA BÁSICO DEL CICLO PROTOTIPADO

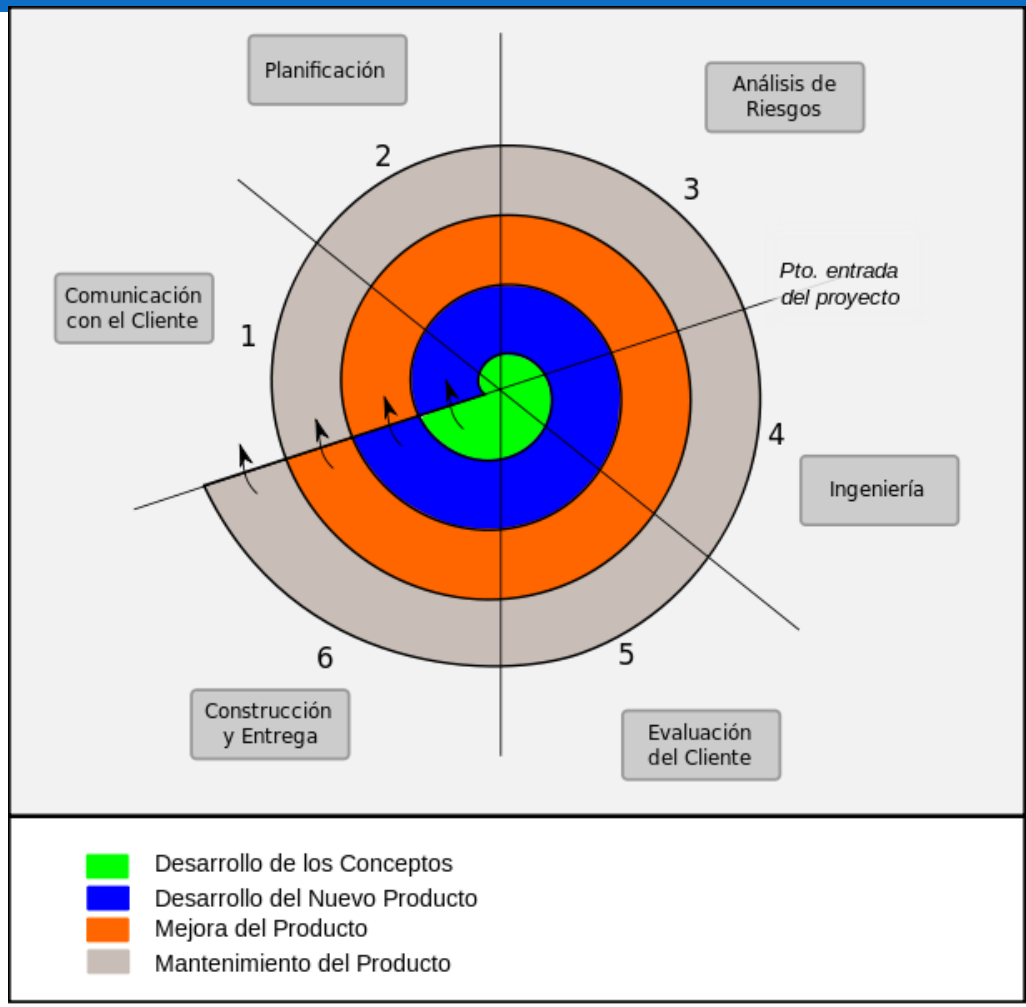
# Modelos de ciclo de vida IV

- **Modelo de prototipos**
  - **Ventajas**
    - Reduce riesgos
    - Reduce costes y aumenta la probabilidad de éxito
    - Incorpora requisitos de calidad


  - **Inconvenientes**
    - El cliente puede pensar que el prototipo es el producto final.
    - El desarrollador puede caer en la tentación de ampliar el prototipo para construir el sistema final sin tener en cuenta compromisos de calidad y mantenimiento que tiene con el cliente.
  - **Se recomienda su uso en:**
    - Proyectos de gran tamaño y que necesitan constantes cambios.
    - Proyectos donde sea importante el factor de riesgo
    - Muy utilizado en sistemas orientados a objetos.

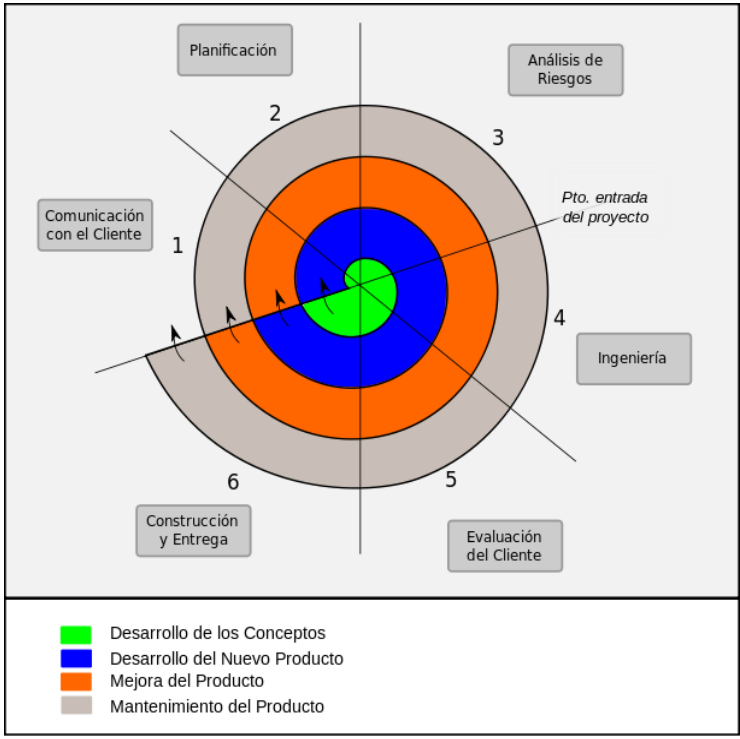
# Modelos de ciclo de vida V

- **Modelo en espiral**
  - Une las mejores características de los dos anteriores añadiendo el análisis de riesgos.
  - El software se va construyendo repetidamente en sucesivas versiones que son cada vez mejores por incrementar la funcionalidad de cada versión



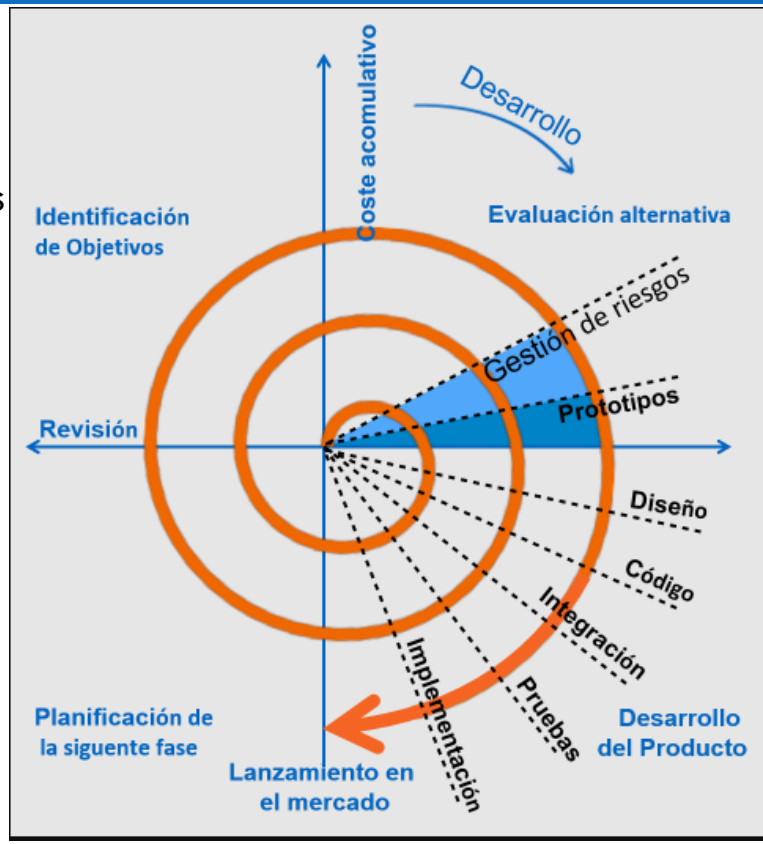
# Modelos de ciclo de vida V

- **Modelo en espiral**
  - Durante las primeras iteraciones se entrega un prototipo. En iteraciones posteriores se producen versiones más completas. Fases:
    - Determinar objetivos.
    - Análisis de riesgos. Un riesgo puede ser: requisitos no comprendidos, mal diseño, errores en la implementación.
    - Desarrollar y probar
    - Evaluación. Revisar y evaluar lo hecho y decidir si se continua, planificando las fases del ciclo siguiente.



# Modelos de ciclo de vida V

- **Modelo en espiral**
  - **Ventajas**
    - No se requiere una definición completa de los requisitos al principio.
    - Análisis de riesgos en todas las etapas.
    - Reduce riesgos mediante los prototipos
    - Incorpora objetivos de calidad.
  - **Inconvenientes**
    - Costo del proyecto aumenta a medida que la espiral pasa por sucesivas iteraciones.
    - Requiere de mucha experiencia en la evaluación de riegos
  - **Se recomienda su uso en:**
    - Proyectos de gran tamaño y que necesitan constantes cambios.
    - Proyectos donde sea importante el factor de riesgo.
    - Muy utilizado en sistemas orientados a objetos.



# 7. Metodologías de desarrollo del software

# Metodologías de desarrollo del software

- **Metodología**: conjunto de filosofías, fases, procedimientos, reglas, técnicas, herramientas …
- Una metodología puede seguir uno o varios modelos de ciclos de vida. El ciclo de vida indica qué es lo que hay que obtener, pero no cómo hacerlo.
- El ciclo de vida es el modelo general que se sigue en el desarrollo del software y la metodología establece las tareas y herramientas concretas.
- Las herramientas que permiten automatizar las tareas del desarrollo del software reciben el nombre de **Herramientas CASE** (**C**omputer **A**ided **S**oftware **E**ngineering).
- Una metodología incluye un conjunto de métodos para realizar cada una de las tareas necesarias para el desarrollo del software.

# Metodologías de desarrollo del software

- **Tipos de metodologías:**
  - **Convencionales:**
    - Principios del software. Condujeron a la crisis del software.
  - **Estructuradas:**
    - Como reacción a la crisis del software comienza la programación estructura, a la que siguió métodos de diseño y análisis estructurado que abarcan todo el ciclo de vida del software.
  - **Orientadas a objetos:**
    - A partir de la década de los 80 surgen los lenguajes orientados a objetos y más tarde, métodos para el diseño y análisis orientado a objetos.

En las metodologías estructuradas las aplicaciones se conciben como programas compuestos por módulos entre los que se realizan llamadas. En cambio, en las metodologías orientadas a objetos, la atencion se centra en los procesos, en los datos. Una aplicación consta de objetos, cada uno de los cuales tiene un estado definido por sus atributos y un comportamiento, definido por sus métodos.

# Metodologías de desarrollo del software

- Metodologías de desarrollo Ágil:

En 2001 un grupo de desarrolladores, liderados por Kent Beck, firmaron el manifiesto por el desarrollo ágil del software.

- Muy empleadas hoy día.
- El **objetivo** perseguido es el desarrollo de proyectos en poco tiempo.
- El objetivo es desarrollar productos y servicios de calidad que respondan a las necesidades de unos clientes cuyas prioridades cambian a una velocidad cada vez mayor.
- Para el desarrollo de proyectos que precisan de rapidez y flexibilidad.
- Se eliminan algunos procesos tediosos y se agilizan las fases de desarrollo, las iteraciones se hacen en **periodos cortos**, se desechan los riesgos y se da rápida solución a los problemas.
- El cliente se ve involucrado durante todo el desarrollo del proyecto, mientras que en las metodologías clásicas, solo lo hace en la primera fase de análisis

•https://www.wearemarketing.com/es/blog/metodologia-scrum-que-es-y-como-funciona.html
•https://www.iebschool.com/blog/metodologia-agil-agile-scrum/
•https://proyectosagiles.org/category/ejemplos/

# 8. Fases del desarrollo de una aplicación

# 1. Análisis

## OBJETIVOS DEL ANÁLISIS

Algunos autores suelen llamar a esta parte " Análisis de Requisitos " y lo dividen en cinco partes:

- Reconocimiento del problema.
- Evaluación y Síntesis.
- Modelado.
- Especificación.
- Revisión.

Identificación de Necesidades

Es el primer paso del análisis del sistema, en este proceso en Analista se reúne con el cliente y/o usuario (un representante institucional, departamental o cliente particular), e identifican las metas globales, se analizan las perspectivas del cliente, sus necesidades y requerimientos, sobre la planificación temporal y presupuestal, líneas de mercadeo y otros puntos que puedan ayudar a la identificación y desarrollo del proyecto.

# 1. Análisis

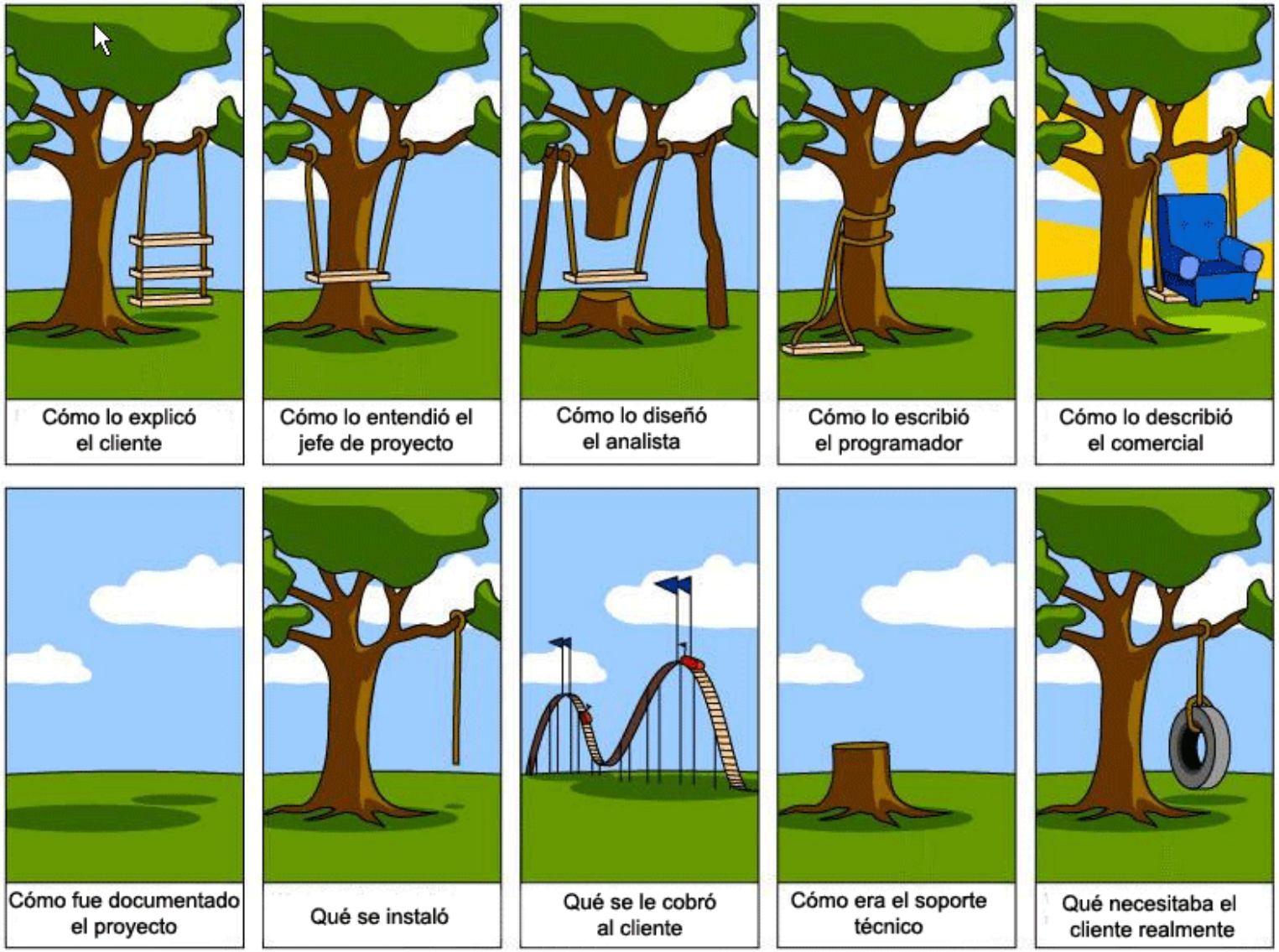
- **Objetivo:** entender, comprender y dar solución al problema → Especificar los requisitos funcionales.
- **Dificultades:**
  - El cliente puede no tener claros los requisitos de la aplicación
  - El cliente tiene claros los requisitos pero no sabe expresarlos
  - Pueden surgir nuevos requisitos
  - Pueden cambiar los requisitos ya especificados
  - Puede existir malos entendidos entre el equipo de desarrollo y el cliente por falta de conocimiento del equipo de desarrollo sobre el problema
- **Esencial:** Buena comunicación analista/programador y cliente/usuario
- Además hay otro tipo de **requisitos no funcionales:**
  - **Fiabilidad:** no existencia de fallos
  - **Escalabilidad:** capacidad del sistema ante aumentos de carga sin disminuir el rendimiento
  - **Extensibilidad:** capacidad del sistema para añadir nuevas funciones
  - **Seguridad** y **mantenibilidad.**

# 1. Análisis

- Tipos de requisitos
  - **Funcionales**. Describen la función del sistema, cómo reacciona ante determinadas entradas, cómo se comporta, qué resultados muestra…
  - **No funcionales**: características del sistema (características de ordenadores del sistema, sistema operativo, hardware, limitaciones, restricciones, …)

| Requisitos funcionales | Requisitos no funcionales |
|---|---|
| El usuario puede agregar un nuevo contacto | La aplicación debe funcionar en sistemas operativos Linux y Windows |
| El usuario puede ver una lista con todos los contactos | El tiempo de respuesta a consultas, altas, bajas y modificaciones ha de ser inferior a 5 segundos |
| A partir de la lista de contactos el usuario puede acceder a un contacto | Utilizar un sistema gestor de base de datos para para almacenar los datos |
| El usuario puede eliminar un contacto o varios de la lista | Utilizar un lenguaje multiplataforma para el desarrollo de la aplicación |
| El usuario puede modificar los datos de un contacto seleccionado de la lista | La interfaz de usuario es a través de ventanas, debe ser intuitiva y fácil de manejar |
| El usuario puede seleccionar determinados contactos | El manejo de la aplicación se realizará con el teclado y el ratón |
| El usuario puede imprimir la lista de contactos | Espacio libre en disco, mínimo: 1GB. Mínima cantidad de memoria 2GB |

# 1. Análisis

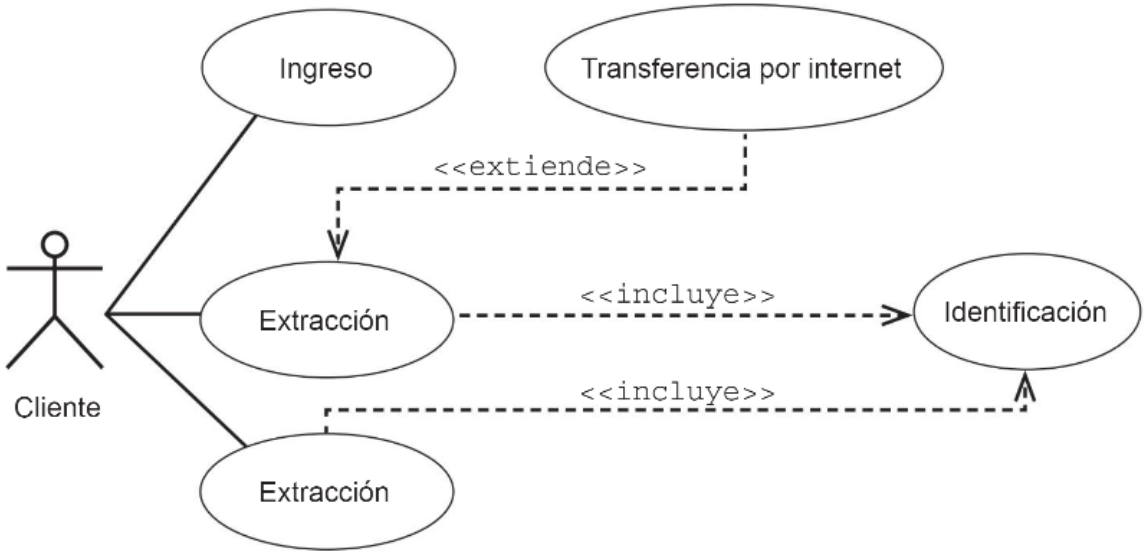

Cómo lo explicó el cliente
Cómo lo entendió el jefe de proyecto
Cómo lo diseñó el analista
Cómo lo escribió el programador
Cómo lo describió el comercial
Cómo fue documentado el proyecto
Qué se instaló
Qué se le cobró al cliente
Cómo era el soporte técnico
Qué necesitaba el cliente realmente

# 1. Análisis

- Herramientas para facilitar la comunicación con el cliente:
  - **Entrevistas**
  - **Prototipos.** Versión inicial del sistema.
  - **Reuniones de grupo** con el objetivo de generar ideas desde diferentes puntos de vista para la resolución del problema.
- Como resultado de esta etapa se obtiene la **especificación de los requisitos del software (ERS)**, que sirve para la siguiente etapa.

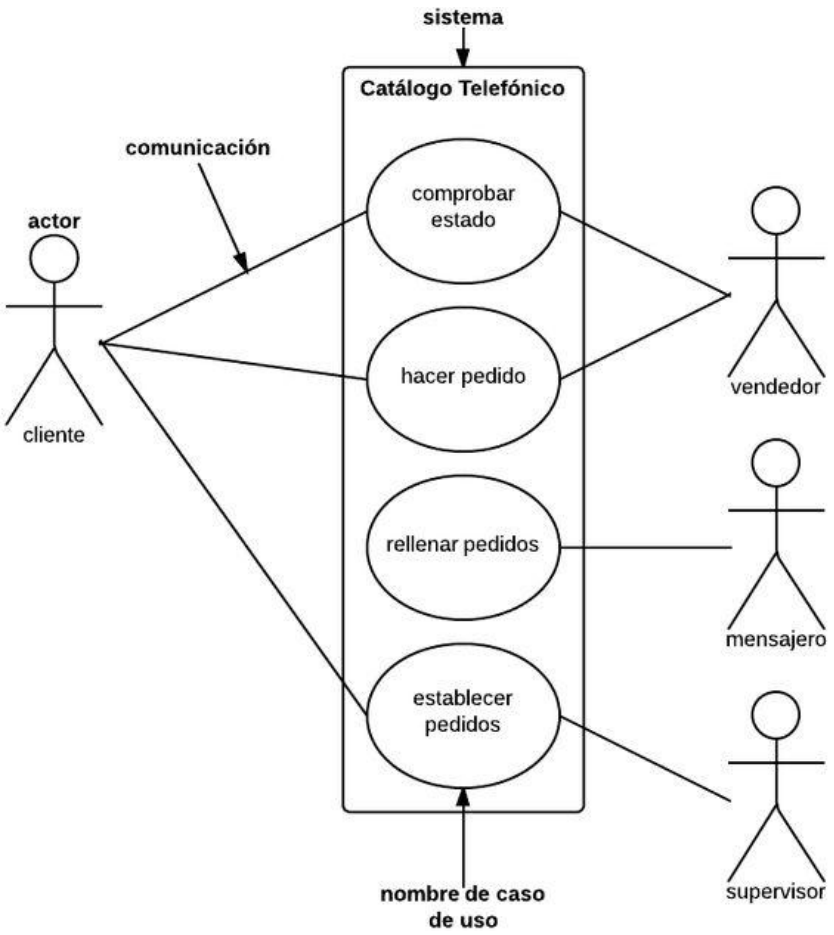
# 1. Análisis

- En la ERS se incluyen modelos gráficos con apoyos textuales. Estos modelos no son ambiguos a diferencia del lenguaje natural.
- En esta etapa se crean **diagramas de clases, diagramas de casos de uso.**



# 1. Análisis

- **Casos de uso:** se utilizan en la especificación de requisitos funcionales del sistema.
  - Utilizadas en metodologías OO
  - Cada caso de uso es una acción que realiza el sistema.
  - Consta de dos partes: una descripción de los casos de uso y un diagrama gráfico que muestra las relaciones entre los actores del sistema y los casos de uso.



# 1. Análisis

- **Resultado de esta etapa:**

Especificación de Requisitos del Software:

  - No ambigüedades
  - Completo
  - Consistente
  - Fácil de verificar
  - Fácil de modificar
  - Fácil de usar en Explotación y mantenimiento
  - Fácil de identificar el origen y las consecuencias de los requisitos

- Estructura del documento ERS propuesta por el IEEE, en última versión del estándar 830 es:

1. Introducción.
    - 1.1 Propósito.
    - 1.2 Ámbito del Sistema.
    - 1.3 Definiciones, Acrónimos y Abreviaturas.
    - 1.4 Referencias.
    - 1.5 Visión general del documento.
2. Descripción General.
    - 2.1 Perspectiva del Producto.
    - 2.2 Funciones del Producto.
    - 2.3 Características de los usuarios.
    - 2.4 Restricciones.
    - 2.5 Suposiciones y Dependencias.
    - 2.6 Requisitos Futuros.
3. Requisitos Específicos.
    - 3.1 Interfaces Externas.
    - 3.2 Funciones.
    - 3.3 Requisitos de Rendimiento.
    - 3.4 Restricciones de Diseño.
    - 3.5 Atributos del Sistema.
    - 3.6 Otros Requisitos.
4 Apéndices.

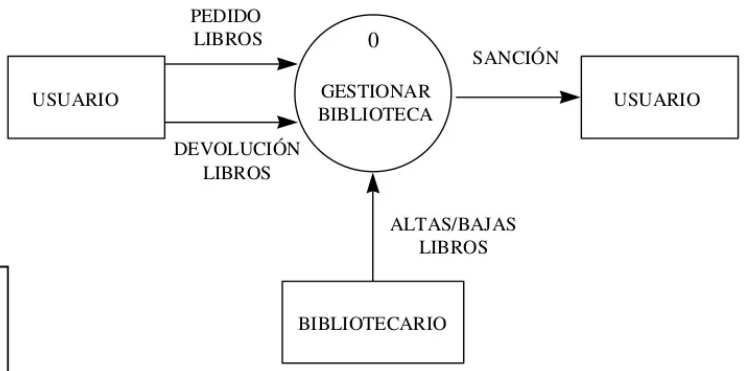
http://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf

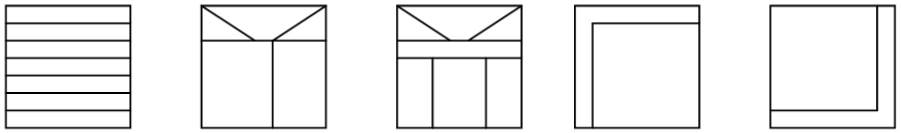
# 2.Diseño

- Partiendo de la ERS de la etapa anterior se determina cómo solucionar el problema. Para ello se partirá de los diagramas obtenidos en la etapa anterior, refinando algunos de ellos y se crearán otros que indiquen los pasos para dar respuesta a los requisitos establecido.
- Tipos de diseño:
  - **Diseño Estructurado (diseño clásico)**: basado en el flujo de datos del sistema.
  - **Diseño orientado a objetos:** es difícil. Aquí se definen todas las clases del sistema, las operaciones, y atributos asociados, las relaciones y comportamientos, así como las comunicaciones entre clases. En este caso se utiliza UML (lenguaje unificado de modelado).

# 2.Diseño

- **Diseño Estructurado:** se emplean herramientas gráficas como:
  - **Diagramas de flujo**
  - **Diagramas de cajas**
  - **Tablas de decisión**
  - **Pseudocódigo**




| Número de hijos | S | S | S | C | C | C | NG | NG | NG | NE | NE | NE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Nivel de estudios | P | M | S | P | M | S | P | M | S | P | M | S |
| Plus de primera | | | | | | X | | | X | | X | X |
| Plus de segunda | | | X | | X | | X | X | | X | | |
| Plus de tercera | | X | | X | | | | | | | | |
| Plus de cuarta | X | | | | | | | | | | | |

# 3. Codificación

- Transformación de las especificaciones del diseño en un conjunto de instrucciones escritas en un **lenguaje de programación** de alto nivel, almacenadas dentro de un programa (**código fuente**).
- Para obtener el código fuente se debe respetar la sintaxis y semántica del lenguaje de programación empleado.
- 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.

# 3. Codificación

- **Código Fuente**
  - Escrito por el programador; instrucciones de alto nivel empleando un determinado LP → debe ser traducido **a lenguaje máquina.**
  - Según tipo de licencia:
    - **Código fuente abierto**. Disponible para que cualquier usuario pueda estudiarlo, modificarlo o reutilizarlo.
    - **Código fuente cerrado**. No tenemos permiso para editarlo.

```
import java.util.Scanner;
public class AppCirculo {
   public static void main(String[] args) {
      float radio, superficie, circunferencia;
      System.out.println("Introduce el radio del
circulo");
      Scanner teclado = new Scanner (System.in);
      radio = teclado.nextFloat();
      superficie = (float) (Math.PI*radio*radio);
      circunferencia=(float) (2*Math.PI*radio);
      System.out.println("La superficie del círculo
es"+superficie);
      System.out.println("La circunferencia del círculo
es"+circunferencia);
   }

}
```

# 3. Codificación.

- A la hora de codificar se establecen ciertas reglas en relación a:
  - Comentarios
  - Declaraciones de variables y parámetros.
  - Nombres de clases, atributos, variables, constantes, etc
  - Sangrados para facilitar la legibilidad del código

# 3. Codificación

- **JDK o Java Development Kit.**
  - Proporciona herramientas de **desarrollo** para la creación de programas en Java.
  - No incluye herramienta gráfica.
  - Incluye el **JRE** (Java Runtime Environment): incluye como mínimo componentes para ejecutar una aplicación Java (máquina virtual y librerías de clase).
- Herramientas de consola incluidas en el JDK:
  - **java:** VM de java
  - **javac:** compilador
  - **javap:** desensamblador de clases
  - **jdb:** depurador de consola
  - **javadoc:** generador de documentación
  - **Appletviewer:** visor de Applets.
  - …

# 3. Codificación: Frameworks

- Estructura de ayuda al programador para desarrollar proyectos sin partir desde cero.
- **Ventajas** de utilizar un framework:
  - **Desarrollo rápido** de software.
  - **Reutilización** de partes de código para otras aplicaciones.
  - **Diseño** uniforme del software.
  - **Portabilidad** de aplicaciones de un computador a otro, ya que los bytecodes que se generan a partir del lenguaje fuente podrán ser ejecutados sobre cualquier máquina virtual.
- **Ejemplos:**
  - **Spring, Hibernate** para Java
  - **Django** para Phyton
  - **Angular** para JavaScript
  - **Symfony, Laravel** para PHP
  - …

# 3. Codificación: Frameworks

- **Inconvenientes:**
  - Gran dependencia del código respecto al framework utilizado.
  - La instalación e implementación del framework en nuestro equipo consume bastantes recursos del sistema.

https://openwebinars.net/los-10-mejores-frameworks- php-que-solicitan-las-empresas/

# 4. Pruebas

- El objeto de esta etapa es detectar errores antes de entregar el software al cliente.
- La estrategia es que dichas pruebas se apliquen desde los elementos más pequeños a los más grandes.
- El objetivo de las pruebas es: **verificar** y **validar** el software.
  - **Verificación**: determina si el software se ha construido correctamente, es decir, si las tareas que realiza las lleva a cabo de manera adecuada.
  - **Validación**: comprobar si el software es el que el usuario desea.
- **Tipos de Errores.** Según la fase en la que se detecten los errores pueden ser:
  - Errores de **compilación** (sintácticos).
  - Errores de **ejecución**. Más difíciles de detectar porque dependen de los datos de entrada.
  - Errores de **lógica**. Cuando se obtienen resultados incorrectos. Para detectarlos se emplean ejecuciones de prueba con varios grupos de datos de ensayo. Después se comprueban los resultados con los que se deben obtener.
  - Errores de **especificación**: El peor y el más costoso de corregir, y se debe a la realización de unas especificaciones incorrectas motivadas por una mala comunicación entre el analista y quien plantea el problema. Hay que repetir el trabajo.

# 4. Pruebas

- Tipos de pruebas:
  - **Pruebas de caja blanca o estructurales:** se basan el código fuente.
  - **Prueba de caja negra o funcionales:** se basan en las entradas que recibe el software y las salidas que debe producir. Su objetivo es validar los requisitos funcionales del software.
- Si se detectan errores entonces se procede a la **depuración del código**, que consiste en localizar el error y solucionarlo. Esto puede suponer realizar cambios en las etapas anteriores.

# 5. Documentación.

- Todas las etapas en el desarrollo de software deben quedar perfectamente documentadas.
- Uso:
  - Dar toda la información a los usuarios de nuestro software
  - Poder acometer futuras revisiones del proyecto
- Tres grandes documentos en el desarrollo de software
  - Guía técnica
  - Guía de uso
  - Guía de instalación

# 5. Documentación.

| | GUÍA TÉCNICA | GUÍA DE USO | GUÍA DE INSTALACIÓN |
|---|---|---|---|
| **Quedan reflejados:** | ✓ El diseño de la aplicación.<br>✓ La codificación de los programas.<br>✓ Las pruebas realizadas. | ✓ Descripción de la funcionalidad de la aplicación.<br>✓ Forma de comenzar a ejecutar la aplicación.<br>✓ Ejemplos de uso del programa.<br>✓ Requerimientos software de la aplicación.<br>✓ Solución de los posibles problemas que se pueden presentar. | Toda la información necesaria para:<br>✓ Puesta en marcha.<br>✓ Explotación.<br>✓ 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. Documentación

- Guía técnica
  - Para un correcto desarrollo y permitir mantenimiento
  - Uso por parte de los analistas y programadores.
  - Contiene el diseño, codificación y pruebas realizadas
- Guía de uso
  - Da a los usuarios finales información para su uso.
  - Uso por parte de los clientes (usuarios finales)
  - Contiene una descripción de la aplicación, ejecución, ejemplos de uso, requisitos del software, y solución de posibles problemas.
- Guía de instalación
  - Para garantizar la correcta instalación de la aplicación.
  - Personal informático responsable con los usuarios finales.
  - Contiene puesta en marcha, explotación y seguridad del sistema.

# 6. Explotación.

- La explotación es la fase en que los usuarios finales conocen la aplicación y comienzan a utilizarla.
- En esta fase se requiere la presencia del cliente.
- **Instalación:**
  - Transferencia de los programas al computador del usuario cliente, configuración y verificación.
  - Presencia del cliente
  - Se realizan las Beta Test (Prueba Beta).
- **Configuración.**
  - Asignación de los parámetros de funcionamiento normal de la empresa.
  - Prueba de la aplicación.
  - Posibilidad de la realización por parte del cliente con guía de instalación.
  - Posibilidad de programar la configuración de manera que se realice automáticamente tras instalarla.
- **Producción normal**
  - Explotación por parte del cliente

# 7. Mantenimiento.

- La etapa de mantenimiento es la más larga de todo el ciclo de vida del software.
- Los **tipos de cambios** del 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).

# Roles en el desarrollo del software

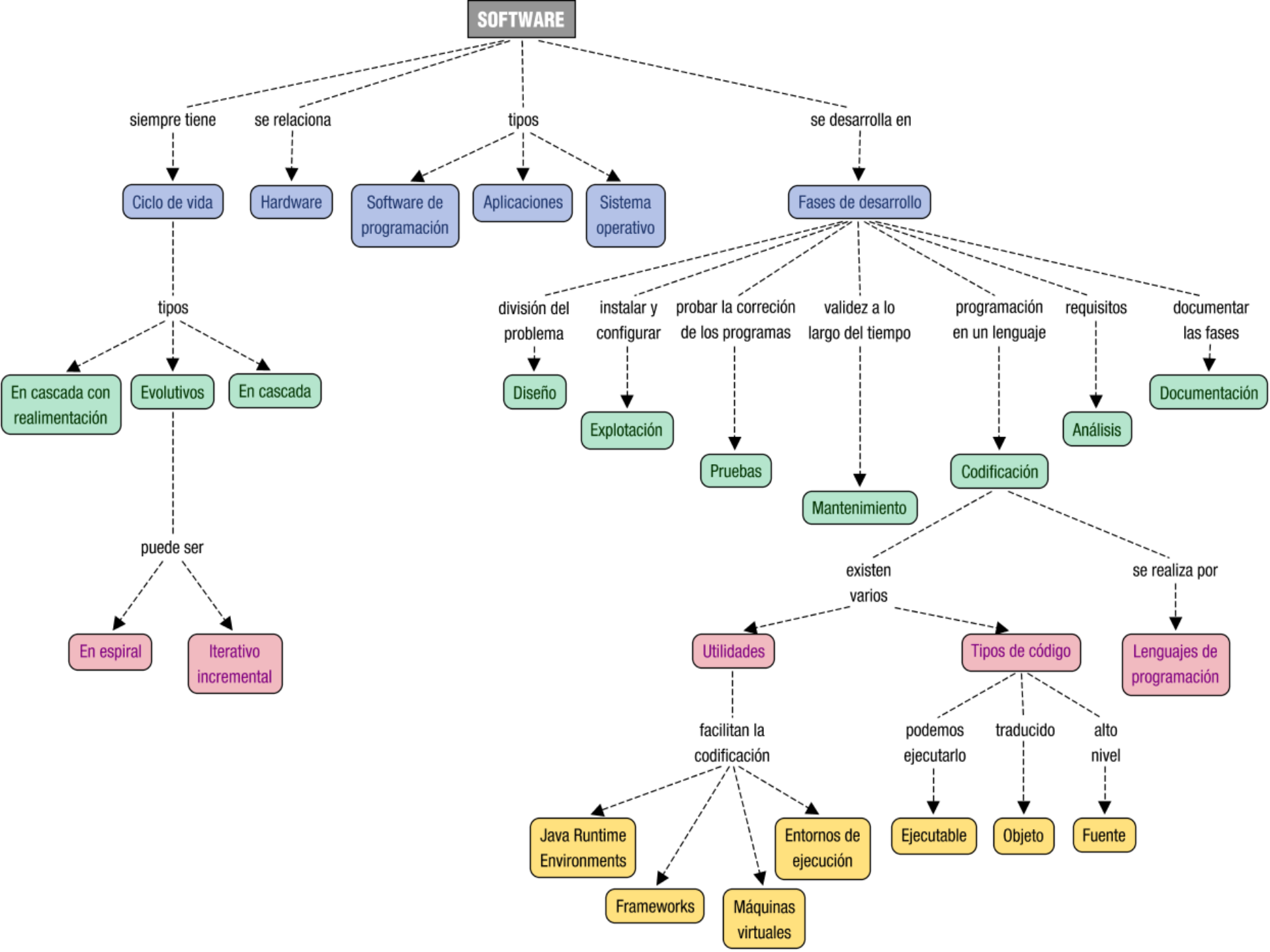
- El equipo de desarrollo se compone de una serie de roles:
  - **Jefe de proyecto**
    - Dirige el proyecto y es el máximo responsable. Puede ser un analista con experiencia, un arquitecto o una persona dedicada a ese puesto. Tiene que saber gestionar un equipo y los tiempos y tener una relación fluida con el cliente.
  - **Analista de sistemas**
    - Persona con experiencia que realiza un estudio exhaustivo del problema que ha de analizarse y ejecuta el análisis de todo el sistema. La experiencia es fundamental para establecer los requisitos de la aplicación con el cliente.
  - **Arquitecto de software**
    - A partir del trabajo realizado por el analista, debe definir las líneas del diseño, estableciendo la arquitectura del software, es decir, los componentes en que se divide. Esta arquitectura es tomada como referencia por el resto de personas desarrolladoras.
  - **Analista programador**
    - Es el diseñador. Parte de los requisitos definidos por el analista y la arquitectura del sistema y diseña las partes del mismo con un nivel de detalle suficiente para que el programador acometa su tarea.
  - **Programador**
    - Conoce en profundidad el lenguaje de programación y codifica las tareas que le ha asignado el analista o analista-programador.

# Roles en el desarrollo del software

- El equipo de desarrollo se compone de una serie de roles:
  - **Probador**
    - Se encarga de las tareas de pruebas y garantizar la calidad de la aplicación. Partiendo de los requisitos y de los programas los prueba para decidir si está en condiciones de ser entregados al cliente.
  - **Encargado de implantación**
    - Es el encargado de realizar el empaquetado de los programas y su instalación en el entorno del cliente.

# Calidad de los programas

- Las características generales que debe reunir un programa son:
  - **Legibilidad**: ha de ser claro y sencillo para una fácil lectura y comprensión.
  - **Fiabilidad**: ha de ser robusto, es decir, capaz de recuperarse frente a errores o usos inadecuados.
  - **Portabilidad**: su diseño debe permitir la codificación en diferentes lenguajes de programación, así como su instalación en diferentes sistemas.
  - **Modificabilidad**: ha de facilitar su mantenimiento, esto es, las modificaciones y actualizaciones necesarias para adaptarlo a una nueva situación. Relacionado con la legibilidad.
  - **Eficiencia**: se deben aprovechar al máximo los recursos de la computadora minimizando la memoria utilizada y el tiempo de proceso de ejecución. Esto hoy en día no se cumple: Windows, Office, …


SOFTWARE
siempre tiene
se relaciona
tipos
se desarrolla en
Ciclo de vida
Hardware
Software de programación
Aplicaciones
Sistema operativo
Fases de desarrollo
tipos
En cascada con realimentación
Evolutivos
En cascada
puede ser
En espiral
Iterativo incremental
división del problema
instalar y configurar
probar la correción de los programas
validez a lo largo del tiempo
programación en un lenguaje
requisitos
documentar las fases
Diseño
Explotación
Pruebas
Mantenimiento
Codificación
Análisis
Documentación
existen varios
se realiza por
Utilidades
Tipos de código
Lenguajes de programación
facilitan la codificación
podemos ejecutarlo
traducido
alto nivel
Java Runtime Environments
Entornos de ejecución
Frameworks
Máquinas virtuales
Ejecutable
Objeto
Fuente