Tipos de Software:

- **Software de Sistema**: Es **software**, es decir el conjunto de algoritmos intangibles. Es el tipo de software más cercano al hardware. (Sistema operativo, drivers, firmware…)

Firmware: Software de sistema muy cercano al hardware. ej. Cualquier cosa que tenga un microcontrolador tendrá firmware. Un programa muy simple y pequeño que está a muy bajo nivel.
Driver: Está en término de niveles entre el firmware y el sistema operativo. Es el controlador de algunos sistemas de hardware externos.

- **Software de Aplicación**: Es un software para funciones concretas usados por el usuario. (Suite ofimática, navegador, editor de imágenes…)
- **Software de Desarrollo**: Es el software usado para crear software, (Editores, compiladores, intérpretes…)

Relación hardware-software

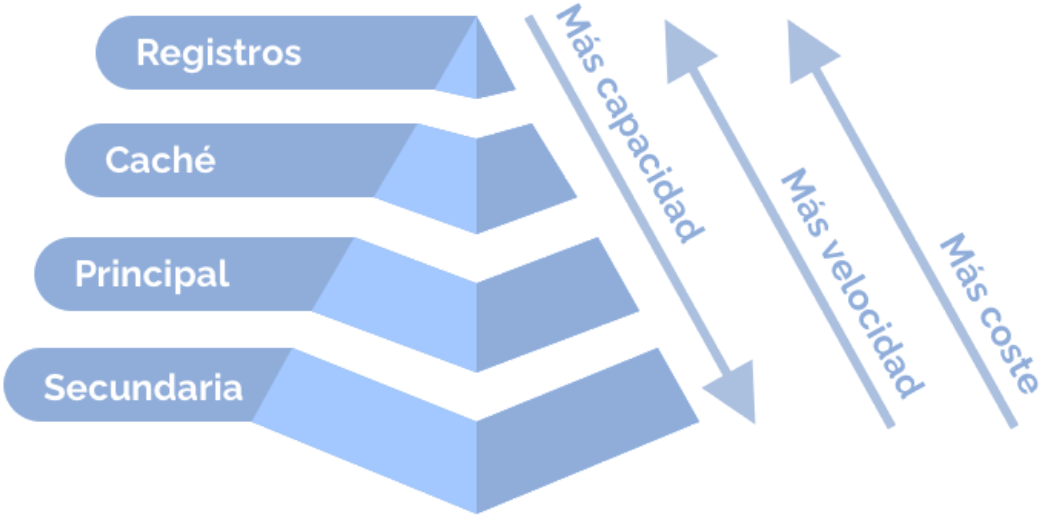
- **Microprocesador o procesador**: También llamado CPU (UCP en español) aunque de forma algo incorrecta ya que hoy en día un “micro” puede contener varias CPUs. Esta parte del ordenador se encarga de leer y ejecutar instrucciones, es el cerebro o el procesador, compuesto por una o varias Unidades Centrales de Procesamiento. Este componente es el más rápido, pero su memoria llamada caché es extremadamente limitada y volátil.

Hay varios tipos de caché. De la más rápida a la más lenta, de la más cercana a la ALU del micro a la más alejada y de la de menor capacidad a la de más son: L1, L2 y L3.
Los hilos de los cores de las CPUs buscan instrucciones para ejecutarlas en un orden óptimo.

- **Memoria principal o RAM**: Es un tipo de memoria volátil, es decir temporal, situada a nivel de procesos en el sistema informático entre la memoria secundaria y el microprocesador. Esta memoria es mucho más rápida que la memoria principal, pero de mucha menor capacidad, y es utilizada por el microprocesador para almacenar temporalmente los datos que pueda necesitar. La capacidad de la memoria principal es mucho mayor que la de la caché, pero también es mucho más lenta que esta.
- **Memoria secundaria**: Este tipo de memoria es permanente, por lo que no se borra al apagar el sistema. Tiene una capacidad mucho más masiva pero mucha menor velocidad, y se usa para guardar el groso de la información y datos que no se vayan a procesar en breves. Los sistemas operativos tienen la posibilidad de usar parte de la memoria secundaria como memoria principal creando una suerte de partición, pero es algo que ralentiza gravemente los procesos y solo se hace si la memoria

principal no tiene más espacio libre y no queda más remedio. (Disco duro, disco SSD…)

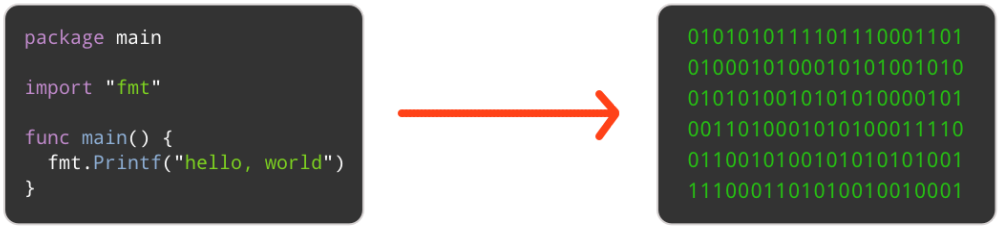
- **E/S o I/O**: Es la entrada y salida para los periféricos utilizados para la entrada y salida de datos.



Tipos de código en un lenguaje compilado

En los lenguajes interpretados sólo hay código fuente.

- **Código fuente**: Archivo de texto entendible por un ser humano usado para construir el programa.
- **Código objeto**: Es un código intermedio. Es un archivo en binario y se genera a partir del código fuente. A partir de este se genera el código ejecutable.
- **Código ejecutable**: Lo puede correr el ordenador, es un archivo binario, se puede ejecutar, se genera a partir del código objeto, se comienza a ejecutar en la CPU o micro.



Lenguaje de alto nivel que entiende el programador

Lenguaje de máquina que entiende el procesador

El código objeto y el código ejecutable no son portables.
¿Qué son los lenguajes compilados? Deben ser compilados por los compiladores. Al ser compilados se crea un código objeto o código compilado que sólo sirve para un hardware concreto y que es fácil de leer para la máquina pero difícil para el humano, de bajo nivel. (C++, C…)
Un lenguaje interpretado sin embargo va compilando y ejecutando las líneas una a una sin generar código objeto. (PHP, Javascript…)

Java es especial, es un lenguaje que se compila y después se interpreta. Se compila en una máquina virtual, por lo que teniendo la máquina virtual instalada el código objeto es portable, y después se interpreta el código objeto de la máquina virtual, que es más rápido que interpretar el código fuente. El código objeto de la máquina virtual en Java se llama Bytecode.

Ciclo de vida del software

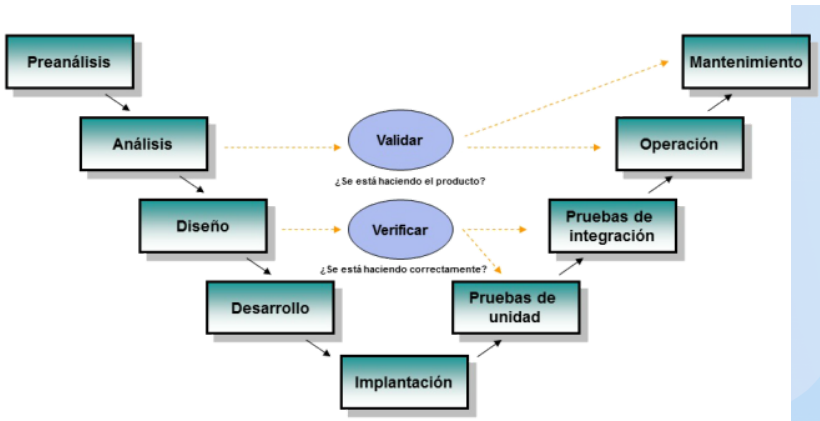
- **Análisis**: Qué necesito. Qué se necesita, cuáles son los requisitos. Los requisitos deben pedirse al cliente. Los requisitos deben ser completos, conciso, con lenguaje formal sin ambigüedades, poco detallado, entendible por el cliente, debe separar los requisitos funcionales y no funcionales. Debe dividir y jerarquizar el modelo y establecer criterios de validación.
  El resultado del análisis deben ser las especificaciones de requisitos del software.
- **Diseño**: Cómo lo vamos a hacer, la arquitectura también, ¿Cuál es la solución? Se le proponen al cliente varias soluciones. Se descompone y organiza el sistema en elementos y componentes que puedan desarrollarse en paralelo. Se especifica a interrelaciones y funcionalidades de los componentes. Las actividades habituales del diseño son: El diseño arquitectónico, el diseño detallado, el diseño de datos y el diseño de interfaz de usuario.
  El resultado del diseño arquitectónico es el documento de arquitectura del software.
  El resultado del diseño detallado son las especificaciones de módulos y funciones.
  Mockup: Un diseño estático como prueba del diseño final.
- **Codificación**: Picar código. Se escribe el código fuente de cada componente.
  El resultado de la codificación es el código fuente.
- **Pruebas**: Definir las pruebas y realizarlas. Descubrir malos funcionamientos y errores sometiendo al sistema a las máximas situaciones. Hay pruebas de unidades, de integración y de sistema.
  El resultado de las pruebas de unidades son los módulos utilizables.
  El resultado de las pruebas de integración es el sistema utilizable.
  El resultado de las pruebas de sistema es el sistema aceptado.
- **Documentación y mantenimiento**: Documentar, actualizar, parchear. Al pasar el tiempo el software necesitará cambios Tipos de mantenimiento: Correctivo, preventivo, perfectivo, evolutivo y el adaptativo (Este es para adaptar el programa a otros medios).
  El resultado de la documentación es la documentación técnica y de usuario.
  El resultado del mantenimiento es el informe de errores y el control de cambios.

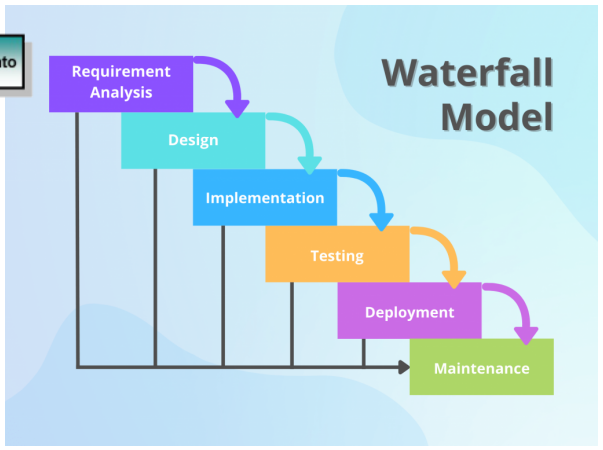
Modelos de desarrollo del software

- Modelos clásicos o predictivos: Modelo en cascada, modelo en V.
- Modelo de construcción de prototipos.
- Modelos evolutivos o incrementales: Modelo en espiral o iterativo, metodologías ágiles o adaptativas.

Modelo en cascada

Uno a uno todos los pasos del desarrollo. Véase figura inferior derecha.





Modelo en V

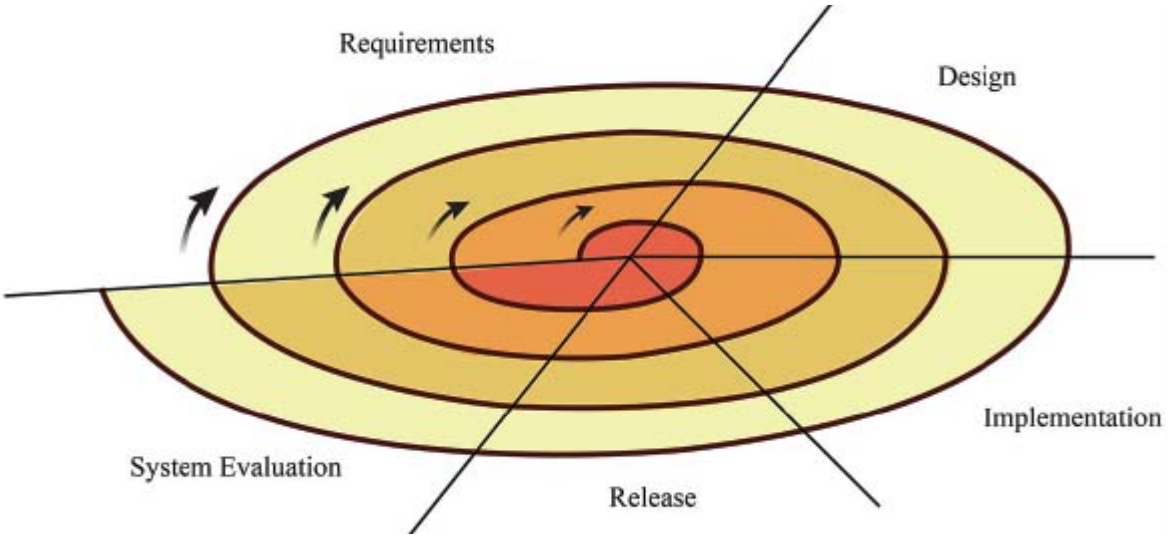
Los pasos del desarrollo se van mezclando o intercalando para ver los problemas antes. Véase figura superior izquierda.

Prototipos

Se crean en la fase de análisis y pueden rehacerse hasta estar enfocados correctamente. Hay prototipos rápidos hechos con código hecho para hacer prototipos o sin código incluso y prototipos lentos que se hacen en el código del programa final.

Modelo en espiral

Se vuelven a trabajar las partes que no sirven una y otra vez hasta que funcionen



Metodologías ágiles

Las metodologías ágiles son metodologías de trabajo que se basan en subdividir el desarrollo para hacerlo más manejable, crear una organización fluida y flexible y plantear unos principios que propicien la eficiencia y mejora en el tiempo.
Las más conocidas son Kanban, Scrum, XP (eXtreme Programming)
Estas metodologías son iterativas e incrementales, osea que van poco a poco añadiendo funcionalidades en unidades de tiempo de trabajo.
Los requisitos y soluciones pueden cambiar a lo largo del tiempo, lo cuál es algo que las metodologías ágiles admiten.
Estas metodologías permiten tener un seguimiento exhaustivo de las historias, cómo hacerlas, sus etapas, de las modificaciones, quienes la hacen y el historial no versiones.

Scrum

Suele usarse en desarrollos de software. Es útil para gestionar proyectos. Usa las historias de usuario (las subdivisiones útiles mínimas de los requisitos del proyecto) y los sprint (iteraciones de trabajo de 2 a 4 semanas a los que se asigna una historia a cada equipo que debe terminar antes de que termine la iteración y que tiene como resultado una entrega parcial entregable), los roles (posiciones y responsabilidades dentro de la metodología scrum) y las reuniones (que sirven para organizar todo, ponerse al día, comunicarse con el cliente y ver cómo va el proyecto o cómo cambia).
Los roles son Scrum master, Miembro de un equipo, product owner.
El Scrum master se encarga de que la metodología se aplique correctamente.
Los equipos son autogestionados y multidisciplinarios.
El product owner comunica con el cliente y es representante del mismo.
Las reuniones se hacen antes de empezar cada proyecto, entre sprints, cada día , en momentos extraordinarios y después de cada proyecto. En las reuniones que se hacen cada sprint hay comunicación con el cliente. Las reuniones diarias o Daily no duran más de 15 min. Después de cada proyecto y antes del siguiente se hace una retrospectiva a cosas a mejorar llamada Scrum retrospective. Este mirar al pasado por cosas que mejorar es retrofeedback.
Al comenzar un proyecto este se divide en requisitos y estos se organizan en historias. Estas historias se ponen en el product backlog, y en cada sprint un equipo saca y se asigna

una historia a completar ese sprint poniéndola en el sprint backlog. Esto último, la planificación de las historias que los distintos equipos van a realizar en el sprint se llama Sprint planning.

Kanban

Esta metodología es muy similar a Scrum, aunque tiene algunas diferencias importantes. En Kanban se establecen estados en los que pueden estar las historias, por ejemplo, “Pendiente”, “En proceso” y “Terminado”. Tiene un flujo de trabajo continuo, sin iteraciones. Las historias se ordenan por urgencia e importancia. Hay mucha comunicación con el cliente. Tiene mucha flexibilidad.

XP

Tiene 4 valores y 12 prácticas. Los valores son: Simplicidad, Comunicación, Retroalimentación, Valentía y Respeto. Sus características son que tiene un diseño sencillo, tiene mejoras sencillas continuas, posee pruebas y refactorización, usa la integración continua, se utiliza la programación por parejas en iteraciones semanales de 40 horas, el cliente se integra en el equipo de desarrollo y usa estándares de codificación.

Lenguajes de programación

Para obtener un código ejecutable podemos interpretar o compilar según qué tipo de código sea.
Script = Código interpretado

Paradigmas de los lenguajes de programación

Los lenguajes pueden ser declarativos (Que no especificas los pasos para obtener los resultados) e imperativos (Que sí especifican)
Los lenguajes declarativos suelen ser interpretados y pueden ser lógicos (Que usan reglas) como prolog, funcionales (Que usan funciones) como Lisp o Haskell, o algebraicos (Que usan sentencias) como SQL.
Los lenguajes declarativos suelen ser compilados y pueden ser estructurados como C, orientados a objetos como Java o multiparadigma como C++ o Javascript.
Los lenguajes pueden ser de alto nivel (C++, Java), de medio nivel(C) o de bajo nivel (ensamblador)
El lenguaje ensamblador trabaja directamente sobre el microprocesador.

Compilación/Interpretación

Al compilar o interpretar se hace primero el análisis léxico y después el análisis sintáctico, y si en los análisis no se detecta ningún error se genera el código objeto.
Los lenguajes compilados como C o C + + son muy eficientes, pero hace falta compilar cada vez que el código fuente es modificado.
Los lenguajes interpretados como PHP o Javascript no generan código objeto . Se ejecuta directamente pero es menos eficiente.

Java

Es un lenguaje de programación compilado e interpretado.
El código fuente genera un código objeto llamado bytecode que es un código binario intermedio.
El bytecode puede considerarse un código objeto pero para la máquina virtual en vez de para la máquina nativa.
El bytecode se interpreta desde la máquina virtual.
Por esto en Java debes instalar su máquina virtual.

Java es un lenguaje orientado a objetos. Es fácil de aprender, es bueno para documentar y tiene una comunidad activa.