**El ordenador tiene dos partes diferenciadas: hardware y software**

•Según su función se distinguen tres tipos de software:

**- Sistemas:**

**- Software de programación:** IDE ENTORNOS DE DESARROLLO INTEGRADOS.

**-Aplicaciones informáticas:** son un conjunto de programas que tienen una finalidad más o menos concreta.

**•Hardware:** conjunto de dispositivos físicos que conforman un ordenador.

El software se ejecutará sobre los dispositivos físicos

**-Hardware:**

CPU-(cerebro)

MEMORIA PRINCIPAL O RAM

UNIDAD DE ENTRADA/SALIDA

Esta relación software-hardware se puede ver desde dos puntos de vista:

**● Desde el punto de vista del sistema operativo (SO)**

Todas las aplicaciones necesitan recursos hardware durante su ejecución

Será siempre el sistema operativo el encargado de controlar todos estos

aspectos de manera "oculta" para las aplicaciones (y para el usuario).

**● Desde el punto de vista de las aplicaciones**

**Aplicación:** programas, escritos en algún lenguaje de programación

lenguajes de programación

**El hardware:** de interpretar señales eléctricas

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

(un proceso de traducción de código)

**-Desarrollo del software:** Todo el proceso que ocurre desde que se concibe una idea hasta que un programa está implementado en el ordenador y funcionando.

**-Ciclo de vida del software:** Describe el desarrollo de software desde la fase inicial hasta la fase final, proponiendo etapas que sirven como referencia para realizar este proceso.

**•Las fases que conforman el ciclo de vida son:**

• Análisis

• Diseño

• Desarrollo o codificación

• Pruebas

• Implantación

• Mantenimiento

**1. Modelo en cascada**

-Pequeños desarrollos, ya que las etapas pasan de una a otra sin retorno posible.

**2. Modelo en cascada con retroalimentación**

-Es uno de los modelos más utilizados. Proviene del modelo anterior, pero se introduce una realimentación entre etapas, de forma que podamos volver atrás en cualquier momento para corregir, modificar o depurar algún aspecto.

**-Ventajas:** fácil de comprender, de planificar y de seguir, y existen herramientas que lo soportan.

**-Inconvenientes:** los cambios generan confusiones conforme el proyecto avanza, es difícil para el cliente especificar todos los requisitos al principio, cuanto más tarde se detecte un fallo más costoso serán hacer los cambios que implique (rehacer todas las fases).

**3 Modelos de proceso incremental**

-Para resolver el problema del tiempo del modelo en cascada, podemos ir haciendo entregas con una funcionalidad parcial del software al cliente y aumentarla en entregas posteriores.

**Modelo iterativo incremental**

-Es decir, se realiza el modelo anterior (cascada) eliminando la retroalimentación y haciendo los cambios necesarios en cada incremento y entrega.

**4. Modelos evolutivos**

-Son más modernos que los anteriores. Tienen en cuenta la naturaleza cambiante y evolutiva del software.

**4.1 Construcción de prototipos**

-En la práctica, la construcción de prototipos ayuda al ingeniero de sistemas y al cliente a entender de mejor manera cuál será el resultado de la construcción cuando los requisitos estén satisfechos.

**-Desventajas:** El cliente considera la mayoría de las veces al prototipo como el producto final.

Es conveniente seguir este modelo cuando el cliente es capaz d definir un conjunto de objetivos generales para el software, pero no puede identificar los requerimientos en detalle o cuando los desarrolladores tienen dudas considerables.

**4.2 Modelo en espiral**

Es una combinación del modelo anterior con el modelo en cascada añadiendo un nuevo elemento, el análisis de riesgo.

**•Siempre hay una serie de etapas que debemos seguir para construir software fiable y de calidad.**

**• Análisis de requisitos:** Se especifican los requisitos funcionales y no funcionales del sistema.

**• Diseño:** Se divide el sistema en partes y se determina la función de cada una.

**• Codificación:** Se elige un lenguaje de programación y se codifican los programas.

**• Pruebas:** Se prueban los programas para detectar errores y se depuran.

**• Documentación:** De todas las etapas, se documenta y guarda toda la información.

**• Explotación:** Instalamos, configuramos y probamos la aplicación en los equipos del cliente.

**• Mantenimiento:** Se mantiene el contacto con el cliente para actualizar y modificar la aplicación el futuro.

**• Autoevaluación**

**4.1. ANÁLISIS.**

**¿Cómo se hace?**

**•Mediante:**

• entrevistas

• dinámicas de grupos estructuradas (JAD, Joint Application Development)

• planificación conjunta de requisitos (dinámicas JAD dirigidas a la alta dirección)

• Brainstorming

• Prototipos (versiones iniciales del sistema)

• Casos de uso. Técnica UML (representación de escenarios que describan lo que hace el sistema, no cómo lo hace)

**•Se especifican dos tipos de Requisitos:**

**•Funcionales, que detallan:**

• Qué funciones tendrá que realizar la aplicación.

• Qué respuesta dará la aplicación ante todas las entradas.

• Cómo se comportará la aplicación en situaciones inesperadas.

**•No funcionales, que tratan sobre las características del sistema:**

• Tiempos de respuesta del programa,

• Legislación aplicable,

• Fiabilidad

• Mantenibilidad

• Sistema operativo

• Restricciones

**TÉCNICAS DE REPRESENTACIÓN DE LOS REQUISITOS**

**• Diagramas de flujo de datos, DFD:** representa el flujo de datos entre los diferentes procesos, entidades externas y almacenes que forman el sistema.

**• Diagramas de flujo de control, DFC:** similar, pero muestra el flujo de control en lugar de los datos (es decir el orden en que se realizan los procesos)

**• Diagramas de transición de estados, DTE:** representa cómo se comporta el sistema como consecuencia de sucesos externos.

**• Diagrama Entidad / Relación, DER:** representa los datos y la forma en que se relacionan entre ellos (recordar lo visto en Bases de Datos)

**• Diccionario de datos, DD:** es una descripción detallada de los datos utilizados por el sistema que gráficamente se encuentran representados en el DFD.

**•Especificación de Requisitos del Software (ERS), que:** no debe tener ambigüedades, debe ser completo, consistente, fácil de verificar y modificar, fácil de utilizar y fácil de identificar el origen y las consecuencias de los requisitos.

**4.2. DISEÑO.**

**Donde ya sabemos lo que hay que hacer, el siguiente paso es: ¿Cómo hacerlo?**

Tenemos que traducir los requisitos en una representación de software.

**•Hay dos tipos de diseño:**

**• Estructurado:** basado en el flujo de datos a través del sistema

**• Orientado a objetos:** el sistema se entiende como un conjunto de objetos que tienen propiedades y comportamientos, existen eventos que activan operaciones que modifican el estado de dichos objetos y los objetos interactúan con otros.

--------------------------------------------------------------------------------------------------------------------

**• Diseño a nivel de componentes o diseño procedimental:** El diseño a nivel de componentes se presenta a menudo después que se ha terminado la primera iteración del diseño arquitectónico y el objetivo de esta fase es traducir el diseño en software operacional.

**• Construcciones lógicas para diseñar cualquier programa:**

**-Secuencial:** Implementa los pasos del proceso esenciales para la especificación

de cualquier algoritmo.

**-Condicional:** Permite seleccionar un proceso u otro a partir de la evaluación de

una condición lógica.

**-Selección múltiple:** Es una extensión de la anterior. Decisiones sucesivas hasta que

ocurre una condición verdadera