INTRODUCCIÓN AL DESARROLLO DEL SOFTWARE

2.1.- CONCEPTO DE CICLO DE VIDA

El problema más importante en cualquier departamento de sistemas de información de una empresa es definir un marco de eferencia común que:

- Pueda ser empleado por todas las personas que participan en el proceso informático.
- Defina los procesos, las actividades y las tareas a desarrollar.

Acorde con lo anterior, se define por **ciclo de vida del software** al conjunto de fases por la que pasa a lo largo del tiempo, desde la fase de estudio y concepción hasta la de realización, explotación y mantenimiento.

Diversas organizaciones profesionales (IEEE o ISO/IEC) han publicado normas relativas al ciclo de vida del software. Concretamente:

- La norma IEE 1074 entiende por **ciclo de vida del software** "una aproximación lógica ala adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del software".
- La norma ISO 12207-1 entiende por **modelo de ciclo de vida** "un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso".

Ambas normas consideran una actividad como un conjunto de tareas u una tarea como una acción que transforma entradas en salidas.

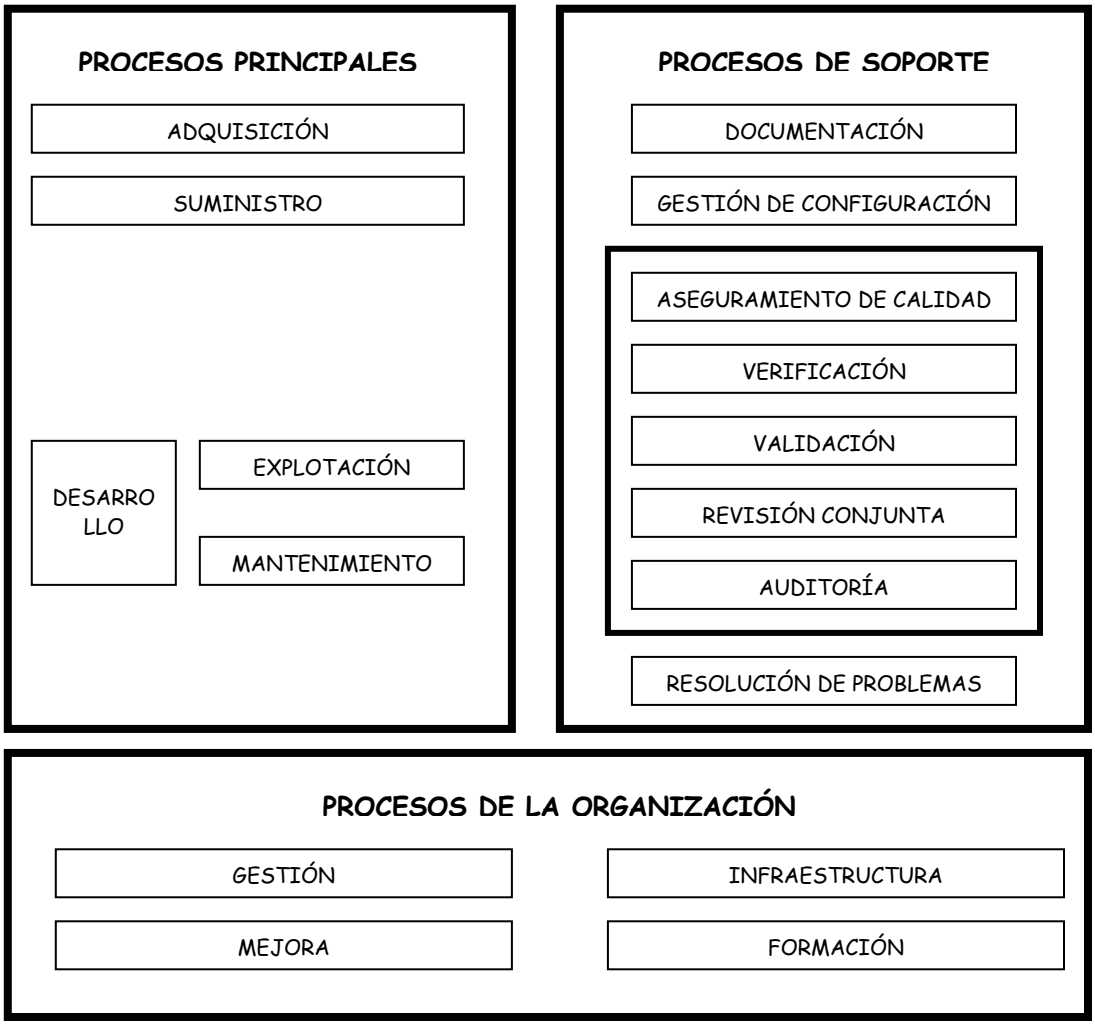
El ciclo de vida abarca por tanto toda la vida del sistema, desde su concepción hasta que deja de utilizarse, por eso, a veces se habla de **ciclo de desarrollo** como el subconjunto del anterior que empieza en el análisis y finaliza con la entrega del sistema al usuario.

2.2.- PROCESOS DEL CICLO DE VIDA SOFTWARE

La norma ISO 12207 - 1 establece las actividades a realizar durante el ciclo de vida software, agrupadas según en procesos según la figura, indicando que la norma no fomenta ni especifica ningún modelo concreto de ciclo de vida, gestión de software o método de ingeniería ni establece cómo realizar ninguna de las actividades.

Dichas actividades se agrupan en cinco procesos principales, ocho procesos de soporte y cuatro procesos de la organización.

A continuación se analizan cada uno de estos procesos para posteriormente resumir los principales modelos de ciclo de vida.



2.2-1.- Procesos principales

Son aquellos que resultan útiles a las personas que inician o realizan el desarrollo, la explotación o el mantenimiento del software durante su ciclo de vida.

2.2-1-1.- Proceso de adquisición

Son las actividades y tareas que el comprador, el cliente o el usuario realiza para adquirir un sistema, un producto o un servicio software:

- ★ Preparación y publicación de solicitud de ofertas
- ★ Selección del suministrador
- ★ Gestión de procesos desde la adquisición hasta la aceptación del producto
- ★ Etc.

2.2-1-2.- Proceso de suministro

Son las actividades y tareas que el suministrador realiza:

- ★ Propuesta para responder a la petición de un comprador
- ★ Firma de contrato de suministro del producto
- ★ Identificación de recursos y procedimientos para garantizar el éxito del proyecto
- ★ Desarrollo de los planes del proyecto

- ★ Ejecución de dichos planes
- ★ Entrega del producto al comprador

2.2-1-3.- Proceso de desarrollo

Tradicionalmente, el proceso de desarrollo se ha distribuido por fases como indica la figura:

Actualmente, en proceso de desarrollo tradicional desglosa sus fases en actividades, siendo las más principales las siguientes:

- ★ Análisis de los requisitos del sistema (Estudio de viabilidad, requisitos de seguridad, interacción hombre – máquina, interfaces, restricciones aplicables al diseño, etc.)
- ★ Diseño de la arquitectura del sistema (Componentes de hardware y software, operaciones manuales del sistema, etc.)
- ★ Análisis de requisitos de software
- ★ Diseño de la arquitectura del software
- ★ Diseño detallado del software
- ★ Codificación y prueba de software
- ★ Integración del software
- ★ Prueba del software
- ★ Integración del sistema
- ★ Prueba del sistema
- ★ Instalación del software en el entorno de explotación final donde vaya a funcionar
- ★ Aceptación del software por parte del comprador

2.2-1-4.- Proceso de explotación

También Llamado *proceso de operación*, incluye la explotación del software y el soporte operativo a los usuarios.

2.2-1-5.- Proceso de mantenimiento

Es el proceso que aparece cuando el software necesita modificaciones, ya sea en código o en la documentación aportada

2.2-2.- Procesos de soporte

Son procesos de apoyo al resto de los procesos y se aplican en cualquier punto del ciclo de vida del software. Son los siguientes:

2.2-2-1.- Proceso de documentación

Registra la información producida por un proceso o actividad del ciclo de vida. El proceso permite planificar, diseñar, desarrollar, producir, editar, distribuir, y mantener los documentos necesarios para todas las personas involucradas en el software.

2.2-2-2.- Proceso de gestión de la configuración

Aplica procedimientos administrativos y técnicos durante todo el ciclo de vida del sistema para:

- ★ Identificar, definir y establecer la línea base de los elementos de configuración del software del sistema.
- ★ Controlar las modificaciones y las versiones de los elementos
- ★ Registrar el estado de los elementos y las peticiones de modificación
- ★ Asegurar la complexión, la consistencia y la corrección de los elementos
- ★ Controlar el almacenamiento, la manipulación y la entrega de los elementos

2.2-2-3.- Proceso de aseguramiento de la calidad

Garantiza que los procesos y los productos software del ciclo de vida cumplen los requisitos especificados y cumplen con los planes establecidos.

2.2-2-4.- Proceso de verificación

Verifica si los requisitos de un sistema o del software están completos y son correctos y si los productos software de cada fase del ciclo de vida cumplen los requisitos impuestos sobre ellos en las fases previas.

2.2-2-5.- Proceso de validación

Determina si el sistema o software final cumplen los requisitos previos para su uso.

2.2-2-6.- Proceso de revisión conjunta

Sirve para evaluar el estado del software y sus productos en una actividad del ciclo de vida o una fase de un proyecto.

2.2-2-7.- Proceso de auditoría

Permite establecer en momentos predeterminados si se han cumplido los requisitos, los planes y el contrato.

2.2-2-8.- Proceso de resolución de problemas

Permite analizar y eliminar los problemas (disconformidades con los requisitos o el contrato) descubiertos durante el desarrollo, la explotación, el mantenimiento u otro proceso.

2.2-3.- Procesos de Organización

Son los procesos que emplea una organización para llevar a cabo funciones tales como la gestión, la formación del personal o la mejora del proceso. Suelen llevarse a cabo en el ámbito organizativo, fuera del ámbito de proyectos y contratos específicos.

2.2-3-1.- Proceso de gestión

Comprende las actividades y tareas genéricas que puede emplear una organización que tenga que gestionar sus procesos (planificación, seguimiento y control, evaluación, revisión, etc.)

2.2-3-2.- Proceso de infraestructura

Establece la infraestructura necesaria para cualquier otro proceso (hardware, software, herramientas, normas, técnicas e instalaciones para el desarrollo, la explotación o el mantenimiento).

2.2-3-3.- Proceso de mejora

Establece, valora, mide, controla y mejora los procesos del ciclo de vida del software.

2.2-3-4.- Proceso de formación

Proporciona formación y mantiene al personal formado, incluye, por tanto, el desarrollo del material de formación y la implementación de un plan de formación.

2.3.- MODELOS DE DESARROLLO

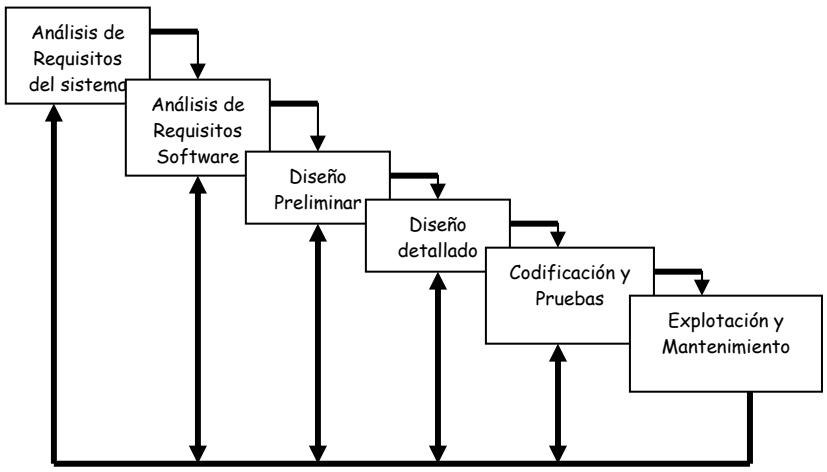
Para desarrollar el ciclo de vida se han propuesto distintos modelos de desarrollo, tanto para sistemas convencionales como para sistemas orientados al objeto. En el primer caso se considera el *modelo en cascada o Waterfall* propuesto por Royce (1970) y refinado por distintos autores (Boehm (1981), Sommerville (1985), Sigwart (1990), etc.). El *modelo incremental* (Lehman (1984) y Amescua (1995) y el *modelo en espiral* (Boehm (1988)). Por su parte, para sistemas orientados al objeto se considera el *modelo de agrupamiento o cluster* (Meyer (1990)), el *modelo fuente* (Henderson, Sellers y Edwards (1990), el *modelo remolino* (Rumbaugh (1992)) y el *modelo pinball* (Ambler (1994).

A continuación se hace una somera descripción de cada modelo.

MODELOS PARA DESARROLLO DE SISTEMAS CONVENCIONALES

2.3-1.- Modelo en cascada

El modelo consta de un número variable de fases o etapas que se proponen para el ciclo de vida del sistema pero que generalmente suelen ser las indicadas en la figura:



Las características del ciclo según este modelo son:

- ★ Cada fase empieza cuando ha terminado la fase anterior.
- ★ Para pasar de una fase a otra es preciso conseguir todos los objetivos de la etapa previa
- ★ Ayuda a prevenir que se sobrepasen las fechas de entrega y los costes esperados.
- ★ Al final de cada fase, el personal técnico y el usuario tienen la oportunidad de revisar el progreso del proyecto.

Aunque es el ciclo de vida más antiguo y el mas utilizado, tiene los siguientes inconvenientes:

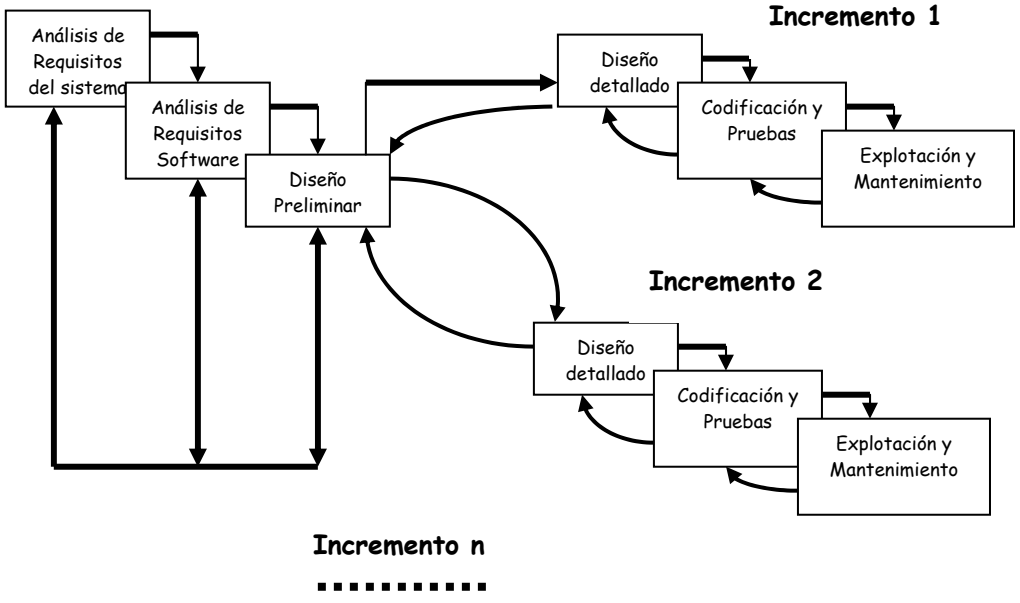
- ★ No refleja el proceso "real" de desarrollo del software. (Los proyectos reales raramente tienen un flujo secuencial dado que siempre hay iteraciones que pueden cubrir mas de una etapa.
- ★ Se tarda mucho tiempo en recorrer todo el ciclo dado que hasta que no se finaliza una fase no se pasa a la siguiente.
- ★ Acentúa el fracaso de la industria del software con el usuario final puesto que el sistema no estará en funcionamiento hasta las fases finales del proyecto.

2.3-2.- Modelo incremental

Corrige la necesidad de una secuencia no lineal de pasos de desarrollo. Según dicho modelo, se va creando el sistema software añadiendo componentes funcionales al sistema (llamados *incrementos*). En cada paso sucesivo se actualiza el sistema con nuevas funcionalidades o requisitos (esto es, cada versión o refinamiento parte de una versión previa a la que le añade nuevas funciones.

Es un modelo que se ajusta a entornos de alta incertidumbre, por no poseer un conjunto exhaustivo de requisitos, especificaciones, diseños, etc., al comenzar el sistema, ampliándose éstos en cada refinamiento.

Representa una mejora sobre el modelo en cascada y, aunque permite el cambio continuo de requisitos, no se puede determinar si los requisitos propuestos son válidos, por lo que los errores de detectan tarde y su corrección resulta muy costosa. Un ejemplo de modelo en cascada utilizando el desarrollo incremental es el de la figura:



2.3-3.- Modelo espiral

El modelo en espiral consta de una serie de ciclos. Cada uno empieza identificando los objetivos, las alternativas y las restricciones del ciclo. Una vez evaluadas las alternativas respecto de los objetivos del ciclo y considerando las restricciones, se realiza el ciclo correspondiente para, una vez finalizado, iniciar el siguiente ciclo.

Cada ciclo en espiral realiza los siguientes pasos:

- ★ *Identificación* de:
  - *Los Objetivos* de la parte del producto que está siendo elaborada.
  - *Las Alternativas* principales de la implementación de esa porción del producto.
  - *Las Restricciones* para cada alternativa.
- ★ *Evaluación* de las diferentes *Alternativas* teniendo en cuenta los objetivos a conseguir y las restricciones impuestas.
- ★ *Formulación,* si existen riesgos, *de una estrategia efectiva* para resolver o minimizar dichos riesgos
- ★ *Revisión de los resultados del análisis de riesgos*
- ★ *Planificación de la fase posterior*

Realizado el primer ciclo se volvería a empezar.

En el modelo en espiral, cada ciclo se completa con una revisión en la que participan las principales personas u organizaciones que tienen relación con el ciclo, que cubre todos los productos desarrollados en el ciclo anterior e incluye los planes para el siguiente ciclo y los recursos necesarios para llevarlo a cabo.

Estos planes para las sucesivas fases pueden incluir una partición del producto en incrementos (para desarrollos sucesivos) o en componentes (para ser desarrollados por organizaciones individuales o por personas). En este último caso pueden preverse una serie de ciclos en paralelo, uno por cada componente, añadiendo una tercera dimensión al modelo en espiral

Las principales diferencias entre el modelo en espiral y los modelos anteriores son:

- ★ Existencia reconocida de diferentes alternativas
- ★ Identificación de riesgos para cada alternativa. La resolución de los riesgos es realmente el centro del modelo.
- ★ División del proyecto en ciclos, cada uno con un acuerdo al final de cada ciclo.
- ★ El modelo se adapta a cualquier tipo de actividad.

MODELOS PARA DESARROLLO DE SISTEMAS ORIENTADOS AL OBJETO

La tecnología de objetos pretende acelerar el desarrollo de sistemas de una manera iterativa (porque las tareas de cada fase se llevan a cabo de forma iterativa, a la vez que existe un ciclo de desarrollo análisis – diseño – instrumentación – análisis que permite evolucionar el sistema) e incremental (porque divide el sistema en un conjunto de particiones cada una de las cuales se desarrolla de manera completa hasta que finaliza el sistema), aprovechando aspectos importantes de la tecnología de objetos como el de "generalizar" los componentes para que sean reutilizables.

Con ello, las tareas de validación, verificación y aseguramiento de la calidad pueden realizarse para cada iteración de cada fase de cada incremento en el desarrollo del sistema, es decir, de forma continuada.

Dado que esta generalización incrementa los costes de desarrollo apreciablemente, es imprescindible un desarrollo que optimice la inversión efectuada.

	PROYECTO	PRODUCTO
Producto Final	Resultados	Componentes / Bibliotecas
Resultados Económicos	Beneficios	Inversión
Unidad	Departamento	Empresa / Industria
Tiempo	Corto Plazo	Largo Plazo
Objeto	Programas	Sistemas
Piezas	Elementos de Programa	Componentes de Software
Estrategia	Descendente	Ascendente
Metodología	Funcional	Orientada al Objeto
Lenguaje	C, Basic, Pascal, etc.	C++, Smalltalk, etc

Los modelos convencionales de ciclo de vida basan su filosofía en el "*proyecto*", (ver la tabla anterior) mientras que el desarrollo orientado al objeto se fundamenta en el "*producto*" entendido como elementos software reutilizables y cuyo beneficio económico aparece a largo plazo.

En general, los modelos de desarrollo orientado al objeto se caracterizan por:

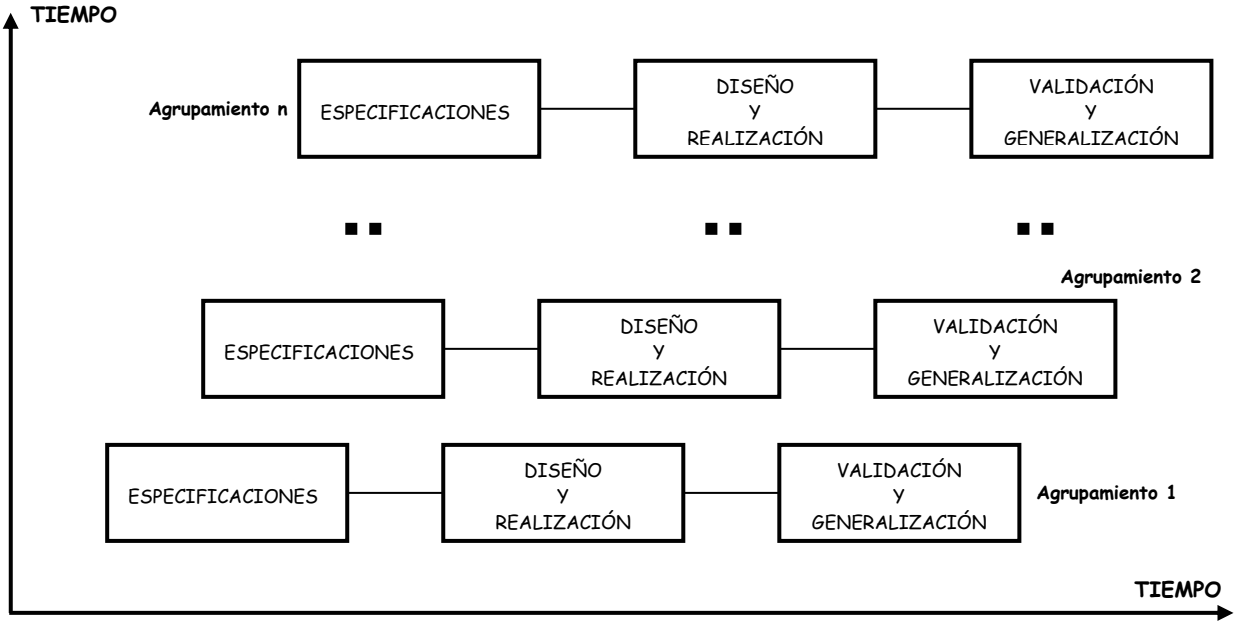
- ★ La eliminación de fronteras entre fases, debido a la naturaleza iterativa del desarrollo.
- ★ Una nueva forma de concebir los lenguajes de programación y su uso, ya que incorporan bibliotecas de clases y otros componentes reutilizables.
- ★ Un alto grado de iteración y solapamiento, dinamizando la forma de trabajo.

Para abordar esta problemática se han propuesto diferentes modelos:

2.3-4.- Modelo de agrupamiento

Dentro de esta filosofía del producto, el modelo de agrupamiento tiene en cuenta esta nueva fase de *generalización* que aparece combinada con la fase de *validación*.

El concepto fundamental del modelo es el de *agrupamiento (cluster)* definido como un conjunto de clases relacionadas con un objetivo común.



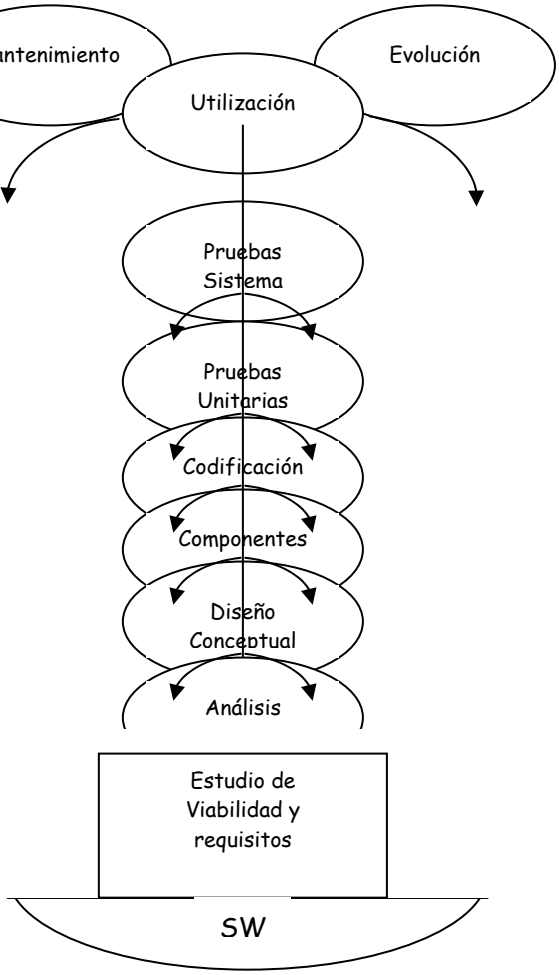
Los distintos agrupamientos representan así diferentes "subciclos" de vida que pueden solaparse en el tiempo. Cada agrupamiento incluye una fase de Especificación, otra de Diseño y Realización y una de Validación y Generalización.

Cada agrupamiento depende de los desarrollados con anterioridad, siguiendo un enfoque ascendente: se empieza desarrollando las clases básicas que pueden ser incluidas en bibliotecas.

2.3-5.- Modelo Fuente

Es el mas utilizado en el desarrollo orientado al objeto ya que representa gráficamente un alto grado de iteración y de solapamiento que hace posible la tecnología de objetos.

En la base del modelo se encuentra el *análisis de requisitos*, a partir de la cual va creciendo el ciclo de vida donde cada "burbuja" representa una fase, cayendo sólo para el mantenimiento necesario. La "piscina" sería el repositorio de clases. Su representación gráfica se indica en la figura:



2.3-6.- Modelo remolino

En la realidad, las metodologías anteriores en el desarrollo orientado al objeto no ofrecen una visión real del mismo ya que éste es mucho mas "desordenado" y ofrece múltiples "iteraciones interrelacionadas".

Los modelos anteriores ofrecen sólo una dimensión de iteración consistente en la fase del proceso, sin embargo,` pueden identificarse otras dimensiones:

- ★ *Amplitud* o tamaño del desarrollo
- ★ *Profundidad* referida al nivel de abstracción o detalle
- ★ *Madurez* entendida como grado de compleción, corrección y "elegancia"
- ★ *Alternativas* diferentes a las soluciones a un problema
- ★ *Alcance* en cuanto a objetivos del sistema, ya que los requisitos han cambiado a lo largo del proceso.

Las distintas dimensiones se pueden anidar de muchas maneras, estableciendo un desarrollo multicíclico que adopta la forma de un remolino.

2.3-7.- Modelo pinball

Este modelo establece una analogía con el juego de la máquina de "petacos" o "flipper", en el que la "bola" representa un proyecto completo o un subproyecto y el jugador al equipo de desarrollo.

Se procede de forma iterativa a encontrar clases, atributos, métodos e interrelaciones (fase de análisis) y definir colaboraciones, herencia, agregación y subsistemas (fase de diseño), por último se pasa a la programación, prueba e implementación. Todo ello, al igual que en el juego, puede darse en cualquier orden y de forma simultánea.

Pueden destacarse dos formas de jugar:

- ★ *A lo seguro.-* Con tecnologías y métodos probados.
- ★ *Al limite.-* Con mayor riesgo y también con mas ventajas.

En este modelo la habilidad y la experiencia representan, junto con la suerte el factor mas importante.

2.4.- METODOLOGÍAS DE DESARROLLO DEL SOFTWARE

Para desarrollar un proyecto de software es preciso establecer un enfoque disciplinado y sistemático del trabajo a realizar; la metodología de desarrollo a seguir se elaboran a partir del marco definido por uno o varios ciclos de vida, y deben tener determinadas características que dependerán del enfoque de desarrollo.

No existe una definición universalmente aceptada sobre el concepto de metodología, aunque si existe un acuerdo en considerarla como "un conjunto de pasos y procedimientos que deben seguirse para el desarrollo del software".

Una definición de metodología podría ser "*el conjunto de filosofías, fases, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de sistemas de información*" (Maddison 1983). La metodología es pues el conjunto de componentes que especifican:

- ★ Cómo se debe dividir un proyecto en etapas.
- ★ Qué tareas se realizan en cada etapa.
- ★ Qué salidas se producen y cuándo se deben producir.

- ★ Qué restricciones se aplican.
- ★ Qué herramientas se van a utilizar.
- ★ Cómo se gestiona y controla un proyecto.

La metodología normalmente consistirá en un conjunto de fases descompuestas en subfases (módulos, etapas, pasos, etc.) de forma que esta descomposición guíe a los desarrolladores en la elección de las técnicas que se debe elegir para cada estado del proyecto, facilitando la planificación, gestión, control y evaluación de los proyectos.

Una metodología representa el camino para desarrollar software de una manera sistemática.

Pueden identificarse como Necesidades principales que se intentan cubrir con una metodología las siguientes:

- *Mejores aplicaciones*: el seguimiento de una metodología no basta para asegurar la calidad del producto final.
- *Un mejor proceso de desarrollo*: que identifica las salidas de cada fase de forma que se pueda planificar y controlar el proyecto. Los sistemas se desarrollan más rápidamente y con los recursos apropiados.
- *Un proceso estándar en la organización*: lo que aporta claros beneficios (mayor integración de los sistemas, mas facilidad en el cambio de una persona de un proyecto a otro, etc.).

Por otra parte, las metodologías pueden tener distintos **objetivos** que hacen que difieran unas de otras; estos objetivos pueden ser:

- ★ Registrar los requisitos de un sistema de información de forma acertada.
- ★ Proporcionar un método sistemático de desarrollo de forma que se pueda controlar su progreso.
- ★ Construir un sistema de información dentro de un tiempo apropiado y unos costes aceptables.
- ★ Construir un sistema que esté bien documentado y que sea fácil de mantener.
- ★ Ayudar a identificar lo más pronto posible cualquier cambio que sea necesario realizar dentro del proceso de desarrollo.
- ★ Proporcionar un sistema que satisfaga a todas las personas afectadas por el mismo, ya sean clientes, directivos, auditores o usuarios.

La descomposición del proyecto llega hasta las *tareas o actividades elementales*.

Para cada tarea se identifica un procedimiento que define la forma de ejecutarla y representa el marco de comunicación entre usuarios y desarrolladores. Como resultado de seguir un procedimiento, se obtienen uno o más productos (que pueden ser *productos intermedios* como base para realizar nuevos productos durante el desarrollo o *productos finales*). El sistema deseado está constituido por un conjunto de productos finales.

Para aplicar un procedimiento se pueden utilizar una o más **técnicas**. Suelen ser con mucha frecuencia gráficas con apoyos textuales formales y determinan el formato de los productos resultantes de cada tarea.

Para la realización de una técnica pueden utilizarse **herramientas software** que automatizan en mayor o menor grado su aplicación. Algunas herramientas dan soporte específico a una metodología, otras son de propósito más general.

Una metodología puede seguir uno o varios modelos de ciclo de vida. El ciclo de vida indica qué es lo que hay que obtener a lo largo del desarrollo del proyecto pero no la forma de obtenerlo. Esta es precisamente la labor de la metodología.

2.4-1.- Visión histórica de las metodologías de desarrollo de sistemas de información

Los enfoques metodológicos han ido evolucionando a lo largo del tiempo. Esta evolución histórica ha sido como sigue:

DESARROLLO CONVENCIONAL: se basaban en funciones básicas de procesos de datos. Los programadores estaban más enfocados en las tareas de codificar que en la de recoger y comprender las necesidades de los usuarios. Se observó que había más de un papel en el proceso de desarrollo de sistemas: operadores, programadores y analistas de sistemas (funcionales y técnicos u orgánicos).

Este enfoque presenta los siguientes **problemas**:

- Los resultados finales son impredecibles.
- No hay forma de controlar lo que está sucediendo en el proyecto, dado que no hay fases establecidas ni productos intermedios sobre los que hacer verificaciones.
- Los cambios organizativos afectan negativamente al proceso de desarrollo.

DESARROLLO ESTRUCTURADO: sigue unos métodos de ingeniería sentando las bases para el desarrollo automatizado, Estas técnicas están dirigidas a aspectos tanto técnicos como de gestión en la construcción de software.

Su desarrollo histórico se ha fundamentado en lo siguiente:

- *Programación estructurada*. Para determinar cómo se debería ver un programa, tanto de forma estática como dinámica de forma que fuera lo mas comprensible posible se establecieron normas para la aplicación de estructuras de datos y de control.
- *Diseño estructurado*. Define un nivel de abstracción utilizando el módulo de programa como componente básico de construcción. Se refina el concepto de modularidad normalizando la estructura de un módulo del programa, restringiendo las relaciones entre módulos y estableciendo medidas de calidad de los programas.
- *Análisis estructurado*. Hasta la aparición de los primeros conceptos sobre el análisis estructurado, en la gran mayoría de proyectos de desarrollo se hacía una especificación narrativa de los requisitos tal y como los percibía el analista. Estas especificaciones adolecían de los siguientes problemas:
  - ✓ Eran **monolíticas**. Es preciso leer la especificación de los requisitos completamente parea poder entenderla
  - ✓ Eran **redundantes**. Se repite frecuentemente la misma información en distintas partes del documento. Por lo que, si cambian algún requisito, el documento debe modificarse en diferentes lugares.
  - ✓ Eran **ambiguas**. El informe detallado de requisitos se interpreta de forma distinta por usuarios, analistas y diseñadores
  - ✓ Eran **imposibles de mantener**. Cuando se llega al final del proceso de desarrollo. La especificación funcional era casi obsoleta.

  Debido a estos problemas, ha habido un movimiento gradual hacia las especificaciones funcionales:
  - ✓ **Particionadas.** De forma que se puedan leer porciones independientes de la especificación
  - ✓ **Gráficas.** Compuestas por diagramas y técnicas textuales que sirven como material de referencia a la especificación

- ✓ **Mínimamente redundantes**. Afectando los cambios a una parte de la especificación(a requisitos específicos, sean funcionales o no.)

Este enfoque, conocido como *análisis estructurado o análisis descendente o análisis top-down* se utilizaba principalmente en organizaciones de desarrollo de sistemas de gestión. Posteriormente ha sufrido los siguientes cambios:

- ★ Dar menor importancia a la construcción de modelos físicos actuales y modelos lógicos actuales
- ★ Diferenciar mas los modelos físicos y los modelos lógicos.
- ★ Ampliar las técnicas de análisis estructurado para poder modelar sistemas a tiempo real.
- ★ Enfocarse en el modelo de datos.
- ★ Estudiar los eventos.

La evolución histórica de las metodologías mas representativas en la Ingeniería del software se representa en la tabla adjunta:

AÑO	METODOLOGÍA
1968	*Conceptos sobre programación estructurada (DIJKSTRA)*
1974	*Técnicas de programación estructurada (WARNIER, JACKSON)*
1975	*Primeros conceptos sobre diseño estructurado (MYERS, YOURDON)*
1977	*Primeros conceptos sobre análisis estructurado (GANE , SARSON)*
1978	*Análisis estructurado (DEMARCO, WEINBERG)*
1978	*Modelo MERISE*
1981	*Versión inicial de SSADM*
1981	*Versión inicial de NFORMATION ENGINEERING*
1985	*Análisis y diseño estructurado para sistemas en tiempo real (WARD, MELLOR)*
1986	*SSADM (Versión 3)*
1987	*Análisis y diseño estructurado para sistemas en tiempo real (HATLEY, PIRHBAY)*
1989	*MÉTRICA (Versión inicial)*
1990	*SSADM (Versión 4)*
1993	*MÉTRICA (Versión 2)*
1995	*MÉTRICA (Versión 2.1)*

DESARROLLO ORIENTADO AL OBJETO: el paradigma orientado a objetos, a diferencia del enfoque estructurado, trata los procesos y los datos de forma conjunta, esto es, modula tanto la información como el procesamiento. La orientación a objetos empieza con los lenguajes de programación orientados a objetos (LOO) . En estos lenguajes se daba énfasis a la abstracción de datos y los problemas del mundo real se representaban como un conjunto de objetos de datos para los que se adjuntaban un conjunto de operaciones (SIMULA, Smaltalk como prototipos y ADA, C++, Smaltalk y Objetive – C como lenguajes de desarrollo).

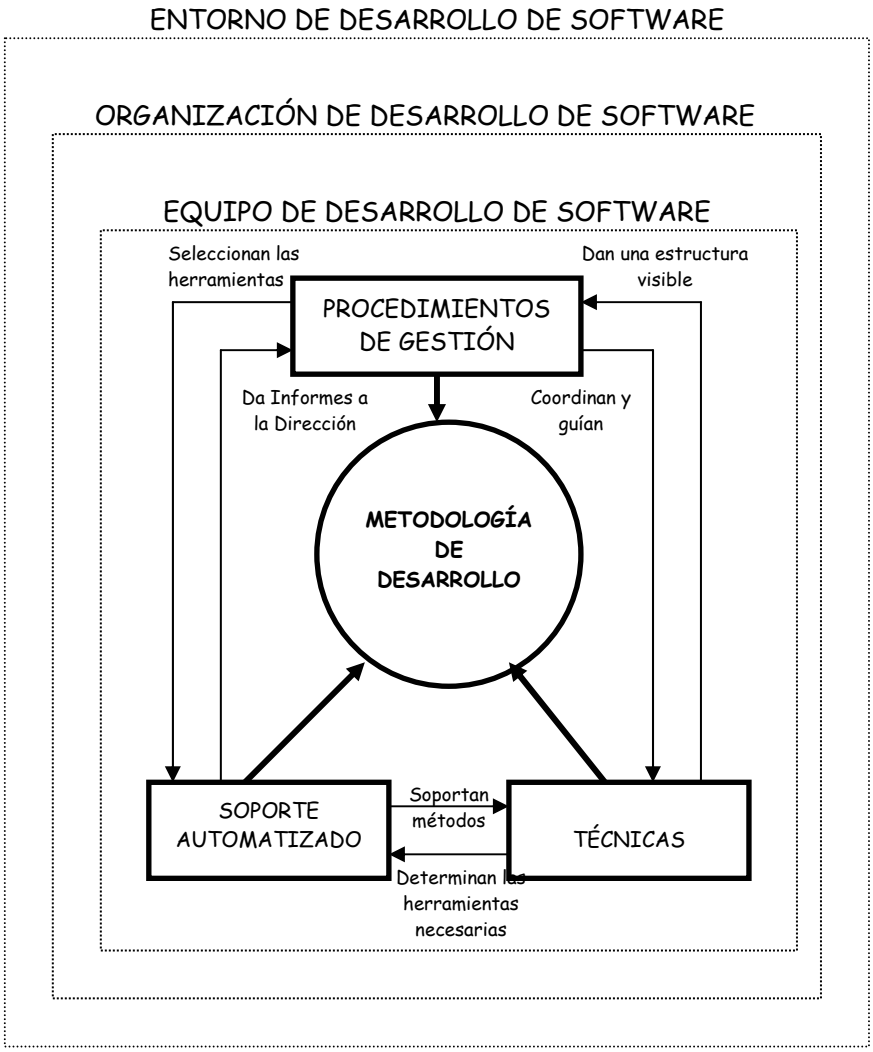
2.5.- CARACTERÍSTICAS PRINCIPALES DE LAS METODOLOGÍAS.

2.5-1.- Impacto de la metodología en el entorno de desarrollo de software

Todo entorno de desarrollo incluye un conjunto de componentes que condicionan la construcción del software. Cualquier cambio que se realice dentro del entorno puede tener un efecto inmediato sobre la productividad del personal de desarrollo. La productividad en

sí no basta y debe estar asociada a la calidad de los productos finales. La metodología de desarrollo influye muy directamente en estos dos factores.

En la figura adjunta se muestra la relación entre la metodología y el entorno de desarrollo.



Dentro de este entorno, la organización mantiene a un equipo de desarrollo de software. Los procedimientos de gestión determinan el tipo de soporte automatizado, tanto hardware como software, que se debe proporcionar especificando las herramientas necesarias para el desarrollo. Los procedimientos de gestión coordinan y guían a los desarrolladores en el empleo de las técnicas. A su vez, el soporte automatizado mejora la productividad automatizando diversas tareas y verificando su correcta realización.

Debe tenerse en cuenta que no todos los resultados de las tareas pueden ser verificados de forma automatizada y siempre es necesaria la realización de revisiones manuales incluidas en los procedimientos de gestión.

Todos los entornos de desarrollo de software tienen formas de trabajo muy diferentes. La organización de desarrollo tiene dos opciones:

- ⋆ **Seleccionar entre un gran número de posibilidades y combinaciones** de métodos de gestión, técnicas de desarrollo y soporte automatizado, para crear y desarrollar la metodología de desarrollo software más apropiada.

- ⋆ **Analizar y evaluar metodologías existentes y adoptar en la organización la que más se ajuste a sus necesidades.** En general, esta es la opción más común.

La metodología está también influenciada por consideraciones como el tamaño y la estructura de la organización, así como por el tipo de las aplicaciones que va a desarrollar. No es razonable pensar, por tanto, que dos organizaciones utilicen la misma metodología sin realizar cambios sobre ella, ajustándola a su organización y a sus proyectos.

2.5-2.- Características deseables en una metodología

Una metodología debería incluir una serie de características deseables, entre las que se destacan:

1) ***Existencia de reglas predefinidas:*** que definan sus fases, tareas, productos intermedios, técnicas, herramientas, ayudas al desarrollo y formatos de documentación estándar.
2) ***Cobertura total del ciclo de desarrollo:*** pasos que hay que realizar desde el planteamiento de un sistema hasta su mantenimiento, proporcionando mecanismos para integrar los resultados de una fase a la siguiente, de forma que se pueda referenciar a fases previas y comprobar el trabajo realizado.
3) ***Verificaciones intermedias:*** sobre productos generados en cada fase para comprobar su corrección. Por medio de revisiones software que detectan inconsistencias, inexactitudes o cualquier otro tipo de defecto que se genera durante el proceso de desarrollo evitando que salgan a relucir en la fase de pruebas o en las pruebas de aceptación o durante la fase de mantenimiento.
4) ***Planificación y control:*** una forma de desarrollar software de manera planificada y controlada para que no se disparen los costes ni se amplíen los tiempos de entrega. También debería incorporar alguna técnica de control de proyectos.
5) ***Comunicación efectiva:*** entre los desarrolladores para facilitar el trabajo en grupo y con los usuarios.
6) ***Utilización sobre un abanico amplio de proyectos:*** debe ser flexible. No se deberían utilizar metodologías diferentes para cada proyecto.
7) ***Fácil formación:*** los desarrolladores deben comprender las técnicas y los procedimientos de gestión. La organización debe formar a su personal en todos los procedimientos definidos por la metodología
8) ***Herramientas CASE:*** debe estar soportada por herramientas automatizadas que mejoren la productividad del equipo de desarrollo y la calidad de los productos resultantes. Como una metodología define las técnicas que hay que seguir en cada tarea, es preciso disponer de una herramienta que soporte la automatización de dichas tareas.
9) ***La metodología debe contener actividades que mejoren el proceso de desarrollo:*** es necesario disponer de datos que muestren la efectividad de la aplicación del proceso sobre un determinado producto. Definir mediciones que indiquen la calidad y el coste asociado a cada etapa del proceso.
10) ***Soporte al mantenimiento:*** el campo de reingeniería del software debería ser tomado en cuenta por las metodologías para facilitar las modificaciones sobre los sistemas existentes.
11) ***Soporte de la reutilización del software:*** las metodologías estructuradas existentes no proporcionan mecanismos para la reutilización de componentes software. Se deberían incluir procedimientos para la creación, mantenimiento y recuperación de componentes reutilizables que no se limiten sólo al código.

2.6.- CLASIFICACIÓN DE LAS METODOLOGÍAS.

En función de tres dimensiones o puntos de vista (enfoque, tipo de sistema y formalidad) las metodologías pueden clasificarse en:

ENFOQUE	TIPO DE SISTEMA	FORMALIDAD
ESTRUCTURADAS ★ Orientadas a procesos ★ Orientadas a datos • Jerárquicos • No jerárquicos ★ Mixtas	GESTIÓN	NO FORMAL
ORIENTADAS A OBJETOS	TIEMPO REAL	FORMAL

2.6-1.- Metodologías estructuradas

Proponen la creación de modelos del sistema que representan los procesos, los flujos y la estructura de los datos de una manera descendente (top down), pasando de una visión mas general del problema (un nivel de abstracción mas cercano a las personas) hasta llegar a un nivel de abstracción mas sencillo (mas cercano al hardware). Esta visión se puede enfocar en las funciones (o procesos) del sistema, en la estructura de los datos o a ambos aspectos dando lugar a los siguientes tipos de metodologías:

2.6.1.1.- Metodologías orientadas a procesos

La ingeniería del software está fundamentada en el modelo básico de **entrada ➔ proceso ➔ salida** de un sistema. Los datos se introducen en el sistema y el sistema responde ante ellos transformándolos para obtener las salidas.

Las metodologías estructuradas se basan en la utilización de un método descendente de descomposición funcional para definir los requisitos del sistema. Se apoyan en técnicas gráficas dando lugar a un nuevo concepto: especificación estructurada

Una **especificación estructurada** es un modelo gráfico, particionado, descendente y jerárquico de los procesos del sistema y de los datos utilizados por los procesos.

Se compone de:

- **Diagramas de flujos de datos (DFD):** representan los procesos (funciones) que debe llevar a cabo un sistema a distintos niveles de abstracción y los datos que fluyen entre las funciones. Los procesos más complejos se descomponen en nuevos diagramas hasta llegar a procesos sencillos.

- **Diccionario de datos:** es el conjunto de definiciones de todos los datos que aparecen en el DFD, tanto almacenados como en los flujos de datos.

- **Especificaciones de proceso:** describen con más detalle lo que ocurre dentro de un proceso.

Como **Ejemplos** de metodologías estructuradas pueden considerarse:

Metodología de DeMarco

1) *Estudio del entorno físico actual*: Antes de estudiar las necesidades del usuario, se comienza haciendo un modelo del sistema actual que muestra los procedimientos actuales, estén o no automatizados, realizando un conjunto de DFD's físicos actuales.
2) *Derivación del correspondiente modelo lógico actual*.: Se obtiene en esta etapa el modelo derivado del anterior, pero sin connotaciones físicas.
3) *Derivación del nuevo modelo lógico:* se toman en consideración las nuevas necesidades de los usuarios, estableciendo un modelo que describe lo que hay que hacer pero no cómo, obteniendo una especificación estructurada formada por los DFD, el diccionario de datos y las especificaciones de procesos del sistema.
4) *Crear un conjunto de modelos físicos alternativos*.
5) *Valorar cada opción.*
6) *Seleccionar una opción.*
7) *Empaquetar la especificación*.

Metodología de Gane y Sarson:

La metodología en sí es parecida a la de Demarco. La diferencia principal es que hay una etapa en la que se definen los contenidos de los almacenes que aparecen en el DFD en tercera forma normal. Los pasos que sigue son los siguientes:

1) *Construir el modelo lógico actual* (DFD lógico actual).
2) *Construir el modelo lógico del nuevo sistema*. Elaborar una especificación estructurada y construir un modelo lógico de datos en tercera forma normal que exprese el contenido de los almacenes de datos.
3) *Seleccionar un modelo lógico*.
4) *Crear el nuevo modelo físico del sistema*.
5) *Empaquetar la especificación*.

Metodología de Yourdon/Constantine:

Consta de las siguientes fases:

1) Realizar los DFD del sistema.
2) Realizar el *diagrama de estructuras*, obteniéndolo a partir de los DFD's mediante dos técnicas, al *análisis de transformación* y el *análisis de transacción*.
3) Evaluación del diseño, midiendo la calidad de la estructura resultante mediante el *acoplamiento* y la *cohesión*.
4) Preparación del diseño para la implantación dividiéndolo en unidades físicas de implantación (*cuadernos de carga*).

2.6-1-2.-Metodologías orientadas a datos jerárquicos

Dentro del modelo básico de **entrada ➔ proceso ➔ salida** de un sistema, estas metodologías se orientan más a las entradas y salidas. Se definen las estructuras de datos y a partir de estas se derivan los componentes procedimentales. En este enfoque destaca:

- La estructura de control del programa debe ser jerárquica y se debe derivar de la estructura de datos del programa.
- El proceso de diseño consiste en definir primero la estructura de los datos de entrada y salida, mezclarlas todas en una estructura jerárquica de programa y después ordenar detalladamente la lógica procedimental par que se ajuste a esta estructura.
- El diseño lógico debe preceder y estar separado del diseño físico.

Como ejemplos de metodologías pueden considerarse, los métodos de JSP y JSD de Jackson, la construcción lógica de programas (LCP)y el desarrollo de sistemas

estructurados de datos (LCS) , basado en la identificación de datos primarios y secundarios (calculados). A partir de la construcción de un diccionario de datos, el método se centra en el conjunto de actualizaciones y modificaciones necesarias para obtener la salida.

Originalmente, el LCS trabaja con ficheros como estructuras de datos.

2.6-1-3.- Metodologías orientadas a datos no jerárquicos

Las metodologías basadas en la información se centran en la creencia de que los datos (tipos de datos) constituyen el corazón del sistema de información ya son más estables que los proceso que actúan sobre ellos.

La metodología que identifique con éxito la naturaleza de los datos de una organización partirá de una base estable para construir los sistemas de información. Los procesos también se estudian en este tipo de metodologías, pero vienen derivado de una definición inicial de los datos (modelo de datos) utilizados por la organización. Este modelo de datos está constituido por el conjunto de entidades de datos básicas y las interrelaciones entre ellas.

Con este enfoque, todos los sistemas construidos están integrados sobre el mismo modelo de datos.

Como ejemplo de esta metodología puede considerarse la **Ingeniería de la Información** dividida en cuatro etapas:

- ***Planificación:*** construir una arquitectura de la información y una estrategia que soporte los objetivos de la organización.
- ***Análisis***: comprender las áreas del negocio y determinar los requisitos del sistema.
- ***Diseño:*** establecer el comportamiento del sistema deseado por el usuario y que sea alcanzable por la tecnología.
- ***Construcción:*** construir sistemas que cumplan los tres niveles anteriores.

2.6-2.- Metodologías orientadas a objetos

Entre las metodologías clásicas de análisis y diseño estructurado y las de orientación al objeto existe un cambio de filosofía apreciable. Mientras en las primeras se examinan los sistemas desde el punto de vista de las funciones o tareas que deben realizar (tareas que se van descomponiendo sucesivamente en otras tareas mas pequeñas y que forman los bloques o módulos de las aplicaciones), en la orientación al objeto cobra mucha mas importancia el aspecto de "modelado" del sistema, examinando el dominio del problema como un conjunto de objetos que interactúan entre sí.

En las metodologías tradicionales se produce una dicotomía entre los dos elementos constituyentes de un sistema: las funciones que llevan a cabo los programas y los datos que se almacenan en ficheros o en bases de datos.

Sin embargo, la orientación al objeto intenta obtener un enfoque unificador de ambos aspectos, que se encapsulan en los objetos.

Pueden identificarse dos **enfoques** en las metodologías orientadas al objeto:

- ***Revolucionarios*** o ***puros***: que entienden la orientación al objeto como un cambio profundo que convierten a las metodologías estructuradas en obsoletas.
- ***Sintetistas*** o ***evolutivos***: que piensan que el análisis y diseño estructurado constituyen la base para el desarrollo orientado al objeto, pudiéndose combinar elementos del análisis y diseño estructurado con los de orientación a l objeto.

2.6-3.- Sistemas en tiempo real

Son sistemas muy dependientes del tiempo que procesan información mas orientada al control que orientada a los datos. Se controlan y son controlados por eventos externos.

Una definición de sistema en tiempo real podría ser: "aquel sistema que controla un ambiente recibiendo datos, procesándolos y devolviéndolos con la suficiente rapidez como para influir en dicho ambiente en ese momento".

Se caracterizan porque:

- Se lleva a cabo el proceso de muchas actividades de forma simultanea (**concurrencia**).
- Se asignan prioridades a determinados procesos (algunos requieren tratamiento inmediato y otros pueden aplazarse por periodos razonables de tiempo (**batch**).
- Se interrumpe una tarea antes de que concluya, para comenzar otra de mayor prioridad.
- Existe comunicación entre tareas.
- Existe acceso simultáneo a datos comunes, tanto en memoria como en almacenamiento secundario.

No deben confundirse los sistemas en tiempo real con los sistemas en línea (interactivos u on-line). Estos últimos interactúan con las personas, mientras que los primeros interactúan tanto con las personas como con los dispositivos físicos del mundo exterior. Si un sistema en tiempo real no responde con la suficiente rapidez, el ambiente puede quedar fuera de control, perdiendo datos de entrada. En un sistema en línea no se pierden datos, sólo "hay que esperar un poco más".

Para especificar los requisitos de estos sistemas hay que incluir nuevos conceptos para :

- El manejo de interrupciones.
- La comunicación y sincronización entre tareas.
- Gestionar procesos concurrentes.
- Dar respuesta oportuna y a tiempo ante eventos externos.
- Datos continuos o discretos.

2.7.- PRINCIPALES METODOLOGÍAS DE DESARROLLO.

Distintos países han promovido el desarrollo de metodologías para unificar la forma de desarrollar sus sistemas de información. Las mas conocidas son la francesa MERISE, la inglesa SSADM y la española MÉTRICA.

2.7–1 .- MERISE

Las mayores aportaciones de la metodología son:

- Un ciclo de vida más largo que se materializa en un conjunto definido por etapas, entre las se incluye una etapa de planificación previa al desarrollo (**esquema director**).
- Introducción de dos ciclos complementarios: ciclo de **abstracción** y ciclo de **decisión**. El primero se basa en la percepción de tres niveles de abstracción: conceptual, organizativo y físico u operativo. Se definen con dos modelos para cada nivel, un modelo de datos y un modelo de tratamientos:
  - ★ **Nivel conceptual**: corresponde a las finalidades de la empresa. Se ocupa de definir el ***Qué*** a través de un conjunto de reglas de gestión, objetivos y limitaciones que pesan sobre la empresa.
  - ★ **Nivel organizativo**: corresponde a la organización adecuada que hay que implantar para alcanzar los objetivos asignados al sistema. Requiere una especificación de los puestos

de trabajo, cronología de las operaciones y la elección de la automatización así como la distribución geográfica de datos y tratamientos.

- ★ **Nivel físico**: corresponde a la integración de los medios técnicos necesarios para el proyecto. Se expresan en términos de materiales o componentes software. Este nivel es el que está más sujeto a cambios como resultado de los progresos tecnológicos.

NIVELES	DATOS	TRATAMIENTOS
CONCEPTUAL	Modelo Conceptual de datos	Modelo Conceptual de Tratamientos
ORGANIZATIVO	Modelo Lógico de Datos	Modelo Organizativo de Tratamientos
FÍSICO	Modelo Físico de Datos	Modelo Operativo de Tratamientos

Las **Fases de la Metodología** son las siguientes:

- ★ **Estudio preliminar**: analiza la situación existente para proponer una solución global atendiendo a los criterios de gestión, de la organización y decisiones adoptadas por quien tiene capacidad para ello.
- ★ **Estudio detallado**: define, en el ámbito funcional, la solución a implementar.
- ★ **Implementación**: realiza los programas que corresponden a las especificaciones detalladas. Se divide en dos partes:
  - ***Estudio técnico***. Distribuye los datos en ficheros físicos y los tratamientos en módulos de programas
  - ***Producción de programas***. Codifica y verifica los programas mediante juegos de pruebas
- ★ **Realización y puesta en marcha**: implantación de los medio técnicos (instalación de los materiales, iniciación, captura de datos, etc.) y organizativos (formación del personal, lanzamiento de la aplicación, etc.) y recepción definitiva por parte del usuario.

2.7-2.- SSADM

Los aspectos claves de esta metodología son:

- Énfasis en los usuarios: sus requisitos y participación.
- Definición del proceso de producción.
- Tres puntos de vista: datos, eventos y procesos.
- Máxima flexibilidad en herramientas y técnicas de implementación.

SSADM proporciona un conjunto de procedimientos para llevar a cabo el análisis y diseño, pero no cubre aspectos como la planificación estratégica ni entra en la construcción del código.

2.7-3.- MÉTRICA

La metodología MÉTRICA versión 2 está estructurada mediante una sucesión de fases, módulos, actividades y tareas que hay que seguir para el desarrollo de sistemas, e indica los productos que se obtienen en cada una de las tareas. Algunos son productos finales, y otros, productos intermedios que servirán de base para la realización de tareas posteriores.

La metodología está dividida en las siguientes fases:

- **Fase 0:** Plan de Sistemas de Información.
- **Fase 1:** Análisis de Sistemas.
- **Fase 2:** Diseño de Sistemas.

- **Fase 3:** Construcción de Sistemas.
- **Fase 4:** Implantación de Sistemas.

Hace uso de una serie de técnicas que dan soporte a la realización de las tareas que tiene definidas. Se enfoca directamente en el desarrollo y no pretende soportar todas las actividades relacionadas con aspectos como la Gestión de Proyectos, Gestión de la Calidad y la Gestión de la Configuración. Si proporciona mecanismos para unir dichos conceptos identificando el lugar donde se realizan dentro de la metodología.

2.8.- ANÁLISIS DE NECESIDADES Y ESTUDIO DE VIABILIDAD

Conocido el concepto de ciclo de vida del software, los modelos para el ciclo de vida y las metodologías utilizadas para desarrollar dicho ciclo de vida, el objeto de esta unidad se centra en establecer las primeras actividades del desarrollo del software, indicando cuales pueden ser los puntos de partida de un proyecto para, a partir de ellos, abordar el análisis de las necesidades del cliente / usuario del sistema que se va a construir.

Una vez obtenida una idea clara de los objetivos que se desean conseguir, se debe evaluar la viabilidad del proyecto, desde un punto de vista técnico, económico, etc. Aplicando distintas técnicas de recogida de información.

2.8-1.- Como comienza un proyecto

Para cada proyecto de desarrollo de software existen dos puntos de partida: uno desde la empresa y otro considerando el propio proyecto. El primero es anterior al segundo, al considerar que el segundo sólo tiene lugar cuando la empresa ha tomado la decisión de emprender el proyecto.

2.8-1-1.- Inicio en el ámbito de la empresa

Los elementos que marcan el inicio de un proyecto en el ámbito de la empresa son:

- Decisión de emprender el proyecto
- La selección del director o jefe del proyecto al que se le asignará la responsabilidad de establecer el entorno de inicio del proyecto.

Decisión de emprender el proyecto

Las situaciones que determinan el inicio de un proyecto, desde el punto de vista de la organización que va a desarrollar el software, pueden ser:

★ En el caso de un departamento de desarrollo de una empresa, deben satisfacerse las necesidades de soporte informático de la compañía. Otros departamentos pueden solicitar mejoras de las aplicaciones ya existentes o soluciones para realizar su trabajo de forma mas eficaz. En cualquier caso, la ***idea o necesidad inicial*** puede tener diferentes orígenes:

1) La constatación de que los requisitos software existentes están cambiando.
2) Una petición específica de un cliente o de un usuario.
3) Una propuesta generada dentro de la organización de desarrollo.
4) Una necesidad detectada por el departamento de marketing.
5) El personal de mantenimiento de las aplicaciones existentes realiza una recomendación específica.
6) Se detecta a partir de la información obtenida de los usuarios.
7) Como respuesta a la demanda de un cliente.

Debe pues obtenerse una idea clara de lo que se pretende y, posteriormente, evaluar la viabilidad del proyecto poniendo un especial énfasis en los beneficios y gastos que representa.

★ Si se trata de una empresa de servicios informáticos, el proyecto suele surgir como respuesta a la demanda de un cliente.

Además de obtenerse una idea clara de lo que se pretende y evaluar la viabilidad del proyecto poniendo un especial énfasis en los beneficios (normalmente establecidos en las condiciones contractuales que se establezcan) y gastos (que contendrá, los necesarios para la preparación de propuestas para luchar por la concesión del contrato) que representa, será preciso convencer al cliente y estudiar las ventajas que proporcionará la aplicación a su negocio.

La mayoría de las organizaciones tienen establecidos procedimientos de asignación de recursos para investigar las posibilidades de un proyecto, hasta que se toma la decisión de realizarlo y se comprometen los fondos necesarios para su desarrollo. Se trata de comprender el alcance del proyecto y la complejidad de su desarrollo para poder determinar el esfuerzo que requiere.

Será necesario definir y analizar los requisitos del sistema que se debe construir.

Los resultados de los estudios previos se suelen recoger en un documento llamado ***informe de necesidades.*** Con esta información, la decisión de emprender un proyecto implica estudiar su viabilidad. Se deberá analizar entonces:

- Las diferentes alternativas que se pueden concebir.
  - Comprar un producto software comercial ya construido
  - Desarrollar el producto internamente
  - Desarrollarlo de forma externa mediante un subcontrato
  - Automatizar sólo parcialmente el sistema
  - Etc.
- La evaluación de cada una de las alternativas, incluyendo su viabilidad económica, técnica, legal, operativa, etc.
- La especificación detallada de la alternativa seleccionada.
- El establecimiento de fechas y de compromisos de trabajo por parte de las personas y departamentos implicados (definición de un plan para el proyecto).

2.8-1-2.-Selección del director del proyecto

La selección del director suele ser posterior a la decisión de emprender el proyecto, aunque sería deseable que el director interviniera desde el inicio del proceso.

El director es posiblemente el elemento mas crítico para el éxito del proyecto, ya que debe disponer de la cualificación necesaria para el puesto que puede variar mucho de unos proyectos a otros. No obstante, las características mas deseables para dirigir un proyecto son:

- ✓ **Liderazgo**: habilidad para motivar a los componentes del equipo del proyecto.
- ✓ **Comprensión técnica:** conocimientos para tomar decisiones técnicas correctas.
- ✓ **Competencia en la gestión:** capacidad para controlar las actividades, los costes y los presupuestos del proyecto.
- ✓ **Presteza y decisión:** para observar, evaluar y decidir.
- ✓ **Versatilidad y flexibilidad**: para encauzar acontecimientos imprevistos.
- ✓ **Integridad:** para reclutar al personal más capacitado y ganarse la confianza del cliente.

- ✓ **Previsión:** para anticiparse a los problemas y aportar soluciones.

2.8-2.- Inicio en el ámbito del proyecto

El director del proyecto seleccionado debería establecer el entorno inicial de trabajo del proyecto, incluso antes de verificarse la viabilidad del mismo.

Esta tarea de preparación incluye la identificación de las áreas de gran riesgo del proyecto y el establecimiento de presupuestos, calendarios, planes de trabajo y asignaciones de tareas. También debe definir el soporte necesario para el equipo del proyecto, las técnicas de comunicación entre sus miembros, los requisitos que deberán cumplir los subcontratistas y la manera de abordar la interacción con el cliente. Los resultados serán la primera versión del **Plan del Proyecto,** que debe definir qué hay que hacer, cómo se hará, quién lo hará, cuándo, y qué herramientas y técnicas de gestión y de desarrollo se utilizarán

2.9.- ESTUDIOS DE VIABILIDAD

Los proyectos de desarrollo software se caracterizan por la escasez de recursos y la dificultad para cumplir los plazos establecidos. Antes de proceder con un desarrollo debe evaluarse su viabilidad y, preferentemente, analizar los riesgos que comporta.

Estos estudios de viabilidad pueden considerarse en dos vertientes: ***desde el punto de vista del cliente*** y ***desde el punto de vista del suministrador del software***.

- **Económico:** determinar si el beneficio obtenido compensa los costes.
- **Técnico**: estudiar si la funcionalidad, el rendimiento o las restricciones marcadas son realizables.
- **Legal**: dilucidar si los requisitos atentan contra alguna ley o reglamento o a disposiciones legales de contratos, responsabilidad civil, etc.
- **Operativa:** determinar si se puede implantar de manera efectiva en la empresa.

Las conclusiones obtenidas en el primer apartado son las mas determinantes.. Las conclusiones del análisis económico del coste / beneficio harán que la organización tome la decisión de continuar o de cancelar el proyecto.

2.9-1.- Análisis coste / beneficio desde la empresa

El análisis coste / beneficio permite seleccionar la alternativa más beneficiosa y prever necesidades financieras, teniendo en cuenta no sólo los elementos **tangibles** que se pueden evaluar directamente, sino también los elementos **intangibles** que no pueden evaluarse con precisión.

Este análisis se representa en forma de tabla, en las columnas aparecen los **años de vida** del proyecto y en las filas los distintos **conceptos de gasto y beneficio** del proyecto.

Entre los **costes** destacan:

- **Coste del personal** informático implicado en el proyecto desde el análisis hasta la instalación del sistema.
- **Coste de consultoría**.
- **Coste de software adicional**.
- **Coste del hardware**.
- **Coste de la infraestructura**.
- **Costes debidos al usuario**.

Por su parte, los **beneficios** pueden aparecer de muy diferentes maneras:

- **Nuevas funcionalidades**.

- **Eliminación de errores.**
- **Reducción de errores.**
- **Aumento de la velocidad.**
- **Aumento de la fiabilidad.**

Uno de los mayores problemas para que el análisis económico sea realista es que la valoración de los beneficios es, en muchos casos, necesariamente subjetiva. Esto constituye una gran desventaja para convencer a la dirección de las ventajas del proyecto.

Uno de los parámetros mas utilizados en la valoración económica de los proyectos es el *plazo de amortización necesario para recuperar el dinero invertido* denominado **ROI** (Return of Investment).

Aspectos importantes para un análisis de coste / beneficio eficaz son:

- La mayoría de las estimaciones de costes y beneficios suelen consistir en rangos de valores probables. A medida que el proyecto avance se puede refinar el análisis económico.
- Es recomendable hacer estimaciones más bien pesimistas o conservadoras.
- Debe contarse con los diversos factores que influyan en toda prospectiva económica. Suele ser recomendable emplear los mismos valores que están recogidos en los planes financieros de la empresa.
- Tratar de valorar y prever todos los riesgos.

Ejemplo de Aplicación

Se considera una empresa que actualmente lleva su contabilidad de forma manual y decide implantar un sistema informático para que el proceso resulte mas rápido y fiable. Actualmente, dispone de dos personas dedicadas a la contabilidad cuyo coste (sueldo bruto mas Seguridad Social a cargo de la empresa) es de 31.253 € cada uno. Se decide comprar un paquete de contabilidad, cuyo precio es 18.030'36 € y el coste de mantenimiento un 15 % de éste, y que requerirá una adaptación (sólo inicialmente) valorada en 44 jornadas de trabajo de un analista / programador cuya hora se presupuesta en 27'05 €. El ordenador que se requiere se presupuesta en 3005'06 € con un coste de mantenimiento del 10% anual.

El departamento de contabilidad estima que se ahorraría un 50% del tiempo de las dos personas dedicadas a la contabilidad si el sistema fuese automático. Se considera que la vida del sistema es de 4 años. La formación para su manejo requiere una semana y se hace con los propios manuales del paquete.

Se considera que el sistema está plenamente operativo en tres meses.

Estableciendo una tabla coste / beneficio con los siguientes criterios:

- **Beneficio**: Si el sistema fuese automático se ahorraría el 50 % del tiempo de dos personas, es decir, se utilizaría una persona. El ahorro (beneficio) será el sueldo de esa persona a partir del segundo año y 9 meses del primer año dado que el sistema tarda 3 meses en estar operativo.
- **Coste del Hardware**: El primer año es el coste de compra del ordenador y los restantes el coste de mantenimiento (10% del coste inicial).
- **Coste de Software**: Es el precio del paquete para el primer año y el de mantenimiento (15 % del precio) en los sucesivos tres años
- **Coste de Personal informático**: Únicamente el primer año se producirá el coste de 44 jornadas de trabajo de 8 horas / jornada, del analista / programador que factura *27'05 €./hora.*

- **Coste Usuario**: Es el coste de formación de los dos usuarios y se produce únicamente el primer año. Este coste, por persona, será su sueldo bruto semanal (sueldo bruto anual / 52 semanas) multiplicado por el número de semanas que dura la formación (en este caso 1 semana). Debe considerarse que se forman 2 personas.

Año	**1º**	**2º**	**3º**	**4º**
Beneficio	**23.439'75**	**31.253'00**	**31.253'00**	**31.253'00**
Coste Hardware	*3005'06*	*300'51*	*300'51*	*300'51*
Coste Software	*18.030'36*	*2.704'55*	*2.704'55*	*2.704'55*
Coste Personal Informático	*9.521'60*	-	-	-
Coste Usuario	*1.202'04*	-	-	-
Total Costes	**31.759'06**	**3.005'06**	**3.005'06**	**3.005'06**
Beneficio Neto	*(8.319'31)*	*28.247'94*	*28.247'94*	*28.247'94*
Beneficio Neto Acumulado	***(8.319'31)***	***19.928'63***	***48.176'57***	***76.424'51***

Como se observa, a partir del segundo año se obtienen beneficios al desarrollar este proyecto. Sin embargo, los cálculos efectuados no tienen en cuenta la inflación, por lo que se suele calcular el ***valor actual*** corrigiendo el beneficio neto según la siguiente relación:

$$\textbf{Valor actual} = \textbf{Beneficio neto} / (1 + \textbf{tasa inflación})^{\text{año de aplicación} - 1}$$

En el ejemplo, si se considera una tasa de inflación del 10% se tendrían que corregir los datos anteriores de la siguiente manera:

Año	**1º**	**2º**	**3º**	**4º**
Valor Actual	*(8.319'31)*	*25.679'95*	*23.345'40*	*21.233'10*
Beneficio Neto actual Acumulado	*(8.319'31)*	*17.360'64*	*40.706'04*	*62.039'14*

En cuanto al ROI o tiempo en el que se tarda en recuperar el dinero invertido, esto es, el tiempo que se tarda en llegar a un beneficio acumulado nulo, será, supuesta una tasa de inflación nula:

12 Meses del primer año, dado que se alcanza en este año un beneficio negativo mas:

$$8.319'31 + (3005'06 / 12) \times t = (31.253'00 / 12) \times t \Rightarrow t = 3'53 \text{ meses}$$

Por lo tanto el ROI será 12 + 3'453 meses = 15' 53 meses

Para un análisis eficaz de coste / beneficio conviene resaltar:

- La mayoría de las estimaciones de costes y beneficios suelen consistir en rangos de valores probables. Quien tome la decisión de la continuidad del proyecto no puede pretender mayor precisión en esta etapa de decisión. A medida que avance el proyecto podrá refinarse el análisis económico, pero la decisión de realizarlo está ya tomada.
- Es recomendable hacer estimaciones mas bien pesimistas o conservadoras.
- Debe contarse con los distintos factores que influyen en toda prospectiva económica (inflación, tipos de interés, volúmenes de crecimiento, etc.).
- Tratar de valorar y prever todos los riesgos

2.9-2.- Análisis coste/beneficio desde el proyecto.

En lo que respecta al caso del suministrador del software, el análisis coste / beneficio se fundamenta en conceptos distintos:

- **Líneas de codificación**.- Es la forma de cuantificar el trabajo informático. El volumen de líneas de codificación de un proyecto determina la magnitud de éste.
- **Coste por línea de codificación.-** Es el esfuerzo, expresado en términos monetarios, para añadir una línea de codificación en el proyecto en curso.
- **Esfuerzo temporal requerido.-** Es el tiempo en el que se tarda en realizar el proyecto
- **Esfuerzo humano requerido.-** Es el numero de personas necesario para llevar a cabo el proyecto en el tiempo requerido.

Los conceptos anteriores se ilustran en el siguiente ejemplo

Ejemplo de Aplicación

Tomando como base la especificación de un proyecto software, se han identificado las siguientes funciones principales:

- *Subsistema de captura de datos*
- *Subsistema de gestión de la base de datos*
- *Subsistema de gestión de procesos*
- *Subsistema de edición de resultados*
- *Interface de usuario*
- *Control de dispositivos*

Partiendo de un archivo histórico de proyectos, se ha determinado para cada una de las funciones los siguientes valores optimistas, mas probables y pesimistas relativos a las líneas de código necesarias para implantar cada función:

FUNCIONES	Líneas de Código OPTIMISTAS	Líneas de Código MAS PROBABLES	Líneas de Código PESIMISTAS
Subs. Captura de Datos	1500	2000	2500
Subs. Gestión BD	3000	3300	3700
Subs. Gestión Procesos	2500	2800	3200
Subs. Edición Resultados	2400	2900	3300
Interface de Usuario	1800	2200	2600
Control Dispositivos	2000	2300	2600

Se ha observado que la probabilidad de obtención de valores optimistas es la misma que la probabilidad de obtención de valores pesimistas, y que la probabilidad de obtención de valores mas probables es cuatro veces cualquiera de las anteriores.

Por otra parte, según las características del equipo de desarrollo existente, se tienen estipulados, para cada una de las funciones, los siguientes valores unitarios:

FUNCIONES	€ /Línea de Código	Líneas de Código/Mes
Subs. Captura de Datos	60	300

Subs. Gestión BD	78	220
Subs. Gestión Procesos	102	200
Subs. Edición Resultados	108	190
Interface de Usuario	120	180
Control Dispositivos	114	195

Determinar:

1. *El número total de líneas de código del proyecto*
2. *El coste estimado en Euros (€)*
3. *El esfuerzo requerido en meses*

Con los datos especificados en el enunciado se plantea construir el siguiente cuadro de valores:

Funciones	O	MP	P	E	€/L	L/M	Coste	Mes
Subsistema captura de datos	1500	2000	2500		60	300		
Sub. gestión de base de datos	3000	3300	3700		78	220		
Sub. Gestión de procesos	2500	2800	3200		102	200		
Sub. Edición de resultados	2400	2900	3300		108	190		
Interface de Usuario	1800	2200	2600		120	180		
Control de dispositivos	2000	2300	2600		114	185		

Para realizar el cálculo de cada una de las columnas que están en blanco de la tabla anterior, se considera:

$$\textbf{Valor Estimado} = \frac{Optimista + 4 * probable + pesimista}{6}$$

$$\textbf{Coste} = \frac{€}{línea} * \textbf{Valor Estimado}$$

$$\textbf{Meses} = \frac{ValorEstimado}{Líneas\ /\ mes}$$

Calculando estos valores e introduciéndolos en la tabla anterior, se tiene:

Funciones	O	MP	P	E	€/L	L/M	Coste	Mes
Subsistema captura de datos	1500	2000	2500	2000	60	300	120000	6'67
Sub. gestión de base de datos	3000	3300	3700	3315	78	220	258570	15'07
Sub. Gestión de procesos	2500	2800	3200	2815	102	200	287130	14'08

Sub. Edición de resultados	2400	2900	3300	2885	108	190	311580	15'18
Interface de Usuario	1800	2200	2600	2200	120	180	264000	12'22
Control de dispositivos	2000	2300	2600	2300	114	185	425500	12'43

El número total de líneas del proyecto será la suma de las estimadas para cada apartado, esto es: 15515 líneas.

El coste estimado en € será la suma de los costes de cada apartado, es decir: 1.666.780 €

Por último el esfuerzo requerido en meses será: 75'65 meses

2.9-2-1.- Técnicas de análisis coste/beneficio: el modelo *COCOMO*

Este tipo de técnicas sirven para estudiar aquellas características del proyecto que influyen en su viabilidad técnica y económica. Entre dichas técnicas, una de las mas usadas es el modelo COCOMO (COsts COnstrucive MOdel) o Modelo Constructivo de Costes.

El modelo COCOMO define los siguientes **factores de estimación**:

Esfuerzo Estimado (EE).- Es la relación existente entre el número de personas que trabajan en un proyecto y el tiempo de realización del mismo. Su unidad de medida es (persona/mes) y se estima mediante:

$$\mathbf{EE = a.(MLC)^b}$$

Donde:

MLC es el ***tamaño de la aplicación*** expresado en ***miles de líneas de código***

a y **b** son constantes de estimación.

Tiempo Estimado (TE).- Es la cantidad de tiempo que se prevé tardará en desarrollarse el proyecto. Se mide en meses y se estima aplicando la siguiente relación:

$$\mathbf{TE = c.(EE)^d}$$

Donde **c** y **d** son constantes de estimación.

Coste Humano ($Coste_H$).- Es la cantidad de personas que se prevé tendrán que trabajar en el proyecto para cumplir la estimación de tiempos. Se mide en personas y se calcula aplicando:

$$\mathbf{Coste_H = EE/TE}$$

Coste Monetario ($Coste_m$).- Es la cantidad de dinero que se prevé costará el proyecto. Se mide en € y se calcula:

$$\mathbf{Coste_m = Coste_H * TE * Tarifa = EE * Tarifa}$$

Donde **Tarifa** representa el coste total que genera cada persona que trabaja en el proyecto (salario, Seguridad Social, etc.).

Los dos primeros factores son de estimación directa (dependen de la estimación de las constantes a, b, c y d) mientras que los otros dos son derivados de los anteriores.

Las estimaciones de coste y esfuerzo dependen de factores característicos del proyecto como el número de líneas de código del proyecto, o de constantes que dependen de la naturaleza del proyecto.

Por ello, COCOMO distingue tres tipos o escenarios del proyecto, llamados **modos de desarrollo de software**:

Modo orgánico: Se trata de proyectos de *tamaño reducido* (no mas de 50 MLC) para los que se forma un equipo de trabajo de pocas personas con mucha experiencia en el ámbito del problema. Otra característica habitual es la *estabilidad de los requisitos* (esto es, el cliente sabe lo que quiere y no cambia constantemente de opinión) que, unido a la *flexibilidad de las fechas de entrega* crea un ambiente de trabajo relajado.

Modo semilibre: Se trata de proyectos de tamaño medio (en torno a 300 MLC) caracterizados por su heterogeneidad. En el equipo se encuentran miembros con mucha experiencia en ciertas áreas junto con otros muy inexpertos. Los propios requisitos suelen ser también muy variados, pudiendo haber restricciones muy rigurosas y definidas y otras apenas esbozadas.

Modo rígido: Se trata de proyectos de cualquier tamaño (con tendencia, eso si, a no ser pequeños) cuyas características principales son la complejidad y la inflexibilidad, El catálogo de requisitos suele estar plagado de limitaciones estrictas y especificaciones rigurosas (generalmente por tener que cumplir alguna normativa). El entorno de hardware y software suele estar fuertemente acoplado (es decir, no son independientes entre sí) y el área de desarrollo frecuentemente es desconocida y muy especializada, lo que complica bastante las cosas.

Por otra parte, el modelo COCOMO ofrece tres modelos distintos para realizar la estimación, en función de la cantidad de información que se maneje relativa al proyecto o de los factores que se considere puedan afectar a dicha estimación.

Los tres modelos, por orden creciente de complejidad, son los siguientes:

A.- Modelo básico.

Es el adecuado para estimar proyectos pequeños o de tamaño medio desarrollados por personal de la propia empresa y de los que *no se tiene mayor información* acerca ede otros factores que puedan afectar a la estimación.

Suele aplicarse a proyectos de modo orgánico aunque, raramente, peden darse casos de su utso en modos de desarrollo semilibre y rígido.

Para este modelo los valores de las constantes de estimación son:

MODO	*a*	*b*	*c*	*d*
Orgánico	*2'40*	*1'05*	*2'50*	*0'38*
Semilibre	*3'00*	*1'12*	*2'50*	*0'35*
Rígido	*3'60*	*1'20*	*2'50*	*0'32*

Ejemplo de Aplicación

La Empresa CUTRE especializada en el desarrollo de pequeños proyectos de software en breve espacio de tiempo, comienza un nuevo proyecto. Se trata de un sistema cuyos requisitos se encuentran perfectamente definidos y existen pocas probabilidades de que

vayan a modificarse. Se estima que el tamaño del producto final será de 30 MLC. El entorno de trabajo es muy similar al utilizado en otros proyectos anteriores. Determinar:

a.- El esfuerzo de desarrollo del proyecto

b.- El tiempo necesario para desarrollarlo

c.- El coste humano del proyecto

d.- El coste monetario del proyecto si se sabe que el coste mensual por persona afecta al proyecto asciende a 903 €.

Según el enunciado, los puntos

- Requisitos perfectamente definidos
- Entorno de trabajo muy conocido
- Proyecto de pequeño tamaño

Indican que el modo de desarrollo del software es orgánico por lo que se seguirá el modelo básico. Por ello:

a.- Esfuerzo estimado: **EE = a.MLC^b = 2'40*30$^{1'05}$** = 85'35 personas/mes

b.- Tiempo estimado: **TE = c.EE^d = 2'50*85'35$^{0'38}$** = 13'55 meses

c.- Coste humano: **Coste$_H$ = EE /TE = 85'35/13'55** = 6'3 personas

d.- Coste monetario: **Coste$_m$ = Coste$_H$ * TE * Tarifa = 6'3*13'55*903** = 77084'6 €

B.- Modelo intermedio.

Se utiliza cuando en el proyecto se han de tener en cuenta determinados factores que pueden influir en la estimación de costes. A estos factores se les llama ***modificadores***. El modelo COCOMO considera hasta 15 modificadores que se clasifican en cuatro categorías:

- Atributos del software

1º.- Fiabilidad (**RELY**)

2º.- Tamaño de la base de datos (**DATA**)

3º.- Complejidad de las funciones, datos, interfaces, etc. (**CPLX**)

- Atributos del Hardware

4º.- Limitaciones en el porcentaje de uso de la CPU (**TIME**)

5º.- Limitaciones en el porcentaje de uso de memoria (**STOR**)

6º.- Volatilidad de la máquina virtual (**VIRT**)

7º.- Frecuencia de cambio en el modelo de explotación (**TURN**)

- Atributos del personal

8º.- Cualificación de los analistas (**ACAP**)

9º.- Experiencia del personal en aplicaciones similares (**AEXP**)

10º.- Cualificación de los programadores (**PCAP**)

11º.- Experiencia del personal de la máquina virtual (**VEXP**)

12º.- Experiencia en el lenguaje de programación a usar (**LEXP**)

- Atributos del proyecto

13º.- Uso de prácticas modernas de programación (**MODP**)

14º.- Uso de herramientas de desarrollo del software(**TOOL**)

15º.- Limitaciones en el cumplimiento de la planificación (bonificaciones o penalizaciones por adelanto o atraso en la entrega, etc.) (**SCED**).

Cada uno de los quince modificadores puede tomar distintos valores numéricos según sea la valoración que del mismo haga el cliente para el proyecto en cuestión (muy poco, poco, normal, bastante, mucho, etc.).

Multiplicando los valores numéricos de los quince modificadores se obtiene la **función correctora** de la estimación de costes:

m(x)=RELY*DATA*CPLX*TIME*STOR*VIRT*TURN*ACAP*AEXP*PCAP*VEXP*LEXP*MODP*TOOL*SCED

que corrige la Estimación del esfuerzo según:

$$EE' = EE * m(x)$$

Y, consecuéntemente:

$$Coste_H' = EE'/TE$$

Y

$$Coste_m = Coste_H' * TE * Tarifa = EE' * Tarifa$$

Donde EE' representa la **estimación del esfuerzo corregida**.

Una cuestión a la que el modelo intermedio dedica especoal atención es si el proyecto es de ***desarrollo*** o de ***mantenimiento***. Los valores numéricos de los modificadores no son exactamente los mismos en ambos casos y esto sucede porque algunos modificadores se interpretan de manera especial en los proyectos de mantenimiento: El parámetro **SCED** es

El parámetro **SCED** es ***indiferente***, es decir, vale siempre 1. Esto es así porque SCED lo que realmente mide es la importancia de las desviaciones temporales con respecto a lo planificado en a fase de desarrollo, no en la fase de mantenimiento.

El parámetro **RELY** cambia sus valores porque se parte de la base de un software ya existente y probado que, por tanto, ya tiene una fiabilidad determinada.

El parámetro **MODP** pasa a depender de MLC (es decir, del tamaño de la aplicación) y no sólo de la valoración verbal dcl cliente.

Los valores numéricos de los quince modificadores para **proyectos de desarrollo**están recogidos en la siguiente tabla:

GRADO	Muy Bajo	Bajo	Normal	Alto	Muy Alto	Extra Alto
RELY	0.75	0.88	1.00	1.15	1.40	
DATA		0.94	1.00	1.08	1.16	
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
TIME			1.00	1.11	1.30	1.66
STOR			1.00	1.06	1.21	1.56
VIRT		0.87	1.00	1.15	1.30	

TURN		0.87	1.00	1.07	1.15	
ACAP	1.46	1.19	1.00	0.86	0.71	
AEXP	1.29	1.13	1.00	0.91	0.82	
PCAP	1.42	1.17	1.00	0.86	0.70	
VEXP	1.21	1.10	1.00	0.90		
LEXP	1.14	1.07	1.00	0.95		
MODP	1.24	1.10	1.00	0.91	0.82	
TOOL	1.24	1.10	1.00	0.91	0.83	
SCED	1.23	1.08	1.00	1.04	1.10	

Para el caso de ***proyectos de mantenimiento*** los valores de ***RELY*** son

GRADO	**Muy Bajo**	**Bajo**	**Normal**	**Alto**	**Muy Alto**	**Extra Alto**
RELY	1.35	1.15	1.00	0.98	1.10	

Y los de MODP en función del tamaño del proyecto (MLC) son:

MLC	**Muy Bajo**	**Bajo**	**Normal**	**Alto**	**Muy Alto**	**Extra Alto**
2 < x < 8	1.25	1.12	1.00	0.90	0.81	
8 < x < 32	1.30	1.14	1.00	0.88	0.77	
32 < x < 128	1.35	1.14	1.00	0.86	0.74	
128 < x < 512	1.40	1.18	1.00	0.85	0.72	
> 512	1.45	1.20	1.00	0.84	0.70	

Por último, las ***constantes de estimación a, b, c*** y ***d*** para este modelo son los siguientes

MODO	***a***	***b***	***c***	***d***
Orgánico	*3'20*	*1'05*	*2'50*	*0'38*
Semilibre	*3'00*	*1'12*	*2'50*	*0'35*
Rígido	*2'80*	*1'20*	*2'50*	*0'32*

Ejemplo de Aplicación

La editorial TBO, especializada en cómics desea un nuevo sistema de gestión parea sus colecciones, campañas de promoción y suscripciones. Se trata de un área de trabajo completamente nuevo para el equipo de desarrollo del proyecto. Además existen severas restricciones en cuanto a los procesos de tratamiento de los datos se refiere y el entorno de hardware es fuertemente acoplado. El tamaño del producto final se estima en 130 MLC.

Se piensa emplear en este proyecto un pequeño equipo de analistas expertos, al mando de un grupo numeroso de programadores con un nivel medio de experiencia.

Se trata de un software de baja fiabilidad, en el que se piensan emplear muchas prácticas modernas de programación. Por otra parte, el uso de herramientas de desarrollo se limitará a la mínima expresión. El sistema maneja una base de datos de pequeño tamaño.

Los procesos que componen el software son bastante sencillos y se trata de software bastante interactivo. Se pide estimar:

a) El esfuerzo de desarrollo del proyecto, ajustado.
b) El tiempo necesario para desarrollarlo.
e) El coste humano del proyecto.
d) El coste monetario del proyecto, si la tarifa por hombre y mes de trabajo es de 1750 €.

Tamaño = MLC = 130

Área de trabajo completamente nuevo, Severas restricciones, Entorno de hardware fuertemente acoplado implican la existencia de un ***Modo Rígido***

Analistas expertos, Programadores nivel medio de experiencia, Baja fiabilidad, Muchas prácticas modernas de programación, Mínimo uso de herramientas de desarrollo, Base de datos pequeña, Procesos bastante sencillos, Software bastante interactivo son todos ellos síntomas de un ***Modelo Intermedio***

Por tanto:

ACAP	(Muy Alto)	0'71 (analistas expertos)
PCAP	(Normal)	1'00 (programadores con un nivel medio de experiencia)
RELY	(Bajo)	0'88 (baja fiabilidad)
MODP	(Alto)	0'91 (muchas prácticas modernas de programación)
TOOL	(Muy Bajo)	1'24 (mínimo uso de herramientas de desarrollo)
DATA	(Bajo)	0'94 (Base de datos pequeña)
CPLX	(Bajo)	0'85 (Procesos bastante sencillos)
TURN	(Alto)	1'07 (Software bastante interactivo)

La función correctora es

m(x) = RELY*DATA*CPLX*TURN*ACAP*PCAP*MODP*TOOL = 0'6

por tanto:

Esfuerzo estimado **EE** = $\mathbf{a.MLC^b.m(x)}$ = $\mathbf{2'8 * (130)^{1'2} * 0'6}$ = 578'15 personas/mes

Tiempo Estimado **TE** = $\mathbf{c.EE^d}$ = $\mathbf{2'50 *(578'15)^{1'2}}$ = 19'13 meses

Coste humano $\mathbf{Coste_H}$ = **EE/TE** = **578'15/19'13** = 30'22 personas

Coste monetario $\mathbf{Coste_m}$= **TE * $Coste_H$ * Tarifa** = **19'13 * 30'22 * 1750** = 1011690 €

C.-Modelo detallado

Es, sin duda, el mas preciso pero también el mas complejo. Trata de superar lasa limitaciones que el modelo intermedio manifiesta en grandes proyectos: Distribución por fases del esfuerzo inadecuada y dificultad de los cálculos en la estimación del desarrollo del software formado por muchos componentes. Para ello define unos ***multiplicadores*** que redistribuyen el esfuerzo por fases y propone una ***descomposición jerárquica del producto*** en tres niveles: módulo, subsistema y sistema.

2.10.- TÉCNICAS DE RECOGIDA DE INFORMACIÓN

Surgen como medio para mejorar la comunicación entre usuarios / clientes y los desarrolladores software.

Normalmente los técnicos de desarrollo de software no conocen todos los detalles del trabajo de la empresa para la cual van a desarrollar la aplicación. Por otra parte los usuarios no saben que información es necesaria o relevante para el desarrollo de una aplicación. Para facilitar la comunicación de ambos colectivos en el proceso de análisis de necesidades, se recurre a técnicas de comunicación y recopilación de información.

Este proceso de análisis debería seguir los siguientes cinco pasos:

1) **Identificar las fuentes de información** relevantes para el proyecto.
2) **Realizar las preguntas apropiadas** para comprender sus necesidades.
3) **Analizar la información** recogida para detectar los aspectos que quedan poco claros.
4) **Confirmar con los usuarios** lo que parece haberse comprendido de los requisitos.
5) **Sintetizar los requisitos** en un documento de especificación apropiado. La razón del uso de técnicas de recogida de información reside en las diversas barreras que existen para una comunicación eficaz entre analista y usuario.

El resultado del proceso será un documento que especifique, lo mas claramente posible, los requisitos que debe cumplir el software.

Las técnicas que ayudan a obtener la información necesaria, bien para genera la especificación, bien para realizar un análisis de viabilidad, mas utilizadas son:

- **Entrevistas.** La técnica mas empleada. Requiere una mayor preparación y experiencia por parte del analista
- **Desarrollo conjunto de aplicaciones (JAD)** *(Join Application Design).* Creación de equipos de usuarios y analistas que se reúnen para trabajar conjuntamente en la determinación de las características que debe tener el software para satisfacer las necesidades de los usuarios. Involucra al usuario en el proyecto que lo aprecia como algo propio.
- **Prototipado**. Construcción de un modelo o maqueta del sistema que permite a los usuarios evaluar mejor sus necesidades, analizando si el prototipo que ven tiene las características de la aplicación que necesitan.
- **Observación**. Análisis *in situ* de cómo funciona la unidad o el departamento que se quiere informatizar. Aporta grandes ventajas sobre las otras técnicas, ya que se pueden analizar mejor todos los detalles del proceso y se llega a captar el funcionamiento real de la empresa, así como el ambiente en el que se va a desarrollar el proyecto y se va a instalar en el futuro sistema.
- **Estudio de Documentación**. En casi todas las organizaciones existen documentos que describen el funcionamiento del negocio, desde planes estratégicos hasta manuales de operación. El analista debe estudiar esta documentación para hacerse una idea de la

normativa que rige la empresa. Es conveniente, asimismo que recopile y estudie los impresos habitualmente utilizados para conocer los datos que manejan.

- **Cuestionarios**. Útiles para recoger información de un gran número de personas en poco tiempo. Es especialmente útil en situaciones en las que se da una gran dispersión geográfica.
- **Tormenta de ideas** (*Brainstorming*). Reuniones de cuatro a diez personas (usuarios) en las cuales, en una pruimera fase , se sugieren toda clase de ideas sin juzgarse su validez, por muy disparatadas que parezcan. En una segunda fase se realiza un análisis detallado de cada propuesta. La utilización de este tipo de reuniones es muy útil para identificar un primer conjunto de requisitos en aquellos caso en los que no están muy claras las necesidades que hay que cubrir.
- **ETHICS** *(Effective Technical and Human Implementation of Computer – based Systems)*. Es un método bastante evolucionado para fomentar la participación de los usuarios en los proyectos. Coordina la perspectiva social de los sistemas con su implementación técnica. Un sistema no tiene éxito si no se ajusta a los factores sociales y organizativos que rigen en la empresa. Se busca la satisfacción de los empleados en el trabajo mediante estudios integrales (similares a los realizados por RR. HH.) basados en factores como el conocimiento, la psicología , la eficiencia, la motivación, etc. Los requisitos técnicos del sistema serán los necesarios para mejorar la situación de los empleados, y por tanto su productividad, en función de dicho análisis.

En general es habitual realizar una combinación de distintas técnicas para recoger la información de los usuarios. Las entrevistas, el JAD y el prototipado suelen emplearse coordinadas por lo que se detallan a continuación.

2.10-1.- Las entrevistas

Se puede definir como un intento sistemático de recoger información de otra persona, a través de una comunicación interpersonal que se lleva a cabo por medio de una conversación estructurada.

El aspecto mas destacable de una entrevista es que se propone un fin ajeno al simple placer de la conversación por lo que la preparación resulta esencial para cumplir sus objetivos.

En la entrevista destacan dos elementos principales: el entrevistador y el entrevistado, existiendo entre ellos una relación asimétrica, dinámica y única. No debe bastar con hacer preguntas para obtener toda la información necesaria. Es muy importante la forma en la que se plantea la conversación y la relación que se establece en la entrevista.

El entrevistador, pues, debe poseer una serie de cualidades como:

- **Imparcial.**
- **Ponderado.**
- **Buen oyente.**
- **Cierto grado de habilidad en el trato.**
- **Cordialidad y accesibilidad.**
- **Paciencia.**

Entre las **Fases** que se pueden distinguir en una entrevista se pueden considerar:

PREPARACIÓN

De forma resumida se presentan algunas de las actividades que el entrevistador debería realizar durante la fase de preparación

- **documentarse e investigar** la situación de la organización, analizando los documentos de la empresa disponibles, incluso recurriendo a fuentes externas
- **Identificar las personas a las que se debe entrevistar.** Suele comenzarse por un enfoque top – down comenzando por entrevistar a los directivos que pueden ofrecer una visión global y terminando por hablar con los empleados que pueden aportar detalles importantes del funcionamiento real de la empresa. No deben desestimarse "entrevistas de cortesía"
- **analizar el perfil de los entrevistados.** Ello permite "romper el hielo" al empezar la entrevista, buscando algún punto de contacto.
- **definir el objetivo y contenido de la entrevista.**
- **Planificar el lugar y la hora en la que se va a desarrollar.**

Algunos autores proponen enviar previamente al entrevistado un cuestionario que permite que éste conozca previamente los temas a tratar y el analista pueda recoger información valiosa información para preparar la conversación. Un ejemplo de este pequeño cuestionario sería:

CUESTIONARIO PREVIO

Puesto: Vendedor de Mostrador

1. ¿Qué pasos sigue para gestionar el cobro de una venta?
2. ¿Existen distintos procedimientos según el tipo de artículo vendido, tipo de cliente, forma de pago, domicilio del cliente, etc.?
3. ¿Qué datos se requieren habitualmente para realizar una gestión de cobro?
4. ¿Se necesitan datos adicionales? ¿Bajo qué circunstancias?
5. ¿Cuántas ventas se realizan diariamente o a la semana? ¿Qué picos y valles existen en las ventas?
6. ¿Existen problemas en el proceso? Identificar los posibles cuellos de botella, retrasos, etc.

REALIZACIÓN.

Es el núcleo central de la entrevista, como en todas las interacciones humanas conviene seguir un protocolo o serie de actos habituales para las conversaciones y reuniones que marcan las costumbres sociales

En el acto de la entrevista se distinguen tres etapas:

- **Apertura:** presentación e información al entrevistado sobre la razón, uso de la información, etc.
- **Desarrollo:** Etapa en la que se pueden emplear las técnicas:
  - *Preguntas abiertas.*
  - *Utilizar las palabras y las frases apropiadas, evitando tecnicismos.*
  - *Asentir y muestras de escucha.*
  - *Repetir las respuestas dadas.*
  - *Pausas.*

Lo ideal sería que el entrevistador hable durante aproximadamente un 20% del tiempo y el entrevistado el 80% restante.

Tomar notas puede ser un asunto delicado y más aún grabar las entrevistas, dado que pueden cohibir al entrevistado. Lo más usual es tomar notas escritas centrándose en los puntos esenciales.

- **Terminación:** dejando abierta la posibilidad de volver a contactar para aclarar conceptos

ANÁLISIS

Es la fase mas descuidada de la entrevista pero no por ello la menos importante. Consiste en leer las notas, reorganizar la información, contrastarla con otras entrevistas u otras fuentes de información, así como evaluar cómo ha ido la entrevista y qué aspectos se pueden mejorar para las sucesivas.

2.10–2.- Desarrollo conjunto de aplicaciones (JAD).

Promueve la cooperación y el trabajo en equipo entre usuarios y analistas, mediante un conjunto de reuniones (Workshop) aprovechando la dinámica de grupos, de una duración de unos cuatro días, en las que participan usuarios cualificados junto con analistas de software.

Las razones que sirven de base a JAD son:

- El **tiempo** requerido por el método JAD es sensiblemente inferior al requerido en la realización de entrevistas
- Es más fácil **apreciar posibles errores** en la especificación de requisitos ya que todo el JAD puede actuar como revisor y detectar defectos.
- El JAD propugna una **participación más profunda** de los usuarios en el proyecto.

Un proceso típico de JAD consta de las siguientes fases:

- **Adaptación o preparación**: Las distintas tareas a realizar para la preparación de la sesión por la persona que actúe como jefe del JAD son:
  - *Selección de los participantes*.
  - *Recabar una cierta información* sobre el sistema que se va a construir.
  - *Organizar la reunión*, lugar, fecha, agenda de trabajo, redacción de un documento de trabajo.
- **Sesión JAD**: se parte de un documento de trabajo que hay que analizar para completar el conjunto de requisitos del sistema. Al final de la sesión se tendrá concluido un documento de especificación que deberá ser aprobado por los presentes.
- **Documentación:** redactar y documentar los detalles, etc para obtener el documento de especificación de requisitos en su forma final.

El JAD no se utiliza demasiado ya que requiere una mayor organización que las entrevistas y porque el ambiente y los métodos de trabajo convencionales no facilitan este tipo de actividades. Sin embargo, la utilización de este método produce importantes ahorros de tiempo en el desarrollo del software así como una mayor satisfacción en el usuario con los sistemas construidos.

2.10–3.- Prototipado

Consiste en la elaboración de un modelo o maqueta del sistema que se construye para evaluar mejor los requisitos que se desea que cumpla.

Es particularmente útil cuando:

- *El área de aplicación no está bien definida.*
- *El coste del rechazo de la aplicación por los usuarios, por no cumplir sus expectativas, es muy alto.*

- *Es necesario evaluar previamente el impacto del sistema en los usuarios y en la organización.*

Estos modelos o prototipos suelen consistir en versiones reducidas, *demos* o conjunto de pantallas (que no son totalmente operativos) de la aplicación pedida.

Las razones principales para emplear el prototipado, ordenadas por frecuencia de uso, son:

1) **Prototipado de la interfaz de usuario** par asegurar que está bien diseñada. Efectivo para evitar los múltiples cambios que suelen solicitar los usuarios.
2) **Modelos de rendimiento** para evaluar un diseño técnico.
3) **Prototipado funcional**, relacionado con un ciclo de vida iterativo, supone una primera versión limitada del sistema.

La cualidad fundamental del prototipo debe ser la posibilidad de ser construido más rápidamente que la aplicación correspondiente. Las herramientas empleadas pueden ser:

- En el nivel inferior se encuentran las herramientas comunes (programas de dibujo, de presentación, hojas de cálculo o generadores de informes, etc.) El objeto es que el usuario pueda ver la entrada / salida de la aplicación cuando esté acabada.
- En un nivel intermedio estarían los gestores de bases de datos y sistemas de cuarta generación que permiten prototipos mas sofisticados no sólo de interfaces, sino también de manejo de datos,
- Herramientas CASE o generadores de aplicaciones, etc.

El prototipado para análisis de necesidades es un medio que permite solventar objeciones del usuario del tipo: "No sé exactamente lo que quiero, pero lo sabré en cuanto lo vea".

El prototipado se evalúa en aproximadamente un 10 % del presupuesto del proyecto y suele llevar muy pocos días en su desarrollo. En algunos casos, se utiliza para formalizar la aceptación previa (firma del contrato) del cliente de los requisitos del proyecto.