

U.D.1

Desarrollo de Software

(1ª parte)

ED – IES EL MAJUELO

ÍNDICE

1. El software de ordenador.
2. Ciclo de vida del software.
3. Fases del desarrollo de una aplicación.
 1. Análisis.
 2. Diseño.
 3. Codificación.
 4. Pruebas.
 5. Documentación.
 6. Explotación.
 7. Mantenimiento.

1.- EL SOFTWARE DEL ORDENADOR

- RAE: *“Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora”*.
- IEEE 729: *“Conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación”*.

SOFTWARE

(1) instrucciones de ordenador que cuando se ejecutan proporcionan la función y el comportamiento deseado, (2) estructuras de datos que facilitan a los programas manipular adecuadamente la información, y (3) documentos que describen la operación y el uso de los programas.

Naturaleza lógica e inmaterial

ELEMENTOS FUNDAMENTALES

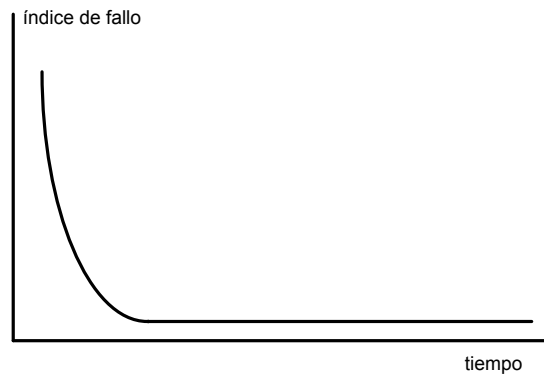
- Programas:
 - Encargados de dar instrucciones para realizar tareas con el hardware o para comunicarnos con otro software.
- Datos:
 - Necesarios para la ejecución de los programas.

CARACTERÍSTICAS DEL SOFTWARE (I)

- El software se desarrolla, no se fabrica en sentido estricto.
 - Los costes del software se encuentran en la ingeniería (incluyendo en ésta el desarrollo), y no en la producción.

CARACTERÍSTICAS DEL SOFTWARE (II)

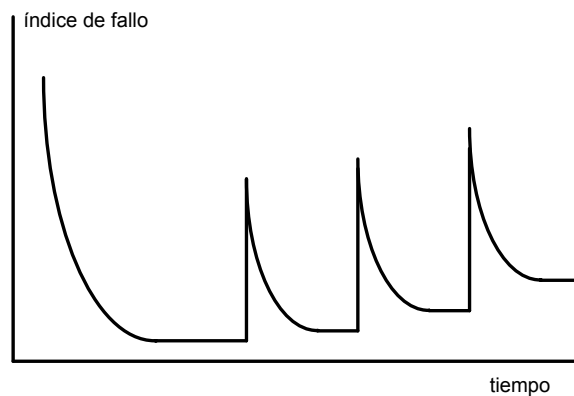
- El software no se estropea...



Curva ideal de fallos del software

CARACTERÍSTICAS DEL SOFTWARE (III)

... pero se deteriora.



Curva real de fallos del software

CARACTERÍSTICAS DEL SOFTWARE (IV)

- La mayoría del software se construye a medida.
 - No existen catálogos de componentes.
 - El uso de bibliotecas (librerías) es costoso y no siempre fácil.
 - La documentación y difusión de módulos en programación estructurada son deficientes.
 - La orientación a objetos no está extendida.

PROBLEMAS DEL SOFTWARE

- La planificación y la estimación de costes son muy imprecisas.
- La productividad es baja.
- La calidad es mala.
- El cliente queda insatisfecho.

LA INGENIERÍA DEL SOFTWARE

No existe una fórmula mágica para solucionar estos problemas, pero:

- combinando métodos aplicables a cada una de las fases del desarrollo de software,
- construyendo herramientas para automatizar estos métodos,
- utilizando técnicas para garantizar la calidad de los productos desarrollados y
- coordinando todas las personas involucradas en el desarrollo de un proyecto.

podremos avanzar mucho en la solución de estos problemas. De ello se encarga la disciplina llamada ***Ingeniería del Software***.

DEFINICIÓN DE ISW

El establecimiento y uso de principios de ingeniería robustos, orientados a obtener software económico, que sea fiable y funcione de manera eficiente sobre máquinas reales.

Tres elementos clave:

- Métodos.
- Herramientas.
- Procedimientos.

ELEMENTOS DE ISW (I)

- **MÉTODOS:**
 - Indican cómo construir técnicamente el software.
 - Abarcan una amplia gama de tareas que incluyen:
 - Planificación y estimación de proyectos.
 - Análisis de requisitos.
 - Diseño de estructuras de datos, programas y procedimientos.
 - Codificación.
 - Pruebas y Mantenimiento.
 - Introducen frecuentemente una notación específica para la tarea en cuestión y una serie de criterios de calidad.

ELEMENTOS DE ISW (II)

- **HERRAMIENTAS:**
 - Proporcionan un soporte automático o semiautomático para utilizar los métodos.
 - Existen herramientas automatizadas para cada una de las fases vistas anteriormente, y sistemas que integran las herramientas de cada fase de forma que sirven para todo el proceso de desarrollo.

ELEMENTOS DE ISW (III)

- **PROCEDIMIENTOS:**
 - Definen la secuencia en que se aplican los métodos.
 - Definen los documentos que se requieren.
 - Definen los controles que permiten asegurar la calidad.
 - Definen las directrices que permiten a los gestores evaluar los progresos.

SEGÚN EL TIPO DE TAREA QUE REALIZA

- Software de sistema.
 - Permite que el hardware funcione.
 - Administra los recursos del ordenador.
 - Interactúa con el usuario y con el hardware.
- Software de aplicación.
 - Ayudan al usuario a realizar tareas específicas en cualquier campo.
 - Convierte el ordenador en una herramienta útil para el usuario.
- Software de programación o desarrollo.
 - Proporciona al programador herramientas para ayudarle a escribir programas informáticos.
 - Está asociado a lenguajes de programación.

SEGÚN EL MÉTODO DE DISTRIBUCIÓN

- Shareware.
 - Permite evaluar un producto por un tiempo especificado.
 - Requiere adquirir finalmente una licencia para continuar usándolo transcurrido ese tiempo.
- Freeware.
 - Se distribuye sin cargo.
- Adware.
 - Añaden programas extra y/o publicidad en el proceso de instalación.
- Software de uso específico.
 - Se desarrolla específicamente para resolver un programa determinado.

LICENCIAS DE SOFTWARE

- Contrato entre el desarrollador de un software (sometido a propiedad intelectual y derechos de autor) y el usuario.
- Se definen los derechos y deberes de ambas partes.

SOFTWARE DE DOMINIO PÚBLICO

- Carece de licencia o no hay forma de determinarla pues se desconoce el autor.
- El propietario abandona los derechos que le acreditan o se produce la extinción de la propiedad por expiración del plazo de la misma.
- No pertenece a una persona concreta sino que todo el mundo lo puede utilizar.

SOFTWARE PROPIETARIO

- Se distribuye en formato binario.
- No incluye el código fuente.
- El propietario prohíbe alguna o todas las siguientes posibilidades:
 - Redistribución.
 - Copia.
 - Uso en varias máquinas simultáneamente.
 - Transferencia de titularidad.
 - Difusión de fallos.

SOFTWARE LIBRE

- El autor cede una serie de libertades al usuario:
 - Libertad de utilizar el programa con cualquier fin en cuantos ordenadores se desee.
 - Libertad de estudiar cómo funciona el programa y de adaptar su código a necesidades específicas.
 - Libertad de distribuir copias a otros usuarios (con o sin modificaciones).
 - Libertad de mejorar el programa y de hacer públicas y distribuir las modificaciones.
- La licencia más utilizada es la GPL.

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

2.- CICLO DE VIDA DEL SOFTWARE

“Un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento del un producto de software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso”.

ISO/IEC 12207-1

EL CICLO DE VIDA (I)

- Por **ciclo de vida**, se entiende la sucesión de etapas por las que pasa el software desde que un nuevo proyecto es concebido hasta que se considera una solución completa, correcta y estable (o incluso hasta que se deja de usar).
- Cada una de estas etapas lleva asociada una serie de tareas que deben realizarse, y una serie de **documentos** (en sentido amplio: *software*) que serán la salida de cada una de estas fases y servirán de entrada en la fase siguiente.

EL CICLO DE VIDA (II)

- Existen diversos modelos de ciclo de vida, es decir, diversas formas de ver el proceso de desarrollo de software, y cada uno de ellos va asociado a un paradigma de la ingeniería del software, es decir, a una serie de métodos, herramientas y procedimientos que debemos usar a lo largo de un proyecto.
- La elección de un paradigma u otro se realiza de acuerdo con la naturaleza del proyecto y de la aplicación, los métodos a usar y los controles y entregas requeridos.

ETAPAS

- Análisis.
 - Diseño.
 - Codificación.
 - Pruebas.
 - Documentación de usuario.
 - Implantación.
 - Mantenimiento.
-
- Cada etapa genera documentación.
 - Hay más de un modelo de etapas de desarrollo (ciclos de vida).

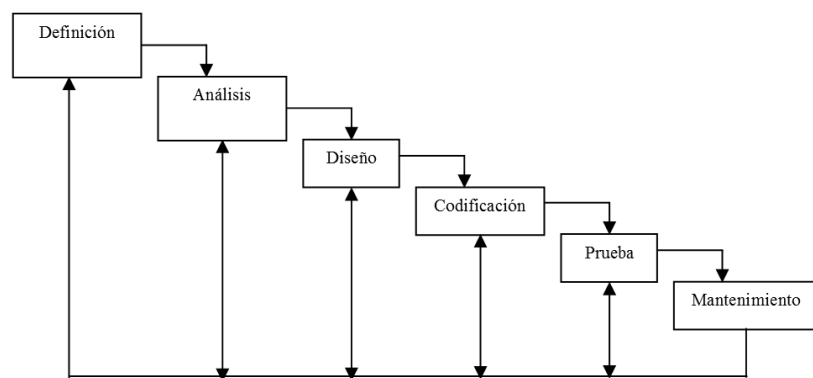
MODELOS DE CICLO DE VIDA

- Ciclo de vida en cascada.
- Modelos evolutivos.
 - Modelo iterativo incremental.
 - Modelo en espiral.
 - Modelo de prototipos.

CICLO DE VIDA EN CASCADA

- Para empezar una etapa es necesario finalizar la etapa anterior.
- Este modelo permite hacer iteraciones por detectar fallos o posibles mejoras en cualquiera de las etapas.
 - Requiere revisar todas las etapas anteriores a aquella en la que se ha detectado.

CICLO DE VIDA EN CASCADA



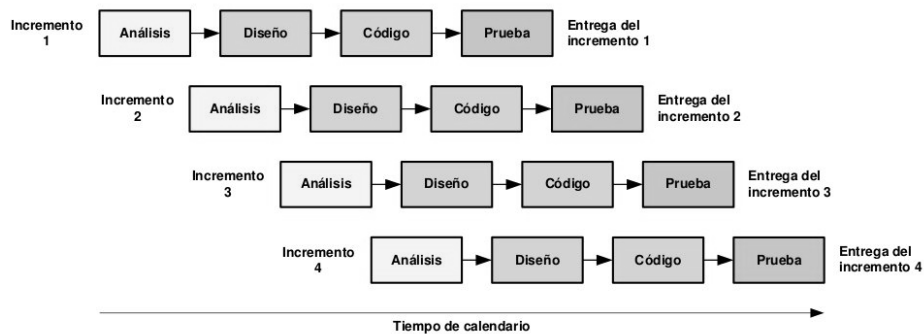
CICLO DE VIDA EN CASCADA

- **Ventajas:**
 - Fácil de comprender, planificar y seguir.
 - Calidad alta.
 - Permite trabajar con personal menos cualificado.
- **Inconvenientes:**
 - Necesidad de tener todos los requisitos definidos desde el principio.
 - Dificultad en volver atrás.
 - El producto no está disponible para su uso hasta que no está terminado.
- **Se recomienda cuando:**
 - El proyecto es similar a alguno que se haya completado con éxito anteriormente.
 - Los requisitos son estables y están bien comprendidos.
 - El cliente no necesita versiones intermedias.

MODELO ITERATIVO INCREMENTAL

- Está basado en varios ciclos cascada realimentados, aplicados repetidamente.
- Entrega el software en partes pequeñas, pero utilizables (incrementos).
- Cada incremento se construye sobre aquél que ya ha sido entregado.

MODELO ITERATIVO INCREMENTAL



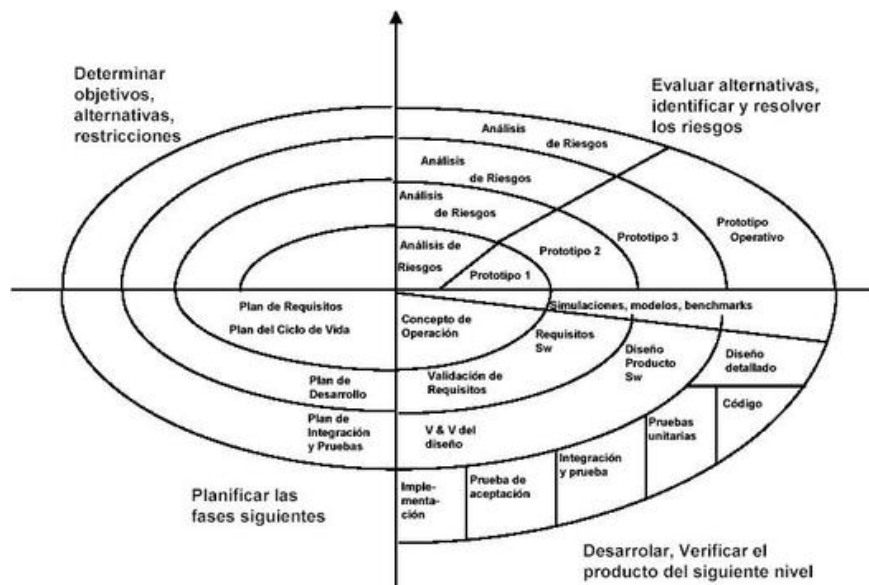
MODELO ITERATIVO INCREMENTAL

- **Ventajas:**
 - No se necesitan conocer todos los requisitos al comienzo.
 - Permite la entrega temprana al cliente de partes operativas del software.
 - Las entregas facilitan la realimentación de los próximos entregables.
- **Inconvenientes:**
 - Es difícil estimar el esfuerzo y el coste final necesario.
 - Se tiene el riesgo de no acabar nunca.
 - No recomendable para desarrollo de sistemas de tiempo real, de alto nivel de seguridad, de procesamiento distribuido y/o de alto índice de riesgos.
- **Se recomienda cuando:**
 - Los requisitos o el diseño no están completamente definidos y es posible que haya grandes cambios.
 - Se están probando o introduciendo nuevas tecnologías.

MODELO EN ESPIRAL

- El proceso de desarrollo se representa como una espiral. En cada ciclo se desarrolla una buena parte del mismo.
- Está formado por cuatro fases:
 - Determinación de objetivos.
 - Identificación y resolución de riesgos.
 - Desarrollo y pruebas.
 - Planificación.
- Cuando termina produce una versión incremental del software con respecto al ciclo anterior.
- En todos los ciclos se hace un análisis de riesgo donde se evalúan las alternativas (según los requisitos y restricciones) y se construyen prototipos para analizarlas y seleccionar una.

MODELO EN ESPIRAL

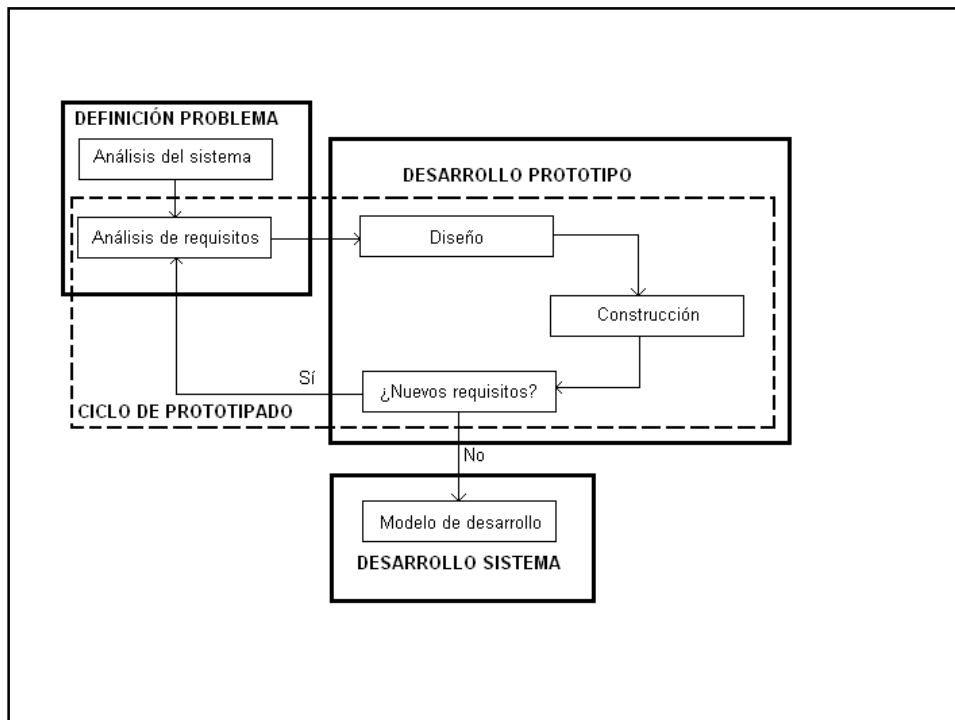


MODELO EN ESPIRAL

- **Ventajas:**
 - No requiere una definición completa de los requisitos para empezar a funcionar.
 - Se analiza el riesgo en todas las etapas.
 - Incorpora objetivos de calidad.
- **Inconvenientes:**
 - El costo del proyecto aumenta a medida que la espiral pasa por sucesivas iteraciones.
 - El éxito del proyecto depende en gran medida de la fase de análisis de riesgos (dificultad de evaluarlos).
- **Se recomienda cuando:**
 - Proyectos de gran tamaño y que necesitan constantes cambios.
 - Proyectos donde sea importante el factor riesgo.

MODELO DE PROTOTIPOS

- **Diseño rápido:**
 - Representación de aquellos aspectos del software que serán visibles para el cliente o el usuario final.
 - Evaluado por el cliente para una retroalimentación.
 - Permite que el desarrollador entienda mejor lo que se debe hacer y el cliente vea resultados a corto plazo.
- **Etapas:**
 - Plan rápido.
 - Modelado, diseño rápido.
 - Construcción del Prototipo.
 - Desarrollo, entrega y retroalimentación.
 - Comunicación.
 - Entrega del desarrollo final.



MODELO DE PROTOTIPOS

- **Ventajas:**
 - Útil cuando el cliente conoce los objetivos generales para el software, pero no identifica los requisitos detallados de entrada, procesamiento o salida.
 - Útil cuando el responsable del desarrollo del software está inseguro de la eficacia de un algoritmo, de la adaptabilidad de un sistema operativo o de la forma que debería tomar la interacción humano-máquina.
 - Se puede reutilizar el código.
- **Inconvenientes:**
 - El usuario tiende a crearse unas expectativas cuando ve el prototipo de cara al sistema final.
 - Se suelen desatender aspectos importantes.

3.- FASES DEL DESARROLLO DE UNA APLICACIÓN

3.1.- ANÁLISIS

- Entender el problema que se necesita resolver → Darle una solución.
- Se analizan y especifican los requisitos o capacidades que el sistema debe tener porque el cliente así lo demanda.
- Problemas:
 - El cliente puede no tener claros los requisitos.
 - Pueden surgir nuevos requisitos que cambien lo especificado.
 - Falta de conocimientos técnicos por parte del cliente.
 - Falta de entendimiento.

TÉCNICAS PARA LA ESPECIFICACIÓN DE REQUISITOS

- Entrevistas.
 - Hablar con el cliente.
- Desarrollo conjunto de aplicaciones.
 - Entrevista estructurada aplicable a grupos de personas.
- Planificación conjunta de requisitos.
 - Con la alta dirección, requisitos de alto nivel o estratégicos.
- Brainstorming.
 - Generar ideas desde diferentes puntos de vista.
- Prototipos.
 - Versión inicial del sistema para clarificar y demostrar conceptos.
- Casos de uso.
 - Representación de escenarios que describen el comportamiento deseado del sistema.

TIPOS DE REQUISITOS

- **Funcionales:**
 - Funciones que realiza el sistema.
 - Cómo reacciona ante determinadas entradas.
 - Cómo se comporta en situaciones particulares.
 - Etc.
- **No funcionales:**
 - Características del sistema.
 - Parámetros: fiabilidad, mantenibilidad.
 - Sistema Operativo.
 - Plataforma hardware.
 - Restricciones, limitaciones.

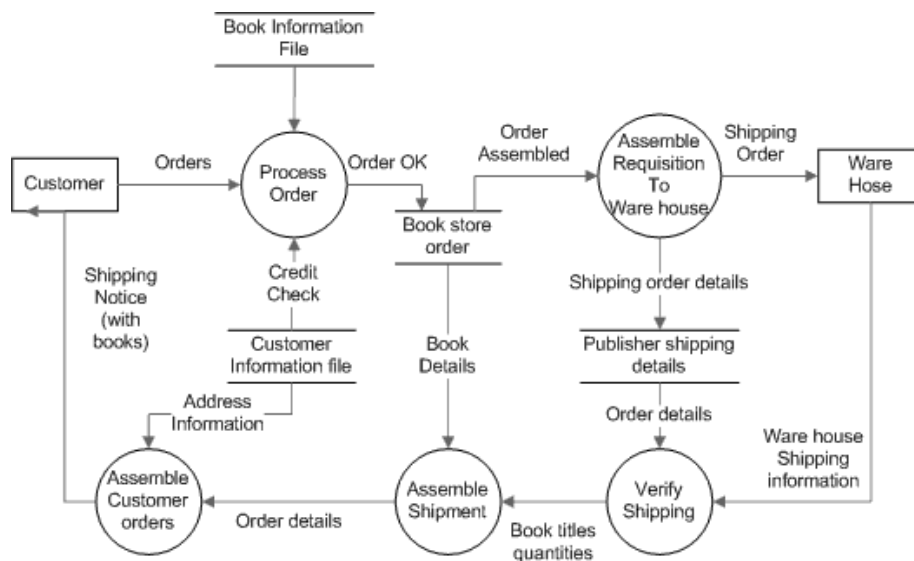
EJEMPLO (AGENDA DE CONTACTOS)

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

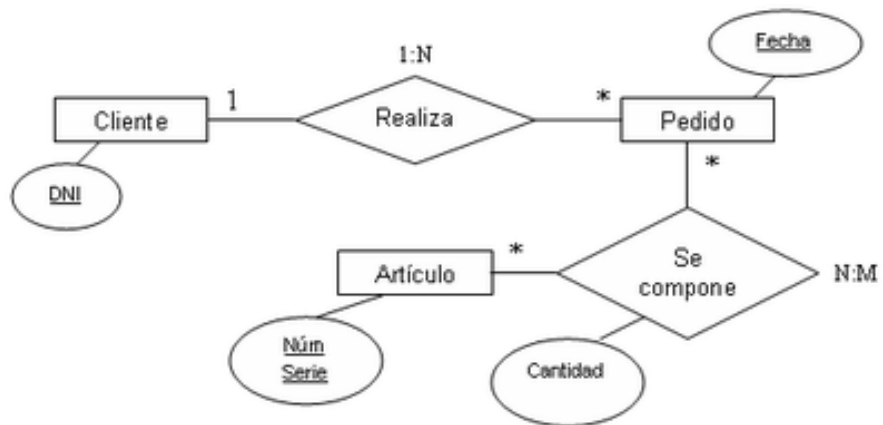
TÉCNICAS PARA LA REPRESENTACIÓN DE REQUISITOS

- Diagrama de flujo de datos.
 - Representa el flujo de datos entre los distintos procesos, entidades externas y almacenes que forman el sistema.
- Diagrama de flujo de control.
 - Similar al anterior pero con flujos de control, no de datos.
- Diagrama de transición de estados.
 - Representa cómo se comporta el sistema como consecuencia de sucesos externos.
- Diagrama entidad / relación.
 - Se usa para representar los datos y las relaciones entre ellos.
 - Basado en la arquitectura relacional de bases de datos.
- Diccionario de datos.
 - Descripción detallada de los datos utilizados por el sistema.

DFD



DER



ERS (ESPECIFICACIÓN DE REQUISITOS DE SOFTWARE). IEEE 830 (1998)

1. Introducción.

1. Propósito.
2. Ámbito del Sistema.
3. Definiciones. Acrónimos y Abreviaturas.
4. Referencias.
5. Visión general del documento.

2. Descripción general.

1. Perspectivas del Producto.
2. Funciones del Producto.
3. Características de los usuarios.
4. Restricciones.
5. Suposiciones y Dependencias.
6. Requisitos Futuros.

3. Requisitos Específicos.

1. Interfaces Externas.
2. Funciones.
3. Requisitos de Rendimiento.
4. Restricciones de Diseño.
5. Atributos del Sistema.
6. Otros Requisitos.

4. Apéndices.

3.- FASES DEL DESARROLLO DE UNA APLICACIÓN

3.2.- DISEÑO

- Componer la forma en que se solucionará el problema.
- Traducción de los requisitos funcionales y no funcionales en una representación del software.
- Dos estrategias de diseño:
 - Diseño estructurado.
 - Basado en el flujo de datos a través del sistema.
 - Diseño orientado a objetos.
 - El sistema se entiende como un conjunto de objetos, con propiedades y comportamiento, y de eventos que modifican su estado.

ELEMENTOS DEL DISEÑO ESTRUCTURADO

- Diseño de datos.
 - Estructuras de datos.
 - Relaciones entre datos.
- Diseño arquitectónico.
 - Estructura de los componentes del software.
- Diseño de la interfaz.
 - Cómo se comunica el software con el sistema y con el usuario.
- Diseño de procedimientos (componentes).
 - Descripción procedimental de los componentes del software.
 - Guía para la codificación.

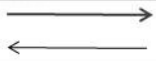


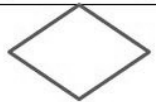


CONSTRUCCIONES (PROGRAMACIÓN ESTRUCTURADA)

- Secuencial.
- Condicional.
 - Permite seleccionar un proceso u otro a partir de la evaluación de una condición lógica.
- Repetitiva.
 - Permite repetir un proceso mientras/hasta que una condición se cumpla o deje de cumplirse.
 - Bucles.

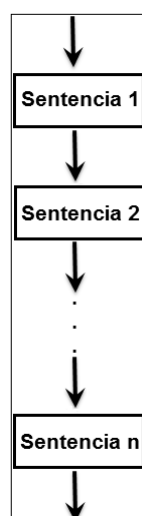
NOTACIONES GRÁFICAS

- Diagrama de flujo (ordinograma).
- Diagrama de cajas (*Nassi-Schneidermann*).
- Tabla de decisión.
- Pseudocódigo.

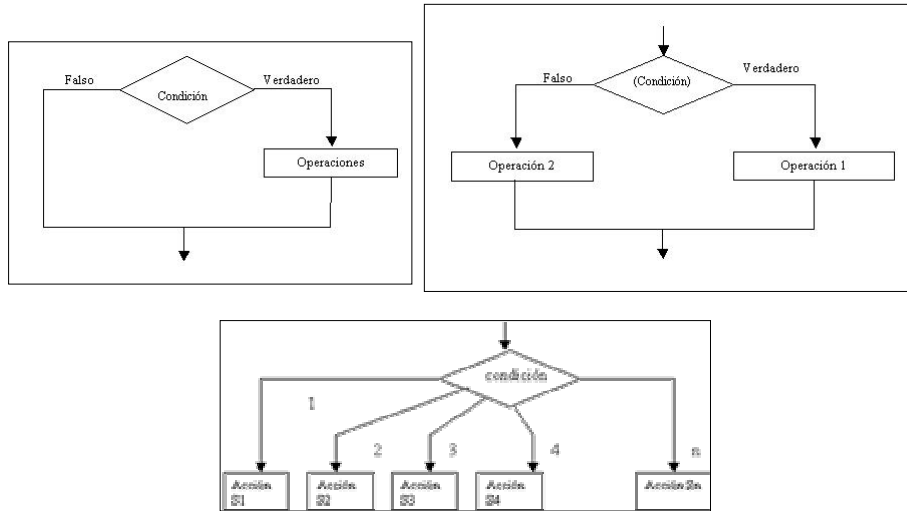
ELEMENTOS DE UN DIAGRAMA DE FLUJO

SIMBOLO	OPERACIÓN	DESCRIPCIÓN
	Flechas de flujo	Marcan la dirección de los datos
	Inicio/Fin	Indica el comienzo y el termino del diagrama
	Entrada y salida de datos	Sirve para solicitar entrada de datos
	Toma de decisión	Evalúa alguna condición y elige alguno de dos posibles caminos
	Conector dentro de la página	Continuación del flujo del diagrama sigue en otra parte de la hoja
	Conector fuera de la página	Continuación del flujo del diagrama sigue del lado derecho de la hoja.

ESTRUCTURA SECUENCIAL



ESTRUCTURA CONDICIONAL



ESTRUCTURA REPETITIVA

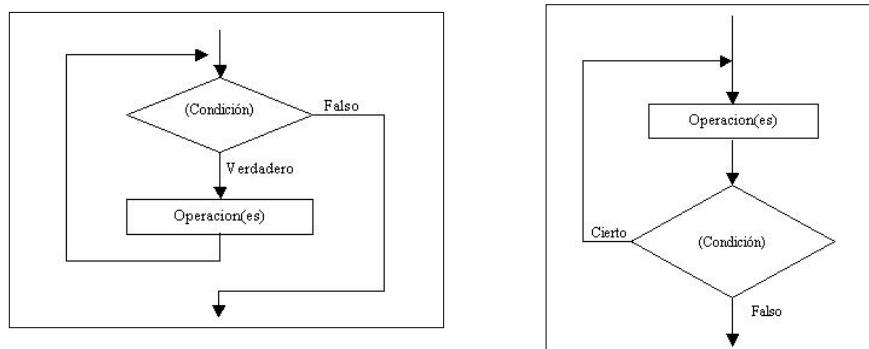
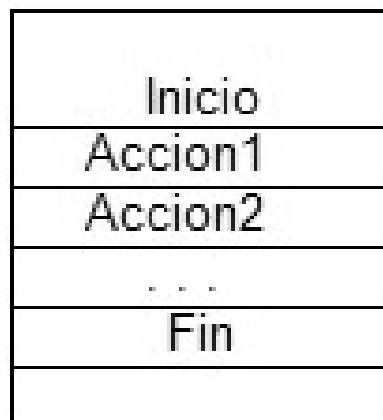


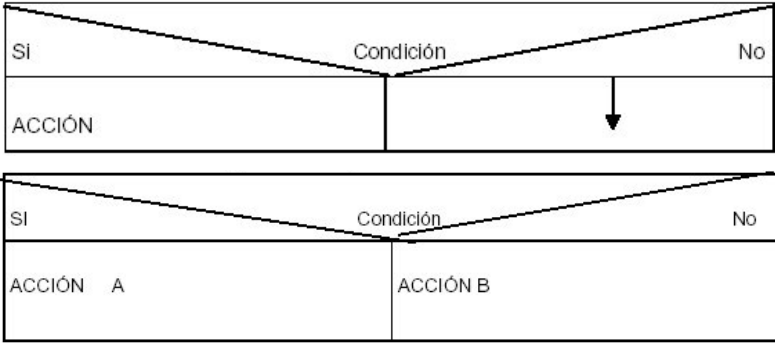
DIAGRAMA DE CAJAS

- Para representar una secuencia se conectan varias cajas seguidas.
- Para representar una estructura condicional (simple o doble) se representa una caja para la parte del SI y otra para la parte del NO, y encima se indica la condición a evaluar.
- Para representar una estructura condicional múltiple en la parte superior se indica el caso de condición y en la parte inferior tantas columnas como valores se vayan a comprobar en ella, indicándose en ellas la parte a realizar en cada caso.
- Para representar una estructura repetitiva, el proceso se encierra en una caja que se encuentra dentro de la que indica la condición del bucle.

ESTRUCTURA SECUENCIAL



ESTRUCTURA CONDICIONAL



ESTRUCTURA REPETITIVA

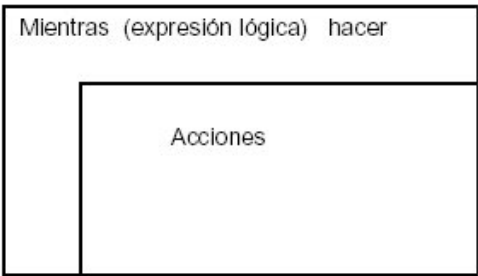


TABLA DE DECISIÓN

- Permiten representar en forma de tabla las condiciones y las acciones asociadas que se llevan a cabo en un algoritmo.
- Se dividen en cuatro cuadrantes:
 - Superior izquierdo: lista de todas las condiciones posibles.
 - Inferior izquierdo: lista de todas las acciones posibles.
 - Superior derecho: entrada de las condiciones.
 - Inferior derecho: entrada de las acciones.
- Cada columna de la parte derecha constituirá una regla de decisión.

Condiciones	Reglas							
Condición 1	S	S	S	S	N	N	N	N
Condición 2	S	S	N	N	S	S	N	N
Condición 3	S	N	S	N	S	N	S	N
Acción 1	X	X						
Acción 2				X		X		X
Acción 3			X				X	
Acción 4					X			

CONDICIONES	1	2	3	4
¿Paga contado?	S	S	N	N
¿Compra > \$ 50000?	S	N	S	N
ACCIONES				
Calcular descuento 5% s/importe compra	X	X		
Calcular bonificación 7% s/importe compra	X		X	
Calcular importe neto de la factura.	X	X	X	X

PASOS PARA SU CONSTRUCCIÓN

- Hacer una lista de todas las acciones y todas las condiciones.
- Asociar conjuntos específicos de condiciones con acciones específicas, eliminando combinaciones imposibles de condiciones.
- Determinar las reglas indicando qué acción o acciones ocurren para un conjunto de condiciones.

PSEUDOCÓDIGO

- Utiliza texto descriptivo para realizar el diseño de un algoritmo.
 - Se parece a un lenguaje de programación ya que utiliza frases en lenguaje natural con estructuras sintácticas que incluyen palabras clave.
 - Permite construir las estructuras básicas de la programación estructurada y modular.
- No existe un estándar definido.
- No puede ser compilado.

SECUENCIAL	Instrucción 1 Instrucción 2 Instrucción n
CONDICIONAL	SI <condición> ENTONCES <Instrucciones> SI NO <Instrucciones> FIN SI
CONDICIONAL MÚLTIPLE	SEGÚN <variable> HACER CASO valor1: <Instrucciones> CASO valor2: <Instrucciones> EN OTRO CASO: <Instrucciones> FIN SEGÚN
REPETIR-HASTA	REPETIR <Instrucciones> HASTA QUE <Condición>
HACER-MIENTRAS	MIENTRAS <Condición> HACER <Instrucciones> FIN MIENTRAS

```

INICIO

    Abrir fichero

    Leer registro del fichero

    MIENTRAS no sea fin de fichero HACER

        Procesar registro leído

        Leer registro del fichero

    FIN MIENTRAS

    Cerrar fichero

FIN

```


DISEÑO ORIENTADO A OBJETOS

- Capas:
 - Subsistema: Se centra en el diseño de las partes que implementan las funciones principales del sistema.
 - Clases y objetos: Especifica la arquitectura de objetos global y la jerarquía de clases requerida para implementar el sistema.
 - Mensajes: Indica cómo se realiza la colaboración entre los objetos.
 - Responsabilidades: Identifica las operaciones y atributos que caracterizan cada clase.
- UML: *Unified Modeling Language*.
 - Lenguaje de modelado basado en diagramas.
 - Análisis y diseño.

3.- FASES DEL DESARROLLO DE UNA APLICACIÓN

3.3.- CODIFICACIÓN

- El programador recibe las especificaciones del diseño y las transforma en un conjunto de instrucciones escritas en un lenguaje de programación, almacenadas dentro de un programa.
 - A este conjunto de instrucciones se le llama código fuente.
 - El programador debe conocer la sintaxis del lenguaje de programación utilizado.
- Deben existir unas normas de codificación y estilo, claras y homogéneas, en el equipo de desarrollo.
 - Estas normas facilitan las tareas de corrección y mantenimiento de los programas, sobre todo cuando se realizan por personas que no los han desarrollado.

3.- FASES DEL DESARROLLO DE UNA APLICACIÓN

3.3.- CODIFICACIÓN

- Una vez generado el código fuente es necesario traducirlo a un lenguaje que sea entendido por la máquina. Para ello se utilizan traductores de programas: compiladores e intérpretes.
 - El compilador genera un resultado de la traducción: el código objeto.
 - El código objeto se enlaza con las librerías y se genera un código ejecutable.
- Una vez obtenido el código ejecutable es necesario probar el programa con el fin de comprobar que cumple con las especificaciones del diseño.
- A la vez que se va desarrollando código se deben ir escribiendo los manuales técnicos y de referencia necesarios, así como la parte correspondiente del manual de usuario.
 - Esta documentación es esencial para las etapas de pruebas y mantenimiento, así como para la entrega final del producto.

NORMAS Y CONVENCIONES

- Nombres de ficheros. Extensiones.
- Organización de ficheros.
- Indentación.
- Comentarios.
- Declaraciones.
- Sentencias. Bloques. Estructuras de control.
- Separaciones.
- Identificadores.

3.- FASES DEL DESARROLLO DE UNA APLICACIÓN

3.4.- PRUEBAS

- En esta etapa ya se dispone del software y se trata de encontrar errores, no solo de codificación sino también relativos a la especificación o el diseño.
- Durante la prueba del software se realizarán tareas de:
 - Verificación: se comprueba si se está construyendo el producto correctamente, es decir, si el software implementa correctamente una función específica.
 - Validación: se comprueba si el producto es correcto, es decir, si el software construido se ajusta a los requisitos del cliente.
- Objetivo de esta etapa: planificar y diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores.

RECOMENDACIONES (I)

- Cada prueba debe definir los resultados de salida esperados.
- Un programador o una organización debe evitar probar sus propios programas.
- Es necesario revisar los resultados de cada prueba en profundidad.
- Las pruebas deben incluir datos de entrada válidos y esperados, así como no válidos e inesperados.

RECOMENDACIONES (II)

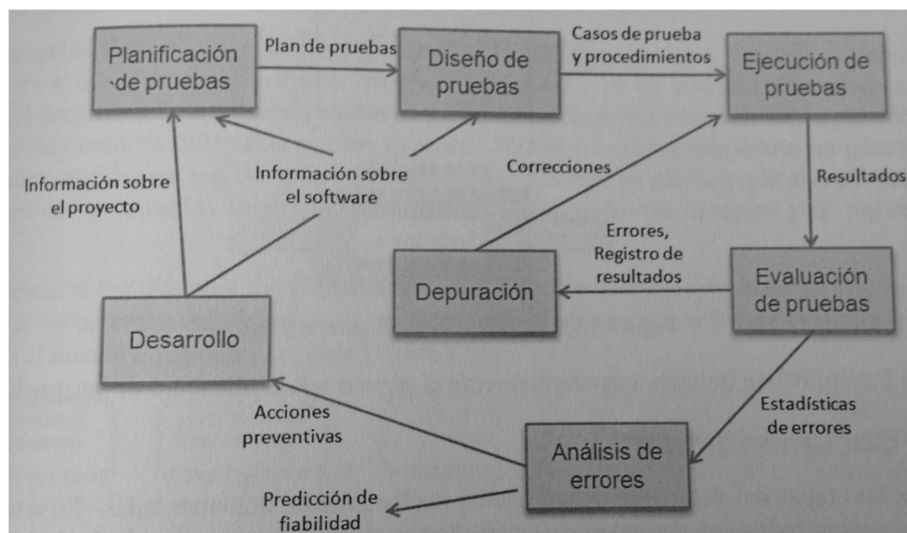
- Centrar las pruebas en dos objetivos:
 - Comprobar si el software no hace lo que debe hacer.
 - Comprobar si el software hace lo que no debe hacer.
- Evitar hacer pruebas que no estén documentadas ni diseñadas con cuidado.
- No planear pruebas asumiendo que no se encuentran errores.
- La probabilidad de encontrar errores en una parte del software es proporcional al número de errores ya encontrados.
- Las pruebas son una tarea creativa.

FLUJO DEL PROCESO (I)

1. Generar un plan de pruebas, partiendo de la documentación del proyecto y de la documentación sobre el software a probar.
2. Diseñar las pruebas (técnicas).
3. Generar los casos de prueba.
4. Definir los procedimientos de la prueba (cómo, quienes, ...).
5. Ejecutar las pruebas.

FLUJO DEL PROCESO (II)

6. Evaluar las pruebas, identificando los posibles errores al comparar los resultados obtenidos.
 - Informe del resultado de ejecución de las pruebas.
7. Depurar. Tratar de localizar y corregir los errores. Volver a probar el software para ver si se ha resuelto el problema.
8. Analizar los errores. Para predecir la fiabilidad del software y mejorar los procesos de desarrollo.



TÉCNICAS BÁSICAS

- Pruebas de caja blanca.
 - Se centran en validar la estructura interna del programa.
 - Necesita conocer los detalles procedimentales del código.
- Pruebas de caja negra.
 - Se centran en validar los requisitos funcionales sin fijarse en el funcionamiento interno del programa.
 - Necesita saber la funcionalidad que el código debe proporcionar.

3.- FASES DEL DESARROLLO DE UNA APLICACIÓN

3.5.- DOCUMENTACIÓN

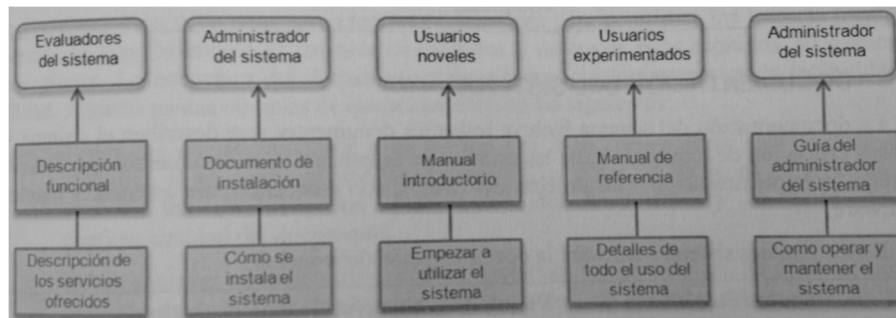
- Todas las etapas del desarrollo deben quedar documentadas.
- Los documentos relacionados con un proyecto de software:
 - Deben actuar como un medio de comunicación entre los miembros del equipo de desarrollo.
 - Deben ser un repositorio de información del sistema para ser utilizado por el personal de mantenimiento.
 - Deben proporcionar información para ayudar a planificar la gestión del presupuesto y programar el proceso del desarrollo del software.
 - Deben indicar a los usuarios cómo utilizar y administrar el sistema.

CLASES DE DOCUMENTACIÓN

- Documentación del proceso.
 - Estos documentos registran el proceso de desarrollo y mantenimiento.
 - Se indican planes, estimaciones y horarios que se utilizan para predecir y controlar el proceso de software.
 - Informan sobre cómo usar los recursos durante el proceso de desarrollo.
 - Informan sobre normas de cómo se ha de implementar el proceso.
- Documentación del producto.
 - Define dos tipos de documentación:
 - La documentación del sistema que describe el producto desde un punto de vista técnico, orientado al desarrollo y mantenimiento del mismo.
 - La documentación del usuario que ofrece una descripción del producto orientada a los usuarios que utilizarán el sistema.

DOCUMENTACIÓN DEL USUARIO

- Usuarios finales.
 - Utilizan el software para realizar alguna tarea.
 - Quieren saber cómo el software les ayuda a realizar su tarea.
 - No están interesados en el equipo informático o en los detalles del programa.
- Administradores del sistema.
 - Responsables de la gestión de los programas informáticos utilizados por los usuarios finales.
 - Pueden actuar como gestores de red o como técnicos para usuarios finales.
 - Sirven de enlace entre los usuarios finales y el proveedor de software.



DOCUMENTOS

- Descripción funcional del sistema.
 - Introducción a las funciones principales del sistema.
- Documento de instalación del sistema.
- Manual introductorio.
 - Explicaciones sencillas de cómo empezar a utilizarlo.
- Manual de referencia del sistema.
 - Descripción detallada de instalación, uso, errores, ...
- Guía del administrador del sistema.
- Tarjeta de referencia rápida.

DOCUMENTACIÓN DEL SISTEMA

- Fundamentos del sistema.
 - Objetivos de todo el sistema.
- Análisis y especificación de requisitos.
- Diseño.
 - Arquitectura. Aplicación de los requisitos. Descomposición del sistema.
- Implementación.
- Plan de pruebas del sistema.
- Plan de pruebas de aceptación.
 - Pruebas que el sistema debe pasar antes de que los usuarios las acepten.
- Diccionario de datos.

3.- FASES DEL DESARROLLO DE UNA APLICACIÓN

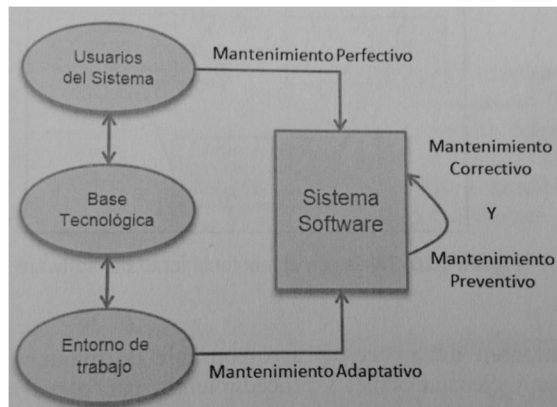
3.6.- EXPLOTACIÓN

- Se define la estrategia para la implementación del proceso de instalación y explotación.
- Se realizan las pruebas de operación.
- Se utiliza el sistema en un entorno previsto.
- Se proporciona soporte al usuario.

3.- FASES DEL DESARROLLO DE UNA APLICACIÓN

3.7.- MANTENIMIENTO

- Modificación de un producto de software después de la entrega para corregir los fallos, para mejorar el rendimiento y otros atributos, o para adaptar el producto a un entorno modificado.



TIPOS DE MANTENIMIENTO

- **Adaptativo.**
 - Objetivo: modificación del producto para adaptarlo a cambios que se produzcan (en el hardware o en el software) en el entorno en el que se ejecuta.
- **Correctivo.**
 - Objetivo: corregir errores o defectos encontrados por el cliente y que no han sido detectados en la etapa de pruebas.
- **Perfectivo.**
 - Objetivo: añadir funcionalidades y mejoras a demanda del cliente.
- **Preventivo.**
 - Objetivo: hacer cambios en los programas con el fin de que se puedan corregir, adaptar y mejorar más fácilmente (reingeniería del software).

TAREAS

- Implementación del proceso.
- Análisis de problemas y modificaciones.
- Implementación de las modificaciones.
- Revisión/aceptación del mantenimiento.
- Migración.
- Retirada del software.

