TEMA

44

Análisis y diseño de aplicaciones informáticas

M.ª Luisa Garzón Villar

CUERPO DE PROFESORES TÉCNICOS DE FORMACIÓN PROFESIONAL

ÍNDICE SISTEMÁTICO

1. INTRODUCCIÓN

Sin duda una de las tareas más complejas que puede abordar un profesional de la informática es el desarrollo de un sistema software, debido a la multitud de tareas que hay que llevar a cabo, desde el estudio previo de los requisitos, hasta su definitiva implementación y puesta en marcha.

La ingeniería del software es el marco de referencia para el estudio de las diferentes alternativas con las que contamos para el desarrollo de productos software de calidad, estudiando las herramientas, metodologías y procedimientos empleados en proyectos de desarrollo de sistemas software.

2. CICLO DE VIDA DEL SOFTWARE

La norma IEEE 1074 define el ciclo de vida software como *una aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del software*, mientras que 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.*

A veces también se habla de "ciclo de desarrollo", pero no hay que confundir estos términos; mientras que el "ciclo de vida" abarca toda la vida del sistema, comenzando con su concepción y terminando cuando ya no se utiliza, el "ciclo de desarrollo" es un subconjunto de éste que empieza con el análisis y termina con la entrega del producto finalizado al usuario.

La norma ISO 12207-1 describe las actividades que se pueden realizar durante el ciclo de vida del software y las agrupa en cinco procesos principales, ocho procesos de soporte y cuatro procesos generales (de la organización), así como un proceso que permite adaptar el ciclo de vida a cada caso concreto.

2.1. Procesos en el ciclo de vida del software

Procesos principales	Proceso de adquisición	Actividades y tareas del comprador.
	Proceso de suministro	Actividades y tareas del suministrador.
	Proceso de desarrollo	Análisis de requisitos, diseño, codificación, integración, pruebas e instalación y aceptación.
	Proceso de explotación	Explotación del software y soporte a los usuarios.
	Proceso de mantenimiento	Modificación del software existente cuando necesita modificaciones, ya sea en el código, o en la documentación asociada, debido a un error, una deficiencia, un problema o la necesidad de mejora o adaptación.

.../...

.../...

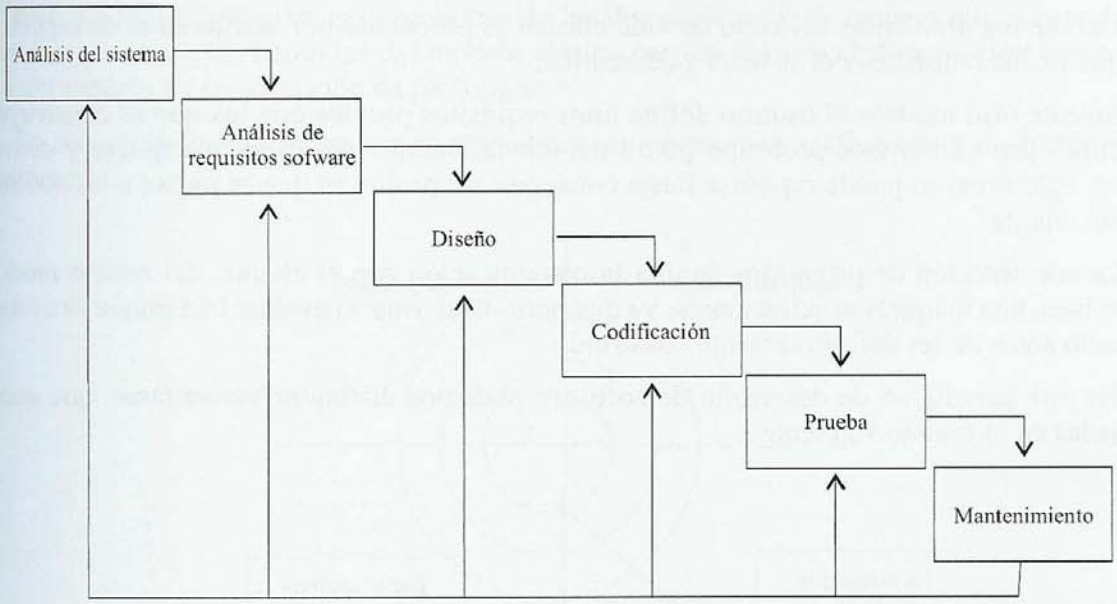
Procesos de soporte	Proceso de documentación	Registra la información producida por un proceso o actividad del ciclo de vida.
	Proceso de gestión de la configuración	Controla las modificaciones y las versiones de los elementos que componen el software.
	Proceso de aseguramiento de la calidad	Controla que los procesos y los productos software del ciclo de vida cumplen con los requisitos especificados y se ajustan a los planes establecidos.
	Proceso de verificación	Determina 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 o condiciones impuestos sobre ellos en las fases previas.
	Proceso de validación	Determina si el sistema o software final cumple con los requisitos previstos para su uso.
	Proceso de revisión conjunta	Evalúa el estado del software y sus productos en una actividad del ciclo de vida o una fase de un proyecto.
	Proceso de auditoría	Determina, en los *hitos* predeterminados, si se han cumplido los requisitos, los planes y el contrato.
	Proceso de resolución de problemas	Analiza y elimina los problemas descubiertos durante el desarrollo, la explotación o el mantenimiento.
Procesos generales	Proceso de gestión	Planificación, seguimiento, control, revisión y evaluación de las tareas genéricas de cualquier organización.
	Proceso de infraestructura	Establece la infraestructura necesaria para cualquier proceso.
	Proceso de mejora	Controla y mejorar los procesos del ciclo de vida del software.
	Proceso de formación	Mantiene al personal formado.
Proceso de adaptación	Realiza la adaptación básica de la norma ISO 12207-1 a los proyectos de software.	

2.2. Modelos de ciclo de vida del software

2.2.1. El modelo en cascada

Es el ciclo de vida clásico y se deriva de otras ingenierías. Se denomina así porque para comenzar una fase del ciclo debemos completar la fase anterior.

Gráficamente se representa de la siguiente forma:



Las fases en que se divide este ciclo de vida puede presentar ligeras variaciones según el autor, pero en esencia serán similares a las siguientes:

- **Análisis del sistema**. Un sistema software forma parte de un sistema mayor con el cual se relaciona. En este punto analizamos el sistema en su conjunto para poder establecer objetivos y poder determinar cuáles de ellos deben ser abordables por el software.
- **Análisis de requisitos software**. En esta fase nos centramos en los requisitos del software, definiendo la funciones a realizar, los datos, el comportamiento y la interacción entre los elementos funcionales.
- **Diseño**. En esta otra fase tomamos los requisitos definidos en la etapa anterior y diseñamos los componentes del sistema software a construir, indicando las estructuras de datos a emplear, módulos a construir, procedimientos, algoritmos e interfaces que forman el sistema final.
- **Codificación**. Tomando la documentación generada en el paso anterior, debemos implementar el sistema en una máquina concreta, codificando los procedimientos definidos en un lenguaje de programación, construyendo las interfaces y las estructuras de datos necesarias.
- **Prueba**. Tras la codificación debemos someter a prueba el sistema para cercionarnos de su buen funcionamiento y del cumplimiento de los requisitos fijados al inicio del proyecto.
- **Mantenimiento**. La producción de software no finaliza con la entrega del mismo, ya que debe ser revisado para ajustarlo a nuevos requisitos, corrección de errores detectados tras la entrega o por la aparición de requisitos no contemplados en el proyectos inicial, y que hace necesaria la revisión del producto.

Este modelo presenta dificultades en su aplicación, ya que en la práctica se producen realimentaciones, de modo que al finalizar una fase se puede detectar la necesidad de introducir correcciones sobre etapas anteriores.

Al poner en práctica este modelo, los problemas que se presentan se deben en gran medida a una mala definición de los requisitos del sistema.

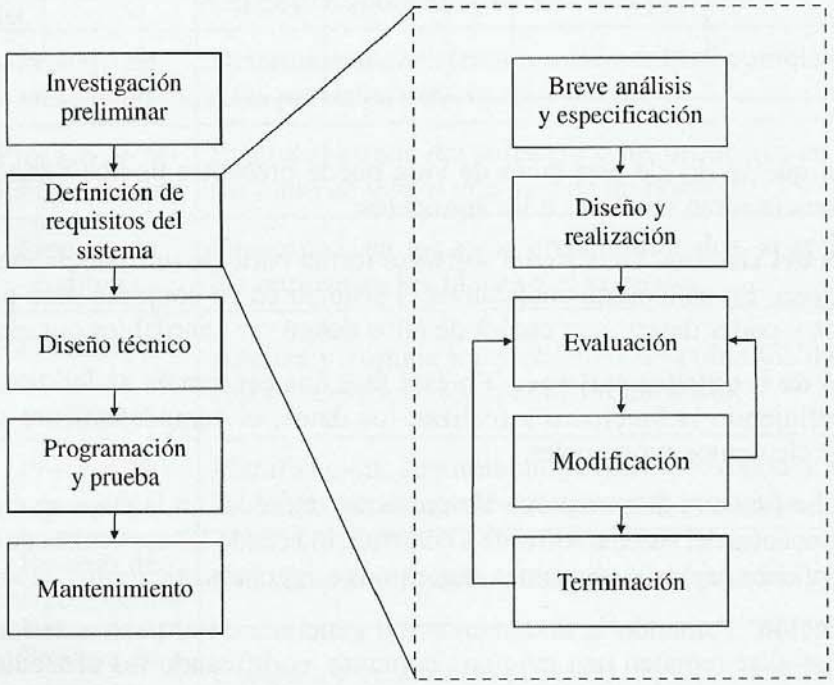
2.2.2. El modelo de construcción de prototipos

Uno de los problemas del ciclo de vida clásico es producido por la dificultad de especificar claramente los requisitos del sistema a desarrollar.

En este otro modelo el usuario define unos requisitos previos con los que se construye un prototipo. Revisando este prototipo podrá decidir realmente qué es lo que quiere y cómo lo quiere. Este proceso puede repetirse hasta conseguir un prototipo que se ajuste a las necesidades del cliente.

La construcción de prototipos facilita la comunicación con el cliente, del mismo modo en que lo hace una maqueta arquitectónica, ya que permite al usuario evaluar con mayor facilidad el producto antes de ser definitivamente construido.

En este paradigma de desarrollo de software podemos distinguir varias fases, que quedan reflejadas en el gráfico siguiente:



- **Breve análisis y especificación**: se efectúa un rápido análisis del sistema que sirva de base para la construcción del prototipo.
- **Diseño y realización**: construcción del prototipo.
- **Evaluación**: el cliente evalúa el prototipo a fin de afinar los requisitos.
- **Modificación**: se modifica el prototipo para satisfacer los requisitos especificados por el cliente en el paso anterior.
- **Terminación**: se termina la definición de requisitos del sistema final.

Debe ponerse en conocimiento del cliente que el prototipo mostrado no es el sistema final, sino un modelo utilizado para facilitar la comunicación con él, y que servirá como base para la construcción del sistema final.

2.2.3. El modelo en espiral

Este modelo conjuga las características de los dos anteriores, de manera que consta de una serie de pasos o etapas como las del modelo clásico pero existe una realimentación como en el caso del modelo de construcción de prototipos.

En este modelo se definen cuatro etapas que se representan gráficamente mediante cuatro cuadrantes.

- **Planificación**: se determinan los objetivos, requisitos y restricciones del proyecto.
- **Análisis de riesgo**: se analizan alternativas, se identifican riesgos y se resuelven.
- **Ingeniería**: desarrollo del producto.
- **Evaluación del cliente**: se valoran los resultados obtenidos.

Este modelo produce productos de manera incremental, añadiendo nuevos requisitos a cada vuelta de la espiral.

En la fase de análisis de riesgo se determinan las diferentes alternativas y los riesgos presentes en cada alternativa.

En la fase de ingeniería podemos emplear un enfoque clásico o de construcción de prototipos.

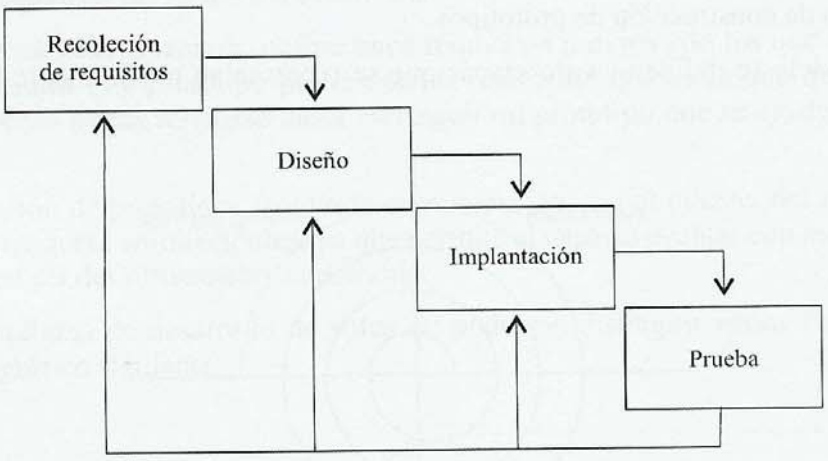
2.2.4. Técnicas de cuarta generación

Las herramientas de cuarta generación son herramientas que facilitan la especificación de características del software de una forma mas intuitiva, quedando como responsabilidad de la herramienta la conversión de dicha especificación a código fuente.

Los entornos de cuarta generación deben incluir herramientas para:

- Consulta de bases de datos mediante lenguajes no procedimentales.
- Generadores de formularios.
- Generadores de informes.
- Generadores de código.
- Manipulación de datos.
- Componentes software reutilizables.

Al igual que en otros enfoques, debemos partir de una recolección de requisitos siguiendo un esquema similar al mostrado en el siguiente gráfico.



Este tipo de herramientas facilita enormemente la producción, especialmente en proyectos pequeños.

3. METODOLOGÍA DE DESARROLLO

Una metodología de desarrollo es una recopilación de técnicas y procedimientos estructurados en fases para la producción de productos software de manera eficaz y englobando todo el ciclo de vida del mismo.

- Indica qué hacer, cómo, cuándo y quién debe hacerlo.
- Determina las etapas y controles a aplicar.

La metodología de desarrollo de software ha ido cambiando a lo largo de la historia; mientras que en un principio las prácticas eran totalmente artesanas y sin metodologías definidas (desarrollo convencional), poco a poco se llega la necesidad de definir unas reglas que eviten los problemas propios de no seguir unas normas concretas. Partiendo de la programación estructurada surgieron métodos de análisis y diseño que cubrieron el ciclo de vida completo (desarrollo estructurado). Actualmente el nuevo enfoque en ingeniería del software lo constituye el paradigma de la orientación a objetos (desarrollo orientado a objetos).

3.1. Principales metodologías de desarrollo

3.1.1. Metodología MERISE

Es una metodología de desarrollo de sistemas de información diseñada en Francia a finales de los años setenta con el fin de ser utilizada por la administración pública.

Merise se divide en varias etapas:

- Esquema director (*Schma directeur*).
- Estudio preliminar (*Etude préalable*).
- Análisis detallado (*Analyse dètaillèe*).
- Análisis técnico (*Analyse technique*).
- Realización (*Rèalisation*).
- Mantenimiento (*Maintenance*).

3.1.2. Metodología SSADM

Es una metodología de desarrollo diseñada en el Reino Unido, con el objetivo de ser utilizada por la Administración Pública, además de por las empresas privadas con las que ésta contrata su servicios, y que ha sido adoptado igualmente por otras empresas privadas con la intención de mejorar sus productos software. La primera versión vio la luz en el año 1981.

SSADM emplea tres técnicas fundamentales durante todo el ciclo de desarrollo, que son:

- *Logical Data Modelling* (LDM): se trata de un modelo que utiliza a su vez LDS (*Logical Data Structure*), para representar entidades de datos y las relaciones entre estas entidades, de un modo similar a los diagramas DED empleados en Métrica V2.
- *Data Flow Modelling* (DFM): utilizados para la representación de los tratamientos y de los flujos de datos entre éstos. Utiliza para ello un conjunto de DFD´s.
- *Entity Even Modelling* (EVM): consiste en un conjunto de diagramas ELH (*Entity Life Histories*), con los que se representan los eventos del sistema y la secuencia en la que éstos tienen lugar.

3.1.3. Metodología MÉTRICA

Es una metodología de desarrollo propuesta por la administración pública española para ser utilizada en el desarrollo de productos software de la propia administración y que ha sido adaptado por otras administraciones públicas (local y autonómica), así como por algunas empresas privadas.

Métrica V3, a diferencia de su versión anterior, se divide en procesos que a su vez se encuentran estructurados en actividades y estas últimas, en tareas. Para cada tarea se definen los participantes, productos, técnicas y prácticas.

Esta metodología de desarrollo ha sido ideada para abarcar una gran variedad de proyectos de diferente complejidad, por lo que su estructura deberá adaptarse a cada proyecto particular en función de su dimensión y características.

Métrica V3 se estructura en 3 procesos principales:

- Planificación de Sistemas de Información (PSI).
- Desarrollo de Sistemas de Información (DSI).
- Mantenimiento de Sistemas de Información (MSI).

El segundo proceso (DSI) es demasiado amplio, pro lo que se subdivide, a su vez, en los siguientes cinco procesos:

- Estudio de Viabilidad del Sistema (EVS).
- Análisis del Sistema de Información (ASI).
- Diseño del Sistema de Información (DSI).
- Construcción del Sistema de Información (CSI).
- Implantación y Aceptación del Sistema (IAS).

4. ANÁLISIS

Sea cual sea la metodología utilizada existen una serie de puntos que deben ser cubiertos y que indicamos a continuación:

1. **Identificación de las necesidades del cliente**, recopilando información sobre el sistema actual y sus deficiencias, así como los objetivos marcados por el cliente. En esta primera etapa el analista se reunirá con el cliente para recopilar toda la información precisa para poder especificar los requisitos del sistema a desarrollar teniendo en cuenta los siguientes puntos:
   - Lista de problemas y necesidades en el sistema de información.
   - Datos empleados y relación entre ellos.
   - Aplicaciones actuales.
   - Nuevas aplicaciones.
2. **Estudio de viabilidad**, para lo cual debemos definir claramente los recursos (tiempo, presupuesto, herramientas, personal necesario...) con los que vamos a disponer, ya que serán estos factores los que limiten la viabilidad del proyecto. Es evidente que cualquier proyecto es viable si los recursos son infinitos, pero éste es un caso ideal. Para el estudio de viabilidad nos centraremos en las siguientes áreas:
   - **Viabilidad económica**: se lleva a cabo mediante análisis coste-beneficios.
   - **Viabilidad técnica**: debemos determinar si existen las herramientas y técnicas necesarias para el desarrollo del sistema.
   - **Viabilidad legal**: determinará la existencia de problemas legales en el desarrollo del sistema, como puede ser la vulneración de algún derecho (LORTAD).
3. **Análisis económico**: en el que se detallarán los gastos y beneficios.
4. **Análisis técnico**: en el que se detallan las herramientas a emplear.
5. **Modelado del sistema**: debemos crear un modelo del sistema a construir que sea lo mas fidedigno posible, especificando los datos, transformaciones realizadas sobre estos datos y comportamiento dinámico del sistema. En esta fase se utilizan técnicas como los diagramas E/R, DFD, DFC, diccionarios de datos... Además se definen requisitos de seguridad y de control del sistema a desarrollar. En esta fase se pueden emplear herramientas CASE para modelar el sistema y estudiar su comportamiento.

4.1. Análisis de requisitos. Especificación de Requisitos del Software (ERS)

La definición de los requisitos que debe cumplir el software a construir es un trabajo que deben realizar conjuntamente los analistas y el cliente, ya que ni el cliente conoce el proceso de diseño, ni el analista conoce completamente el trabajo que deberá realizar el software.

Según el estándar IEEE 1074 se deben realizar tres actividades:

- **Definir los requisitos del software**: mediante técnicas de recogida de información (entrevistas, prototipado, etc.) se debe crear una especificación preliminar de los requisitos que debe cumplir el software.
- **Definir los requisitos de las interfaces del software con el resto del sistema y con el exterior**: también son requisitos a definir cómo se relacionará el sistema con otros elementos software, con los elementos hardware y con el usuario.
- **Integrar los requisitos en un documento de especificación y asignarles prioridades**: este documento será lo que denominamos ERS (Especificación de Requisitos Software) y deberá ser revisado y posteriormente aprobado por el cliente para poder seguir adelante con el proyecto.

4.2. Estudio de viabilidad

4.2.1. Selección de una alternativa

Para la selección de una alternativa se observarán las discrepancias entre ellas, debiéndose seleccionar la más viable, en base a los criterios siguientes:

- Estudio de las ventajas e inconvenientes de cada una, teniendo en cuenta la facilidad de uso, los recursos que se necesita emplear para su desarrollo y la estimación del tiempo que va a transcurrir hasta su implantación.
- Estimación de costes y tiempo, en función del coste del hardware, del coste del personal y del coste de implantación.
- Establecimiento de los beneficios para la empresa desarrolladora y para el cliente.

4.2.2. Análisis de la viabilidad técnica y económica

La implantación de un nuevo sistema requiere un estudio previo de las repercusiones positivas o negativas que se produzcan sobre el usuario y la empresa. Para ello se analizan las limitaciones que puedan aparecer durante su desarrollo, implantación y explotación. También hay que pensar que no siempre es necesario la modificación total o parcial del sistema, dependiendo de su aceptación y de su eficacia.

4.3. Los elementos del modelo de análisis

El análisis debe perseguir tres objetivos fundamentales: debe describir fielmente lo que quiere el cliente, debe servir de base a la construcción del software, y debe establecer los requisitos que éste debe cumplir, de forma que al finalizar su construcción se pueda comprobar si el producto final da respuesta o no a todos estos requisitos.

Para lograr este cometido el modelo de análisis abre tres frentes:

- **Modelado de datos**: utilizando la técnica del diagrama entidad/interrelación (DER), representa entidades y asociaciones utilizando las relaciones entre los objetos de datos.
- **Modelado de procesos**: utilizando la técnica del diagrama de flujo de datos (DFD), representa la transformación de los datos y la división en funciones y subfunciones de la aplicación.
- **Modelado del comportamiento**: utilizando la técnica del diagrama de transición de estados (DTE), representa el comportamiento del sistema ante sucesos externos.

Como nexo de unión a estos tres modelos se encuentra el diccionario de datos (DD), un almacén donde se definen todos los objetos que aparecen en los tres modelos.

4.3.1. Modelado de funciones

4.3.1.1. Diagrama de flujo de datos. DFD

Es una técnica definida por DeMarco a finales de los años setenta. Esta técnica es empleada para modelar el sistema tomando en consideración las funciones a realizar por el mismo y los flujos de información entre diferentes funciones. El DFD emplea diferentes niveles de abstracción para definir el sistema, utilizando para ello símbolos gráficos para definir los elementos básicos del sistema. Es importante resaltar que con esta técnica se pone el énfasis en "el qué" sin entrar en "el cómo".

Entre los elementos básicos de un DFD, tenemos:

- Procesos.
- Entidades externas.
- Almacenes de datos.
- Flujos de datos.

Un DFD debe cumplir las siguientes características:

- Gráfico.
- Legible.
- Compresible.
- Debidamente particionado.
- Bien documentado.
- No redundante.
- Establecer "qué" hacer, y no "cómo".

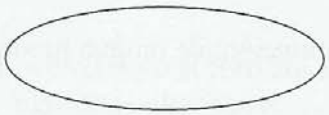
- **Entidades externas**. Una entidad externa representa un elemento externo al sistema que aporta o recibe información del mismo. Las entidades externas suelen aparecer en el diagrama de contexto, que representa el primer nivel de abstracción.

Hemos de tener presente lo siguiente:

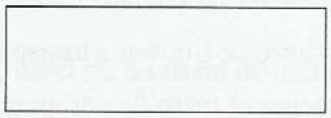
* Nos informan sobre los flujos de información hacia y desde el exterior.
* Los flujos entre entidades externas no nos interesan y, por tanto, no se representan.
* Pueden aparecer repetidos para evitar entrecruzamiento de líneas.

En cuanto a la representación gráfica existen dos alternativas: la primera se debe a Yourdon y DeMarco y la segunda es la empleada por Métrica V2 y SSADM.

Métrica V2, SSADM



Yourdon, DeMarco



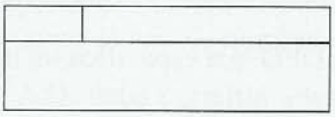
- **Procesos**. Representan funciones de transformación de datos, de manera que reciben flujos de datos de entrada y generan flujos de datos de salida obtenidos mediante transformación de los datos de entrada.

  Hemos de tener en cuenta:

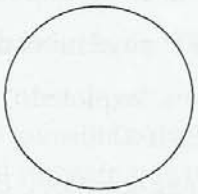
  * Un proceso no generará ni destruirá datos, sólo los transforma.
  * Entre una entidad externa y un almacén de datos es necesario un proceso.

  La representación gráfica es:

Métrica V2, SSADM



Yourdon, DeMarco



  Los procesos se identifican con un número (que depende del nivel de abstracción en el que nos encontremos) y de un nombre que debe ser significativo.

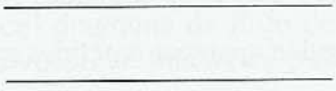
- **Almacén de datos**. Como indica su nombre, se trata de un depósito de información que puede ser utilizado por los procesos para almacenar y/o recuperar información.

  Al igual que los procesos los almacenes de datos son identificados mediante un nombre.

  Hemos de considerar:

  * Los almacenes de datos se identifican con un nombre representativo de los datos que almacena.
  * Se puede repetir en un mismo diagrama para contribuir a su legibilidad.
  * No se hacen referencias al entorno físico.

Su representación gráfica es:

Métrica V2, SSADM	Yourdon, DeMarco
	

- **Flujo de datos**. Sirven para conectar procesos, almacenes y entidades, representando la información que fluye entre ellos, así como el sentido de dicho flujo.

  Hemos de tener en cuenta:

  * Los flujos se limitan a transportar información; de ningún modo los crea o destruye.
  * Conectan el resto de componentes.
  * La flecha indica la dirección de los datos.

  Su representación gráfica es la mostrada a continuación:

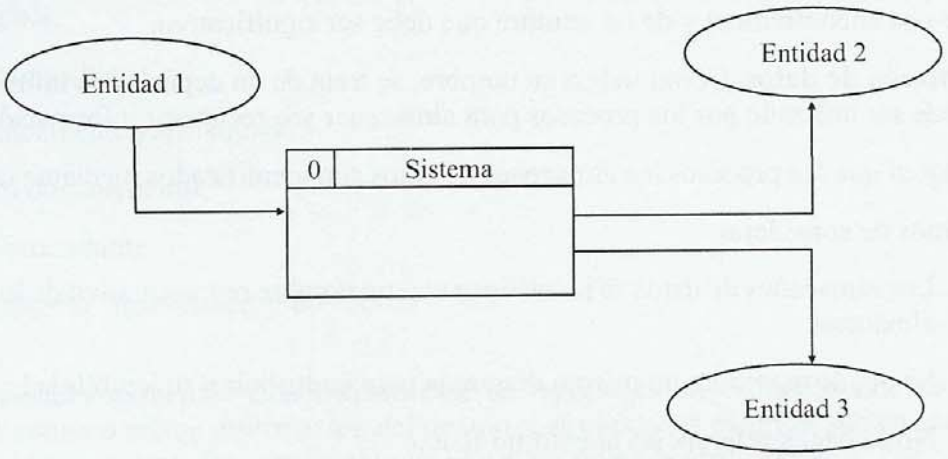
- **Descomposición por niveles**. Esta técnica se basa en la filosofía *top-down*, en el sentido de que se analiza y se representa el sistema usando varios niveles de abstracción, comenzando por el mayor nivel de abstracción posible y evolucionando de forma progresiva a niveles de mayor detalle.

  Se utilizan:

  * Un diagrama de contexto.
  * Varios DFD en niveles intermedios.
  * Varios DFD en el nivel de detalle.

  Cada proceso es "explotado" en un nuevo DFD que especifica un mayor nivel de detalle (menor abstracción).

  1. El nivel mas abstracto, denominado también diagrama de contexto, representa todo el sistema en un único proceso y las entidades externas con las que se relaciona, así como los flujos de información entran en el sistema desde dichas entidades externas:



2. A continuación descomponemos el proceso del diagrama de contexto en un DFD con un mayor nivel de detalle.
3. Repetimos el proceso anterior hasta llegar a un nivel de detalle suficiente.

4.3.1.2. Diccionario de datos

El diccionario de datos es un elemento muy importante en el análisis, ya que en él se toma nota de todos los elementos a los que hacemos referencia en los diagramas empleados para modelar el sistema que queremos construir. Este elemento nos servirá para tomar nota de los datos, objetos, entidades, almacenes y elementos de control a los que hacemos referencia en los DFD, DFC, DTE...

Existen variaciones en cuanto al formato empleado en el diccionario de datos, pero encontramos casi siempre los siguientes elementos:

- Nombre del elemento en cuestión.
- Alias o nombres alternativos con los que se conoce a dicho elemento.
- Dónde se usa dicho elemento, indicando los procesos.
- Cómo se usa dicho elemento en cada proceso.
- Descripción detallada del elemento.
- Información adicional, como pueden ser restricciones, limitaciones, condiciones o cualquier otra información pertinente.

El diccionario de datos es un elemento muy voluminoso y que se hace difícil de mantener; es por ello que haremos uso de él empleando alguna herramienta CASE, que será la encargada de la actualización del mismo a medida que vayamos modelando el sistema.

Una condición importante a cumplir es que no pueden existir duplicidades, de modo que dos elementos no puedan tomar el mismo nombre.

La herramienta CASE debe permitir, además, efectuar consultas al diccionario, de manera que en caso de modificaciones al proyecto podamos saber el impacto que conlleva dicho cambio y determinar qué elementos son afectados y en qué procesos aparecen dichos elementos.

La descripción del contenido de los elementos en el diccionario de datos puede seguir una sintaxis similar a la siguiente:

ELEMENTO	SIGNIFICADO
=	Designa el contenido.
+	Un elemento compuesto puede estar formado por la concatenación de varios simples.
[|]	Un elemento compuesto por varias alternativas.
{ }n	Un elemento compuesto por varios simples repetidos "n" veces.
( )	Elemento opcional.

4.3.1.3. Especificación de procesos

Una vez completada la descomposición de procesos representados en los DFD y alcanzado el nivel de máximo detalle, tenemos un esquema de los procesos que forman parte de la aplicación analizada. El siguiente paso consiste en describir las transformaciones efectuadas por estos procesos, ya que reciben flujos de información y producen flujos de salida de información efectuando transformaciones que no se han especificado en el DFD.

Debemos describir el modo de acceso a los datos del sistema (que se recogen en el diccionario de datos), frecuencias de ejecución de los procesos e información sobre la ejecución especificando los algoritmos empleados.

Para describir los procesos contamos con varias técnicas que podemos combinar:

- Lenguaje natural.
- Lenguaje estructurado.
- Tablas de decisión.
- Árboles de decisión.

4.3.2. Modelado de datos

El objetivo del modelado de datos es la representación gráfica mediante unas reglas y convenciones de los datos del mundo real que queremos almacenar en una base de datos.

- **Elementos del modelo E/R.**
  - * **Entidad**: cualquier objeto del que guardamos información.
  - * **Tipo de entidad**: el conjunto de todas las entidades que cumplen una determinada condición. Las entidades serían, por lo tanto, ocurrencias de un tipo de entidad.
  - * **Asociación o interrelación**: relaciones o correspondencias entre entidades.
  - * **Tipo de asociación o interrelación**: conjunto de todas las asociaciones que cumplen una cierta definición. En este punto muchos autores indican que a partir de este momento, y por motivos de claridad, utilizarán el término "entidad" para hacer referencia tanto a las entidades como a los tipos de entidad, y el término "asociación" o "interrelación" para hacer referencia tanto a interrelaciones como a tipos de interrelación. Nosotros haremos lo mismo y dejaremos de utilizar los citados términos a partir de este momento.
  - * **Tipo de la asociación**: representa el grado de participación de cada entidad en la interrelación. Puede ser:
    - **(1:1)**: cada ocurrencia de la entidad A está asociada con 0 ó 1 de la entidad B y viceversa.
    - **(1:M)**: cada ocurrencia de la entidad A está asociada con 0, 1 o varias ocurrencias de la entidad B, y cada ocurrencia de B con 0 o 1 de A, o viceversa.
    - **(M:N)**: cada ocurrencia de la entidad A está asociada con 0, 1 o varias ocurrencias de la entidad B y viceversa.
  - * **Atributo de entidad o de interrelación**: información acerca de una entidad o de una interrelación, que sirve para identificarla o describirla.

* **Atributos identificadores o claves**: los que tienen diferentes valores para cada ocurrencia de entidad o interrelación. Pueden estar compuestos por varios atributos.
* **Atributo identificador principal o clave principal**: el más importante de los anteriores.
* **Entidades y asociaciones débiles**: son entidades débiles las que basan la existencia de sus ocurrencias en la existencia de ocurrencias de otra entidad "padre" de las que dependen. Son asociaciones débiles las que unen este tipo de entidades débiles con sus entidades "padre".
* **Dominio**: el conjunto de valores, agrupados bajo un nombre, que puede tomar un atributo. El dominio de un atributo podrá definirse por intención: indicando el tipo de datos que podrá contener, o por extensión: indicando todos los valores posibles.
* **Grado de una asociación**: indica el número de entidades que participan en una asociación. Podremos encontrarnos con asociaciones de grado dos o binarias, de grado tres o ternarias, etc. Existe un caso especial de asociaciones binarias, las asociaciones reflexivas, donde las dos entidades que participan son la misma.
* **Cardinalidad**: número máximo y mínimo de ocurrencias de una entidad que pueden relacionarse a través de una asociación con otra entidad. Se representa de la siguiente forma:

(cardinalidad-mínima, cardinalidad-máxima)

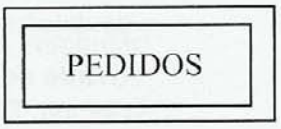
Y las cardinalidades posibles son: (0,1), (1,1), (0,M), (1,M), o (M,N).

- **Construcción de un esquema E/R**. Las etapas que habrá que realizar para conseguir el modelo completo serán las siguientes:
  a) Identificación de entidades, sus atributos y la clave principal.
  b) Identificación de las interrelaciones entre entidades, indicando el tipo de interrelación y la cardinalidad, así como los atributos que pudieran tener.
  c) Presentación del modelo entidad-interrelación.
- **Cómo descubrir entidades y asociaciones**. El diseño de una base de datos es una labor creativa, y aunque no existen reglas que indiquen qué elemento va a ser entidad y cual otro interrelación, sí se pueden dar algunas recomendaciones que se pueden seguir a la hora de descubrir entidades e interrelaciones.
  * Los sustantivos utilizados como sujetos o complementos directos en una frase suelen ser entidades, aunque también pueden representar atributos. Por ejemplo, en la frase: "Los clientes que deseen comprar los productos que ofrece la empresa ..." pueden descubrirse dos entidades, los clientes (o socios) y los productos (o artículos). O en esta otra: "... almacenes donde guardan los libros que distribuyen" podemos distinguir también otras dos almacenes y libros (o artículos).
  * Los nombres propios suelen ser ocurrencias de entidades.
  * Los verbos, suelen indicar interrelaciones. Por ejemplo en la frase: "... pudiendo un socio avalar a varios otros socios". Indica una interrelación, en este caso reflexiva, de la entidad socios.
  * Las preposiciones y las frases preposicionales entre dos nombres suelen indicar interrelación o atributo de una entidad. Por ejemplo, la frase: "..el número de volúmenes de que consta", indica que la entidad tendrá un atributo que almacene el número de volúmenes de que consta la colección.

– **Presentación del modelo.** Se mostrarán en el modelo todas las entidades y asociaciones descubiertas utilizando la siguiente notación:

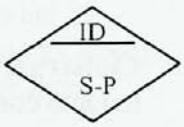
* **Entidades**: un rectángulo con el nombre en su interior.

* **Entidades débiles**: dos rectángulos concéntricos con el nombre en su interior.



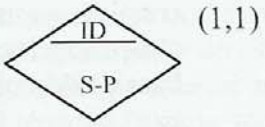
* **Interrelaciones**: un rombo con el nombre en su interior.

* **Asociaciones débiles**: un rombo indicando en su interior el tipo de debilidad ID, identificación ó EX existencia.

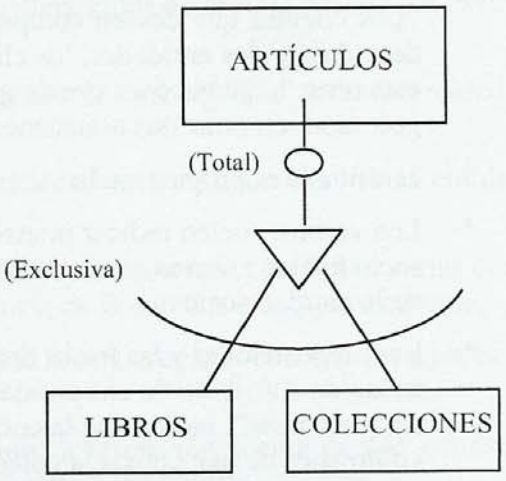


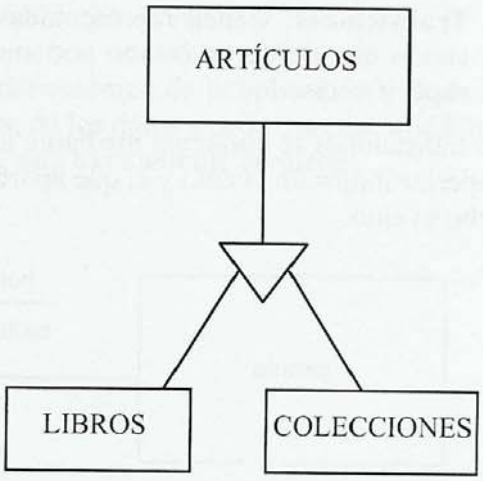
* **Tipo de asociación**: caracteres 1, M o N al lado de la asociación.

* **Cardinalidad con la que una entidad participa en una asociación**: caracteres entre paréntesis (1,1), (1,M), ...al lado de la entidad.



* **Especialización**: total y exclusiva.





* **Especialización**: parcial e inclusiva.

* **Atributos**: el nombre del atributo al lado del símbolo.

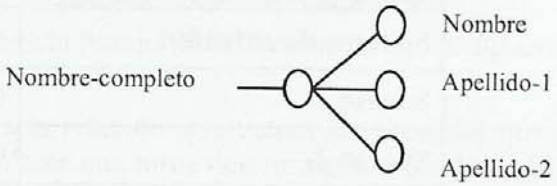
* **Atributos**: identificadores.

* **Atributos**: claves alternativas.

* **Atributos**: atributos multivaluados.

* **Atributos**: atributos opcionales.

* **Atributos**: atributos compuestos.



4.3.3. Modelado de comportamiento

A parte de los aspectos referidos a los datos y a las funciones, al especificar un sistema también es necesario dar una perspectiva de su comportamiento. Para describir estos aspectos respectivos al control se utilizan técnicas como los diagramas de transición de estados, las tablas de activación de procesos, o las redes de Petri. Para no extendernos demasiado describiremos sólo dos de estas técnicas.

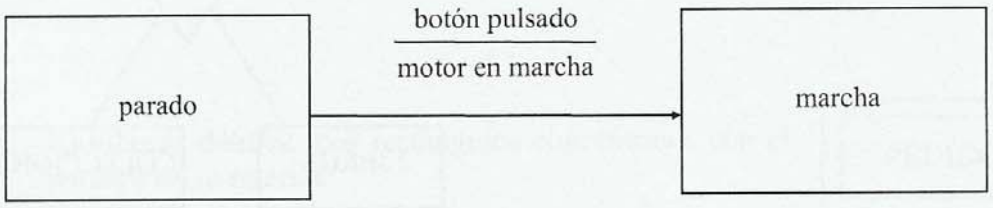
4.3.3.1. Diagrama de transición de estados

Representa el comportamiento de un sistema a lo largo del tiempo mostrando los cambios de estado que se producen a consecuencia de los eventos, así como las acciones a llevar a cabo.

- **Estados**: representan un modo observable de comportamiento. Por ejemplo, un ascensor puede encontrarse en uno de los siguientes estados: parado, marcha, averiado.

- **Transiciones**: vienen representadas por flechas que parten de un estado a otro, indicando que desde un estado podemos pasar a otro diferente si se dan las condiciones especificadas.

Las transiciones se etiquetan mediante una fracción, de modo que el texto que aparece en la parte superior indica un evento y el que aparece en la parte inferior indica una acción que responde a dicho evento.



4.3.3.2. Tabla de activación de procesos

La tabla de activación de procesos representa sucesos y procesos que son activados por estos sucesos.

En la tabla se representan las posibles combinaciones de sucesos de entrada, así como los sucesos de salida provocados por las combinaciones de los de entrada.

Al final de la tabla se especifican los procesos a activar, así como las condiciones bajo las que dichos sucesos son activados.

TABLA DE ACTIVACIÓN DE PROCESOS								
Sucesos de entrada								
Suceso 1	0	0	0	0	0	0	0	0
Suceso 2	0	0	1	1	0	0	1	1
Suceso 3	0	1	0	1	0	1	0	1
Salida								
Señal 1	0	0	0	0	0	1	1	1
Señal 2	1	0	0	0	0	0	0	0
Activación de procesos								
Proceso 1	0	0	1	1	0	1	0	1
Proceso 2	1	0	0	0	0	1	0	1
Proceso 3	0	1	1	0	0	0	1	0

5. DISEÑO

La finalidad de esta fase es lograr el diseño arquitectónico de la aplicación y de la base de datos. Para ello se realizará el diseño de las funciones, de los datos y de la interfaz, quedando toda esta información recogida en el **cuaderno de carga**, que básicamente contiene:

- Descripción de ficheros.
- Descripción de registros.
- Descripción de las estructuras de datos.
- Estructura de los módulos.
- Diseño de entrada de datos por pantalla.
- Diseño de salida de datos por pantalla.
- Diseño de salida por impresora.
- Descripción de los elementos utilizados en cada módulo.
- Definición de las variables.
- Diseño de algoritmos.

5.1. Diseño lógico de funciones

El objetivo fundamental de esta tarea es desarrollar la estructura del programa, o sea, a partir de los DFD´s, describir los módulos que componen la aplicación y las relaciones existentes entre ellos representando esta información en lo que denominamos "diagrama de estructura" o "Cuadros de Constantine".

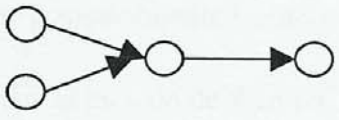
Para ello deberemos haber refinado los DFD´s de tal forma que de cada proceso primitivo podamos extraer un módulo.

Los módulos se caracterizan por su independencia funcional. Los criterios que se siguen para lograr esta independencia son dos:

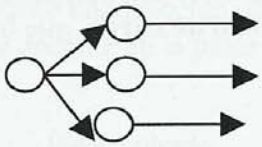
- **Cohesión**: la cohesión hace referencia a la relación que tienen los procesos que integran un módulo, de manera que para realizar una tarea determinada todos los procesos involucrados deben estar incluidos en un mismo módulo, mientras que los procesos que realicen otra tarea estén disponibles en otros módulos de forma independiente, logrando así una transacción mínima de datos entre los módulos que conforman un programa. En resumen, el grado de cohesión se establece en función de la relación funcional de los procesos que forman los módulos de un programa.
- **Acoplamiento**: hace referencia a la interconexión existente entre los distintos módulos. Los módulos deben ser independientes en el mayor grado posible, es decir, la cantidad de información que comparten debe ser mínima o nula, lo que facilita su diseño.

  Los flujos de datos que unen los procesos primitivos podrán ser de dos tipos: de transformación o de transición. Esta diferencia influirá a la hora de determinar la estructura jerárquica de los módulos.

  * **Flujo de transformación**: existirán unos datos de entrada procedente del exterior que deben ser transformados mediante el software existente en datos de salida. A los datos de salida se le denomina **flujo saliente**, mientras que los datos de entrada han seguido un camino llamado **flujo entrante**.



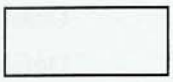
* **Flujo de transacciones**: en este caso los datos de entrada son evaluados y en función del resultado seguirá uno de los caminos de acción. El lugar donde los datos son evaluados para seguir un camino se denomina **centro de transacción**.



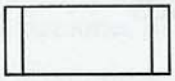
5.1.1. Diagrama de estructura o Cuadros de Constantine

Los elementos que intervienen en el diagrama de estructura, básicamente son:

- **Módulos**: son trozos de códigos que cumplen una determinada función, pudiendo ser llamados en cualquier momento, devolviéndonos un resultado. Un módulo puede ser un programa o un subprograma. El nombre que se le asigne debe indicar la función que realiza. Su representación es la siguiente:

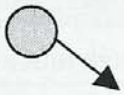


- **Módulos predefinidos**: se encuentran dentro de las bibliotecas de módulos o librerías.



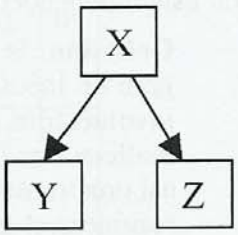
- **Comunicación de datos**:

- **Comunicación de control**:

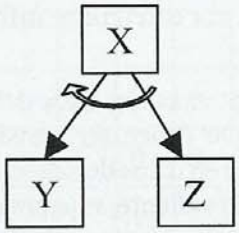


- **Tipos de estructuras**:

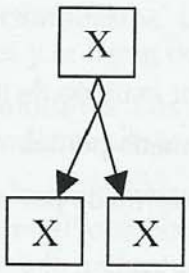
1. **Secuencia**: cuando un módulo llama a varios módulos y se van a ejecutar uno a continuación de otro siguiendo un orden preestablecido. La ejecución irá siempre de izquierda a derecha y de arriba a abajo.



2. **Iteración**: al igual que en caso anterior, se llamarán a varios módulos, pero con la diferencia de que éstos se ejecutarán más de una vez. Junto a las líneas se situarán el símbolo de "datos" o de "control", según el proceso o el resultado de la operación.



3. **Decisión**: se aplica cuando el módulo situado en la parte superior tiene que decidir qué modulo es el que se va a ejecutar.



Las fases a seguir para obtener un diseño modular basándonos en el DFD, son:

1. Revisión y afinado de los DFD a diversos niveles.
2. Comprobación del tipo de DFD: de transformación o de transacción.
3. Identificación del centro de transacción o transformación. Especificación de los flujos.
4. Conversión del diagrama de flujo de datos en la estructura modular del programa.
5. Refinado de la estructura modular obtenida empleando criterios de calidad de software.
6. Revisión de la funcionalidad del diagrama de estructuras obtenido.

5.1.2. Diseño detallado

Después de haber realizado la fase de diseño del sistema, comienza la fase de diseño detallado, donde se realizará el desarrollo de la lógica interna, o sea, los algoritmos que componen cada uno de los módulos.

La calidad de un algoritmo no depende sólo de la efectividad de su funcionamiento, sino de la claridad de su código. Es muy importante que sea fácilmente comprensible y que esté bien estructurado, ya que esta característica facilitará su mantenimiento, actualización y adaptación a nuevas situaciones. Un algoritmo bien hecho deberá cumplir como mínimo las siguientes condiciones:

- Deberá ser **fiable**. Los resultados deberán ser exactos y precisos.
- Deberá ser **eficiente**. Utilizará de forma óptima los recursos del ordenador, no deberá ocupar mucha memoria, a la vez que será rápido en su ejecución.
- Deberá ser **robusto**, tener prevista una respuesta clara sean cuales sean los datos de entrada introducidos y sean cualesquiera las circunstancias bajo las que se ejecuta.
- Deberá ser **transportable**, deberá estar diseñado de forma que se pueda poner en funcionamiento en cualquier ordenador independientemente del hardware de que disponga y del sistema operativo instalado.
- Deberá **ofrecer todas las facilidades posibles al usuario**, con un interfaz amigable, mensajes claros y concisos, y una buena documentación.

El diseño de algoritmos es una labor creativa, sin embargo existen técnicas de desarrollo, como pueden ser la programación modular o la estructurada, que facilitan esta labor. Sin embargo, para la descripción de algoritmos se utilizan sistemas normalizados, entre los que poder elegir en el momento de la trascripción de las acciones que lo componen y que posteriormente faciliten la codificación en un lenguaje de programación.

Como sistemas de representación entre los que podremos elegir, están: el método de Warnier, el método de Jackson, el método de Bertini, el método de Tabourier, o el método de Chapin.

5.2. Diseño lógico de datos

5.2.1. El modelo relacional

El modelo relacional está formado por la unión de tres componentes:

a) **Componente de estructura**: formado por relaciones, filas columnas, claves, etc.

b) **Componente de manipulación**: formado por una colección de operadores para acceder a los datos.

c) **Componente de integridad**: formado por una colección de reglas de integridad general.

Repasemos ahora algunos conceptos básicos de este modelo que nos servirán para convertir un modelo conceptual en un modelo relacional:

- **Relación o Tabla**: matriz rectangular donde se representan como filas (tuplas) las ocurrencias y como columnas los atributos. La intersección fila-columna representa el valor del atributo para la ocurrencia concreta. El número de columnas de la tabla se denomina grado de la relación.

  Las relaciones deben cumplir una serie de propiedades:

  * Cada inserción fila-columna debe contener un dato atómico (indivisible y univaluado).
  * Todos los datos consignados en una misma columna son del mismo tipo (definidos sobre el mismo dominio).
  * Cada columna tiene asignado un nombre (el del atributo que representa).
  * El contenido de cada fila debe ser distinto (no puede haber dos filas iguales).
  * Ni el orden de las filas ni el de las columnas es significativo.

- **Clave candidata**: atributo o grupo de atributos capaz de identificar cada fila de la relación. Una clave candidata no debe ser redundante, es decir, si está formada por un grupo de atributos, para que no sea redundante, si se quitara uno de los atributos del grupo, debería dejar de ser clave candidata.
- **Clave principal**: la clave candidata más importante.
- **Clave extranjera**: atributo de una relación que es clave principal de otra relación.
- **Reglas de integridad**: en el modelo relacional existen principalmente dos reglas de integridad:
  * **Integridad de la clave**: los atributos clave o que formen parte de la clave no pueden contener valores nulos (desconocidos).
  * **Integridad de referencia**: una clave extranjera debe contener un valor o bien igual a un valor en la tabla que contiene este atributo como clave principal, o bien un valor nulo.

5.2.1.1. Preparación de los modelos

Vamos a describir brevemente unas reglas preliminares que nos permitirán preparar los modelos conceptuales basados en el modelo E/R, para posteriormente traducirlos a modelos lógicos basados en el modelo relacional. El seguimiento de estas reglas facilitará y garantizará la fiabilidad del proceso de transformación.

El primer paso será transformar los atributos compuestos y los atributos múltiples, que no podrán ser representados en el modelo relacional.

- **Transformación de atributos compuestos**. Los atributos compuestos se descompondrán en sus atributos más simples y se harán depender directamente de la entidad.
- **Transformación de atributos múltiples**. Los atributos múltiples se convertirán en entidades débiles en existencia dependientes de la entidad o asociación de la que proceden.
- **Transformación de jerarquías**. Las relaciones de jerarquía no son fáciles de representar en el modelo relacional, por este motivo se hace necesaria la eliminación de este tipo de interrelaciones como paso previo a la traducción de modelos.

5.2.1.2. Pasos a seguir para la obtención del modelo lógico a partir del modelo conceptual

Una vez que se haya transformado el esquema conceptual para facilitar la traducción al modelo relacional, pasaremos a traducir entidades y asociaciones a relaciones, único objeto utilizado en este último modelo.

- **Transformación de entidades**. Todas las entidades se transformarán en relaciones manteniendo el tipo y número de atributos. Para ello se podrán utilizar dos notaciones distintas:
  * Como una tabla con los atributos como cabeceras de columnas.
  * Con el nombre de la ENTIDAD seguido de los atributos entre paréntesis y separados por comas.

  La clave principal se señalará subrayando el atributo o atributos que la componen; las claves alternativas se indicarán con un doble subrayado y las claves foráneas o extranjeras en negrita.
- **Transformación de interrelaciones**. Para la transformación de interrelaciones deberemos seguir las reglas que dependerán del tipo de asociación y de la cardinalidad con la que cada entidad participe en la asociación (ver volumen práctico).

5.3. Diseño de interfaces de usuario

Teniendo en cuenta que el proceso de diseño debe ir siempre enfocado a las necesidades del usuario y no del sistema que se implemente, y que la interfaz de usuario es el medio del que dispone el hombre para comunicarse con un objeto y llevar a cabo una tarea, podemos deducir la importancia que tiene un buen diseño de la interfaz de usuario.

5.3.1. Objetivos del diseño

- **Diseñar una salida para satisfacer el objetivo planteado**: toda salida debe responder a un propósito descrito en la especificación de requisitos.
- **Diseñar una salida que se adapte al usuario**: aunque no es posible diseñar una salida que se adapte a todos los usuarios que tendrá la aplicación, a base de entrevistas prototipos, etc. se diseñarán las salidas que mejor se adapten a las preferencias de los usuarios.

- **Suministrar la cantidad adecuada de información** de manera que no sea excesiva la cantidad de información que aparezca en una sola pantalla, pero tampoco hagamos que el usuario se pierda a través de muchas pantallas.
- **Asegurar que la salida está disponible donde se necesita**, teniendo en cuenta que la información se suministre al usuario adecuado para que sea bien aprovechada.
- **Proporcionar la salida en el momento oportuno** para que se puedan recibir datos por días por meses, etc.
- **Elegir el método correcto de salida en cada momento**, por pantalla, como informe impreso, etc.

5.3.2. Criterios de diseño

Existe una serie de criterios que se han de seguir para el diseño de una interfaz. Una de las principales características es la usabilidad del software. De nada sirve crear un software que lleve a un estado de confusión al usuario, por lo que dejaría de ser útil y eficiente. Una interfaz debe cumplir las siguientes características:

- Debe ser fácil de asimilar por parte del usuario.
- Debe ser flexible, dejando que el usuario pueda dirigir los procesos de forma sencilla desde distintos tipos de interfaz.
- Debe mantener la consistencia en el diseño de toda la aplicación para facilitar su uso.
- Debe ser sólido, de modo que el usuario pueda obtener los resultados deseados sin tener que subsanar las dificultades que pueda presentar el sistema.

Son principios fundamentales en el diseño de la interfaz:

- La consistencia. Para una misma función todas las pantallas deben mantener la misma estructura y utilizar el mismo método.
- La minimización del volumen de información a recordar basándonos en estructuras intuitivas.
- Proporcionar ayuda al usuario mostrando mensajes e incluyendo una opción que el usuario pueda consultar en cualquier momento.
- Evitar el cansancio del usuario mostrando la información de forma clara, sencilla y en el lugar de la pantalla apropiado.
- Proporcionar distintos modos de trabajo en las entradas de datos (ratón, teclado...).
- Minimizar la cantidad de información que el usuario deba introducir.
- Permitir que el usuario pueda "investigar" sin miedo a provocar errores irreversibles.

5.3.3. Documentación

El diseño de la interfaz de usuario debe generar también su propia documentación. Se deberán describir:

- **Las pantallas**:
  * Representación gráfica de cada uno de los formularios con las partes que lo componen.
  * Mapas de pantallas donde se muestran gráficamente cada uno de los formularios junto con las relaciones que mantienen.

  * Establecer los formularios que son más importantes porque van a ser compartidos por distintos usuarios, por su uso reiterado, etc. y que deben ser creados sin dar lugar a posibles errores.

- **Los informes**:
  * Descripción de los formatos de los informes generados.
  * Elementos vinculados directa o indirectamente con los informes.

6. CODIFICACIÓN

La finalidad del análisis y el diseño es la consecución de un software descrito en un lenguaje que pueda ser "entendido" por el ordenador. Nos encontramos ahora en la fase en la que se realiza la traducción a un lenguaje de programación. Parece que esta tarea es la que menos importancia tiene, que si se ha hecho un buen análisis y un buen diseño, el resultado tiene que ser una buena aplicación, pero sin duda, la elección del lenguaje de programación y el estilo de programación tienen mucho que decir en la calidad del software. Por ejemplo, a la hora de realizar tareas de mantenimiento, la estructura del programa y las posibilidades del lenguaje de programación serán factores muy importantes que facilitarán o encarecerán esta labor.

Características que han de tenerse en cuenta a la hora de seleccionar un lenguaje de programación u otro pueden ser:

- **Facilidad de traducción del diseño al código**: se deberá elegir un lenguaje que permita implementar las estructuras de datos y las estructuras de control utilizadas en el diseño.
- **Eficiencia del compilador**: si la aplicación necesita de un código eficiente y rápido con en el menor tamaño posible será imprescindible la elección de un lenguaje con un buen compilador.
- **Portabilidad del código fuente**: para no tener que realizar tareas de mantenimiento al cambiar el sistema operativo o el procesador de la máquina para la que fue diseñado, e incluso para facilitar la reutilización del software.
- **Disponibilidad de herramientas de desarrollo**: que reducen el tiempo de construcción y mejoran algunas características del software.
- **Facilidad de mantenimiento**: al disponer de elementos de documentación para el código.

Una vez elegido el lenguaje de programación sólo queda realizar la codificación que, como hemos dicho anteriormente, también es una tarea que influye en la calidad del producto final. No pretendemos exponer aquí cómo se realiza la codificación, pero sí indicaremos algunos principios que deben observarse a la hora de realizar esta tarea y que ayudarán a obtener un código estructurado, claro, fácil de comprender, sencillo y elegante.

- Utilizar un estilo de programación estructurada usando estructuras de control o simulándolas en el caso de que el lenguaje no las implemente directamente.
- Utilizar las instrucciones GOTO de forma controlada, sobre todo, para simular estructuras de control en lenguajes que no las implementan.
- Utilizar un estilo de programación modular intentando en lo posible aislar las estructuras de datos junto a sus funciones de acceso.

- Utilizar todas las posibilidades del lenguaje para documentar cada una de las llamadas a los módulos que componen el programa.
- Utilizar rutinas de entre cinco y 25 líneas de código (salvo algunas excepciones).
- Cuidar el formato de presentación introduciendo sangrías, líneas en blanco, etc.
- Buscar siempre la simplicidad y la sencillez; buscar siempre la solución más simple.
- Evitar anidaciones profundas y proposiciones "then" nulas.
- Utilizar nombres apropiados para los identificadores.
- Otro principio fundamental en la codificación de los programas es la documentación. Se deben utilizar todas las posibilidades del lenguaje para documentarlo de forma concisa y clara, sin comentarios obvios.

7. PRUEBAS

Las estrategias de prueba de programa van a tener una triple finalidad: van a ser útiles para el programador, para medir la calidad del software, y para el cliente.

Si queremos obtener el máximo rendimiento a una prueba debemos realizar un diseño de la prueba. En este diseño debemos incluir:

1. Planificación de la prueba.
2. Diseño de los casos de prueba.
3. Ejecuciones de las pruebas.
4. Recopilación de datos y evaluación de los resultados.

Una prueba del software debe ser lo suficientemente flexible para promover la creatividad y la adaptabilidad del software, y lo suficientemente rígida para permitir un buen seguimiento de la misma.

7.1. Pruebas del software

Las pruebas del software son un conjunto de actividades que permiten llevar a cabo una planificación y seguimiento ordenado de las mismas.

En estas pruebas el ingeniero debe ayudarse de una plantilla para su seguimiento. Esta plantilla debe tener las siguientes características:

- Comienza a nivel de módulo y va avanzando hasta el sistema completo ya integrado.
- Podemos aplicar simultáneamente varias técnicas de prueba.
- La prueba la llevan a cabo el programador y un grupo de prueba.
- Podemos incluir la depuración dentro de las técnicas de prueba.

Una estrategia de prueba debe integrar pruebas de bajo nivel y de alto nivel. Debe ser una guía tanto para el programador como para el director del proyecto.

7.2. Verificación y validación

Las pruebas del software forman parte de lo que podemos denominar como verificación y validación del software. La verificación se refiere al hecho de comprobar si estamos desarrollando el software correctamente. La validación se encarga de comprobar si estamos construyendo el producto correcto.

7.3. Organización de la prueba

Normalmente son los desarrolladores del software las personas que empiezan a probar el software. Empiezan probando cada uno de los módulos que han ido diseñando, hasta llegar a las pruebas del sistema completo.

Los desarrolladores del software sí van a encontrar bastantes errores en las pruebas modulares, pero normalmente no suelen encontrar demasiados fallos en las pruebas del sistema. Este hecho se puede entender fácilmente, porque son ellos mismos quienes han diseñado el programa y conocen demasiado bien su uso y manejo. Saben siempre qué botón, tecla, o función deben usar en cada momento.

Para las pruebas del sistema son mucho mejores los grupos independientes de prueba o GIP. El GIP pertenece al equipo del proyecto, pero deben ser usuarios o programadores ajenos al mismo. Su misión es probar las versiones beta del programa e informar a los desarrolladores del mismo de los fallos, incoherencias, posibles mejoras aplicables al programa, etc. Cuanto más alejado del proyecto se esté, mejor. El GIP debe estar en contacto permanente con los desarrolladores durante las fases de prueba.

8. INSTALACIÓN

Esta tarea tiene como finalidad la puesta en marcha del sistema desarrollado y la realización de una evaluación del resultado para, por último, hacer entrega de la aplicación al cliente.

8.1. Instalación

Una vez instalado el equipo y antes de que el sistema se ponga en funcionamiento, se habrán debido convertir los datos almacenados bajo el viejo sistema al sistema nuevo y poco a poco se irán convirtiendo los datos históricos.

Con el nuevo sistema se hará entrega de la documentación que le acompaña. Esta documentación la componen: la guía de instalación, que indicará a nivel técnico cuáles serán los pasos a seguir para la instalación; las especificaciones de la aplicación, que indicarán detalles técnicos sobre la aplicación y cada uno de sus módulos; las notas de liberación, como complemento a la guía de instalación y para corregir posibles errores detectados después de terminada la guía; el manual de procedimientos, donde se indica al usuario la manera de proceder con el nuevo sistema y, por último, el manual de administración del sistema, para la persona responsable del sistema en la empresa.

Se deberá comenzar a utilizar el sistema para realizar transacciones reales, y así comprobar su correcto funcionamiento antes de la entrega definitiva al usuario. Para ello, en un paso previo, los usuarios habrán recibido la formación adecuada que les permita operar con el nuevo sistema y será en este momento cuando, comprobando el funcionamiento real, puedan surgirle dudas que estarán a tiempo de resolver.

El traspaso final al nuevo sistema puede realizarse de alguna de las siguientes formas:

- **Proceso encadenado**: en el proceso encadenado continúa funcionando el sistema antiguo para la realización de las transacciones, sin embargo, en un proceso posterior, el nuevo sistema realizará la misma transacción. Los resultados del nuevo sistema se evaluarán y compararán con los producidos por el sistema antiguo. Este método de implantación se utiliza para sistemas que requieren mucha fiabilidad, ya que se pueden comprobar los resultados del sistema antes de que su operatividad sea definitiva.
- **Proceso directo**: el proceso directo consiste en la desactivación del sistema antiguo y la activación del nuevo sistema, que será el encargado de realizar las transacciones. Evidentemente no es recomendable para sistemas que requieran extrema fiabilidad; es una posibilidad cuando el sistema no es demasiado complejo y en algunas ocasiones es la única posibilidad si no es posible la convivencia de los dos sistemas.
- **Proceso en paralelo**: los dos sistemas, en este proceso, convivirán realizando las transacciones simultáneamente de esta forma. Cuando la evaluación de los resultados del nuevo sistema sea positiva se podrá desactivar el antiguo. Se suele utilizar este método si el sistema exige una alta fiabilidad o si los dos sistemas son tan distintos que no se producirán salidas duplicadas.
- **Proceso por subsistemas**: los dos sistemas, antiguo y nuevo, se reparten el trabajo en este método. Unas transacciones se realizarán en un sistema y otras en el otro, de forma que se irán sumando subsistemas poco a poco al sistema nuevo de una forma planificada.

8.2. Evaluación y ajuste

Una vez obtenidos resultados del nuevo sistema se evalúan y se corrigen los posibles errores que se hayan detectado, ya sean de diseño o de desarrollo, y se corrigen detalles que no sean del total agrado del cliente, teniendo en cuenta los siguientes aspectos:

- Los errores serán corregidos en su totalidad y sin excepciones.
- Si los errores son críticos, se debe suspender la actividad del nuevo sistema y volver a activar el antiguo.
- Si no lo fueran, debería seguir la actividad del nuevo sistema y resolver el problema lo antes posible.

8.3. Informe final

Se habrá de realizar un documento final donde se informe del estado del nuevo sistema y se incluyan la correcciones realizadas, las observaciones de los usuarios, etc. Dicho informe deberá ser aprobado por el cliente.

9. EXPLOTACIÓN

En esta etapa el usuario se hace cargo de la aplicación y de la operación del nuevo sistema. Para ello cuenta con la formación adecuada, la documentación necesaria y un sistema en perfecto funcionamiento.

10. MANTENIMIENTO

Tarde o temprano cualquier aplicación va a necesitar ser modificada, ya sea por que en el sistema se introduce un hardware nuevo, porque se descubren errores que no habían sido detectados, o simplemente porque cambian las necesidades del cliente. A este proceso es a lo que se le denomina "mantenimiento". Por este motivo y para facilitar el futuro trabajo de mantenimiento los desarrolladores deben poner especial cuidado a la hora de elaborar la documentación del proyecto, que deberá ser estructurada, clara, legible y detallada.

Otros factores que también facilitarán esta tarea son:

- La independencia de los módulos que conforman la aplicación, de esta forma se podrá proceder a la modificación de los que se vean implicados en el mantenimiento sin que esto afecte a los demás.
- El lenguaje de programación elegido, ya que dependiendo del nivel del lenguaje (si es un lenguaje de alto nivel se facilita la tarea de comprensión de las líneas de código) y de las características del mismo (posibilidad de introducir comentarios, etc.), la tarea de localización de errores y de modificación es más sencilla.
- El trabajo y el tiempo invertido en la validación y prueba de los programas que componen la aplicación, ya que son una inversión a largo plazo.

10.1. Tipos de mantenimiento

Ya que existen diferentes factores que producen que se dispare el mantenimiento de una aplicación, podemos también encontrar distintos tipos de mantenimiento: correctivo, adaptativo, perfectivo y preventivo.

- El mantenimiento **correctivo** es el que se encarga de la corrección de errores detectados en la codificación, en el diseño o en los requisitos. Evidentemente, el coste de este mantenimiento irá creciendo a medida que los errores se encuentren en una fase más temprana en el proceso de elaboración de la aplicación. Por ejemplo, un error de codificación será siempre menos costoso que un error en los requisitos.
- El mantenimiento **adaptativo** es el que se produce por la necesidad de adaptar la aplicación a un hardware (p.ej. nuevos soportes para los datos) o un software nuevo (p.ej. nuevo sistema operativo) introducido en el sistema.
- El mantenimiento **perfectivo** es el que se encarga de adaptar el software a nuevas necesidades del cliente producidas por cambios en su empresa.
- El mantenimiento **preventivo** introduce cambios en el software para aumentar su calidad y su seguridad sin introducir nuevas funcionalidades pero que servirán para facilitar posteriores tareas de mantenimiento.

En algunos casos se pueden entender también como tareas propias del mantenimiento las labores encaminadas a la localización y almacenamiento de funciones y procedimientos elaborados para su inclusión en una biblioteca y reutilizar para nuevas aplicaciones el software ya construido.

10.2. Actividades de mantenimiento

Es un error comparar el mantenimiento del software con el mantenimiento del hardware. Mientras que este último puede resumirse en el cambio de componentes desgastados por otros nuevos, para poder mantener el software se han de realizar tareas más complejas.

- Se deberá empezar por comprender la aplicación, para lo que se deberá estudiar su estructura y su funcionalidad, además de comprender las modificaciones que se deben realizar y los nuevos requisitos que se han de añadir.
- El segundo paso será modificar propiamente el software, teniendo en cuenta los efectos que tendrá la modificación sobre el resto de componentes, así como modificar la documentación y las interfaces de la aplicación.
- El tercer paso consiste en la realización de pruebas que permitan confirmar la funcionalidad del módulo, modificado así como la funcionalidad de la aplicación completa.

Todas estas acciones deberán tener en cuenta también que sucesivas tareas de mantenimiento sobre una aplicación no deberían contribuir a su deterioro progresivo, ya que se podría perder la estructura de la aplicación, la cohesión de sus módulos y hacer que la documentación no sea fiable.