DISEÑO DE PROCESOS Y PANTALLAS

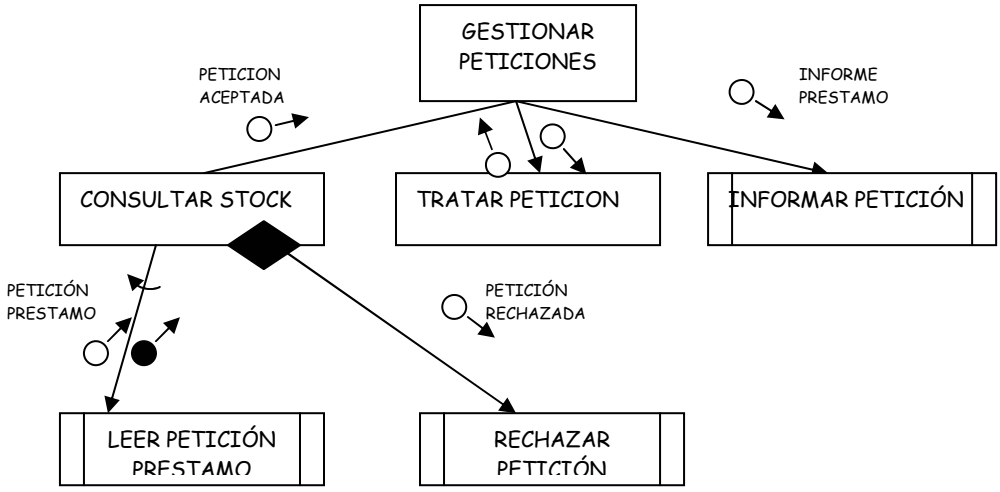
8.1.- Introducción

El objetivo principal del diseño estructurado es desarrollar la estructura de programa así como las relaciones entre los elementos, denominados *módulos* que componen dicha estructura.

El diseño, orientado al flujo de datos, permite establecer la transición del DFD a una descripción de la estructura del programa que se representa mediante un *diagrama de estructuras*.

Este capítulo pretende, pues, explicar el diagrama de estructuras y los pasos a realizar para convertir un diagrama de flujo expandido (esto es, formado sólo por los procesos primitivos) en diagrama de estructuras.

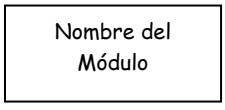
8.2.- Diagrama de estructuras



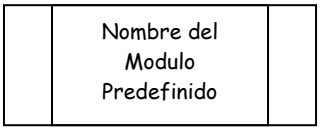
Los elementos principales de un diagrama de estructura (en metodología Métrica se denomina *Diagrama de Cuadros de Constantine*) son módulos (rectángulos), conexiones entre módulos (flechas) y comunicaciones entre módulos. Los módulos pueden comunicarse entre sí por medio de estructuras de datos o de control (*flags*).

Alguna de las características del diagrama de estructura son las siguientes:

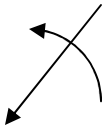
El nombre del módulo tiene que representar la función que realiza.



Un módulo "*predefinido*" es aquel que se encuentra disponible en la biblioteca del sistema o de la aplicación (puede ser un módulo del Sistema Operativo o de un Gestor de Bases de Datos) y, por tanto no es necesario codificarlo. Los módulos predefinidos se marcan de un modo especial:

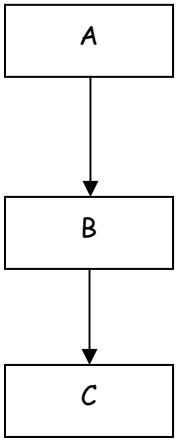


Existen convenciones gráficas para representar estructuras repetitivas:



O alternativas:

Un diagrama de estructura y un organigrama aunque pueden parecer semejantes son conceptualmente distintos. Si se considera la figura adjunta y se simula que un ordenador la ejecuta en primer lugar como Organigrama y posteriormente como diagrama de estructura, se obtendrá que:



Dado que un organigrama muestra un flujo de control, es decir, la secuencia de los pasos a ejecutar, esta secuencia será A , B , C .

Sin embargo, el diagrama de estructura muestra la jerarquía, es decir, qué funciones son Subfunciones de una función, por tanto, en la figura, si se asume que cada llamada se ejecuta una sola vez, la secuencia sería:

A empieza

A llama a B

B empieza

B llama a C

C empieza

C hace su función

C devuelve el control a B

B acaba su función

B devuelve el control a A

A acaba su función

Es decir, B es una subfunción de A y C es una subfunción de B por lo que el orden en el que se han ejecutado es C , B , A (suponiendo que las secuencias "A empieza" y "B empieza" son vacías.

El orden de las llamadas de un diagrama de estructura es de Arriba abajo y de Izquierda a derecha.

8.2.1.- Módulo

Dado que la arquitectura implica modularidad, el software se divide en elementos (módulos) que se integran entre sí para, al ejecutarse, satisfacer los requisitos del sistema.

El módulo es pues una unidad claramente definida y manejable, con interfaces modulares perfectamente definidas. Suele definirse como "*La unidad mas pequeña de código que puede ser compilada independientemente*" aunque definiciones mas académicas son:

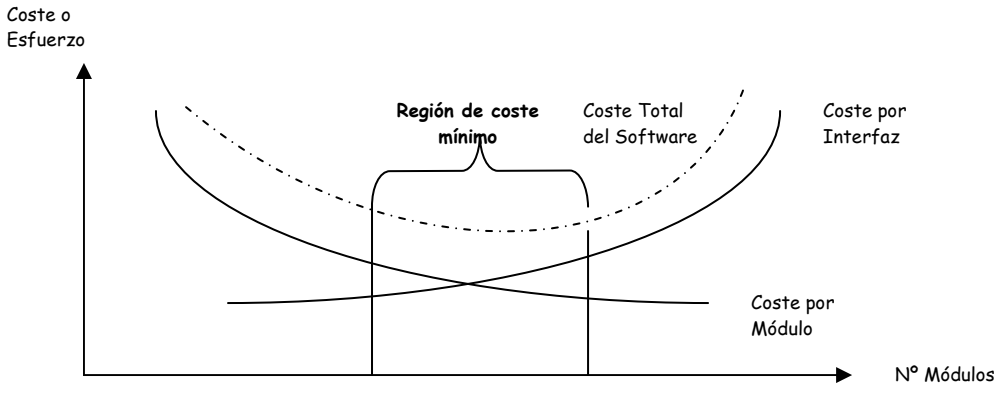
Parte lógica separable de un programa (AECC, 1986)

Secuencia contigua de sentencias de programa limitada por delimitadores y que tiene un identificador global (Yourdon y Constantine, 1979)

Cualquier objeto que, en un nivel de abstracción dado, quiera considerarse como un concepto simple (Fenton, 1991)

En Teoría del Diseño Estructurado un módulo es aquella parte de código que se puede llamar (Page-Jones,1988).

En general puede considerarse el módulo como el conjunto de sentencias que hacen una actividad. Suele tener 40 ó 50 líneas de código aunque no existen restricciones al respecto.



Puede demostrarse que, aunque no existe un número máximo de módulos en un diagrama, si se subdivide el software indefinidamente, el esfuerzo requerido para el desarrollo de cada módulo será infinitamente pequeño, pero, a medida que crece el número de módulos, el esfuerzo asociado con las interfaces entre módulos también crece ., Existe, sin embargo, una región de coste mínimo, no predecible con seguridad, que determinará el intervalo óptimo en el número de módulos en la aplicación.

8.2.2.- Conexión entre módulos

Un sistema está compuesto por módulos organizados jerárquicamente, cooperando y comunicándose entre sí para realizar una tarea. La llamada de un módulo se realiza mediante una flecha.

En la figura de inicio del capítulo, el módulo GESTIONAR PETICIONES llamará al módulo CONSULTAR STOCK y éste al módulo LEER PETICIÓN PRÉSTAMO. Una vez realizada la lectura, el control lo tendrá el módulo CONSULTAR STOCK, que rechazará la petición o devolverá el control al módulo principal GESTIONAR PETICIONES.

Posteriormente, el módulo principal llamará al módulo TRATAR PETICIÓN, que se ejecutará y devolverá el control al módulo principal para, finalmente ejecutar el módulo INFORMAR PETICIÓN. Ejecutado este último módulo se devuelve el control al módulo principal que finalizaría la ejecución.

8.2.3.- Comunicación entre módulos

La comunicación intermodular se realiza por medio de los *datos* y de los *flags* . Los datos se procesan; por el contrario, los flags sólo sirven como valores de condición para comunicar condiciones entre los módulos. Además los datos están relacionados con el problema y son importantes en el mundo exterior, mientras que los flags solo importan para la comunicación de la información. Los datos y los flags se representan mediante un círculo con una flecha, y es el sentido de la flecha el que indica la dirección del dato o flag. Para los datos el fondo del círculo es blanco, mientras que para los flags es negro.

En la figura anterior los módulos sólo se pasan datos entre ellos, que podrán ser de entrada, de salida , de entrada/salida o, incluso, no pasarse datos.

Normalmente, los flags no se representan en el diagrama de estructura, pero hay que tenerlos en cuenta a la hora de programar. En el ejemplo anterior se considera un flag de control que va desde el módulo LEER PETICIÓN PRÉSTAMO al módulo CONSULTAR STOCK. El flag indicaría el fin de petición de prestamo (su nombre sería FIN PETICIÓN PRÉSTAMO) e indicaría al módulo que lo recibe (CONSULTAR STOCK) si existen o no mas peticiones de préstamo.

8.3.- Tabla de Interfaz

Los parámetros que se pasan entre módulos se suelen representar en una *tabla de interfaz* que permite una mejor especificación de los parámetros y sirven de apoyo a los diagramas de estructuras mejorando su calidad (cuando el número de parámetros en la interfaz es mayor de cuatro, el diagrama de estructuras puede resultar muy confuso).

MÓDULO	PARÁMETRO FORMAL	ENTRADA	SALIDA	USO	SIGNIFICADO PARÁMETRO
F(x,y)	X	Sí	No	P	Fecha Nacimiento
	Y	No	Si	M	Edad

La tabla de interfaz señala, como puede verse, para cada llamada:

1º.- El módulo llamado

2º.- Cada parámetro formal

3º.- Si el parámetro es de entrada (marcando la columna correspondiente)

4º.- Si el parámetro es de salida (marcando la columna correspondiente)

5º.- El uso de cada parámetro

6º.- El significado de cada parámetro.

El uso y el significado son constantes para cada llamada del módulo. La primera ventaja de la utilización de la tabla de interfaz consiste en verificar que todos los parámetros utilizados sirvan para un propósito determinado.

La columna de uso normalmente se especifica mediante nemotécnicos que indican cómo se utiliza el parámetro en el módulo. Así:

NEMOTÉCNICO	SIGNIFICADO
P	El parámetro es PROCESADO: a = b + 2
M	El parámetro es MODIFICADO: a = 3 + b
T	El parámetro es TRANSFERIDO por el módulo llamado a otro módulo que éste llama, sin modificar su valor
C	El parámetro se usa como VARIABLE DE CONTROL, para actuar como índice conmutador, como un valor de un flag o como especificación de una función que es usada en el módulo llamado
I	El parámetro es TRANSFERIDO a otro módulo y es MODIFICADO en este segundo módulo

En el ejemplo del diagrama de estructura de la figura que se está considerando, el problema consiste en gestionar las peticiones de préstamos de libros en una biblioteca. El proceso a seguir consiste en leer esta petición, consultar si se encuentra disponible dicho libro en la biblioteca, caso en el que se tratará la petición generando el informe del libro que se guardará hasta que el libro sea devuelto, o no se encuentra disponible dicho libro o esté siendo utilizado por otra persona, caso en el cual se rechazará esta petición.

Aunque el número de parámetros a pasar entre módulos es muy pequeño, por lo que haría innecesaria la tabla de interfaz, la tabla de interfaz entre los módulos TRATAR PETICIÓN e INFORMAR PETICIÓN será:

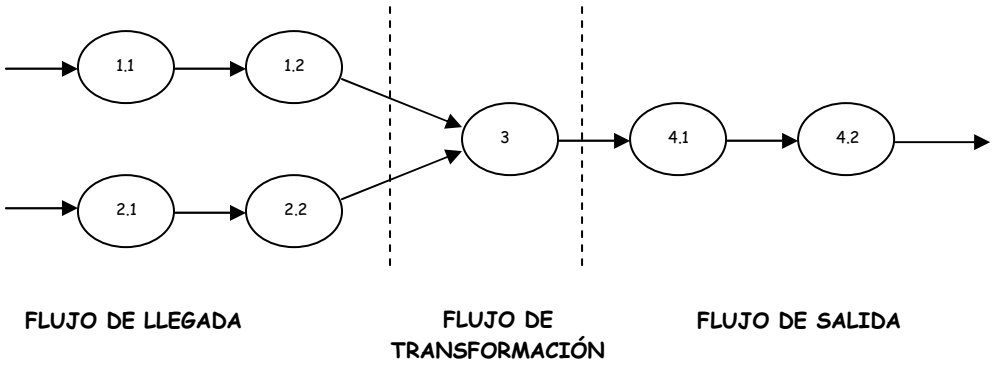
MÓDULO	PARÁMETRO FORMAL	ENTRADA	SALIDA	USO	SIGNIFICADO PARÁMETRO
TRATAR PETICIÓN	Petición Aceptada	Sí	No	P	Petición Aceptada
	Informe Préstamo	No	Sí	I	Informe del Préstamo
INFORMAR PETICIÓN	Informe Préstamo	Sí	No	P	Informe de Préstamo

8.4.- Estrategias de Diseño

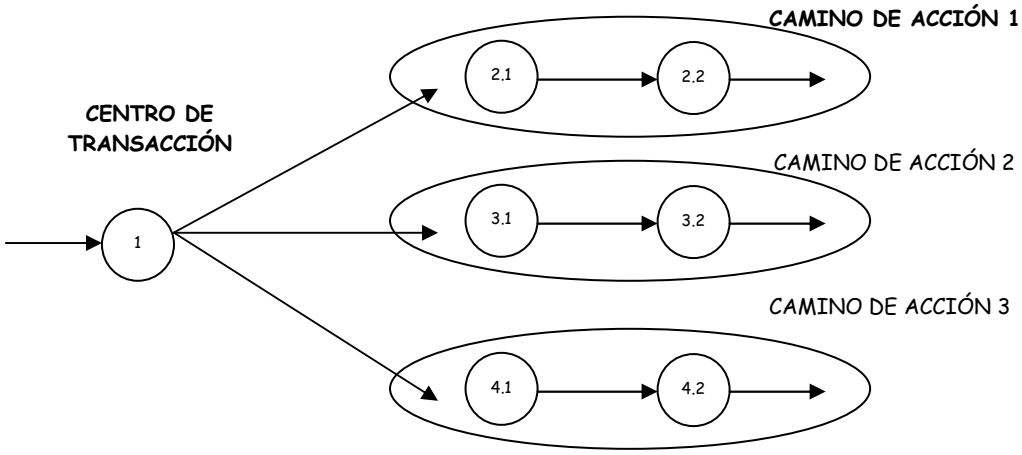
El diseño estructurado ofrece dos estrategias para conseguir una creación rápida de un buen diseño a partir de una ERS (Especificación de Requisitos del Sistema). Estas dos estrategias son el Análisis de Transformación y el Análisis de Transacción.

El Diseño Estructurado permite una transición de las representaciones de la información (DFD o Diagrama de Flujo de Datos) a una descripción de diseño de la estructura del programa, y existe una variación en los pasos para lograrlo en función del tipo de información de que se trate.

En el *Análisis de Transformación* los datos entran en el sistema mediante caminos que se denominan **flujos de llegada**. En el núcleo se realiza la transformación de los datos que llegaron anteriormente. Finalmente dichos datos se mueven por caminos que conducen a la salida, constituyendo el **flujo de salida**. Cuando un DFD es de este tipo, se dice que tiene características de *Transformación.*



En el *Análisis de Transacción*, el centro de transacción es el **centro de flujo** de información desde el que emanan muchos **caminos de acción** de forma exclusiva . Los caminos de acción son los caminos por donde puede pasar el flujo de información. Cuando un DFD es de este tipo se dice que tiene características de *Transacción*.



8.4.1.- Análisis de Transformación

Es un conjunto de pasos de diseño que permiten a un DFD obtenido en la fase de Análisis, con características de flujo de transformación, convertirse en una estructura (o *template*) predefinida del sistema.

Los pasos del Análisis de Transformación son:

Revisión del modelo fundamental del sistema

Si el análisis estructurado está precedido del diseño estructurado se tendrán los DFD del sistema, en caso contrario habrá que obtenerlos de las especificaciones funcionales.

Como mínimo han de obtenerse en el análisis estructurado 3 niveles de profundidad (*diagrama de contexto, diagrama de sistema y diagramas de nivel 2*) para poder aplicar el diseño estructurado con suficiente detalle.

Determinación de las características del DFD (*Transformación o Transacción*)

Aunque generalmente el flujo de información de un sistema puede representarse siempre como transformación, conviene elegir, para este análisis únicamente aquellos DFD's con claras características de transformación.

Para determinar la característica global del flujo se atiende a la naturaleza prevalente del DFD. Si existe un proceso tal que tiene salidas exclusivas, se estará tratando posiblemente con un problema de transacción, aunque, a veces, este proceso está oculto (es decir, no está reflejado en sl DFD) en la especificación de requisitos software del sistema a tratar

Aislamiento del Centro de Transformación, especificando los límites de flujo de llegada y de salida

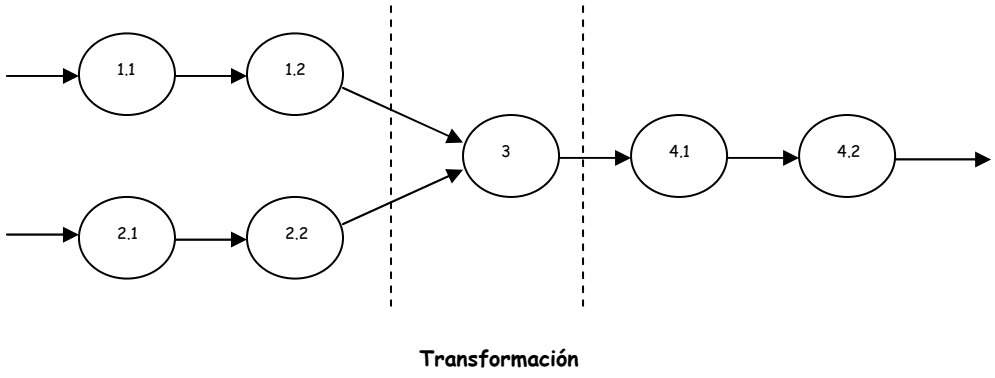
El Centro de transformación es la parte del DFD que contiene las funciones esenciales del sistema, independientemente de la implementación particular de la entrada y de la salida.

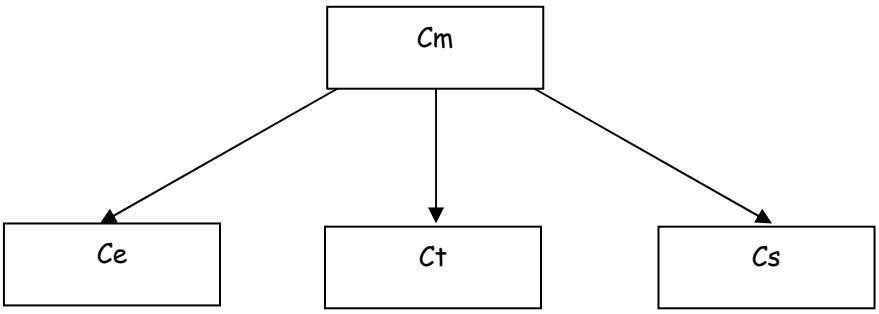
Los límites del flujo de llegada y de salida están sujetos a la interpretación del diseñador por lo que pueden presentarse soluciones alternativas del diseño en función de dichos límites. Si bien, cualquier pequeña variación tendrá un impacto en la estructura final del sistema.

Realización del primer corte del diagrama de estructuras

El propósito del análisis de transformación es convertir un DFD en un diagrama de estructuras para este tipo de transformación. Ya que el diagrama de estructuras describe una estructura jerárquica y el DFD no.

La estructura del sistema representa una distribución descendente del control. La aplicación del análisis de transformación (*descomposición o factorización del sistema*) da como resultado una estructura del sistema en la que los módulos de nivel superior toman las decisiones de ejecución y los módulos de nivel inferior ejecutan la mayoría del trabajo de entrada, de cálculo y de salida. Los módulos de nivel intermedio ejecutan algún control y realizan cantidades de trabajo moderadas.





En la figura se representa el primer nivel de factorización o descomposición del diagrama de estructuras, que consta de un módulo Cm que es el principal y coordina los módulos colocados en el primer nivel del diagrama de estructuras, que son:

- *Un módulo controlador del procesamiento de la información de llegada al sistema* Ce que coordina la recepción de todos los datos que llegan.
- *Un módulo coordinador del Centro de Transformación* Ct que supervisa todas las operaciones sobre los datos.
- *Un módulo controlador del procesamiento de la información de salida al sistema* Cs que coordina la producción de la información de salida.

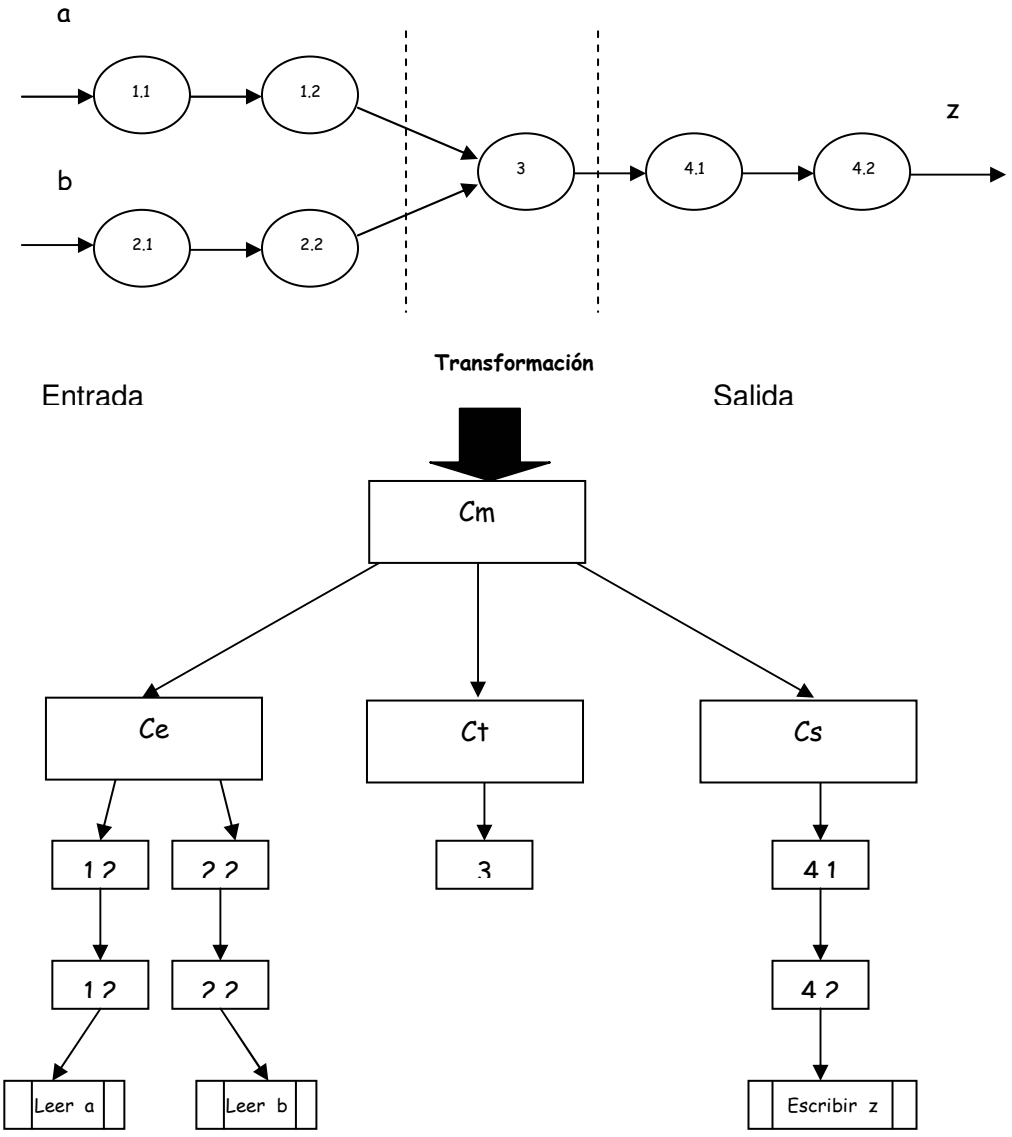
Todos estos módulos han de tener un nombre significativo que refleje todo lo que se encuentra por debajo de cada uno de ellos. Así, por ejemplo, el nombre del módulo C m suele hacerse coincidir con el nombre del diagrama de contexto obtenido en la fase de análisis. Para los módulos C e , C t y C s sus respectivos nombres deben reflejar la entrada, transformación y salida de la información (El módulo C e deberá reflejar las funciones 1.1, 1.2, 2.1 y 2.2)

Ejecución del "segundo nivel de factorización"

El segundo nivel de factorización se realiza mediante la conversión de las transformaciones individuales de un DFD (procesos) en los módulos correspondientes de un diagrama de estructuras.

Se empieza en el límite del centro de transformación y, dirigiéndose hacia fuera a lo largo de los caminos de llegada y salida, las transformaciones (procesos del DFD) se convierten en módulos subordinados de la estructura del software. Además, es necesario introducir módulos predefinidos que proporcionen las diferentes entradas y/o salidas que

necesita y/o genera el sistema. La representación gráfica de esta ejecución se muestra en la figura adjunta:



Refinamiento la estructura del sistema utilizando medidas y guías de diseño

Se puede aumentar o disminuir el número de módulos para producir una factorización lógica que tenga una buena calidad y una estructura que se implemente sin dificultad, se pruebe sin confusión y se mantenga sin problemas. Los refinamientos están dictados por consideraciones prácticas y por el sentido común, además de por los requisitos del software.

Así, en el ejemplo anterior, un refinamiento sería eliminar el módulo controlador C t dado que no parece tener mucho sentido tener un módulo que controle a otro módulo, por lo que el módulo C m llamaría directamente al módulo 3.

En general, los refinamientos típicos suelen consistir en reducir el nivel de profundidad del diagrama de estructuras.

Definición de los parámetros necesarios para la ejecución de los módulos

Una vez realizado el diagrama de estructuras, debe reflejarse cuales son los parámetros que necesita para que pueda ejecutarse cada uno de los módulos que componen el diagrama. Dichos parámetros, datos y/o flags se obtienen del DFD realizado en la fase de análisis.

Los datos se corresponden con los flujos de información de los DFD's y se han de reflejar en el diagrama de estructuras como datos que se pasan entre los módulos.

Los flags se obtienen de las descripciones de los procesosy se reflejarán en el módulo correspondiente del diagrama de estructuras cuando se necesite una variable de control. Normalmente los flags no se representan en el diagrama de estructuras.

Debe indicarse que, cuando un proceso del DFD tenga que acceder a un almacén, tanto en modo de lectura como de escritura, se refleja en el diagrama de estructuras haciendo depender del módulo correspondiente a ese proceso los módulos predefinidos que correspondan, de lectura o escritura, según indique el DFD. Así, lo que se intenta conseguir es que el acceso a bases de datos de algunos módulos se refleje en el diagrama de estructuras, de forma que la programación sea mucho mas fácil.

Verificación del trabajo realizado por el diseño obtenido.

Debe garantizarse que la funcionalidad que realiza el diseño creado es la correcta y se corresponde con el enunciado del problema diseñado. Para ello, por ejemplo, se revisará el diagrama de estructura observando que el orden de ejecución de los módulos es el correcto.

8.4.1.1.- Ejemplo de Análisis de transformación

Se intenta desarrollar un Sistema de Información que apoye a la Gestión de una Central de Compras (CC), que permita hacer pedidos globales por temporada (juguetes en Reyes, Artículos de Camping en verano, etc.).

La función de la CC consiste en obtener, a partir del Catálogo de los proveedores y del Archivo Histórico de Ventas de los almacenes, el Pedido Global, teniendo en cuenta los pedidos individuales (Pedido rellenado). No obstante la CC puede modificar el alta o la baja de la cantidad pedida por almacén en función de una serie de factores (umbrales de descuento por diversas cantidades, etc.), notificándoselo a cada almacén (Notificación de Pedido) a la vez que comunica el Pedido Global (cantidades definitivas de los productos) de todos los almacenes a cada proveedor seleccionado.

El diccionario de datos es el siguiente:

Documentos Almacén = Histórico Ventas + Pedido Rellenado

Pedido Global = Notificación Pedido

El Diagrama de Flujo de Datos del problema es el que se ofrece en las figuras adjuntas.

Se pide identificar la característica global del flujo de datos, junto con el centro de transformación o transacción correspondiente, y generar el Diagrama de Estructuras asociado.

Diagrama de Contexto (Nivel 0)

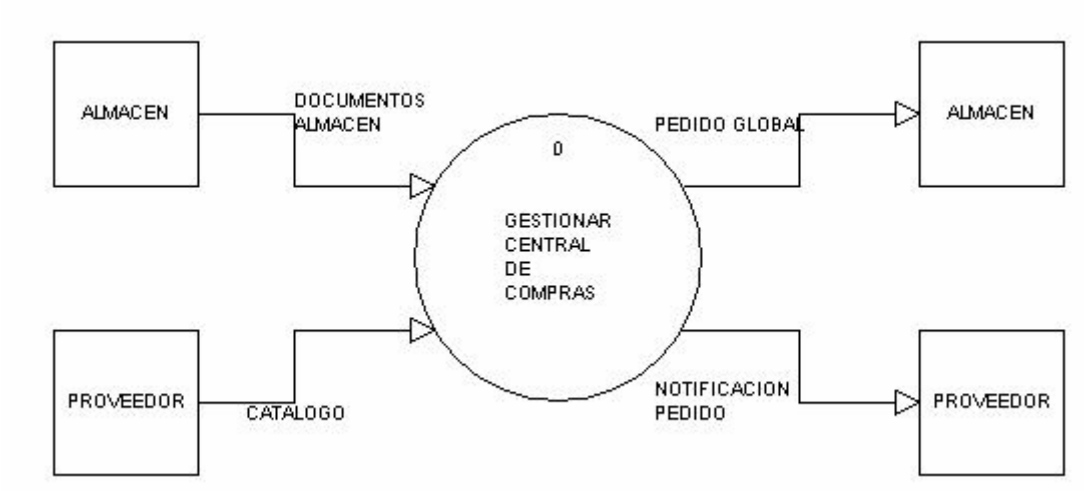


Diagrama del Sistema (Nivel 1)

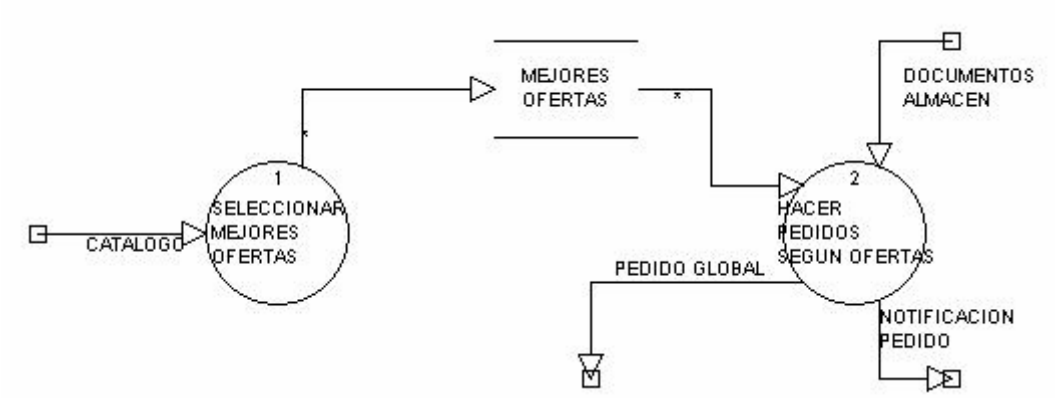


Diagrama "Seleccionar Mejores Ofertas" (Nivel 2)

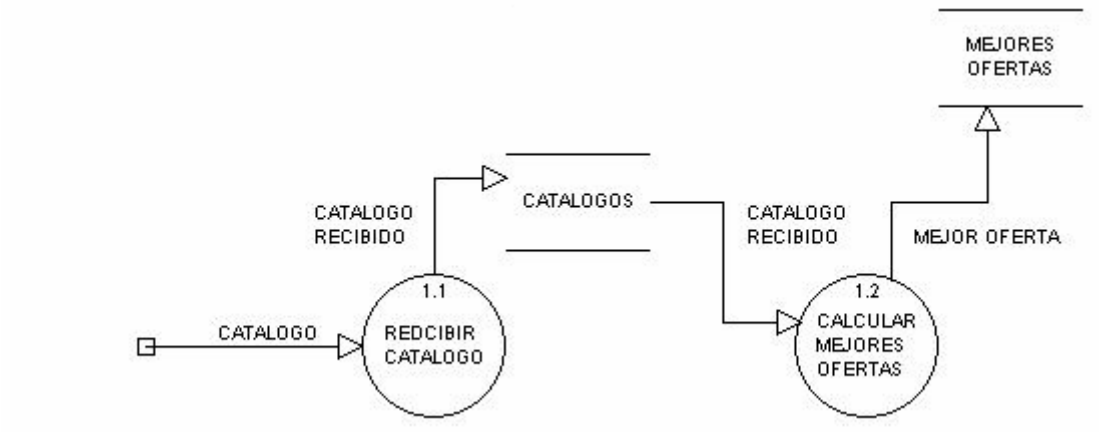
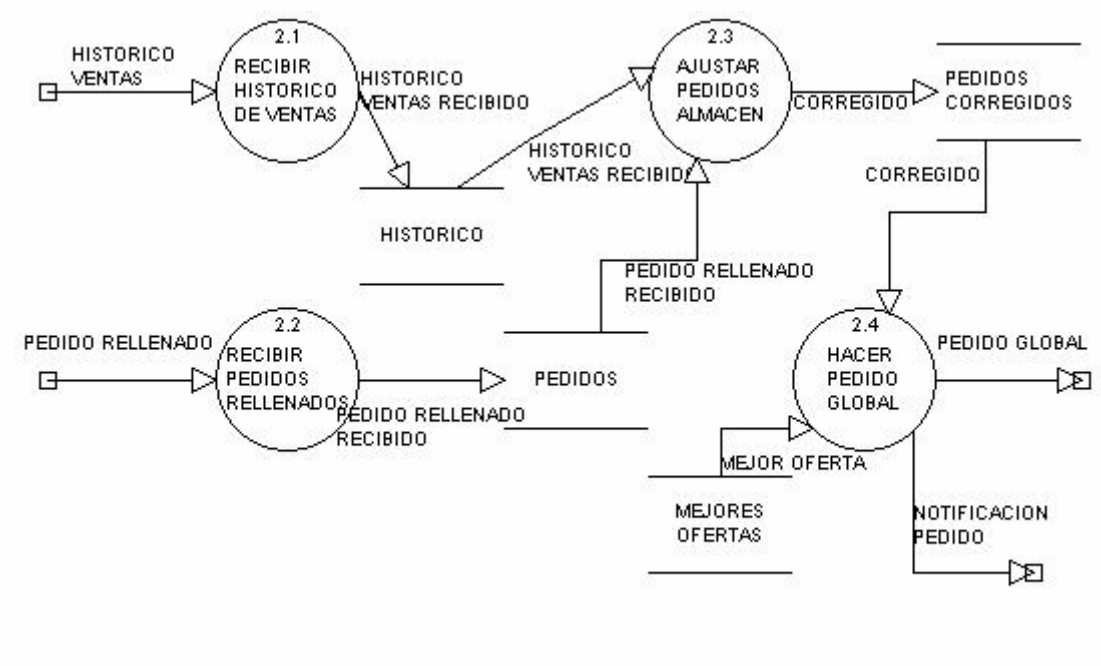


Diagrama "Hacer Pedidos según Ofertas" (Nivel 2)



Solución:

La característica global del flujo es de transformación, dado que no existe ningún procese que tenga salidas exclusivas (excluyentes entre sí) y el enunciado del problema tampoco indica nada al respecto.

Para identificar el Centro de transformación se considera que, por cada uno de los diferentes de entrada al proceso de contexto (flujos de Histórico de Ventas, Pedido Rellenado, y Pedido Global) y de salida (flujos de Notificación Pedido y Pedido Global) se ha de progresar hacia la parte central del DFD hasta el momento en que el flujo se transforme totalmente.

Por ejemplo, por el camino de entrada de "Histórico de Ventas" se pasa a "Histórico de Ventas Recibido" (el flujo de datos no ha sufrido una transformación total y puede seguir hacia delante) y pasa posteriormente a "Corregido". En este último caso, cuando la información pasa de "Histórico de Ventas Recibido" a "Recibido", se produce una verdadera transformación total del flujo de llegada, por lo cual, el proceso que realiza tal transformación (proceso 2.3 "Ajustar Pedidos Almacén") estaría dentro de la parte de transformación.

Analizando los otros caminos de entrada ("Pedido Rellenado" y "Catálogo") se observa que, para el primero, la transformación se produce también en el proceso 2.3, y, para el "Catálogo", la transformación de este flujo de datos se produce en el proceso 1.2. El corte pues, para la entrada al sistema (Flujo de llegada) se realiza en las entradas de los procesos 2.3 y 1.2.

Para la salida se realiza el mismo procedimiento, pero en sentido inverso. En este caso se estudiarán los dos flujos de salida ("Notificación Pedido" y "Pedido Global") por separado. (En este caso, dado que según el diccionario de datos, ambos flujos contienen la misma información, se pueden tratar de manera única).

Al seguir el flujo de información desde la salida hacia la parte central del DFD se observa que de "notificación Pedido" se pasa a "Corregido" y "mejor Oferta". Como existe

una diferencia notable entre dichas informaciones y la de "Notificación Pedido" se llega a la conclusión de que el proceso 2.4 es el que realiza esta transformación.

En resumen, el centro de transformación está formado por los procesos 2.3 , 1.2 y 2.4.

Según los diferentes diseñadores que aborden el problema, pueden darse diferentes soluciones dependiendo de los cortes que se realicen. Por ejemplo, puede suponerse que el proceso 2.4 lo único que hace es reunir los datos de "Corregido" y los de "Mejor Oferta", por lo que, en este caso, no formaría parte del centro de transformación, pasando a formar parte de la salida. El centro de transformación lo constituirán entonces los procesos 2.3 y 1.2.

En esta última solución propuesta, hay que destacar que en el caso de la entrada, y dado que existen dos flujos de información distintos, se opta por añadir dos módulos controladores de entrada ("Recibir Documentación Almacén" y "Recibir Catálogo"). Uno de los módulos controladores se ha agrupado con el módulo de "recibir Catálogo" en el paso de refinamiento.

El módulo controlador de la transformación, como depende del controlador y siempre se puede evitar, se ha eliminado.

El módulo controlador de la salida se ha agrupado con el proceso 2.4 en el paso de refinamiento debido a que se tendría un módulo controlador y, por debajo de éste otro módulo que realizaría la función del proceso 2.4.

Deben indicarse el paso de parámetros entre los diferentes módulos que forman el diagrama de estructuras. Estos coinciden con los flujos de datos que se mueven a lo largo del DFD correspondiente.

El diagrama de estructuras obtenido es el siguiente:

H-V = Histórico Ventas

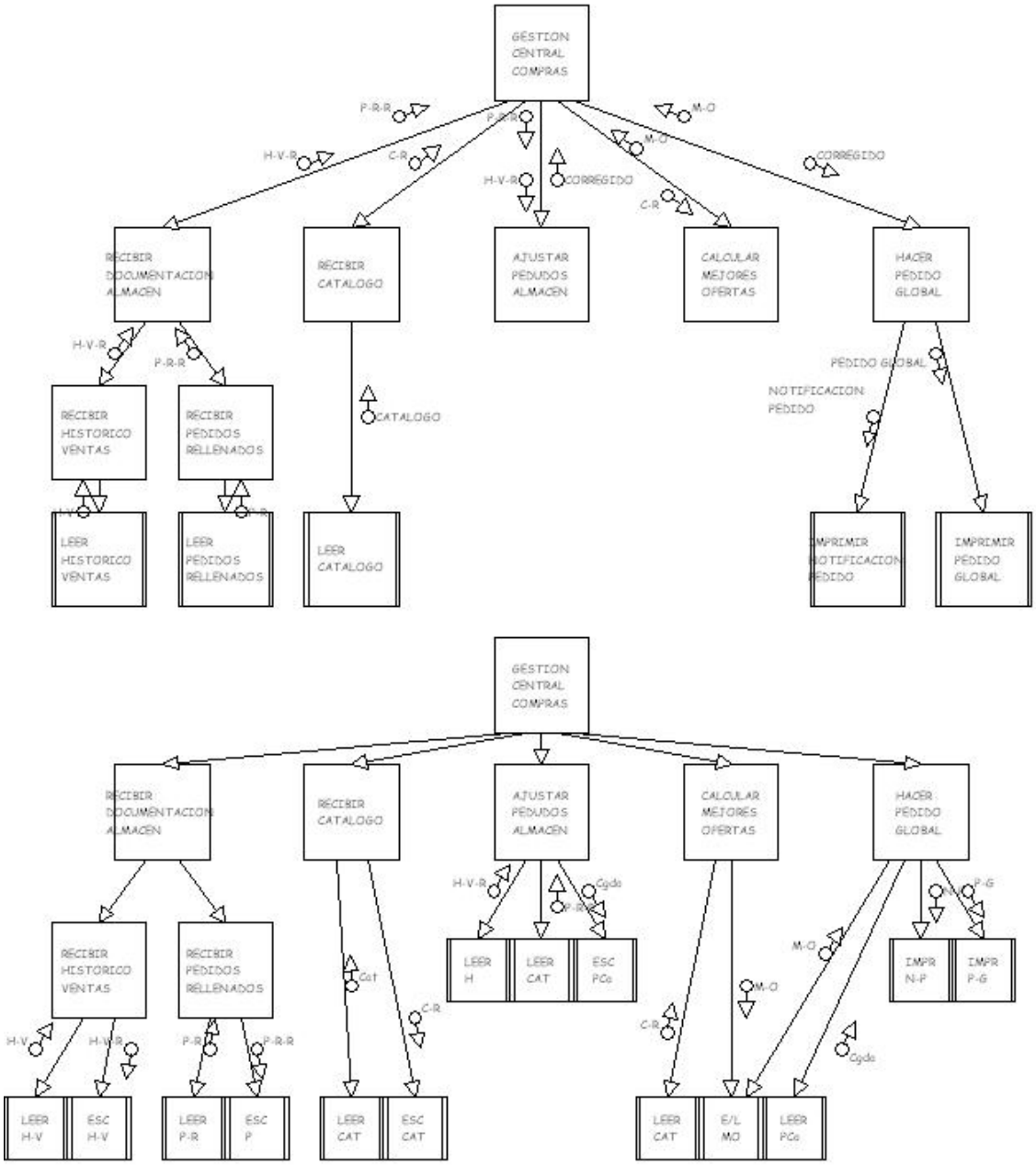
H-V-R = Histórico Ventas Corregido

P-R = Pedido Rellenado

P-R-R = Pedido Rellenado Recibido

C-R = Catálogo Recibido

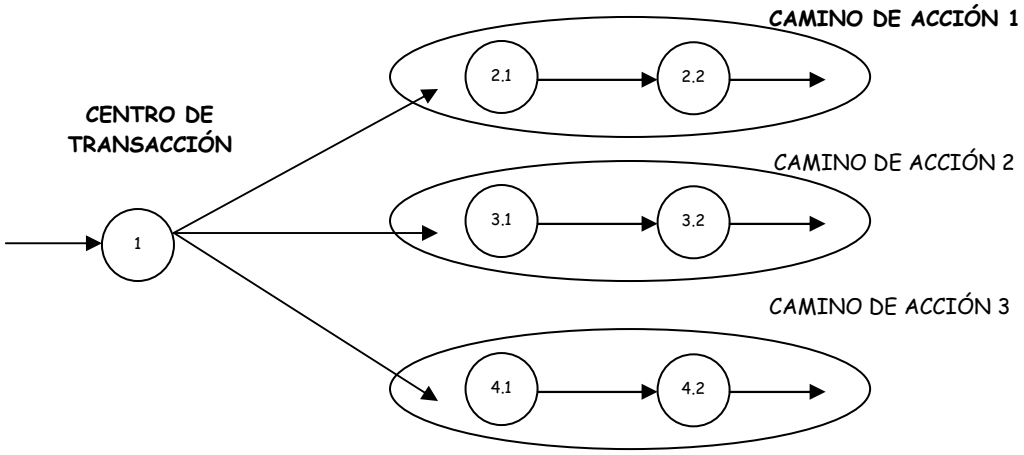
M-O = Mejores Ofertas



Además de los parámetros, conviene representar en el diagrama de estructuras los accesos a almacenes. En este caso, dado que un almacén se corresponderá con una o varias tablas correspondientes al diseño de la base de datos, lo que se hace es añadir al diagrama de estructuras generado los módulos predefinidos correspondientes de lectura y/o escritura, según los procesos del DFD lean o escriban en ellos.

8.4.2.- Análisis de transacción

En muchas aplicaciones software un dato determina caminos alternativos por los que puede transitar el flujo de información; dependiendo del camino tomado varía la función realizada sobre el dato tratado. En estos casos el elemento de datos se llama *transacción*. Este elemento desencadena otro flujo de datos a lo largo de uno de los muchos caminos.



El ***centro de transacción*** es el centro del flujo de información desde el que emanan muchos *caminos de acción* (exclusivos entre sí).

En un flujo orientado a transacción, el flujo de información a lo largo de un cam9ino de acción puede ser tanto de transformación como de transacción.

Los **pasos del diseño para el Análisis de Transacción** son similares y, en algunos casos, idénticos, a los pasos seguidos en el análisis de transformación. La principal diferencia se refiere a la conversión del DFD en la estructura del sistema.

Revisión del modelo fundamental del sistema

Este paso es idéntico al paso correspondiente del análisis de transformaciones

Determinación de las características del DFD (Transformación o Transacción)

Este paso es idéntico al paso correspondiente del análisis de transformaciones

Identificación del Centro de Transformación y las características del flujo de cada camino de acción

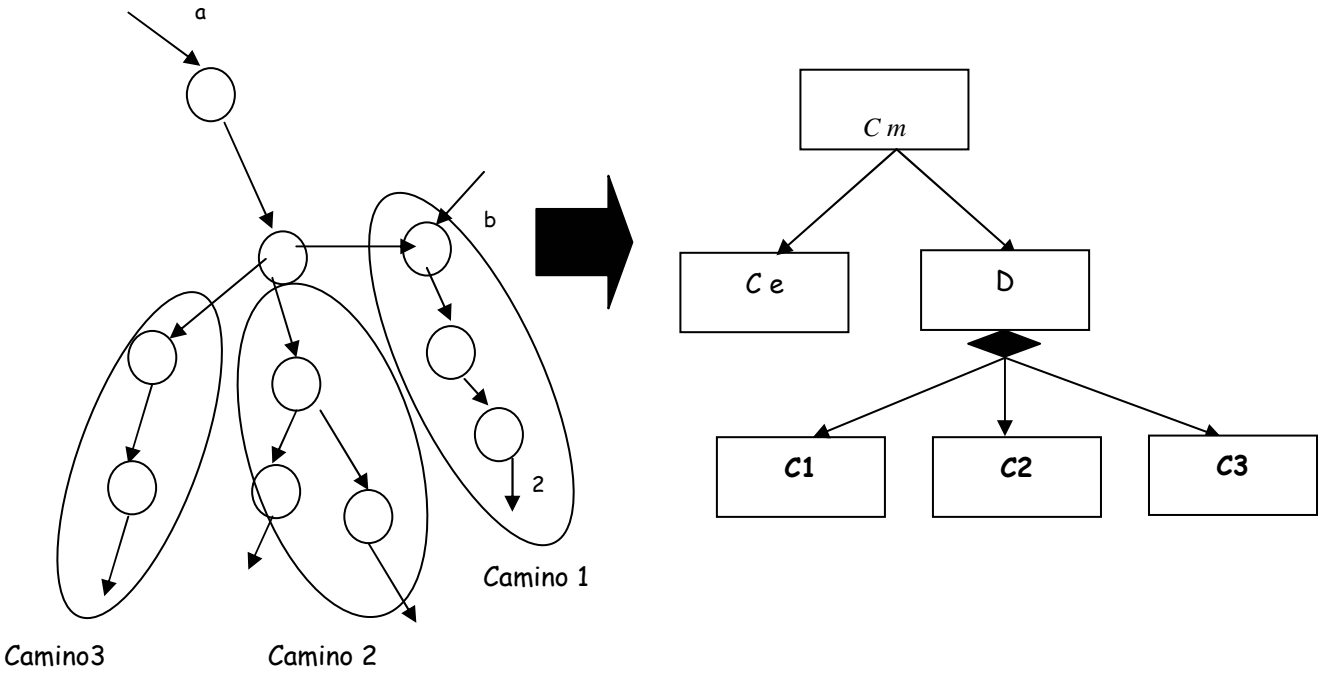
La posición del centro de transacción se descubre inmediatamente a partir del DFD. Está ligado al origen de varios caminos de información que fluyen radialmente de él.

En general. El proceso del DFD que corresponde a la transacción, no suele reflejarse en dicho DFD, por lo que es preciso conocer bien el sistema para identificar dónde hay entradas al sistema que son exclusivas entre sí, y que, lógicamente se corresponden con cada una de las entradas de los diferentes caminos de acción.

El camino de llegada y todos los caminos de acción deben aislarse también, cada campo de acción debe evaluarse en función de sus características individuales de flujo (tipo transformación o tipo transacción).

Realización del primer corte del diagrama de estructuras (primer nivel de factorización)

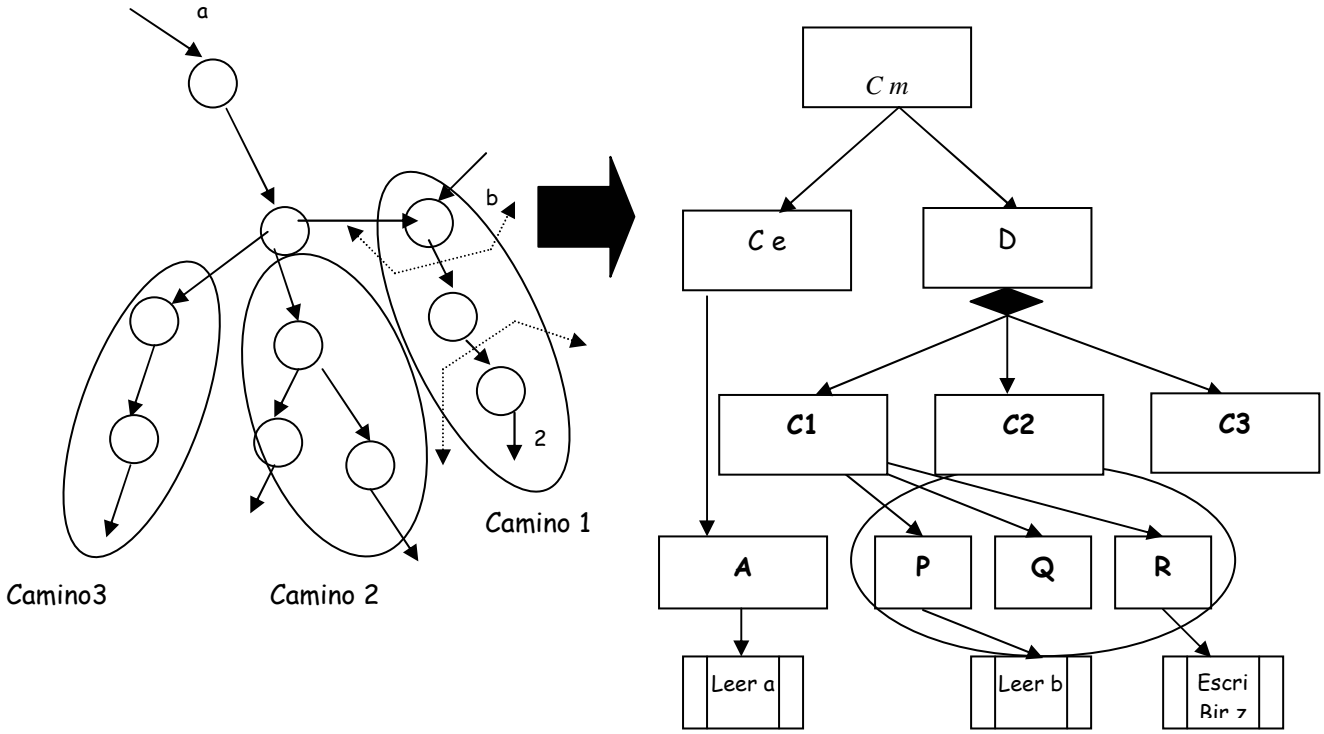
Hay que convertir el flujo de transacción en una estructura del sistema con una bifurcación de entrada y una bifurcación de salida. Para la entrada se hace igual que en el análisis de transformación. Para la salida, se añade un módulo controlador por cada flujo de acción



Nótese que el módulo que se corresponde con el centro de transformación refleja la exclusividad de los diferentes caminos por medio de un rombo del cual parten los diferentes módulos controladores de cada camino de acción.

Realización del segundo nivel de factorización

Se desarrolla cada camino de acción dependiendo de su tipo de flujo, Por ejemplo, la figura adjunta sigue un flujo de tipo transformación.



Refinado de la estructura del programa

Este paso es idéntico al paso correspondiente del análisis de transformaciones

Definición de los parámetros necesarios para la ejecución de los módulos

Este paso es idéntico al paso correspondiente del análisis de transformaciones

Verificación del trabajo realizado por el diseño obtenido.

Este paso es idéntico al paso correspondiente del análisis de transformaciones

8.4.2.1.- Ejemplo de Análisis de Transacción

La Comunidad autónoma ha decidido hacerse cargo de la gestión de la piscina "La Pistolera". Para ello sólo dejará acceder a dichas instalaciones, de forma gratuita, a aquellos usuarios que sean estudiantes o trabajadores locales. Dependiendo del tipo de usuario (estudiante o trabajador) se realizarán diferentes tratamientos. Se pide:

Aislar el centro de transformación o de transacción.

Realizar el diagrama de estructuras.

Diagrama de Contexto

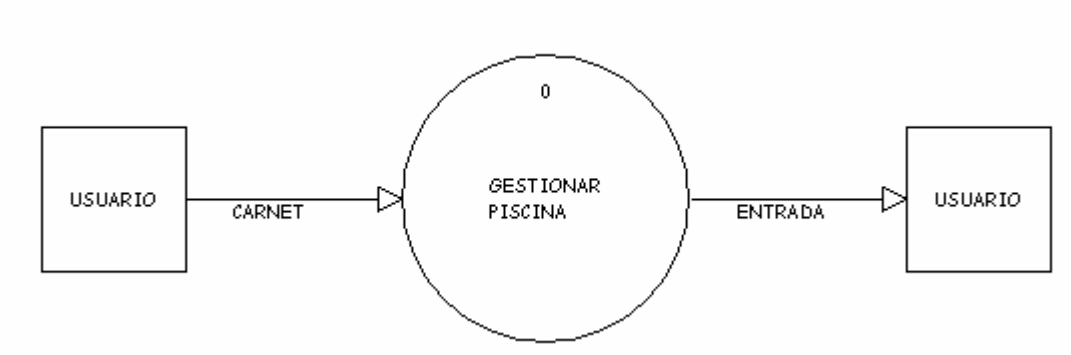


Diagrama de Sistema

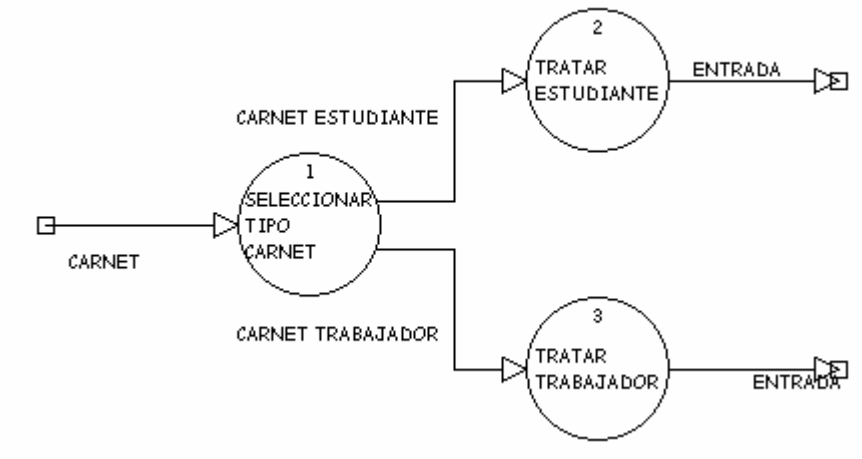
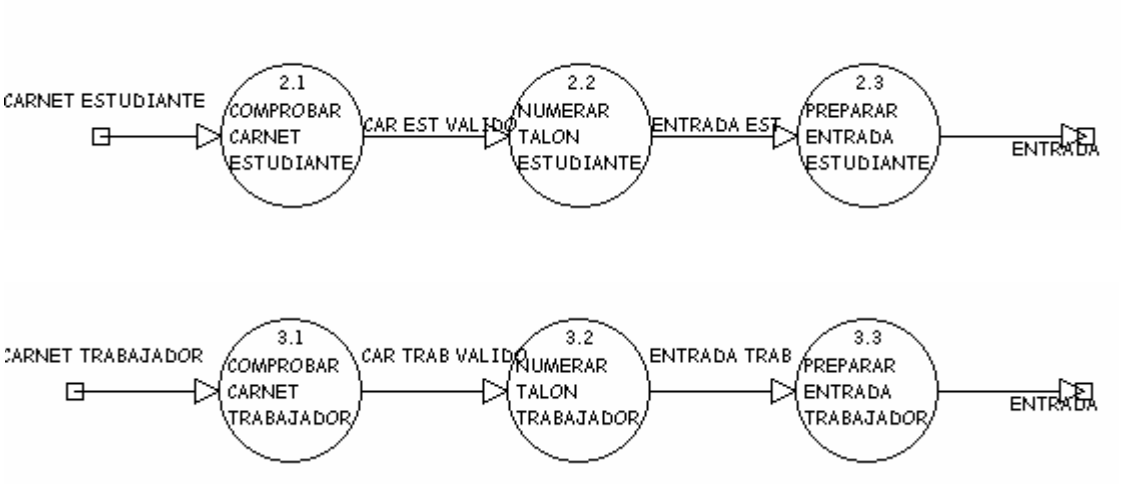


Diagrama de Nivel 2



La característica global del flujo es de transacción, dado que existe un proceso (centro de transacción) del que se obtienen dos salidas excluyentes entre sí. En el DFD se refleja por los flujos "CARNET ESTUDIANTE" y "CARNET TRABAJADOR". Este proceso no tiene por qué expresarse en el DFD por lo que tendría que ser necesario deducirlo a partir de la Especificación de Requisitos de Software (en particular, en este ejemplo, el enunciado indica que: "dependiendo del tipo de usuario se realizarán diferentes tratamientos").

El proceso 1 del DFD es pues el centro de transacción.

Seguidamente se realiza el primer corte, esto es, la división en:

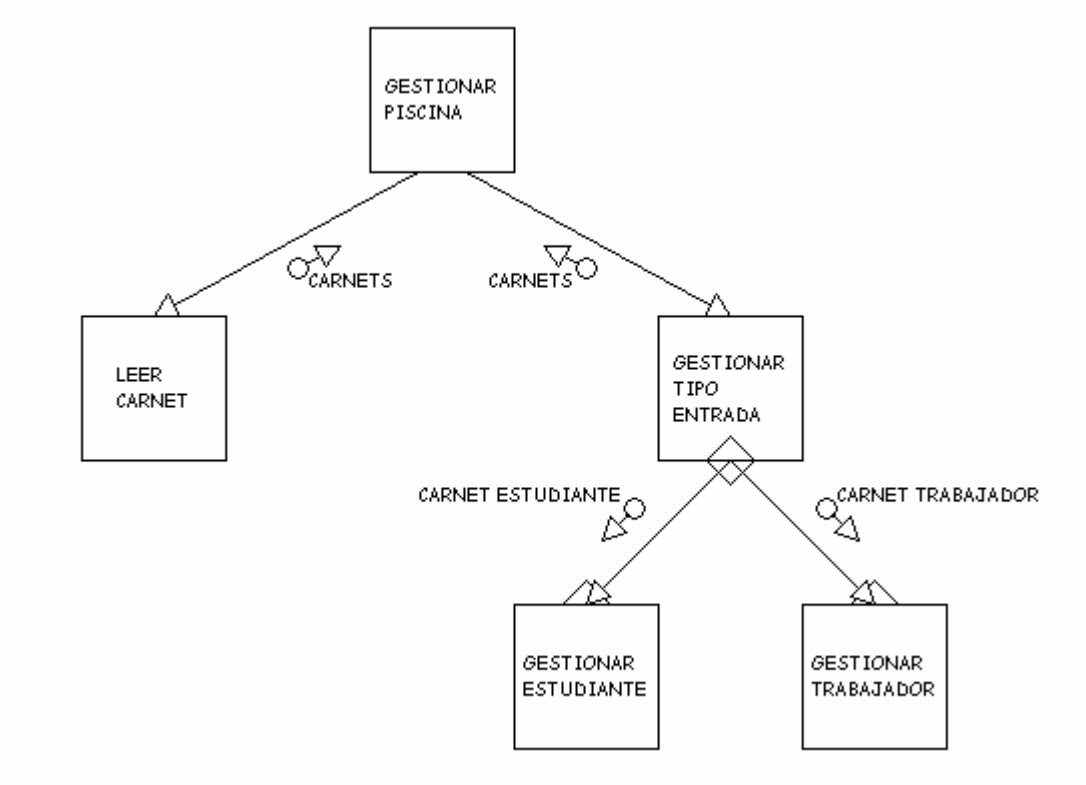
- *Módulo que controla el sistema* (que coincide con el nombre del diagrama de contexto de la fase de análisis)
- *Módulo que controla la entrada* (Corresponde al módulo "LEER CARNET", que podría considerarse como predefinido)
- *Módulo que controla la transacción* Es el módulo 1 al que se ha puesto por nombre el sinónimo "GESTIONAR TIPO DE ENTRADA")
- *Dos módulos controladores, uno para cada camino alternativo* ("GESTIONAR ESTUDIANTE" y "GESTIONAR TRABAJADOR")

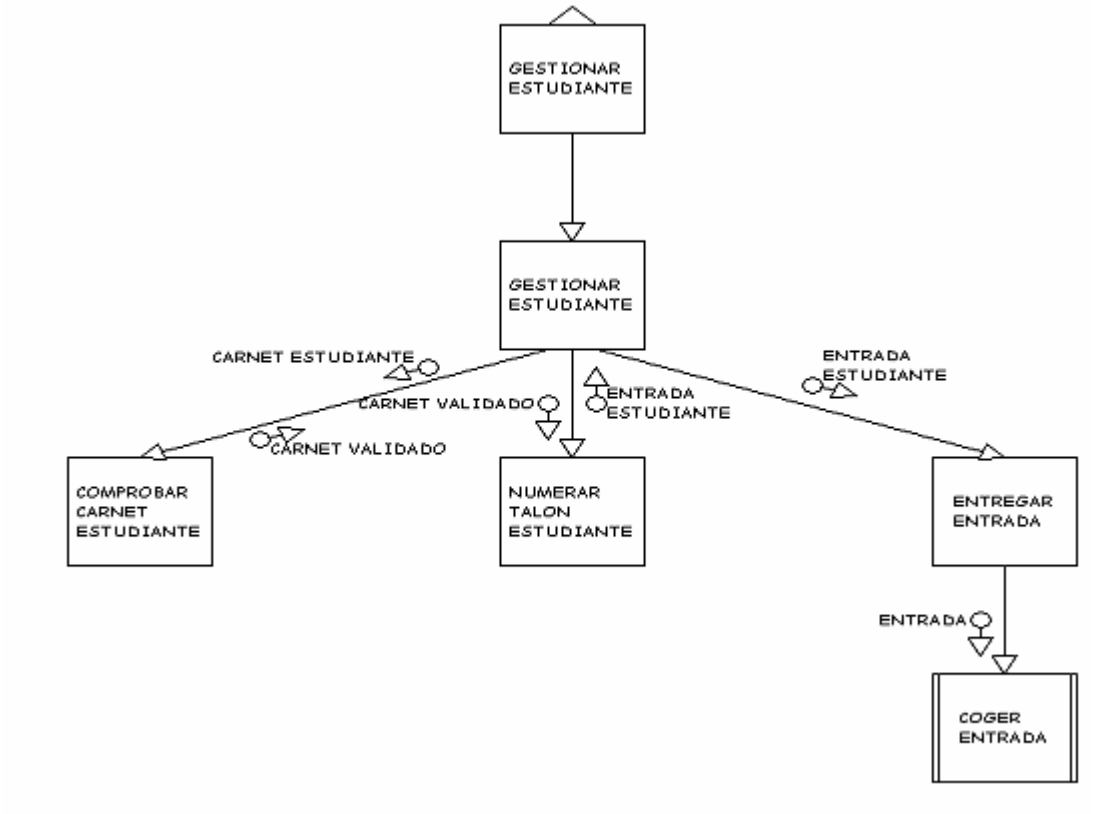
Una vez realizado el primer corte se ha de evaluar nuevamente cada uno de los caminos de acción que se obtengan. En este caso, los dos caminos son similares y presentan una transformación, por lo que se ha de desarrollar la estructura de transformación para cada uno de ellos.

En la solución propuesta en la figura adjunta se han eliminado los módulos controladores (ya que no tiene sentido poner un módulo controlador para controlar un único módulo), poniendo únicamente los módulos correspondientes a procesos del DFD y los módulos predefinidos correspondientes.

Por otra parte, en el ejemplo no se desarrolla el segundo camino de acción por ser totalmente similar al primero.

El diagrama de estructuras obtenido es el siguiente:





Deben tenerse en cuenta las siguientes observaciones:

1º.- Empleo de conectores cuando el diagrama es demasiado grande. (En la figura se ha utilizado como conector un rectángulo con un triángulo encima)

2º.- Paso de parámetros. Debe considerarse la posibilidad de utilizar un flag de control desde "LEER CARNET" hasta "GESTIONAR PISCINA" y desde este módulo hasta "GESTIONAR TIPO DE ENTRADA", ya que, en este momento, el módulo "GESTIONAR TIPO DE ENTRADA" sabe qué tipo de acción va a realizar, por lo que cada camino tendrá que leer su propia entrada con lo que la estructura de los caminos se corresponde con la vista en la teoría.

3º.- En el ejemplo, al no existir almacenes en los DFD's, tampoco existen módulos predefinidos de lectura y/o escritura en el diagrama de estructuras correspondiente.

8.5.- Atributos de la calidad de un diseño

En cada proyecto debe decidirse cuales son los requisitos de calidad a cumplir y decidir cuales son los mas importantes. Pero para poder asegurar y evaluar la calidad del software, ésta se debe poder medir. Para ello se emplean métricas del software (que se verán posteriormente) y medidas. En este apartado se consideran las métricas que miden los aspectos de calidad estructural en el diseño: la cohesión y el acoplamiento.

8.5.1.- Acoplamiento

Se define el ***acoplamiento como el grado de interdependencia entre módulos.***

Cuanto menor sea el grado de acoplamiento mejor será el diseño del sistema, esto es, el hecho de hacer los módulos tan independientes unos de otros como sea posible conseguirá un acoplamiento mínimo de forma que ningún módulo tenga que preocuparse de los detalles de la construcción interna del resto de los módulos.

8.5.1.1.- Niveles de acoplamiento

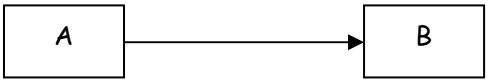
En la figura adjunta se presentan los distintos niveles de acoplamiento ordenados desde "mas deseable" hasta "menos deseable". La línea punteada indica el límite entre acoplamientos deseables y no deseables

Acoplamiento Normal

Se dice que dos módulos A y B están normalmente acoplados si:

- *El módulo A llama al B*
- *B retorna el control al módulo A.*

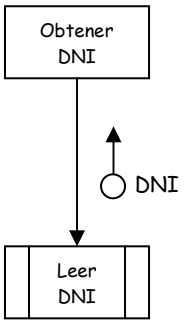
Es decir, dos módulos están acoplados normalmente si no se pasan ningún parámetro entre ellos, sólo existe una llamada de uno al otro.



Acoplamiento de datos

En el acoplamiento de datos los módulos se comunican mediante parámetros, y, cada parámetro constituye una unidad elemental de datos.

Esto implica que todas las entradas y salidas al y desde el módulo llamado se pasan como argumentos y todos los argumentos son elementos de datos (o estructuras de datos) y no de control.

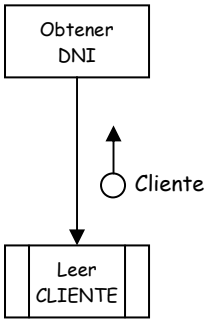


Debe reducirse por tanto el número de parámetros que se intercambian entre los módulos, a menos que tengan una utilidad práctica.

El acoplamiento de datos es el tipo de acoplamiento deseable en el Diseño estructurado.

Acoplamiento por estampado

Dos módulos están acoplados por estampado si ambos hacen referencia a la misma estructura de datos, siempre que ésta no sea una estructura de datos global.



Es importante que la estructura sea local, es decir, utilizable sólo por los dos módulos acoplados y no por otros, ya que, si la estructura fuera global el acoplamiento se degradaría.

Un módulo pasa al otro sólo una parte de la estructura de datos (por ejemplo, un registro con campos), por lo que hay que tener cuidado con dos aspectos:

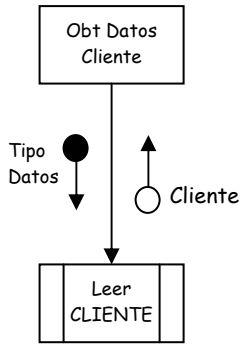
No pasar campos innecesarios del registro procurando comunicar el número mínimo de datos. Campos innecesarios oscurecen el diseño y reducen la flexibilidad.

No agrupar parámetros sin relación alguna en un mismo registro. Esto es, no almacenar los parámetros desordenadamente, ya que se complica el diseño.

Acoplamiento de control

Dos módulos están acoplados por control cuando uno de ellos pasa al otro módulo elementos de control (código de funciones, flags, switches) como argumentos.

No es un buen tipo de acoplamiento porque no permite que los módulos sean totalmente independientes debido a que un módulo controla a otro, influyendo en su ejecución.



Cuando un módulo necesite flags o códigos de función, será conveniente reemplazar dicho módulo por tantos módulos como sea necesario para que éstos tengan una única funcionalidad y sea suficiente que intercambien datos. Así, el intercambio de parámetros de control (flags) no será necesario, el acoplamiento disminuye y se obtiene un mejor diseño.

A partir de ahora se examinarán los tipos de acoplamiento que salen fuera del concepto de la buena modularidad y que, por tanto, son poco aceptables.

Acoplamiento externo

Dos módulos tienen acoplamiento externo si ambos hacen referencia a una variable global, pero las referencias entre los módulos consisten en registros individuales de datos y no en una estructura global de datos.

Este tipo de acoplamiento hace difícil una modificación sin que se tenga que revisar todo el sistema.

Un ejemplo típico de este tipo de acoplamiento puede darse cuando una variable (por ejemplo, "nombre del cliente") se define como global en el sistema y puede acceder a ella más de un módulo.

Acoplamiento común

Un grupo de módulos están acoplados comúnmente cuando comparten una estructura global de datos (entorno común). Un ejemplo de este tipo puede darse cuando una estructura de datos, por ejemplo "datos del cliente" (DNI, Nombre y Apellidos, Dirección, ...) se define como global en el sistema y se puede acceder a ella desde mas de un módulo.

El acoplamiento externo es un caso particular del acoplamiento común ya que en el acoplamiento externo la referencia es una variable global y en el acoplamiento común la referencia es una estructura de datos global.

Acoplamiento por contenido

Es un tipo de acoplamiento patológico, por lo que hay que evitarlo a toda costa. Lo que es lo mismo que decir que, un diseño, con este tipo de acoplamiento es inaceptable.

Dos módulos presentan acoplamiento por contenido si uno hace referencia al interior del otro. Estas referencias pueden hacerse de maneras diversas:

- *Un módulo modifica algún elemento en el otro módulo.*

- *Un módulo utiliza una variable local del otro.*
- *Desde un módulo se salta a otro, perola sentencia a la que pasa no está definida como punto de entrada.*
- *Dos módulos comparten los mismos contenidos.*

Los módulos acoplados por contenido son muy dependientes los unos de los otros y una pequeña modificación en uno puede producir un mal funcionamiento en el otro. Esta situación de dependencia hace muy difícil realizar un cambio aislado en un módulo sin que afecte a todo el programa.

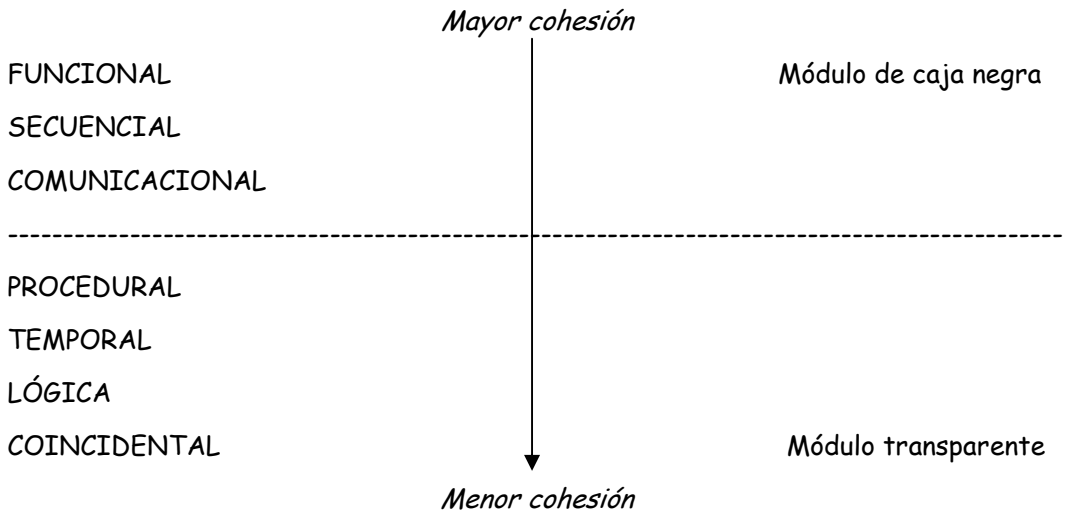
8.5.2.- Cohesión

Estudia la relación existente entre los elementos de un mismo módulo.

La intención del concepto de cohesión es organizar estos elementos de tal manera que tengan una mayor relación a la hora de realizar una tarea pertenezcan al mismo módulo y los elementos no relacionados se encuentren en módulos separados.

La cohesión se define como la medida de la relación funcional de los elementos de un módulo. Entendiendo por elemento cualquier pieza de un módulo (instrucción, grupo de instrucciones, procedimientos, etc.) que lleva a cabo algún trabajo o define algún dato.

Idealmente, un módulo coherente debe hacer una cosa única. Una escala de los tipos de cohesión ordenados de mayor a menor es la siguiente: (La línea puntead indica el límite entre cohesiones deseables e indeseables).



Cohesión funcional

Todos los elementos que componen un módulo están relacionados en el desarrollo de una única función. En este caso no tiene por que haber parámetros de entrada o de salida. Es el grado mas alto de cohesión.

Cohesión secuencial

Existe cohesión secuencial cuando el módulo representa el empaquetamiento físico de varios módulos con cohesión funcional.

Se utiliza cuando varios módulos con cohesión funcional trabajan sobre la misma estructura de datos , pero han de existir tantos puntos de entrada como número de funciones realice dicho módulo.

Cohesión comunicacional

Un módulo con cohesión comunicacional es aquel cuyos elementos o actividades utilizan los mismos datos de entrada y salida.

Es similar a la cohesión secuencial, siendo la mayor diferencia entre los dos tipos el hecho de que los módulos con cohesión secuencial realizan sus actividades en un orden específico mientras que en los módulos con cohesión comunicacional el orden en el que se ejecutan sus actividades no es relevante.

Cohesión procedural

Este tipo de cohesión se da cuando el módulo tiene una serie de elementos (funciones) relacionados por un procedimiento efectuado por el código. Cuando se alcanza una cohesión procedural se pasa a un nivel en el que los módulos son difícilmente mantenibles, por lo que la cohesión es poco aceptable.

Cohesión temporal

Un módulo con cohesión temporal es aquel cuyos elementos están implicados en actividades que están relacionadas con el tiempo. Uno de los ejemplos mas claros son los módulos de inicialización y finalización y todos aquellos que representen unas acciones que deban ejecutarse en un momento determinado del tiempo sin que sean necesarios parámetros de control.

La diferencia entre la cohesión procedural y la temporal es que el orden de ejecución de las actividades que componen el módulo es mas importante en los módulos coherentes proceduralmente.

Cohesión lógica

Un módulo tiene cohesión lógica cuando existe alguna relación entre los elementos del módulo, aunque sea débil, tal que pueda dar lugar a confusiones por no estar bien definidas las fronteras entre los diferentes elementos del módulo.

Este tipo de cohesión presenta muchos problemas:

- *Da como resultado un código confuso y por tanto difícil de entender*
- *Entrelaza diferentes funciones produciendo una complejidad mayor a la hora de realizar modificaciones*
- *Tiene una sola interfaz para varias funciones, lo que implica introducir elementos no deseados en el diseño estructurado (como, por ejemplo, parámetros de control)*
- *Los módulos son poco reutilizables en otras partes*

Son razones muy poderosas para no crear módulos con este tipo de cohesión si se quiere conseguir un buen diseño.

Cohesión coincidental

Se dice que en un módulo existe cohesión coincidental cuando entre los elementos que lo componen no existe ninguna relación con sentido.

Ello puede ser debido a tres causas:

- *Procede de un programa ya existente que simplemente se ha dividido en módulos*

- *Se crean módulos con grupos de sentencias que se repiten en un mismo módulo o en módulos diferentes. Es un intento de evitar la duplicidad de código pero no es la forma mas conveniente de hacerlo.*
- *Un programa ya creado se fragmenta debido a diversas causas, como puede ser la limitación de memoria.*

Este tipo de cohesión es muy perjudicial ya que el objetivo de la modularización es crear módulos con sentido y no simplemente romper en trozos el código.

8.6.- Diseño de pantallas

El diseño de las pantallas del sistema se ocupa de la apariencia externa que tendrá la aplicación de cara a los usuarios que la utilicen. Es la interfaz gráfica de la aplicación. Tradicionalmente se define como ***diseño de pantallas*** debido a que las aplicaciones informáticas se consideraban como una sucesión de pantallas que iban mostrando información del funcionamiento del sistema, desde que se introducen los datos por la entrada, hasta que se recogen los datos por la salida.

Una vez establecido el análisis de la aplicación e iniciado el diseño de ésta con el diseño lógico de los datos y de los procesos del sistema, parece lógico pensar que el diseño de pantallas es una transformación o consecuencia lógica del ***modelos de eventos.*** Y, así es, aunque sólo en parte. El modelo de eventos aporta la información necesaria para ***encadenar*** unas pantallas con otras. En resumen, los eventos establecen los vínculos que hacen que de una pantalla se pase a la siguiente. Es lo que se conoce como ***encadenamiento de sucesos entre pantallas***.

Pero además, el diseño de pantallas de alimenta del ***modelo de procesos*** (tanto los DFD's como de los diagramas de |estructuras de Constantine). Es en el modelo de procesos donde mas claramente se identifican los módulos encargados de realizar las entradas y salidas de datos del sistema, que es lo que, en definitiva, condiciona el diseño de las pantallas.

En definitiva, la parte funcional del sistema condiciona también su apariencia externa.

8.7.- Elementos de la interfaz gráfica de usuario (GUI)

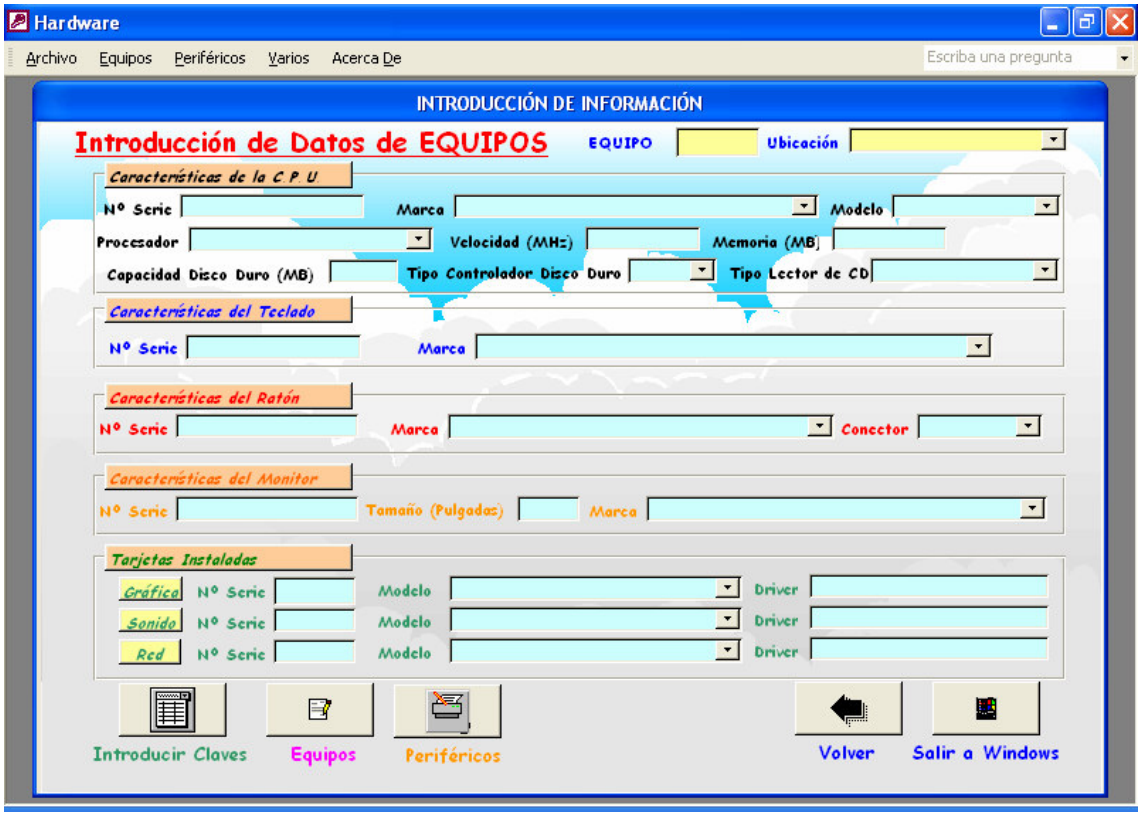
El conjunto de las pantallas del sistema constituyen lo que se llama ***interfaz gráfica de usuario (GUI)*** que es todo aquello que el usuario ve de la aplicación que utiliza, y, por tanto, la parte del sistema con la que interactúa el usuario.

Desde que se impusieron los sistemas operativos y los programas basados en ventanas (tipo "Windows"), las interfaces de la gran mayoría de las aplicaciones que se desarrollan son de tipo ***gráfico*** en contraposición con las interfaces de tipo ***texto*** que se utilizaban en sistemas operativos antiguos como MS-DOS y se utilizan mas recientemente en sistemas operativos como UNIX o Linux.

El estudio de las GUI contempla un área muy extensa dentro del diseño y la implementación de aplicaciones, tanto, que en este tema únicamente se tratarán aquellos aspectos que puedan considerarse fundamentales para completar el desarrollo de aplicaciones y, entre esta variedad de elementos que pueden conformar una GUI se centrará la atención únicamente en dos tipos de interfaces: los ***menús*** y los ***formularios***

Un ***menú*** es un conjunto de opciones o acciones que se le ofrecen al usuario para que él elija *una y sólo una de ellas.*

Un ***formulario*** es una pantalla que contiene campos de datos de distinto tipo que el usuario debe rellenar.



Ambos tipos de elementos se utilizan para que la aplicación reciba ***datos de entrada*** con los que va a trabajar. Menús y formularios corresponden, sin embargo, a distintos tipos de datos de entrada.

Los menús, en general, se utilizan para ***tipos de datos prefijados***, que son aquellos cuyo valor el usuario puede elegir entre varias opciones posibles, en contraposición con los ***tipos de datos indefinidos*** o aquellos cuyo valor puede ser introducido libremente por el usuario. Son los datos que, con frecuencia, aparecen en un formulario.

Una diferencia fundamental entre ambos tipos de datos que se pueden recibir por la entrada del sistema estriba en que *los datos prefijados no pueden presentar errores*, pues el usuario sólo elige entre las opciones que se le ofrecen, mientras que *los datos indefinidos han de pasar un estricto control* para impedir que se pueda introducir un valor indebido en el sistema.

El ***diseño de un menú*** es muy sencillo: consiste en la enumeración de las opciones entre als que el usuario puede elegir. En los entornos gráficos actuales, la opción se elige por ***selección***, pinchando con el ratón o ejecutando algún procedimiento similar para seleccionar la opción deseada.

No obstante, existe otra forma de elegir una opción: Las entradas del menú pueden *estar identificadas mediante un número o letra* que el usuario introduce desde el teclado para realizar la selección. En este caso, el dato de entrada de tipo prefijado deberá someterse al mismo control de errores que cualquier otro dato de tipo indefinido.

Este último estilo de menús tiene su origen en los primeros tipos de aplicaciones, conversacionales, pero que se ejecutaban todavía en modo texto.

En el ***diseño de un formulario*** participan, como puede observarse en el ejemplo mostrado, diferentes tipos de campos de datos. Los mas importantes son:

- **Campos de texto:** Son los recuadros donde el usuario debe teclear el valor que quiere introducir para el campo en cuestión. Por ejemplo, campos del tipo Nombre o Apellidos.

- **Botones de opción:** Permiten elegir un valor entre varios posibles para el dato en cuestión. Por ejemplo, campos del tipo Estado Civil.

- **Casillas de Verificación:** Permiten elegir uno o varios valores para campos de datos que no son excluyentes entre sí: Por ejemplo, campos del tipo Idiomas.

- **Listas desplegables:** Son menús formados por una serie de valores entre los cuales el usuario puede elegir aquel con el que va a rellenar el campo en cuestión. Por ejemplo, Categoría Profesional.

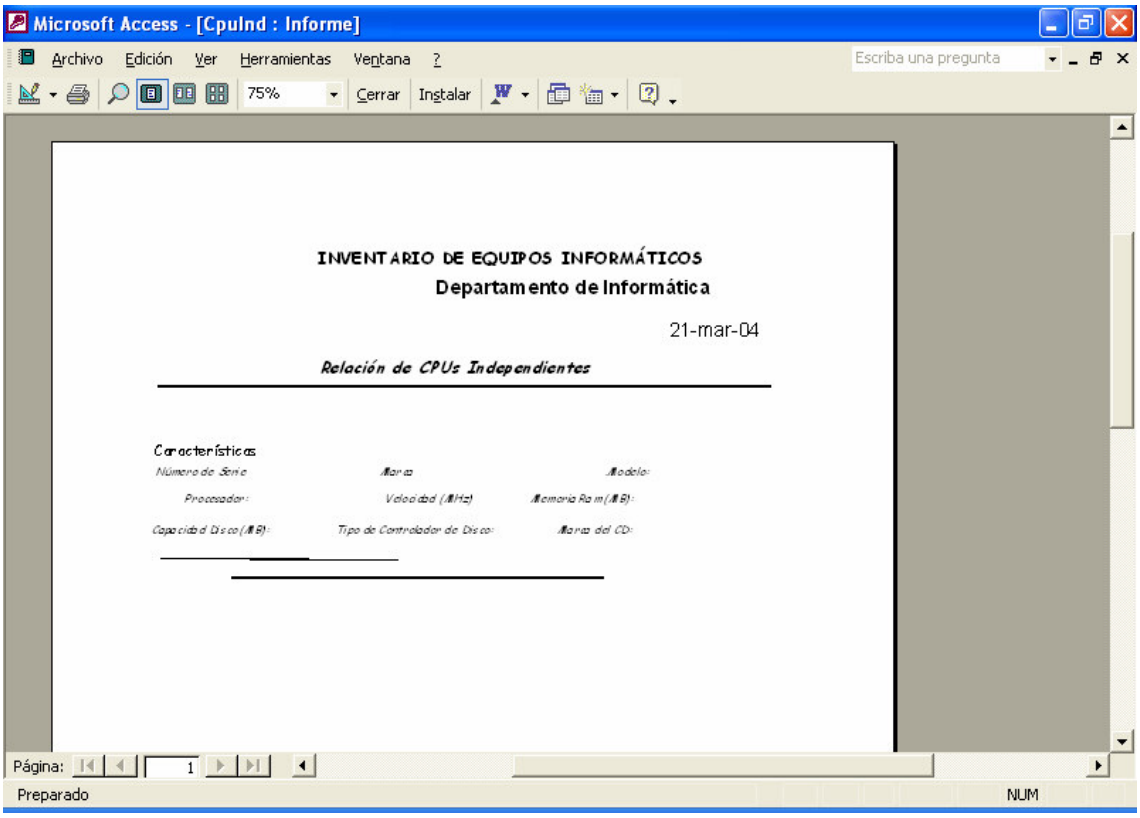
- **Botones de Ejecución de Comandos:** Permiten ejecutar ciertas operaciones sobre la pantalla diseñada. Las mas comunes son:

    - **Aceptar.-** Envía los datos recogidos por el formulario o menú al subprograma de la aplicación que lo está esperando.

    - **Cancelar.-** Aborta la ejecución de la pantalla y, por lo tanto, la correspondiente recogida de datos de entrada.

    - **Limpiar.-** Vacía todos los campos de datos del formulario o menú.

Hasta ahora se ha hablado únicamente de los elementos de la GUI relacionados con la entrada de datos. Existe, no obstante, ***elementos de la GUI*** que se encargan ***de la entrega de resultados para la salida***, como, por ejemplo, listados de datos, informes, resguardos, tickets, etc.



El diseño de estos elementos es, sin embargo, mucho mas libre y, a la vez dependiente de las características concretas de la aplicación de gestión que se desarrolla. Por este motivo su estudio se suele realizar para la aplicación concreta a desarrollar.

8.8.- Generación de la Interfaz Gráfica de Usuario (GUI)

Para diseñar las pantallas de una aplicación es preciso disponer de los modelos de eventos y de procesos ya desarrollados.

Del **modelo de procesos** se obtiene la información que permite:

- Decidir ***cuántas pantallas*** hay que construir
- Decidir ***de qué tipo*** será cada pantalla (entrada o salida de datos).
- Decidir ***qué formato*** han de tener las pantallas: menús, formularios, informes, ...
- Decidir cuál debe ser el ***diseño interno*** de cada pantalla.

Por su parte, el **modelo de eventos** aporta y confirma la información del modelo de procesos que permite ***conectar unas pantallas con otras*** de forma que, en función de los valores de los datos, las condiciones internas del sistema determinan el ***encadenamiento de pantallas*** asociado a cada ejecución de la aplicación.

El proceso a seguir para diseñar el conjunto de pantallas que forman el GUI de la aplicación es bastante libre y flexible. No obstante, se pueden sugerir una serie de pasos a seguir con objeto de no olvidar ningún aspecto importante que pueda influir en la interfaz resultante.

8.8.1.- Identificación de los procesos de entrada y de salida del sistema

El paso inicial consiste en estudiar el diseño de procesos con el fin de localizar todos aquellos procesos que tienen contacto directo con el usuario, ya sea porque leen datos de entrada procedentes de éste, ya porque entregan resultados de salida.

Para cada ***proceso externo*** (que así se llaman los que tratan directamente con el usuario) hay que decidir:

- Tipo de función que realiza: entrada o salida de datos
- Tipo de pantalla que genera: menú, formulario, informe, etc.
- Conjunto de datos que recibe o entrega al usuario

8.8.2.- Diseño de las pantallas de entrada y salida de datos

Con el trabajo realizado en el punto anterior se dispone de la información necesaria para diseñar cada pantalla identificada.

Este punto es el que mas libertad ofrece al responsable de realizar esta tarea, pues los aspectos estéticos de las pantallas no suelen estar sujetos a restricciones ni requisitos específicos. No obstante, es conveniente que todas las pantallas cumplan una serie de propiedades mínimas para el buen funcionamiento de la interfaz de la aplicación:

- **Ser mínima y completa**: Una pantalla debe tener todos los datos que le corresponde tratar, pero sólo éstos. El resto de información que pudiera añadirse no hará sino complicar el diseño y desviar la atención del usuario de lo fundamental.
- **Ser clara y concisa**: Una pantalla debe explicar el significado de toda la información que muestra de una manera breve, sin ambigüedades y que no pueda ser malinterpretada por parte del usuario.
- **Ser sencilla y atractiva**: Desde el punto de vista estético, deben hacerse las pantallas lo mas centradas en los datos que se pueda. En este sentido, la elección de colores, formatos, dibujos, gráficos y demás elementos que adornan una GUI deben estar siempre al servicio de llamar la atención del usuario, pero para centrarla en el contenido esencial de cada ventana.
- **Ser coherente**: Cada pantalla debe guardar, en relación con el resto de la GUI, una proporcionalidad en el contenido y en la forma que permita al usuario seguir claramente la ejecución del sistema.

8.8.3.- Comprobación de las condiciones que controlan a las pantallas diseñadas

Cuando un proceso realiza operaciones de entrada o de salida de datos, generalmente asocia a éstas, comprobaciones de estado o de contenido de los datos:

- Las comprobaciones de estado permiten al proceso averiguar si la operación se ha realizado con normalidad o si se ha visto alterada por alguna anomalía de funcionamiento.
- Las comprobaciones de contenido incluyen el control de errores asociado a la validez de los valores de los datos, pero también ofrecen, a determinadas condiciones internas del sistema, la información necesaria para tomar una decisión de tipo alternativo - estructura de bifurcación - o repetitivo - estructura de bucle-.

8.8.4.- Conexión de unas pantallas con otras

Por último, para lograr que las pantallas del sistema formen una interfaz, es decir, que funcionen como un conjunto, se estudian las condiciones de entrada o de salida de las mismas.

Este trabajo se realiza inspeccionando simultáneamente los modelos de eventos y de procesos, para poder establecer las relaciones de precedencia que ponen en relación unas pantallas con otras dentro del conjunto (GUI) que forman.

8.8.5.- Documentación del diseño de pantallas

1°.- Pantallas que forman la GUI de la aplicación:

.- Tipo de pantalla

#.- Formulario

#.- Menú

#.- Informe

.-Descripción de elementos que forman la pantalla.

.- Procesos a los que está vinculada la pantalla

.- Eventos a los que está vinculada la pantalla

.- Encadenamiento dentro del sistema

#.- Pantalla anterior

#.- Pantalla posterior

#.- Condiciones de salto

2° Alternativas de diseño para las pantallas del sistema

#.- Refinamientos y mejoras

#.- Planteamientos de ampliación futura

8.9.- Ergonomía del diseño de la interfaz

Como ya se ha indicado en reiteradas ocasiones, el sistema será mas fácil de utilizar si lo diseñara el usuario, por lo que el diseño del interfaz debe estimular a éste haciéndolo cómplice del sistema. Es decir, el diseño de pantallas debe comenzar con una discusión con el usuario, la parte mas importante de cualquier sistema informatizado.

Para realizar un buen trabajo, la interfaz con el ordenador debe ser, entre otras cosas, lo mas fácil, amigable y agradable posible, se debe usar un diálogo que se acerque al lenguaje natural en vez de a la jerga informática. Entre las consideraciones a tener en cuenta a la hora de diseñar pantallas se encuentran:

Características deseadas: simplicidad, claridad y fácil de comprender.

Situación de la información en la pantalla, indicando el punto de partida obvio en la esquina superior izquierda de la pantalla, reservando áreas específicas de la pantalla para diferentes tipos de información, manteniendo la consistencia en el diseño de todas las pantallas y proporcionando una composición que guste visualmente

Determinación de la información a situar en la pantalla: En cuanto a lasa fuentes de letras se recomienda utilizar minúsculas para el texto, con la letra inicial de la

frase en mayúsculas; para etiquetas, encabezamientos o subtítulos, utilizar mayúsculas. En cuanto a palabras se recomienda no utilizar ningún tipo de jerga, utilizando mas bien palabras cortas, familiares, etc.

Recogida de todos los datos necesarios en la interfaz de entrada de manera que no se introduzcan errores para el sistema. La interfaz entonces contendrá una protección contra errores de entrada. Debe recoger los datos minimizando el número de teclas utilizadas por el usuario. Las entradas deben estar bien estructuradas y ser fáciles de comprender y de utilizar. Se deben utilizar nombres precisos y permitir abreviaturas cuando se necesiten introducciones rápidas de datos. Deben evitarse las entradas repetitivas. El diseño de salida debe asegurar que se extraen todos los datos suministrados por el sistema y que esas salidas están estructuradas de forma que sean fáciles de leer.

Utilización del color en el diseño de la pantalla ya que atrae la atención del usuario. Si se utiliza de forma adecuada puede resaltar la organización lógica de la pantalla, facilitar la separación entre los componentes y acentuar las diferencias.

Por el contrario, si se utiliza inadecuadamente, puede distraer y fatigar la visión debilitando la facilidad de uso del sistema (En pantallas gráficas se recomienda no utilizar mas de seis colores a la vez, evitar los colores extremos así como colores que no tengan contraste.

Por último debe tenerse en cuenta que el diseño de pantallas es un proceso ordenado que se inicia en los requisitos y finaliza con la implementación.