



CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

www.cef.es

info@cef.es

Índice Tema 10

Introducción.

1. El diseño de sistemas de información. Visión general.
2. El diseño de la arquitectura del sistema
 - 2.1. Tareas del diseño de la arquitectura del sistema.
3. Técnicas de diseño de procesos. El diseño estructurado.
 - 3.1. Objetivos y características del diseño estructurado.
 - 3.2. El diagrama de estructuras.
 - 3.2.1. Módulos.
 - 3.2.2. Conexiones entre Módulos.
 - 3.2.3. Comunicación entre Módulos.
 - 3.3. Principios del diseño estructurado.
 - 3.4. Un ejemplo de diagrama de estructura de cuadros.
4. Estrategias de diseño.
 - 4.1. Análisis de transformación.
 - 4.2. Análisis de transacción.
5. Evaluación de la calidad del diseño.
 - 5.1. Acoplamiento.
 - 5.2. Cohesión.
 - 5.3. Otros parámetros de calidad: el fan-in y el fan-out.
6. El lenguaje de diseño de programas (PDL).



CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

www.cef.es

info@cef.es

TEMA 10

Diseño de programas. Diagramas estructurados. Análisis de transformación. Análisis de transacción. PDL (lenguaje de definición de programas).

INTRODUCCIÓN.

Para llevar a cabo el proceso de desarrollo de un sistema de información, con independencia del área de aplicación, el tamaño del proyecto, su complejidad y el modelo de ciclo de vida elegido, hay que pasar por una serie de fases. Primeramente, durante el Análisis, se estudia y analiza el dominio, el problema y las necesidades del usuario en base a los requisitos planteados y se presenta un modelo de los objetivos a lograr. Es una fase orientada al usuario, o, si se quiere, al problema, e independiente del entorno en que se vaya a implementar el sistema.

De aquí, se pasa al Diseño. El modelo conceptual que recoge la necesidad del usuario se convierte en un conjunto de modelos formales, que en refinamientos sucesivos llegarán a ser modelos computables por un ordenador. Es una fase orientada a la máquina, cuyo objetivo es determinar una vez comprendido el problema, cómo puede ser éste tratado por un ordenador. Durante el Diseño se deberá especificar cómo han de diseñarse las estructuras de datos y la arquitectura del sistema, y cómo han de implementarse los detalles procedimentales.

Finalmente, se traslada el diseño a un lenguaje de programación, esto es, se genera el código, y una vez efectuadas las pruebas, se instala el sistema y se pone en explotación.

El desarrollo tradicional o estructurado de un Sistema de Información, a diferencia del desarrollo orientado a objetos, mantiene una clara separación entre los datos y las funciones o procesos del sistema.

1. Durante el Análisis de Procesos, se lleva a cabo el modelado conceptual de procesos, cuyo objetivo es captar y representar el dominio funcional del sistema, sin considerar cómo se va a implementar éste, y para lo cual se utiliza la técnica formal conocida como Diagrama de Flujo de Datos (DFD).



2. En el proceso de Diseño, durante el Diseño de Procesos, cuyo estudio es el objetivo de este tema, se trasladan los requerimientos funcionales del sistema a una representación que describa la arquitectura del mismo y el procedimiento algorítmico, utilizando para ello la técnica conocida como Diagrama de Estructura de Cuadros (DEC).

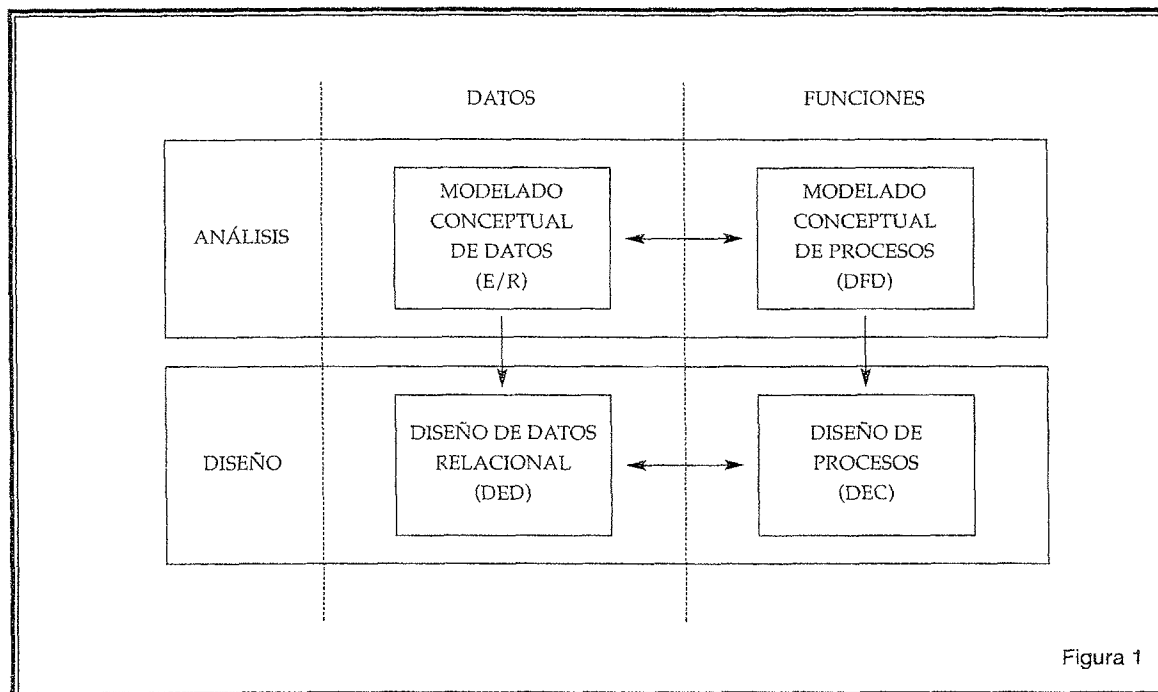


Figura 1

A fin de desarrollar este tema, comenzaremos primeramente dando una visión general del proceso de Diseño, distinguiendo entre el diseño de la Arquitectura del Sistema y el Diseño del Software propiamente dicho. A estos efectos, se aportarán una serie de definiciones y distintos puntos de vista a la hora de conceptualizar el proceso de Diseño de un Sistema de Información.

El objetivo siguiente será estudiar el primer paso del Diseño, esto es, el Diseño de la Arquitectura del Sistema. En este punto se hará referencia a los distintos tipos de arquitecturas que se pueden considerar y a las tareas que comprende el diseño de la Arquitectura del Sistema.

Seguidamente nos centraremos en el Diseño del Software y, en concreto, en el diseño orientado a los procesos. Se estudiará con detalle el diseño estructurado, sus objetivos, principios y características, y la técnica del Diagrama de Estructura de Cuadros. Asimismo, se analizarán las dos principales estrategias de diseño estructurado, esto es, el Análisis de Transformación y el Análisis de Transacción, y dada la criticidad de la fase de Diseño a efectos de garantizar la calidad del software, se estudiarán los parámetros esenciales a considerar en el diseño estructurado, a fin de garantizar un mínimo de calidad (acoplamiento, cohesión, fan-in y fan.out).

Por último, se hará referencia al lenguaje de diseño de programas PDL, como elemento esencial del diseño procedimental.

1. EL DISEÑO DE SISTEMAS DE INFORMACIÓN. VISIÓN GENERAL.

En términos generales, el diseño se puede definir como: «El proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, proceso o sistema, con los suficientes detalles como para permitir su implementación física». El proceso de Diseño del Sistema de Información es el primero, dentro del proceso de desarrollo, en el que el trabajo se orienta a la máquina, y según señala la metodología Métrica v.3, tiene por objeto la definición de la arquitectura del sistema y del entorno tecnológico que le va a dar soporte, así como la especificación detallada de los componentes del sistema de información.

En consonancia con lo anterior, las actividades que comprende el Diseño del Sistema de Información se pueden agrupar de la siguiente manera:

- Un primer bloque de actividades, que se realizarán independientemente del paradigma que se vaya a seguir en el desarrollo (estructurado u orientado a objetos), cuyo objetivo es diseñar la Arquitectura del Sistema. Para ello:
 - Se establece el particionamiento físico del sistema, su organización en subsistemas de diseño, la especificación del entorno tecnológico y sus requisitos de operación, administración, seguridad y control de acceso.
 - Se obtiene el diseño detallado de los subsistemas de soporte, el establecimiento de las normas y requisitos propios del diseño y construcción, y la identificación y definición de los mecanismos genéricos de diseño y construcción.
- Un segundo bloque de actividades cuyo objetivo es el Diseño del Software, que transforman el «qué hacer» de las especificaciones de Análisis en el «cómo hacerlo» de las especificaciones de Diseño. Dependiendo de que el desarrollo se aborde de forma estructurada o mediante orientación a objetos las actividades y técnicas utilizadas serán diferentes, pero, en cualquier caso, se deberá:
 - Realizar el diseño de detalle de los subsistemas específicos del sistema de información y la revisión de la interfaz de usuario.
 - Diseñar y optimizar las estructuras de datos del sistema, asignándolas a los nodos de la arquitectura propuesta.
 - Revisar y validar el diseño de detalle, a fin de analizar la consistencia entre los distintos modelos y conseguir la aceptación del diseño por los responsables correspondientes.
- Por último, un conjunto de actividades que complementan el diseño del sistema de información, y cuyo objetivo fundamental es generar todas las especificaciones necesarias para la construcción del sistema.

El Diseño del Sistema Software pretende desarrollar una representación coherente y bien organizada del software, que satisfaga la especificación de requisitos que el cliente ha aceptado como objetivo a lograr por el proyecto. Según el «Glosario de Términos de Calidad e Ingeniería del Software» de la Asociación Española para la Calidad (AECC), el diseño se define como «el proceso de definición de la arquitectura software, los componentes, los módulos, las interfaces, los procedimientos de prueba y los datos de un sistema software para satisfacer unos requisitos especificados»; y comprende dos partes:

1. El diseño externo, que comienza a realizarse en la fase de análisis, y se encarga de todas las relaciones con el exterior que tendría el sistema, esto es, de las definiciones de pantallas, los formatos de informes, las definiciones de entradas y salidas, etc.
2. El diseño interno, que incluye la concepción, planificación y especificación de la estructura interna y de los detalles de proceso del producto software.

En general, el proceso de diseño interno suele ser complejo, de ahí que suele realizarse en, al menos, dos fases consecutivas:

- a) El diseño arquitectónico, o diseño de alto nivel, que se centra en las funciones y en la estructura de los componentes software que comprenderán el sistema. En esta línea, el diseño arquitectónico identifica las funciones internas del proceso, descompone las funciones de alto nivel en subfunciones, identifica las interconexiones entre las funciones, los datos y su almacenamiento, etc.

El «Glosario de Términos de Calidad e Ingeniería del Software», antes citado, define el diseño arquitectónico como: «El proceso de definición de una colección de componentes y sus interfaces para establecer la estructura del desarrollo de un sistema».

- b) El diseño detallado, o diseño de bajo nivel, que se deriva del diseño arquitectónico y se centra en la especificación de algoritmos, la implementación de funciones las estructuras de datos específicas que implementan el almacenamiento de los mismos, y en las interacciones entre datos y funciones.

El «Glosario» ya mencionado define el diseño detallado como: «El proceso de refinar y expandir el diseño preliminar para que contenga descripciones más detalladas de la lógica del proceso, estructuras de datos, y definiciones de datos».

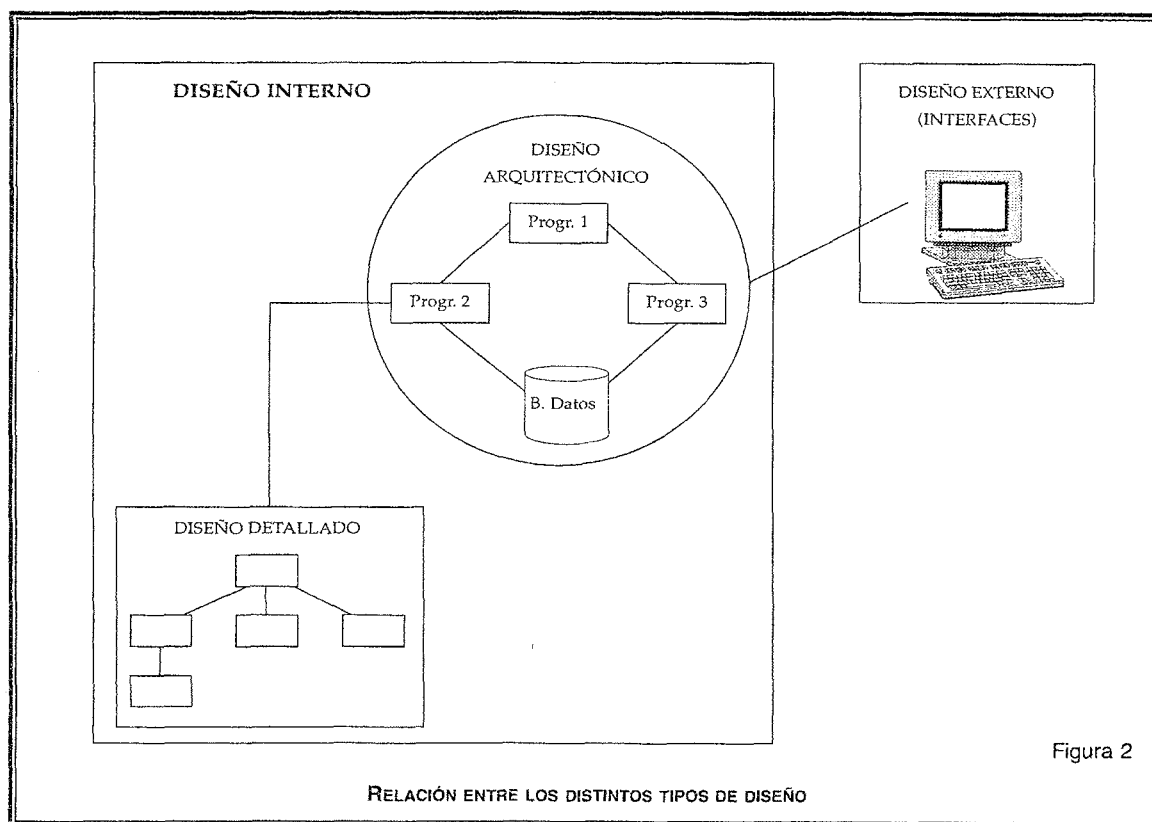


Figura 2

En definitiva, el conjunto de actividades incluidas en todo el proceso de diseño del software, desde el comienzo del nivel de arquitectura hasta el final del diseño detallado, son las siguientes, algunas de las cuales pueden realizarse en paralelo:

1. Realizar el diseño arquitectónico. Durante esta actividad se determinan los componentes de software que constituyen el sistema y la estructura mediante la que se unen, se especifican los datos, relaciones entre ellos y restricciones que les afectan, y se definen las interfaces externas e internas.
2. Analizar el flujo de información (diseño orientado a los procesos). Su objetivo es jerarquizar la información a manejar por el sistema, definir los datos asociados, y cómo afecta el flujo de información a los componentes software y a la estructura del sistema.
3. Diseñar la base de datos (diseño orientado a los datos). Es decir, crear una estructura para la información. El diseño de la base de datos engloba tres pasos separados, pero dependientes: el diseño conceptual, el diseño lógico y el diseño físico.
4. Diseñar las interfaces de usuario, de software y de hardware.
5. Seleccionar o desarrollar algoritmos. Esto es, seleccionar o desarrollar una representación procedimental de las funciones especificadas para cada componente software y para la estructura de datos.
6. Realizar el diseño detallado. Se escogen las alternativas de diseño para implementar las funciones especificadas para cada componente software. Al final de esta actividad quedarán especificadas las estructuras de datos, los algoritmos y los componentes detallados a programar.

A tenor de las actividades mencionadas, podemos considerar, siguiendo a prestigiosos autores en la materia, que el diseño de un sistema software comprende: el diseño de datos, el diseño arquitectónico y el diseño procedimental.

- A) El diseño de datos se enfoca sobre la definición de la estructura de los datos. Es la primera y más importante de las actividades de diseño realizadas durante el proceso de la Ingeniería del Software, y consiste en seleccionar las representaciones lógicas de los objetos de datos (estructuras de datos) identificadas durante la fase de definición y análisis de requerimientos.
- B) El diseño arquitectónico define las relaciones entre los principales elementos estructurales del programa. Su objetivo es desarrollar una estructura de programa modular y representar las relaciones de control entre los módulos. El diseño arquitectónico mezcla la estructura de programas y la estructura de datos, y define las interfaces que facilitan el flujo de datos a lo largo del programa.
- C) El diseño procedimental transforma los elementos estructurales en una descripción procedimental del software. Se realiza después de que se ha establecido la estructura del programa y la de los datos, y consiste en definir los detalles de los procedimientos que han de llevarse a cabo. Para ello, como no sería conveniente usar el lenguaje natural, se recurre a métodos como la programación estructurada, es decir, al uso de construcciones lógicas con las que puede formarse cualquier programa.

Por último, señalar que durante el diseño es donde se asienta principalmente la calidad del software. A estos efectos, los principales criterios de calidad del diseño son:

- Un diseño debe exhibir una organización jerárquica que haga uso del control entre los elementos del software.
- Un diseño debe ser modular, es decir, el software debe estar particionado en elementos que realicen funciones y subfunciones específicas.
- Un diseño debe contener una representación distinta y separable de los datos y los procedimientos.
- Un diseño debe conducir a módulos que exhiban características independientes (subrutinas, procedimientos, etc.).

2. EL DISEÑO DE LA ARQUITECTURA DEL SISTEMA.

El diseño de la Arquitectura del Sistema es la primera fase del Diseño de un Sistema de Información, en la cual se selecciona la aproximación básica para resolver el problema y construir una solución. Tal como señala la metodología Métrica v.3., durante el diseño de la Arquitectura del Sistema es necesario establecer el particionamiento físico del sistema, su organización en subsistemas de diseño, la especificación del entorno tecnológico y sus requisitos de operación, administración, seguridad y control de acceso.

La arquitectura proporciona el contexto en el cual se toman decisiones más detalladas en una fase posterior del diseño, e incluye decidir la estructura y el estilo global del sistema, organizar éste en subsistemas, la asignación de subsistemas a componentes hardware y software, y decisiones fundamentales conceptuales y de política que son las que constituyen un marco de trabajo para el diseño detallado.

Existe un cierto número de estilos frecuentes de arquitectura, cada uno de los cuales es adecuado para ciertas clases de aplicaciones. Algunas de las arquitecturas de sistemas más comunes son:

- Arquitectura centrada en los datos. Sistemas en los cuales existe un constante acceso y actualización de un almacén de datos que es utilizado masivamente.
- Arquitectura de flujos de datos. Sistemas que se ven como una serie de transformaciones de trozos sucesivos de datos de entrada. Los datos entran en el sistema y fluyen a través de los componentes uno a uno hasta que se asignan a un destino final (salida o almacén de datos).
- Arquitectura de máquinas virtuales. Una máquina virtual es un estilo de software que simula alguna funcionalidad que no posee el hardware y/o el software en el cual se va a implementar.
- Arquitectura de llamada y retorno. Es la típica arquitectura de programa principal y llamada a subrutina o a procedimiento remoto.
- Arquitectura de componentes independientes. Consta de una serie de procesos independientes que se comunican a través de mensajes.

Puesto que en el diseño se asienta la calidad del software, es importante comenzar eligiendo la arquitectura del sistema (estilo arquitectónico) más adecuada.

Hay distintos patrones para conseguir los atributos de calidad necesarios: patrones del sistema (estilos arquitectónicos) patrones de diseño y patrones de código.

Un patrón de sistema o estilo arquitectónico se determina por:

- Un conjunto de tipos de componentes que realizan una determinada función.
- La topología de dichos componentes.
- Un conjunto de restricciones semánticas.
- Un conjunto de conectores que median la comunicación, coordinación y cooperación entre los componentes.

Algunas reglas para determinar el estilo arquitectónico son las siguientes:

- Flujo de datos. Tiene sentido cuando el sistema produce una salida bien definida y fácilmente identificable que es a su vez el resultado directo de una transformación secuencial de una entrada bien definida y fácilmente identificable.
- Llamada y retorno. El orden de computación es fijo y los componente no pueden progresar hasta que no reciben los resultados de otros componentes.
- Componentes independientes. Tiene sentido cuando el sistema se vaya a ejecutar en una plataforma multiprocesador. El sistema puede estructurarse como un conjunto de componentes poco acoplados. El ajuste del rendimiento en este estilo es esencial.
- Centrada en los datos. Cuando el almacenamiento de los datos es clave. Se centra en la representación, gestión y recuperación de un gran volumen de información que sobrevivirá durante largo tiempo.

2.1. TAREAS DEL DISEÑO DE LA ARQUITECTURA DEL SISTEMA.

Durante el Diseño de la Arquitectura del Sistema, una vez elegido el estilo arquitectónico más adecuado, se deben realizar las siguientes tareas:

1. Organizar el sistema en subsistemas.
2. Identificar la concurrencia inherente al problema.
3. Asignar los subsistemas a los procesadores y tareas.
4. Seleccionar una aproximación para la administración de almacenes de datos.
5. Manejar el acceso a recursos globales.
6. Seleccionar la implementación de control en software.
7. Manejar las condiciones de contorno.

1. DESCOMPOSICIÓN DEL SISTEMA EN SUBSISTEMAS.

Un subsistema es un conjunto de clases (o entidades), asociaciones y operaciones que están interrelacionadas y tienen una interface bien definida con el resto de los subsistemas. Normalmente se identifican por los servicios que proporcionan, siendo un servicio un conjunto de funciones de propósito común. Los subsistemas deben ser implementados de forma que existan pocas interacciones entre ellos, siendo mejor que éstas permanezcan lo más posible en el mismo subsistema.

Las relaciones entre dos subsistemas pueden ser:

- Cliente-servidor. El cliente es quien establece la comunicación y, por tanto, debe conocer la interface con su servidor.
- De igual a igual. Cada subsistema puede iniciar la comunicación con el resto, con lo que las interacciones son más complicadas.

La descomposición de un sistema en subsistemas puede organizarse:

- Mediante una secuencia de niveles horizontales. Cada nivel define un mundo abstracto diferente construido en base a los niveles inferiores, definiendo así una relación cliente-servidor. Este tipo de arquitectura puede ser abierta, si un nivel puede usar características de cualquier nivel inferior, o cerrada, si sólo utiliza las del nivel inmediatamente inferior. La arquitectura cerrada es más robusta y más fácilmente mantenible.
- Mediante particiones verticales. Cada subsistema es independiente de los demás y realiza un servicio específico. Generalmente, las relaciones son de igual a igual.

La mayoría de los sistemas contienen tanto niveles horizontales como particiones verticales.

2. IDENTIFICACIÓN DE LA CONCURRENCIA DE LOS SUBSISTEMAS Y LOS OBJETOS.

Aunque conceptualmente todos los objetos son concurrentes, es decir, pueden actuar a la vez, en la práctica muchos objetos de un sistema son interdependientes, esto es, el comportamiento de uno de ellos puede afectar al comportamiento del resto. Dicho de otra forma, en una implementación no todos los objetos del software son concurrentes, porque un procesador puede dar soporte a muchos objetos. En la práctica, se pueden implementar muchos objetos en un único procesador si los objetos no pueden estar activados a la vez. Un objetivo importante del diseño del sistema es identificar los objetos que deben estar activados concurrentemente, y los objetos que tienen actividad que sea mutuamente exclusiva. Estos últimos objetos se pueden plegar y juntar en un único hilo de control o tarea.

3. ASIGNACIÓN DE LOS SUBSISTEMAS A PROCESADORES Y TAREAS.

Para llevar a cabo esta función hay que definir los procesadores que se van a emplear en el sistema y su tipo, y a continuación, las conexiones entre ellos. Los factores a considerar para determinar el número de procesadores a utilizar son, entre otros:

- La decisión de usar sistemas multiprocesadores (compuestos por varias CPU) o unidades funcionales hardware (compuestos por una única CPU), en función de las necesidades de rendimiento que se tengan.

- Que cada subsistema concurrente debe ser asignado a una unidad hardware distinta.
- La posibilidad de tener que asignar funciones de los diversos subsistemas software a procesadores (CPU) distintos, ya sea por la necesidad de localizaciones específicas, o para evitar que un solo procesador trate un excesivo volumen de datos.

Una vez definido el número y el tipo de las unidades físicas (procesadores) a emplear, se determinarán las conexiones entre ellas, la topología de dichas conexiones, la forma de los canales de conexión y los protocolos de comunicación.

4. DETERMINAR LA GESTIÓN DE LOS DATOS.

Los almacenes de datos pueden separar subsistemas en una arquitectura. Cada almacén puede combinar estructuras de datos, ficheros y bases de datos, implementados en memoria o en dispositivos de almacenamiento secundario.

Diferentes tipos de almacenes (ficheros convencionales, bases de datos, etc.) proporcionan distintas combinaciones de coste, tiempo de acceso, capacidad y fiabilidad, por lo cual, para tomar la mejor decisión sobre la gestión de los datos que debe mantener el sistema, se deberá alcanzar un equilibrio entre los parámetros anteriores.

5. CONTROLAR LOS RECURSOS GLOBALES.

Los recursos globales están constituidos tanto por las unidades físicas (procesadores, unidades de disco, etc.) como por las entidades lógicas (ficheros, nombres de clases, etc.). Entre los mecanismos de control más utilizados para controlar el acceso a los recursos destacan:

- La asignación a cada recurso, o conjunto de recursos, de un objeto guardián que controla el acceso.
- La división del recurso en partes independientes controladas cada una, bien por un objeto guardián, o bien por un objeto llave, en cuyo caso el acceso es directo.

6. ELEGIR LA IMPLEMENTACIÓN DE CONTROL.

En un sistema existen dos tipos de flujos de control: el interno, que es el flujo de control dentro de un proceso, y el externo, que es el flujo de control entre los objetos de un sistema. Atendiendo a la diferente implementación del control externo los sistemas pueden ser de los siguientes tipos:

- Sistemas secuenciales accionados por procedimientos.
- Sistemas secuenciales accionados por eventos.
- Sistemas concurrentes.

La elección de uno u otro tipo dependerá, fundamentalmente, del lenguaje en el que se vaya a realizar la implementación.

7. ESTABLECIMIENTO DE LAS CONDICIONES LÍMITE.

Aunque la mayoría del esfuerzo durante el diseño se concentra en el comportamiento del sistema en el estado estable, es decir, durante el funcionamiento normal de la aplicación, hay que considerar también las condiciones límite de inicialización, terminación y fallos.

- Entre los objetos a inicializar están las constantes, los parámetros, las tareas, los objetos guardianes, etc.
- Respecto a la terminación, hay que tener en cuenta que las tareas deben liberar cualquier recurso externo que tuvieran reservado. En los sistemas concurrentes una tarea debe comunicar a otras su terminación.
- Finalmente, ante la aparición de fallos, hay que abandonar el entorno obteniendo la mayor información posible sobre la causa del error.

3. TÉCNICAS DE DISEÑO DE PROCESOS. EL DISEÑO ESTRUCTURADO.

Existen varias técnicas de diseño orientadas a los procesos. Entre ellas la más utilizada es el «diseño estructurado», que estudiaremos a continuación. No obstante existen otras, como HIPO (Hierarchical Input Process Output), diseño lógico del proceso, etc., que se utilizan en metodologías particulares, bien a nivel de empresa, o a nivel de organismos públicos.

Las técnicas de diseño orientadas a los procesos han ido evolucionando a lo largo del tiempo. Una de las más antiguas es el organigrama o diagrama de flujo; sin embargo, el hecho de que no manejara la organización de los sistemas hizo que no sobreviviese a los años 60, utilizándose hoy día sólo a nivel de programa unitario.

Tras ella surgió la programación estructurada, que ya trata la complejidad de los sistemas y se basa en una filosofía «top-down». El diseño top-down divide los problemas en otros más pequeños, pero desafortunadamente no da ideas de cómo realizar esta partición.

A la técnica «top-down» le sucedió el diseño estructurado, que trata de completar las deficiencias anteriores y que fue propuesto por Yourdon y Constantine en 1978. Esta será la técnica en que nos centremos.

3.1. OBJETIVOS Y CARACTERÍSTICAS DEL DISEÑO ESTRUCTURADO.

El Diseño Estructurado produce sistemas fáciles de entender, fiables, flexibles, duraderos, de desarrollo rápido y eficientes. En definitiva, produce sistemas a bajo coste que funcionan. Los objetivos del Diseño Estructurado son tres básicamente:

1. Obtener la estructura modular y los detalles de proceso del sistema, partiendo sólo de los productos obtenidos en la fase de Análisis del Sistema (pasar del qué al cómo).

2. Obtener un diseño que además de funcionar, sea mantenible, mejore la reutilización y se pueda probar y entender fácilmente.
3. Utilizar herramientas gráficas (Diagrama de Estructura de Cuadros) para representar la estructura modular del sistema.

A tenor de los objetivos citados, las características más destacables del Diseño Estructurado son:

- Permite obtener la solución a partir de la definición del problema, puesto que la solución tiene los mismos componentes e interrelaciones entre ellos que el problema original.
- Simplifica el problema, ya que lo subdivide creando una división del mismo en cajas negras organizadas en jerarquías.
- Utiliza técnicas gráficas, como, por ejemplo, el diagrama de estructuras, que hacen que el sistema sea mucho más comprensible.
- Ofrece un conjunto de estrategias (p. ej., el análisis de transformación y el análisis de transacción) para desarrollar una solución de diseño a partir de una definición o especificación completa del problema.
- Proporciona criterios para evaluar la calidad de una solución dada con respecto al problema a resolver (p. ej., el grado de acoplamiento y la cohesión).

3.2. EL DIAGRAMA DE ESTRUCTURAS.

Aunque existen varias técnicas dentro del Diseño Estructurado, por ejemplo, las Tablas de Interfaz, la más importante, sin duda, es el Diagrama de Estructuras (Structure chart), también denominada Diagrama de Estructura de Cuadros (DEC).

Finalizada la fase de Análisis del Sistema, al iniciar la de Diseño se dispondrá de un conjunto de especificaciones funcionales que describan: las entradas que suministran al sistema las entidades externas y las salidas por éste suministradas (Diagrama de Contexto); las funciones descompuestas que se han de realizar por dicho sistema (Diagrama de Flujo de Datos); y el esquema lógico de datos.

Para construir el sistema es preciso convertir toda esa información en especificaciones de programas. A ese respecto, las tareas a realizar son:

1. Determinar qué módulos implantarán los procesos obtenidos en la fase de Análisis del Sistema.
2. Organizar la estructura de dichos módulos y definir las conexiones entre los mismos.
3. Describir el pseudocódigo o lenguaje de definición de programas para cada módulo.

Para ello se seguirá la Técnica del Diagrama de Estructura de Cuadros (DEC), que será más fácil de instrumentar cuanto mayor nivel de detalle tengan los DFD obtenidos en la fase de Análisis.

Un Diagrama de Estructura de Cuadros está constituido por los siguientes elementos principales:

- Módulos.
- Conexiones entre módulos.
- Comunicación entre módulos.

Seguidamente se estudia cada uno de ellos.

3.2.1. Módulos.

Puesto que la arquitectura implica modularidad, el software debe dividirse en elementos, llamados módulos, que se integran entre sí de forma que con su ejecución satisfagan los requisitos del sistema. Un módulo es una unidad claramente definida y manejable, con interfaces perfectamente definidas. En otras palabras, es la unidad mínima con sentido propio en un sistema software.

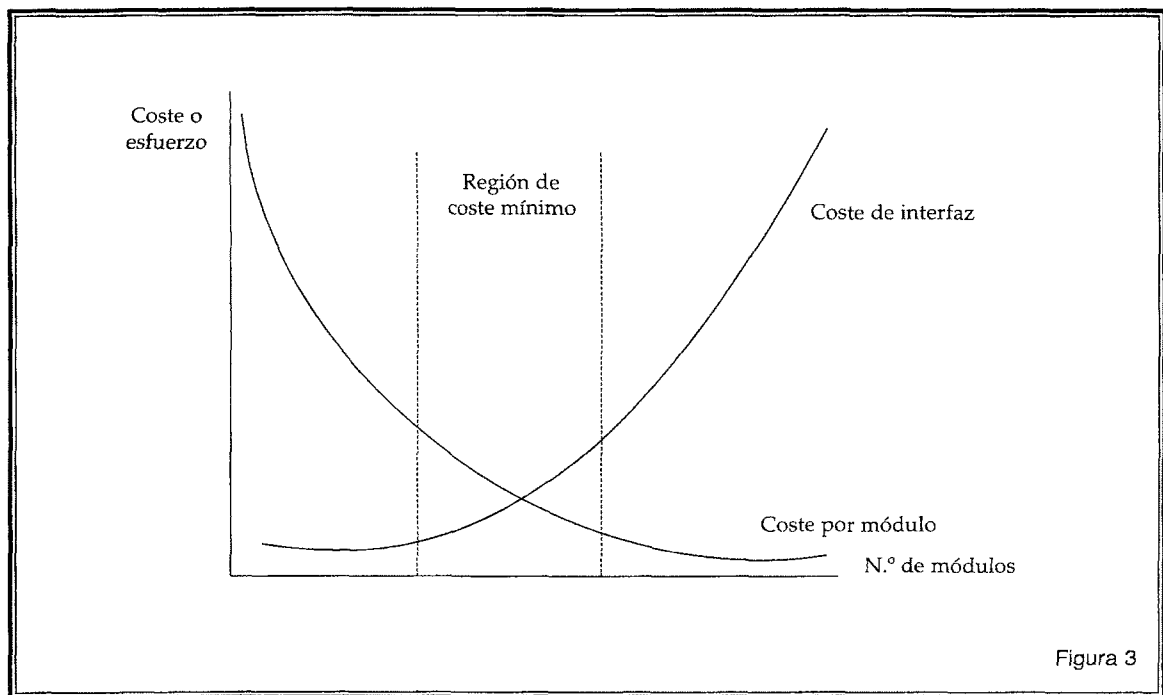
Entre las muchas definiciones que se han dado del término «módulo», destacamos las siguientes:

- Módulo es la parte lógica separable de un programa.
- Módulo es una secuencia contigua de sentencias de programa, limitada por delimitadores, y que tiene un identificador global.
- Módulo es la unidad más pequeña de código que puede ser compilada independientemente.
- Módulo es un programa, subprograma o rutina, que no tiene porqué ser compilado independientemente, y que admite parámetros de llamada y retorna a algún valor, si es preciso.
- Módulo es aquella parte de código que se puede llamar.

El diseño estructurado no impone la restricción de que el módulo tenga que ser compilado independientemente, pero sí, que se cumplan las siguientes características:

1. Módulos de pequeño tamaño, ya que cuantas menos líneas de código tengan, más fácil resulta la mantenibilidad del sistema, pues aunque en el caso de una modificación ésta afectará a más módulos, sin embargo la cantidad de código a considerar será menor.

Respecto a lo pequeño que debe ser el módulo, algunos autores consideran que lo ideal es que tengan alrededor de 50 líneas de código, mientras que otros hablan de 400 o 500, una página, etc. En cualquier caso, es importante destacar que a medida que crece el número de módulos, crece el coste por interfaz, si bien decrece el esfuerzo para desarrollar cada módulo; por tanto, el diseñador debe conseguir estructurar el sistema en un número de módulos tal, que el coste del sistema se mantenga en la región de coste mínimo.

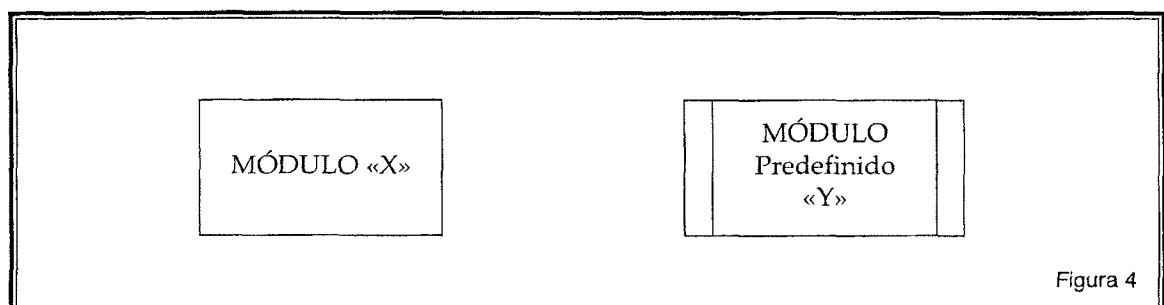


2. Independencia modular, ya que cuanto más independiente sea el módulo, más sencillo es trabajar con él. (El diseño debe reducir la compartición de ficheros, datos, dispositivos, etc.).
3. Caja negra, es decir, dar una visión exclusiva de sus entradas y salidas sin tener en cuenta los detalles de cómo se realiza el proceso. (Los detalles se darán en una etapa posterior).

Según la visión que se tenga del módulo, se tienen diferentes atributos. En el caso de la visión externa (¿qué hace el módulo?), que es de la que más se preocupa el diseño estructurado, los atributos son: Entrada, Salida y Función; mientras que en el caso de la visión interna (¿cómo hace su función el módulo?), los atributos son el código y los datos internos.

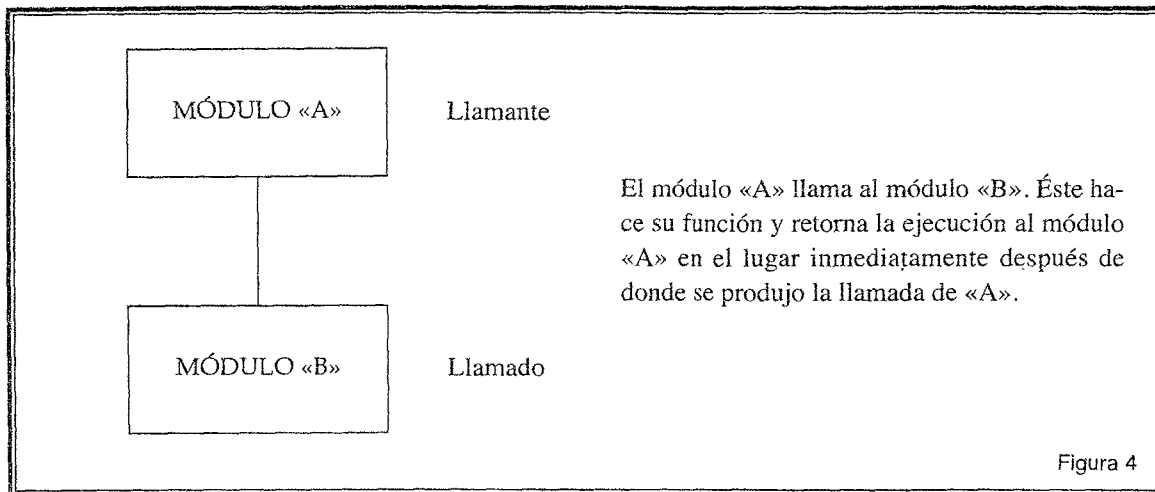
Con la técnica del Diagrama de Estructuras de Cuadros, los módulos se representan mediante un rectángulo y se nombran de forma que representen una acción. Por ejemplo, consultar stock, tratar petición, etc.

Los módulos predefinidos, que son aquellos que están disponibles en la librería del sistema o de la propia aplicación, y, por tanto, no es necesario codificarlos, se representan mediante un rectángulo con dos barras perpendiculares dentro.



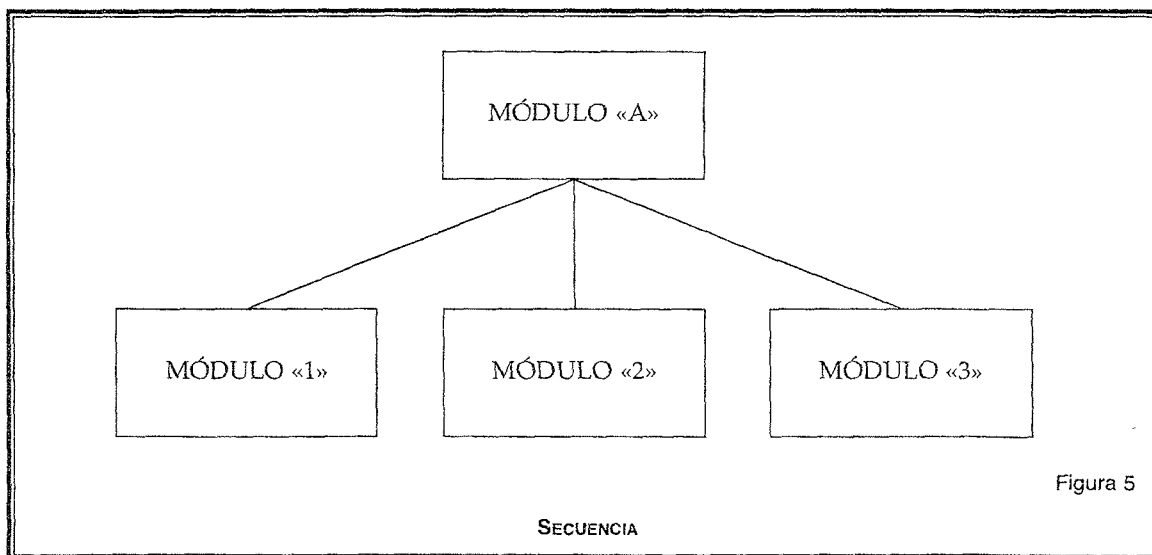
3.2.2. Conexiones entre Módulos.

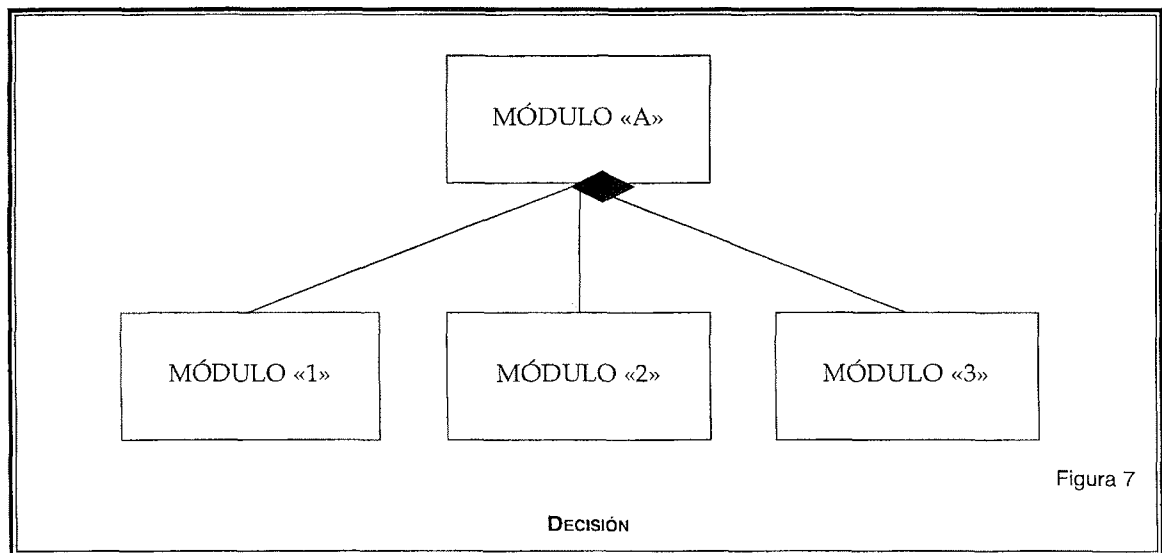
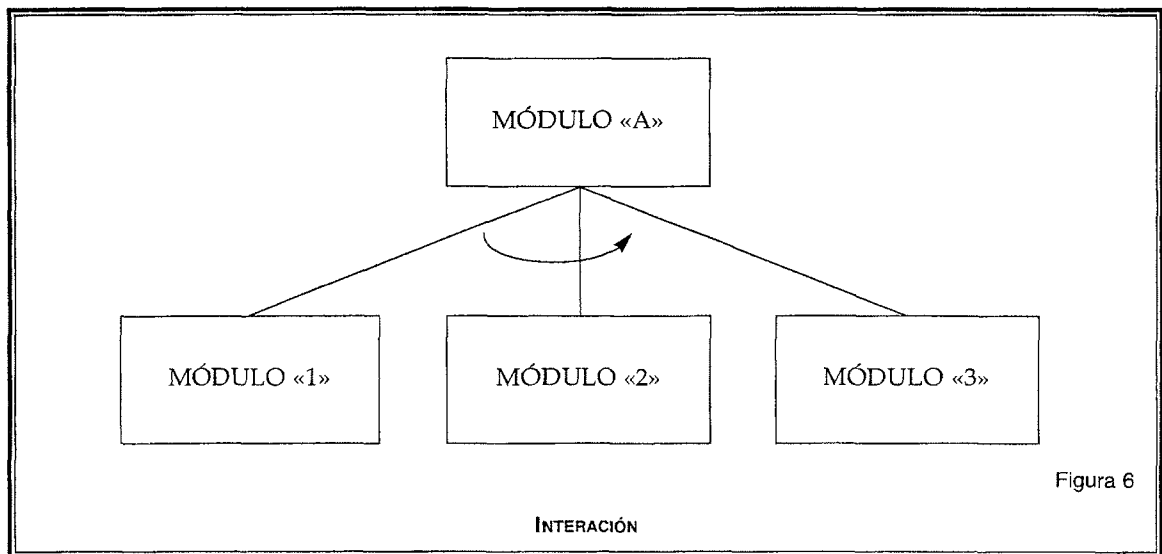
Un sistema está compuesto por módulos organizados jerárquicamente, cooperando y comunicándose entre sí para realizar una tarea. La conexión entre los módulos se representa mediante una línea e indica que un módulo (el superior en la jerarquía) llama a otro.



Un módulo puede llamar a varios a la vez. Dependiendo del tipo de llamada, se presentan los siguientes casos y simbolismos:

- Secuencia. Un módulo llama a varios y cada uno de los llamados se ejecuta una sola vez. La secuencia de ejecución suele ser de izquierda a derecha y de arriba hacia abajo.
- Iteración. Un módulo llama a varios y cada uno de los llamados se ejecuta un número «n» de veces.
- Decisión. Cuando existe una selección de camino, el módulo superior tendrá que realizar una decisión acerca del módulo de nivel inferior que ha de ejecutarse.



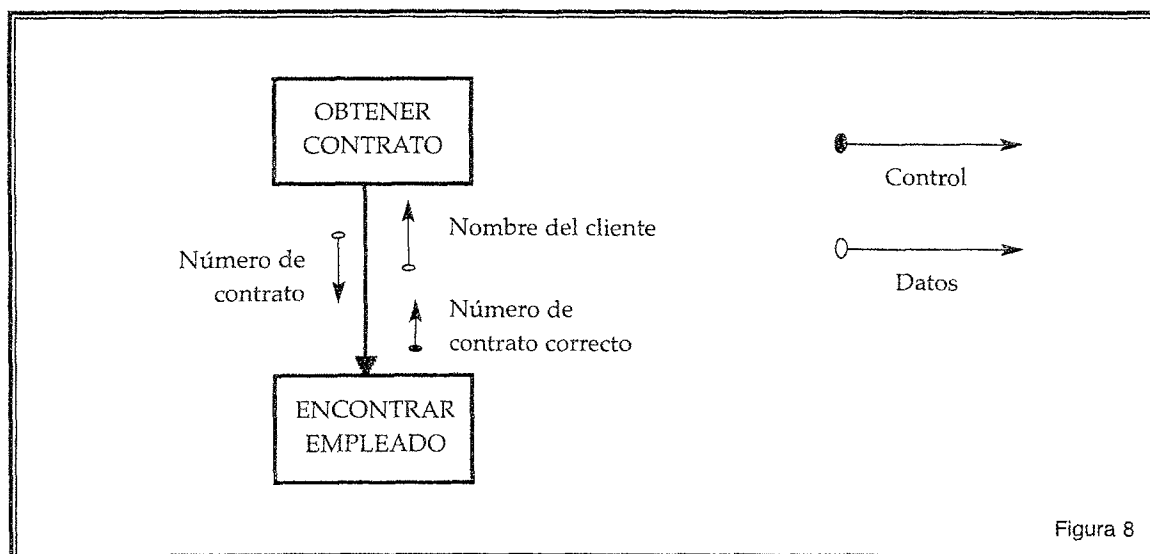


3.2.3. Comunicación entre Módulos.

La comunicación entre módulos se realiza a través de los datos y de los «flags» o controles.

Los «flag» sólo sirven como valores de condición para comunicarse condiciones entre los módulos y representan el paso de control entre módulos (p. ej., un módulo comunica a otro que su proceso ha terminado y le traspasa el control del sistema). Los «flag» no se procesan, pero sincronizan la operativa de los módulos.

Los datos son la información compartida por los módulos (el que llama y el llamado). Los datos se procesan y son de gran importancia para el sistema en sí mismo y hacia el exterior.



3.3. PRINCIPIOS DEL DISEÑO ESTRUCTURADO.

Los principios del Diseño Estructurado son tres: Descomposición, Jerarquía e Independencia.

1. DESCOMPOSICIÓN.

La descomposición es la separación de una función en otras que estuvieran contenidas en la primera.

Los objetivos que persigue la descomposición son los siguientes:

- Reducir el tamaño del módulo.
- Hacer el sistema más fácil de entender y de modificar.
- Minimizar la duplicidad del código. Esto es, evitar tener que realizar una función en más de un módulo
- Crear módulos útiles, es decir, reutilizables.

Al seguirse una estrategia «top-down», de arriba hacia abajo, la descomposición se va realizando paso a paso, según se vaya observando que los módulos realizan múltiples funciones, acercándonos cada vez más a la solución óptima. Ahora bien, el problema de la descomposición es saber en qué momento debemos dejar de descomponer módulos. Algunos criterios son:

- Dejar de descomponer cuando no se encuentren funciones bien definidas.
- Parar la descomposición cuando la interface con un módulo sea tan complicada como el módulo mismo.

2. JERARQUÍA.

Al dividir los módulos jerárquicamente se pretende controlar el número de ellos que interactúan directamente con cualquiera de los otros.

El objetivo de la jerarquía es conseguir separar los módulos que realizan tareas de cálculo y edición de aquellos que toman decisiones y llaman a otros módulos. Se debe lograr una organización en que los módulos de niveles altos y medios coordinen y manipulen a los módulos de bajo nivel, que son los que deben realizar las tareas de cálculo y edición.

3. INDEPENDENCIA.

Un módulo no tiene que preocuparse de los detalles de la construcción interna del resto de los módulos. Hay que ver los módulos como «cajas negras», esto es, sólo por su función y por su apariencia externa.

Si los módulos individuales son completamente independientes unos de otros, el esfuerzo de desarrollo del sistema es una función lineal del número de módulos de éste.

3.4. UN EJEMPLO DE DIAGRAMA DE ESTRUCTURA DE CUADROS.

El siguiente ejemplo muestra un proceso de emisión de cheques para el pago de nóminas de los empleados de una empresa. En él se diferencian los cálculos relativos a los trabajadores empleados por horas y los que poseen contrato. La lectura del fichero de empleados y la impresión de los cheques son módulos ya disponibles en las librerías del sistema, es decir, módulos predefinidos.

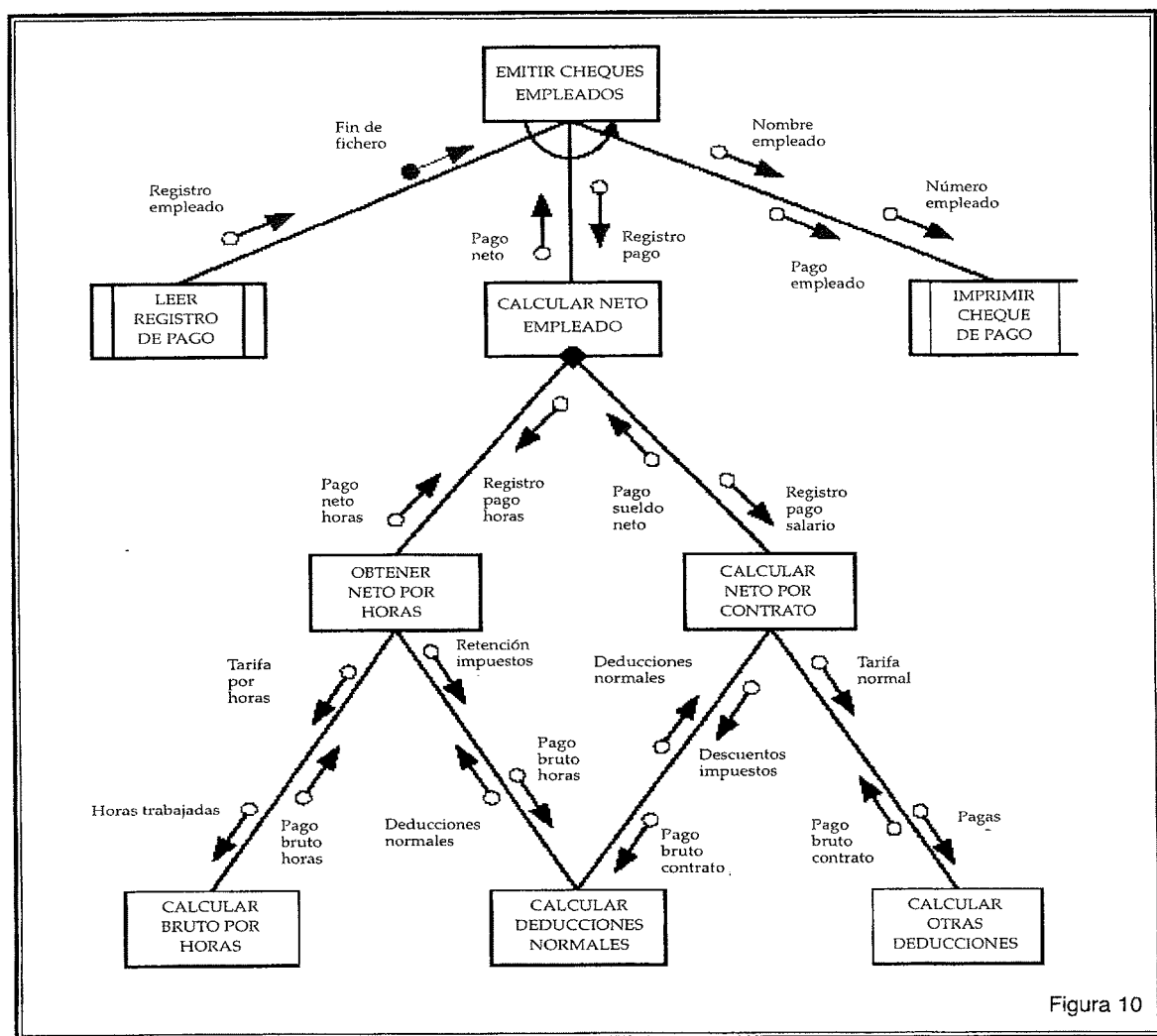


Figura 10

4. ESTRATEGIAS DE DISEÑO.

El Diseño Estructurado permite pasar de las representaciones de la información obtenidas en la fase de Requisitos del Software (Diagramas de Flujo de Datos) a una descripción del diseño de la estructura del programa (Diagramas de Estructura de Cuadros). En función del tipo de flujo de información de que se trate, los pasos para lograrlo son diferentes y así, distinguiremos las dos estrategias que se estudian a continuación: el análisis de transformación y el análisis de transacción.

El punto de partida es el Diagrama de Flujo de Datos sobre el que se vaya a realizar el diseño. Se estudiará su forma y dependiendo de su estructura se utilizará una de las dos estrategias mencionadas para obtener el Diagrama de Estructura de Cuadros. El uso de una de las estrategias no significa que la otra no pueda ser utilizada; dependerá únicamente de la forma del DFD y del peso de la actividad de los procesos.

4.1. ANÁLISIS DE TRANSFORMACIÓN.

El Análisis de Transformación es un conjunto de pasos de diseño que permiten obtener, a partir de un DFD con características de flujo de transformación, la estructura del sistema.

En un DFD con características de flujo de transformación se pueden distinguir tres zonas:

- Flujo de llegada o de entrada.
- Flujo de transformación o Centro de Transformación.
- Flujo de salida.

La información entra al sistema mediante caminos que transforman los datos externos (flujo de llegada). A continuación los datos son tratados en el núcleo o Centro de Transformación y se dirigen por caminos que conducen a la salida.

Esta división en tres partes facilita que los datos que necesite el sistema se recojan por los módulos que se encuentran en las ramas de la izquierda y los datos que se intercambian en esas ramas serán ascendentes (información de entrada al sistema). En las ramas centrales habrá información tanto ascendente como descendente porque aquí los módulos elaboran nuevos datos, En las ramas de la derecha la información será ya la definitiva y el sentido de los datos deberá ser descendente.

Una vez determinado que el DFD tiene características de transformación, los pasos de que consta el Análisis de Transformación son los siguientes:

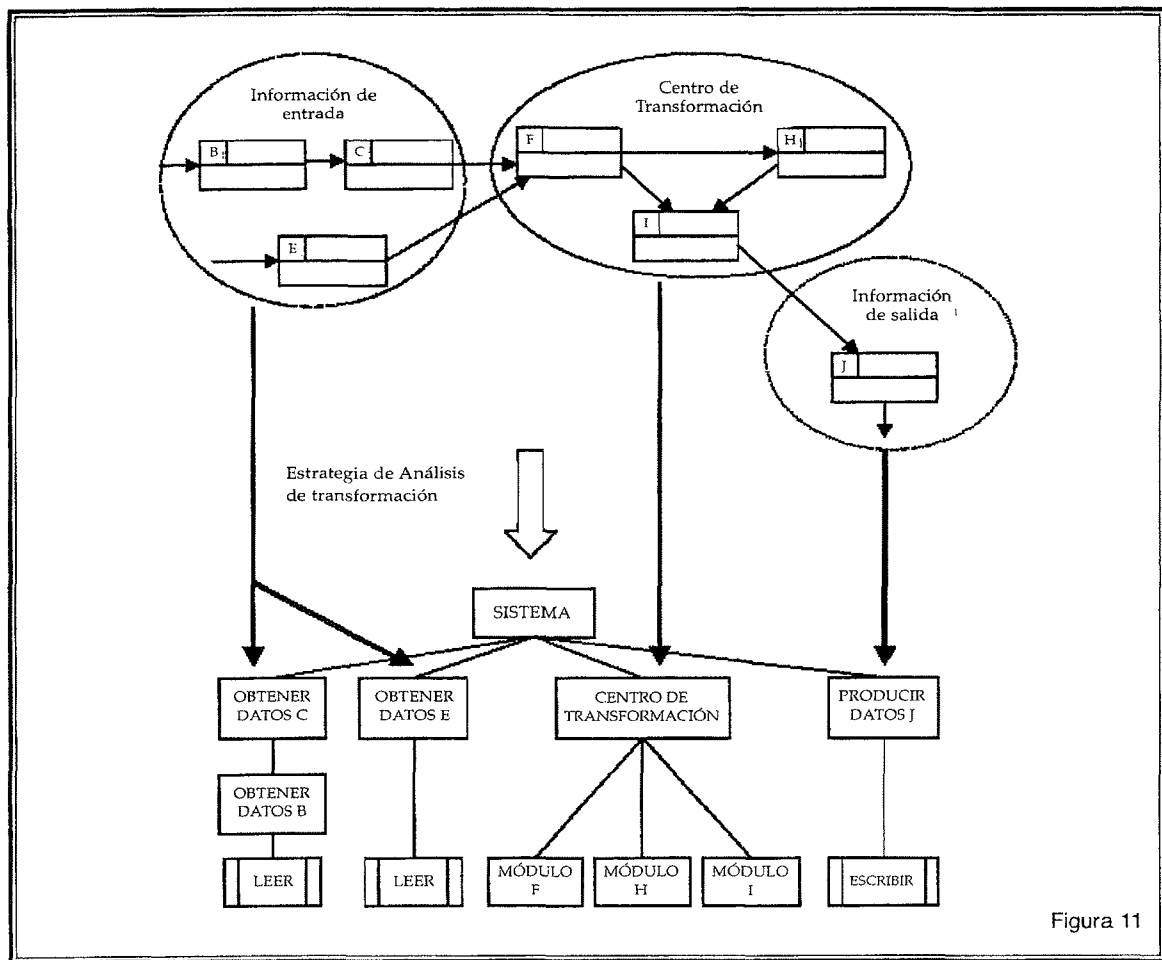


Figura 11

Aislar el Centro de Transformación.

El Centro de Transformación es la parte del DFD que contiene las funciones esenciales del sistema y es independiente de las características particulares de la entrada y la salida. Para aislar el Centro de Transformación habrá que especificar los límites del flujo de llegada y los del flujo de salida. Ambos límites están abiertos a la interpretación de cada diseñador, pudiendo derivarse soluciones de diseño alternativas variando la colocación de los límites del flujo. Cualquier pequeña variación tendrá un cierto impacto sobre la estructura final.

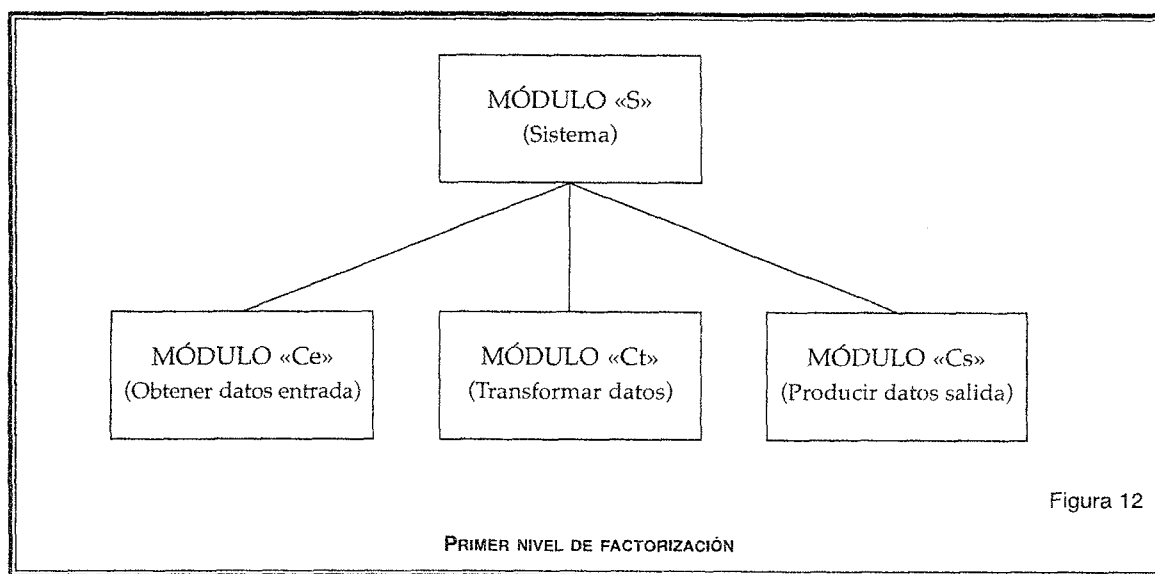
Ejemplo. Sobre el DFD anterior se ha considerado que el Centro de Transformación está constituido por los procesos F, H e I.

Realizar el primer nivel de factorización en Diagrama de Estructura de Cuadros.

La estructura del sistema representa una distribución descendente del control. La aplicación del Análisis de Transformación (pasar del DFD al Diagrama de Estructura de Cuadros, DEC) 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.

En el primer nivel de factorización o descomposición aparecerán subordinados a un módulo de control del sistema (cuyo nombre suele coincidir con el diagrama de contexto obtenido en la fase de análisis) tres módulos:

- Un módulo controlador del proceso de la información de llegada, que coordina la recepción de todos los datos que le llegan. En el ejemplo, Ce constituido por los procesos B, C y E.
- Un módulo controlador del Centro de Transformación, que supervisa todas las operaciones sobre los datos en su forma interna. En el ejemplo, Ct constituido por los procesos F, H e I.
- Un módulo controlador del proceso de la información de salida, que coordina la producción de la información de salida. En el ejemplo, Cs constituido por el proceso J.



Elaborar el segundo nivel de factorización.

El segundo nivel de factorización se realiza mediante la conversión de las transformaciones de cada proceso de un DFD en los módulos correspondientes del diagrama de estructura.

Para ello, se empieza en el límite del Centro de Transformación y yendo hacia fuera a lo largo de los caminos de llegada y salida, las transformaciones se convierten en niveles subordinados de la estructura del software. Además, es necesario introducir módulos predefinidos que den las diferentes entradas y/o salidas que necesita y/o genera el sistema.

Ejemplo. El segundo nivel de factorización es el que aparece representado en la figura.

Refinar 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 de sentido común, además de por los requisitos del software. No obstante, se utilizan dos unidades de medida, el acoplamiento y la cohesión, que se verán más adelante, como mecanismos para evaluar y mejorar el diseño obtenido.

4.2. ANÁLISIS DE TRANSACCIÓN.

El Análisis de Transacción es un conjunto de pasos de diseño que permiten obtener, a partir de un DFD con características de flujo de transacción, la estructura del sistema.

Un DFD con características de flujo de transacción se caracteriza por tener una forma en la que 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 denomina «transacción» y desencadena otro flujo de datos a lo largo de uno de los muchos caminos.

El centro de flujo de información desde el que emanan muchos caminos de acción alternativos de forma exclusiva se denomina Centro de Transacción.

Una vez determinado que el DFD tiene características de transacción, los pasos de que consta el Análisis de Transacción son los siguientes:

1. Identificar el Centro de Transacción.

El Centro de Transacción puede identificarse a partir del DFD viendo cuál es el origen de una serie de caminos de información que fluyen radialmente de él.

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

Ejemplo. Sobre el DFD anterior el Centro de Transacción está constituido por el proceso 1.

2. Transformar el DFD en la estructura adecuada al proceso de transacciones (primer nivel de factorización).

El flujo de transacciones se convierte en una estructura de sistema formada por una bifurcación de entrada y una bifurcación de salida.

3. Factorizar la estructura de cada camino de acción (segundo nivel de factorización).

La estructura de la bifurcación de entrada se desarrolla de la misma forma que en análisis de transformación, comenzando en el centro de la transacción y continuando desde el límite hacia fuera a lo largo del camino de llegada.

Se desarrolla cada camino de acción dependiendo de su tipo de flujo. Es decir, cada camino de flujo de acción del DFD se convierte en una estructura que se corresponde con las características específicas del flujo (de transformación o de transacción).

4. Refinar la estructura del sistema utilizando medidas y guías de diseño.

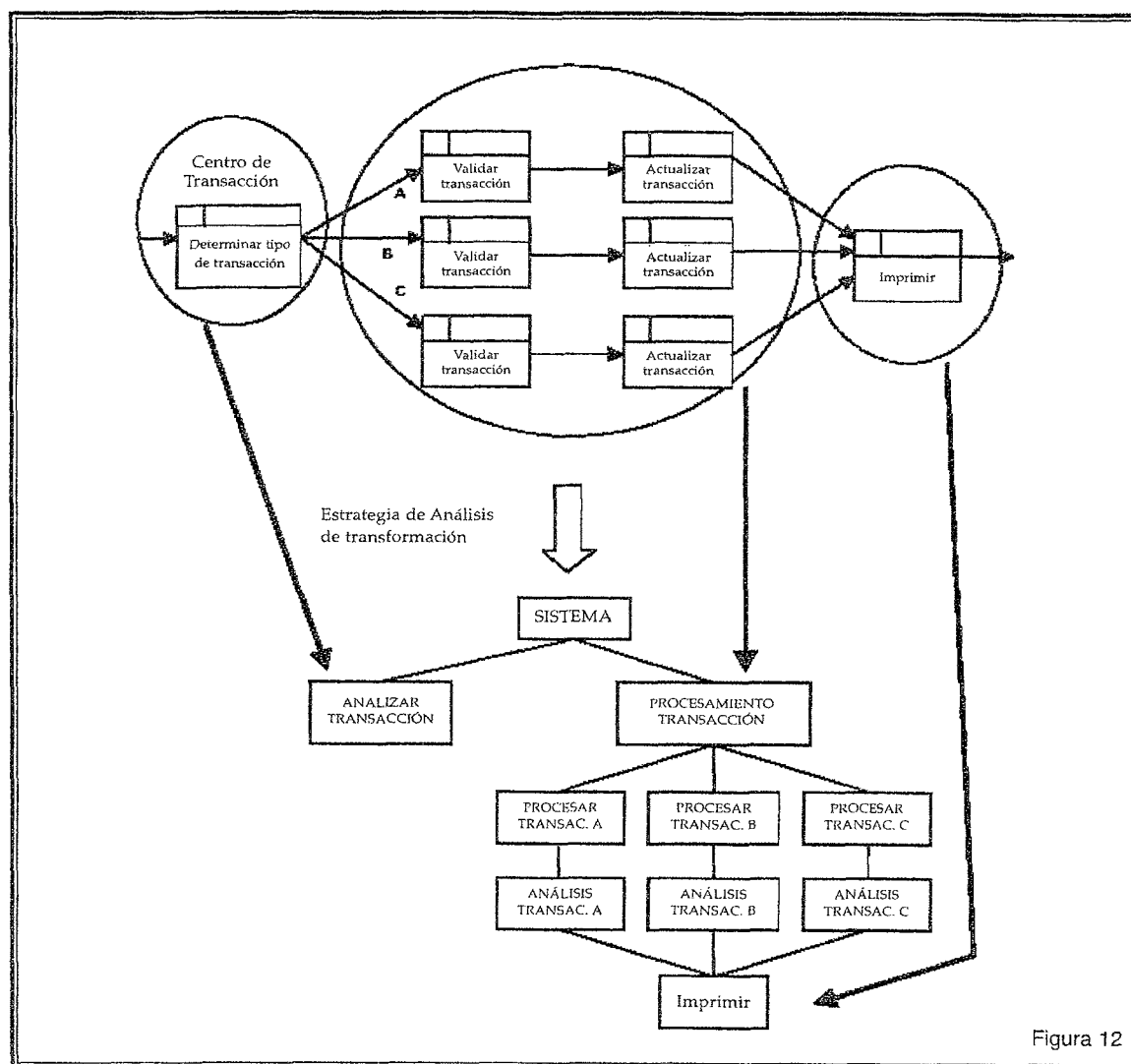


Figura 12

5. EVALUACIÓN DE LA CALIDAD DEL DISEÑO.

Un buen diseño debe organizar la complejidad del problema de manera que el esfuerzo asociado con su desarrollo, prueba, entendimiento y mantenimiento pueda ser controlado y minimizado. En relación con esto, uno de los principios fundamentales del diseño estructurado es que el sistema debe ser dividido en una serie de módulos que resulten fácilmente manejables y la división se ha de llevar a cabo de forma que los módulos sean tan independientes como sea posible y que cada módulo sólo realice una función simple relacionada con el problema.

Se hace necesario, pues, evaluar la calidad de la estructura de los diseños software y, a estos efectos, se utilizan dos unidades de medida: el acoplamiento y la cohesión, que se estudian a continuación.

5.1. ACOPLAMIENTO.

Se define acoplamiento como el grado de interdependencia entre los módulos. Para un buen diseño, se debe intentar minimizar el acoplamiento, es decir, hacer que los módulos sean tan independientes unos de otros como sea posible.

Un bajo acoplamiento indica que se ha hecho una buena descomposición, y es deseable, entre otras cosas, porque:

- Cuantas menos conexiones existan entre dos módulos, menos oportunidad habrá para que se produzca el «efecto onda», es decir, que un defecto en un módulo pueda afectar a otro.
- Posibilita que cada cambio realizado en un módulo afecte lo menos posible a otros, esto es, permite cambiar un módulo con el mínimo riesgo de tener que cambiar otro.
- Garantiza que mientras se está manteniendo un módulo no hay necesidad de preocuparse de los detalles internos (código) de cualquier otro módulo.

De mejor a peor, se contemplan los siguientes niveles de acoplamiento:

- Acoplamiento normal:
 - De datos.
 - De marca o por estampado.
 - De control.
- Acoplamiento externo.
- Acoplamiento común.
- Acoplamiento por contenido.

El acoplamiento normal es el que sucede cuando dos módulos intercambian datos, pero éstos no interfieren en la operativa normal de la función que realiza el módulo de nivel inferior. Es decir, en el acoplamiento normal un módulo «A» llama a otro «B», el módulo «B» se ejecuta y retorna el control al módulo «A».

En el acoplamiento normal, dependiendo de los datos que intercambien los módulos en su comunicación se distinguen tres tipos:

- Acoplamiento de datos. Los módulos se comunican mediante parámetros, donde cada parámetro es una unidad elemental de datos. Es el tipo de acoplamiento más deseable, salvo el normal, propiamente dicho, donde los módulos no intercambian ningún parámetro entre ellos.
- Acoplamiento de marca o por estampado. En la comunicación, los módulos se pasan datos con estructura de registro. Este tipo de acoplamiento no es muy deseable si el módulo que recibe el registro no necesita todos los elementos de datos que se le pasan, sino parte de ellos.
- Acoplamiento de control. Sucede si los datos que se intercambian en la comunicación entre módulos son controles. No es un buen acoplamiento porque no permite que los módulos sean totalmente independientes, debido a que un módulo controla a otro influyendo en su ejecución.

El acoplamiento externo es el que sucede cuando los módulos están ligados a componentes externos, por ejemplo, dispositivos de E/S, protocolos de comunicaciones, etc.

El acoplamiento común es el que sucede cuando un número indeterminado de módulos (más de dos) hacen referencia a un área común de datos. Este tipo de acoplamiento es desaconsejable porque un defecto en un módulo que utiliza el área común puede transmitir ese error a otro módulo que utilice dicho área y porque los módulos así acoplados se modifican peor que los acoplados normalmente.

El acoplamiento por contenido se presenta cuando un módulo hace referencia a la parte interior de otro, ya sea porque le modifica algún elemento, porque usa una variable local de ese módulo, porque ambos módulos compartan los mismos contenidos, etc. En cualquier caso, este tipo de acoplamiento es totalmente inaceptable porque rompe la jerarquía de funcionalidad de la estructura.

5.2. COHESIÓN.

La cohesión hace referencia a la relación existente entre los elementos de un mismo módulo. Su objetivo es organizar los elementos de tal manera que los que tengan más relación a la hora de realizar una tarea pertenezcan al mismo módulo, y los elementos no relacionados figuren en módulos separados.

La cohesión se puede definir como la medida de la relación funcional de los elementos de un módulo; entendiendo por elementos, tanto la sentencia o grupo de sentencias que lo componen, como las definiciones de datos o las llamadas a otros módulos. Idealmente, un módulo coherente sólo debe hacer una única cosa.

El concepto de cohesión es importante a la hora del diseño, ya que cuanta mayor cohesión tengan los módulos, es menor el coste de programar y es mayor la calidad del sistema

Ordenados de mayor a menor, se distinguen los siguientes tipos de cohesión:

- Cohesión funcional. Es el grado más alto de cohesión. Todos los elementos que componen el módulo están relacionados en el desarrollo de una única función.
- Cohesión secuencial. Un módulo realiza distintas tareas dentro de él en secuencia, de forma que las entradas de cada tarea son las salidas de la anterior.
- Cohesión comunicacional. Un módulo realiza actividades paralelas usando los mismos datos de entrada y salida.
- Cohesión procedimental. El módulo tiene una serie de funciones relacionadas por un procedimiento efectuado por el código. Es decir, este tipo de cohesión es similar a la secuencial, pero con paso de controles.
- Cohesión temporal. Un módulo presenta este tipo de cohesión cuando sus elementos están implicados en actividades relacionadas con el tiempo.
- Cohesión lógica. Se produce cuando las actividades que realiza el módulo tienen la misma categoría, es decir, es como si se tuvieran partes independientes dentro del mismo módulo.

- Cohesión casual o coincidente. Un módulo posee este tipo de cohesión cuando sus elementos contribuyen a las actividades relacionándose mutuamente de una manera poco significativa. Este tipo de cohesión viola el principio de independencia y de caja negra de los módulos, ya que es necesario que el módulo «padre» tenga conocimiento de la estructura interna de los módulos «hijos».

5.3. OTROS PARÁMETROS DE CALIDAD: EL FAN-IN Y EL FAN-OUT.

Además del acoplamiento y la cohesión, otros parámetros a considerar a efectos de garantizar la calidad del diseño son: el grado de absorción o «fan-in» y la diseminación del control o «fan-out».

El grado de absorción o fan-in de un módulo es el número de superordinados inmediatos que tiene dicho módulo. Siempre que sea posible se debe maximizar el fan-in durante el proceso de diseño, ya que cada instancia de fan-in múltiple indica que se ha evitado la duplicación de código.

El fan-in se consigue a través de un proceso analítico que acompaña los pasos del procedimiento de diseño estructurado. A medida que se identifica un nuevo módulo en el diagrama de estructuras debemos preguntarnos si existe algún módulo que realice la función requerida.

La diseminación del control o fan-out de un módulo es el número de subordinados inmediatos de dicho módulo. Un fan-out muy alto o muy bajo es un posible indicador de un diseño pobre, aunque es preferible un fan-out bajo que uno alto.

6. EL LENGUAJE DE DISEÑO DE PROGRAMAS (PDL).

El lenguaje de diseño de programas (PDL Programs Design Language), también llamado «lenguaje de especificación de problemas» (PSL Problems Specification Language) es un lenguaje de especificación que hace uso de un vocabulario y una sintaxis limitados; es decir, es un subconjunto de palabras del idioma elegido, de las cuales unas se utilizan para formar las construcciones propias de la programación estructurada (secuencia, selección y repetición) y otras incluyen un conjunto de verbos que reflejan acciones simples.

Su objetivo es evitar todas las imprecisiones y las ambigüedades del lenguaje natural y es un lenguaje intermedio entre éste y los lenguajes de programación.

Los componentes principales del PDL son:

- Verbos transitivos. Los verbos que se pueden utilizar se han de corresponder con acciones que un sistema informático puede realizar. Por ejemplo, mostrar, escribir, buscar, leer, sumar, etc.
- Palabras reservadas. Se heredan de la programación estructurada y se corresponden con acciones de repetición y condición.
- Objetos. Sólo pueden usarse nombres de objetos definidos en el diccionario de datos (almacenes, flujos, componentes de flujos, etc.) o términos conocidos por su utilización en el sistema, términos locales (términos que se definen dentro de la especificación del proceso, donde ocurren, y no en el diccionario de datos).

El PDL es la técnica más conveniente a efectos de definir un programa, siempre que se utilice siguiendo una reglas:

- Restringir la especificación del proceso a una sola página de texto.
- No usar más de tres niveles de anidamiento.
- Utilizar sangrías para evitar confusiones en los niveles de anidamiento.

Las principales ventajas del PDL son que permite desarrollar descripciones complejas de forma estricta respecto al vocabulario y la organización, y los usuarios, después de una explicación, pueden ser capaces de leerlo y entenderlo.

BIBLIOGRAFÍA

- Ingeniería del Software. Un enfoque práctico. Roger S. Pressman. Ed. McGraw Hill.
- Análisis y Diseño detallado de Aplicaciones Informáticas de Gestión. Mario Piattini y otros. Ed. Ra-Ma.
- Ingeniería del Software. Ian Sommerville. Ed. Addison-Wesley.
- Metodología de Planificación, Desarrollo y Mantenimiento de Sistemas de Información. Métrica versión 3. Técnicas y Prácticas. Ministerio para las Administraciones Públicas.
- Metodología de Planificación y Desarrollo de Sistemas de Información. Métrica versión 2.1. Guía de Técnicas. Ministerio para las Administraciones Públicas. Ed. Tecnos.
- Temario de las pruebas selectivas para ingreso en el Cuerpo Superior de Sistemas y Tecnologías de la Información de la Administración del Estado. ASTIC.
- Temario de las pruebas selectivas para el acceso, por promoción interna, al Cuerpo de Gestión de Sistemas e Informática de la Administración del Estado. Ministerio para las Administraciones Públicas.
- Temario del Máster en Ingeniería del Software. Facultad de Informática. Universidad Politécnica de Madrid. Ed. Centro de Estudios Financieros.

