TEMA 52

DISEÑO LÓGICO DE FUNCIONES. DEFINICIÓN DE FUNCIONES. DESCOMPOSICIÓN MODULAR. TÉCNICAS DESCRIPTIVAS. DOCUMENTACIÓN

ÍNDICE

1. INTRODUCCIÓN

Las especificaciones de programa que se producen al final del diseño de programas deben asegurar que los programas satisfacen los requerimientos de usuario definidos por los procedimientos de usuario.

Además de satisfacer los requerimientos de usuario, los programas deben satisfacer otros criterios. Primero, deben ser fáciles de leer y comprender. Esto permitirá que varias personas trabajen en el mismo programa. También permitirá que otras personas distintas a las que generaron el código lo modifiquen. Segundo, los programas se deben acomodar a los cambios del sistema que se dan después de su construcción. Deben ser fáciles de mantener para que soporten esos cambios. El mantenimiento se mejora si las funciones bien definidas aparecen en la misma sección del código de programa. Los cambios en esas funciones sólo afectarán a esa sección de código y no a todo el programa. Finalmente, el tercer requerimiento es que el programa se escriba para utilizar los recursos del ordenador en la forma más efectiva.

Estos objetivos se alcanzan con diseño modular de programas y programación estructurada. El diseño modular de programas sitúa cada función del sistema de usuario bien definida en un módulo del programa. La programación estructurada usa estructuras de bloque para hacer programas legibles. Obsérvese que la modularidad y la programación estructurada son consistentes con el criterio para buenos diagramas de flujo de datos (DFD). En un buen DFD cada proceso representa una función bien definida. Además, cada proceso se describe utilizando inglés estructurado, que emplea una estructura similar a la programación estructurada. Estos DFD se deben convertir en programas de forma que se pase una buena estructura de DFD a la especificación de programa.

El diseño de programas utiliza las otras dos partes del diseño detallado, el diseño de procedimientos de usuario y el diseño de base de datos. Utiliza los procedimientos de usuario para definir la interfaz de ordenador, y el diseño de base de datos para escribir el código de entrada salida (E/S) que accede a las partes de la base de datos necesarias para el programa.

El diseño de programas comienza con los DFD dentro de los límites de automatización y utiliza las salidas del diseño de base de datos y del diseño de procedimientos de usuario para producir los programas.

2. PASOS EN EL DISEÑO DE PROGRAMAS

La figura 52.1 muestra lo que ocurre después de terminar el diseño amplio. El diseño detallado que sigue al diseño amplio comienza con tres actividades: diseño de bases de datos, diseño de procedimientos de usuario y programas de diseño.

El *diseño de base de datos* utiliza el modelo de datos conceptual para producir un diseño de base de datos. El *diseño de los procedimientos de usuario* usa aquellas partes del DFD fuera de los límites de automatización para diseñar los procedimientos de usuario. El *diseño de programas* utiliza las partes del DFD interiores a los límites de automatización para diseñar programas. El diseño de programas se describe en este capítulo. Está compuesto por los procesos 5, 6, 7, 8, 9 y 10 de la figura.

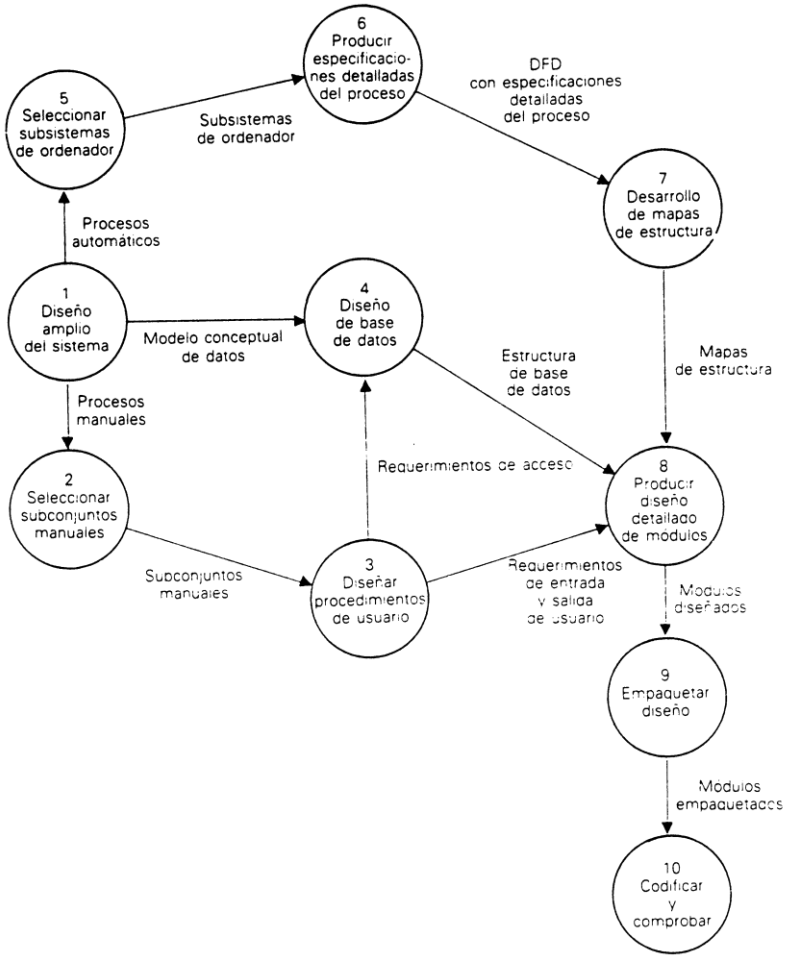

Figura 52.1. Diseño del programa

2.1. División en Subsistemas de Ordenador

El primer paso del diseño de programa divide el DFD en subsistemas de ordenador. Esta división es similar a la división utilizada en el diseño de procedimientos de usuario. Al igual que en el diseño de procedimientos de usuario, los procesos lógicamente conectados se agrupan en sistemas de ordenador. El método más común es agrupar todos los procesos que siguen a través de un tipo de entrada.

La figura 52.2 muestra cómo los procesos se pueden dividir en subsistemas. El DFD de la figura 52.2 describe el procesamiento de pedidos de cliente. Los procesos 3.1, 3.2 y 3.3 comprueban si los elementos pedidos están en el almacén y crean una orden de almacén para esos elementos. Los procesos 3.4 y 3.5 reciben mercancías de los almacenes, comprueban si se encuentran en el pedido y las envían a los clientes.

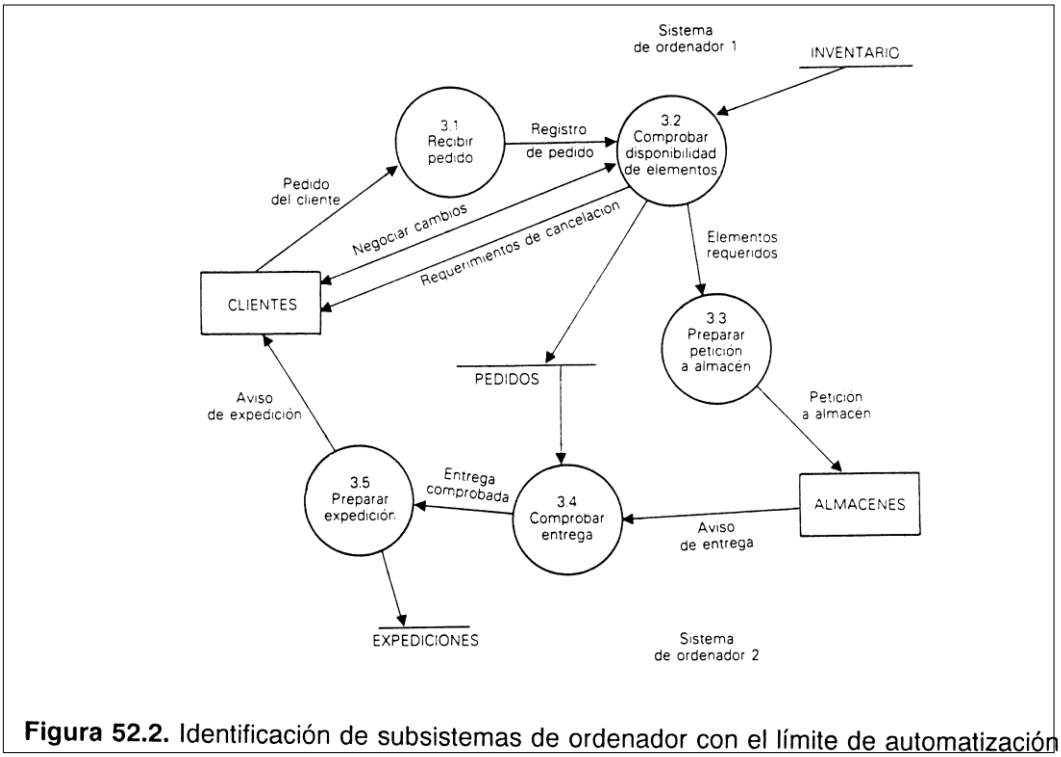

Figura 52.2. Identificación de subsistemas de ordenador con el límite de automatización

Los procesos de la figura se agrupan en dos subsistemas. El primero contiene los procesos 3.1, 3.2 y 3.3. Estos procesos siguen los pedidos del cliente hasta que se genera la orden de almacén. La orden requiere que el almacén prepare los elementos de los pedidos para la entrega. El segundo subsistema está formado por los procesos 3.4 y 3.5. Éstos siguen la notificación de entrega hasta que la orden de expedición se envía al cliente.

Después de elegir los subsistemas, se desarrollan las especificaciones detalladas del proceso para cada proceso de cada subsistema. Entonces se convierten estos procesos en módulos de programas, que se muestran en el mapa de estructura. En este mapa también se muestran las llamadas entre esos módulos. El siguiente paso produce una especificación detallada de programa para cada módulo. Finalmente, los módulos de programa se agrupan en módulos de carga. Esta transición gradual desde el DFD al diseño de programas se ilustra en la figura 52.3 y se describe a continuación.

2.2. Transición de diagramas de flujo de datos a especificaciones detalladas de programa

La figura 52.3 describe el diseño de programas con más detalle. Utiliza el proceso 3.2 del ejemplo de la figura 52.2 para ilustrar esos pasos. El primer paso desarrolla las especificaciones detalladas del proceso. Como muestra la figura, las especificaciones detalladas del proceso se escriben en inglés estructurado y empieza a acercarse al código de programa. Incluyen todas las condiciones que pueden surgir en el proceso y cómo se tratan. Así, por ejemplo, el paso 1 de la figura define tres cosas que suceden si no hay elementos suficientes en el almacén para el pedido. El pedido se puede cancelar, se puede reducir la cantidad pedida o borrar el elemento del pedido. Sin embargo, las especificaciones de proceso no son completas, porque no incluyen cosas como referencias a registros, comprobación detallada de errores, operaciones de E/S y diálogos de usuario. Éstas se añaden más tarde a la especificación.

A continuación, los DFD se convierten en mapas de estructuras y cada proceso del DFD pasa a ser un módulo de programa en el mapa de estructura. Así, el proceso 3.2 será ahora el módulo **comprobar-disponibilidad-elementos** en el mapa de estructura. Este será uno de los módulos que llame el programa principal **programa-principal.** Los mapas de estructura muestran los módulos de programa y cómo se pasan información. Así, **programa−principal** pasa un valor del parámetro "pedido" al módulo **comprobar-disponibilidad-elementos** y obtiene un valor del parámetro "estado". El valor de "estado" dice al **programa-principal** si el pedido se acepta o se rechaza.

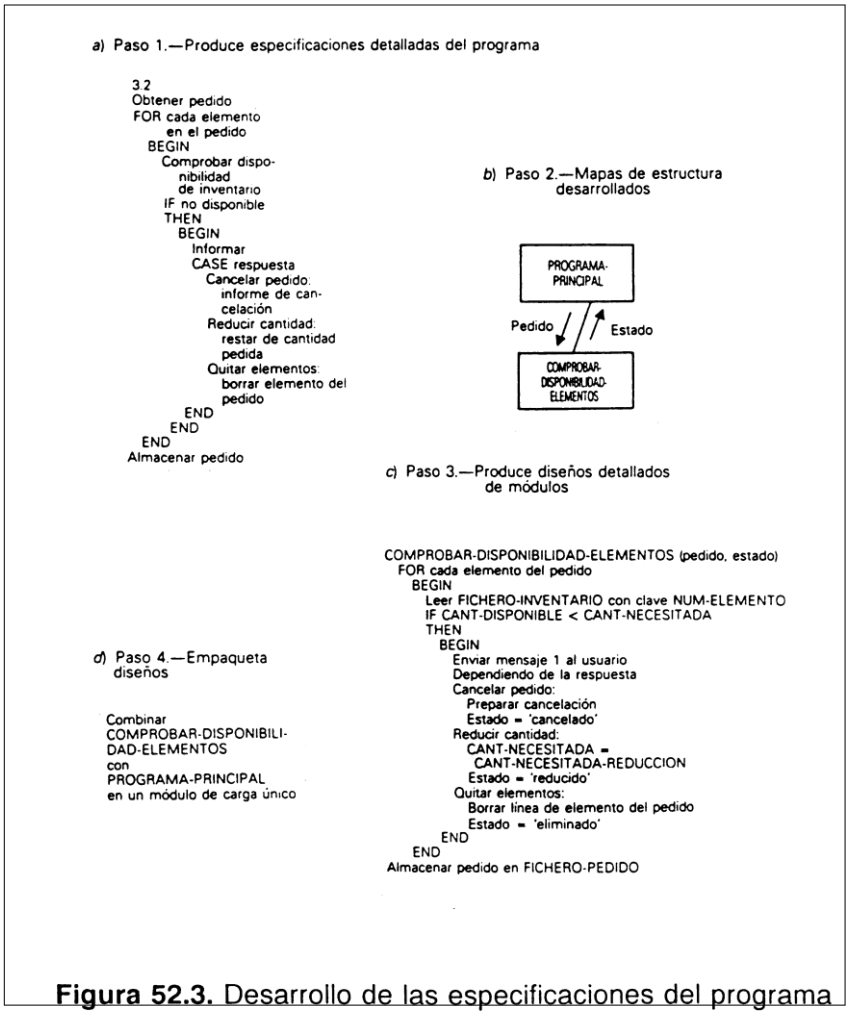

Figura 52.3. Desarrollo de las especificaciones del programa

Una vez que se sabe cómo se llaman unos módulos de programa a otros, se convierten las especificaciones detalladas del proceso en diseño detallado de módulo, que incluye, como se indica en el paso 3 de la figura, procedimientos llamadas y parámetros pasados al módulo. También incluyen diálogos de usuario, comprobación de errores, código de E/S y referencias a bases de datos. Utilizan el diseño de base de datos y el diseño de procedimientos de usuario para elegir correctamente las referencias a bases de datos y el diálogo de usuario.

Finalmente, los módulos se empaquetan en módulos de carga. Los módulos de carga son grupos de módulos de programa que se cargan juntos en memoria.

El resto de este capítulo describirá en detalle las técnicas usadas. El capítulo se concentra en los mapas de estructura y en el diseño de módulos de carga.

3. MAPAS DE ESTRUCTURA

El mapa de estructura está formado por los módulos de programa y sus interconexiones. Cada módulo de programa se representa por una caja rectangular. Los módulos en el nivel superior del mapa de estructura llaman a módulos de nivel inferior. Las interconexiones entre módulos se representan por líneas entre las cajas rectangulares. Las conexiones describen los flujos de datos entre módulos. La figura 52.4 muestra un mapa de estructura simple. Está compuesto por cuatro módulos. El módulo de alto nivel se llama **calcular-total-venta.** Este módulo llama a tres módulos de nivel inferior para completar la tarea. Estos son **leer-transacción-venta** para leer las transacciones individuales de venta. Entonces llama a **sumar-a-total** para suma la cantidad de cada transacción. Finalmente, llama a **sacar-total** para sacar la suma.

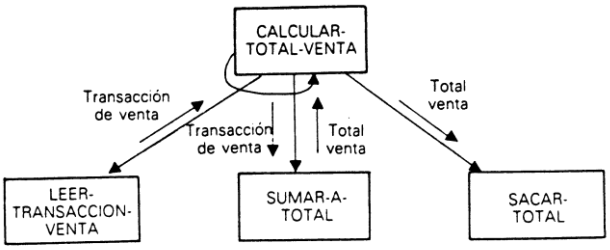

Figura 52.4. Mapa de estructura

CONVENCIONES DEL MAPA DE ESTRUCTURA

Los mapas de estructura usan una serie de convenciones para describir las operaciones del sistema. Las más importantes especifican la secuencia de ejecución y los parámetros que se pasan los módulos.

Paso de parámetros

El módulo que llama pasa un conjunto de valores al módulo llamado y recibe un conjunto de valores de vuelta. Estos valores se pasan como valores de parámetro. Los parámetros se muestran en el mapa de estructura junto a la conexión. Así, en la figura se pasa un valor de "transacción de venta" del módulo **leer-transacción-venta** al módulo **calcular-total-venta.** Entonces, el módulo **calcular-total-venta** pasa el valor de "transacción de venta" al módulo **sumar-a-total** y obtiene un valor de "total venta" a la vuelta. El valor de "total venta" se pasa luego del módulo **calcular-total-venta** al módulo **sacar-total.**

Secuencia de ejecución

Por convención, los módulos se ejecutan de izquierda a derecha. Así, en la figura, el módulo **leer-transacción-venta** se llama antes que el módulo **sumar-a-total.** El módulo **sacar-total** es el último en ser llamado.

Para representar decisiones y repeticiones, también se utilizan ciertas convenciones. Las decisiones se toman si un módulo ha de decidir llamar sólo a uno de entre varios. La repetición, por otro lado, ocurre cuando un módulo llama repetitivamente a otros.

La repetición se modela con una flecha de bucle (que acaba en el mismo lugar que termina). Como ejemplo, en la figura, el módulo **calcular-total-venta** llama a los módulos **leer-transacción-venta y sumar-a-total** un número cualquiera de veces.

Las decisiones se modelan con un rombo. En la figura 52.5 se da un ejemplo. En ella, el módulo A:

- Llama al módulo B o al C.
- Entonces ejecuta un bucle que llama a los módulos D, E y, a veces, al F.

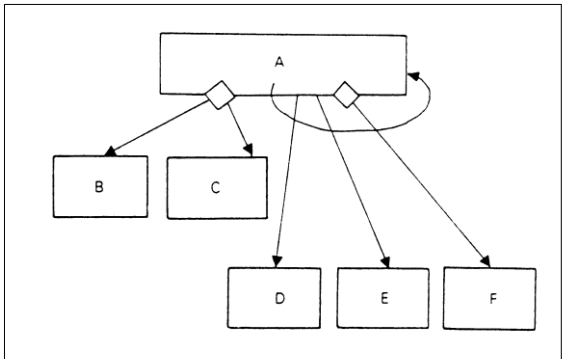

Figura 52.5. Anotaciones de procedimientos

4. MAPAS DE ESTRUCTURA Y DISEÑO ESTRUCTURADO

Los mapas de estructura se desarrollan por un proceso llamado diseño estructurado, cuyo objetivo es producir mapas de estructuras con propiedades proporcionadas por una práctica de buena programación. Para juzgar cómo satisfacen los mapas de estructura estos objetivos, se utilizan varias propiedades. Estas propiedades se conocen normalmente como *acoplamiento* y *cohesión* de módulos. La cohesión de módulos se conoce también como consistencia de módulo.

El acoplamiento describe la naturaleza, dirección y cantidad de parámetros pasados entre módulos, mientras la cohesión describe cómo se codifican las funciones del sistema en los módulos. Un objetivo del diseño estructurado es minimizar la complejidad de acoplamiento entre módulos. El mapa de estructura con acoplamiento simple se dice que tiene bajo acoplamiento. Otro objetivo del diseño estructurado es representar una función bien definida del sistema en un solo módulo. Cuando ocurre esto se tiene alta consistencia, Los mapas de estructura con bajo acoplamiento y alta consistencia resultan con más independencia entre módulos y más fáciles de mantener, ya que cada módulo se puede modificar independientemente de los otros módulos.

Los que proponen el diseño estructurado han diseñado medidas de acoplamiento y de consistencia de módulo. Para describir el acoplamiento y la cohesión se utilizan varios términos. Estos se ordenan en un rango desde el menos hasta el más deseable.

ACOPLAMIENTO DE MÓDULOS

El acoplamiento de módulos mide la calidad de las conexiones entre los módulos del mapa de estructura. El objetivo es diseñar un mapa de estructura que sólo pase datos y no información de control entre módulos de programa. Sin embargo, un primer corte en el diseño no puede alcanzar este objetivo y el mapa de estructura puede exhibir otros tipos de acoplamiento. Hay muchas formas de describir el acoplamiento. El mejor es el acoplamiento de datos, seguido por el acoplamiento de control, acoplamiento de entorno común y acoplamiento de contenido.

Acoplamiento de contenido

Dos módulos están acoplados en contenido si uno de ellos hace una referencia directa al contenido del otro. Este tipo de acoplamiento permite que el módulo de llamada modifique una sentencia de programa del otro módulo. También permite que un módulo salte dentro del otro. Este acoplamiento se debería evitar a toda costa y los mapas de caracteres deberían asegurarse de que el único camino para pasar información entre módulos es el paso de parámetros durante la llamada a subrutinas.

Acoplamiento de entorno común

Dos módulos están en acoplamiento de entorno común si hacen referencia a la misma estructura de datos o elementos de datos en un entorno común.

Ejemplos de acoplamiento de entorno común incluyen áreas comunes en programas de usuario o ficheros compartidos. El efecto de este acoplamiento es que los módulos que aparecen sin relación en un mapa de estructura están acoplados mediante el uso de datos comunes.

Acoplamiento de control

Dos módulos están acoplados en control si uno de ellos pasa un elemento de control al otro. Este elemento de control afecta al procesamiento del módulo receptor. Ejemplos típicos de acoplamiento de control son indicadores (flags), Códigos de función y conmutadores.

Este acoplamiento viola el principio de ocultamiento de la información. Pasar un elemento de control de un módulo a otro implica que el módulo que llama debe conocer el método de operación del módulo llamado. Cualquier cambio requerido en el módulo llamado puede necesitar un cambio en el que lo llama.

Acoplamiento de datos

Dos módulos están acoplados en datos si no están acoplados en contenido, en entorno común ni en control. Sólo se pasan elementos de datos como parámetros entre ellos.

El acoplamiento de datos es la forma más deseable de acoplamiento. El módulo que llama pasa los valores de los datos mediante parámetros al módulo llamado y espera que se haga algún cálculo con esos valores. Los resultados de esos cálculos se devuelven como valores de parámetros al módulo que llamó. El módulo que llama no necesita saber cómo se realizan esos cálculos.

Ejemplos comunes de acoplamiento de datos son las llamadas a módulos de entrada o de salida. Un módulo que llama puede necesitar alguna entrada; éste llama a otro módulo, **entrada**, que se la proporciona. El módulo que llama no se preocupa de cómo se obtienen esas entradas, sólo le conciernen los datos que recibe.

Las llamadas a módulos de salida son similares. El procedimiento que llama pasa los datos que van a salir al módulo **salida**; no le concierne cómo saca los datos el módulo salida.

CONSISTENCIA DE MÓDULOS

La consistencia de módulo mide el motivo por el cual un código aparece en el mismo módulo. Muchos escritores utilizan las seis razones siguientes de consistencia de módulo (de la menos a la más deseable):

Casual
Lógica
Temporal
De procedimiento
De comunicación
Funcional

A continuación se da una breve descripción de ellas.

Consistencia casual

Existe consistencia casual si no hay ninguna relación de significados entre las partes de un módulo. Generalmente, ocurre cuando se modulariza código existente. La modularización generalmente se hace buscando secuencias de comandos que aparecen varias veces y reemplazando esas secuencias por módulos. Generalmente, tales módulos no se relacionan con funciones del sistema, ya que son resultado de las técnicas utilizadas para escribir el programa.

Consistencia lógica

La consistencia lógica tiene lugar cuando todos los elementos de un módulo tratan tareas similares. Por ejemplo, módulos que incluyen toda la edición o que incluyen todos los accesos a un fichero.

En el nivel de consistencia lógica puede existir considerable duplicación. Por ejemplo, pueden existir comprobaciones de ediciones similares en más de un elemento de datos –así, más de un elemento en una transacción de entrada podría ser una fecha-. Se escribirían códigos separados para comprobar si cada una de esas fechas es una fecha válida. Un camino mejor es construir un módulo **comprobar-fecha** y llamar a ese módulo cuando se necesite comprobar una fecha.

Consistencia temporal

La consistencia temporal es muy parecida a la lógica. Todas las funciones relacionadas con el tiempo se agrupan en un módulo. Ejemplos típicos son **comienzo** y **terminación** de módulos.

La consistencia temporal parece más fuerte que la lógica, pero todavía es una característica indeseable en lo concerniente a un cambio. Por ejemplo, añadir un nuevo fichero a un sistema implicaría

cambios en los dos módulos, además de en aquellos módulos que estén relacionados con operaciones sobre ese nuevo fichero.

Consistencia de procedimiento

La consistencia de procedimiento generalmente ocurre cuando un diagrama (mapa o grafo) de flujo se divide en un número de secciones y cada sección se representa en un módulo. Esta división puede que no sea ideal, pues el diagrama de flujo puede representar una función bien definida del sistema; las divisiones distribuyen estas funciones autocontenidas en varios módulos.

Consistencia de comunicación

La consistencia de comunicación tiene lugar cuando se incluyen en un mismo módulo aquellos procesos que se comunican. Así, todas las acciones relacionadas con un fichero se pueden incluir en el mismo módulo. Este módulo leerá el fichero, lo procesará y escribirá los resultados de nuevo en el fichero.

El problema más conocido de la consistencia de comunicación es la interdependencia de procesos en el módulo. Por ejemplo, se pueden incluir códigos que utilicen la entrada de un fichero en el mismo módulo como el código que lee el fichero. Sin embargo, la "lectura" del fichero y el uso de la información de ese fichero suceden en diferentes tramas de tiempo y esto permite compartir buffers comunes. Un cambio que permite "lecturas" concurrentes y "escrituras" concurrentes de un fichero puede ser un problema inesperado.

Consistencia funcional

Un módulo que tiene consistencia funcional contiene una función bien definida. Estos módulos no tienen las propiedades que caracterizan a las coincidencias casual, lógica, temporal, de procedimiento o de comunicación.

Además, algunos escritores definen consistencia secuencial y consistencia de información. La consistencia secuencial tiene lugar cuando las salidas de los elementos de un módulo sirven como entrada a otros elementos en un módulo. En términos de diagrama de flujos de datos, la consistencia secuencial combina una cadena de transformaciones sucesivas en el dato. La consistencia de información existe cuando los módulos tratan de funciones múltiples, estando cada función representada por un punto diferente de entrada al módulo. Cada una de tales entradas tiene las características de un módulo de consistencia funcional. Las consistencias secuencial y de información están entre las consistencias funcional y de comunicación en la escala de consistencia de módulos.

MORFOLOGÍA

El acoplamiento y la cohesión son criterios aplicados localmente para unos pocos módulos. Además de estos criterios locales, los mapas de estructura se pueden evaluar considerando su estructura total. Algunos criterios utilizados para esta evaluación son las reglas ligaduras de control, fan-in, espacios de control y espacios de efecto.

Ligadura de control

La ligadura de control comprende el número de subordinados inmediatos de un módulo. En la figura 52.5 los módulos B, C, D, E y F son los subordinados inmediatos del módulo A. Luego, la ligadura de control del módulo A es cinco. Idealmente, la ligadura de control no debe exceder de siete.

Fan-in (números de entradas a un módulo)

El fan-in es el número de módulos que llaman a un módulo en particular. Idealmente, los mapas de estructura deberían tener un fan-in de cinco. Esto significa que se han identificado las funciones autocontenidas que se pueden usar varias veces.

Espacios de control

El espacio de control comprende todos los subordinados de un módulo. Incluye los subordinados inmediatos de un módulo, sus inmediatos subordinados, etc.

Espacio de efecto

El espacio de efecto de una decisión consta de todos los módulos cuyo procesamiento es condicional según los resultados de la decisión.

Un buen diseño debe reducir los efectos de una decisión a tan pocos módulos como sea posible. Si se hace esto, entonces las comprobaciones que se basen en una decisión no se repetirán innecesariamente o los resultados de una decisión no pasarán a través de un número excesivo de módulos.

Es deseable que la decisión se ajuste tanto como sea posible a los módulos que la usan. Una forma de llegar a esto es tener solamente subordinados inmediatos en el espacio de efecto de una decisión.

ALGUNAS ESTRUCTURAS COMUNES

Generalmente, los mapas de estructura se caracterizan por algunas construcciones que tienden a repetirse en muchas aplicaciones. Dos de las más importantes son *transformación centralizada* y **transacción centralizada**.

Estructuras de transformación centralizada

Las estructuras de transformación centralizada reciben una entrada que se transforma por una secuencia de operaciones con cada operación producida en un módulo. La figura 54.6 es una de estas estructuras donde:

- El módulo **principal** llama al módulo **obtener-X** para obtener el parámetro de entrada, X. El módulo **obtener-X** llama al módulo **obtener-Y** para leer un valor desde el dispositivo de entrada y devolverlo a **obtener-X** como parámetro Y. **Obtener-X** llama entonces al módulo **cambiar-Y** para calcular un valor de X a partir del valor de Y.

- Se llama entonces un módulo (T1) para calcular un valor de A a partir del valor de X.

- Otro módulo T2 se llama después para calcular un valor de B a partir del valor de A. Para hacer esto, el módulo T2 llama a sus módulos subrutinas, **calcular-VI** y **calcular-B.**

- El módulo **principal** llama entonces al módulo **poner-B** para sacar el resultado del cálculo. El módulo **poner-B,** primero llama al módulo **calcular-C** para calcular un valor de C y entonces llama al módulo **poner-C** para sacar ese valor particular de C.

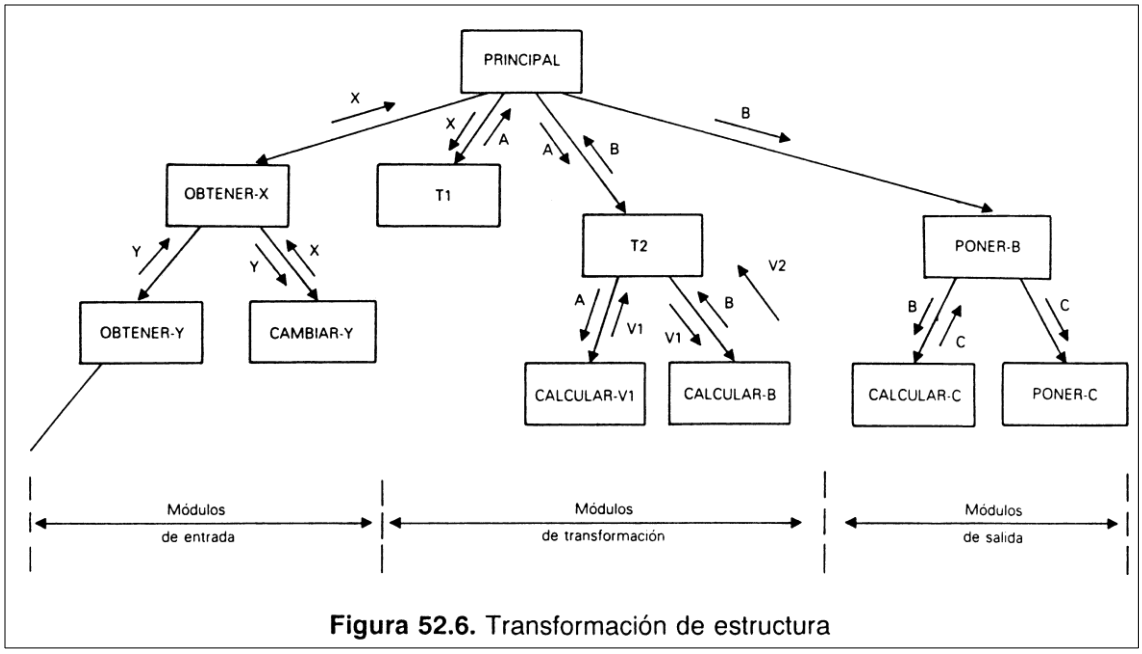

Figura 52.6. Transformación de estructura

Estructuras de transacción centralizada

Una estructura de transacción centralizada describe un sistema que procesa un número de tipos diferentes de transacciones. Esto se ilustra en la figura 52.7, donde el módulo PRINCIPAL controla las operaciones del sistema. Estas funciones son:

- Llamar al módulo **entrada** para leer una transacción.

- Determinar el tipo de transacción y seleccionar un módulo de un número de transacciones para procesar esa transacción, y

- Sacar los resultados del proceso llamando al módulo **salida.**

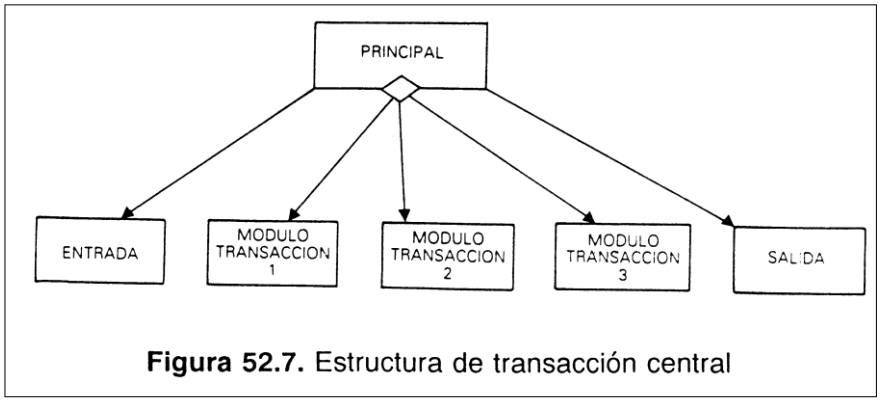

Figura 52.7. Estructura de transacción central

Generalmente se pueden encontrar en el mismo mapa de estructura, estructuras de transacción y transformación centralizada. Por ejemplo, la estructura de transacción centralizada lee una transacción y entonces llama a otro módulo para procesar esa transacción. Este módulo puede ser un módulo **principal** de una estructura de transformación centralizada.

5. CONVERSIÓN DE DIAGRAMAS DE FLUJO DE DATOS A MAPAS DE ESTRUCTURA

Ahora queda desarrollar técnicas para convertir una especificación propuesta en mapa de estructura. El mayor trabajo en este área se ha asociado con los diagramas de flujo de datos y su conversión a mapas de estructura. Las técnicas de diseño propuestas para esto se llaman *análisis de transformación* y *análisis de transacción.*

El origen de estas técnicas se remonta de nuevo al trabajo de Constantin en IBD, este trabajo ha sido sustancialmente reelaborado más tarde por Myers (1975) y Yourdon, y Constantine (1979).

Ambos análisis comienzan generando un mapa de estructura inicial a partir de un diagrama de flujo de datos. Después, ese mapa inicial se refina para producir el mapa de estructura final.

Los análisis de transformación y transacción convierten cada proceso del DFD en un módulo del mapa de estructura y conectan esos módulos de forma consistente con los flujos de datos del DFD. La naturaleza de estas conexiones reside en las ideas de las estructuras de transformación y de transacción. Se deben identificar en el DFD los flujos que corresponden a estructuras de transformación y transacción y convertir esos flujos en tales estructuras.

Análisis de Transformación

El análisis de transformación examina en el DFD los procesos que se pueden convertir a transformaciones centralizadas de un mapa de estructura. Busca el proceso central junto con las corrientes bien definidas de entrada y salida. Uno de tales procesos es el proceso F2 de la figura 52.8. El siguiente paso es crear un módulo principal y otro módulo para el proceso central, entonces el módulo principal llama al módulo de procesamiento central (en este caso F2).

Luego se tienen que convertir los procesos que proporcionan la entrada al proceso F2 a módulos del mapa de estructura. Estos módulos proveerán la entrada al módulo **principal,** que lo pasará al módulo F2. Así, en la figura los módulos de entrada se crean para los procesos F0 y F1 del DFD. El proceso F0 se convierte en tres módulos. El módulo **obtener-A** que llama al módulo **obtener-Z** para leer un valor y pasarlo en el parámetro Z. Entonces llama al módulo F0 para calcular un valor de A a partir del valor de Z y pasa el valor de **A** al módulo **obtener-B.** El módulo **obtener-B y** el módulo F1 se derivan del proceso F1. El módulo **obtener-B** llama antes al módulo **obtener-A** para obtener un valor de A y entonces llamar al módulo F1 para calcular un valor de B a partir del valor de A. B se pasa entonces al módulo **principal.**

Finalmente, los procesos del DFD que toman las salidas de F2 se convierten en módulos de salida que obtienen salidas del módulo **principal** y las entrega a un dispositivo de salida. En la figura el proceso F3 del DFD se convierte en los módulos **poner-C,** F3 y **poner-E.** El módulo **principal** llama al módulo **poner-C,** al cual le pasa un valor de C. El módulo **poner-C** llama al módulo F3 para calcular un valor de E a partir del valor de C. Después llama al módulo **poner-E** para sacar el valor de E. De forma similar, el proceso F4 se convierte en los módulos **poner-D,** F4 y **poner-F.**

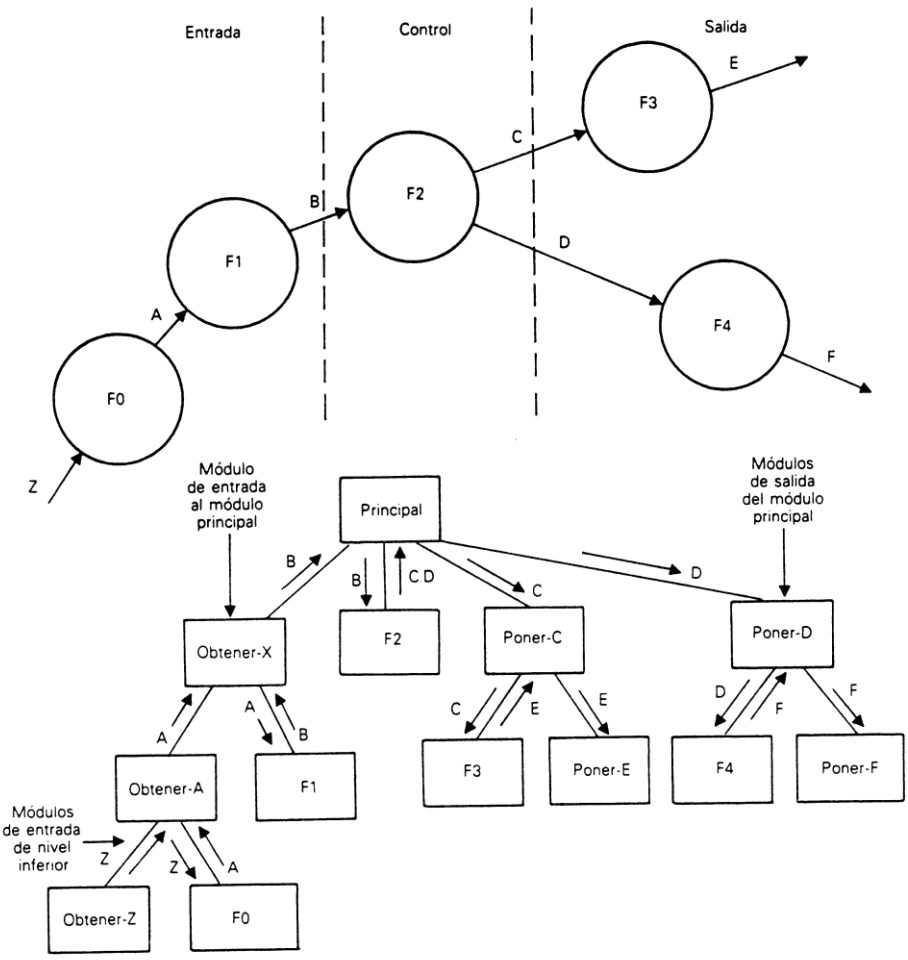

Figura 52.8. Análisis de transformación

Análisis de Transacción

El análisis de transacción decide sobre la identificación de aquellas partes del diagrama de flujo de datos que se pueden convertir en mapas de estructura de transacción centralizada. En el diagrama de flujo de datos se busca una corriente de entrada que se divida en varias corrientes de entrada para un proceso. Uno de tales ejemplos es el proceso T1 de la figura 52.9. El proceso T1 recibe la entrada P y produce tres flujos de datos de salida etiquetados Q, R y S. Cada uno de estos flujos se transforma y recombina en un solo flujo de datos W, por el proceso T5.

El diagrama de flujo de datos de la figura se convierte en el mapa de estructura de la misma. El proceso T1 se convierte en el módulo **principal** de la figura. Cada uno de los procesos de transacción T2, T3 y T4 se convierte en los módulos de mapa de estructura T2, T3 y T4. Estos tres módulos son llamados por el módulo **principal.**

El proceso de entrada T0 se convierte en los módulos de entrada, **obtenerP, obtener-N** y T0. El módulo **principal** llama al módulo **obtener-P** para obtener el valor de P. Entonces, éste determina el tipo de transacción en P. Dependiendo de la transacción se activará uno de los tres módulos, T2, T3 o T4. Las salidas de esos módulos de transacción se devuelven al módulo **principal** que inicia la salida llamando al módulo **poner-W.** El módulo **poner-W** ha derivado del proceso T6. El proceso T6 se ha convertido en los módulos **poner-W,** T6 y **poner-X.** El módulo **principal** pasa el valor de W al módulo **poner-W.** El módulo **poner-W** llama al módulo T6 para calcular un valor de X a partir del valor de W y entonces llama al módulo **poner-X** para sacar el valor de X.

Se debe recordar que los análisis de transformación y transacción son el primer paso para un mapa de estructura. Generalmente es necesario mucho refinamiento para producir un mapa de estructura que tenga módulos con buen acoplamiento y alta cohesión.

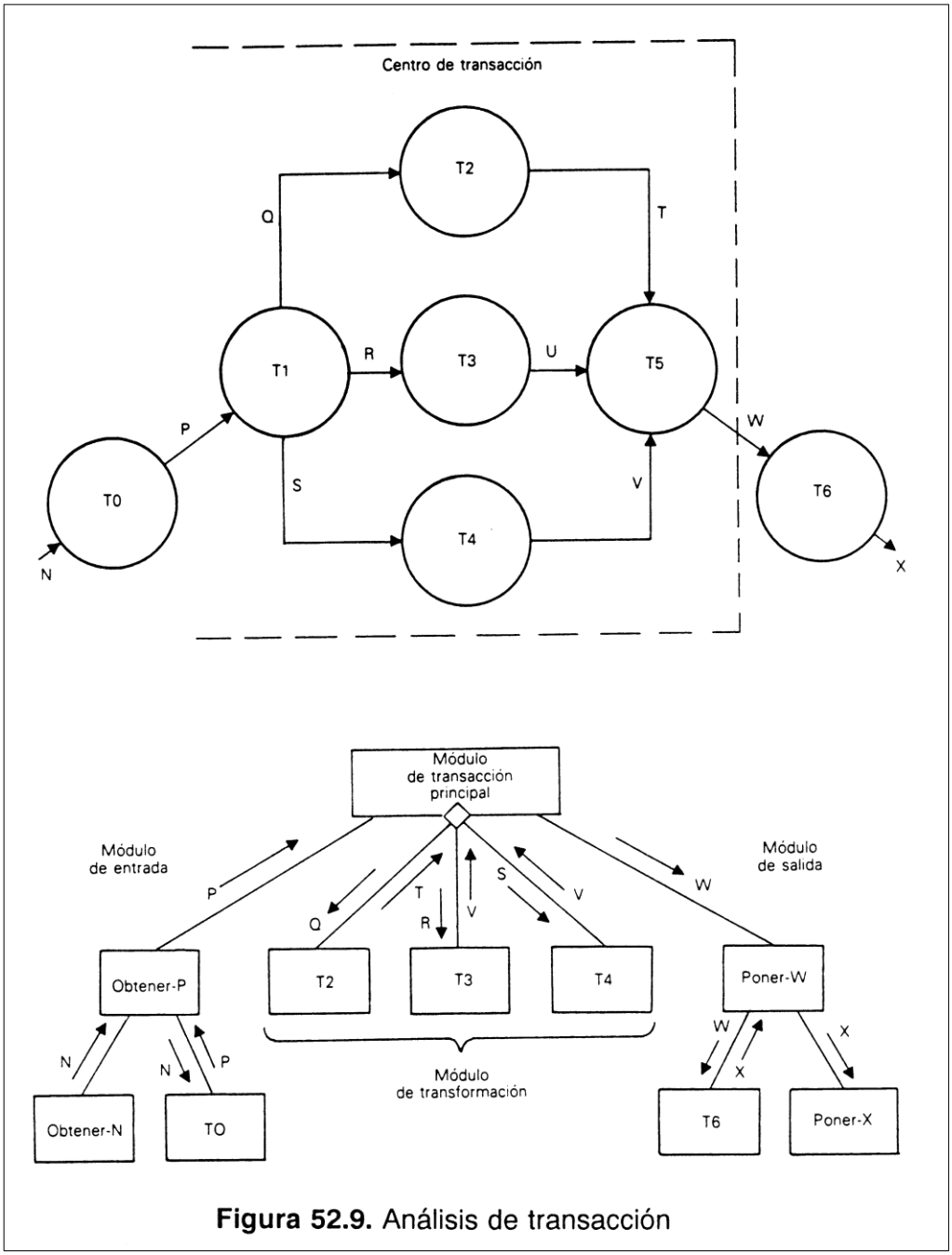

Figura 52.9. Análisis de transacción

6. BIBLIOGRAFÍA

Pressman, Roger S.
Ingeniería del Software
Mc Graw-Hill, 1993

López, Antonio
Metodologías de Desarrollo
Ra-ma, 1990