Capítulo

5

Técnicas Generales de Diseño Software

Centro Asociado Palma de Mallorca

Tutor: Antonio Rivero Cuesta

5 Técnicas Generales de Diseño Software

5.1 Introducción.

Diseñar software requiere cierta experiencia, el conocimiento de las técnicas de diseño es esencial para poder abordar la tarea con éxito.

5.2 Objetivos

Descomposición modular y decisión sobre algoritmos y estructuras de datos fundamentales.

5.3 Descomposición modular.

Para lograr una descomposición modular es necesario concretar los siguientes aspectos:

- Identificar los módulos.
- Describir cada módulo.
- Describir las relaciones entre módulos.

Podemos decir que con carácter general, un módulo es un fragmento de un sistema software que se puede elaborar con relativa independencia de los demás. La idea básica de esto, es que la elaboración de los diferentes módulos se pueda encargar a personas distintas y que todas ellas trabajen en paralelo.

Dependiendo de lo que se entienda por módulo tenemos la siguiente clasificación:

Código fuente: Es el considerado como tal con mayor frecuencia, contiene el texto fuente escrito.

Tabla de datos: Se utiliza para tabular ciertos datos de inicialización experimentales o de difícil obtención.

Configuración: Agrupamos en un mismo módulo toda aquella información que permite configurar el entorno concreto de trabajo.

Otros: Un módulo sirve para agrupar ciertos elementos del sistema relacionados entre sí y que se pueden tratar de forma separada del resto.

Solo en casos excepcionales se sacrificará el objetivo de tener un sistema mantenible para lograr una mayor velocidad de proceso o un menor tamaño de código.

Una descomposición modular debe poseer ciertas cualidades mínimas para que se pueda considerar suficientemente válida.

5.3.1 Independencia funcional

Para que un módulo posea independencia funcional debe realizar una función concreta o un conjunto de funciones afines, sin apenas ninguna relación con el resto de los módulos del sistema.

El mayor grado de independencia se consigue cuando no existe ninguna relación entre los distintos módulos. Evidentemente al descomponer un sistema en sus módulos es necesario que existan ciertas relaciones entre ellos. En todo caso se trata de reducir las relaciones entre módulos al mínimo.

A mayor independencia mayor mantenibilidad y reutilización del módulo.

Para medir la independencia funcional entre varios módulos tenemos dos criterios, Acoplamiento y Cohesión.

Acoplamiento: el grado de acoplamiento entre módulos es una medida de la interrelación que existe entre ellos. Tenemos varios tipos de acoplamiento que pueden ser fuerte., moderado, débil, y en ese orden son: (Ver esquema página 161).

Acoplamiento por Contenido (fuerte): se produce cuando desde un módulo se pueden cambiar los datos locales e incluso el código de otro módulo. En este caso no existe una separación real entre módulos y hay solapes entre ellos.

Este tipo de acoplamiento solo se puede lograr utilizando un lenguaje ensamblador o de muy bajo nivel y debe ser evitado siempre.

Ver figura 5.1.a Pag.161.

Acoplamiento Común (fuerte): en este caso se emplea una zona común de datos a la que tienen acceso varios o todos los módulos del sistema.

Cada módulo puede estructurar y manejar la zona común con total libertad sin tener en cuenta al resto de módulos.

El empleo de este acoplamiento exige que todos los módulos estén de acuerdo en la estructura de la zona común y cualquier cambio adoptado por uno de ellos afecta al resto que deberían ser modificados según la nueva estructura. Esta tipo de acoplamiento también debe ser evitado siempre.

Ver figura 5.1.b Pag.161.

Acoplamiento Externo (fuerte): aquí la zona común está constituida por algún dispositivo externo al que están ligados todos los módulos.

En este caso la estructura de la zona común la impone el formato de los datos que maneja el dispositivo y cualquier modificación exige el cambio de todos los módulos.

Acoplamiento de control (moderado): una señal o dato de control que se pasa desde un módulo A a otro B es lo que determina la línea de ejecución que se debe seguir dentro de B.

Ver figura 5.2. Pag.162.

Acoplamiento por etiqueta (débil): se produce cuando en el intercambio se suministra una referencia que facilita el acceso no solo a los datos estrictamente necesarios sino también a la estructura completa de la que forman parte, por ejemplo, vector, pila, etc.

Acoplamiento de datos (el más débil): Los módulos sólo se intercambian los datos que únicamente necesitan. **Este es el mejor tipo posible de acoplamiento.**

Ver figura 5.3.a Pag.163.

Estos dos tipos de acoplamiento débil son los más deseables en una descomposición modular. Con ellos, cualquier modificación en un módulo afecta muy poco o nada al resto.

Evidentemente el acoplamiento más débil es el que no existe. Este caso es el que se produce entre los módulos E y B de la figura 5.3 de la página 163, entre los que no existe ningún tipo de acoplamiento directo.

En resumen, el objetivo es conseguir el acoplamiento más débil posible.

Cohesión: hace referencia a que el contenido de cada módulo ha de tener la mayor coherencia posible.

Tenemos varios tipos que pueden ser baja, media, alta, y en ese orden son: (Ver esquema página 164).

Cohesión coincidental (la más baja): se produce cuando los elementos del módulo no guardan absolutamente ninguna relación entre ellos.

La existencia de este tipo de módulos indica que no ha sido realizada ninguna labor de diseño y que simplemente se ha efectuado un troceado del sistema.

Es la peor posible y debe evitarse siempre.

Cohesión lógica (baja): se produce cuando se agrupan en un módulo elementos que realizan funciones similares desde un punto de vista del usuario. Por ejemplo, un módulo de funciones matemáticas.

Cohesión temporal (baja): se agrupan en el mismo módulo elementos que se ejecutarán en el mismo momento. Esta es la situación que se produce en la fase de inicialización o finalización del sistema en que necesariamente se deben "arrancar" o "parar" dispositivos completamente heterogéneos, teclado, pantalla, ratón, etc.

Cohesión de comunicación (media): se produce cuando todos los elementos del módulo operan con el mismo conjunto de datos de entrada o producen el mismo conjunto de datos de salida.

Cohesión secuencial (media): se produce cuando todos los elementos del módulo trabajan de forma secuencial. Es decir, la salida de un elemento del módulo es la entrada del siguiente de una manera sucesiva.

***Cohesión funcional*:** cada elemento está encargado de la realización de una función concreta y especifica.

Cuando se utilicen técnicas de Diseño Funcional Descendente este nivel de cohesión será el máximo que se puede lograr dentro de un módulo.

Cohesión abstraccional: se logra cuando se diseña un módulo como tipo abstracto de datos con la técnica basada en abstracciones o como una clase de objetos con la técnica orientada a objetos. En ambos casos se asocia una organización de datos con las correspondientes operaciones encargadas de su manejo.

Con la técnica de diseño basado en abstracciones, un módulo con cohesión funcional sería una abstracción funcional.

La cohesión Baja debe evitarse siempre.

Con una cohesión Media se puede reducir el número de módulos. Sin embargo, esto no se debe realizar a costa de aumentar el grado de acoplamiento entre módulos. Por ejemplo, no se deben unir dos módulos con acoplamiento Débil para obtener uno único con acoplamiento Moderado, en el que se requiere pasar una señal para indicar con cuál de los dos antiguos submódulos se quiere trabajar.

Una cohesión Alta debe ser el objetivo que se debe perseguir en cualquier descomposición modular. Con ello se facilitara el mantenimiento y la reutilización de los módulos así diseñados.

Para conocer y mejorar la cohesión de un módulo se suele realizar una descripción de su comportamiento a partir de la cual, se puede establecer el grado de cohesión. Se tienen en cuenta los siguientes criterios:

Si la descripción es una frase compuesta que contiene comas o más de un verbo es muy probable que se esté incluyendo más de una función y la cohesión será Media de tipo Secuencial o de Comunicación.

Si la descripción contiene palabras relacionadas con el tiempo tales como "primero", "después", "entonces", la cohesión será de tipo temporal o secuencial.

Si la descripción no se refiere a algo especifico a continuación del verbo, es muy probable que tengamos una cohesión lógica. Por ejemplo "escribir todos los mensajes de error" o "calcular todas las funciones trigonométricas".

Si se utilizan palabras como “inicializar”, “preparar”, "configurar", etc., la cohesión será probablemente de tipo temporal.

En resumen, la descomposición modular con una mayor independencia funcional se logra con un acoplamiento DÉBIL entre sus módulos y una cohesión ALTA dentro de cada uno de ellos.

5.3.2 Comprensibilidad

La dinámica del proceso de diseño e implementación de un sistema hace que los cambios sean frecuentes. A menudo estos cambios deben ser realizados por personas que no participaron ni en el diseño ni en la implementación. Para facilitar estos cambios es necesario que cada módulo sea comprensible de forma aislada.

El primer factor que facilita la comprensión de un módulo es su independencia funcional. Sin embargo, es necesario además cuidar los siguientes factores:

***Identificación*:** una elección adecuada del nombre del módulo y de los nombres de cada uno de sus elementos facilita mucho su comprensión. Los nombres deben reflejar de manera sencilla el objetivo de la entidad que representan.

***Documentación*:** deben ser aclarados todos los aspectos de diseño o implementación con la documentación apropiada.

***Simplicidad*:** los algoritmos sencillos son los mejores. El diseñador debe simplificar al máximo las soluciones propuestas.

5.3.3 Adaptabilidad

Al diseñar un sistema se pretende resolver un problema concreto.

Por tanto, la descomposición modular estará muy unida al objetivo concreto del diseño. Esto dificulta la adaptabilidad del diseño a otras necesidades y la posible reutilización de algunos de sus módulos.

Las adaptaciones suelen ser múltiples y variadas a lo largo de la vida del sistema. Así, es necesario cuidar otros factores adicionales para facilitar la adaptabilidad y de ellos destacaremos los siguientes:

***Previsión*:** los módulos que se prevén que puedan cambiarse se deben agrupar en módulos con un acoplamiento lo más débil posible con el resto de los módulos. Así, las adaptaciones se podrán realizar con correcciones que solo afectaran a los módulos previstos.

***Accesibilidad*:** para poder adaptar un sistema debe ser sencillo acceder a todos los documentos de especificación, diseño e implementación. Esto requiere una organización minuciosa que se hará normalmente con herramientas CASE.

***Consistencia*:** cuando se modifican los programas fuente se deben modificar todos los documentos implicados. Esto se puede hacer automáticamente con herramientas para el control de versiones y configuración. En los entornos orientados a objetos no existen estas herramientas por lo que se debe imponer una disciplina férrea dentro de la biblioteca.

5.4 Técnicas de diseño funcional descendente.

La descomposición del sistema se hace desde un punto de vista funcional. Se van descomponiendo la función del sistema en funciones más sencillas, las cuales se encomiendan a módulos separados.

5.4.1 Desarrollo por refinamiento progresivo

Es la programación estructurada, en la que se emplean estructuras de control claras y sencillas como secuencia, selección e iteración. Se plantea el programa como una operación global única para ir descomponiéndola en otras operaciones más sencillas.

La aplicación de esta técnica a la fase de diseño consiste en realizar solo los primeros niveles de refinamiento, asignando a módulos separados las operaciones parciales que se van identificando.

5.4.2 Programación estructurada de Jackson (JSP)

Es similar a la programación estructurada, la diferencia está en las recomendaciones para ir construyendo la estructura del programa, que debe hacerse similar en lo posible a las estructuras de los datos de entrada y salida.

Los pasos de esta técnica son:

- **Analizar** el entorno del problema y describir las estructuras de datos a procesar.
- **Construir** la estructura del programa basada en las estructuras de datos.
- **Definir** las tareas a realizar en términos de las operaciones elementales disponibles y situarlas en los módulos apropiados de la estructura del programa.

Como técnica de diseño los pasos significativos son los dos primeros, mientras que el tercero corresponde a la fase de codificación.

Ver ejemplo páginas 171, 172 y 173.

Diseño estructurado

La tarea de diseño consiste en pasar de los DFD a los diagramas de estructura. La dificultad reside en que hay que establecer una jerarquía entre los diferentes módulos que no se ve en los DFD.

Ver ejemplo páginas 174.

A veces se usa un módulo de coordinación para construir esta jerarquía.

Para establecer una jerarquía de control razonable entre las diferentes operaciones descritas en los diagramas de flujo de datos, la técnica de diseño estructurado recomienda realizar los análisis denominados de Flujo de Transformación y de Flujo de Transacción.

Análisis de flujo de transformación: se identifica el flujo global de información desde los elementos de entrada al sistema, hasta los de salida.

Los procesos se deslindan en tres regiones, flujo de entrada, de transformación y de salida. Para obtener la estructura modular del programa se asignan módulos para las operaciones del diagrama y se añaden módulos de coordinación que realizan el control de acuerdo con la distribución del flujo de transformación.

Ver figura 5.8 página 175.

Análisis del flujo de transacción: se puede aplicar cuando el flujo de datos se puede descomponer en varias líneas separadas, cada una de las cuales corresponde a una función global o transacción distinta, de manera que solo una de estas líneas se activa para cada entrada de datos de tipo diferente. El análisis consiste en identificar el llamada Centro de Transacción del que salen las líneas de flujo y las regiones correspondientes a cada una de esas líneas o transacciones.

Ver figura 5.9 página 176. Ver ejemplo página 178.

5.5 Técnicas de diseño basado en abstracciones.

La idea general es que los módulos se correspondan o bien con funciones o bien con tipos abstractos de datos.

5.5.1 Descomposición modular basada en abstracciones

Consiste en ampliar el lenguaje existente con nuevas operaciones y tipos de datos definidos por el usuario. Se dedican módulos separados para cada tipo abstracto de datos y cada función importante. Se puede aplicar:

De forma descendente: es como el refinamiento progresivo. En cada paso la operación a refinar se define separadamente como abstracción funcional o como tipo abstracto de datos.

De forma ascendente: se van ampliando primitivas existentes en el lenguaje de programación y librerías con nuevas operaciones y tipos de mayor nivel, más adecuados para el campo de aplicación que se está diseñando.

Ver ejemplo página 179, 180.

5.5.2 Método de Abott

Con este método se sugiere una forma metódica de conseguir aquellos elementos del modelo del sistema que son buenos candidatos para ser considerados como abstracciones, a partir de las descripciones del sistema hechas en lenguaje natural.

Se procede así:

Identificar en el texto de la descripción los tipos de datos como sustantivos, los atributos como sustantivos y a veces como adjetivos, las operaciones como verbos o como nombres de acciones.

Hacer dos listas: una con nombres y otra con verbos.

Reorganizar las listas extrayendo los posibles datos y asociándoles sus atributos y operaciones; además eliminar los sinónimos y añadir los elementos implícitos en la descripción.

Para **obtener el diseño** asignamos un módulo a cada abstracción de datos o grupo de abstracciones relacionadas entre sí.

El módulo puede corresponder a un dato encapsulado si solo se maneja un dato de ese tipo en todo el programa.

Este método se puede usar tanto en diseño basado en abstracciones como en diseño orientado a objetos.

Ver ejemplo páginas 180 - 185.

5.6 Técnicas de diseño orientadas a objetos.

El diseño orientado a objetos es similar al diseño basado en abstracciones, solo que añadiendo la herencia y el polimorfismo.

Cada módulo contendrá la descripción de una clase de objetos o de varias relacionadas entre si. Además del diagrama modular, nos apoyamos en diagramas ampliados del modelo de datos, como los diagramas de estructura.

5.6.1 Diseño orientado a objetos

Se basa en los siguientes pasos:

Estudiar y comprender el problema**:** esto debe haberse realizado en la fase de Análisis de requisitos. Sin embargo, puede ser necesario repetirlo en parte porque la especificación no sea suficientemente precisa, o porque el diseño va a ser realizado por personas diferentes de las que confeccionan la especificación.

Desarrollar una posible solución**:** es posible que se haya realizado en la ***fase de análisis***, aunque es más probable que se tenga que hacer o completar en la ***fase de diseño***. Se consideran varias alternativas y se elegirá la que se considere más apropiada.

Identificar las Clases y Objetos**:** puede hacerse siguiendo la técnica de Abbott. Se identifican las clases de objetos y sus atributos. Si un objeto contiene otros objetos, no se suelen tratar como atributos, sino que se establece una relación de composición entre el objeto compuesto y los objetos componentes. Tras este primer paso puede ya confeccionarse un diagrama inicial del modelo de objetos.

Identificar las operaciones sobre los objetos**:** también puede hacerse siguiendo la técnica de Abbott. Además de identificar las operaciones hay que decidir a qué objeto o clase se asocia. Esto puede ser un problema de Diseño no trivial. Por ejemplo: la operación de escribir una fecha en pantalla con caracteres puede asociarse al objeto Fecha, o bien al objeto Pantalla. Cada decisión tiene sus ventajas e inconvenientes.

En algunos casos puede fraccionarse la operación en varias más sencillas. Por ejemplo, se puede definir una operación de conversión de fecha a texto, sobre el objeto Fecha, y otra operación de escritura del texto sobre el objeto Pantalla. Las operaciones pueden reflejarse sobre el diagrama de modelo de objetos.

Aplicar Herencia**:** una vez identificados los objetos y sus operaciones asociadas, hay que detectar analogías entre ellos, si las hay, y establecer las relaciones de herencia apropiadas. Estas relaciones de herencia se incluirán en el diagrama de modelo de objetos que se va desarrollando.

Describir las Operaciones**:** esta es una manera de verificar que el diseño es consistente. Cada operación se describe en pseudocódigo, haciendo referencia a operaciones o clases de datos definidos en este mismo diseño, o bien predefinidos en el lenguaje de programación a usar. En caso necesario habrá que añadir nuevas operaciones a las ya identificadas, o incluso nuevas clases de objetos, y se actualizara el modelo de objetos.

***Establecer la estructura modular*:** hay que asignar clases, objetos y operaciones a módulos separados. En principio se intentara que cada módulo corresponda a una clase de objetos (o a un objeto en particular). Si el módulo es demasiado complicado, ciertas operaciones pueden establecerse como módulos separados . También es posible agrupar en un solo módulo varios objetos o clases muy relacionados entre sí, para mejorar las características de acoplamiento y cohesión.

Como resultado de esta etapa se obtendrá el diagrama de estructura del sistema.

Una vez llegado a este punto hay que analizar si el diseño modular resultante es apropiado para pasar a la fase de codificación. Si algún módulo es todavía demasiado complejo, o no está definido con suficiente precisión, se repetirán los pasos anteriores de diseño para ese módulo, con objeto de refinarlo.

Ver ejemplo pagina 188 - 195.

5.7 Técnicas de diseño de datos.

La mayoría de las aplicaciones informáticas requieren almacenar información de forma permanente. La forma típica de hacerlo es apoyando la aplicación en una base de datos.

En la organización de la base de datos se pueden ver ***tres niveles***:

Nivel Externo: corresponde a la visión del usuario. La organización de los datos se realiza siguiendo esquemas en el campo de la aplicación. Por ejemplo, en forma de ficha de cliente.

Nivel Conceptual: establece una organización lógica de los datos, a través de un diagrama de modelo de datos, bien E-R, bien diagrama de Modelo de objetos.

Nivel Interno: organiza los datos según los esquemas del gestor de base de datos;

Si es una base de datos relacional serian tablas.

El paso del nivel Externo al Conceptual se puede realizar durante la etapa de análisis de requisitos.

El paso del nivel Conceptual al nivel interno se puede realizar durante la fase de Diseño.

Ver esquema página 196.

5.8 Diseño de bases de datos relacionales.

Es posible dar reglas prácticas para obtener los esquemas de las tablas de una base de datos relacional que reflejen la visión lógica de los datos, y que sean eficientes.

En el modelo relacional la eficiencia se ve desde dos puntos de vista, que son ***Formas Normales*** para evitar las redundancias, **Índices** para mejorar la velocidad de acceso a los datos.

5.8.1 Formas normales

Las Formas Normales de Codd definen criterios para establecer esquemas de tablas que sean claros y no redundantes. Estos criterios se numeran correlativamente, de menor a mayor nivel de restricción, dando lugar a las formas normales 1ª, 2ª, 3ª, etc. Una tabla que cumpla con una cierta forma normal cumple también con las anteriores.

***1ª Forma Normal*:** se dice que una tabla se encuentra en 1ª Forma Normal si la información asociada a cada una de las columnas es un valor único y no una colección de valores de número variable.

***2ª Forma Normal*:** se dice que una tabla se encuentra en 2ª Forma Normal si esta en 1ª Forma Normal y además hay una Clave Primaria que distingue cada fila; además cada casilla que no sea de la clave primaria depende de toda la clave primaria.

***3ª Forma Normal*:** se dice que una tabla se encuentra en 3ª Forma Normal si esta en 2ª forma normal y además el valor de cada columna que no es Clave Primaria depende directamente de la Clave Primaria, esto es, no hay dependencias entre columnas que no son Clave Primaria.

Ver ejemplo páginas 196, 197, 198.

5.8.2 Diseño de las entidades

Cada entidad del modelo E-R se traduce en una tabla por cada clase de entidad. Cada elemento de esa clase es una fila y Cada atributo de esa entidad es una columna

Si una entidad está relacionada con otras, y se quiere tener una referencia rápida entre las entidades relacionadas, se puede incluir una columna con un número de referencia que identifique cada fila de la tabla.

El número o código de referencia si se usa sirve como Clave Primaria.

5.8.3 Tratamiento de las relaciones de asociación

En el modelo de objetos se tienen dos tipos especiales de relación, las de composición o agregación y las de herencia o especialización. Las demás son las de asociación. En el modelo E-R todas las relaciones se consideran relaciones de asociación.

La manera de almacenar en tablas la información de las relaciones de asociación depende de la cardinalidad de la relación.

La técnica general válida para todas las cardinalidades es traducir la relación a una tabla conteniendo referencias a las tablas de las entidades relacionadas, así como los atributos de la relación si los hay.

La referencia a las entidades relacionadas se hará mediante la clave primaria de cada una.

Ver figura 5.20 (a), página 200.

Si la cardinalidad es 1-N: se incluyen los datos de la relaciones en la misma tabla de una de las entidades relacionadas.

Ver figura 520 (b), página 200.

Si la cardinalidad es 1-1: se pueden fundir las tablas de las dos entidades en una sola. Ver figura 5.20 (c), página 200.

5.8.4 Tratamiento de las relaciones de composición

La cardinalidad del lado del objeto compuesto es casi siempre 1. Se aplican los mismos criterios anteriores.

5.8.5 Tratamiento de la herencia

Cuando una clase tiene varias subclases hay tres formas de almacenar en tablas la información de las entidades.

1º: se usa una tabla para la superclase, con los atributos comunes heredados por las subclases, mas una tabla por cada subclase con sus atributos específicos.

2ª: se repiten los atributos comunes en las tablas de cada subclase, por lo que desaparece la tabla de la superclase.

3º: se amplía la tabla de la superclase con todos los atributos de cada una de las subclases, prescindiendo de las tablas de cada subclase.

Ver figura 5.21 página 202.

5.8.6 Diseño de índices

Permiten acceder rápidamente a un dato concreto, a costa de aumentar el espacio de almacenamiento y el tiempo de almacenamiento de nuevos datos y la modificación del valor de un atributo indexado.

Si hay que acceder a datos a través de sus relaciones con otros, es conveniente mantener índices sobre las Claves Primarias y columnas de referencia de las entidades relacionadas.

Ver ejemplo 201 - 204.

5.9 Diseño de bases de datos de objetos.

Hay una mayor variedad de estructuras disponibles pero distintas en de cada caso. Podemos ver dos enfoques en el diseño:

1º: cuando la base de datos de objetos permite usar una gran variedad de estructuras. En este caso, el sistema de gestión de base de datos aporta como complemento la persistencia de los datos.

2º: cuando no existe esa variedad de estructuras y la base de datos de objetos es análoga a una base de datos relacional. En este caso, el sistema de gestión de base de datos aporta la existencia implícita de identificadores de objetos, que hacen innecesarias las columnas explícitas de códigos o números de referencia.

Ver ejemplos páginas 210 a 232.