

Diseño lógico de funciones. Definición de funciones. Descomposición modular.
Técnicas descriptivas. Documentación

TEMA 52

ABACUS NT

Oposiciones 2021

Índice

- 1. Introducción**
- 2. Diseño Lógico de funciones.**
 - 2.1. Diseño estructurado.**
 - 2.2. Atributos de la calidad de un diseño.**
 - 2.3. Acoplamiento de módulos**
 - 2.3.1. Acoplamiento de contenido
 - 2.3.2. Acoplamiento de entorno común
 - 2.3.3. Acoplamiento de control
 - 2.3.4. Acoplamiento de datos
 - 2.4. Consistencia de módulos**
 - 2.4.1. Consistencia casual
 - 2.4.2. Consistencia lógica
 - 2.4.3. Consistencia temporal
 - 2.4.4. Consistencia de procedimiento
 - 2.4.5. Consistencia de comunicación
 - 2.4.6. Consistencia funcional
 - 2.5. Morfología**
- 3. Definición de funciones**
 - 3.1. Funciones.**
 - 3.2. Procedimientos.**
- 4. Descomposición modular**
 - 4.1. Módulos.**
 - 4.2. Divide y Vencerás**
 - 4.3. Diseño Top-Down (Diseño Descendente)**
- 5. Técnicas descriptivas**
 - 5.1. Pseudocódigo**
 - 5.2. Ordinograma**
- 6. Documentación**
- 7. Conclusión.**
 - 7.1. Relación con el sistema educativo**
- 8. Bibliografía**

1. Introducción

Una vez finalizada la fase de análisis del sistema, se procede a la implementación del mismo. Este proceso pasa por la modularización, especificación y documentación, con el objetivo de **crear programas fáciles de revisar**, ampliar y modificar con posterioridad.

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.

El diseño de programas comienza con los **DFD** y genera a la salida el **diseño arquitectónico del sistema, el diseño de base de datos, el diseño de procedimientos, diseño de interfaces**.

En este tema nos centraremos el **diseño procedural**.

2. Diseño Lógico de funciones.

El diseño procedural se realiza en tres etapas distintas

- Análisis del problema: realizamos la descomposición del problema en subproblemas menores siguiendo las directrices de la programación estructurada.
- Diseño lógico: Se especifica mediante pseudocódigo, ordinogramas u otra especificación el funcionamiento interno de cada módulo
- Diseño físico: Se implementa el módulo en un lenguaje concreto de programación

2.1. Diseño estructurado.

El objetivo principal del diseño estructurado es describir la estructura del programa, así como las relaciones entre los elementos, denominadas módulos, que componen esta estructura. Para ello, en Métrica v.3, se propone el uso de diagramas de cuadros de Constantine o diagramas de estructura.

Estos diagramas cumplen las siguientes características:

- Los módulos son representados por **rectángulos**, y su nombre representa a la función que realiza.
- Las conexiones entre módulos son representadas mediante **flechas**
- Los módulos pueden **comunicarse** entre sí por medio de estructuras de datos o de control (flags)
- Un **módulo “predefinido”** es aquel que se encuentra disponible en la biblioteca del sistema o de la aplicación, y es representado con un rectángulo dentro de otro.

2.2. Atributos de la calidad de un diseño.

Para medir la **calidad estructural** de un diseño se utilizan las siguientes métricas principalmente, las cuales son contrapuestas:

- **Acoplamiento:** Grado de interdependencia entre los módulos. Un buen diseño minimiza el acoplamiento al máximo.
- **Consistencia o cohesión:** Mide la relación que existe entre los elementos de un mismo módulo, es decir, las relaciones intramodulares. Por el contrario, mientras mayor es la cohesión de un módulo, mejor está diseñado. Los autores Stevens, Myers y Constantine desarrollaron una escala para medir la cohesión, cuyos valores ordenados de mayor a menor van desde la cohesión funcional hasta la coincidental.

2.3. 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.

2.3.1. 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.

2.3.2. 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.

2.3.3. 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.

2.3.4. 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.

2.4. 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.

2.4.1. 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.

2.4.2. 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.

2.4.3. 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.

2.4.4. 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 autocontenidoas en varios módulos.

2.4.5. 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.

2.4.6. 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.

2.5. 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. 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 autocontenidoas 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.

3. Definición de funciones

3.1. Funciones.

Subprograma u operación que puede tomar uno o más valores de entrada, llamados **argumentos**, y devuelve un único valor denominado **resultado** tras ejecutar un conjunto de **instrucciones**. Se **invoca** o referencia mediante un nombre y una lista de parámetros, y una vez ejecutada **retorna** al punto de llamada.

Ejemplo de definición de una función en Javascript:

```
function cuadrado(n) {
    return n * n;
}
```

Los argumentos de la declaración de la función se denominan **parámetros formales**, en este caso sería n. Por el contrario, los argumentos utilizados en la llamada de la función se denominan **parámetros actuales**, y pueden ser constantes, literales, variables o expresiones. A continuación, se muestra un ejemplo de invocación o llamada de la anterior función:

```
var acum= cuadrado(3);
```

3.2. Procedimientos.

Al igual que las funciones, son utilizados para definir **subprogramas**, identificando un conjunto de instrucciones con un **identificador**, sin embargo, atendiendo a su definición formal, se diferencian de éstas en:

- No devuelven ningún **valor asociado a su nombre**.
- No pueden ser utilizados dentro de **expresiones**.
- Además de parámetros de entrada, pueden tener **parámetros** de entrada/salida y de salida, por lo que pueden devolver varios valores en una misma llamada.

Ejemplo de definición de un procedimiento en Java

```
private void proc1(List<String> l1)
{
    l1.add("ejemplo");
}
```

4. Descomposición modular

4.1. Módulos.

La programación modular surge **como mejora de la programación estructurada**, con el objetivo de evitar la creación de programas de un único bloque de tamaño desmesurado, y así mejorar el mantenimiento de los mismos.

Para ello, los programas se descomponen en fragmentos denominados módulos, los cuales, según el autor Luis Joyanes Aguilar, en su libro *Fundamentos de programación. Algoritmos, estructuras de datos y objetos* (2008), son **unidades de código claramente definidas** y manejables, con **interfaces modulares**, creados para ejecutar una **tarea específica**, que son compilados (o interpretados) de forma **independiente**. Es por ello que son analizados y desarrollados por separado. Por lo que la estructura de un programa modular consistirá en un módulo principal el cual se encargará de llamar (o invocar) al resto de módulos o subprogramas.

Las ventajas que ofrece este tipo de programación son:

- Facilita el desarrollo de programas en **equipo**
- Permite la **reutilización** de módulos, reduciendo el tiempo de desarrollo.
- Simplifica el diseño y ejecución de las **pruebas**
- Reducción de los costes de **mantenimiento**
- Mejora la **claridad del diseño** y de la **documentación**
- Se reduce el número de **bugs** en el programa al reutilizarse módulos ya probados anteriormente.

Esto nos podría llevar a pensar que la mejor estrategia es dividir el software en el mayor número de módulos posibles, no obstante, esto provoca, a su vez, que el esfuerzo necesario para desarrollar y mantener las interfaces por las que se comunican se incremente. Es por ello que es necesario buscar un equilibrio entre ambos esfuerzos.

A la hora de resolver un problema, se nos plantean distintas formas de abordarlo para elaborar el algoritmo que lo solucione, lo que ha dado lugar a distintas técnicas:

4.2. Divide y Vencerás

Los problemas complejos se pueden enfrentar más eficazmente si los descomponemos en subproblemas que sean más sencillos de solucionar. En programación este método se llama “**Divide y Vencerás**” (“Divide et Vincere”, frase célebre de Julio César) y consiste en dividir un problema complejo en otros más simples. La subdivisión sucesiva de los subproblemas en otros aún más simples da lugar al **diseño Top-Down**.

4.3. Diseño Top-Down (Diseño Descendente)

Este diseño pasa por una serie de descomposiciones sucesivas del problema inicial, que va recibiendo un refinamiento progresivo.

La utilización de la técnica de diseño Top-Down tiene los siguientes objetivos básicos:

- Simplificación del problema y de los subprogramas de cada descomposición.
- Las diferentes partes del problema pueden ser programadas de modo independiente e incluso por diferentes personas.
- El programa final queda estructurado en forma de bloques o módulos, lo que hace más sencilla su lectura y mantenimiento.

5. Técnicas descriptivas

Para representar algoritmos hay diferentes métodos y herramientas que permiten describirlos de forma independiente respecto al lenguaje de programación que finalmente se utilice en su implementación.

Destacamos las siguientes:

- Pseudocódigo
- Tablas de decisión
- Diagrama de flujo
- Diagramas de Nassi-Scheiderman
- Diagrama estructurado

5.1. Pseudocódigo

Un pseudocódigo es una notación mediante la cual podemos describir un algoritmo, utilizando palabras y frases del lenguaje natural, sujetos a unas determinadas reglas. Se puede considerar como un paso intermedio, entre la solución de un problema y su codificación en un lenguaje.

Todo pseudocódigo debe posibilitar la descripción de los siguientes elementos:

- Instrucciones de entrada/salida
- Instrucciones de proceso
- Sentencias de control de flujo
- Acciones compuestas (subprogramas)
- Comentarios

La estructura general de un pseudocódigo es la siguiente:

- Programa: Nombre
- Preconiciones: Tips y rangos de las variables y estados en los que funcionará.
- Postcondiciones: Resultado de las acciones si se cumplen las precondiciones.
- Entorno: Declaración de estructuras de datos.

- Algoritmo: Secuencia de instrucciones que forman el programa.
- Fin de programa.

La primera parte del pseudocódigo contiene el nombre que el programador asigna al programa. Este nombre debe mantener algún tipo de relación directa o similitud con el funcionamiento del programa.

Así mismo es conveniente que contenga las precondiciones y postcondiciones de funcionamiento del algoritmo, aunque con frecuencia esta parte se omite, es de gran ayuda para documentar el funcionamiento del algoritmo.

La segunda parte es una descripción de los elementos que forman el entorno del propio programa. Incluiremos la declaración de los objetos del programa, es decir, declaración de variables (numéricas enteras, numéricas reales, alfanuméricas, lógicas) y debe ajustarse a las siguientes normas:

La declaración se puede realizar en una o varias líneas.

Si van en una línea debemos separarlas mediante comas.

La tercera parte indica el secuenciamiento de instrucciones que posteriormente se irán codificando en un lenguaje de programación. Es el algoritmo que resuelve el problema.

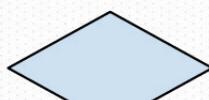
5.2. Ordinograma

Los ordinogramas representan gráficamente paso a paso todas las instrucciones del algoritmo, reflejando la secuencia lógica de las operaciones necesarias para la resolución de un problema. Estas operaciones van encaminadas a su resolución por medio del ordenador.

Al igual que en el caso de los organigramas, para la construcción de un ordinograma se requieren unos determinados símbolos que difieren de los organigramas:



Proceso



Decisión



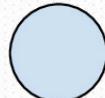
E/S



Llamada a procedimiento



Comienzo/Fin



Conector



Conector fuera de la página

El ordinograma detalla paso a paso el proceso que debemos seguir en la etapa de programación, y debe contar con los siguientes elementos:

- Un comienzo que viene señalado con la palabra “Inicio” y que determina el principio del programa.
- Una secuencia de operaciones lo más detallada posible y siguiendo siempre el orden en que se debe ejecutar (De arriba abajo y de izquierda a derecha).
- Un fin de programa marcado por la palabra “Fin”.

Las reglas que hay que seguir para la elaboración de un ordinograma son las siguientes:

- Todos los símbolos utilizados en el diseño deben estar conectados con líneas de flujos de datos.
- El diseño debe realizarse con la máxima claridad, siempre de arriba abajo y de izquierda a derecha.
- Queda terminantemente prohibido el cruce de líneas de conexión: eso indicaría un mal diseño.
- A un símbolo de proceso pueden llegarle varias líneas de conexión o flujo de datos, pero de él puede salir sólo una línea de conexión.
- A un símbolo de final de proceso o ejecución de programa pueden llegar muchas líneas de conexión, pero de él no puede salir ninguna.

6. Documentación

Al igual que en la fase de análisis, la fase de diseño y los diagramas y elementos generados en ella forman parte de la propia documentación.

La solución o diseño de la aplicación queda reflejada de forma detallada en el cuaderno de carga, que se construye basándose en las necesidades del usuario y en el análisis efectuado previamente por los analistas con el fin de obtener los requerimientos de la aplicación.

El cuaderno de carga está destinado a los programadores que van a desarrollar los programas de la aplicación, con objeto de que la codificación sea efectuada con la mayor calidad y rapidez posible.

Este documento debe estar realizado de tal forma que permita la división del trabajo de programación, para que varios programadores puedan trabajar de forma independiente, aunque coordinados por uno de los programadores o por un analista.

De una forma general, contiene información acerca de:

- Descripción de ficheros.
- Descripción de registros.

- Descripción de las ED.
- Estructura de los módulos.
- Diseño de la entrada de datos por pantalla.
- Diseño de la salida de datos por pantalla.
- Diseño de salida por impresora.
- Descripción de los elementos utilizados en cada módulo.
- Definición de las variables.
- Diseño de algoritmos.
- Etc.

7. Conclusión.

El diseño modular de funciones es un aspecto relevante del diseño estructurado. La forma de plasmar dicho diseño sin lanzarse directamente a su implementación pasa por el uso de técnicas descriptivas que aporten visión y valor al desarrollo del proyecto.

En este tema hemos visto las propiedades que cualifican a las funciones, así como técnicas de diseño de las mismas, que van a permitir desarrollar un software de calidad dentro del ciclo de vida de desarrollo del proyecto.

7.1. Relación con el sistema educativo

- GS – DAW - DAM –Entornos de Desarrollo

8. Bibliografía

- Pressman, R. S. **Ingeniería del Software. Un enfoque práctico**, 3^a edición. Ed. McGraw-Hill, 2000.
- Sommerville, I.: **Ingeniería de Software**. 6^a Edición. Addison-Wesley Iberoamericana, 2002.
- Piattini, M .y otros: **Análisis y diseño detallado de Aplicaciones Informáticas de Gestión**. RA-MA, 1996.
- Cerrada, J. A.: **Introducción a la Ingeniería del Software**. 1^a Edición de 2000. Editorial Centro de Estudios Ramón Areces, S.A.

