Preparador Informática

www.preparadorinformatica.com

TEMA 52 INFORMÁTICA

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

TEMA 52 INF: DISEÑO LÓGICO DE FUNCIONES. DEFINICIÓN DE FUNCIONES. DESCOMPOSICIÓN MODULAR. TÉCNICAS DESCRIPTIVAS. DOCUMENTACIÓN.

1. INTRODUCCIÓN

El proceso de desarrollo de un sistema de información debe pasar por una serie de etapas. En primer lugar, en el análisis, se estudia y analiza el dominio, el problema y las necesidades del usuario respecto a los requisitos planteados. Se trata de una etapa orientada al usuario (cliente) que es independiente del entorno en que se vaya a implementar el sistema.

En una segunda etapa se pasa al diseño, que es la etapa previa a la codificación. En la etapa del diseño, las necesidades del usuario recogidas en el análisis se convierten en un conjunto de modelos formales, que en refinamientos sucesivos llegarán a ser modelos computables por un ordenador. Se trata de una fase orientada a la máquina y donde el objetivo es determinar cómo puede ser el problema tratado por un ordenador. Durante la etapa de diseño se deberá especificar cómo se han de diseñar las estructuras de datos (diseño de datos), la arquitectura del sistema (diseño arquitectónico) y cómo ha realizarse la implementación de los detalles funcionales (diseño procedimental). Actualmente además del diseño de datos, arquitectónico y procedimental se debe de prestar especial atención al diseño de la interfaz.

En el presente tema nos centraremos en la etapa de diseño y más concretamente en el diseño procedimental.

2. DISEÑO DEL SOFTWARE

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.

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.

2.1. CRITERIOS DE CALIDAD DEL DISEÑO

Durante el diseño es donde se asienta principalmente la calidad del software. De este modo, 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.)

3. DISEÑO LÓGICO DE FUNCIONES

En la fase del diseño lógico o procedimental las especificaciones de operación de los procesos o funciones en que se descompone el sistema deben ser independientes del lenguaje final en que se realice la implementación.

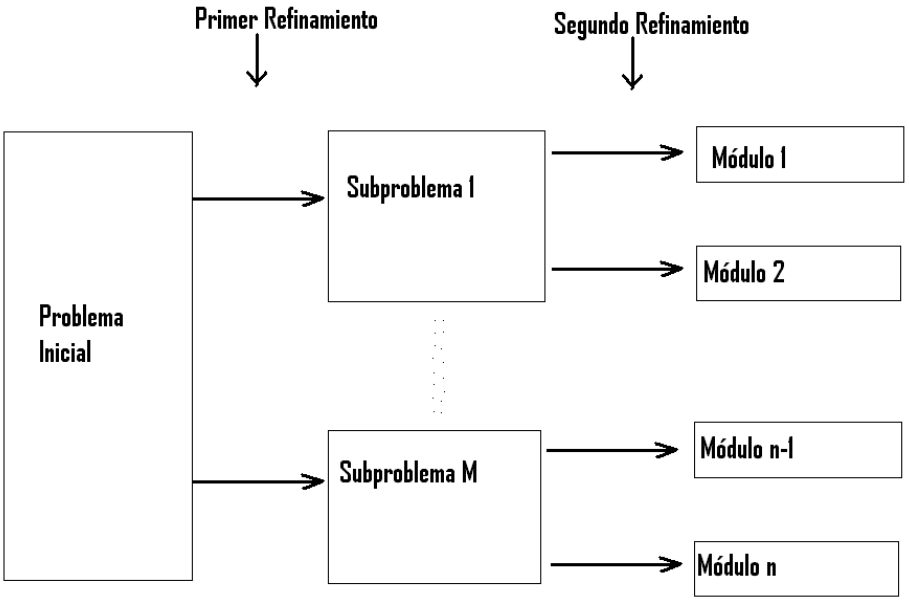
3.1. DEFINICIÓN DE FUNCIONES. DESCOMPOSICIÓN MODULAR

La descomposición modular consiste en dividir un componente software en unidades de menor tamaño en donde cada una de estas partes realiza una tarea explicita y única. Cada uno de estas partes recibe el nombre de módulo o función.

Las principales características de la descomposición modular son:

- **Facilita el trabajo en equipo,** ya que cada programador puede desarrollar un módulo diferente, y posteriormente enlazarlos.
- **Facilita la depuración**, ya que es más sencillo detectar y localizar errores en un único módulo que en el programa completo.
- **Facilita la reutilización**, ya que un módulo lo podemos usar en diferentes programas directamente.
- **Facilita el mantenimiento**, ya que es más sencillo modificar un único módulo que el programa completo.

La descomposición modular resuelve un problema efectuando descomposiciones en otros problemas más sencillos a través de distintos niveles de refinamiento.



4. 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 solo realice una función simple relacionada con el problema.

Por tanto, se hace necesario evaluar la calidad de la estructura de los diseños software y, a estos efectos, se utilizan principalmente dos parámetros de medida: el acoplamiento y la cohesión.

4.1. ACOPLAMIENTO

Se puede definir el 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.

Cuando el acoplamiento es bajo lo que indica es que se ha hecho una buena descomposición. Un acoplamiento bajo es deseable por:

- Cuantas menos conexiones haya entre dos módulos, habrá menos oportunidades para que se produzca el efecto onda, es decir, que un defecto en un módulo pueda afectar a otro.
- Cada cambio realizado en un módulo afecta lo menos posible a otros módulos. Es decir, 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.

A continuación, se describen de mejor a peor los diferentes niveles de acoplamiento:

1) **Acoplamiento normal**: es el que ocurre cuando se intercambian datos dos módulos, pero estos datos no interfieren en la operativa normal de la función que realiza el módulo de nivel inferior. Ejemplo: Un módulo A llama a otro módulo B, el módulo B se ejecuta y devuelve el control al módulo A. Dentro del acoplamiento normal, dependiendo de los datos que se intercambian entre los módulos se distingue:

- 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.
- Acoplamiento de marca o por estampado: los módulos se comunican 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 cuando los datos que se intercambian los módulos son controles. Este tipo de acoplamiento no es muy deseable, ya que no permite que los módulos sean totalmente independientes.

2) **Acoplamiento externo**: ocurre cuando los módulos están relacionados a componentes externos, por ejemplo, dispositivos de E/S, etc.

3) **Acoplamiento común**: ocurre cuando varios módulos 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 dicha área y porque los módulos así acoplados se modifican peor que los acoplados normalmente.

4) **Acoplamiento por contenido**: ocurre 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, etc. Ese tipo de acoplamiento es desaconsejable porque rompe la jerarquía de funcionalidad de la estructura.

4.2. COHESIÓN

La cohesión hace alusión a la relación que hay entre los elementos de un mismo módulo. Con la cohesión se tiene el objetivo de 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.

Por tanto, se puede definir la cohesión 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 daos o las llamadas a otros módulos. En el caso ideal, un módulo coherente solo debe hacer una única cosa.

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

Se distinguen los siguientes tipos de cohesión en un módulo (de mayor a menor):

- 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: se produce 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.
- Cohesión coincidente: se produce cuando sus elementos contribuyen a las actividades relacionándose entre si de una manera poco significativa.

5. TÉCNICAS DESCRIPTIVAS

Para representar gráficamente las funciones existen diferentes técnicas descriptivas que ayudan a describir su comportamiento de una forma precisa y genérica, para luego poder codificarlos con el lenguaje de programación que corresponda.

Destacan las siguientes técnicas:

A. Pseudocódigo o lenguaje de diseño de programas

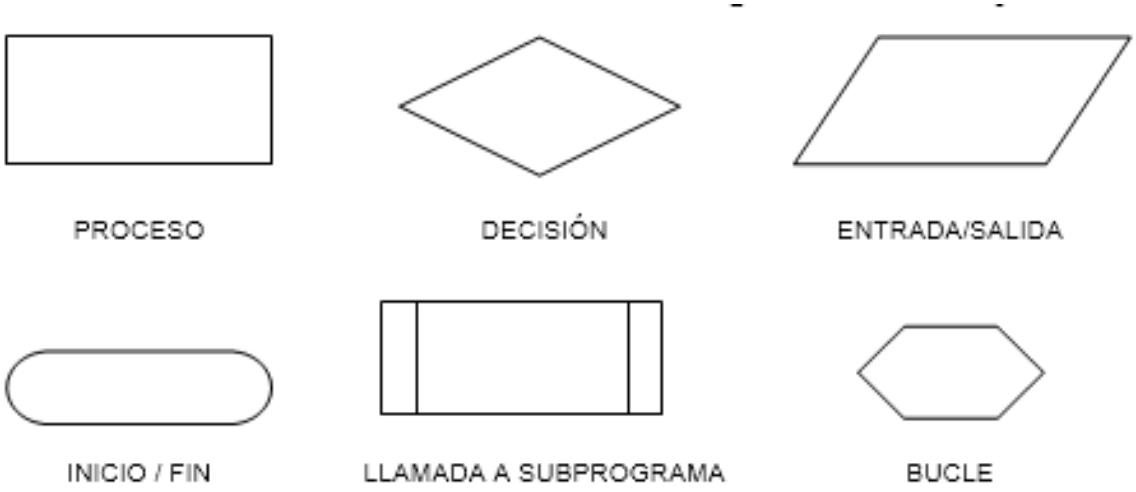
Esta técnica se basa en el uso de un vocabulario y una sintaxis limitados, es decir, un subconjunto de palabras del idioma elegido, de las cuales unas se utilizan para formar las construcciones propias de la programación estructurada (secuencia, selección y repetición) y otras incluyen un conjunto de verbos que reflejan acciones simples. Su fin es evitar las imprecisiones y ambigüedades del lenguaje natural y se puede considerar como un lenguaje intermedio entre este y los lenguajes de programación.

B. Diagramas de flujo

Los diagramas de flujo o flujogramas consisten en representar gráficamente las distintas operaciones que componen un procedimiento o parte de este, estableciendo su secuencia cronológica.

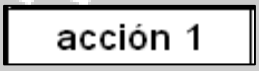
Se basan en estructurar de forma gráfica el algoritmo o función del programa antes de pasar a su codificación. Para ello, se dispone de grafismos para representar el diagrama.

Los símbolos más utilizados en los diagramas de flujo son:

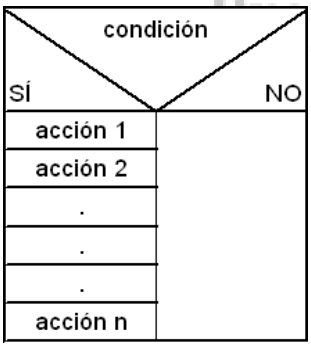


C. Otras técnicas descriptivas

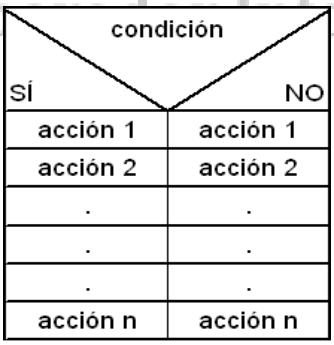
- Diagramas N-S: Los diagramas N-S (Nassi-Schneiderman) permiten representar gráficamente algoritmos, utilizando una representación en forma de cajas o bloques contiguos. Los símbolos más utilizados en los diagramas N-S son:



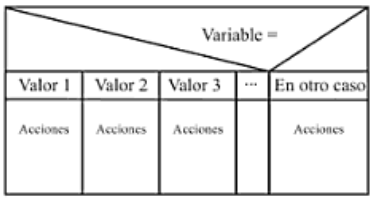
Tarea o proceso simple



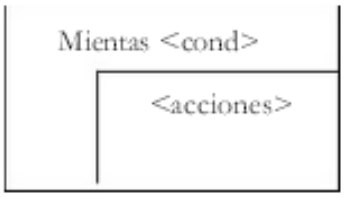
Selectiva simple



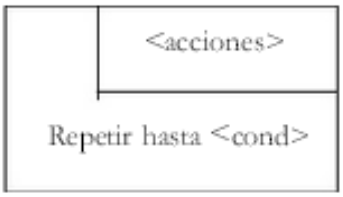
Selectiva doble



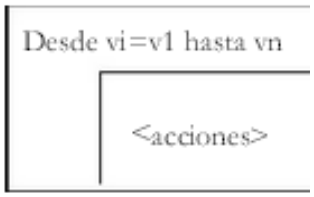
Selectiva múltiple



Repetitiva mientras

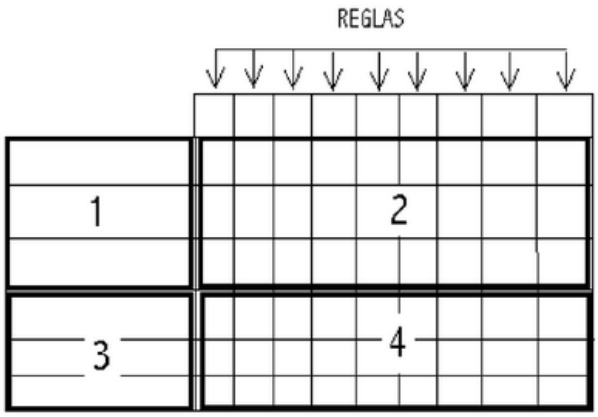


Repetitiva hasta



Desde o para

- Tablas de decisión: En una tabla son representadas las posibles condiciones del problema con sus respectivas acciones. Los pasos a seguir para elaborar una tabla de decisiones son:
  - Listar todas las acciones involucradas en el procedimiento
  - Listar todas las condiciones que se consideren durante la ejecución del procedimiento.
  - Asociar conjuntos específicos de condiciones con acciones, eliminando combinaciones imposibles.
  - Definir reglas indicando que acciones ocurren para un conjunto de condiciones



6. DOCUMENTACIÓN

La documentación de software es una etapa transversal a todas las demás etapas del diseño: se iniciará en los primeros pasos del ciclo de vida y continuará hasta la entrega final del producto. Concretamente la labor realizada en la etapa de diseño se recoge en un documento que se utilizará como elemento de partida para las sucesivas etapas del proyecto, y que se suele denominar Documento de Diseño de Software *(Software Design Document, SDD*)

En ocasiones las organizaciones tienen sus propios procedimientos de documentación. Por ejemplo, organismos internacionales como IEEE o la Agencia Espacial Europea tienen sus propias normas de documentación, bien como simples recomendaciones o como normas de obligado cumplimiento cuando se realiza un trabajo para dichos organismos.

7. CONCLUSIÓN

Una de las fases más importantes en la ingeniería del software es el diseño. Si esta fase se realiza adecuadamente tenemos grandes opciones de que nuestro software sea un software de calidad y por lo tanto un software exitoso.

La importancia del presente tema radica en abordar la fase del diseño procedimental o diseño lógico de funciones, la cual forma parte de la etapa del diseño del software, centrada en desarrollar una representación procedimental de las funciones especificadas para cada componente software y para la estructura de datos.

8. BIBLIOGRAFÍA

- Pressman, Roger. ***Ingeniería del software: un enfoque práctico.*** Editorial McGraw-Hill
- Sommerville, I. ***Ingeniería del software***. Editorial Addison Wesley
- Gómez, Sebastian y otros. ***Aproximación a la Ingeniería del software***. Editorial UNED
- Piattini, Mario G. y otros. ***Análisis y diseño de Aplicaciones Informáticas de Gestión. Una perspectiva de Ingeniería del Software***. Editorial Ra-Ma.
- Barlow y Bentley, ***Análisis y diseño de sistemas de información.*** Editorial McGraw-Hill