1. Caso Práctico

Unidad Didáctica VI

Caso Práctico

*En la unidad anterior dejamos a nuestros amigos de **SI Andalucía** programando la actualización de una aplicación ya existente para la gestión de almacén de una empresa. Esta aplicación fue inicialmente estudiada por **José** para comprobar que seguía los patrones de Programación Estructurada. Una vez confirmado este punto imprescindible se decidió por hacer un reparto modular de las tareas a desarrollar por cada uno de los miembros de su equipo. Como el código antiguo de la aplicación ayudaba bastante, lo dividió en tres módulos:*
Compras a proveedores y entrada de productos al almacén.
Gestión de inventario. Con todo tipo de consultas sobre los productos almacenados.
Salida de productos del almacén, (ventas a clientes).
Cada uno de estos módulos lo dividió además en otros submódulos con el fin de mejorar el rendimiento en el trabajo y reducir la complejidad de cada tarea. Así por ejemplo:

Módulos	Submódulos
Compras a proveedores	▪ **Entrada de productos pedidos a proveedores.** ▪ **Generación de pedidos de productos por debajo de stock mínimo.**
Gestión del Inventario	▪ **Consulta del inventario general del almacén.** ▪ **Reubicación de productos.** ▪ **Consulta por secciones.** ▪ **Listados estadísticos.** ▪ **Caducidad o devolución de productos desechados.**
Salida de productos	▪ **Ventas a clientes con actualización del número de existencias.** ▪ **Salida de productos** ▪ **Servicio de pedidos de clientes.**

***José** recuerda la primera aplicación que desarrolló con unos amigos. Se trataba de gestionar de forma automática el control y manejo de depósitos de sustancias de una fábrica. Aquella experiencia le resultó muy positiva, ya que consiguieron reutilizar la mayor parte del código en aplicaciones posteriores. En esa ocasión la división de la aplicación en módulos fue algo diferente de la que está tratando en estos momentos. En primer lugar decidieron dividir la aplicación basándose en el método de trabajo establecido en la fábrica, y lo sintetizaron en tres partes:*
Definición de los diferentes depósitos de sustancias con los que se va a trabajar.
Operaciones de procesamiento de las diferentes sustancias en los depósitos.
Obtención de resultados como consecuencia de los procesos aplicados a los depósitos.
Con este trabajo aprendió a utilizar con garantías la programación modular, y esto le ha permitido entender la programación orientada a objetos y la gestión de clases, que es la metodología de programación más utilizada actualmente.

Unidad Didáctica VI

Introducción

__José__ intenta inculcar a __Víctor__ buenos hábitos de programación y quiere que para cada módulo que programe se ajuste a determinados aspectos que permitirán que pueda ser empleado de forma independiente. Además espera:
que esté bien documentado (con comentarios),
que sea un código claro (fácil de leer con indentación y variables bien definidas) y que esté bien estructurado.
Esto a __Víctor__ le parecía una verdadera pérdida de tiempo al principio, pero se ha dado cuenta de que si hace lo que __José__ le pide, ese código puede ser utilizado en cualquier momento para otras aplicaciones, lo que supone un considerable ahorro de trabajo y tiempo. Por ejemplo __Víctor__ tuvo que programar un módulo para entrada de datos numéricos. Estaba compuesto por varias funciones de entrada de datos numéricos (euros con dos decimales, fechas, números enteros, números reales en notación científica, etc.) y recuerda que estuvo más de una semana desarrollando ese módulo, porque cada vez que lo entregaba a __José__ como terminado, éste lo rechazaba indicándole cambios y más cambios. Pero cuando lo dio por concluido, __Víctor__ se sorprendió de que ese módulo se utilizara en todas las futuras aplicaciones que requerían la entrada de cualquier valor numérico, y eso le hizo sentir muy satisfecho.

Recordarás que ya en la unidad anterior hablábamos de la programación modular como una de las "reglas de buena programación", que junto a la programación estructurada y otras nos permitían hacer programas:

más claros => más fáciles de mantener => menos costosos => más competitivos.

Autoevaluación

¿De qué manera consigue la programación modular que podamos hacer programas más claros, más fáciles de mantener, menos costosos y más competitivos? Señala la afirmación correcta.

- a) Facilitando la depuración, las pruebas, la integración, el ajuste y la modificación de un sistema, que puede abordarse por partes, disminuyendo la complejidad.
- b) La programación modular no consigue que podamos realizar programas más claros, sino que lo que permite es la programación por

capas.

c) Las afirmaciones a y b son correctas.

d) Ninguna de las anteriores es correcta.

Comprobar

¿De qué manera hacía esto la programación modular?

- **Dividiendo nuestro problema en partes independientes que puedan abordarse por separado.**
- **Solucionando cada parte de forma independiente, incluso por personas o equipos distintos.**
- **Permitiendo reutilizar módulos (trozos de código o programas) ya desarrollados (clases, herencia...).**
- **Independizando dentro de los módulos distintas partes y funciones distintas de nuestro problema (Diseño en tres capas: acceso a datos, negocio, presentación).**
- **Separando la definición de las estructuras de datos de los programas que las usan (definiendo tipos de datos abstractos).**
- **Independizando nuestra aplicación de las características físicas de los soportes sobre los que se almacenan los datos**.
- **Facilitando la depuración, las pruebas, la integración, el ajuste y la modificación de un sistema**, que puede abordarse por partes, **disminuyendo la complejidad**.
- **Aislando las dependencias funcionales.**

Una vez más, la utilidad de la programación modular se basa en la técnica del divide y vencerás, permitiendo descomponer problemas complejos en subproblemas o módulos más sencillos.

Autoevaluación

¿De qué manera consigue la programación modular que podamos hacer programas más claros, más fáciles de mantener, menos costosos y más competitivos? Señala la afirmación correcta.

Aislando las dependencias funcionales.

Separando la definición de las estructuras de datos de los programas que las usan (definiendo tipos de datos abstractos).

Las afirmaciones a y b son correctas.

Ninguna de las anteriores es correcta.

Comprobar

Unidad Didáctica VI

Concepto de módulo

*Pero al principio a **Víctor** le costó entender a qué se referían sus compañeros cuando hablaban de módulos. Ellos lo habían estudiado en clase y todos lo tenían muy claro. Daba la impresión de ser algo trivial y generalizado, que todo aquel que estaba relacionado con el tema conocía bien. Por eso no se atrevía a preguntar dando la impresión de que también sabía a lo que se estaban refiriendo.*
*El problema vino cuando **José** dividió la aplicación de gestión de almacén y le asignó que desarrollara el módulo de Gestión del Inventario. En aquel momento preguntó a **Carmen** que le dio una explicación convincente de lo que ella entendía por módulo: **"Básicamente se trata de un archivo que contiene funciones o rutinas que realizan tareas concretas dentro de un programa y que suelen estar relacionadas de algún modo. Estas funciones deben estar definidas de modo que puedan ser invocadas de forma generalizada cada vez que se necesite realizar esa tarea, para lo que será preciso hacer uso del archivo que la contiene, o sea del módulo."Víctor** no parecía muy convencido pero **Carmen** le puso un ejemplo claro de división de una aplicación en módulos. Le pidió que recordara una aplicación que desarrollaron juntos para el cálculo del IVA de un producto, para el que sólo necesitaban dos datos (precio y tipo de IVA). Se trataba de una aplicación muy sencilla, que según **Carmen**, no valía la pena dividir en módulos, pero que serviría como ejemplo. Para aquella aplicación se requerían varias tareas; pedir precio, pedir tipo de IVA, calcular el importe y mostrar el PVP definitivo. Con este panorama lo ideal sería dividir la aplicación en tres módulos:*
Uno para introducción de datos.
Otro para los cálculos y operaciones a realizar.
Y un tercero para mostrar los resultados del modo deseado.

***Carmen** le advirtió que en ocasiones se crea un módulo exclusivamente para definición de los datos y estructuras de datos que se utilizan, pero que en este pequeño ejemplo iba a quedar ridículo.*

Algo ya sabemos, o al menos podemos intuir, sobre lo que puede ser un módulo. Se pueden dar definiciones más o menos generales:

- Un módulo es **una asignación de trabajo para un programador**, que lo puede desarrollar de forma independiente.
- Un módulo es **un conjunto de estructuras de datos y subprogramas que tienen cierta relación y coherencia entre sí** (un **paquete**, en java).
- Un módulo es una **subrutina**, **procedimiento**, **función** o **método** que realiza una tarea concreta dentro de otro programa.

Todas esas definiciones, y otras similares que podrían darse, son ciertas. Y a pesar de ello seguramente seguirás sin tener claro lo que es un módulo. Puede deberse a que un módulo no es una única cosa concreta, sino una "**abstracción mental**" que hacemos para organizar mejor la tarea de programación, y que luego plasmamos de distintas maneras en nuestra forma de construir los programas. Esas abstracciones pueden referirse:

- **A los datos** (**independizar la definición de las estructuras de datos de las aplicaciones que las usan**, definiendo esas estructuras dentro de módulos distintos)

Autoevaluación

¿Cómo definirías en programación un módulo? Señala la afirmación correcta:

- ◯ Un módulo es una subrutina, procedimiento, función o método que realiza una tarea concreta dentro de otro programa.
- ◯ Un módulo es una asignación de trabajo para un programador, que lo puede desarrollar de forma independiente.
- ◯ Un módulo es un conjunto de estructuras de datos y subprogramas que tienen cierta relación y coherencia entre sí.
- ◯ Todas las anteriores son correctas.

Comprobar

Por **ejemplo**, hacer un módulo que defina qué va a ser para mí un alumno de PLE. (Su estructura y su comportamiento).

- **La estructura definirá que tipo de valores va a tomar**, qué datos van a constituir la entidad alumno. Todo alumno va tener **un nombre, de tipo cadena de caracteres, una edad, de tipo entero positivo, un sexo, de tipo lógico y una nota final de tipo entero positivo con valores posibles en el rango 1-10.**
- **También definirá qué operaciones se podrán realizar** con un alumno y cómo se harán. Podré **crear un nuevo alumno, dar de baja un alumno, modificar el valor de alguno de sus campos** (por ejemplo, evaluarlo escribiendo la nota que tiene). Y es en la definición de la estructura de datos "alumno" donde definiré qué operaciones se van a poder hacer sobre los alumnos, y de qué manera se harán, pero no es donde realmente se harán. Serán otros módulos de la aplicación los que usen la estructura. El módulo de **matriculación** creará alumnos nuevos, el módulo de **evaluación** les asignará notas (y sólo podrá asignarle notas válidas, porque la estructura de datos alumno definida no permite otra cosa), el módulo de **gestión académica** dará de baja a alumnos y mantendrá actualizados sus datos personales, etc.
- **Al control** (**independizar tareas distintas de nuestra aplicación dentro de módulos distintos**). En el ejemplo anterior, nuestra aplicación de gestión de alumnos será la que realmente use la estructura de datos "alumno", pero claramente desde el principio hemos identificado varias tareas que a simple vista son más o menos independientes. Es por eso por lo que nos planteamos que constituyan módulos separados. Cada uno de esos módulos (matriculación, evaluación, gestión académica) podrá desarrollarse por programadores distintos, o incluso por equipos de programación distintos, si el tamaño lo hace necesario, ya que cada uno de ellos a su vez puede estar compuesto de otros submódulos. Esos módulos se relacionarán entre sí de una forma bien definida, intercambiando información. La forma de relacionarse un módulo con los demás constituye su **interfaz (o interface),** que consistirá en definir claramente los datos de entrada que debe recibir de otros módulos, y el valor que devolverá para que otros módulos puedan usarlo.

Autoevaluación

¿En qué consiste, según tu opinión, la interface de un módulo? Señala la afirmación correcta:

- ○ En definir claramente los datos de entrada que debe recibir de otros módulos y el valor que devolverá para que otros módulos puedan usarlo. (correcta)
- ○ En la forma como presentamos el módulo ante el usuario.
- ○ Las afirmaciones a y b son correctas.
- ○ Ninguna de las anteriores es correcta.

Comprobar

Unidad Didáctica VI

Características de los sistemas modulares

*Con esta metodología de programación, **Víctor** comprendió que se obtenían ciertas ventajas que le permitían ganar en rapidez de desarrollo y sólo tenía que pensar las cosas una vez. Le bastaba con tener una biblioteca de módulos personales que reutilizaba cuando le interesaba, incluso actualizaba sus propios módulos mejorándolos para futuros usos.*

En el apartado anterior hemos visto que no es fácil dar una definición única de módulo que se adapte a todas las posibilidades. Parece que más que una definición concreta de módulo, lo que necesitamos es una **lista de características que deben cumplir los sistemas modulares**.

¿Cuáles serán esas características, a la luz de lo estudiado en la unidad 4? Un módulo será cualquier abstracción que cumpla (en mayor o menor medida) estas características:

- Cada módulo se corresponde con **un subsistema** claramente definido, y **que puede resultar útil para otras aplicaciones.**
- La función de cada módulo tiene **un propósito específico**, definido claramente.
- Cada módulo maneja no más de **una estructura de datos principal o relevante** del sistema.
- Las funciones que manejan **instancias** de un **tipo abstracto de datos** se definen y quedan **encapsuladas** dentro de la estructura de datos de la que se trate.
- Las **funciones de un módulo comparten datos globales** de forma selectiva.

La modularidad de un sistema tiene grandes ventajas. Mejora la claridad del diseño de la aplicación, lo que facilita la codificación, la depuración, las pruebas, la documentación y el mantenimiento. De hecho, como parte de la documentación de cualquier aplicación suelen aparecer diagramas que representen los módulos de los que consta el sistema, como en el caso de los diagramas o cartas de estructura.

Autoevaluación

¿Qué característica debe de cumplir un sistema modular? Señala la afirmación correcta:

- ◯ Cada módulo se corresponde con un subsistema claramente definido, y que puede resultar útil para otras aplicaciones.
- ◯ Las funciones de un módulo no pueden compartir datos globales de forma selectiva.
- ◯ Las afirmaciones a y b verdaderas.

Comprobar

Por ejemplo:

DEMO: Veamos un Diagrama de flujo

Pulsa en el cursor para ver las distintas etapas presentes en el tratamiento de la información.

Para saber más:
En este enlace podrás conocer algunas curiosidades sobre la programación modular, especialmente con el lenguaje de programación Java.

Estructura Multicapa en Java [Versión en caché]
(Si tienes problemas para acceder a algún enlace, pulsa en "Versión en caché" para visualizar una copia de esa página web)

Para descargar el programa Acrobat Reader pulsa aquí.

Autoevaluación

¿Qué característica debe de cumplir un sistema modular?.Señala la afirmación correcta:

- Cada módulo maneja más de una estructura de datos principal o relevante del sistema
- b. Las funciones que manejan instancias de un tipo abstracto de datos se definen y quedan encapsuladas dentro de la estructura de datos de la que se trate.
- Las funciones de un módulo no pueden compartir datos globales de forma selectiva.
- Todas las anteriores son correctas.

Comprobar

Unidad Didáctica VI

Criterios para definir la modularidad de un sistema

*Todo esto está muy bien, pero **Víctor** se pregunta cómo definir los módulos. Decía que resultaba fácil cuando veía la aplicación como módulos independientes, pero que se sentía incapaz de hacer una división con buen criterio. **Carmen** le cuenta que la única forma de aprender a dividir los problemas es con la práctica, que al principio saldrá peor y que con los errores se aprende, pero no obstante le comenta algunas aspectos que se pueden tener en cuenta y que ayudan en la descomposición.*

*Desde aquel momento **Víctor** aplicó esos criterios de división a la mayoría de las aplicaciones que desarrolló y no le fue mal del todo.*

Venimos hablando hace rato de las ventajas que aporta el uso de la programación modular, como si los programas se dividiesen en módulos por sí mismos. Pero ¿piensas que realmente es fácil decidir qué módulos tendrá nuestra aplicación? Para un problema concreto,

- ¿cuántos módulos debemos hacer?,
- ¿qué función tiene cada uno?,
- ¿qué debemos tener en cuenta?

Ese tipo de preguntas deben ser contestadas en la fase de análisis de la aplicación, y darán como resultado la decisión de estructurar la aplicación en diversos módulos.

Unidad Didáctica VI

Objetivos al dividir el problema en módulos

¿Cómo saber si la división en módulos que he realizado es adecuada? Una vez más no existen reglas fijas, ni una única forma de estructurar en módulos un problema dado. Tener claros algunos objetivos previos y algunos ejemplos, ayuda.

Cada módulo será una entidad bien definida que tiene las siguientes características:

- Los módulos **contienen tanto instrucciones o procesamiento lógico como estructuras de datos.**
- Los módulos **pueden ser compilados aparte y almacenados en una biblioteca o librería para poder ser reutilizados en otras aplicaciones.**
- Habrá **partesde un módulo que podrán utilizarse invocando un nombre con algunos parámetros** (llamadas a métodos o subrutinas)
- Los módulos **pueden usar a otros módulos**.

Autoevaluación

De las siguientes, ¿cuál es en tu opinión una característica que debe de cumplir un módulo?.Señala la afirmación correcta:

- ◯ Los módulos contienen tanto instrucciones o procesamiento lógico como estructuras de datos.
- ◯ Los módulos pueden ser compilados aparte y almacenados en una biblioteca o librería para poder ser reutilizados en otras aplicaciones.
- ◯ Habrá partes de un módulo que podrán utilizarse invocando un nombre con algunos parámetros.
- ◯ Todas las anteriores son correctas.

Comprobar

Ejemplos de módulos son:

- Las subrutinas, métodos, procedimientos o funciones.
- Los grupos de métodos, subrutinas procedimientos o funciones que están relacionados por realizar una funcionalidad común.
- Las abstracciones de datos. (o **tipos abstractos de datos**)
- Las clases de programación orientada a objetos.
- Los paquetes que agrupan clases por funcionalidad o por pertenecer a la misma aplicación.

Autoevaluación

De los siguientes,¿cuáles son en tu opinión ejemplos de módulos?.Señala la afirmación correcta:

- ◯ Las subrutinas,métodos, procedimientos o funciones.
- ◯ Las abstracciones de datos.
- ◯ Las clases de programación orientada a objetos.
- ◯ Todas las anteriores son correctas.

Comprobar

Unidad Didáctica VI

Criterios de modulación

Llegados a este punto, lo que seguramente necesitarás es tener claros una serie de criterios a valorar en cada caso. Al menos conocer cuales son los principales criterios que se pueden tener en cuenta para que tu problema se estructure en módulos de la mejor manera posible. Quizás convenga volver a recordarte que no existe una única forma de dividir en módulos el problema, y que tampoco hay reglas fijas que garanticen que la división hecha es la mejor posible. Como contrapartida tampoco es fácil asegurar que la división que hagas no sea la mejor.

Existen muchos criterios de modulación y según el que usemos, el resultado será distinto.

- **Criterio convencional** (Cada módulo con sus submódulos corresponden a un paso en la secuencia de ejecución.
- **Criterio de ocultamiento de la información (** Cada módulo oculta a otros módulos una decisión difícil o modificable del diseño, de forma que pueda ser modificada sin que tengan que cambiar los demás módulos.
- **Criterio de abstracción de datos (** Cada módulo oculta los detalles de representación de una estructura de datos bajo un conjunto de funciones o métodos que acceden y modifican la estructura.
- **Criterio de acoplamiento y cohesión (** La estructura del sistema busca maximizar la cohesión interna de cada módulo y minimizar el acoplamiento entre módulos.
- **Criterio de limitar el tamaño máximo de cada módulo (** Cualquier módulo de un tamaño grande seguramente incluye más de una función o más de una estructura de datos principal, susceptible de constituir un módulo independiente. Como regla general, no se aconsejan rutinas de más de 30 líneas de código ni módulos de más de 120.

Autoevaluación

¿En qué consiste el criterio convencional?.Señala la afirmación correcta

- Cada módulo oculta los detalles de representación de una estructura de datos bajo un conjunto de funciones o métodos que acceden y modifican la estructura .
- Cada módulo oculta a otros módulos una decisión difícil o modificable del diseño, de forma que pueda ser modificada sin que tengan que cambiar los demás módulos.
- Cada módulo con sus submódulos corresponden a un paso en la secuencia de ejecución.
- Ninguna de las anteriores es correcta.

Comprobar

En cualquier caso, pueden usarse varios criterios para modularizar un sistema, pero siempre se obtendrá una estructura de más calidad (más fácil de entender, implementar y modificar) usando en el diseño criterios bien definidos.

Autoevaluación

¿En qué consiste el criterio de limitar el tamaño máximo de cada módulo? Señala la afirmación correcta

- No se aconsejan rutinas de más de 120 líneas de código ni módulos de más de 30.
- Cada módulo oculta los detalles de representación de una estructura de datos bajo un conjunto de funciones o métodos que acceden y modifican

la estructura.

- No se aconsejan rutinas de más de 30 líneas de código ni módulos de más de 120.
- Ninguna de las anteriores es correcta.

Comprobar

Programación modular

Unidad Didáctica VI

Acoplamiento entre módulos

*Uno de los aspectos que **Carmen** le comentaba a **Víctor** sobre la descomposición de un problema en módulos consistía en tener en cuenta la posibilidad de conseguir la mayor independencia posible de los módulos, o lo que es lo mismo, un bajo nivel de acoplamiento. Cuanto menos dependa el funcionamiento de un módulo de los resultados obtenidos en otros módulos mejor reutilización presentará, aunque el precio que hay que pagar por esto es mayor dificultad de programación y una mayor dedicación al diseño.*

¿Qué es el acoplamiento entre módulos? ¿En qué medida debemos tenerlo en cuenta para modularizar un sistema?

El acoplamiento entre dos módulos puede definirse como el grado de interdependencia entre ambos, es decir, en qué medida las operaciones que realiza uno pueden afectar a las que realiza otro de una forma clara o por el contrario como efectos colaterales difíciles de prever y controlar. También puede definirse como **el número y complejidad de las interacciones entre distintos módulos**. Evidentemente, si queremos reducir la complejidad de nuestro código, ¿Buscaremos un alto nivel de acoplamiento o bajo?

- Mientras más acoplados estén dos módulos, más probable es que los cambios en uno obliguen a modificar el otro.
- Mientras más complejas sean las relaciones entre dos módulos, más difícil será darse cuenta de qué es lo que hay que modificar en los demás módulos al modificar uno con el que están fuertemente acoplados.

Por tanto**, buscaremos tener un sistema con módulos lo menos acoplados posible**, **y eso se consigue cuando la única interacción entre unos módulos y otros se produce a través de un interfaz bien definido, a través de los parámetros que se pasan en la llamada**. En este caso, si las interfaces de los módulos que se modifiquen permanecen sin cambios, los detalles internos modificados no afectarán a otros módulos, que por tanto no deberán ser modificados.

Si por ejemplo, el módulo es una estructura de datos, su interfaz será la lista de métodos u operaciones que se pueden ejecutar con esa estructura. Si esa lista de operaciones no cambia, poco importa que cambie el código que realiza una operación concreta (para hacerla más eficiente, por ejemplo). Cualquier otro módulo que use esa estructura seguirá ejecutando la operación de la misma forma, y no necesitará ser modificado.

En el lado contrario, **el máximo acoplamiento se dará cuando varios módulos comparten una misma zona de datos en memoria, a la que ambos acceden para hacer consultas y modificaciones.** Cualquier cambio en ese bloque de datos obligará a modificar todos los módulos que estén acoplados a esos datos. Y cuando un módulo modifique los datos, puede afectar en gran medida al funcionamiento del otro módulo que también los usa.

Autoevaluación

Como norma, se ha de buscar tener un sistema con módulos lo menos acoplados posible.En tu opinión, ¿cómo se consigue eso?.Señala la afirmación correcta:

- ○ Se consigue utilizando el menor número posible de valores dependientes entre módulos.
- ○ Se consigue cuando la única interacción entre unos módulos y otro se produce a través de un interfaz bien definido, a través de los parámetros que se pasan en la llamada. (correcta)
- ○ Se consigue utilizando el menor número de funciones y clases dentro de los módulos.

- Todas las anteriores son correctas.

Comprobar

La comunicación entre módulos incluye el paso de datos (a través de los parámetros, de elementos de control tales como banderas, etiquetas o nombres de procedimientos) y las modificaciones del código de otros módulos. El acoplamiento es:

- menor mediante el paso de parámetros,
- mayor para el paso de elementos de control y
- mucho mayor para módulos que son capaces de modificar el código de otros módulos.

Autoevaluación

En relación con el acoplamiento entre módulos, teniendo en cuenta que se ha de buscar tener un sistema con módulos lo menos acoplados posible, ¿cuándo se dará el máximo acoplamiento entre módulos?.Señala la afirmación correcta:

- Cuando utilizamos un gran número de módulos. A más módulos, mayor índice de acoplamiento.
- Cuando varios módulos comparten una misma zona de datos en memoria a la que ambos acceden para hacer consultas y modificaciones.
- La afirmación a y b son correctas.
- Todas las anteriores son falsas.

Comprobar

Unidad Didáctica VI

Cohesión interna de cada módulo

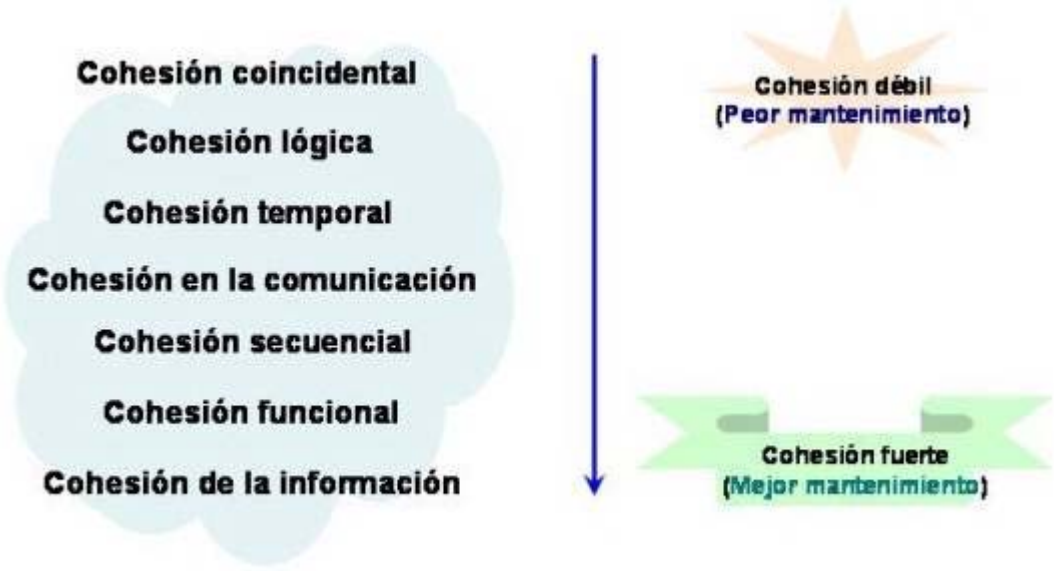
CASO. Aunque parezca paradójico, comparado con el apartado anterior, es recomendable una fuerte cohesión entre los componentes de un módulo. ***Víctor*** *no terminaba de entender esto, pero con la clara explicación de su amiga* ***Carmen*** *lo entiende todo mucho mejor; "Todas las funciones o rutinas que se incluyen en un módulo deben tener una serie de puntos comunes, que en algunos casos llevan a que una función utilice otras funciones de su mismo módulo, lo que aporta una gran potencia al software desarrollado. Por ejemplo en el módulo de entrada de datos, referido antes de la aplicación para el cálculo del IVA, es interesante pensar que si hacemos una función que impida la entrada de valores no numéricos, esta función se puede emplear tanto al pedir el precio como al pedir el tipo de IVA."*

Hemos visto que no es deseable que unos módulos estén estrechamente relacionados con otros. Es deseable que los distintos módulos presenten un bajo acoplamiento, es decir, que sean relativamente independientes. ¿Para qué? Para que cuando tengamos que modificar cualquiera de ellos, podamos tener la tranquilidad de que los demás módulos que lo usan no van a tener que ser modificados también.

Pero si en vez de mirar distintos módulos del sistema miramos distintos elementos dentro del mismo módulo, ¿qué crees que será recomendable, una fuerte o una débil relación? La respuesta es fuerte. Eso es justamente **lo que mide la cohesión interna de un módulo, la fuerza con que están unidos los distintos elementos de ese módulo.**

Una clasificación orientativa del grado de cohesión, de más débil (menos deseada) a más fuerte (más deseada) puede ser la siguiente:

- **Cohesión coincidental** (Aparentemente los distintos elementos no tienen ninguna relación. Puede ser un programa de gran tamaño, que se ha segmentado en módulos arbitrariamente, o cuando se meten en el mismo módulo un conjunto de instrucciones que aparecen frecuentemente en otros módulos.
- **Cohesión lógica** (Distintos elementos se incluyen en el mismo módulo no porque se relacionen entre sí, sino porque realizan funciones similares. Las bibliotecas de funciones matemáticas pueden ser un ejemplo. La función "seno" no tiene más relación con la función "valor absoluto" de la que podría tener con cualquier otro método, procedimiento o función del problema, pero se agrupan en el mismo módulo sólo por ser funciones matemáticas.
- **Cohesión temporal** (Todos los elementos son ejecutados en el mismo momento, sin lógica alguna ni parámetro que determine cuál debe ejecutarse. Un ejemplo puede ser un módulo de inicialización, que asigna valores a un conjunto de variables o estructuras de datos.
- **Cohesión en la comunicación** (Cuando varios elementos no sólo se ejecutan en el mismo momento, sino que además se refieren al mismo conjunto de datos de entrada o salida.
- **Cohesión secuencial** (Cuando la salida de un elemento del módulo es usada como entrada del siguiente elemento del módulo.
- **Cohesión funcional** (Todos los elementos se encuentran relacionados por el desempeño de una única función. Es un grado alto y por tanto deseable de cohesión.
- **Cohesión de la información** (Cuando el módulo contiene una estructura de datos compleja y varias rutinas o métodos que manejan esa estructura, presentando cada rutina cohesión funcional. Es la realización total de una abstracción de datos.



Autoevaluación

Ordena por grado de cohesión, de más débil (menos deseada) a más fuerte (más deseada) la cohesión interna de un módulo.Señala la afirmación correcta:

- a) Cohesión secuencial, lógica, temporal, funcional.
- b) Cohesión funcional, temporal, secuencial, de la información.
- c) Cohesión coincidental, secuencial, temporal, de la información .
- d) Cohesión lógica, temporal, funcional,de la información

Comprobar

Puede que resulte difícil saber de forma exacta el grado de acoplamiento o cohesión de nuestro sistema, pero esos niveles pueden verse como sugerencias a tener en cuenta, más que como criterios cuantitativos estrictos.

Autoevaluación

Define qué entiendes por cohesión lógica. Señala la afirmación correcta:

- Todos los elementos son ejecutados en el mismo momento, sin lógica alguna ni parámetro que determine cuál debe ejecutarse .
- Todos los elementos se encuentran relacionados por el desempeño de una única función
- Distintos elementos se incluyen en el mismo módulo no porque se relacionen entre sí, sino porque realizan funciones similares.
- Cuando varios elementos no sólo se ejecutan en el mismo momento, sino que además se refieren al mismo conjunto de datos de entrada o salida.

Comprobar

Autoevaluación

Define qué entiendes por cohesión temporal.Señala la afirmación correcta:

- a) Todos los elementos son ejecutados en el mismo momento, sin lógica

- alguna ni parámetro que determine cuál debe ejecutarse.
- b) Todos los elementos se encuentran relacionados por el desempeño de una única función
- c) Distintos elementos se incluyen en el mismo módulo no porque se relacionen entre sí, sino porque realizan funciones similares.
- d) Cuando varios elementos no sólo se ejecutan en el mismo momento, sino que además se refieren al mismo conjunto de datos de entrada o salida.

Comprobar

Programación modular

Unidad Didáctica VI

Otros criterios

*Después de todo esto, **Carmen** le recomienda a **Víctor** que a la hora de programar, de dividir una aplicación en módulos o de diseñar pruebas de revisión, lo mejor es ser ordenado y aplicar el sentido común. Además en estos casos la experiencia es muy importante, ya que en ocasiones las posibilidades y herramientas del programador, unidas a los requerimientos del cliente condicionan la aplicación de tal modo que es imposible aplicar los criterios habituales y se hace necesario abordar la aplicación con un grado de abstracción importante.*

***Carmen** cita como ejemplo la primera aplicación que realizó **José** sobre el funcionamiento de los depósitos de una fábrica. Realmente hay que tener mucha imaginación para sintetizar las operaciones que se realizan en depósitos del tamaño que sea, en un programa de ordenador, y que finalmente simule perfectamente el funcionamiento de la fábrica.*

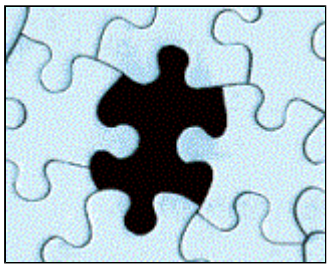
Todos los criterios dados anteriormente te parecerán poco concretos. No te preocupes, son poco concretos. La división adecuada en módulos no es algo que pueda hacerse de forma única, y será la experiencia y la costumbre la que nos hará entender correctamente las ventajas e inconvenientes de aplicar unos criterios u otros. Se trata de tener en mente las "pistas" que nos pueden ayudar.

De forma general pueden tenerse en cuenta, dependiendo del sistema de que se trate, otra serie de criterios, además de los mencionados en los apartados anteriores, que en cada caso deberá sopesar el diseñador, y definir un conjunto consistente de criterios que se usarán para dirigir todo el proceso.

- **Ocultamiento de las decisiones** complejas o modificables del diseño (**lo importante de un módulo es lo que hace, no cómo lo hace**).
- **Limitar el tamaño** físico de cada módulo (Cualquier rutina de más de 30-40 sentencias, seguramente realiza varias funciones susceptibles de estar divididas en más de un módulo).
- **Estructurar el sistema** para mejorar la claridad y facilitar las pruebas.
- **Aislar rutinas** que sean dependientes de la máquina (trozos programados en ensamblador).
- **Facilitar la labor de modificación**.
- Aumentar la eficiencia en la **gestión de la memoria**, en sistemas con memoria limitada.
- Aumentar la eficiencia en **sistemas en tiempo real** o con tiempo de respuesta crítico.

Un gran número de pequeñas rutinas implica una sobrecarga en el tiempo de ejecución a la hora de ligar las llamadas a los distintos módulos durante el tiempo de ejecución.

Como regla general es deseable:



- diseñar e implementar el sistema de forma altamente modular,
- medir el rendimiento del sistema, e
- identificar los cuellos de botella para eliminarlos reconfigurando y combinando módulos, e incluso reprogramándolos en ensamblador si es necesario.

En estos casos, el código altamente modular debe guardarse como documentación para las rutinas realizadas en ensamblador, de forma que facilite su comprensión. Siempre suele ser más fácil reagrupar varios módulos en uno sólo que hacer divisiones en módulos que no se habían previsto inicialmente.

Para saber más:
En este enlace podrás profundizar y comprender la estructura y algunas de las aplicaciones posibles de la estructura n-capas. Pretende ser un acercamiento para la mejor comprensión de la arquitectura multicapa como máximo exponente de la programación modular.
Arquitectura de Capas **[Versión en caché]**
(Si tienes problemas para acceder a algún enlace, pulsa en "Versión en caché" para visualizar una copia de esa página web)