Cohesión y Acoplamiento

Los conceptos de cohesión y acoplamiento surgieron a partir del diseño estructurado para resolver las típicas preguntas:
- ¿Hasta cuanto modularizar?
- ¿Hasta cuanto seguir dividiendo funciones en funciones más específicas?
- ¿Puedo medir la calidad de división de funciones de un sistema?
- ¿Qué tan independientes son los módulos de un sistema?
- ¿El producto de un proceso de diseño es realmente "resistente" a la inclusión de cambios?

Con el cambio en el diseño de sistemas a partir de la incorporación de la programación orientada a objetos al desarrollo de software estos conceptos no han perdido vigencia. El concepto de cada término es el mismo que en la programación estructurada pero su significado específico difiere en cada paradigma.

En conclusión ambos conceptos sirvieron (y sirven) para medir la calidad de un diseño en lo que refiere a la división de funciones o responsabilidades en sus partes.
A continuación se explican ambos conceptos para ambos paradigmas de construcción de software.

Diseño Estructurado

Cohesión

Medida del grado de identificación de un módulo con una función concreta. Se mide en cada módulo (procedimiento o función).

Es el grado de relación que existe entre las tareas de un módulo. Cuanto más relacionadas y orientadas a resolver el propósito del módulo mayor será el nivel de cohesión del mismo. Es por ello que buscamos tener el máximo nivel de cohesión en cada módulo.

Niveles de Cohesión

Aceptables (fuerte)

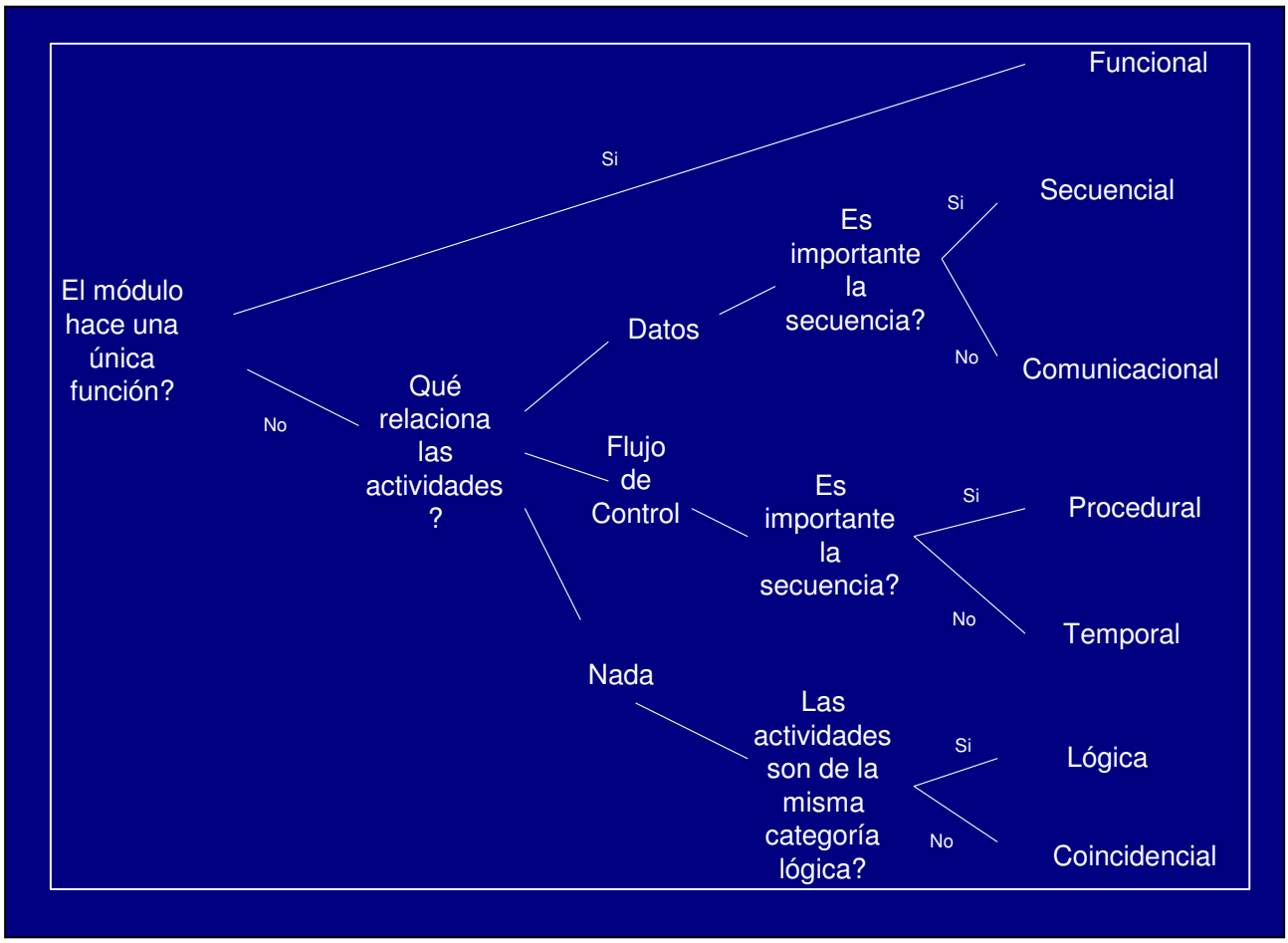
- **Funcional**: El módulo realiza una única acción.
- **Secuencial**: Las acciones que realiza el módulo deben ejecutarse en un orden preestablecido pues operan sobre datos en común. La salida de una acción es la entrada de otra.
- **Comunicacional**: Las acciones que realiza el módulo operan sobre datos en común pero pueden ejecutarse en cualquier orden.
- **Temporal**: Las acciones que realiza el módulo deben realizarse en un momento determinado dentro del flujo de control del sistema.

No Aceptables (débil)

- **Procedural**: Las acciones que realiza el módulo están relacionadas a través de flujo de control por lo que tienen que ejecutarse en un orden específico.
- **Lógica**: Se ejecuta una parte del módulo dependiendo de un parámetro. El módulo tiene una estructura tipo "case" o "if" anidados con la que decida que parte de su código debe ejecutar.

- **Coincidental**: No existe ningún tipo de relación observable entre las acciones que realiza en módulo.

Estrategia práctica para determinar la Cohesión



Acoplamiento

Medida de la interacción de los módulos que constituyen un sistema. Se mide entre dos módulos (procedimiento o función).

Es el grado de relación que existe entre dos módulos. Cuanto menor sea la relación entre módulos, menos interdependientes serán por lo que un cambio en uno afectará en menor medida en el resto. Es por ello que buscamos tener el menor nivel de acoplamiento entre módulos.

Niveles de Acoplamiento (del mejor al peor)

- **Datos**: El módulo llamador le envía al módulo llamado parámetros conteniendo datos simples. Es el mejor nivel de acoplamiento. La dependencia entre módulos es débil.

- **Estampado**: El módulo llamador le envía al módulo llamado parámetros conteniendo estructuras de datos, por ejemplo un registro. El módulo llamado debe conocer la división interna de la estructura recibida. Aumenta el nivel de dependencia.

- **Control**: El módulo llamador le envía al módulo llamado parámetros de control diciéndole que parte de su código debe ejecutar. Síntoma de cohesión lógica en el módulo llamado. El módulo llamador debe conocer la estructura interna del módulo llamado para saber qué parámetro de control debe enviar para que realice la función esperada. Aumenta la dependencia.

- **Inversión de autoridad:** El módulo llamado devuelve un parámetro de control al módulo llamador diciéndole que parte de su código debe ejecutar. A Esta altura sería más simple que ambos conformen un solo módulo.

- **Common**: Es cuando ambos módulos utilizan variables globales a ambos. En ciertas variables de solo lectura es aceptable, por ejemplo la identificación del usuario conectado a la aplicación. En general no es deseable ya que la conexión de los módulos no es visible en forma explícita.

- **Patológico**: Cuando un módulo utiliza parte del código de otro.

Conclusión

Hoy día todavía se escribe código estructurado, por ejemplo en la creación de procedimientos almacenados de bases de datos relacionales, por lo que es muy importante tener en cuenta la cohesión y el acoplamiento a la hora de diseñarlos para garantizar el menor costo de mantenimiento futuro entre otras cosas.

Reducir el acoplamiento entre módulos y aumentar el nivel de cohesión de cada uno de ellos.

Diseño Orientado a Objetos

De la misma manera que en el diseño estructurado buscamos construir software con piezas independientes entre sí, bajo acoplamiento, y con alto nivel de cohesión, en el diseño orientado a objetos perseguiremos similar fin. Los modelos que construyamos intentarán estar conformados por objetos altamente cohesivos y con el menor acoplamiento entre sí.

Cuando hablamos de alta cohesión de un objeto nos referimos al grado de relación que tiene el objeto con sus responsabilidades. En la medida que el objeto implemente exclusivamente las responsabilidades que le corresponde por su naturaleza mayor será el nivel de cohesión que tendrá y por ende más independiente será del resto.
Por su naturaleza, el DOO conduce a la creación de sistemas débilmente acoplados. Para ello es fundamental que la implementación del objeto esté oculta dentro del objeto y que no sea visible a los componentes externos.

Recomendaciones a tener en cuenta para la cohesión y el acoplamiento:

- El sistema no debe tener un estado compartido entre distintos objetos.
- Cualquier objeto debe ser reemplazable por otro con la misma interfaz.
- Si se divide una clase cohesiva las clases resultantes estarán fuertemente acopladas.
- Si se juntan clases con bajo acoplamiento la clase resultante tendrá baja cohesión.
- Las asociaciones bidireccionales aumentan el acoplamiento.
- Crear conexiones estrechas. Número de mensajes que un objeto envía a otro y la cantidad de parámetros deben ser mínimos.

- **Acoplamiento de contenido**: Se da cuando un objeto accede a los atributos del otro sin pasar por su interfaz.
- **Acoplamiento de control:** Un objeto le envía un mensaje a otro conteniendo parámetros de control.
- **Acoplamiento estampado:** Un objeto le envía un mensaje a otro conteniendo una estructura de datos.
- **Acoplamiento por datos:** Un objeto le envía un mensaje a otro conteniendo datos.
  Cuantos más argumentos tiene un método mayor es el acoplamiento. Evitar el uso de datos “excursionistas”, es decir, datos que viajan de un objeto a otro sin ser utilizados hasta que alcanzan su destino.
- **Acoplamiento de subclases:** Una clase cliente tiene una referencia a la subclase en lugar de tener la referencia a la superclase. Aumenta la dependencia.