Acoplamiento y cohesión

Según sabemos, el encapsulamiento y el ocultamiento de información resultan eficientes cuando realizamos un correcto proceso de abstracción. Durante este proceso elegimos qué operaciones quedarán expuestas como parte de la "interfaz pública" de un objeto y cuáles no. En ese sentido, estamos limitando qué operaciones pueden ejecutar otros objetos sobre el que estamos trabajando. La modularidad exige tener entidades de software autocontenidas, o lo más autocontenidas posibles, de forma tal que puedan ser intercambiables sin afectar a otros módulos del sistema.

Una clase diseñada de forma tal que esté autocontenida es una clase que hace solo aquello para lo que fue pensada, sin inmiscuirse en el funcionamiento de otras clases. Así, al momento de producir objetos, las relaciones entre estos estarán bien delimitadas. Que un objeto sepa hasta dónde realizar una operación y dónde le corresponde a otro realizar sus operaciones, nos da indicio de que cada clase que dio lugar a esos objetos tienen una tarea muy específica a cumplir. Una responsabilidad estrictamente definida.

Es decir, un correcto proceso de abstracción nos lleva a definir lo que se conoce como las responsabilidades de las clases: qué hace cada clase. Como sabemos, las clases definen atributos operaciones, que se traducirán, al momento de generar objetos, en estado interno y comportamiento. Dada la importancia del encapsulamiento y el ocultamiento de información, asumimos que cada vez que queremos acceder al valor de un atributo, deberemos hacerlo usando una operación.

Por eso, ponemos foco en lo que "hacen" las clases y los objetos. Es decir, si una clase define atributos a los que queremos que otros objetos accedan, habrá necesariamente, al menos una operación para acceder al valor de ese atributo, una para establecerlo o una combinación de ambos.

Las clases no solamente son contenedores de datos, pueden definir otras operaciones, como hacer algún cálculo o procesar algún dato. Algunas de esas operaciones necesitan valerse de otros objetos, pero las relaciones con esos otros objetos deben ser limitadas o, mejor dicho, delimitadas. Para evitar problemas, es importante definir correctamente las responsabilidades de las clases: es importante que una clase haga solo aquello para lo que fue pensada y no debe inmiscuirse en el trabajo de otra clase.

Veamos un ejemplo: Supongamos una calculadora que emite tiques. La calculadora tiene dos partes que saltan a la vista, por un lado, está el display LCD y, por otro, el teclado numérico. A priori, diríamos que eso es una calculadora, pero… ¿quién hace las cuentas? El teclado podría realizar las cuentas. ¿Y por qué no el display LCD? ¿Por qué no ambos?

Pensemos un momento en un mundo ideal donde la obsolescencia programada no existe: se rompe el teclado de nuestra calculadora y compramos un repuesto. Si se rompe el display también podemos comprar un repuesto y reemplazarlo.

Definitivamente, las cuentas no las pueden hacer ambos, porque si se rompe una parte, se rompe toda la calculadora y ya que emite papel, bien podríamos usar la calculadora sin el display LCD. Es decir, repartir la carga de hacer las cuentas no es correcto. Pensemos en las cosas que hace cada parte de la calculadora como la teoría de la división del trabajo. ¿Qué es lo que hace mejor un display LCD? Mostrar números. ¿Qué es lo mejor que hace un teclado numérico? Tomar entradas del usuario. ¿Qué es lo mejor que hace la pequeña impresora de papel? Imprimir sobre papel térmico. No sería conveniente, entonces, complejizar cada uno de estos tres componentes con una tarea adicional. Cada componente está pensado para una tarea específica y, es más: podría funcionar por sí mismo fuera de la calculadora. El fabricante podría tener varios modelos de impresoras y usar siempre el mismo teclado y display. Podría incluso tener calculadoras que no impriman en papel. Nunca sería conveniente que cada una de estas partes dependa de la otra para funcionar.

Lo que nos lleva de nuevo al inicio, ¿quién hace las cuentas? Bien, deberá haber un componente específicamente pensado, diseñado y construido para hacer cuentas. Para las cuentas, se debe tomar la entrada del usuario y se debe enviar al display, se deben tomar los operandos y mostrar el resultado en el display. Se debe tomar el resultado y enviarlo a imprimir.

Nuevamente, ¿quién debería dirigir el cabezal de impresión y moverlo milimétricamente para un lado y para el otro? La impresora es quien conoce de impresiones, papeles y medidas. El “procesador” solo hace las cuentas y dice al display: “hey, muestra este resultado”, o a la impresora “hey, imprime estos números”.

Entonces, por más que compliquemos y compliquemos las cosas, siempre llegamos a un mismo punto: cada componente debería hacer solo aquello para lo que está pensado y nada más, podría incluso funcionar independientemente y ser reemplazado sin afectar a los otros componentes.

Así como se ha pensado la calculadora, deberemos pensar nuestros programas orientados a objetos. La modularidad da lugar a la cohesión de cada módulo. La cohesión es una medida de la definición de responsabilidades. Si la cohesión es alta, la entidad está más autocontenida, es más autosuficiente, es capaz de cumplir con su función sin depender de los demás, sabe exactamente qué debe hacer y cómo hacerlo.

Por tanto, la cohesión mide qué tan bien están definidas las responsabilidades de las clases. Esto es, qué tan eficiente fue nuestro uso del encapsulamiento y ocultamiento de información al momento de realizar el proceso de abstracción.

Si una clase es altamente cohesiva, hace aquello que debe hacer sin depender de otras clases. Es decir, es más independiente de otras clases, está desacoplada de otras partes.

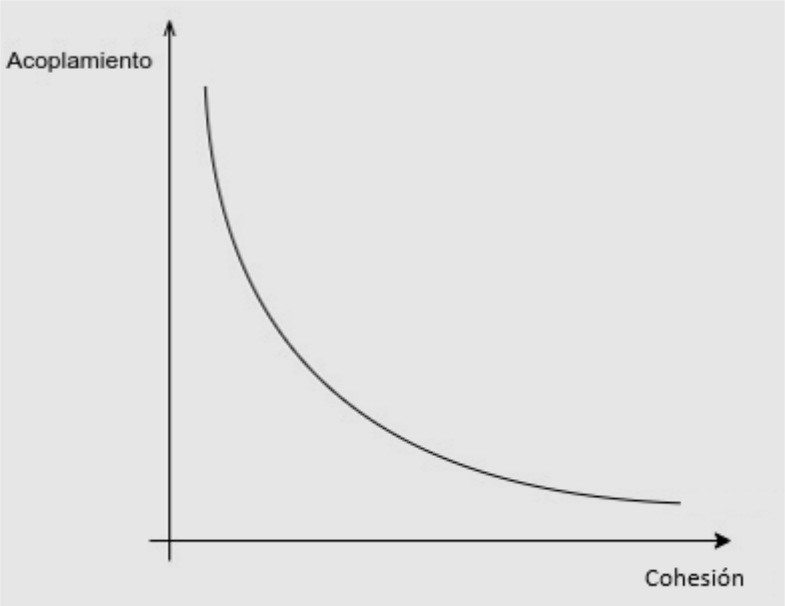
Definimos entonces al acoplamiento como la medida de la interdependencia entre las clases que interactúan en nuestro programa orientado a objetos.

Es importante aclarar que la cohesión y el acoplamiento no son medidas aplicables solo a las clases. Sabemos que la unidad de encapsulamiento más pequeña es la clase, pero la modularidad puede ser aplicada a unidades más grandes. Es decir, cuando planteamos un sistema orientado a objetos debemos tener cuidado de no acoplar clases, pero tampoco subsistemas.

Relación

Dado el análisis que hicimos, podemos ver que ambos conceptos están siempre interrelacionados. La situación ideal de todos nuestros componentes es que sean **altamente cohesivos y muy poco acoplados**. Un correcto proceso de abstracción nos llevará a definir correctamente las responsabilidades de las clases, lo que nos lleva naturalmente a que se interrelacionen solo lo justo y necesario.

Lamentablemente, es prácticamente imposible que nuestras clases estén absolutamente desacopladas y, por ende, nunca serían absolutamente cohesivas. Esto es porque, necesariamente, debe haber algún tipo de relación entre los objetos para que el sistema funcione. Es por eso por lo que se suele representar a la relación entre cohesión y acoplamiento de la siguiente manera, con una función logarítmica.



Justificación

¿Por qué es tan importante la modularidad, la alta cohesión y el bajo acoplamiento? La idea de tener componentes que se puedan reemplazar con otros componentes sin afectar al resto del sistema es muy interesante, pero hay algo más por detrás: los costos. En la industria del software es fundamental poder reutilizar componentes y en muchas otras industrias también, es muy común que diferentes modelos de la misma marca usen los mismos componentes, se ve mucho en el interior de los vehículos, donde las perillas, colores, palancas, etc., son iguales, por más que sea el auto más caro o el más barato de la marca.

Pero lo que nos compete aquí es la industria del software, y obtener componentes reutilizables es de una importancia absoluta. ¿Por qué tener partes cohesivas y que no dependan de otras partes? Para reutilizarlas en otro sistema y no tener que volver a desarrollarlas; para poder reemplazarlas con nuevas partes, mejor testeadas sin esfuerzos de desarrollo; para incrementar la productividad de los empleados: no debemos pagarles todo desde cero cada vez que realizamos un nuevo programa, porque ya tendrán muchos componentes que puedan reutilizar tal cual están e incorporarlos al nuevo sistema. Eso es posible ya que cada uno de esos componentes es altamente cohesivo y, por tanto, poco acoplado a otras partes. No depende del sistema que hicimos anteriormente y es fácilmente adaptable al nuevo que estamos por comenzar.

¿Consideras importante mantener el acoplamiento bajo? ¿O crees que se puede realizar algún compromiso en los niveles de cohesión y acoplamiento?