Unidad 2: Cohesión y Acoplamiento

Introducción y orientaciones para el estudio

En el desarrollo de este módulo abordaremos:

- Motivaciones.
- Síntomas y causes de diseños deficientes.
- Estrategias para mitigar dificultades.
- Conceptos de Modularización.
- Aplicaciones en universos estructurados y en Orientación a Objetos.

Objetivos

Pretendemos que al finalizar de estudiar esta unidad el alumno logre:

- Detección, medición y clasificación de problemas comunes en escenarios estructurados y OO.
- Identificación de las causas y aplicación de estrategias de resolución.
- Ejercitación mediante escenarios.

Aclaraciones previas al estudio

En este módulo, el alumno encontrará:

- Contenidos
- Conceptualizaciones centrales

Usted debe tener presente que los contenidos presentados en el módulo no ahondan profundamente en el tema, sino que pretenden ser un recurso motivador para que, a través de la lectura del material, la bibliografía sugerida y el desarrollo de las actividades propuestas alcance los objetivos planteados.

Cada módulo constituye una guía cuya finalidad es facilitar su aprendizaje.

Introducción

Cualquiera que lleve algún tiempo trabajando desarrollo o gestión informática, seguramente vivió solicitudes de cambios de requerimientos y sus impactos.

Los gestores de cambio (dueño del producto, usuario, arquitecto, analista funcional, o programador), y más aún los de índole técnica introducirán cambios en la aplicación con los que deberemos lidiar.

La cohesión y el acoplamiento son dos métricas que nos permiten revisar nuestro diseño (tanto para escenarios estructurados como para escenarios de Orientación a Objetos), elegir que parte del mismo es aceptable cual no, y presentar abordajes para solucionarlos.

Estas métricas procuran que nuestros requisitos no funcionales y/o nuestros atributos de calidad se mantengan en niveles deseados o aceptables.

¿Cuándo un diseño se pone deficiente?

- **Rigidez**
  - **Difícil de cambiar:** cuando el cambio no fluye, y se consume tiempo buscando estrategias.
  - **Temor al cambio:** cuando los gestores temen a los cambios que ellos mismos promueven.
- **Fragilidad**
  - **Tendencia a "romperse":** cuando se instala la sensación de que el software se romperá de cualquier modo.
  - **"Pincha" en sitios inesperados:** cuando el equipo no puede prever que deberá probar, cuanto tiempo le llevará y asume que deberá probar toda la funcionalidad en cada entrega.
- **Inmovilidad**
  - **"Entrelazado"**: cuando el software está hecho de una manera tal, que es imposible reutilizarlo en otros proyectos.
- **Viscosidad**
  - **Distorsión del Diseño Original**: cuando se presenta dificultad en utilizar métodos que preserven al diseño original cuando nos enfrentamos a un cambio.

Causas de diseño deficiente

- **Cambio de Requisitos**
  - **Son inevitables**: en varios sentidos también necesarios. Muchas veces promovidos también por los propios constructores de la aplicación.
  - **Diseños Pobres**: es necesario conseguir que los diseños sean resistentes a los cambios (a prueba de balas de ser posible).
- **Gestión de dependencias**
  - **Degradación**: los cambios que no contemplan las dependencias, provocan degradación en la calidad, es cuando se introducen nuevas dependencias no planificadas.

Algunas Estrategias

- **Revisión de Atributos de Calidad**: en los criterios de aceptación de las entregas.
- **Revisión de Modularización, Cohesión y Acoplamiento** (tratados en este documento)
- **Patrones de Diseño**, Patrones **de Arquitectura**, Principios **SOLID+**.

La Modularización

La cohesión y acoplamiento surgieron como **técnicas para controlar, medir y mejorar el diseño** de sistemas **dentro del paradigma estructurado**. Estas técnicas fueron tan buenas que fueron adoptadas de inmediato en la Orientación a Objetos, aunque con algunos cambios necesarios.

Sabemos que para tener un **buen diseño hay que modularizar**, que dicho de otra manera **es segregar o dispersar el código** en partes más **pequeñas, específicas y eficientes**. Pero cuando lo hacemos se nos presentan las siguientes 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?

En la Orientación a Objetos, estas mismas preguntas siguen vigentes. Su nivel de aplicación puede diferir en algunos tipos, y debido al propio paradigma su aplicación suele ser más específica. A continuación se explican ambos conceptos para ambos paradigmas de construcción de software.

Cohesión

*…**"Es el grado de relación de las tareas o funciones de un módulos[1] para con el módulo".*** Cuanto más relacionadas y orientadas a resolver el propósito del módulo mayor será el nivel de cohesión del mismo. Medir la cohesión es medir la relación. Alta relación implica una cohesión alta o fuerte.

Es por ello que buscamos tener el máximo nivel de cohesión en cada módulo. Dicho de otro modo, si tenemos un módulo que se llama "facturación", una alta cohesión pretenderá que todas las funciones que haya en el mismo se correspondan con esa tarea.

Los niveles que se mencionan a continuación fueron postulados para el paradigma estructurado, definiendo aquellos que son aceptables, y aquellos que no. Más adelante en este documento, se explican cómo se trasladan a la Orientación a Objetos.

Niveles de Cohesión (Originalmente del paradigma Estructurado)

Aceptables (Concesión Fuerte)

1. **Funcional**: El módulo realiza una única acción. Es el tipo ideal. A veces se logra (cuando la funcionalidad es atómica), muchas veces no.

Vinculadas a Datos en Común:

2. **Secuencial**: Las acciones que realiza el módulo deben ejecutarse en un orden preestablecido debido a los datos en común. La salida de una acción es la entrada de otra.
3. **Comunicacional**: Las acciones del módulo operan sobre datos en común pero pueden ejecutarse en cualquier orden.

Vinculadas a Flujos de Control:

4. **Procedural**: Las acciones del módulo se relacionan a través de flujo de control por lo que tienen que ejecutarse en un orden específico.

No Aceptables (débil)

5. **Temporal**: Las acciones que realiza el módulo deben realizarse en un momento determinado dentro del flujo de control del sistema.

No Vinculadas ni por Datos ni por Flujos de Control:

6. **Lógica**: Se ejecuta una parte del módulo dependiendo de la validación un parámetro. El módulo tiene una estructura tipo "case" o "if" anidados con la que decide que parte de su código debe ejecutar, produciendo distintos resultados de pendiendo de ese parámetro.
7. **Coincidental**: No existe ningún tipo de relación observable entre las acciones o tareas que se realizan dentro de un módulo.

[1] Entendamos por módulo a cualquier componente de software. Puede ser un módulo, paquete, procedimiento, clase, etc…

Acoplamiento

…"Es el grado de relación que existe entre dos módulos". Cuanto menor sea esa relación entre módulos, se dice que son menos interdependientes, por lo que uno tiene que saber menos del otro, y que un cambio en uno afectará menos al otro.

Es por ello que buscamos tener **el menor nivel de acoplamiento entre módulos**. Se mide entre dos componentes. Se **mide que intercambian** y **cuando debe conocer uno respecto del otro**.

Niveles de Acoplamiento (del paradigma Estructurado)

Aceptables

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

2. **Estampado**: El llamador le envía al llamado parámetros conteniendo estructuras de datos (registros, vectores, matrices, listas). El módulo llamado debe conocer la división interna de la estructura recibida. Aumenta el nivel de dependencia.

No Aceptables

3. **Control**: El módulo llamador le envía al módulo llamado parámetros de control diciéndole cual parte de su código (o sub rutina) debe ejecutar. Es un síntoma de cohesión lógica en el módulo llamado. El llamador debe conocer la estructura interna del llamado para saber qué parámetro de control debe enviar para que realice la función esperada. Aumenta aún más la dependencia.

4. **Inversión de autoridad**: El módulo llamado devuelve un parámetro de control al módulo llamador diciéndole que parte de su código debe ejecutar. La dependencia sigue aumentando. A esta altura sería más simple que ambos conformen un solo módulo.

5. **Común**: Es cuando ambos módulos utilizan variables globales a ambos. En ciertas variables de solo lectura es aceptable (ejemplo: la identificación de un usuario conectado a la aplicación). En general no es deseable ya que la conexión de los módulos no es visible en forma explícita. Por otro lado, cuando el dato es de escritura, su valor se suele corromper si no es debidamente tratado entre ambos módulos.

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

Cohesión y Acoplamiento en Orientación a Objetos

En OO, se trasladan los conceptos de cohesión, acoplamiento como así también el tratamiento para otros atributos de calidad en el diseño (como la Cambiabilidad, y la Reusabilidad).

Para esos atributos, en algunos casos, también se aplican prácticas y procesos usados establecidos en paradigmas anteriores (como el estructurado, o el de orientación a eventos).

La cohesión y el acoplamiento en la orientación a objetos, suele tener visiones diferentes de acuerdo a distintos autores.

Cohesión (en OO)

Podemos pensar el alcance a la cohesión en OO desde tres puntos de vista:

- **Cohesión del objeto respecto de sus responsabilidades**:

  Hablamos de **alta cohesión** en un objeto cuando hay alto grado de relación en **la semántica del objeto respecto de las responsabilidades que desarrolla**.

  En la medida que el objeto **i**mplemente exclusivamente las responsabilidades que le corresponden **mayor será el nivel de cohesión**.

- **Cohesión en cada método del objeto**:

  Pensando que lo anterior se cumple, podemos estudiar la **especificidad y segregación de los métodos (públicos o privados)** y su interrelación.

- **Cohesión dentro del paquete que contiene a las clases**:

  Se aplica el concepto de cohesión a la **relación que guardan las clases dentro de un paquete que las contiene**, tal como se verá más adelante en el conjunto de principios SOLID +, visto en la unidad siguiente.

Acoplamiento (en OO)

Se lo describe como el nivel de relación, conocimiento o dependencia que tienen objetos, clases y paquetes entre sí. Tomamos como consigna principal y objetivo que el sistema tenga componentes **lo más independientes posibles.**

Esto implica que cada objeto/clase tenga una **métodos de pocos argumentos** (entre 3 y 5 según algunos autores), **claramente definidos**, con **el menor acoplamiento** (conocimiento sobre otros) posible.

Una **bien implementada OO** conduce a la creación de componentes en un sistema **débilmente acoplados**.

Además de interfaces estrechas, es fundamental que la estructura interna de un objeto esté oculta dentro del objeto. **No respetar el encapsulamiento genera más acoplamiento**.

Un sistema **bien diseñado no necesita compartir su estado interno**. Los valores los valores de los atributos de cada objeto son de cada objeto.

Cualquier objeto usado se debe poder reemplazar por otro con la misma interfaz. Este tema se verá luego en los principios de SOLID+.

Si se **divide erróneamente una clase altamente cohesiva**, las clases resultantes **estarán fuertemente acopladas**.

Si se **integran dos clases con bajo acoplamiento la clase resultante no será altamente cohesiva**.

Si se diseñan las clases simplemente **como una colección de datos**, con un conjunto de operaciones que solo gestionan los mismos, **no hay garantías sobre el nivel de acoplamiento obtenido**.

Las interacciones (por estructura o por llamadas entre métodos) entre objetos/clases, y por ende **el tipo de relación condicionan el acoplamiento**.

Cada **tipo de relación tiene más o menos acoplamiento** respecto de las otras relaciones. No implica que haya relaciones no aceptables, como en la definición original, sin embargo podré comparar las relaciones usadas en función de cómo dependen entre sí.

La **relación más leve es la dependencia (o uso)**, ya que la relación es efímera mientras se ejecuta el método en el método usado. El objeto usado no depende del que lo usa.

En el caso de la asociación, la **navegabilidad entre objetos/clases condiciona también el tipo de acoplamiento**. Una asociación navegable en un sentido, es menos acoplada que si es navegable en los dos sentidos.

Las **agregaciones son más acopladas que las asociaciones**, debido a que **tienen la contención física de los objetos agregados**.

La composición, **es la relación de asociación más acoplada de todas**. Por eso es una relación que **en lo posible tratamos evitar**, debido **a su uso tan exclusivo** y el costo en términos de esfuerzo de desarrollo y mantenimiento debido a su alto acoplamiento.

La generalización/especialización, tiene también un tipo de acoplamiento: **el acoplamiento**

de clase, del que hablaremos luego en el apartado siguiente.

Tipos de acoplamiento en OO

1. Acoplamiento de datos (Aceptable)

Es el mismo primer caso visto en acoplamiento en estructurado. Es la **forma sencilla o común de intercambiar información entre objetos**. Ocurre cuando **dos o más objetos intercambian escalares o primitivas del lenguaje** o parámetros simples sin estructura.

A mayor cantidad de argumentos tiene un método **mayor acoplamiento**. Se debe reducir al mínimo posible el acoplamiento entre objetos, no pasando más argumentos de los que verdaderamente se van a utilizar.

Algunos autores recomiendan **manejar de 3 a 5 parámetros** como referencia a un **acoplamiento lo más estrecho posible**. En cualquier caso el número debe ser el mínimo.

Siguiendo este criterio, se recomienda **evitar datos "excursionistas"**, en referencia a parámetros que **pasan de objeto en objeto sin ser utilizados**.

Además que se **atenta contra el acoplamiento y performance** posiblemente, **se generan vulnerabilidades** en el sistema, al exponer datos que no serán tratados directa o inmediatamente.

2. Acoplamiento por estampado o marca (Aceptable)

También se trata del mismo segundo caso que en acoplamiento en estructurado, pero además con alguna variante propia. Este tipo de acoplamiento ocurre **cuando una objeto/clase** se **declara como parámetro** de **un método de otro objeto/clase**.

La otra forma que ocurra es cuando **dos objetos/clases se pasan formas compuestas de datos (registros o estructuras)** no primitivas del lenguaje, las formas compuestas de datos **añaden oscuridad (o desconocimiento) al diseño**.

Se recomienda **evitar lo más posible el estampado**. Esto produce que se **ensanche la interfaz**, y que **se incremente también el costo de mantenimiento**.
Otro de los riesgos es que **un objeto pueda manipular y cambiar valores que no utiliza**.

3. En relación a la Herencia (Aceptable)

Las clases derivadas (debido a los atributos y operaciones heredadas) **están acopladas a sus clases base** y a que los cambios en las superclases se propagarán a todas las clases que heredan sus características **forzándolas a adaptarse al cambio**. Esto **afecta el objetivo de independencia**.

Solución: usar adecuadamente y donde represente mayor beneficio las generalizaciones/especializaciones. Las **variables de clase también perjudican el acoplamiento**, ya que **fuerzan un grado de interdependencia excesivo entre objetos**. La solución es encapsular la mayor cantidad de variables de clase en instancias declarándolas privadas, donde sea posible.

Solución: reducir el uso de variables de clase a escenarios estrictamente necesarios.

4. **Acoplamiento de Contenido (No Aceptable)**

   No es tolerado que un objeto **modifique de forma solapada datos del estado interno de otro objeto**, es indicativo de la ruptura del encapsulamiento. La solución es que cada objeto tenga privadas todas sus variables de estado interno, y que el mismo modifique sus estados. **Solución**: respetar el encapsulamiento.

5. **Acoplamiento de Control (No Aceptable)**

   Tampoco es tolerado **que un flujo de control salte "alegremente" de un objeto a otro objeto**. Esto pasa cuando un objeto le **pasa a otros datos con la intención de controlar su lógica interna**. **Solución**: en ocasiones reducir y simplificar la interfaz con menos parámetros es la mejor solución. Buscar que las respuestas de los mensajes entre objetos sean predecibles.

   Al igual que en estructurado, **es un síntoma de cohesión lógica**. Cuanto esto ocurre **significa que un objeto conoce los detalles de implementación del otro** objeto en forma implícita y **se pierde independencia**. **Solución**: verificar que todos los parámetros entre una llamada y otra son usados.

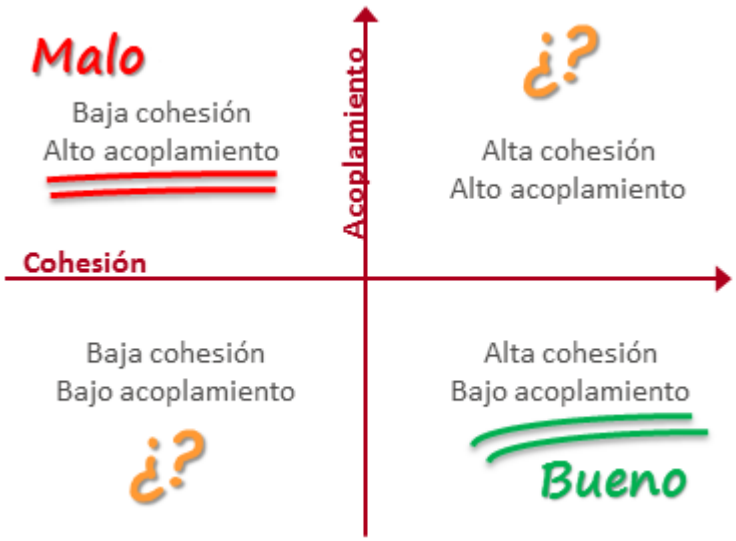
6. **Otros tipos de Acoplamiento (No Aceptables):**

   Invocación con alta recurrencia: si se usa de forma muy repetida una secuencia de dos o más métodos (ejemplo: para realizar un cálculo). **Solución**: se puede reducir este tipo de acoplamiento escribiendo una rutina que encapsule la secuencia en forma integrada.

   External coupling: cuando tenemos módulos que dependen de componentes externos determinados como el sistema operativo, bibliotecas compartidas o hardware. **Solución**: reducir el número de sitios en el código en el que existan este tipo de dependencias

Conclusiones

1. Revisar cohesión y acoplamiento **es saludable sin importar el paradigma** del lenguaje.
2. Los lenguajes estructurados siguen **clasificaciones y reglas detalladas**.
3. Los lenguajes OO, siguen iguales reglas, pero **además tienen casos especiales**.
4. Deben **procurarse niveles aceptables.**
5. La **cohesión debe ser alta y el acoplamiento bajo**.

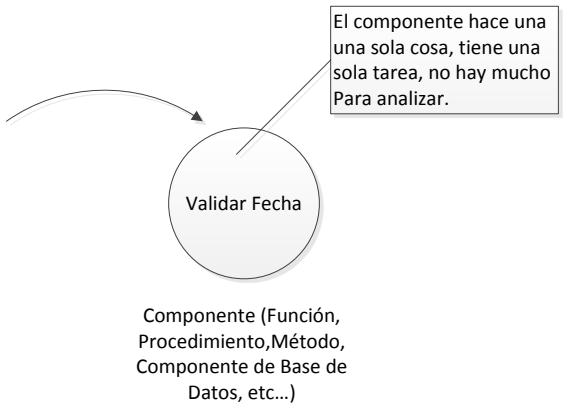


Apéndice de Ejemplos
Cohesión (en Estructurado)

Los siguientes ejemplos son genéricos, de modos que sean aplicables a cualquier tipo de componente no orientado a objetos. Para el caso de la Orientación a Objetos, más adelante habrá ejemplos especiales.

1. Cohesión Funcional (Aceptable)

   El caso es trivial. Se tiene un componente, que hace una sola cosa y que es llamado por todos para hacer esa funcionalidad siempre.



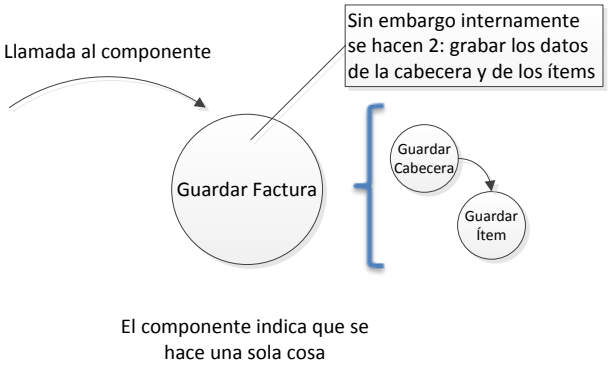
<u>Vinculadas a Datos en Común</u>:

2. Cohesión Secuencial (Aceptable)

   El ejemplo describe a un módulo que tiene 2 tareas.

   En este caso las tareas tienen un nivel de relación entre sí vinculadas a través datos en común (los datos de la factura).

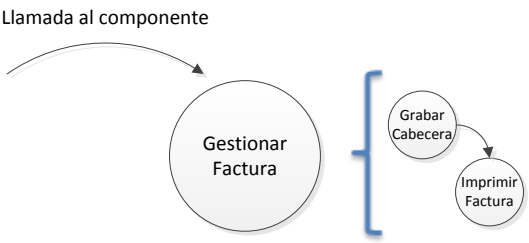
   Se ve que primero se debe ejecutar una tarea, y luego para tratar más adecuadamente los datos en común.



3. Cohesión Comunicacional (Aceptable)

   El caso describe dos tareas, que operan sobre mismos datos, pero se pueden ejecutar en cualquier orden.

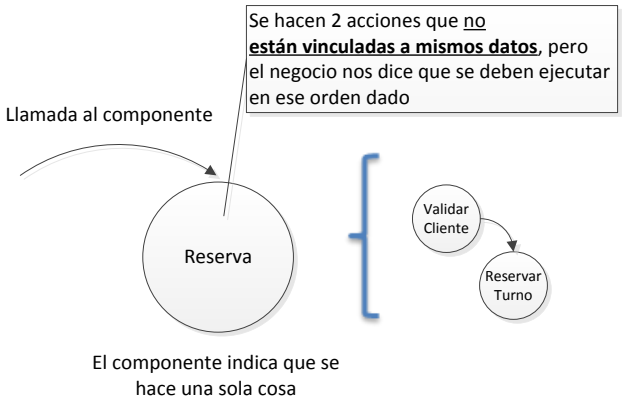
   Es comunicacional, dado que la comunicación entre ambas tareas es arbitraria, y definida por el programador. Este ejemplo es aplicable a la OO.



Vinculadas a Datos en Común:

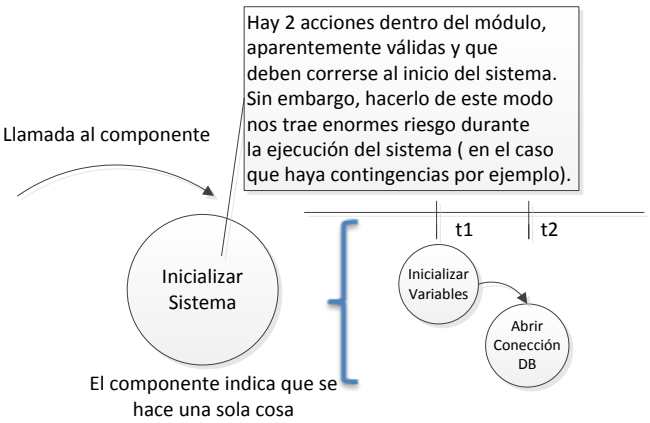
4. Cohesión Procedural (Aceptable)

   Las tareas no operan sobre mismos datos, se pueden ejecutar en cualquier orden. Es procedural, porque la comunicación entre tareas es dada previamente (usualmente por el negocio). Este ejemplo es también aplicable a la OO.



5. Cohesión Temporal (No Aceptable)

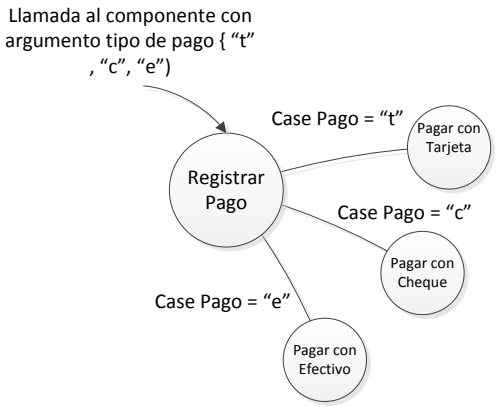
   Un módulo con tareas que deben correrse en un determinado momento (al comienzo). Habrá problemas, si hay sucesos no contemplados contingencias).



No Vinculadas ni por Datos ni por Flujos de Control:

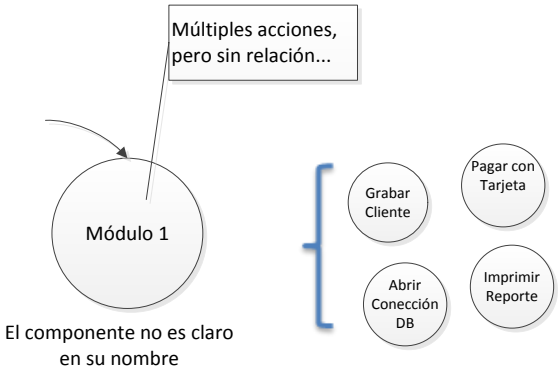
6. Cohesión Lógica (No Aceptable)

   Un módulo con tres sub-tareas, se van ejecutar dependiendo del valor de entrada. El resultado no será predecible, puede terminar de distinta forma en cada caso. Este ejemplo es también aplicable a la OO.



7. Cohesión Coincidental (No Aceptable)

   Tareas agrupadas sin relación. El módulo es contenedor sin sentido. Puede ser usado desde múltiples lugares para muchas cosas pero sin criterio. No puede reusarse.

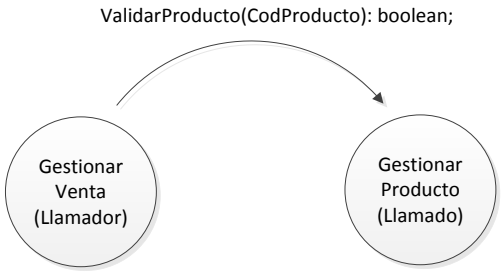


Acoplamiento (en Estructurado)

Los siguientes ejemplos son también genéricos, de modo que sean aplicables a cualquier tipo de componente. Para el caso de la Orientación a Objetos, más adelante habrá ejemplos especiales.

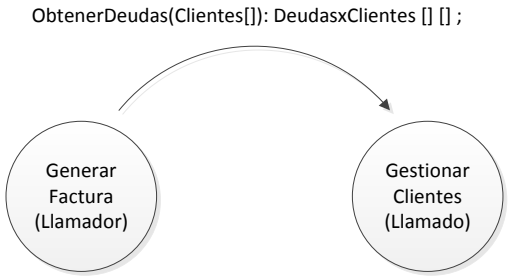
1. Acoplamiento de Datos (Aceptable)

   La interacción de los dos componentes muestra que uno llama al otro con parámetros simples (primitivas de lenguaje).



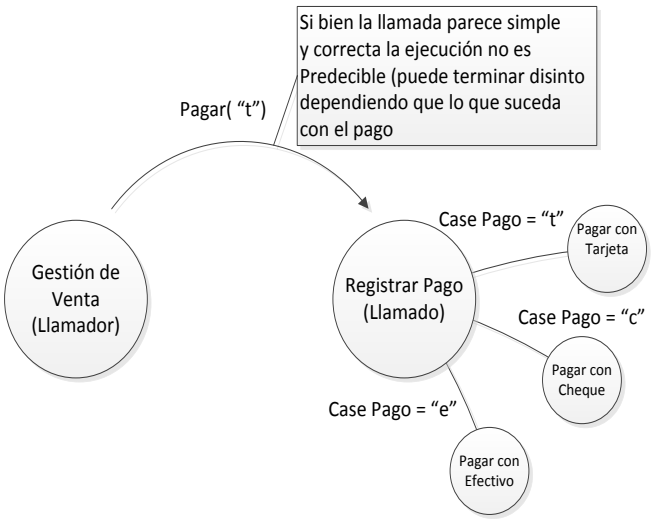
2. Acoplamiento Estampado (Aceptable)

   El ejemplo muestra cómo se pasan estructuras complejas en la llamada y en el retorno (en este caso) entre ambos componentes. El acoplamiento sube (respecto de la categoría anterior) debido a que ambas partes deben saber y manejar la estructura enviada o devuelta.



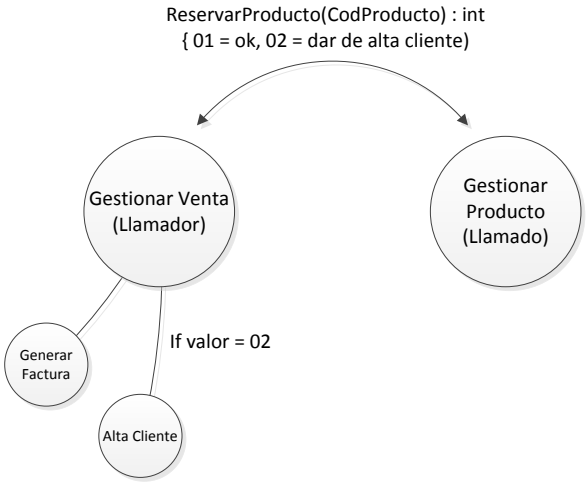
3. Acoplamiento de Control (No Aceptable)

   Se muestra una llamada simple con un argumento de dato simple. Sin embargo el acoplamiento es de control ya que la ejecución no es predecible (puede cambiar debido al tratamiento del parámetro) trayendo errores en tiempo de ejecución. Aquí este tipo de acoplamiento se relacionada directamente con el tipo de cohesión lógica (tampoco aceptable). La cohesión lógica es lo que sucede dentro del llamado, y el acoplamiento de control es lo que sucede desde el llamador. **Solución:** dividir en módulos más pequeños y evitar el parámetro de control.



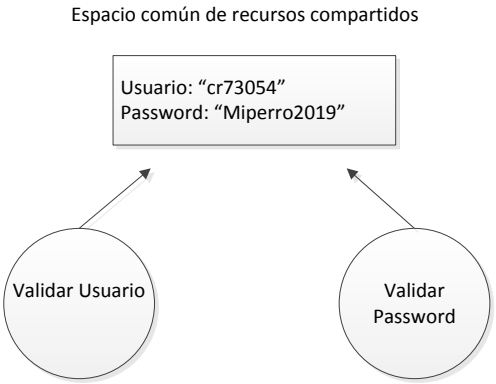
4. Acoplamiento con Inversión de Autoridad (No Aceptable)

Aquí tenemos algo también aparentemente inofensivo, sin embargo, tampoco es aceptable ya que genera una ejecución no predecible en el llamador (a diferencia del caso anterior). **Solución**: evitar que el llamado devuelva esos parámetros de control y reorganizar la lógica del llamador dividiéndola y resolviendo la secuencia o decisión antes.



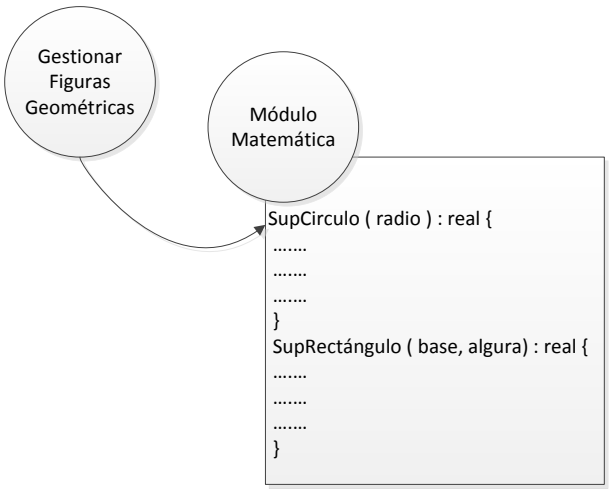
5. Acoplamiento Común o "Common" (No Aceptable)

El ejemplo muestra un área común accedida por módulos en forma global. Este tipo de acoplamiento aparece en variables o recursos compartidos, o de acceso globalizado (todo aquel que no sea de alcance local). Cuando tenemos este tipo de acoplamiento, corremos el riesgo que el dato se contamine (por una escritura de otro módulo que no controlamos, obteniendo lo que se llama lectura sucia. **Solución**: evitarlo lo más posible o hacer o revisar las lecturas con mayor frecuencia.



6. Acoplamiento Patológico (No Aceptable)

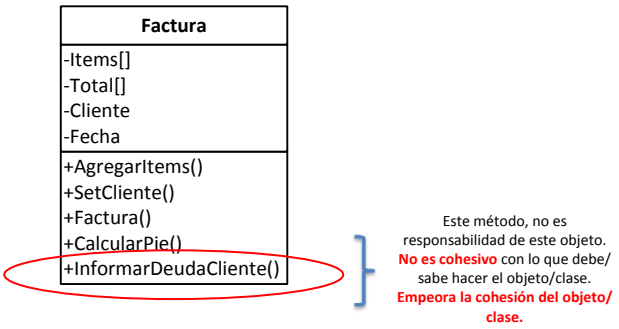
Es el peor escenario. Era algo usado en lenguajes de bajo nivel o con menor organización y mantenibilidad. Hoy en día este tipo de práctica no es aceptable dado que todo el código se vuelve muy difícil de seguir, leer y mantener. **Solución**: con los lenguajes actuales, no tiene sentido su uso. De tenerlo se recomienda normalizar la forma de acceso para no acceder directamente.



Cohesión (en OO)

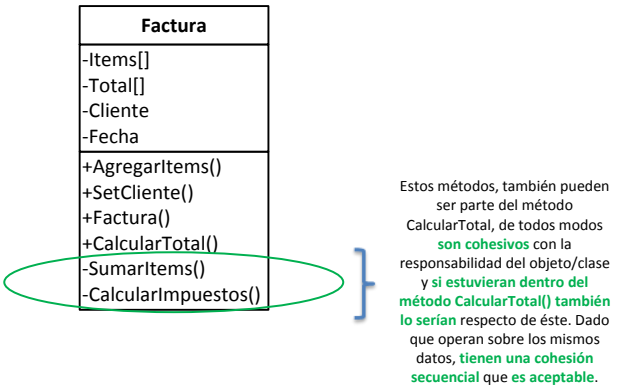
- **Cohesión del objeto respecto de sus responsabilidades**:

  El ejemplo muestra un método que claramente no se corresponde con la responsabilidad del objeto/clase.



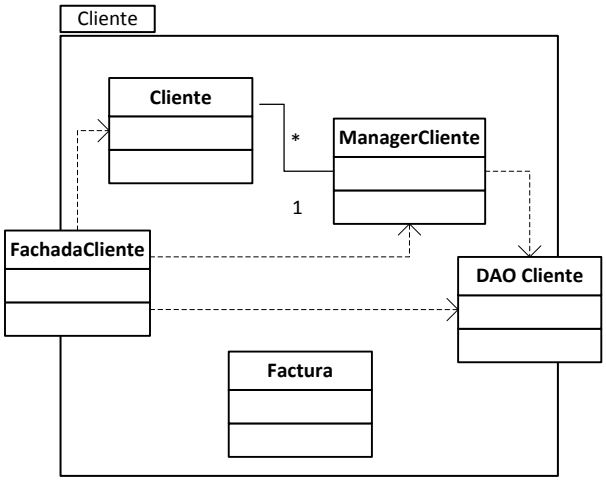
- **Cohesión en cada método del objeto**:

  Este ejemplo que muestra cohesión dentro del objeto/clase para con los métodos en general, o para con métodos que internos a otro. El ejemplo muestra un caso aceptable.



- **Cohesión dentro del paquete que contiene a las clases**:

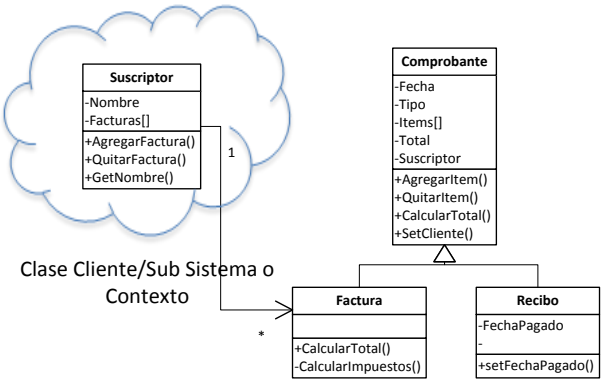
  Se aplica el concepto de cohesión a la **relación que guardan las clases dentro de un paquete que las contiene**, tal como se verá más adelante en el conjunto de principios SOLID +, visto en la unidad siguiente.



Apéndice: Recomendaciones para mejorar el Acoplamiento en OO

Referencia a Clases Base

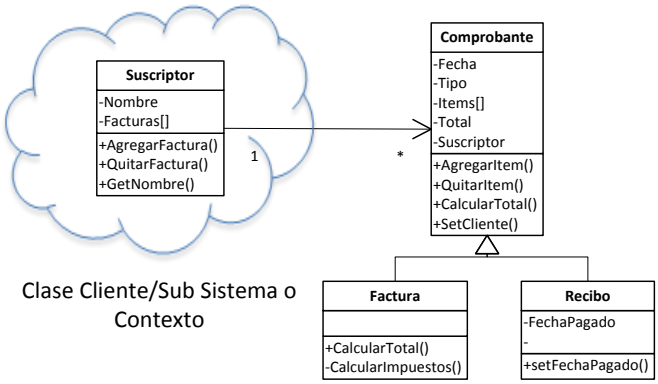
Un objeto/clase cliente, sub sistema o contexto hace referencia a un objeto mediante una referencia a una clase derivada en lugar de utilizar la referencia a la clase base.



- ➔ ¿Por qué es el problema?
- ➔ ¿Cómo puede remediarse?

Mejor Diseño:

Este caso es un mejor diseño, dado que el objeto/clase cliente (sus sistema o contexto), tiene relación con el objeto/clase base, permitiendo un diseño semánticamente más simple y ordenado (tendría una sola colección de comprobantes, que podría buscar y recuperar de una manera general con un desarrollo más sencillo.

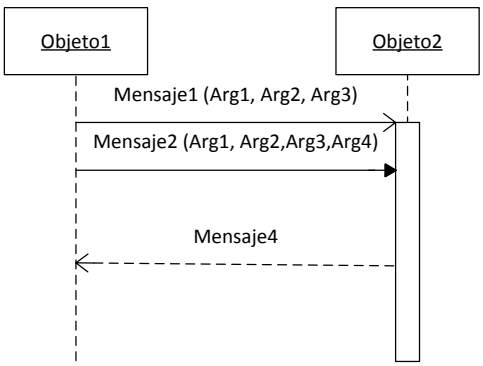


Conexiones Estrecha (Propuesta de Meyer)

Ancho de Interfaz

Se establece a través del número de conexiones que enlazan dos objetos, del número de mensajes intercambiados, y del número de parámetros de esos mensajes.

Una forma de medir estas interacciones es a través de un diagrama de secuencias.



Otras Recomendaciones

Conexiones Directas	El acoplamiento entre dos objetos es más comprensible y menos complejo si el desarrollador no tiene que hacer referencia a otro objeto para comprender la conexión original.
Estados Compartidos	Como se mencionó en el caso del Acoplamiento Common, en condiciones normales, el sistema no debe tener estados internos compartidos, ya que es una violación al principio de encapsulamiento. En ocasiones esto se realiza pero con propósitos específicos.
Sustitución a Igual Interfaz	Cualquier objeto debe ser reemplazable por otro con la misma interfaz.
División de una Clase Cohesiva	Si dividimos una clase que altamente cohesiva, las resultantes serán clases fuertemente acopladas.
Unión de clases poco acopladas	Si se juntan clases con bajo acoplamiento, la clase resultante tendrá baja cohesión.