Programación Orientada a Objetos
Descomposición Modular
(apuntes del profesor)

Juan Antonio Romero del Castillo (aromero@uco.es)
Departamento de Informática y Análisis Numérico
Universidad de Córdoba

15 de noviembre de 2023

Índice general

Capítulo 1

Descomposición. Abstracción y especificación: el camino hacia la orientación a objetos

1.1. Introducción

Parte del apartado 1.26 de [DD03].

- La orientación a objetos es una manera natural de pensar en la vida real y en la labor de programar.
- Entonces, ¿por qué no empezamos programando con objetos?. La respuesta es: por simplicidad de aprendizaje, puesto que aún usando objetos, éstos se componen de piezas de programas estructurados. Por ello es conveniente conocer antes los principios básicos de la programación estructurada.

1.1.1. Objetos del mundo real

Algunos aspectos del mundo real que se usan de forma natural en la programación orientada a objetos:

- En el mundo real hay objetos/entidades: personas, animales, libros... Es habitual pensar en términos de objetos.
- Son abstracciones porque las personas tenemos la habilidad de "abstraernos" de muchos detalles cuando creamos abstracciones para simplificar y comprender mejor las cosas. Por ejemplo, pensar en términos de playa en vez de granos de arena, de bosques en vez de en árboles uno junto al otro, de casas en lugar de ladrillo sobre ladrillo, etc.
- Todos los objetos del mundo real tienen un comportamiento. Y dichos objetos comprenden mejor interactuando con ellos: conversando con ellos, usándolos, mirándolos, etc.

- Todos tienen **atributos**: tamaño, forma, color, etc.
- Todos presentan un **comportamiento** que los definen u operaciones que realizan: la pelota bota, el coche acelera, etc.
- Así, vemos que un objeto se entiende mejor estudiando sus atributos y su comportamiento.
- Otro aspecto importante de estos objetos del mundo real es que se relacionan unos con otros: se parecen entre si, unos se hacen utilizando otros, se comportan parecido, comparten atributos, etc.

1.1.2. La POO

Aplicar esta herramienta tan interesante que es la abstracción en la labor de programar es una de las ideas básicas del paradigma de la programación orientación a objetos.

Dicho paradigma centra la labor de programar en los objetos y sus distintas clases que nos podemos encontrar en nuestros programas, la relación entre ellos, su comportamiento, etc.

Algunas consideraciones:

- Es una forma más natural y cercana a la realidad de programar.
- La base de la programación estructurada es la función, que es más dificil de comprender.
- La POO usa como base la **clase** que integra los siguientes conceptos:
  - Representa un objeto del mundo real.
  - Tiene atributos.
  - Tiene comportamiento.
  - Tiene entidad por si misma.
  - Tiene sentido por si misma, se entiende, se comprende fácilmente.
  - Se puede, por tanto, reutilizar en distintos problemas.
  - Se pueden establecer de forma natural relaciones de distintos tipos entre objetos, etc.

La POO es una forma natural de modelar problemas reales mediante programación.

Existen criterios para evaluar si un método, lenguaje o herramienta son orientados a objetos o no. El capítulo 2 de [Mey99] describe los elementos fundamentales que todo LPOO debe tener (es una lectura recomendada al alumno). En dicho capítulo se describen uno a uno dichos elementos: clases, aserciones, ocultación de la información, excepciones, genericidad, herencia, polimorfismo, ligadura dinámica, etc. que nosotros iremos desarrollando a lo largo de la asignatura.

1.2. Descomposición y Abstracción.

Una de las tareas más importante en el desarrollo de software es la descomposición. La descomposición de un problema en módulos (la modularidad o modularización) es algo que se utiliza desde los comienzos de la programación. Pero no todos los métodos de descomposición modular son buenos.

En este apartado veremos como la abstracción de datos (los TADs) implica un método natural y de calidad de descomposición modular. Pero para llegar ahí, primero tenemos que saber **qué es un buen método de descomposición modular**.

1.2.1. Criterios, reglas y principios de descomposición modular

Del capítulo 3 de [Mey99] podemos extraer estos criterios, reglas y principios de descomposición modular. La idea aquí es mostrar como la POO además de las ventajas que hemos visto que presenta, es también una buena técnica de descomposición modular. El alumno debe leer esta sección completa y analizar cada uno de estos criterios, reglas y principios de descomposición modular.

- Se puede resaltar la importancia de la descomposición en la labor del programador diciendo que precisamente la tarea de un programador es esa: realizar una descomposición de calidad.
- La calidad de una descomposición modular, al igual que la calidad del software, no es un algo subjetivo o difuso. El propósito de esta sección es refinar y formalizar la definición de este concepto.
- Haremos referencia al ejemplo del tema anterior del videoclub en los siguientes apartados dejando ver que la POO hace una descomposición modular buena, mientras que con la TDD cuesta más trabajo.

A partir de un problema desarrollamos una solución software utilizando un método de desarrollo. Ese método concreto pertenecerá a una metodología de desarrollo que será el cuerpo teórico que la defina.

El software resultante será un sistema formado normalmente por varios módulos que se interelacionan entre sí. Los módulos y su forma de relacionarse se denomina **arquitectura del sistema**.

El diseño de los módulos que compondrán la arquitectura del sistema es una labor muy importante.

1.2.2. Criterios/Requisitos para una buena descomposición modular

Un método de descomposición modular para ser bueno debe satisfacer los siguientes 5 requisitos fundamentales:

1. **Descomposición modular**
   - **Def.** Un método de construcción de software debe ayudar en la tarea de descomponer el problema de software en un pequeño número de subproblemas menos complejos, interconectados mediante una estructura sencilla, y suficientemente independientes para permitir que el trabajo futuro pueda proseguir por separado en cada uno de ellos.
   - **Comentarios.** Existen tecnologías que de forma natural favorecen el proceso general de descomposición de un problema. La POO, las tecnologías que usan patrones de diseño, etc. Otras tecnologías, no.
2. **Composición modular**
   - **Def.** Un método satisface la composición modular si favorece la producción de elementos software que se puedan combinar libremente unos con otros para producir nuevos sistemas, posiblemente en un entorno bastante diferente de aquel en que fueron desarrollados inicialmente.
   - **Comentarios.** Los módulos son independientes y fáciles de combinar entre si para varias aplicaciones. Bibliotecas de programas, comando pipe/pipeline (|) de la Shell de UNIX, etc. Como contraejemplo los módulos que dependen de la aplicación en la que se ha desarrollado y no se pueden usar en otras.
3. **Comprensibilidad modular**
   - **Def.** Un método favorece la Comprensibilidad Modular si ayuda a producir software en el cual un lector Humano puede entender cada módulo sin tener que conocer los otros, o, en el peor caso, teniendo que examinar sólo unos pocos de los restantes módulos.
   - **Comentarios.** Afecta mucho a la compresión y al mantenimiento. Se suele asumir que si nos resulta dificil explicar qué hace o para qué se ha creado un módulo, ese módulo no está bien diseñado (lo mismo ocurre a otros niveles con los procedimientos o funciones). Los módulos demasiado grandes tampoco ayudan. Además, los módulos demasiado grandes son poco reutilizables.
4. **Continuidad modular**
   - **Def.** Un método satisface la Continuidad Modular si en el sistema resultante desarrollado con ese método, un pequeño cambio en la especificación o en los requisitos provoca sólo cambios en un único módulo o en un número reducido de ellos.

- **Comentarios.** El mantenimiento es una fase muy importante del ciclo de vida del software. Si no nos preparamos para él, fracasaremos. Si nuestros módulos no facilitan el mantenimiento, no serán unos buenos módulos ya que el cambio en cualquier fase del desarrollo y durante el propio mantenimiento es algo bastante frecuente.

5. **Protección modular**

   - **Def.** Un método satisface la Protección Modular si produce arquitecturas en las cuales el efecto de una situación anormal que se produzca dentro de un módulo durante la ejecución queda confinado a dicho módulo o en el peor caso se propaga sólo a unos pocos módulos vecinos.
   - **Comentarios.** Un error dentro de un módulo debe ser tratado en dicho módulo y solucionado allí si es posible. Si un error dentro de un módulo no es detactado por el módulo y siguen ejecutandose otros módulos arrastrando el error, éste será dificil de corregir y de impredecibles consecuencias.

1.2.3. Reglas generales para una buena descomposición modular

Cuando vayamos a hacer una descomposición, para que el resultado sea de calidad, podemos guiarnos de estas reglas. Cada regla afecta a uno o a varios de los criterios anteriores. Estas reglas deben seguirse para asegurarnos cumplir los requisitos anteriores.

1. **Correspondencia directa**

   - **Def.** La estructura modular obtenida debe ser compatible con la estructura modular del dominio del problema.
   - **Comentarios**. Si ya se ha estructurado el problema en módulos, se han identificado, etc., su correspondencia con módulos software ayudará mucho en el desarrollo del sistema.

2. **Pocas interfaces**

   - **Def.** Un módulo debe comunicarse con el menor número posible de módulos.
   - **Comentarios**. Cada módulo debe intentar ser lo más independiente posible, eso ayuda en genaral a la buena descomposición. Si un módulo se comunica o interactúa con muchos módulos es que depende de ellos en gran medida y esto, en general, no es lo ideal.

3. **Pequeñas interfaces**

   - **Def.** Los módulos deben intercambiar la menor información posible.

- **Comentarios**. "Pequeñas interfaces"tendría la misma argumentación que el apartado anterior "pocas interfaces".

4. **Interfaces explícitas**

   - **Def.** Las interfaces deben ser obvias a partir de su simple lectura.
   - **Comentarios**. La claridad del módulo (la autodocumentación que veremos luego como principio), afecta en gran medida a su calidad y, por tanto, es determinante para que un módulo sea bueno. Lo mismo ocurre si un módulo debe llamar a otro, debe entenderse de forma lógica el motivo de esa llamada.

5. **Ocultación de la información**

   - **Def.** El diseñador de cada módulo debe seleccionar un subconjunto de propiedades del módulo como información oficial sobre el módulo para ponerla a disposición de los autores de módulos clientes.
   - **Comentarios**. El resto de información será privada (encapsulada, oculta) y no debe hacerse pública puesto que no es necesario que se conozca para el uso del módulo, es más, su conocimiento puede resultar contraproducente puesto que será información de bajo nivel de la cual el usuario no debe depender.

1.2.4. Principios para una descomposición modular de calidad

A partir de las reglas y criterios anteriores de pueden derivar algunos principios a seguir para una descomposición modular de calidad. Estos principios refuerzan, repiten y concretan en algunos casos lo ya dicho.

1. **Unidades modulares lingüísticas**

   - **Def.** Los módulos deben corresponderse con las unidades sintácticas del lenguaje de programación que se utilice.
   - Por ejemplo, si hacemos un módulo necesitamos un lenguaje que tenga un buen soporte de módulos. Que se puedan compilar por separado, etc. De otra forma habría que hacer **traducciones** indeseables, adaptaciones complejas y costosas. El esfuerzo y la complejidad del diseño sería mayor y mayor por tanto la posibilidad de error.

2. **Auto-documentación**

   - **Def.** El diseñador de un módulo debiera esforzarse por lograr que toda la información relativa al módulo forme parte del propio módulo.
   - El ideal sería que la simple lectura del código del módulo es su propia documentación.

3. **Acceso uniforme**

   - **Def.** Todos los servicios ofrecidos por un módulo deben estar disponibles a través de una notación uniforme sin importar si están implementados a través de almacenamiento o de un cálculo. La uniformidad del sistema es importante. Ya que el usuario hace un esfuerzo por entender el módulo, no le cambiemos la nomenclatura, las reglas básicas, etc. en el otro módulo.

4. **Principio Abierto-cerrado**

   Puede parecer contradictorio pero son elementos de naturaleza diferente.

   - **Def.** Los módulos deben ser a la vez abiertos y cerrados.
   - Abierto: esforzarse en diseñar el módulo de manera que quede abierto a facilitar la posterior modificación, ampliación, etc. (considerar, como hemos dicho antes, que el 70 % del coste del software se dedica al mantenimiento). Para ello se diseñan arquitecturas abiertas, se siguen estándares y convenciones, etc.
   - Cerrado: El módulo estará cerrado si está disponible para ser usado e integrado en el sistema. Su diseño abierto no impedirá que el módulo quede completado y listo para ser usado en la aplicación para la que se ha creado.

5. **Elección única**

   Relacionado con el principio abierto-cerrado y la ocultación de la información.

   - **Def.** Siempre que un software deba admitir un conjunto de alternativas, habrá un único módulo que conozca su lista completa.
   - Ejemplo. Muchas veces haremos un software que, por ejemplo, usa un tipo de objeto pero se prevé que, en un futuro, podrá usar objetos de otro tipo. Debe prepararse el software para este cambio tan previsible. Así, por ejemplo, en una tienda se venden libros, pero en un futuro se podrán vender discos, ordenadores, etc.
   - Se debe cerrar el sistema y trabajar con los libros, pero dejarlo abierto y preparado para que se amplie con otros productos.
   - Con herencia, por ejemplo, se puede declarar una clase basa abstracta de la que deriven todos los futuros tipos de objetos (nuevas clases). El polimorfismo y la vinculación dinámica permite que el resto del código del programa admita estos nuevos objetos sin modificar nada. La clase base abstracta es el punto de elección única.
   - Así, la ampliación no afecta al resto de módulos que siguen trabajando con figuras. Para el resto de módulos esas ampliaciones y particularidades de cada una están ocultas.

1.2.5. Conclusiones

- La adecuada descomposición, la modularidad, de un proyecto software es fundamental.
- No todas las descomosiciones modulares satisfacen los criterios, reglas y principios de una buena descomposición modular. Hay que tener cuidado en ello y hacer buenos diseños.
- Existen tecnologías de programación que no favorecen la buena descomosición.
- POO y la abstracción de datos en general, es un método de descomposición que satisface todo lo dicho en esta sección.

Bibliografía

[DD03] Harvey M. Deitel and Paul J. Deitel. *Cómo Programar en C++*. Pearson Educación, México, 4 edition, 2003.

[GHJV03] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Patrones de diseño*. Pearson Educación, S.A., Núñez de Balboa 120, Madrid, 2003.

[LG01] Barbara Liskov and John Guttag. *Program Development in Java: Abstraction, Specification, and Object-Oriented Design*. Addison-Wesley, USA, 1 edition, 2001.

[Mey99] Bertrand Meyer. *Construcción de Software Orientado a Objetos*. Prentice Hall Iberia, S.R.L., Madrid, 2 edition, 1999.

[Sch04] Herbert Schildt. *C++. A Beginner's Guide. Second Edition*. McGraw-Hill/Osborne, Emeryville, California 94608, USA, 2004.