Programación (2)

Alejandro Teruel

23 de abril de 2013

Contenido: Programación dirigida por contratos y programación dirigida por casos de prueba. Refactorización (1).

Programación dirigida por contratos

La programación dirigida por contratos consiste en que la programación de ciertos bloques de construcción de programas, notablemente rutinas o métodos, tipos abstractos de datos y clases comienza con la elaboración de sus especificaciones formales o *contratos*. Así la codificación de una rutina en un lenguaje de programación sería precedida por la elaboración de un *contrato* que especifica formalmente la pre-condición de la rutina (condición que cumplen sus insumos al invocarse) y su postcondición (condición que cumplen sus resultados al culminar su ejecución).

El contrato para una clase puede consistir en la especificación formal del Tipo Abstracto de Datos que implementa la clase , o puede consistir en el *invariante* de la clase y los contratos para cada uno de sus métodos. El contrato especifica que:

1. Al construirse o al culminar la ejecución de uno de sus métodos, un objeto de esa clase siempre queda en un estado en que cumple con la invariante de la clase.
2. Si se cumple la precondición de un método y el invariante de la clase cuando se invoca el método en un objeto de esa clase, al culminar la ejecución del método se cumple la postcondición del método y la invariante de la clase.

El código se deriva formal o informalmente a partir de las especificaciones y, en todo caso, permite verificar formalmente la *correctitud* del programa respecto a su especificación.

Programación dirigida por casos de prueba

En la programación dirigida por casos de prueba, antes de escribir código se escriben casos de prueba que debe satisfacer el código al ejecutarse. Más concretamente:

1. Se especifica uno o más casos de prueba.
2. Se selecciona uno de los casos de prueba especificados pero no implementados, se escribe y se ejecuta. El caso debe fallar, dado que aún no se ha escrito código que lo satisfaga (esto se hace como chequeo del caso).
3. Se escribe el código (mínimo) que se espera satisfaga el caso seleccionado y escrito.
4. Se ejecuta el caso de prueba seleccionado.
5. Si la ejecución aprueba el caso de prueba, se ejecutan los demás casos de prueba previamente aprobados (esto para chequear que el nuevo código no "rompe" con las aprobaciones anteriores.
    - Si los casos previos siguen aprobándose se pasa a (6), de lo contrario se depura el

código y/o los casos y se pasa a (2)

Si la ejecución no aprueba el caso de prueba, se depura[1] el código y/o el caso de prueba y se pasa a (4)

6. Se "refactoriza" el código, cuidando que siga aprobando los casos de prueba.
7. Se pasa a (1), a (2) o, sólo en caso que se hayan escrito y aprobado todos los casos de prueba se da por terminado el proceso de codificación.

Hay algunas variantes respecto al paso 2. Algunos proponentes de este estilo de programación prefieren seleccionar primero el caso más sencillo o elemental de los que se han identificado, otros prefieren seleccionar primero el caso "normal" (posponiendo los casos excepcionales y el manejo de errores), otros el que proporciona más valor al producto que se desarrolla para el cliente (puede coincidir con el caso "normal") y otros (los menos) el caso que reduce más los riesgos asociados al desarrollo. En todo caso, aconsejaría probar estas diferentes variantes para ver con cuál se siente más cómodo el programador y/o el equipo de desarrollo, pudiendo incluso variarse la preferencia de acuerdo con las condiciones del proyecto.

Ejemplo:

Desarrollar un método que intercambie los dos últimos caracteres de una cadena de caracteres[2].

Algunas observaciones adicionales

1. El uso correcto de esta técnica produce pequeños avances de desarrollo.
2. Cuando se aprovechan bibliotecas externas es importante no hacer incrementos tan pequeños que representen casos de prueba de la biblioteca - a menos que (a) se esté aprendiendo a usar la biblioteca, (b) se tiene razón para desconfiar de la correctitud de los elementos dela biblioteca o (c) los elementos de la biblioteca cumplen parcialmente con los requerimientos que se tienen sobre ellos.
3. Según sus proponentes esta técnica motiva al programador a pensar en el desarrollo de software en término del desarrollo de elementos pequeños, muy cohesionados y con bajo acoplamiento que faciliten la prueba y la codificación.
4. Algunos autores reportan dificultades para aplicar esta técnica al desarrollo de bases de datos e interfaces con el usuario.
5. Algunos autores advierten que esta técnica no debe aplicarse al desarrollo de variables y métodos privados a las clases.
6. Se advierte que debe prestarse atención a documentar bien los casos de prueba y a evitar *casos frágiles*, es decir casos que después no logran incluirse al escalarse el código *[Este concepto debe aclararse en el futuro -en este escrito no está bien explicado]*.

Programación dirigida por contratos y Programación dirigida por casos de prueba

Algunos autores prefieren obviar los contratos o en todo caso desarrollarlos como *documentación* en un

1 Algunos proponentes de esta técnica prefieren no invertir mucho tiempo en la depuración de código. En tal caso, descartan la ve descartar la versirsión actual del código y regresan a la versión anolver a intentar hacer bien el paso fallido.

2 Se recomienda fuertemente el video que desarrolla este ejemplo en http://www.youtube.com/watch?v=Xc3d6j8Rm_I

paso posterior a la programación dirigida por casos de prueba. Les gusta trabajar en el estilo *bottom-up* que les proporciona el centrarse primero sobre un caso concreto a la vez, y argumentan que les permite ir desarrollando de manera más natural, y paso a paso, un entendimiento gradual y concreto del problema a resolver. Les resulta mejor, en todo caso, elaborar una especificación general después de haber programado los casos de prueba.

Otros autores prefieren elaborar un contrato primero, y usar el contrato para extraer posibles casos de prueba. Les gusta trabajar en el estilo *top-down* que les proporciona concebir primero la especificación general, revisarla al descomponerla en casos de prueba para luego enfocarse en los detalles de la construcción (programación) detallada.

También es posible enfoques híbridos tal como lo sería especificar los casos "normales" y luego enriquecerlos a partir de casos de prueba de casos especiales.

Ejercicios y ejemplos adicionales

- Programe, en el estilo de la programación dirigida por casos de prueba, una función que chequee si una secuencia de entrada constituye un palíndrome o no.
- Programe, en el estilo de la programación dirigida por casos de prueba, un programa que traduzca números enteros expresados en dígitos a números expresados en palabras. Tal función toma como entrada un número entero como *101* y produce como resultado la cadena "ciento uno". http://www.codeproject.com/Articles/18181/Test-Driven-First-Development-by-Example constituye un ejemplo interesante de este ejercicio. En mi opinión el estilo en que se desarrolla el ejemplo es extremo, dado que el autor insiste en agregar y programar un caso de prueba a la vez.

Lecturas adicionales

- La entrada sobre *Test Driven Development* en Wikipedia constituye una excelente y breve introducción al tema.
- Bertrand Meyer: *Test or spec? Test and spec? Test from spec!* EiffelWorld column, Septiembre 2004. http://www.eiffel.com/general/column/2004/september.html. Una interesante opinión sobre la compatibilidad de la programación dirigida por contratos y la programación dirigida por casos de prueba.

Lecturas avanzadas

- http://www.codeproject.com/Articles/25738/Designing-Application-Using-Test-Driven-Developmen Aplica la técnica de la programación dirigida por casos de prueba al desarrollo de una base de datos. Personalmente encuentro confuso el ejemplo.
- Roy Osherove: *Write Maintainable Unit Tests That Will Save You Time And Tears* http://msdn.microsoft.com/en-us/magazine/cc163665.aspx

Refactorización (1)

En la programación dirigida por casos de prueba, para aprobar un nuevo caso de prueba se escribe el mínimo código necesario. Rápidamente esto deteriora la calidad del código, por lo que la técnica incluye un paso de *refactorización:*

> **Refactorización:** Modificar la estructura interna de software,conservando invariable su conducta observable, para que la nueva estructura sea más fácil de entender y cueste menos modificarla en el futuro.
>
> Martin Fowler

El programador ágil se acostumbra a dos modos de programación, uno enfocado a hacer cumplir un caso de prueba lo más rápidamente posible y otro enfocado a mejorar la calidad de su código. En este segundo modo el programador busca construcciones poco elegantes para mejorarlos. A muchos programadores, tal *separación de preocupaciones* le rinde réditos.

Se ha venido desarrollando un catálogo de características que suelen estar asociadas a construcciones poco elegantes, que se le denomina, con cierto humor, *olores sospechosos.* A cada *olor sospechoso*, se han venido asociando también acciones heurísticas que buscan mejorar el código escrito eliminando esos *síntomas* de inelegancia.

Un primer *olor sospechoso* lo constituye *código repetido;* la corrección recomendada es estudiar la posibilidad de transformar ese código en una rutina (probablemente métodos privados), de modo de no duplicar el código sino la invocación al código, acción que en la literatura de refactorización recibe el nombre de *Extraer Método*. Hacer esto en forma automática no mejora automáticamente la legibilidad y mantenibilidad de código, pues hay que siempre buscar un alto nivel de cohesión para la nueva rutina. Fowler, por ejemplo, incluso llega a proponer que que un buen nombre descriptivo de una rutina de este tipo puede las veces de un buen comentario explicativo.

Si la duplicación se da en dos clases no relacionadas por herencia, la acción a estudiar es *Extraer Clase.*

Un segundo *olor sospechoso* lo constituye encontrarse con un cuerpo de método demasiado largo. Los proponentes de código ágil tienden a desarrollar un estilo donde lo común es encontrarse con métodos de menos de veinte líneas de largo. Nuevamente la acción "correctiva" que se propone es *Extraer Método;* en este caso se trata de sustituir tal cuerpo por una secuencia de llamadas a nuevos métodos privados -cuidando, como siempre, la cohesión de tales métodos y además su acoplamiento.

El tercer *olor sospechoso* que trataremos en esta primera aproximación se da en clases que contienen muchas varables de instanciación. En este caso, una acción posible es estudiar si no podemos particionar tales variables en distintas clases (*Extraer Clase*) o incluso si no nos convendría formar una jerarquía de herencia para introducir un número pequeño de variables en cada nueva subclase (*Extraer Subclase*).