LABO “Software Reengineering”	2 LIC INFORMATICA
Profesor Serge Demeyer - Bart Du Bois Traducido : Carlos López	Última modificación: 27 January 2006 Traducido: 12 Noviembre 2007

Sesión de Refactoring

Referencia bibliográfica: "Object-Oriented Reengineering Patterns". Serge Demeyer, Stéphane Ducasse, Oscar Nierstrasz. Colaborador Martin Fowler publicado 2003 Morgan Kaufmann.

Contexto

Estas manteniendo un sistema software que representa una simulación de una red de área local (LAN). El equipo de desarrollo ha sido muy rápido en adaptar los requisitos iniciales para el sistema entregando una versión 1.4 que contiene la funcionalidad para el primer hito. El cliente solicita añadir una nueva funcionalidad y el equipo de desarrollo se percata que el diseño no esta preparado.

Saben que eres un experto en refactorización, por eso te prestan su código para que lo refactorices apropiadamente. No esperan un diseño perfecto, esperan un diseño que permita añadir la nueva funcionalidad fácilmente. Además tienen disponibles pruebas del sistema desarrollado.

Ojear la documentación (OORP – p. 44)

Decides echar un vistazo a la documentación que viene con el sistema

1. Abre el fichero “LANSimulationDocu_es.pdf” y lee su contenido
2. Abre el fichero "toDoList_es" y lee su contenido
3. Abre la documentación (para Java: en el subdirectorio ‘java/doc’, para C++: en el subdirectorio ‘cpp/doxygen/html’) y lee su contenido.
   » *¿Cuál es tu primera impresión sobre el sistema? ¿Dónde centrarías tus esfuerzos de refactorización? Discutir con los miembros del equipo.*

Lee todo el código en 5 minutos (OORP – p. 38)

Confirmas algunas de las impresiones iniciales leyendo todo el código.

4. Carga el código en tu editor preferido y lee el código.
   » *¿Cuál es la segunda impresión sobre el sistema? ¿Estas de acuerdo con la impresión inicial? Con este nuevo conocimiento sobre el código? ¿dónde centrarías tus esfuerzos de refactorización? Discutir con los miembros del equipo.*

Hacer una instalación de prueba (OORP – p. 58)

Finalmente intenta ejecutar el código y las pruebas disponibles con él.

5. Compila y ejecuta (Para Java: abrir proyecto en Eclipse. Para C++: ‘make’ y luego ‘lan s’.)
6. Ejecuta las pruebas (Para Java: RunAs JUnit Test. Para C++: ‘make runtests’.)
7. Cambia algunas líneas en los test y en el código para verificar que los tests prueban el código que estas viendo.
8. Observa los test y comprueba que cubren todos los casos de uso.
   » *¿Crees que el código base esta ya refactorizado? ¿Qué puedes decir de la calidad de los tests: puedes empezar a refactorizar de manera segura? ? Discutir con los miembros del equipo.*

Habla con los de mantenimiento (OORP – p. 31)

Fruto de las actividades realizadas tienes bastante idea sobre lo básico del proyecto. Antes de empezar realmente a refactorizar, verifica algunas características con el equipo de desarrollo original.

9. Lanza todas las preguntas no contestadas al profesor de la sesión.
   » *Desarrolla un plan de proyecto listando a)los riesgos, b) las oportunidades de refactorización (detección de defectos), c) las actividades (plan de refactorizaciones).*

Extract Method (OORP – p. 255)

Una de las cosas que podrías haber visto es que hay una considerable cantidad de código duplicado que representa una importante lógica de dominio dentro de la clase Network. Decides primero deshacerte del código duplicado.

10. El código de “accounting” ocurre dos veces dentro de "printDocument". Elimina las copias aplicando EXTRACT METHOD.
11. El código de “logging” ocurre tres veces, dos dentro de "requestWorkstationPrintsDocument" y una dentro de " requestBroadcast". Elimina las copias aplicando EXTRACT METHOD. Nota que las tres copias no son exactamente la misma por eso tendrás que adaptar el código un poco antes de hacer la refactorización.
12. ¿Hay más código duplicado representando la lógica de dominio que debería ser refactorizado? ¿Puedes refactorizarlo usando EXTRACT METHOD ?
    » *¿Estas seguro que estas refactorizaciones no rompen el código?¿Crees que estas refactorizaciones merecen la pena? ¿La herramienta de refactorización hace un buen trabajo? Discutir con los miembros del equipo.*

Mover el comportamiento cerca de los datos(OORP – p. 190)

Tomando los métodos extraídos, nota que ninguno de ellos se refiere a atributos definidos en la clase Network. Por otro lado, estos métodos acceden a campos públicos de las clases "Node" y "Packet".

13. El método “logging” que acabas de extraer no pertenece a Network porque muchos de los datos a los que accede pertenecen a otra clase. Aplicar MOVE METHOD para definir el comportamiento cerrado de los datos con los que opera un método.
14. De la misma manera, el método "printDocument" accede a atributos de otras clases, pero no accede a sus propios atributos.
15. ¿Hay más métodos que pueden ser movidos junto con los datos con los que operan? Si existen aplica MOVE METHOD hasta que estés satisfecho con los resultados.
    » *¿Estas seguro que estas refactorizaciones no rompen el código?¿Crees que estas refactorizaciones merecen la pena? ¿La herramienta de refactorización hace un buen trabajo? Discutir con los miembros del equipo.*

Eliminar código de navegación (OORP – p. 199)

Existe todavía una porción de código duplicado, si se sigue la pista del puntero "nextNode_" hasta que se recorre completamente la red; la lógica esta duplicada en "requestWorkstationPrintsDocument" y "requestBroadcast" (y con menos grado en "printOn", "printHTMLOn", "printXMLOn"). Esta lógica duplicada es bastante vulnerable, porque accede a atributos definidos en otras clases y de hecho representa una clase especial de código de navegación.

16. Aplicar EXTRACT METHOD sobre la expresión booleana definida al final del bucle creando un método predicado "atDestination".
17. Rescribe los bucles dirigidos por "currentNode = currentNode.nextNode_" dentro de las llamadas recursivas de un método "send".
    » *¿Estas seguro que estas refactorizaciones no rompen el código?¿Crees que estas refactorizaciones merecen la pena? ¿La herramienta de refactorizción hace un buen trabajo? Discutir con los miembros del equipo.*

Transformar códigos de tipo (OORP – p. 217)

Otra porción de código duplicado se puede encontrar en "printOn", "printHTMLOn", "printXMLOn". Esta vez se trata de un duplicado condicional, el código es probable que cambie dada la funcionalidad extra de introducción de un nodo GATEWAY. Por esto merece la pena introducir una nueva subclase aquí:

18. Deberías haber notado que al mover el comportamiento cerca de los datos, que las sentencias switch dentro de "printOn", "printHTMLOn", "printXMLOn" deberían haber sido extraídas y movidas sobre la clase Node. Si no lo hiciste hazlo ahora, nombrando los nuevos métodos "printOn", "printHTMLOn", "printXMLOn".
19. Crea subclases vacías para los diferentes tipos de nodo ("WorkStation", "Printer").
20. Ajusta la invocación de los clientes a los constructores para que creen instancias de la clase apropiada.
21. Mueve el código desde cada cuerpo de la sentencia condicional a la (sub)clase apropiada eliminando eventualmente el condicional.
22. Verifica todos los accesos al atributo "type_" de Node. Mientras encuentres alguna referencia elimínala.
23. Elimina el atributo "type_"
    » *»¿Estas seguro que estas refactorizaciones no rompen el código?¿Crees que estas refactorizaciones merecen la pena? ¿La herramienta de refactorización hace un buen trabajo? Discutir con los miembros del equipo.*

Conclusion

Crees que tu tarea esta hecha, y solicitas una reunión con el equipo de desarrollo, explicándoles cómo has rediseñado su código y cómo el nuevo diseño es más adecuado para los nuevos requisitos.

24. Chequea el fichero "toDoList_es" y argumenta para cada uno de los futuros requisitos cómo tu diseño soportará los cambios.
    » *¿Hay asuntos que no has considerado ? ¿Hay refactorizaciones que parecen innecesarias ? Discutir con los miembros del equipo.*