Universidad de Burgos

Escuela Politécnica Superior

Grado en Ingeniería Informática

Diseño Anazado de Sistemas Software

Simulación de un proceso de refactorización

Alumnos	Elsa Tolín Humberto Marijuan Roberto Arasti
Tutor	Carlos López Nozal Departamento de Ingeniería Civil Área de Lenguajes y Sistemas Informáticos

Burgos, 30 de mayo de 2021

Índice de contenido

INTRODUCCIÓN

Para esta práctica vamos a realizar la refactorización del código que se nos ha sido proporcionado.

Este código representa una simulación de una red de área local (LAN). Este código ha sido desarrollado de forma rápida y el cliente ha pedido añadir una nueva funcionalidad, entonces el equipo se percata de que el diseño no está preparado para esta nueva funcionalidad y que necesita ser refactorizado para poder añadir esta nueva funcionalidad.

El código y toda la documentación se encuentra en el repositorio de la práctica: https://github.com/etc99/reafactoring_lab_session

CUESTIONES

>>¿Cuál es tu primera impresión sobre el sistema? ¿Dónde centrarías tus esfuerzos de refactorización?

Viendo como está escrito el código del sistema, a primera vista, llama la atención una gran cantidad de prints en el main, métodos bastante largos y algunos nombres de variables no muy claros. Los esfuerzos se tendrían que centrar en refactorizar los defectos que más van a afectar al manteninmiento como puede ser el código duplicado o la envidia de carácterísticas o los métodos muy largos con muchos prints. La clase que tiene más defectos a primera vista es LANSimulation.java por lo que se debería de centrar el esfuerzo en refactorizarla.

>>¿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?

Si se lee el código con más detenimiento se puede observar que tiene muchos más defectos de diseño Network.java, por lo que no estamos de acuerdo con nuestra primera impresión. Es por ello que la mejor opción es centrar todo el esfuerzo en refactorizar esta clase, ya que además en ella va gran parte del peso del funcionamiento del programa.

>>¿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.

Es posible que se haya hecho alguna refactorización, pero el código contiene un gran número de defectos de código. Es por este motivo que no parece que esté refactorizado. Los tests tienen una cobertura del código del 61.2%m, es una cobertura bastante baja.

Sobre mover el comportamiento ceca de los datos:

>>¿Estas seguro que estas refactorizaciones no rompen el código?¿Crees que estas refactorizaciones merecen la pena? ¿La herramienta de refactorizción hace un buen

trabajo?

En la clase Network no hay más métodos a los que se les pase parametros de otras clases, solo se ha encontrado String, StringBuffer (de java.lang) y Writer (de java.io).

En nuestro caso en el método simulate de la clase LANSimulation, ya habiamos aplicado algunas refactorizaciones de extract method. Así que, el método printScenarios(Network network, StringWriter report) se podría plantear extraerlo a la clase Network. Y en las clases Node y Packet no se encuentra ningun posible método a mover (sin contar los métodos que se comentaron anteriormete que reciben parametros de tipo String, StringBuffer, Writer que pertencen a paquetes externos a nuestro proyecto, y por tanto se dificulta el refactor move method).

Sobre eliminar código de navegación:

>>¿Estas seguro que estas refactorizaciones no rompen el código?¿Crees que estas refactorizaciones merecen la pena? ¿La herramienta de refactorizción hace un buen trabajo?

Estas refactorizaciones no rompen el código, ya que los tests siguen pasando, es decir, no añade ningún cambio a la funcionalidad del código. En el caso de la eliminación del código de navegación sí que merecen la pena ya que la lógica es vulnerable al acceder a atributos definidos en otras clases.

La herramienta de refactorización, al realizar la extracción del código, prepara los métodos con entrada de parámetros de tal forma que no altere el funcionamiento del código, por lo que hace muy buen trabajo.

Sobre transformar códigos de tipo:

>>¿Estas seguro que estas refactorizaciones no rompen el código?¿Crees que estas refactorizaciones merecen la pena? ¿La herramienta de refactorizción hace un buen trabajo?

Estas refactorizaciones no rompen el código al haber ejecutado las pruebas entre paso y paso, aunque sería posible en algún caso que las pruebas no tuviesen una cobertura total y se escape un error introducido al refactorizar. Estas sirven sobretodo para el mantenimiento a futuro, permiten hacer el código más legible y facilita los cambios y la depuración. La herramienta de refactorización facilita el proceso pero esto no significa que haya que hacer algunos cambios manuales o que haga algunas refactorizaciones al completo, pero en general, hace buen trabajo.

CONCLUSIÓN

Chequea el fichero "toDoList_es" y argumenta para cada uno de los futuros requisitos cómo tu diseño soportará los cambios.

Versión 1.0: Gracias a las refactorizaciones de código duplicado en printDocument sí que va a ser capaz de soportar los cambios, ya que antes de esto habría sido mucho más complicado.

Versión 1.1: Gracias a las refactorizaciones de navegación este cambio va a ser soportado ya que está preparado para que no sea vulnerable.

Versión 1.2: Al mover el método y definir un comportamiento cerrado de los datos con los que opera no va a haber ningún problema al implementar esta función.

Versión 1.3: En la clase Network.java se realizó una extracción de un método printAccounting, este registra el autor por lo que sí que va a soportar la nueva funcionalidad.

Versión 1.4: Se realizó una extracción de método que denominamos atDestination y esta recibe el nodo y el paquete. Esta refactorización va a facilitar la implementación de este cambio.

Versión 2.0: Sí, ya que tiene un método preparado en Network.java que va a interpretar por lo que esta funcionalidad se va a poder implementar.

Versión 2.1: Al tener los nodos implementados en una jerarquía de herencia, se añadiría una clase Gateway que herede de nodo, por lo que sí se podría implementar con más facilidad esta funcionalidad gracias a las refactorizaciones realizadas.

Versión 3.0: De esta nueva funcionalidad no estamos seguros ya que sería más compleja de añadir y el código correspondiente a esta parte está en el main, y en el main no hemos realizado ninguna refactorización.

» ¿Hay asuntos que no has considerado ? ¿Hay refactorizaciones que parecen innecesarias ?

Hay refactorizaciones en otras clases que se podrían realizar, por lo que sí que hay partes que no se han tenido en cuenta, aún así se han realizado las refactorizaciones más necesarias para una mejor implementación de las nuevas funciones. No consideramos que haya refactorizaciones innecesarias dentro de las que hemos realizado, ya que estas han permitido que la compresión del código sea más facil y que los desarrolladores que implementen las nuevas funcionalidades puedan realizarlas sin que se rompa el código.