CS427 MP3: Reverse Engineering & Refactoring

September 2010

Context

You are facing a software system which represents a simulation of a local area network. The development team has been very fast in accommodating the initial requirements for the system and has been able to release version 1.4 of the system, which contains all the functionality for the first milestone. However, the customer now requests for the remaining functionalities and the development team fears that the current design is not up to the task.

Having heard of your refactoring expertise, they hired you to have a look at their code and refactor it appropriately. They do not expect a perfect design, yet they want to be able to add the remaining functionalities easily. They told you that they have regression tests available.

Questions in italics are for you to think about. Questions in light magenta are for you to Answer and submit.

Getting Started

1. Download the MP3.zip file from the class wiki.

2. In Eclipse, select File > Import.

3. In the window that pops-up, expand the “General” folder and select “Existing Projects into Workspace”.

4. Click on the “Select archive file” radio button.

5. Select the location of the *MP3.zip* file that you have just downloaded.

6. At this stage, the project should have been imported and an unzipped version has been created in your workspace folder.

7. Check it into your group’s repository

Skim the Documentation

You decide to first have a look at the documentation that comes with the system.

8. Open the file *LANSimulationDocument.pdf* and read its contents.

9. Open the file *toDoList* and read its contents.

10. Generate the *javadoc* for the project.

   (a) In the Package Explorer, select the *javadoc.xml* file under the MP3 project.

   (b) Right-click on it and select Run As > Ant Build.

(c) Right-click on the MP3 project and select Refresh. You should see the *javadoc* folder in your project.

(d) Right-click on the javadoc folder and select Team > Add to svn:ignore. . . . In the dialog that pops-up, click OK. This step prevents the javadoc folder from being committed to the repository.

(e) Expand the javadoc folder and select the *index.html* file. Read through the documentation.

What are your first impressions about the system? Where would you focus your refactoring efforts? Discuss with your group partner.

Read all the Code in 5 minutes

Next, you confirm some of your initial impressions by reading the source code.

11. From within Eclipse, read the code. Use the features of Eclipse to help you navigate the code quickly. For instance, if you are in the middle of a portion of code and want to see where a *class/method* is defined, hold on *ctrl (or the apple key)* and click on the word.

What are your second impressions about the system? Where do you agree or disagree with you first impressions? Having a better feeling about the code, where would you focus your refactoring efforts? Discuss with your group partner.

Do a Mock Installation

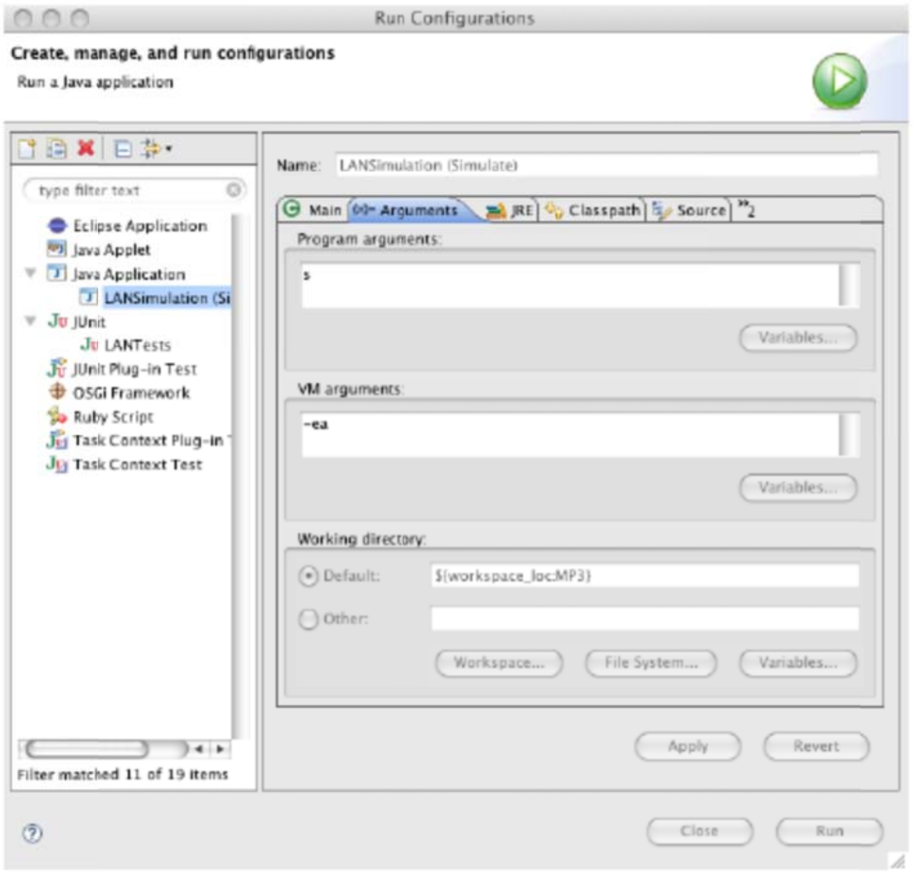
Finally, you try to run the code and the regression tests that come along with it.

12. Try to compile and run the *LANSimulation.java* file. You need to pass it the character “s” as a program argument and turn on assertions on the JVM. *See Figure 1.*

13. Run the regression tests in the *LANTests.java* file. The first time you run the tests, one of them will fail. You need to perform an additional step manually to get it to pass.

14. Change a few lines here and there in the regression tests and the code to verify whether the regression tests do test the code you are looking at. This step is important so that you can verify if that part of the code indeed does what you think it should.

15. Have a look at the regression tests and see whether they cover all the use cases.



Name: LANSimulation (Simulate)

Program arguments:
s

VM arguments:
-ea

Default: ${workspace_loc:MP3}

Figure 1: Passing a program argument and turning on assertions.

Do you feel you the code base is ready to be refactored? How about the quality of the regression tests: can you safely start to refactor? Discuss with your group partner.

For the next few sections, use the automated refactoring tools in Eclipse to perform the refactorings. In a Java file, click on the Refactor menu and select the appropriate refactoring. Look at the help file if you are not sure how to use one of them. There are some refactorings that you still have to perform by hand, but whenever possible, use the automated refactoring support in Eclipse.

Extract Method

One of the things you might have seen is that there is a considerable amount of duplicated code which represents important domain logic inside the class Network. You decide to first get rid of some of the duplicated code.

What code smell(s) do you detect? Explain by giving a concrete example from the code.

16. The accounting code occurs twice within *printDocument*. Get rid of the clones by applying an Extract Method.

17. The logging code occurs three times, twice inside *requestWorkstationPrintsDocument* and once inside *requestBroadcast*. Get rid of the clones by applying an Extract Method. Note that this time the three clones are not exactly the same so you'll first have to modify the code a bit before doing the refactoring.

18. Is there any other duplicated code representing important domain logic which should be refactored? Can you refactor it using Extract Method?

Are you confident that these refactorings did not break the code? Do you believe that these refactorings are worthwhile? Does the tool do a good job? Discuss with your group partner.

Move Behavior Close to the Data

Having extracted the above methods, you note that none of them is referring to attributes defined on the class Network, the class these methods are defined upon. On the other hand, these methods do access public fields from the class Node and Packet.

What code smell(s) do you detect? Explain by giving a concrete example from the code.

19. The logging method you just extracted does not belong in Network because most of the data it accesses belongs in another class. Apply a Move Method to define the behavior closer to the data it operates on.

20. Similarly, the *printDocument* method is also one that accesses attributes from two faraway classes, yet does not access its own attributes.

21. Are there any other methods that are better moved closer to the data they operate on? If so, apply Move Method until you're satisfied with the results.

Are you confident that these refactorings did not break the code? Do you believe that these refactorings are worthwhile? Does the tool do a good job? Discuss with your group partner.

Eliminate Navigation Code

There is still a piece of duplicated logic left in the code, namely the way we follow the nextNode pointers until we cycled through the network; logic which is duplicated both in *requestWorkstationPrintsDocument* and *requestBroadcast* (and to a lesser degree in *printOn*, *printHTMLOn*, *printXMLOn*). This duplicated logic is quite vulnerable, because it accesses attributes defined on another class and in fact it represents a special kind of navigation code.

What code smell(s) do you detect? Explain by giving a concrete example from the code.

22. Apply an Extract Method on the boolean expression defining the end of the loop, creating a predicate *atDestination*.

23. Rewrite the loops driven by the currentNode = currentNode.nextNode into a recursive call of a send method.

Are you confident that these refactorings did not break the code? Do you believe that these refactorings are worthwhile? Does the tool do a good job? Discuss with your group partner.

Transform Self Type Checks

Another striking piece of duplicated logic can be found in printOn, printHTMLOn, printXMLOn. However, this time it is a duplicated conditional, and given the extra functionality (namely the introduction of a Gateway node) one that is likely to change. Thus it is worthwhile to introduce new subclasses here.

What code smell(s) do you detect? Explain by giving a concrete example from the code.

24. Normally, you should have noticed during Move Behavior Close to Data, that the switch statements inside *printOn*, *printHTMLOn*, *printXMLOn* should have been extracted and moved onto the class Node. If you haven't done that, do it now; and name the new methods *printOn*, *printHTMLOn*, *printXMLOn*.

25. Create empty subclasses for the different types of Node that do exist (*WorkStation*, *Printer*).

26. Patch the constructor clients of Node so that they now create instances of the appropriate class.

27. Move the code from the legs of the conditional into the appropriate (sub) class, eventually removing the conditional.

28. Verify all accesses to the type attribute of Node. As long as you find any keep doing a Transform Self Type Checks or Transform Client Type Checks until you completely removed them all.

29. Remove the type attribute.

Are you confident that these refactorings did not break the code? Do you believe that these refactorings are worthwhile? Does the tool do a good job? Discuss with your group partner.

Conclusion

You feel that you're task is done. You request for a meeting with the development team, to explain to them how you redesigned their code and how this design makes its better suited for the new requirements.