

Refactoring Lab Session

This Lab Session is based on the LAN Simulation material that has been used at the university of Antwerp, Mons-Hainaut and Groningen.

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 contained all the functionality for the first milestone. However, the customer now requests the extra functionality 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 new functionality easily. They told you that they have regression tests available.

Skim the Documentation (Re-Engineering Pattern)

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

1. Open the file "LANSimulationDocu.pdf" and read its contents
2. Open the file "ToDoList" and read its contents
3. Open documentation (in the subdirectory 'java/doc') and read its contents
 - » *What are your first impressions about the system ? Where would you focus your refactoring efforts ? Discuss with your neighbour.*

Read all the Code in 5 Minutes (Re-engineering Pattern)

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

4. Load the code into your favourite text editor and read the code.
 - » *What are your second impressions about the system ? Where do you agree or disagree with your first impressions ? Having a better feeling about the code, where would you focus your refactoring efforts ? Discuss with your neighbour.*

Do a Mock Installation (Re-engineering Pattern)

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

5. Try to compile and run (open the project in Eclipse)
6. Run the regression tests (RunAs JUnit Test)
7. Change a few lines here in there in the regression tests and the code to verify whether the regression tests do test the code your looking at.
8. Have a look at the regression tests and see whether they cover all the use cases.
 - » *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 neighbour.*

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.

10. The accounting code occurs twice within "printDocument". Get rid of the clones by applying an EXTRACT METHOD.
11. 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 massage the code a bit before doing the refactoring.
12. 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 neighbour.*

Move Behaviour 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".

13. 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 behaviour closer to the data it operates on.
14. Similarly, the "printDocument" method is also one that accesses attributes from two faraway classes, yet does not access its own attributes.
15. 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 neighbour.*

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.

16. Apply an EXTRACT METHOD on the boolean expression defining the end of the loop, creating a predicate "atDestination".
17. 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 neighbour.*

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.

18. Normally, you should have noticed during MOVE BEHAVIOUR CLOSE TO DATA, that the switch statements inside "printOn", "printHTMLOn", "printXMLOn" should have been extracted and moved onto the class Node. If you didn't do that now, naming the new methods "printOn", "printHTMLOn", "printXMLOn".

19. Create empty subclasses for the different types of Node that do exist ("WorkStation", "Printer").
20. Patch the constructor clients of Node so that they know create instances of the appropriate class.
21. Move the code from the legs of the conditional into the appropriate (sub)class, eventually removing the conditional.
22. Verify all accesses to the "type_" attribute of Node. As long as you'll find any keep doing a Transform Self Type Checks or Transform Client Type Checks until you completely removed them all.
23. 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 neighbour.*

Conclusion

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

24. Check the file "toDoList" and argue for each of the future requirements how your redesign will help you accommodate the changes.
 - » *Are there issues which you did not notice ? Are there refactorings which seem unnecessary ? Discuss with your neighbour.*