CS 581A3 Software Maintenance & Evolution

Spring 2018

Assignment 4: Refactoring OSS

1. Introduction –

This assignment is related to refactoring. We had to work on the previously modified codes of jEdit and PDFSam and perform refactoring options which are detected by the JDeodrant. For each of the applications we have performed two automated refactoring and two manual refactoring. In the two Manual refactoring, one is performed on the bad smell detected by the JDeodrant eclipse plug and the other one is the smell which we feel as bad. The details of the actions performed are discussed in the next sections.

2. Automated Refactoring –

2.1. jEdit –

2.1.1. org/gjt/sp/jedit/buffer/jEditBuffer –

Bad Smell Type: God Class

Refactoring Analysis:

The eclipse automated refactoring created a new class jEditBufferUtils and moved most of the methods into this class from jEditBuffer class. jEditBuffer class now uses the method of jEditBufferUtils by creating a new object and accessing its methods.

2.1.2. org/gjt/sp/jedit/textarea –

Bad Smell Type – Feature Envy

The getOffsets() function in the StructureMatcher.java was accessing methods of TextArea more than the methods/values which already existed in the its own subclass. The automated refactoring changed the parameters of the getOffsets and replaced the Match class object with Highlight, which already is using textArea. This reduced the multiple access of TextArea class.

2.2. PDFSam –

2.2.1. Pdfsam-fx/src/main/java/org/pdfsam/ui/selection/multiple/SelectionChangedEvent::public Boolean canMove(MoveType) –

Bad Smell Type: Type Checking

Refactoring Analysis:

This smell was detected by JDeodorant. In the class SelectionChangedEvent, the canMove method had a switch case for just returning true or false for different independent cases which is considered as type checking code smell. We used automatic refactoring tool provided by eclipse to refactor the code. We selected the highlighted switch case and right clicked and went to the refactor option and then selected "move". Clicked on preview to see the changes and then selected next and finally clicked ok to perform the refactoring operation. Once the refactoring was done, we rebuild the Pdfsam using both maven build, and eclipse build and then we ran the JDeodorant again to check if the changes in effect removed the smell or not.

The method was moved to another class named MoveType (*Pdfsam-fx/src/main/java/org/pdfsam/ui/selection/multiple/move/MoveType::public Boolean canMove(MoveType)*) where it should have been in the first place. The switch conditions were based on the type of move which

were defined in the class MoveType. Therefore, it makes more sense by placing the canMove method in the MoveType class.

There were no further manual changes were required.

2.2.2. Pdfsam-core/src/main/java/org/pdfsam/pdf/PdfDocumentDescriptor –

Bad Smell Type: God Class

Refactoring Analysis:

This smell was detected by Jdeodorant. In the class PdfDocumentDescriptor, the private object of AtomicInteger was detected as code smell and the recommended refactoring was extract class. We selected the highlighted code, right clicked and selected extract class from refactor menu. The pop-up box had all the required settings selected. After viewing the preview, we then made the changes by clicking ok. All the private data members were extracted and initialized in the new extracted class named PdfDocumentDescriptorData(in the same package). We again successfully build the project. The smell was removed from the JDeodorant.

As the class was very big, therefore it was important to divide the class into smaller classes in order to improve the readability and maintainability of the code. There were 8 private data members which were being used just few times by some methods. Instead, after performing the refactoring, all the private data members were moved to the new class and were made public. We just created a private object of the new class and used specific data members where needed. In this way improved the quality of the software.

There were no further manual changes were required.

3. Manual Refactoring –

3.1. jEdit

3.1.1. org/gjt/sp/jedit/textarea/TextArea.java – private void delete(boolean)

Bad Smell Type – Type Checking

Refactoring steps followed –

The refactoring technique suggested by JDeodorant was replacing condition with polymorphism. The automated refactoring didn't give the correct result and the JDeodorant still detected this smell. So, we had to do it manually.

We removed the if condition inside the for loop.

Added the abstract methods `public abstract int getStartColumn(JEditBuffer buffer);`

`public abstract int getEndColumn(JEditBuffer buffer);` in the Selection class

`@Override` tag was added over these functions in the Selection class for exhibiting polymorphism.

Refactoring Analysis –

Performing this refactoring eliminated the unnecessary check of Selection s being an instance of Selection.Rect

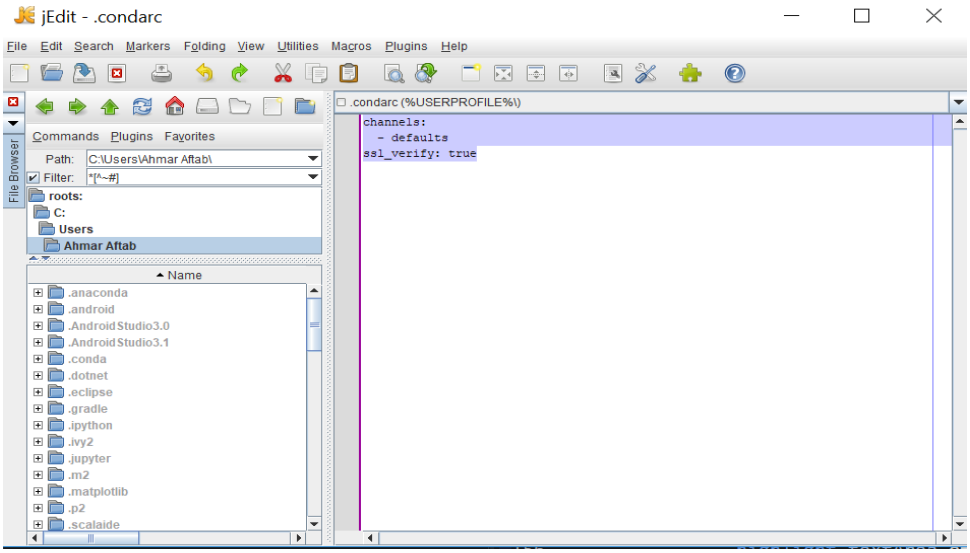

Fig 3.1.1 a. The highlighted text is being selected for deletion operation before and after the change, which triggers the void delete (Boolean) method

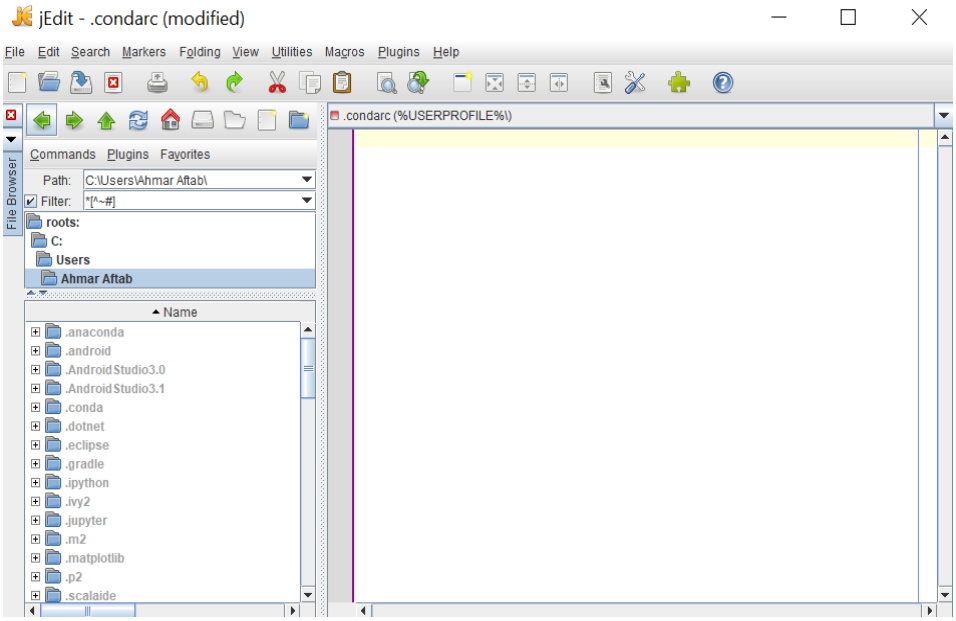

Fig 3.1.1.b. The text is removed on hitting the backspace/delete button on the keyboard. This validates that the refactoring has been successfully implemented.

3.1.2. *org/gjt/sp/jedit/View.java*

Type of Smell – Long Class

The changes done for the change request 2 of Assignment 1 made the size of the class even bigger than before as a new method toggleScrollBar was added. The toggleScrollBar method was moved to the new class ViewUtils so that the View class won't be affected much with our change. The View class is already detected in the bad smell from JDeodrant, and to completely remove it from being detected as bad smell will be challenging as it is one of the biggest classes in the application. So, we thought of just moving the custom method added for the change to a separate class, and this would improve the usability and maintainability of the code apart from adding a method into the already existing smelly class.

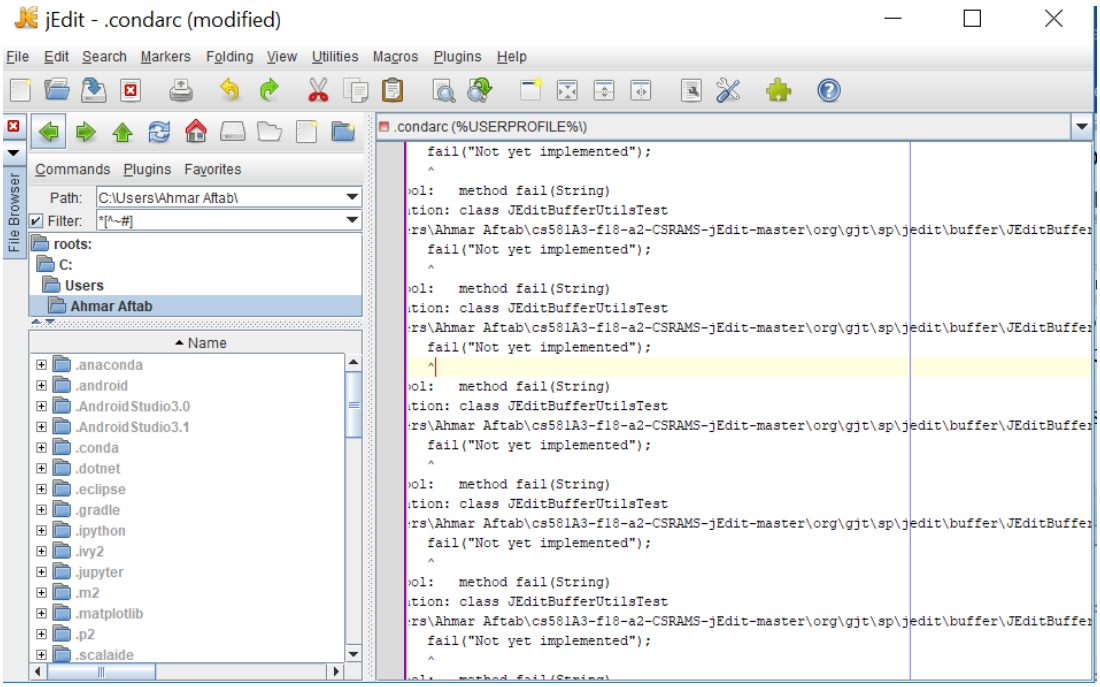

Fig 3.1.2.a The scroll bars get hidden on selecting Toggle Scroll Bar option. This is the result we get before and after the change. Therefore, we can infer that the change has been successfully implemented without affecting the functionalities.

3.2. PDFSam

3.2.1. Pdfsam-alternate-mix/src/main/java/org/pdfsam/alternatemix/AlternateMixModule::public void onLoadWorkspace() –

Bad Smell Type – Long Method, Duplicate Code

Refactoring Analysis –

This smell was detected by the Jdeodorant before doing the previous two refactoring but, was not detected later. So, we choose to identify and refactor the method onLoadWorkspace in the class AlternateMixModule manually. This method was earlier considered as long method because there were more than 10 lines of code in the method. Also, there was code duplication as the same methods were called with just slightly different parameters. So, we replaced the duplicate code with a new method named onLoadWorkspaceProxy and passed two parameters “data” and “proxy”. The second parameter was defined by me to determine the internal values of the proxy method. So, extracted the duplicate code to the new method, declared and initialized some new variables and performed the same functionality in an OO way. This improved the readability and maintainability of the code and removed the duplicate code.

3.2.2. Pdfsam-core/src/main/java/org/pdfsam/support/params/ConversionUtils::private static set<> mergeIntervals() –

Bad Smell Type – Long Class

Refactoring Analysis –

This smell was not detected by JDeodorant. The class ConversionUtils had 4 methods in it originally. But, when we performed the change request in Assignment 2, we added a new feature by adding a new method in the same class. So, we consider it as long class and we extracted this

method to a new class as part of the refactoring process. We moved the method mergeIntervals to the new class MyMergeIntervals and thus created a new class for the new feature that we added. Thus, if we want to make any modifications we only must change the MyMergeIntervals class. If we want to add new features, we can add new class accordingly.

3. Conclusion –

From this assignment we get know various methods of refactoring techniques. Both the automated and the manual refactoring can be done on any smells detected by the plugin. The manual refactoring will require more effort and time in analyzing the code to refactor. However, the automated refactoring enables us to get our work quickly. The automated refactoring is not always perfect, and we must do some changes manually on top of automatic refactored code. But we can expect a much better result in automated refactoring because if the application is huge, then there is possibility of manual errors. All the smells detected by plugins are not actually a smell as there are somethings which are a part of the design or implementation, defined by the developers. We would say automated and manual refactoring can complement each other to get the best possible results.