Suraj Eswaran
CS515 SOFTWARE MAINTAINANCE AND EVOLUTION
ASSIGNMENT 4

Code Smells & Refactoring-PdfSam

Refactoring is the process of changing a software code in such a way that it does not affect the behavior of the code and improves its internal structure. A good design always comes first followed by the coding. Over the time code will be modified and the integrity of the system structure according to that design would gradually fades. Thus, the code sinks from engineering to hacking.

While refactoring the code , we can take a bad design and rework it into a well-designed code. The resulting interaction leads to a program with a design that stays good as development continues.

There are several tools to automatically detect code smells. The tool that I have used for this assignment is JDeodorant. JDeodorant is an Eclipse plug-in that identifies design problems in software, known as bad smells, and resolves them by applying appropriate refactoring.

JDeodorant employs a variety of novel methods and techniques in order to identify code smells and suggest the appropriate refraction that resolve them. For the moment, the tool identifies five kinds of bad smells, namely Feature Envy, Type Checking, Long Method, God Class and Duplicated Code.

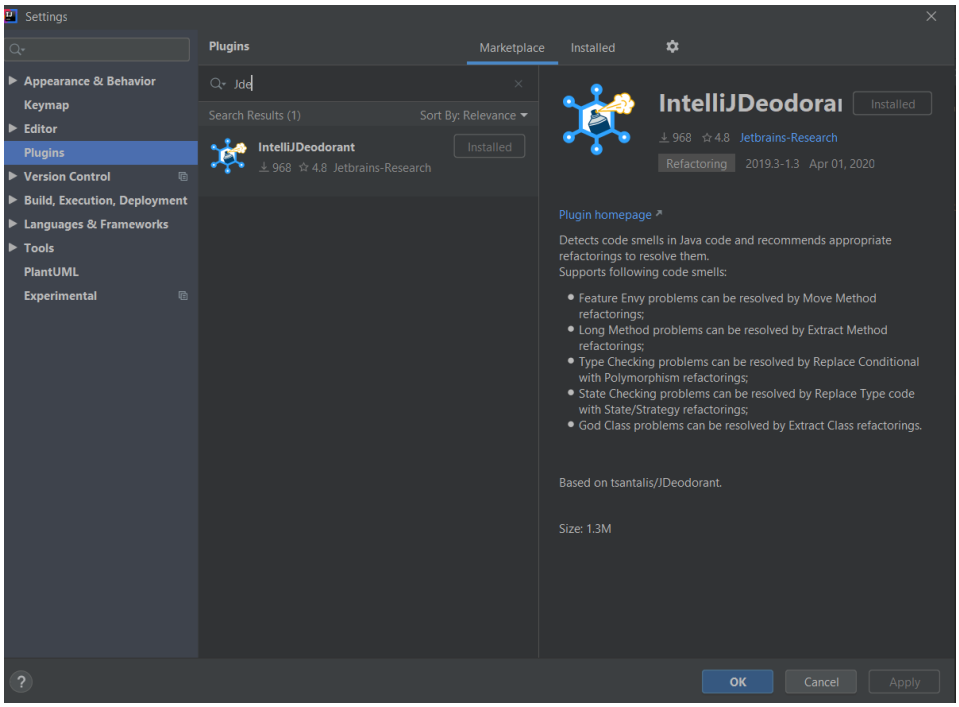

Figure 1: JDeodorant Plugin in Intelli

Suraj Eswaran
CS515 SOFTWARE MAINTAINANCE AND EVOLUTION
ASSIGNMENT 4

Intellij JDeodorant[1] is a plugin which is based on Jdeodorant Eclipse plugin that detects code smells in Java code and recommends appropriate refactorings to resolve them. All of the suggested refactorings can be carried out automatically from within the plugin.

This tool supports several code smells. They are:

1. *Feature Envy: It performs a "Move Method"* when a method uses attributes/methods of another class more than those of the enclosing class. The tool can detect such methods and suggest moving them to a more related class.
2. *Type Checking follows a "Replace Conditional with Polymorphism" refactoring where it refers to cases when a set of conditional statements determine the outcome of the program by comparing the value of a variable representing the current state of an object with a set of named constants.*
3. *Long Method occurs when a method is too long and can be divided into several. For such methods, the tool identifies blocks of code that are responsible for calculating a variable and suggests extracting it into a separate method, i.e. perform an Extract Method refactoring.*
4. *God Class is done on large and complex class that contains too many components. The tool identifies sets of attributes and methods in a class that could be moved into a separate class to simplify the understanding of the code, i.e. an Extract Class refactoring can be performed.*

To run the tool:

- *After the plugin is installed , IntelliJDeodorant tool will appear below in IntelliJ IDEA.*

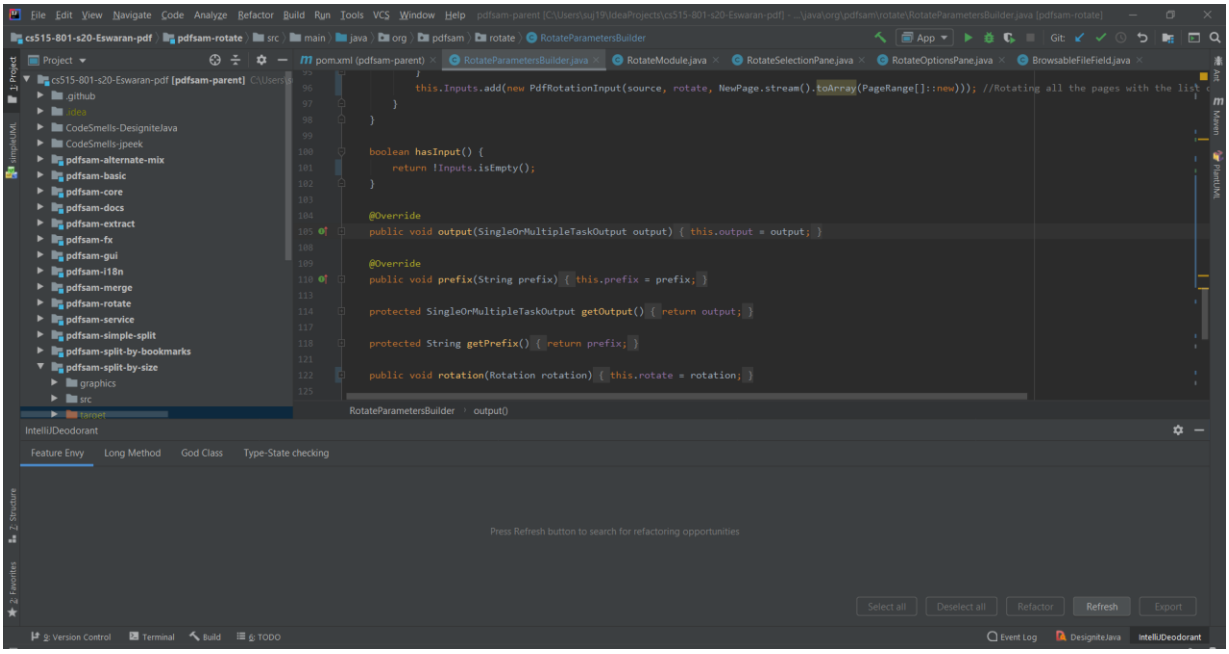

Figure 2: Preview of IntelliJ JDeodorant

- *Press Refresh button to search for refactoring opportunities.*
- *Each tab of this window contains a Refresh button that allows to search for the necessary code smell in the entire project and the table with the results of the search.*
- *For refactoring, select a suggestion in the table and click the Refactor button which is beside Refresh button.*

I. AUTOMATED REFACTORING

a. *Class:*
pdfsam-rotate/src/main/java/org/pdfsam/rotate/RotateParametersBuilder.java
Smell Type: God Class, Method: Extract Class
Rationale: The main for this code smell refactoring is to extract classes into other class.

Before Refactoring

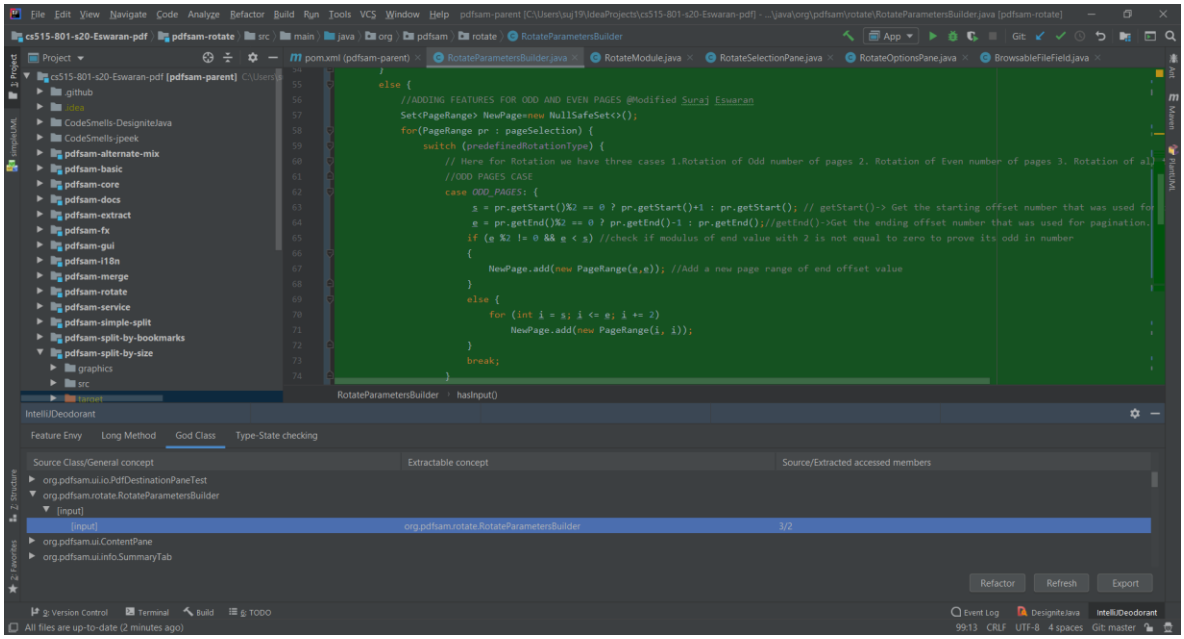

Figure 3: Detection of Code smell in RotateParametricBuilder.java

After Refactoring

For removing the smell from God class RotateParametricBuilder, A class RotateParametricBuilderProduct was created and fields are extracted from RotateParametricBuilder.

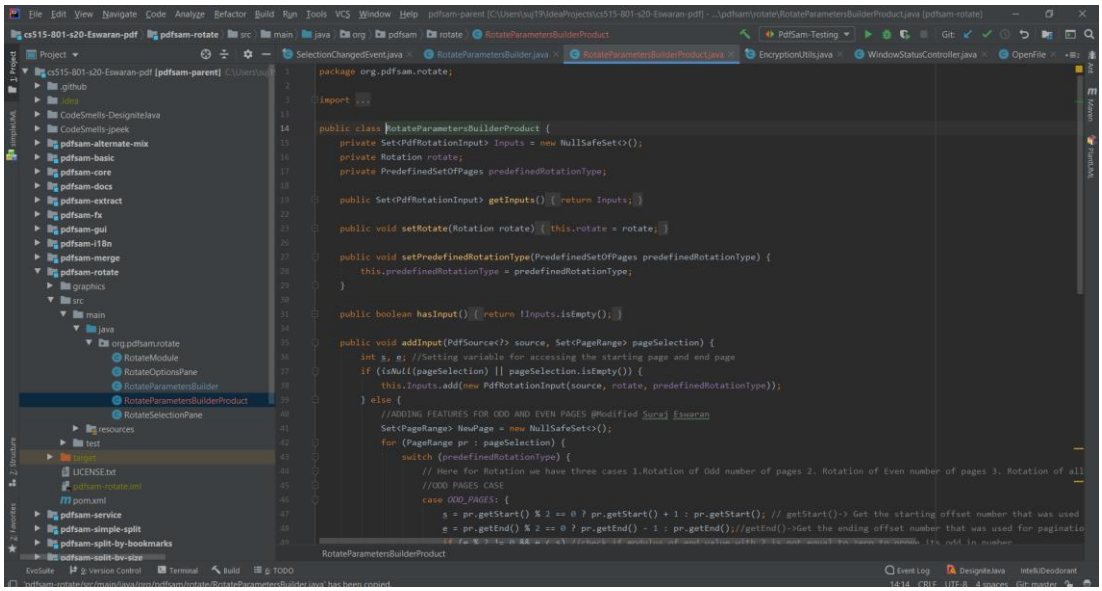

Figure 4: Extraction of fields in RotateParametricBuilderProduct.java

After refactoring is done, the smell is not detected, thus the smell is removed with the help of JDeodorant.

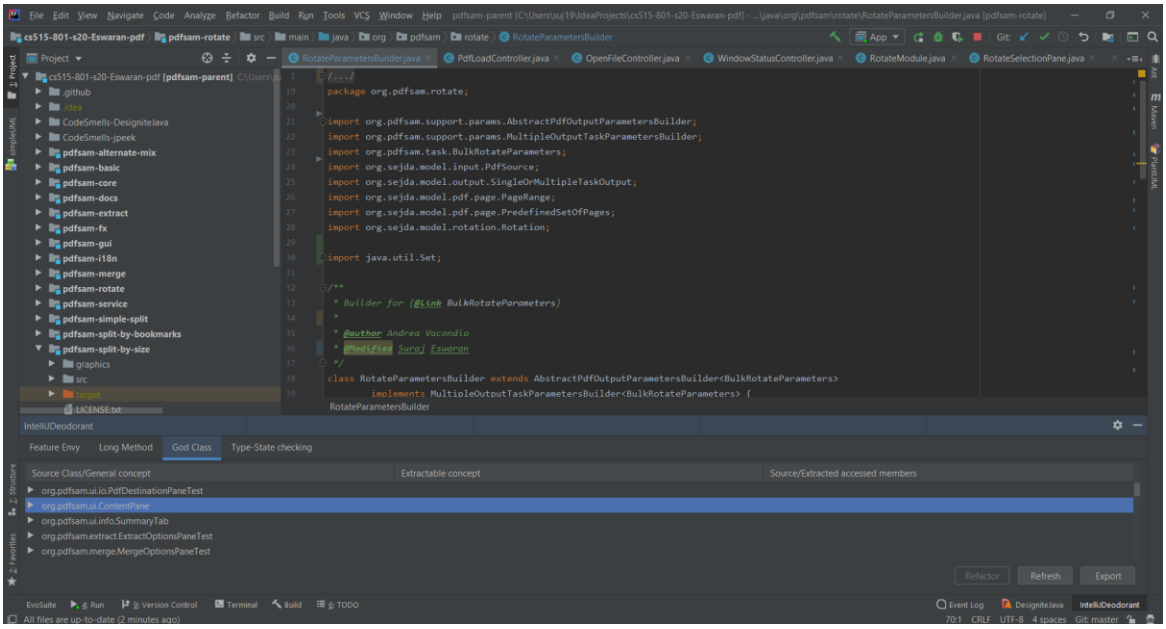

Figure 5: Detection of no smell from RotateParametricBuilder.java

There were not any changes in the code, just extraction was done. Testing is done with the help of Junit testing. All the test were passed before and after refactoring.

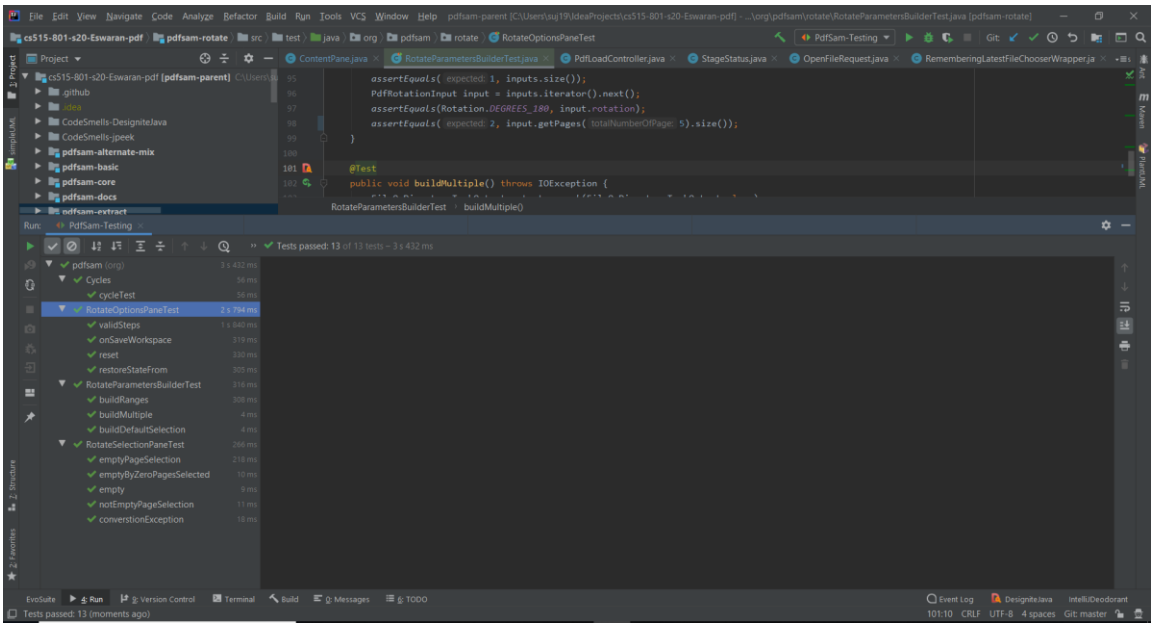

Figure 6: Junit Testing done on RotateParametricBuilderTest

b. *Class:*
pdfsam-core/src/main/java/org/pdfsam/module/ModuleDescriptorBuilder.java
Smell Type: God Class, Method: Extract Class
Rationale: The main for this code smell refactoring is to extract classes into other class.

<u>Before Refactoring</u>

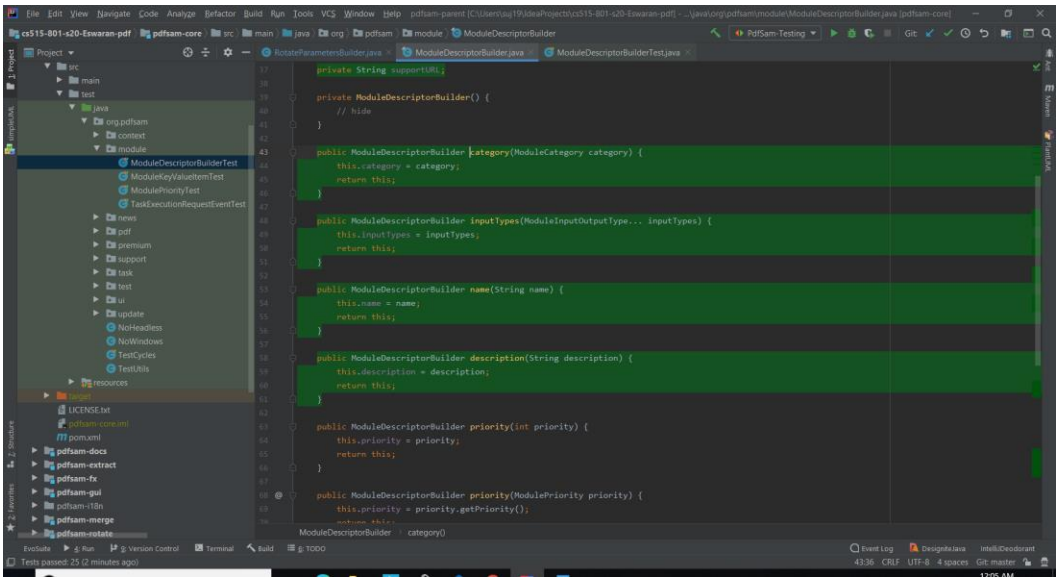

Figure 7: Detection of Code smell in ModuleDescriptionBuilder.java

<u>After Refactoring</u>

For removing the smell from God class ModuleDescriptorBuilder, A class ModuleDescriptorBuilderProduct was created and fields are extracted from ModuleDescriptorBuilder.

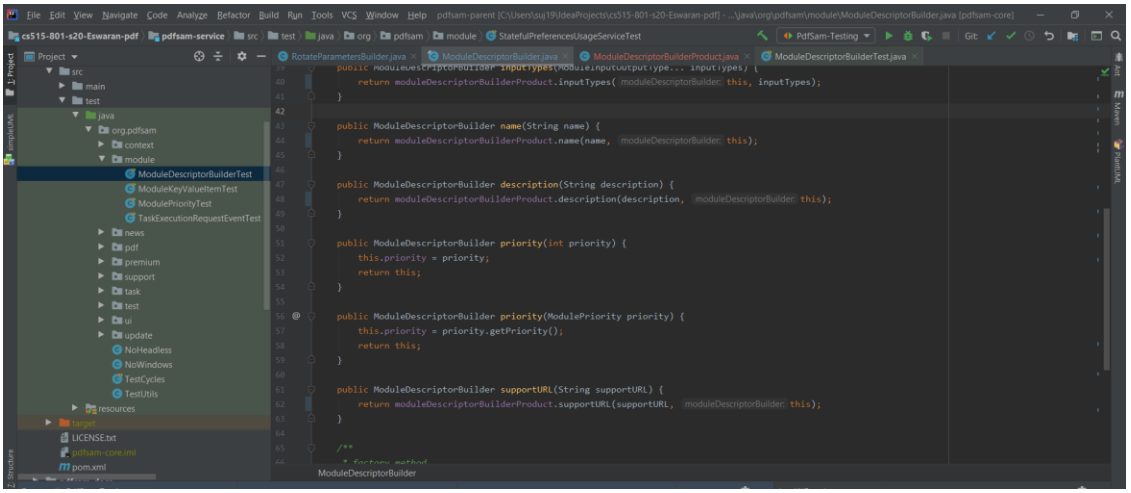

Figure 8: Extraction of fields in ModuleDescriptionBuilderProduct.java

There were not any changes in the code, just extraction was done. Testing is done with the help of Junit testing. All the test were passed before and after refactoring.

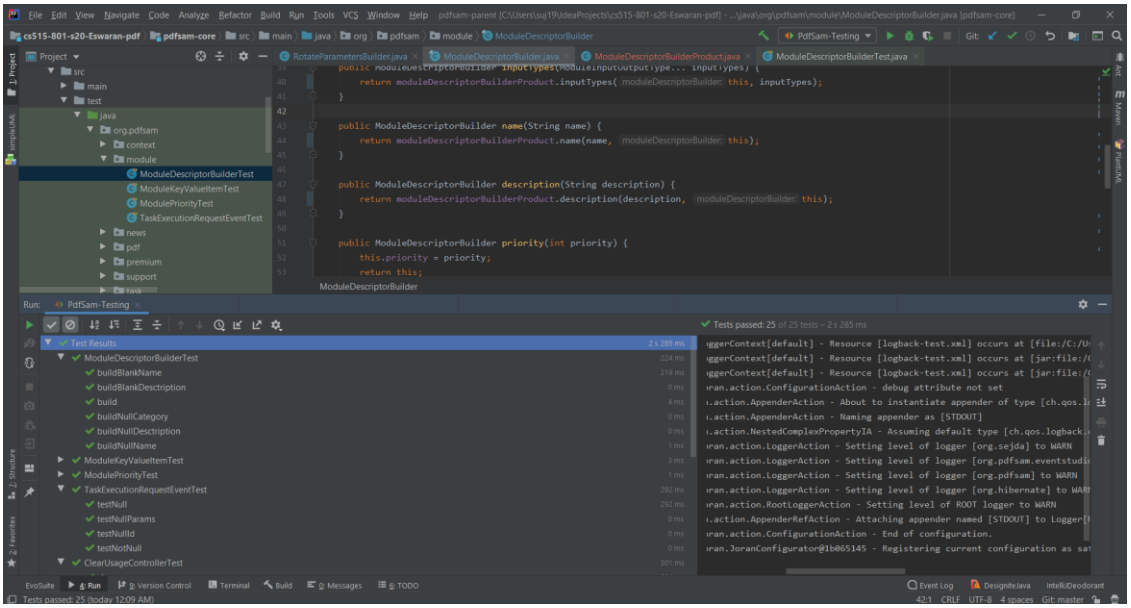

Figure 9: Junit Testing on ModuleDescriptionBuilderTest

c. *Class:pdfsam-fx/src/main/java/org/pdfsam/ui/io/BrowsableFileField.java*
Smell Type: Long Method, Method: Extract Method
Rationale: The main for this code smell refactoring is to extract method into other class.

Before Refactoring

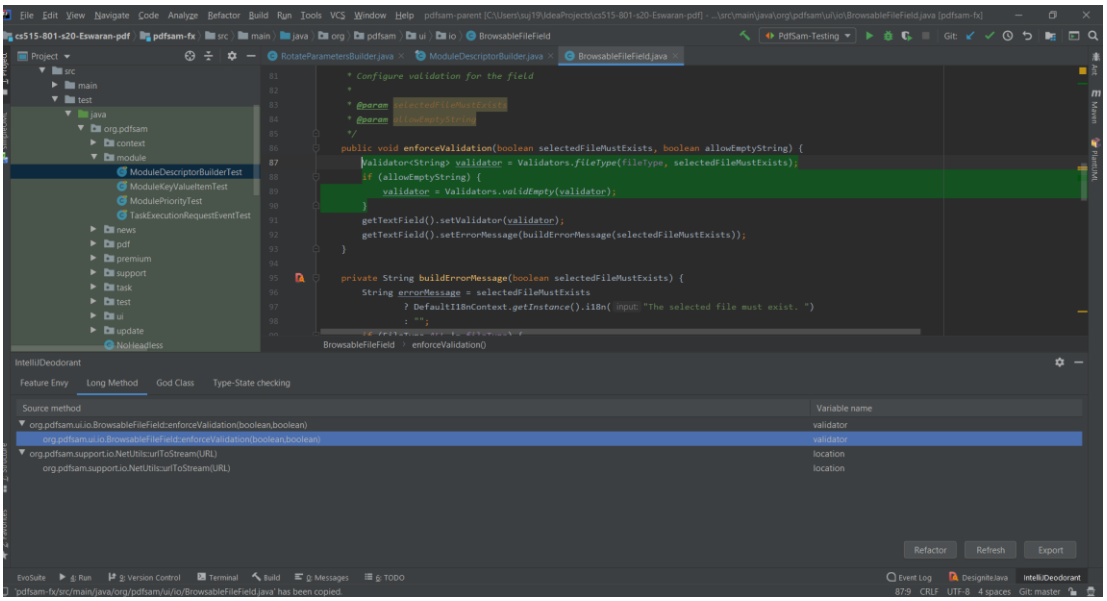

Figure 10: Detection of Code smell in BrowsableFileField.java

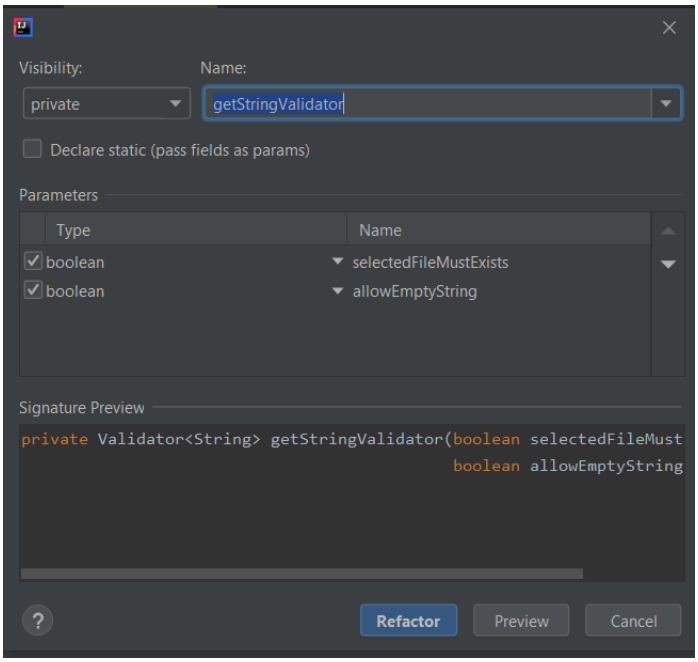

Figure 11:Extract Method in BrowsableFileField.java

After Refactoring

For this, a code smell of Long Method was removed using extract method by IntelliJ JDeodorant. Detected lines were moved to a method getStringValidation.

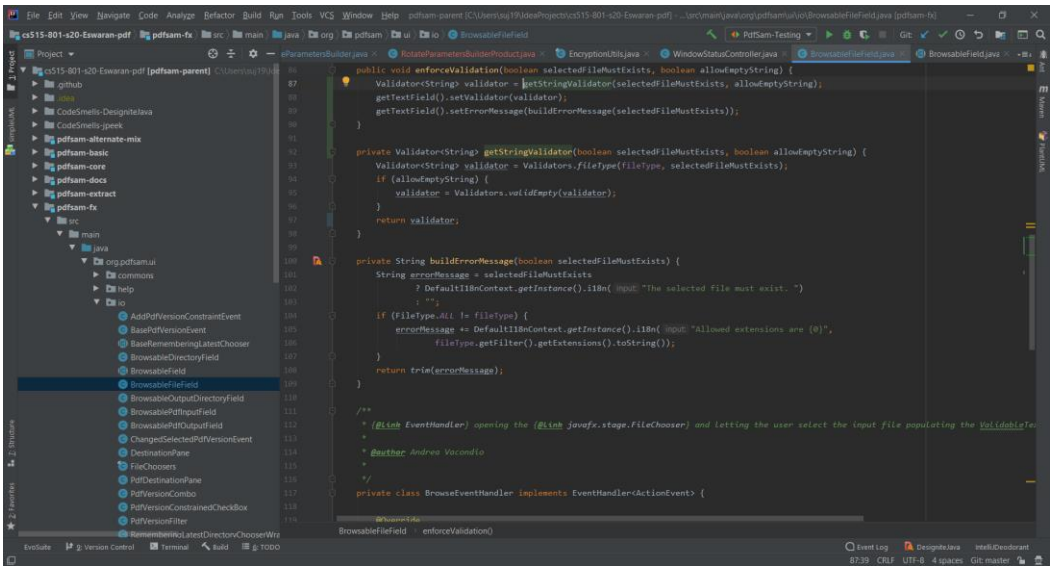

Figure 12: Removal of the Code Smell

There is not any code change , only extraction of method is done. Testing is done with the help of Junit testing. All the test were passed before and after refactoring.

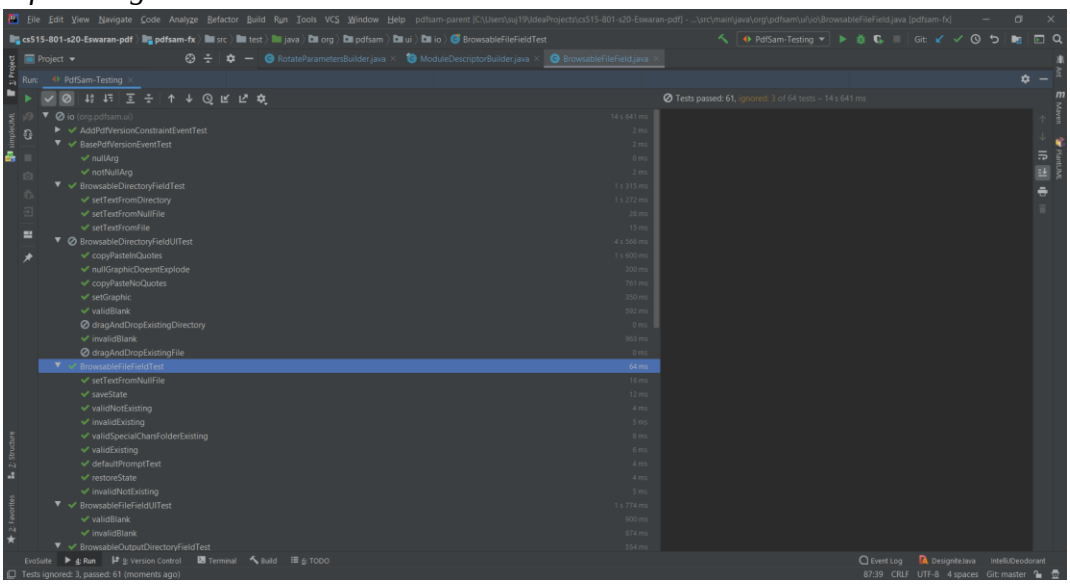

Figure 13: Junit Testing for BrowsableFileFieldTest

From the observation, BrowsableFileField were used only once in the project. For these kinds of code, extraction made the process easier since it would make the function call too long. That is why it better to rely on extracting the method.

d. *Class:*
pdfsam-fx/src/main/java/org/pdfsam/ui/io/RememberingLatestFileChooserWrapper.java
Smell Type: Type/State Checking, Method: Replace Conditional with Polymorphism
Rationale: The main for this code smell refactoring is to replace conditional methods with the help of polymorphism.

Before Refactoring

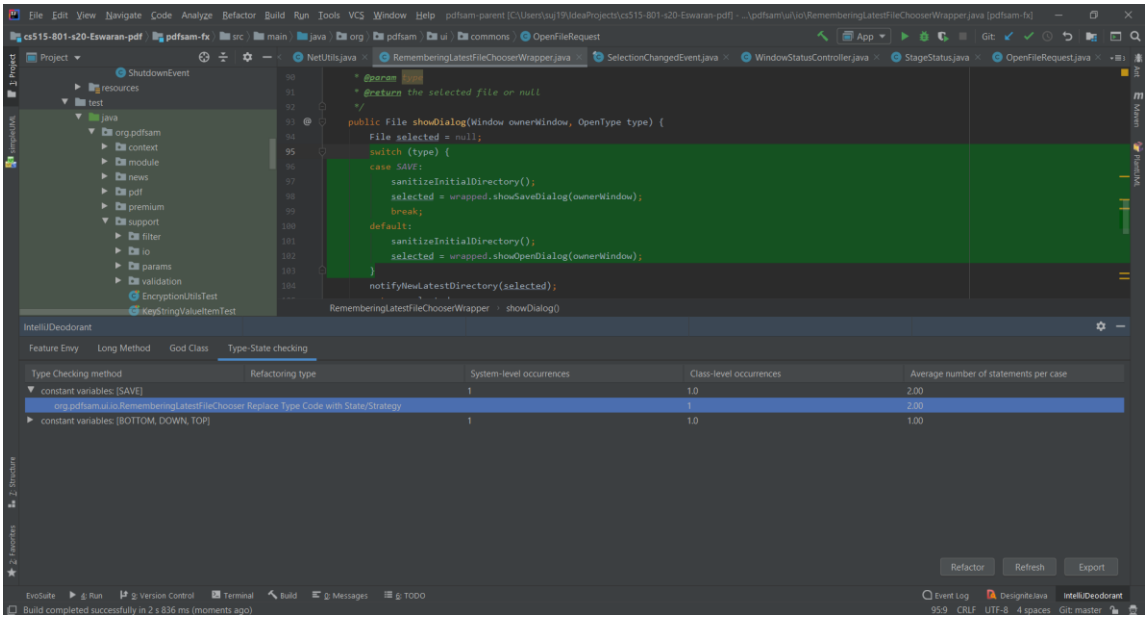

Figure 14: Detection of Code Smell in RememberingLatestFileChooserWrapper.java

After Refactoring

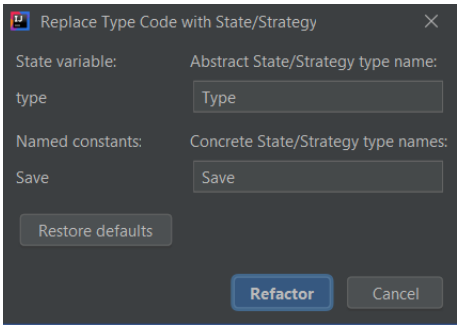

Figure 15: Replace Conditional with Polymorphism-I

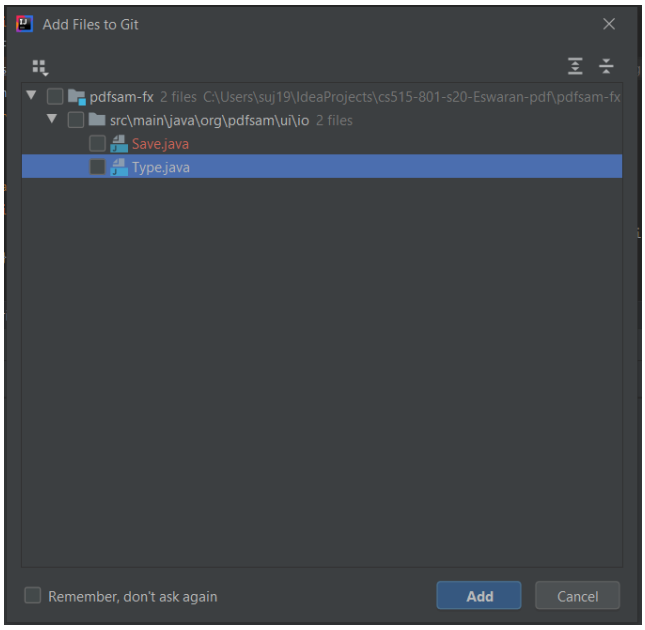

Figure 16: Replace Conditional with Polymorphism-II

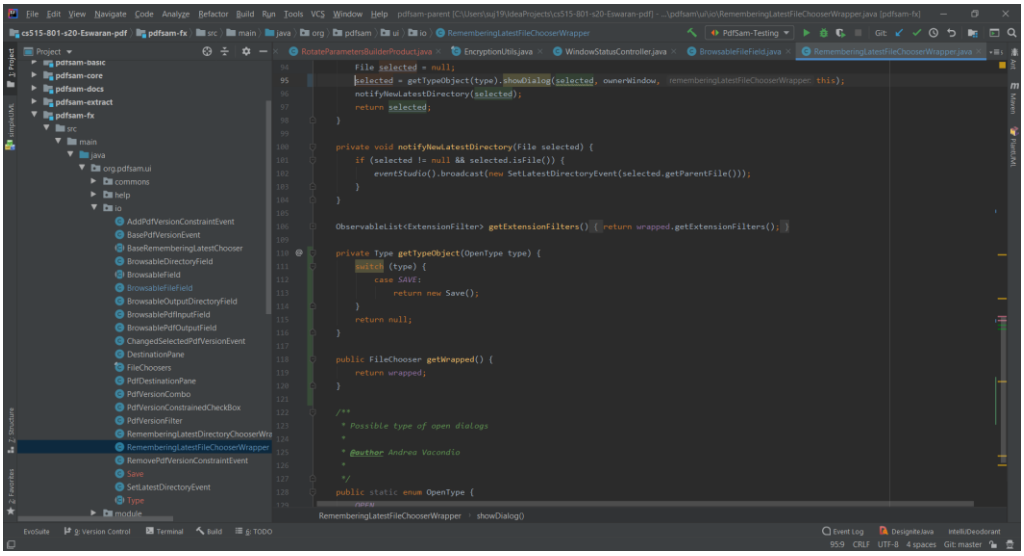

Figure 17: Removal of the Code Smell

There are not any changes in the code, only creation of several classes were implemented using polymorphism. Testing is done with the help of Junit testing. All the test were passed before and after refactoring.

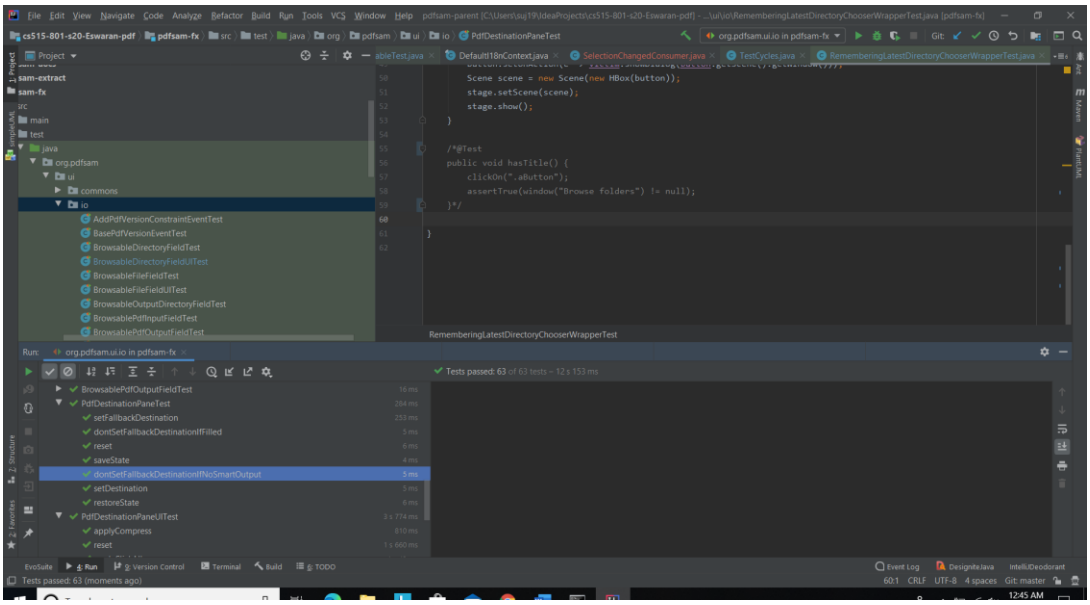

Figure 18: Junit Testing for RememberingLatestFileChooserWrapperTest

II. MANUAL REFACTORING

a. *Class:*
pdfsam-fx/src/main/java/org/pdfsam/ui/selection/multiple/SelectionTable.java
Smell Type: God Class ,Method: Extract Class
Rationale: The main for this code smell refactoring is to extract classes into other class.

Before Refactoring

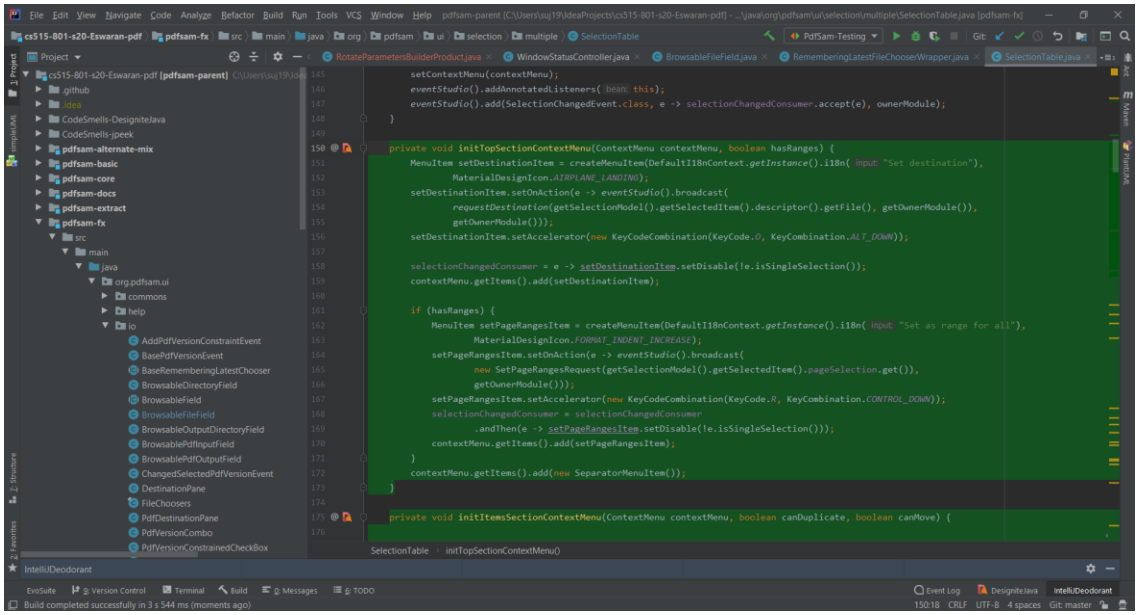

Figure 18: Detection of Code Smell in SelectionTable.java

After Refactoring

After refactoring is done, the smell is not detected, thus the smell is removed with the help of JDeodorant.

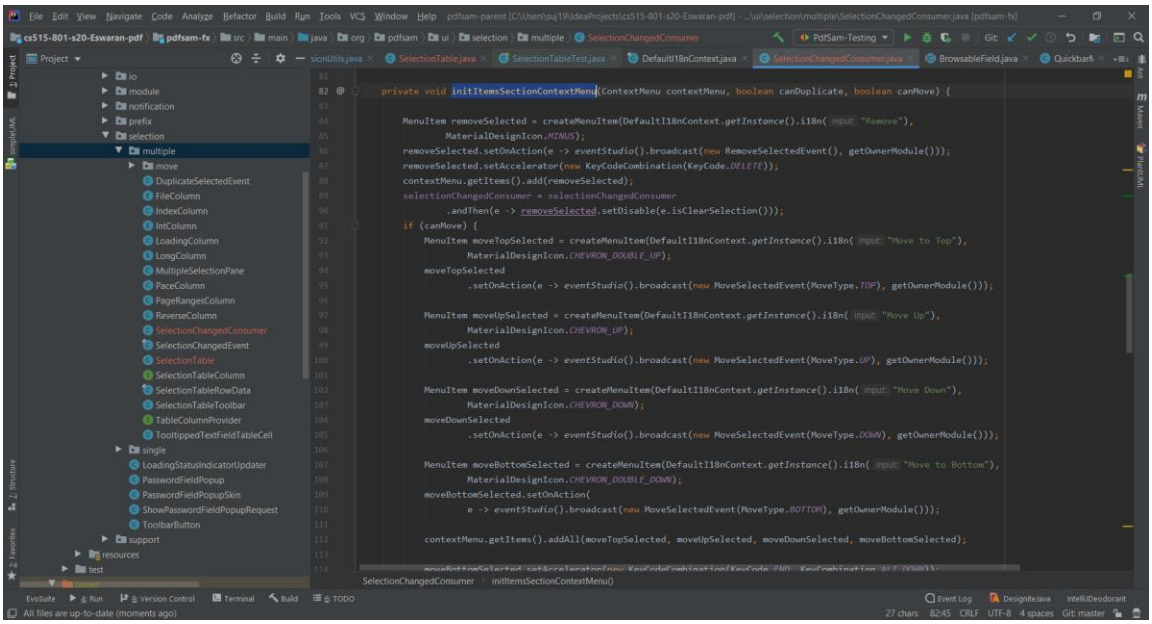

Figure 19: Creation of new Class SelectionChangedCharacter.java

In this case , a new class called SelectionChangedCharacter was created and extracted few of the methods from SelectionTable has been used in it because of the code refactored by the Jdeodorant was not working according to the functionality so have to do it manually in order to have free flow of the software. Testing is done with the help of Junit testing. All the test were passed before and after refactoring.

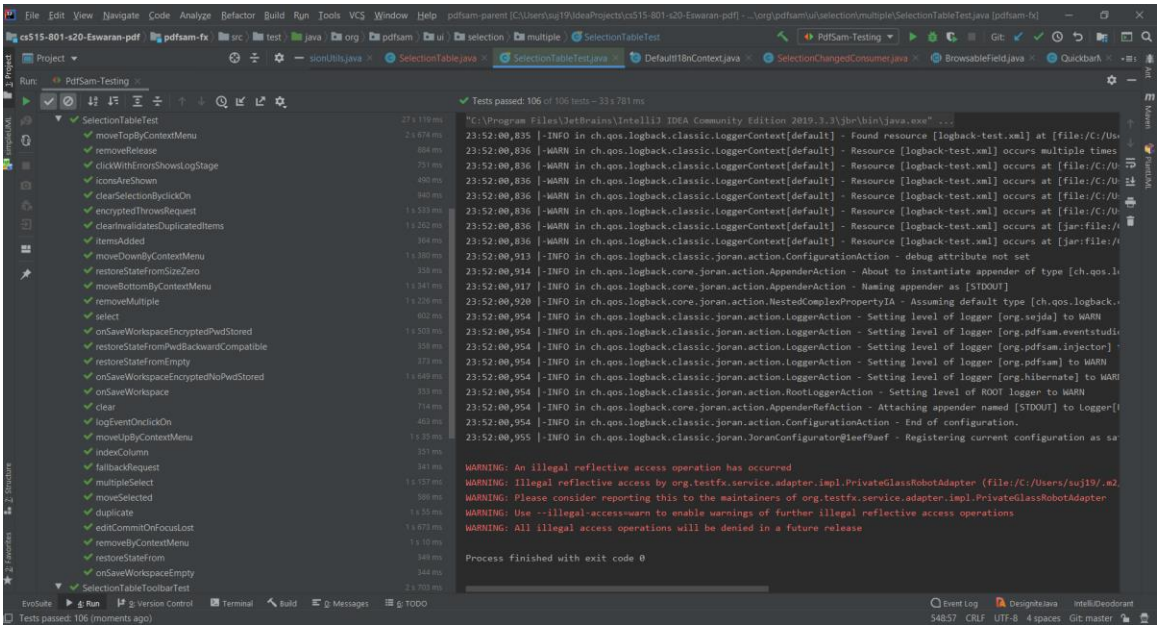

Figure 20:Running of Junit for SelectionChangedCharacterTest

Type of refactoring	*Smelly Classes*	Briefly describe the smell by considering the class, methods, attributes, etc. involved in the smell	Explain why the class/method is flagged as smelly (be specific).	Do you agree that the detected smell is an actual smell? Justify your answer.
Automatic	*RotateParametersBuilder*	*Functionalities of class RotateParametricBuilder seems to be doing all the function operation, thus it can be separated into other different classes based on similar type of responsibilities*	*Since bulkrotate() seem to do all the functionality of RotateParametricBuilder, so this method would be considered as a code smell. Solution for this God class problem is to extract the class to a new default class named RotateParametricBuilderProduct after refactoring.*	*To me, this detected smell can be an actual smell since it is a difficult task for the , programmers to create a new class for the feature. It can be solved by placing a new feature in an existing*

				class which can be helpful in design aspect as well. The changed code did not affect the function ality.
	ModuleDescriptorBuil der	*Functionaliti es of class ModuleDescr iption seems to be doing all the function operation, thus it can be separated into other different classes based on similar type of responsibiliti es.*	*In the case of ModuleDescriptio nBuilder, Status seem to be used a lot, so this method would be considered as a code smell. Solution for this God class problem is to extract the class to a new default class named ModuleDescriptio nBuilderProduct after refactoring.*	*Even this code smell can be an actual smell since it is a difficult task for the , programm ers to create a new class for the feature rather than it can be solved by*

				placing a new feature in an existing class which can be helpful in design aspect as well. The changed code did not affect the function ality.
	BrowsableFileField	*As the method contains too many lines of code, thus it would be hard to add on new features on the existing model of program. In order To reduce the length of a method body, use extract method.*	*It is considered as a flagged smell because too much of lines of code would make it worse, even though it is easier to write code, the smell remains unnoticed until it becomes a disaster.*	*This detected smell should be consider ed as an actual smell because in the program mer's aspect it is harder to create a new method than to*

				add to an existing one when a situation of large code lines arrives.
	RememberingLatestFileChooserWrapper	*The code seems to be smelly since the filed in the class would not be belonging to the class , would be remaining as an ordinary field in it. This causes type checking kind of code smell.*	*It is flagged as a smell because the fields Save, and Type are been considered to be not related to FileChosser, but they might provide its own polymorphic behavior as well.*	*According to me, this can also be considered as an actual smell since it list of parameters might happen after several types of algorithms issue which is not a good sign in design aspect. In order to avoid that replace*

				ment with polymorphism can be implemented.
Manual	*SelectionTable*	*This kind of code smell seems to have additional functionalty of initTopSelectionContext Menu where seems to provide improper design aspect, thus it can be separated into other different classes based on similar type of responsibilities.*	*Since bulkrotate() seem to do all the functionality of RotateParametric Builder, so this method would be considered as a code smell. Solution for this God class problem is to extract the class to a manually added class named SelectionChanged Customer after refactoring.*	*Even though it was a manual refactoring, I think this detected smell is an actual smell since it creates a smelly code which would spoil the Design of Existing Code .As it is necessary to look at design aspect, so it would be a good*

				solution of extracting classes. The changed code did not affect the functionality.