**Assignment 4: Code Smells & Refactoring**

1. **Introduction and Configuration**

In this assignment, two projects jEdit and PDFsam are used for code smells analysis. For each project, two or more classes/methods are selected for code smells detection.

Most code smells in this report are detected by Designite tool. To use the tool on Windows, please download its executable, open the Command Prompt, cd into the directory and enter the following command:

java -jar {your\_dir}\DesigniteJava.jar -i {your\_workspace} -o {your\_output\_dir}

The jEdit workspace is branched from the original version.

The PDFsam workspace is branched from the original 4.0.5 version.

1. **Detecting and Analyzing Code Smells**
2. **jEdit**

By using Designate tool to check for implementation code smells, we collected many types of code smells within the StatusBar and ColumnBlock classes. As required, two or more methods of these two classes were selected for analysis.

Table 1 shows more details and descriptions about the code smells.

Table 1. Some detected code smells in jEdit project

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class | Method & Smell location | Code Smell Type | Explanation | Agree? |
| StatusBar | updateCaretStatus()  Location: StatusBar.java from line 339 to 425 | Complex Method | There are too many conditional IF statements within this method that increases the complexity. The complexity is determined by the number of possible branches can be executed. High complexity makes developers more difficult to understand the behavior of the code. | Yes. Fixing this smell will make the code cleaner and easier to understand. |
| StatusBar | updateCaretStatus()  Location: StatusBar.java from line 400 | Long Statement | Line 400 in StatusBar.java contains two condition statements that take up two full lines of code. Each statement calls a public static method that returns a Boolean. Long statement causes visual difficulty for developers in reading code. | Yes. Fixing this smell will make the code cleaner and easier to understand. |
| StatusBar | setMessageAndClear()  Location: StatusBar.java at line 282 and 292. | Magic Number | Line 282 and 292 have two “magic” numbers being used. Magic numbers are the numbers without obvious meaning. Magic number is a real issue if there are two or more literal numbers of the same meaning are used in different places, because if there is an update, they must be changed all together. | I partially agree with this smell. Each number is only used once. The meaning of each number is understandable from an unexperienced developer’s view. A quick fix like adding a comment or creating a local constant for these numbers does not clear the smell. The smell can only be clear when I create class member constants for these numbers. However, I believe creating extra class member variables to clarify these magic number is unnecessary. |
| ColumnBlock | setTabSizeDirtyStatus()  Location: ColumnBlock.java line 405 | Complex Conditional | There are three conditions within an IF statement. The “recursive” condition does not relate to the other two conditions which check for empty or null nodes. Too many conditions within a single statement can cause confusion for inexperienced developers. | Yes. Fixing this smell will make the code cleaner and easier to understand. |

1. **PDFsam**

Similarly, CloneDR, inCode and Designite tools are used to detect code smell in PDFsam project. Table 2 shows some code smells detected by the tools and provides more information about the smells.

Table 2. Some detected code smells in PDFsam project

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class | Method & Smell Location | Code Smell Type | Explanation | Agree? |
| DefaultUserContext | Many methods within this class that call this method getBoolean() | CloneDR tool identifies the repeated getBoolean() calls as Code Clones. | The same method is called in many places. This smell is usually caused by copy/paste coding practice. | No. Each method that calls this repeated code contains nothing else except this code. It serves the purpose to clarify the usage of the code. The code seems to be factored already, so it is not necessary for refactoring. |
| ModuleUsage | getModuleId() and setTotalUsed()  Location: ModuleUsage.java line 34 and line 54 | inCode tool detects Data Class code smell within this ModuleUsage class. | These two methods are defined as public methods but are not called anywhere. By the definition of the inCode tool for Data Class code smell, the class is exposing a significant amount of data in its public interface, either via public attributes or public accessor methods | No. These methods are implemented to be available for future usage. These get and set methods are typical public methods. Therefore, we cannot either remove these methods or change their types to private. |
| WindowStatusController | a public constant variable named PDFSAM\_DISABLE\_UI\_RESTORE  Location: WindowStatusController.java line 44 | Deficient Encapsulation smell detected by Designite tool | The variable is declared as public type but is not used anywhere else in the source code. This violates the encapsulation principles that it provides more access than required. If the variable is designed to be a global constant, it should be defined in a utility class. | Yes. Fixing this smell does not affect the current code, but will safely prevent unwanted access to the variable in the future. |
| DashboardTile | DashboardTile()  Location: DashboardTile.java line 57. | Magic Number | The value of 5 of the same meaning is repeatedly used throughout the project. However, some places use literal number 5 and some other places use a defined constant value. This causes a confusion that if those values of 5 are related. If they are, they should use the same constant variable. If they are not, they should use variables of different names to avoid the confusion. | Yes. Fixing this smell will make the code cleaner and easier to understand. |

1. **Refactoring to Remove Code Smells**

Only the smells that I agree detected in the previous section are subjected for code refactoring.

JDeodorant does not work correctly on my workspace. Therefore, I performed code refactoring manually for all the smells.

1. **jEdit**

Since jEdit does not have its own unit test, new unit tests are added for each code change.

Table 3 describes in detail the refactoring technique used to remove the code smells and its rationale.

Table 3. Code refactoring in jEdit project

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class & Method | Smell Type | Location | Refactoring method | Refactoring Rationale |
| StatusBar  updateCaretStatus() | Complex Method | Location: StatusBar.java from line 339 to 425 | Split the method updateCaretStatus() into smaller methods. Each method limits the number of conditional statements. | A method that contains many conditional statements can be split into smaller methods to reduce its complexity. The split is necessary especially when the smaller methods are less directly relevant to the current method. |
| Long Statement | Location: StatusBar.java from line 400 | The method getBooleanProperty() contains a long parameter of string type. The method is repeatedly called multiple times. Creating a Boolean value and calling this method once will not only save the unnecessary long statement but also increase the performance. | A temporary variable can be used to reduce the length of a long statement. By naming the variable appropriately, the code will become easier to understand. |
| ColumnBlock  setTabSizeDirtyStatus() | Complex Conditional | Location: ColumnBlock.java line 405 | Split the current IF statements into smaller IF statements. | Too many conditions within one statement can cause confusion. Complex conditional smell can be refactored by splitting one complex condition statement into smaller statements. While splitting, we categorize the relevant ones together in one group and the others into another group. |

1. **PDFsam**

Since the code change in this project does not add new class or method and does not change its original behavior, it is unnecessary to update the existing unit test. Table 4 describes in detail the refactoring technique used to remove the code smells and its rationale.

Table 4. Code refactoring in PDFsam project

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class & Method/Variable | Smell Type | Location | Refactoring method | Refactoring Rationale |
| WindowStatusController  PDFSAM\_DISABLE\_UI\_RESTORE | Deficient Encapsulation | Location: WindowStatusController.java  line 44 | Change access of the variable from public to private. | Variables that are unintended to be used outside of its class should be defined as private.  Global constant variables should be defined in a utility class which is intended to be used globally. |
| DashboardTile  DashboardTile() | Magic Number | Location: DashboardTile.java  line 57. | The value of the number used in this method is already defined in a utility class. Both have the same meaning. Therefore, we should use that global constant. | Global constant does not only provide the meaning of some values but also show the relationship between those distinct values.  Global constants are usually defined in a utility class.  If a number has the same meaning and the same value with a global constant, but it is meant to be unrelated, that number should be assigned to a variable of different name to avoid confusion. |

1. **Conclusion**

The refactoring successfully removed the detected code smells specified in section B without changing the original behavior. After this exercise, I see that the PDFsam code is cleaner than the jEdit as it has less code smells than the other one.