Alex Adams

SER316

Assignment 7

**Task 1:**

**Size**

1. What is the Total Lines of Code (LOC) in the project?

In the whole project: 22539

In the main package: 2187

2. What is the largest single code file in the project and its Total LOC?

main.java.memoranda.ui.htmleditor.HTMLEditor.java is the largest code file with 2144 lines.

In the main package the larges code file is EventsManager.java with 329 LOC.

3. Inspect CurrentNote.java - what method did the Metrics tool use to determine Total LOC? Describe the method.

The metrics tool ignored all lines containing only whitespace and all lines containing only comments.

**Cohesion**

1. The tool calculates “Lack of Cohesion of Methods” (LCOM) using the HendersonSellers method, or what is commonly referred to as LCOM2 (there are LCOM1 through LCOM4 and different methods to calculate them). What is the definition of LCOM2 and how is it calculated? (there are different methods used to calculate LCOM2).

The definition of LCOM2 used by the application is as follows (taken from the documentation for the plugin): "If (m(A) is the number of methods accessing an attribute A, calculate the average of m(A) for all attributes, subtract the number of methods m and divide the result by (1-m). "

2. Which class has the highest Cohersion and do you have an idea why?

The following classes all have 100% cohesion (0 lack of cohesion), because either they have no local attributes and therefore no attributes with which to create lack of cohesion, or every method uses every attribute.

In the main package:

EventsScheduler, Note, EventNotificationListener, ProjectImpl, HistoryListener, Project, ProjectListener, DefaultEventNotifier, CurrentNote, ProjectManager, CurrentProject, EventsManager, History, Start, EventImpl, NoteListener, Task,ResourcesList, Event, NoteList, TaskList

**Complexity**

Make sure you do this after 1-3 so as to not distort those answers!!!

1. What is the cyclomatic complexity in the main package?

Average cyclomatic complexity in the main package is 1.746.

2. What class has, on average, the worst McCabe Cyclomatic Complexity (CC) and what is it?

In the main package the class with the worst CC is EventsManager at 2.5.

3. Go back to your code and reduce the Cyclomatic Complexity. You can choose any class but the Cyclomatic Complexity needs to be reduced at least by a small amount somewhere. Explain what you changed and why, and why it reduced the complexity and how much you were able to reduce the complexity.

In EventsManager.getRepeatableEventsForDate() (the method with the highest complexity) there were some conditional branches laid out like this:

if (condition1) {

//code

}

else if(condition 2){

if(condition 3){

//functionCall

}

}

else if(condition 4){

if(condition 5){

//functionCall

}

}

This code can be refactored as:

if (condition1) {

//code

}

else if( (condition 2 && condition 3) || (condition 4 && condition 5) ) {

//functionCall

}

Which is the change I made to reduce cyclomatic complexity. However, the change did not actually reduce the cyclomatic complexity, which doesn't make sense to me. It should have reduced it by a significant margin, so it must be using a calculation which would not be affected by the change I made.

**Package-level Coupling**

1. What do Afferent and Efferent coupling mean? Look these terms up on Wikipedia and summarize the distinction.

Afferent Coupling is the amount of other classes / packages that depend on this class / package.

Efferent Coupling is the amount of other classes / packages that this class / package depends on.

One is inward, the other is outward.

2. What package has the worse Afferent Coupling measure and what is the value?

main.java.memoranda.util has the worst Afferent Coupling with a value of 57.

3. What package has the worse Efferent Coupling measure and what is the value?

main.java.memoranda.ui has the worst Efferent Coupling with a value of 49.

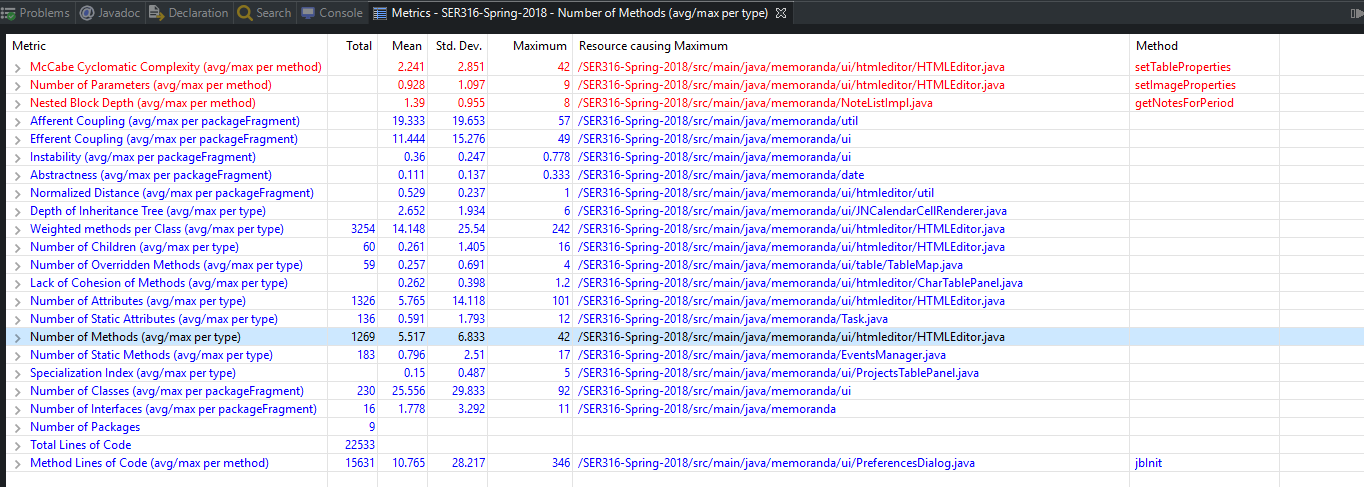
**Worst quality**

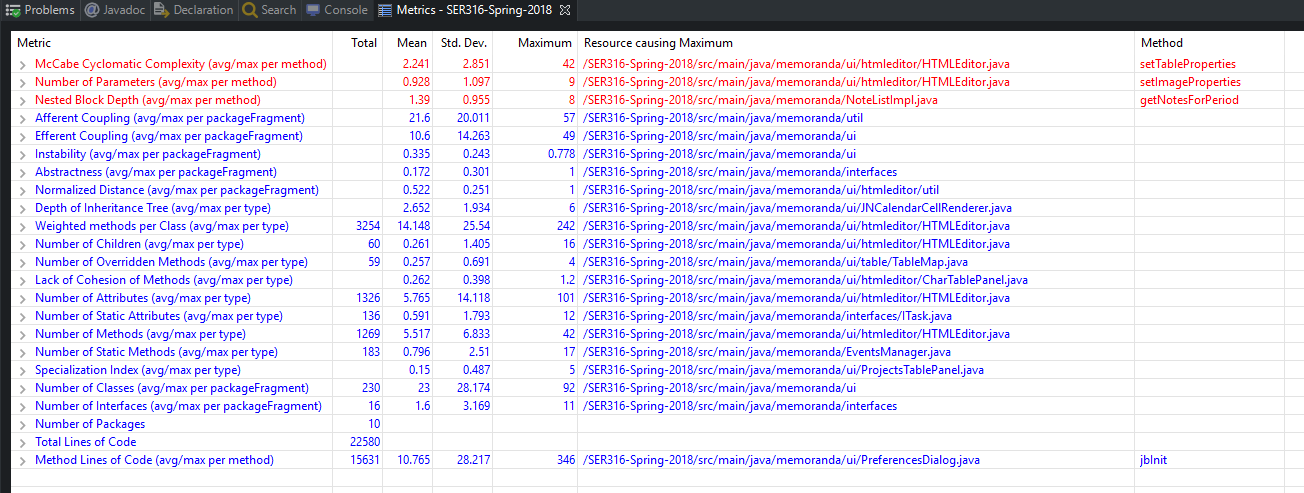
Which class has the worst quality and why? Use the metrics to support your answer. You can use any metric or differnt metrics together but you have to give valid arguments.

main.java.memoranda.ui.htmleditor.HTMLEditor.java definitely has the worst quality. Out of all the metrics measured by the plugin, that class has the highest value of 6 of all the metrics, more than any other class. It has the worst cyclomatic complexity, highest number of parameters, highest weighted methods per class, most number of children, highest number of attributes, and highest number of methods.

**Task 2:**

**Before:**

**After:**

All of the metric totals stayed the same, as expected. But because we added a new package, many of the means changed. Particularly the "Number of Classes" and "Number of Interfaces" both changed for the better, because now some of our interfaces are in their own package, and the actual implementations are in their own package as well. So because of the extra package the averages went down.

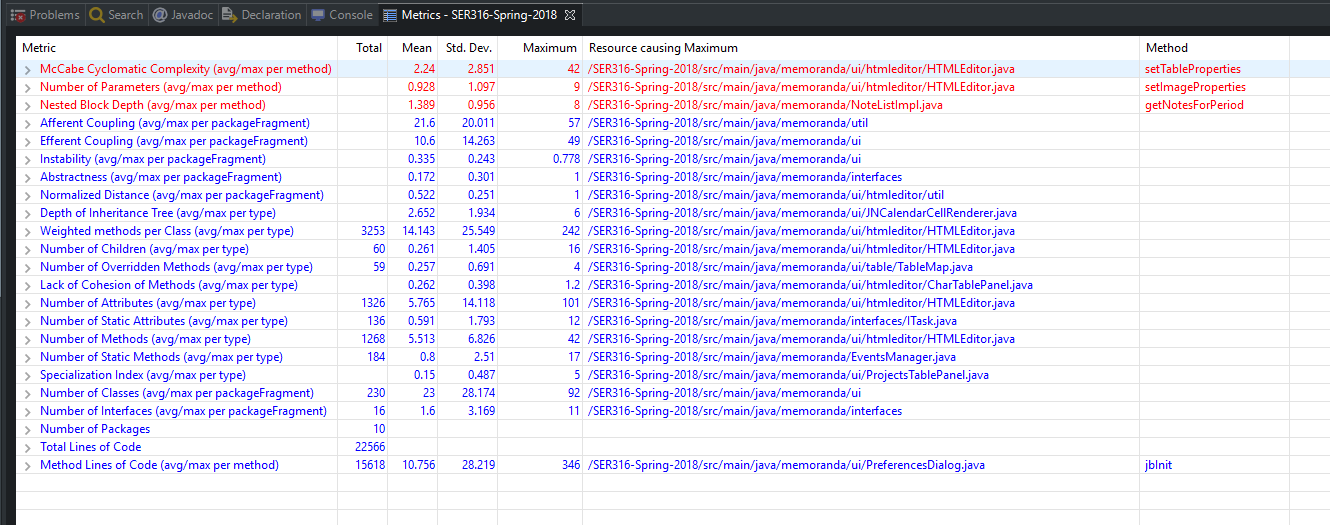
**Task 3:**

1. Refactored the method main.java.memoranda.EventImpl.getTime(). This method suffered from Long Method, and Data Clumps. The method did a bunch of stuff to return a Date object representing the current time, but none of it was necessary because the Date constructor is automatically set to the current system time. So I commented out all of the unnecessary code and just returned new date().

2. The TaskImpl class suffers from the "Data Class" code smell, where the TaskImpl just stores data but does little else, and TaskListImpl has a lot of logic which should be moved to the TaskImpl class. Also there was some duplicate code between TaskImpl and TaskListImpl.

Specifically I moved TaskListImpl.isActive() to TaskImpl.isActive() and I removed TaskListImpl.convertToTaskObjects() because TaskImpl.convertToTaskObjects() did the same thing, and refactored the code to use the TaskImpl version.

3 After:



Not much changed but there were a few minor improvements. Total Lines of Code went down by 24, Cyclomatic Complexity was lowered slightly, Weighted methods per class was lowered slighty, as well as Nested Block Depth, and number of methods. Number of Static Methods went up a little bit.