Assignment 7 Refactoring/Metrics
Michael McCulley

Task 1 – Size

1. 22539
2. HTMLEditor.java with 2144 LOC
3. It appears to take a count of all lines that have some sort of code on them. This includes, closing brackets, but does not count empty lines or comments.

Task 1 – Cohesion

1. LCOM2 measures the amount, for a given class, that methods reference class attributes. The calculation is:

   1 - (total number of attributes referenced across all methods)/(total number attributes * total number of methods).

2. It looks like ResourceTypeDialog.java has the highest mean of 1.1. (If mean is what we're going for, here. I'll be honest, there's barely any information on the internet about this stuff, and the majority of it is aligned with the above calculation. But, I have no idea what these metrics represent, especially given that this has a value greater than 1.)

   That said, it does appear in general that UI classes have higher cohesion. This makes sense because there are often a lot of attributes for UI, and often, many are updated when a change is made, and a subsequent method is called.

Task 1 – Complexity

1. The average cyclomatic complexity (CC) of main is 2.241.
2. Finder.java has the highest CC I could find, at 5.5.
3. I dropped the CC on TableSorter.java from 3.316 to 2.81. I noticed that the method compareRowsByColumn was repeatedly comparing two numbers to return -1, 1, or 0 based on their relative values. So, I pulled out those repeated checks and created the below method to replace it. (I actually created two methods, overloaded, one to compare ints and another to compare doubles).

```
public int compareNums(int n1, int n2) {
    if (n1 < n2) {
        return -1;
    } else if (n1 > n2) {
        return 1;
    } else {
        return 0;
    }
}
```

Assignment 7 Refactoring/Metrics
Michael McCulley

Task 1 – Coupling

1. Afferent coupling measures how many classes depend on the measured class. Efferent coupling measure how many classes the measured class depends on.
2. main.java.memoranda.util has the worst afferent coupling value of 57.
    a. This makes total sense. The nature of a utility class is to provide value for other classes.
3. main.java.memoranda.ui has the worst efferent coupling value of 49.
    a. Again, this makes sense. UI classes tend to use many different classes for specific UI elements.

Task 1 – Worst Quality

I'm going through these metrics, and I'm finding it difficult to find much value in what they represent. It's almost like I can find excuses for the classes pointed by high values of these metrics.

My first instinct is that the lack of cohesion would be perhaps the best indicator of a poorly constructed class. It seems like there should be some value in that metric. However, I don't know if this tools is calculating things correctly or not, but I glanced through several classes with the highest LCOM values, and none of them seem particularly troubling.

Moving onto cyclomatic complexity, the classes with a high value in this metric seem to mostly be around configuration, which makes sense. There will be a lot of if-else scenarios depending on what checkboxes are checked. That's not to say this isn't a useful metric in general. I believe in a lot of circumstances, this metric shows opportunities to pull out methods and get rid of repeated code. But, these values would have to be compared on a 1-by-1 basis, looking at the overall nature of the code.

I'm just not sure that coupling is a good measure in general. Seems like properly segmented and reusable code will show high values of coupling.

That leaves lines of code. Of all the metrics produced by this tool, this metric appears to relate to poor code most clearly. **HTMLEditor.java** has over 2000 lines. I understand that, for the most part, the code does refer to related functionality. However, just for the sake of readability and maneuverability, this should be broken up into subclasses. I select this class as being of the worst quality.

Assignment 7 Refactoring/Metrics
Michael McCulley

Task 2

Before:

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum
McCabe Cyclomatic Complexity (avg/max per method)		2.235	2.799	42	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/HTMLEditor.java
Number of Parameters (avg/max per method)		0.93	1.097	9	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/HTMLEditor.java
Nested Block Depth (avg/max per method)		1.391	0.955	8	/SER316-Spring-2018/src/main/java/memoranda/NoteListImpl.java
Afferent Coupling (avg/max per packageFragment)		19.333	19.653	57	/SER316-Spring-2018/src/main/java/memoranda/util
Efferent Coupling (avg/max per packageFragment)		11.444	15.276	49	/SER316-Spring-2018/src/main/java/memoranda/ui
Instability (avg/max per packageFragment)		0.36	0.247	0.778	/SER316-Spring-2018/src/main/java/memoranda/ui
Abstractness (avg/max per packageFragment)		0.111	0.137	0.333	/SER316-Spring-2018/src/main/java/memoranda/date
Normalized Distance (avg/max per packageFragment)		0.529	0.237	1	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/util
Depth of Inheritance Tree (avg/max per type)		2.652	1.934	6	/SER316-Spring-2018/src/main/java/memoranda/ui/JNCalendarCellRenderer.java
Weighted methods per Class (avg/max per type)	3250	14.13	25.508	242	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/HTMLEditor.java
Number of Children (avg/max per type)	60	0.261	1.405	16	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/HTMLEditor.java
Number of Overridden Methods (avg/max per type)	59	0.257	0.691	4	/SER316-Spring-2018/src/main/java/memoranda/ui/table/TableMap.java
Lack of Cohesion of Methods (avg/max per type)		0.262	0.398	1.2	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/CharTablePanel.java
Number of Attributes (avg/max per type)	1326	5.765	14.118	101	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/HTMLEditor.java
Number of Static Attributes (avg/max per type)	136	0.591	1.793	12	/SER316-Spring-2018/src/main/java/memoranda/Task.java
Number of Methods (avg/max per type)	1271	5.526	6.851	42	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/HTMLEditor.java
Number of Static Methods (avg/max per type)	183	0.796	2.51	17	/SER316-Spring-2018/src/main/java/memoranda/EventsManager.java
Specialization Index (avg/max per type)		0.15	0.487	5	/SER316-Spring-2018/src/main/java/memoranda/ui/ProjectsTablePanel.java
Number of Classes (avg/max per packageFragment)	230	25.556	29.833	92	/SER316-Spring-2018/src/main/java/memoranda/ui
Number of Interfaces (avg/max per packageFragment)	16	1.778	3.292	11	/SER316-Spring-2018/src/main/java/memoranda
Number of Packages	9				
Total Lines of Code	22526				
Method Lines of Code (avg/max per method)	15620	10.743	28.137	346	/SER316-Spring-2018/src/main/java/memoranda/ui/PreferencesDialog.java

After:

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum
McCabe Cyclomatic Complexity (avg/max per method)		2.235	2.799	42	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/HTMLEditor.java
Number of Parameters (avg/max per method)		0.93	1.097	9	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/HTMLEditor.java
Nested Block Depth (avg/max per method)		1.391	0.955	8	/SER316-Spring-2018/src/main/java/memoranda/NoteListImpl.java
Afferent Coupling (avg/max per packageFragment)		21.7	20.13	57	/SER316-Spring-2018/src/main/java/memoranda/util
Efferent Coupling (avg/max per packageFragment)		10.7	14.304	49	/SER316-Spring-2018/src/main/java/memoranda/ui
Instability (avg/max per packageFragment)		0.336	0.243	0.778	/SER316-Spring-2018/src/main/java/memoranda/ui
Abstractness (avg/max per packageFragment)		0.176	0.299	1	/SER316-Spring-2018/src/main/java/memoranda/interfaces
Normalized Distance (avg/max per packageFragment)		0.517	0.249	1	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/util
Depth of Inheritance Tree (avg/max per type)		2.652	1.934	6	/SER316-Spring-2018/src/main/java/memoranda/ui/JNCalendarCellRenderer.java
Weighted methods per Class (avg/max per type)	3250	14.13	25.508	242	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/HTMLEditor.java
Number of Children (avg/max per type)	60	0.261	1.405	16	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/HTMLEditor.java
Number of Overridden Methods (avg/max per type)	59	0.257	0.691	4	/SER316-Spring-2018/src/main/java/memoranda/ui/table/TableMap.java
Lack of Cohesion of Methods (avg/max per type)		0.262	0.398	1.2	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/CharTablePanel.java
Number of Attributes (avg/max per type)	1326	5.765	14.118	101	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/HTMLEditor.java
Number of Static Attributes (avg/max per type)	136	0.591	1.793	12	/SER316-Spring-2018/src/main/java/memoranda/interfaces/ITask.java
Number of Methods (avg/max per type)	1271	5.526	6.851	42	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/HTMLEditor.java
Number of Static Methods (avg/max per type)	183	0.796	2.51	17	/SER316-Spring-2018/src/main/java/memoranda/EventsManager.java
Specialization Index (avg/max per type)		0.15	0.487	5	/SER316-Spring-2018/src/main/java/memoranda/ui/ProjectsTablePanel.java
Number of Classes (avg/max per packageFragment)	230	23	28.245	92	/SER316-Spring-2018/src/main/java/memoranda/ui
Number of Interfaces (avg/max per packageFragment)	16	1.6	2.835	10	/SER316-Spring-2018/src/main/java/memoranda/interfaces
Number of Packages	10				
Total Lines of Code	22573				
Method Lines of Code (avg/max per method)	15620	10.743	28.137	346	/SER316-Spring-2018/src/main/java/memoranda/ui/PreferencesDialog.java

The afferent and efferent coupling values changed. Afferent coupling changed from 19.33 to 21.7. This is for the worse, as good code aims to have lower coupling. This corresponds to the fact that all of the interfaces were moved from the package containing the classes implementing them. Specifically, the afferent value is very high in the new interface package (46), and this is bringing up the average for the codebase. I'm not sure exactly why efferent coupling went down. Perhaps because the interfaces and their implementations both referred to external packages about equally, and when split apart, each's efferent value did not change dramatically.

Assignment 7 Refactoring/Metrics
Michael McCulley

Task 3

Smell within a class: Duplicate code

Altered: main.java.memoranda.ui/AddResourceDialog.java

Code in this class repeatedly used the same pattern to setup the layout for its UI components. I took out the duplicated code to its own method and called that for each component.

```
private void layoutGridBag(Component component, int x, int y) {
    gbc = new GridBagConstraints();
    gbc.gridx = x; gbc.gridy = y;
    gbc.insets = new Insets(5, 5, 5, 5);
    gbc.anchor = GridBagConstraints.WEST;
    gbc.fill = GridBagConstraints.HORIZONTAL;
    areaPanel.add(component, gbc);
}
```

Smell between classes: Data clump

Created main.java.memoranda/ProjectData.java

Altered: CurrentProject.java, IProjectListener.java, AppFrame.java, AgendaPanel.java, DailyItemsPanel.java, TaskPanel.java, TaskTabel.java, SearchPanel.java, ResourcesTable.java, ProjectsPanel.java, JNCalendarPanel.java, NoteList.java

Note: I did not label in all these files the comment // TASK 2-2 SMELL BETWEEN CLASSES. I did this in ProjectData.java, CurrentProject.java, and IProjectListener.java.

I refactored the main data elements in the project (the project itself, the notes list, task list, and resources list) into a separate class. Then, I changed the projectChange method for IProjectListener.java to reference a ProjectData class.

The motivation for this is because anytime you add a collection to the project (as we did several times in our course project) you would have to add it to the projectChange method of every implementation of IProjectListener. This means editing the method parameters in almost all of the above altered classes. Furthermore, the vast majority of those listeners did not do anything with the passed references!

After refactoring, if you add a collection to the project, you only need to add it to the ProjectData class. Then, if you need to access the specific new collection, you can reference it from the passed ProjectData object.

Assignment 7 Refactoring/Metrics
Michael McCulley

Final Metrics:

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum
McCabe Cyclomatic Complexity (avg/max per method)		2.226	2.79	42	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/HTMLEditor.java
Number of Parameters (avg/max per method)		0.926	1.093	9	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/HTMLEditor.java
Nested Block Depth (avg/max per method)		1.388	0.952	8	/SER316-Spring-2018/src/main/java/memoranda/NoteListImpl.java
Afferent Coupling (avg/max per packageFragment)		21.9	20.3	57	/SER316-Spring-2018/src/main/java/memoranda/util
Efferent Coupling (avg/max per packageFragment)		10.9	14.335	49	/SER316-Spring-2018/src/main/java/memoranda/ui
Instability (avg/max per packageFragment)		0.338	0.242	0.778	/SER316-Spring-2018/src/main/java/memoranda/ui
Abstractness (avg/max per packageFragment)		0.176	0.3	1	/SER316-Spring-2018/src/main/java/memoranda/interfaces
Normalized Distance (avg/max per packageFragment)		0.518	0.247	1	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/util
Depth of Inheritance Tree (avg/max per type)		2.645	1.933	6	/SER316-Spring-2018/src/main/java/memoranda/ui/JNCalendarCellRenderer.java
Weighted methods per Class (avg/max per type)	3261	14.117	25.455	242	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/HTMLEditor.java
Number of Children (avg/max per type)	60	0.26	1.403	16	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/HTMLEditor.java
Number of Overridden Methods (avg/max per type)	59	0.255	0.69	4	/SER316-Spring-2018/src/main/java/memoranda/ui/table/TableMap.java
Lack of Cohesion of Methods (avg/max per type)		0.264	0.398	1.2	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/CharTablePanel.java
Number of Attributes (avg/max per type)	1330	5.758	14.088	101	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/HTMLEditor.java
Number of Static Attributes (avg/max per type)	133	0.576	1.768	12	/SER316-Spring-2018/src/main/java/memoranda/interfaces/ITask.java
Number of Methods (avg/max per type)	1282	5.55	6.846	42	/SER316-Spring-2018/src/main/java/memoranda/ui/htmleditor/HTMLEditor.java
Number of Static Methods (avg/max per type)	183	0.792	2.505	17	/SER316-Spring-2018/src/main/java/memoranda/EventsManager.java
Specialization Index (avg/max per type)		0.149	0.486	5	/SER316-Spring-2018/src/main/java/memoranda/ui/ProjectsTablePanel.java
Number of Classes (avg/max per packageFragment)	231	23.1	28.275	92	/SER316-Spring-2018/src/main/java/memoranda/ui
Number of Interfaces (avg/max per packageFragment)	16	1.6	2.835	10	/SER316-Spring-2018/src/main/java/memoranda/interfaces
Number of Packages	10				
Total Lines of Code	22583				
Method Lines of Code (avg/max per method)	15588	10.64	27.916	346	/SER316-Spring-2018/src/main/java/memoranda/ui/PreferencesDialog.java

Nothing changed significantly. This was expected, since packaging up a data clump doesn't really move around much code. This specific type of refactoring, though very helpful to the programmer, doesn't seem to evaluate very well for these metrics. This is not to mention, that so few lines of code were actually refactored compared to the giant 20,000+ codebase.