Neil Sabott

Arizona State University

SER316: Fall 2018

Assignment 7

Task 1

Size

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

2,187

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

EventsManager.java 329

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

Method 1: Count each statement as 1, everything else as 0. There are 28 LOC without including any comments, and each LOC is counted as only 1.

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).

LCOM2 represents the percentage of methods that do not access a specific attribute over all attributes in the class, which is contingent on the number of attributes being greater than zero. It is calculated like this:

If there are no pairs of methods that share attributes, the result is: 0
Otherwise:
LCOM2 = |# of pairs that do NOT share attributes| - |# of pairs that DO share attributes|

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

TaskListImpl.java Because it's managing 3 Objects (Project, Document, and Element) that don't share any attributes, but all work in conjunction in the program.

Complexity

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

1.746

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

EventsManager.java 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.

I edited the setMark function inside the NoteImpl.java class so that instead of checking for null, then also checking for !null, it simply checks once and either adds and returns or defaults to removing the attribute. Since it can either be null or !null, there is no need to check for !null if the null was already checked.

Total:	*Old*	*New*	**NoteImpl:**	*Old*	*New*	**setMark:**	*Old*	*New*
	1.749	1.732		1.9	1.8		4	3

Package-level Coupling

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

Afferent coupling measures the number of classes on which a given class depends.

Usually if this is high, it represents that the class is most likely meant to be part of the App side.

Efferent coupling measures how many classes depend on a given class.

Usually if this is high, it represents that the class is most like meant to be part of the Services side.

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

main.java.memoranda.util 57

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

main.java.memoranda.ui 49

Worst quality

1. Which class has the worst quality and why? Use the metrics to support your answer.

You can use any metric or different metrics together but you have to give valid arguments.

Resource.java is the class with the worst quality, here's my reasoning:

	CC	LCohes	#Param	BDpth	/LOC	Total (%)
[Importance]	*x5*	*x4*	*x3*	*x2*	*x1*	
EventsManager.java	2.5	0	1.125	1.281		
	12.5	*0*	*3.375*	*2.562*	*/329*	***5.60395***
NoteListImpl.java	2.485	0.108	0.667	1.545		
	12.425	*0.432*	*0.2001*	*3.09*	*/286*	***5.64584***
TaskImpl.java	1.919	0.409	0.541	1.297		
	9.595	*1.636*	*1.623*	*2.594*	*/243*	***6.35720***
ProjectImpl.java	2.133	0	0.467	1.267		
	10.665	*0*	*1.401*	*2.534*	*/105*	***13.9048***
TaskList.java	1	0	1.286	0		
	5	*0*	*3.858*	*0*	*/19*	***46.6211***
Resource.java	1	0.667	0.8	1		
	5	*2.668*	*2.4*	*2*	*/23*	***52.4696***
ResourceListImpl.java	61.889	0.667	1	1.222		
	9.445	*2.668*	*3*	*2.444*	*/68*	***25.8191***

Using this chart, it tracks how "bad" each class is factoring in the weighted importance of each category of metrics used. It takes this weighted value and divides it by how many LOC the class has. The higher the "bad" factor is over the amount of LOC in each class gives the total "% bad" each class is.

The results come back that **Resource.java** is the worst class given how many "bad" factors there are **PER LOC** in the class. This should not be confused with the class with **MOST** "bad" factors, in my opinion it is a better judge to take a relationship to determine the worst class.

Task 2

Before:

Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per met		1.743	1.543	16	/SER316-Spring-2018/src/main/java/memoranda/Eve...	getRepeatableEventsFor...
Number of Parameters (avg/max per method)		0.675	1.004	8	/SER316-Spring-2018/src/main/java/memoranda/Eve...	createRepeatableEvent
Nested Block Depth (avg/max per method)		0.997	0.945	8	/SER316-Spring-2018/src/main/java/memoranda/Not...	getNotesForPeriod
Afferent Coupling	34					
Efferent Coupling	21					
Instability	0.382					
Abstractness	0.275					
Normalized Distance	0.343					
Depth of Inheritance Tree (avg/max per type)		0.854	0.607	2	/SER316-Spring-2018/src/main/java/memoranda/Eve...	
Weighted methods per Class (avg/max per type)	584	14.244	16.074	71	/SER316-Spring-2018/src/main/java/memoranda/Tas...	
Number of Children (avg/max per type)	23	0.561	1.624	10	/SER316-Spring-2018/src/main/java/memoranda/Pro...	
Number of Overridden Methods (avg/max per type	3	0.073	0.341	2	/SER316-Spring-2018/src/main/java/memoranda/Tas...	
Lack of Cohesion of Methods (avg/max per type)		0.093	0.211	0.679	/SER316-Spring-2018/src/main/java/memoranda/Tas...	
Number of Attributes (avg/max per type)	30	0.732	1.037	4	/SER316-Spring-2018/src/main/java/memoranda/Tas...	
Number of Static Attributes (avg/max per type)	46	1.122	2.549	12	/SER316-Spring-2018/src/main/java/memoranda/Tas...	
Number of Methods (avg/max per type)	274	6.683	7.687	37	/SER316-Spring-2018/src/main/java/memoranda/Tas...	
Number of Static Methods (avg/max per type)	61	1.488	3.768	17	/SER316-Spring-2018/src/main/java/memoranda/Eve...	
Specialization Index (avg/max per type)		0.05	0.308	2	/SER316-Spring-2018/src/main/java/memoranda/Star...	
Number of Classes	41					
Number of Interfaces	11					
Total Lines of Code	2186					
Method Lines of Code (avg/max per method)	1258	3.755	5.21	33	/SER316-Spring-2018/src/main/java/memoranda/Eve...	getRepeatableEventsFor...

After:

Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per met		2.029	1.733	16	/SER316-Spring-2018/src/main/java/memoranda/Eve...	getRepeatableEventsFor...
Number of Parameters (avg/max per method)		0.707	1.009	8	/SER316-Spring-2018/src/main/java/memoranda/Eve...	createRepeatableEvent
Nested Block Depth (avg/max per method)		1.38	0.841	8	/SER316-Spring-2018/src/main/java/memoranda/Not...	getNotesForPeriod
Afferent Coupling	31					
Efferent Coupling	16					
Instability	0.34					
Abstractness	0					
Normalized Distance	0.66					
Depth of Inheritance Tree (avg/max per type)		1.167	0.373	2	/SER316-Spring-2018/src/main/java/memoranda/Eve...	
Weighted methods per Class (avg/max per type)	491	16.367	17.847	71	/SER316-Spring-2018/src/main/java/memoranda/Tas...	
Number of Children (avg/max per type)	0	0	0	0	/SER316-Spring-2018/src/main/java/memoranda/Eve...	
Number of Overridden Methods (avg/max per type	3	0.1	0.396	2	/SER316-Spring-2018/src/main/java/memoranda/Tas...	
Lack of Cohesion of Methods (avg/max per type)		0.127	0.237	0.679	/SER316-Spring-2018/src/main/java/memoranda/Tas...	
Number of Attributes (avg/max per type)	30	1	1.095	4	/SER316-Spring-2018/src/main/java/memoranda/Tas...	
Number of Static Attributes (avg/max per type)	29	0.967	2.008	7	/SER316-Spring-2018/src/main/java/memoranda/Eve...	
Number of Methods (avg/max per type)	181	6.033	7.834	37	/SER316-Spring-2018/src/main/java/memoranda/Tas...	
Number of Static Methods (avg/max per type)	61	2.033	4.278	17	/SER316-Spring-2018/src/main/java/memoranda/Eve...	
Specialization Index (avg/max per type)		0.068	0.359	2	/SER316-Spring-2018/src/main/java/memoranda/Star...	
Number of Classes	30					
Number of Interfaces	0					
Total Lines of Code	2063					
Method Lines of Code (avg/max per method)	1258	5.198	5.484	33	/SER316-Spring-2018/src/main/java/memoranda/Eve...	getRepeatableEventsFor...

Compare the results of your 2 metrics exports. Did any of the metrics change for the better after your refactorings. Pick and state a metric (or metrics) whose value changed, and indicated why it changed and whether it changed for the better (or worse) because of the refactoring.

It looks like Afferent and Efferent Coupling have changed for the better after refactoring. Also, the Number of Children has been the most obviously affected value, going directly down to 0 after the refactoring for all classes. This means all the classes with children were pushed to the interfaces package, thus improving the metrics for the main.java.memoranda package.

Task 3

WITHIN A CLASS

src/main/java/memoranda/ui/NotesControlPanel.java

I changed the *tabbedPane_stateChanged(ChangeEvent e)* method to correct the switch-statement Code Smell. Switch statements are considered bad coding practice, so I altered it to an if/else if statement, at the same time I also fixed a few other minor issues regarding using tabs and proper spacing between brackets.

BETWEEN CLASSES

src/main/java/memoranda/EventsManager.java
src/main/java/memoranda/ui/EventsPanel.java

I combined the like parameters (int & CalendarDate) into an array so it reduced the number of passed parameters from 8 to 4. Having too many parameters is a code smell and not a good programming habit.

AFTER REFACTORING

Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per met		2.029	1.733	16	/SER316-Spring-2018/src/main/java/memoranda/Eve...	getRepeatableEventsFor...
Number of Parameters (avg/max per method)		0.69	0.918	7	/SER316-Spring-2018/src/main/java/memoranda/Tas...	createTask
Nested Block Depth (avg/max per method)		1.38	0.841	8	/SER316-Spring-2018/src/main/java/memoranda/Not...	getNotesForPeriod
Afferent Coupling	31					
Efferent Coupling	16					
Instability	0.34					
Abstractness	0					
Normalized Distance	0.66					
Depth of Inheritance Tree (avg/max per type)		1.167	0.373	2	/SER316-Spring-2018/src/main/java/memoranda/Eve...	
Weighted methods per Class (avg/max per type)	491	16.367	17.847	71	/SER316-Spring-2018/src/main/java/memoranda/Tas...	
Number of Children (avg/max per type)	0	0	0	0	/SER316-Spring-2018/src/main/java/memoranda/Eve...	
Number of Overridden Methods (avg/max per type	3	0.1	0.396	2	/SER316-Spring-2018/src/main/java/memoranda/Tas...	
Lack of Cohesion of Methods (avg/max per type)		0.127	0.237	0.679	/SER316-Spring-2018/src/main/java/memoranda/Tas...	
Number of Attributes (avg/max per type)	30	1	1.095	4	/SER316-Spring-2018/src/main/java/memoranda/Tas...	
Number of Static Attributes (avg/max per type)	29	0.967	2.008	7	/SER316-Spring-2018/src/main/java/memoranda/Eve...	
Number of Methods (avg/max per type)	181	6.033	7.834	37	/SER316-Spring-2018/src/main/java/memoranda/Tas...	
Number of Static Methods (avg/max per type)	61	2.033	4.278	17	/SER316-Spring-2018/src/main/java/memoranda/Eve...	
Specialization Index (avg/max per type)		0.068	0.359	2	/SER316-Spring-2018/src/main/java/memoranda/Star...	
Number of Classes	30					
Number of Interfaces	0					
Total Lines of Code	2059					
Method Lines of Code (avg/max per method)	1258	5.198	5.484	33	/SER316-Spring-2018/src/main/java/memoranda/Eve...	getRepeatableEventsFor...

Compare the results of your metrics at the end of task 2 and what you have now. Did any of the metrics change for the better after your 2 refactoring. Pick and state a metric (or metrics) whose value changed, and indicated why it changed and whether it changed for the better (or worse) because of the refactoring

Yes, the Number of Parameters dropped from 0.707 to 0.69. This change is for the better because as mentioned previously, having too many parameters is a code smell and bad programming form.