Assignment 8 – Metrics

Paul Traxler – 11-27-18

Task 1

Size –

1. This project has 22,539 LOC. src.main.java.memoranda has 2,189 LOC.
2. The largest single file in the entire project is HTMLEditor.java with 2144 LOC. In src.main.java.memoranda the largest file is EventsManager.java with 329 LOC.
3. In counting CurrentNote.java's LOC, the Metrics plugin used a physical LOC method. Whitespace is omitted and comments are omitted.

Cohesion –

1. LCOM2 gives a percentage telling how many methods use different variables in a class. In a cohesive class they would use the same variables, and if many methods act on a different set of variables the class could potentially be split.
2. In src.main.java.memoranda, HistoryItem.java is the most cohesive class with a value of 0.333 (lower is better with LCOM2). This class is cohesive because every method uses the same variables.

Complexity –

1. In src.main.java.memoranda, the complexity has a mean of 1.746.
2. The worse class in this package is Start.java with a mean of 3.5.
3. I chose to reduce the complexity of ProjectImpl.java in src.main.java.memoranda, because it jumped out at me. Originally the class had a complexity of 2.133, and specifically the method getStatus() had a complexity of 6. This method checked the project status based on if it was before the start date, after the end date or in-between. I realized that you could check the status in all cases by just checking if it was before or after it's start or end dates, because because in all other cases it's in between so we don't need a dedicated conditional for that. This was a small change that just brought the complexity of the method from 6 to 5, and the complexity of the class from 2.133 to 2.067.

Package-level Coupling –

1. Afferent coupling is the amount of classes in a package that depend on each other, while Efferent couple is the amount of classes that depend on external classes.

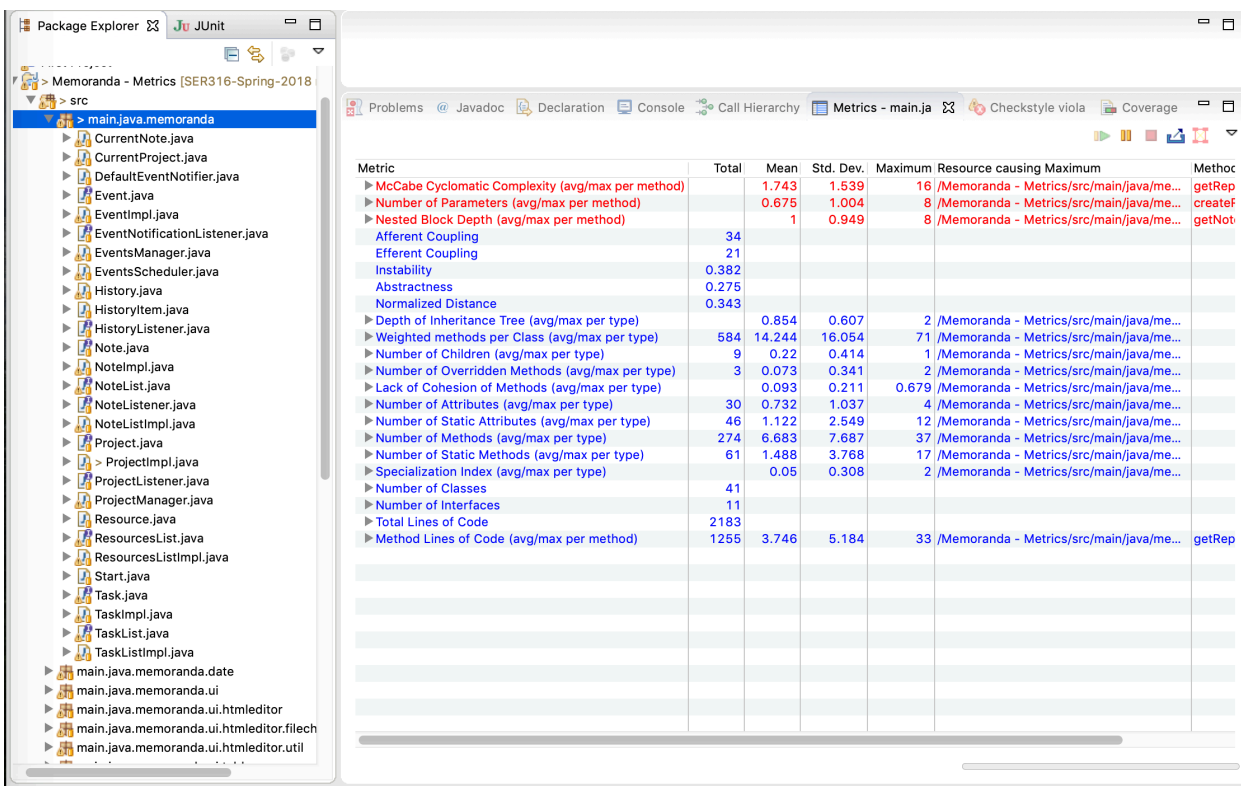
2. The package with the worst Afferent coupling is src.main.java.memoranda.util with a value of 57.
3. The package with the worse Efferent coupling is src.main.java.memoranda.ui with a value of 49.

Worst Quality –

I believe that in the package src.main.java.memoranda the class with the worst quality is TaskListImpl.java. This class has a fairly high complexity at 1.91 (4th highest in package) and has the highest Lack of Cohesion in the package at .679. On top of both these things it has the third most LOC at 246. This adds up to a long and complex class that lacks cohesion, making it difficult to maintain. This is why I think it has the worst quality of this package.

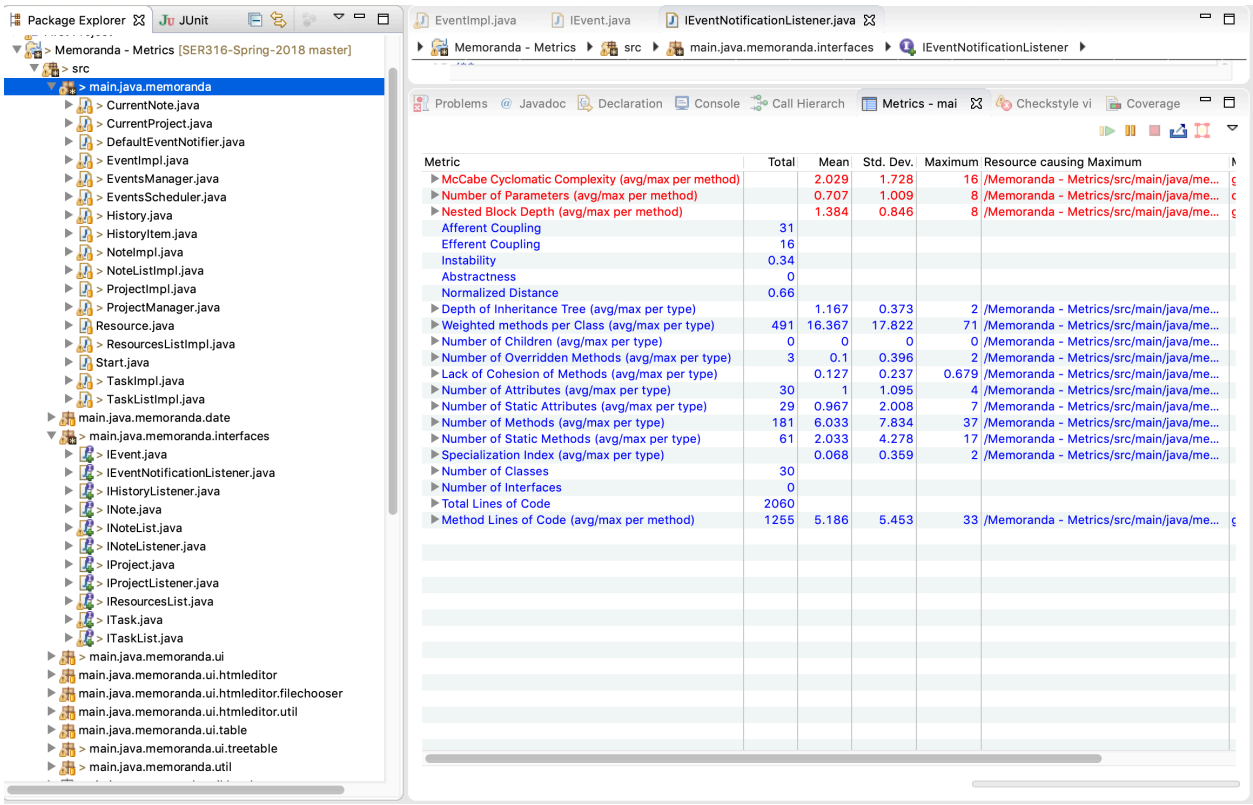
Task 2: Refactoring

Before:



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Methoc
McCabe Cyclomatic Complexity (avg/max per method)		1.743	1.539	16	/Memoranda - Metrics/src/main/java/me...	getRep
Number of Parameters (avg/max per method)		0.675	1.004	8	/Memoranda - Metrics/src/main/java/me...	createF
Nested Block Depth (avg/max per method)		1	0.949	8	/Memoranda - Metrics/src/main/java/me...	getNot
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	/Memoranda - Metrics/src/main/java/me...	
Weighted methods per Class (avg/max per type)	584	14.244	16.054	71	/Memoranda - Metrics/src/main/java/me...	
Number of Children (avg/max per type)	9	0.22	0.414	1	/Memoranda - Metrics/src/main/java/me...	
Number of Overridden Methods (avg/max per type)	3	0.073	0.341	2	/Memoranda - Metrics/src/main/java/me...	
Lack of Cohesion of Methods (avg/max per type)		0.093	0.211	0.679	/Memoranda - Metrics/src/main/java/me...	
Number of Attributes (avg/max per type)	30	0.732	1.037	4	/Memoranda - Metrics/src/main/java/me...	
Number of Static Attributes (avg/max per type)	46	1.122	2.549	12	/Memoranda - Metrics/src/main/java/me...	
Number of Methods (avg/max per type)	274	6.683	7.687	37	/Memoranda - Metrics/src/main/java/me...	
Number of Static Methods (avg/max per type)	61	1.488	3.768	17	/Memoranda - Metrics/src/main/java/me...	
Specialization Index (avg/max per type)		0.05	0.308	2	/Memoranda - Metrics/src/main/java/me...	
Number of Classes	41					
Number of Interfaces	11					
Total Lines of Code	2183					
Method Lines of Code (avg/max per method)	1255	3.746	5.184	33	/Memoranda - Metrics/src/main/java/me...	getRep

After:



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum
McCabe Cyclomatic Complexity (avg/max per method)		2.029	1.728	16	/Memoranda - Metrics/src/main/java/me...
Number of Parameters (avg/max per method)		0.707	1.009	8	/Memoranda - Metrics/src/main/java/me...
Nested Block Depth (avg/max per method)		1.384	0.846	8	/Memoranda - Metrics/src/main/java/me...
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	/Memoranda - Metrics/src/main/java/me...
Weighted methods per Class (avg/max per type)	491	16.367	17.822	71	/Memoranda - Metrics/src/main/java/me...
Number of Children (avg/max per type)	0	0	0	0	/Memoranda - Metrics/src/main/java/me...
Number of Overridden Methods (avg/max per type)	3	0.1	0.396	2	/Memoranda - Metrics/src/main/java/me...
Lack of Cohesion of Methods (avg/max per type)		0.127	0.237	0.679	/Memoranda - Metrics/src/main/java/me...
Number of Attributes (avg/max per type)	30	1	1.095	4	/Memoranda - Metrics/src/main/java/me...
Number of Static Attributes (avg/max per type)	29	0.967	2.008	7	/Memoranda - Metrics/src/main/java/me...
Number of Methods (avg/max per type)	181	6.033	7.834	37	/Memoranda - Metrics/src/main/java/me...
Number of Static Methods (avg/max per type)	61	2.033	4.278	17	/Memoranda - Metrics/src/main/java/me...
Specialization Index (avg/max per type)		0.068	0.359	2	/Memoranda - Metrics/src/main/java/me...
Number of Classes	30				
Number of Interfaces	0				
Total Lines of Code	2060				
Method Lines of Code (avg/max per method)	1255	5.186	5.453	33	/Memoranda - Metrics/src/main/java/me...

Task 2 Analysis –

Most of the metrics changed for the worse after refactoring because the interfaces were holding the averages lower. For example, removing the interfaces brought the mean complexity up from 1.743 to 2.029. In this case I think it's good because it gives a better representation of the quality of the system.

Task 3

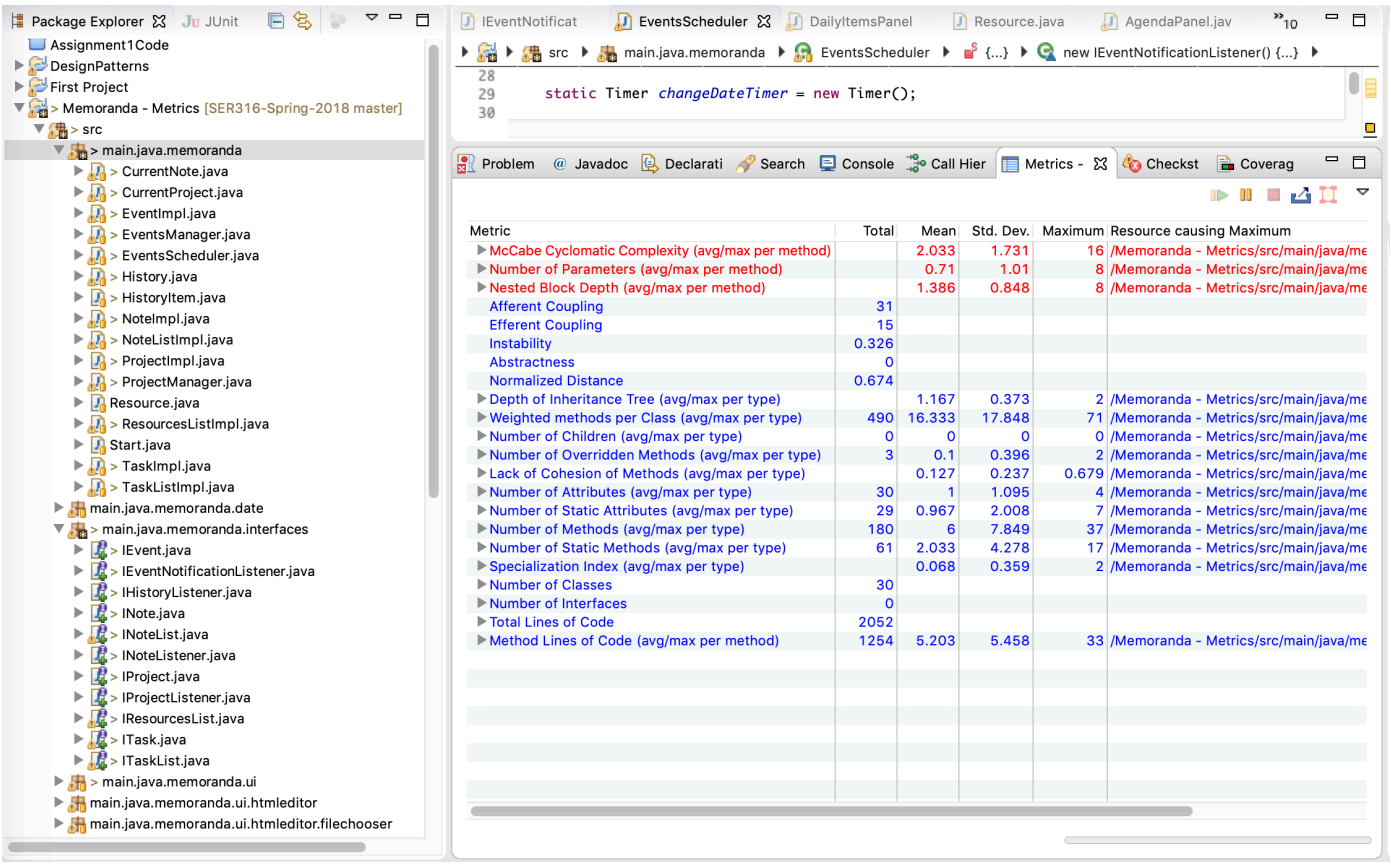
1. In src.main.java.memoranda.History.java on line 34 I found a Code Smell with a variable with too short of an identifier. There variable name was "p" and it was not at

all obvious what it's purpose was or what it's value indicated. I had to run the code in the debugger to make sure I understood exactly what it did so I could even name it. It turned out that it was used as the index of the previous item in the history list, so I named it as such. This makes the code more readable and maintainable.

2. For the smell between classes I went to src.main.java.memoranda.EventsSchedule.java line 32 and added 7 lines of code that made up the entirety of the class DefaultEventNotifier.java. That class did too little, and other places in the memoranda software declared the IEventsNotificationListener object inline. I deleted the DefaultEventNotifier class in favor of the inline method, because the class just wasn't necessary. This was a coding smell because A) it wasn't consistent with other parts of the program and B) having the class so short detracts from readability/maintainability and creates an unnecessary dependency.

Screenshot of new metrics on following page.

3. Post Code Smell Refactor Metrics



Analysis:

This refactor didn't have a major impact on the metrics, but some things did change. In deleting the DefaultEventNotifier class I decreased the Efferent Coupling from 16 to 15. This is because I was able to encase the functionality in the EventSchedule class and removed that external dependency. This is more desirable because it makes the class more reusable, readable, and maintainable since you have one less class to look at in the future.