Christopher Gold

Week 7: Refactoring

04/29/2018

Task 1:

Size:

1. Total LOC for the project (java.memoranda) is 2190
2. Largest code file in the project is EventsManager.java with 329 LOC.
3. It is usuing the basic SLOC method. It ignored white space and counted everything else (including comments and curly braces). It is not counting using logical lines of code.

Cohesion:

1. LCOM2 uses the Henderson-Sellers method. The LCOM2 stands for Lack of Cohesion Among Method of Class, which measures the extent of intersections of individual method parameter types lists with the parameter type list of all methods in the class. LCOM2 = 1 – sum(mA)/(m*a)

m	*number of procedures (methods) in class*
a	*number of variables (attributes) in class*
mA	*number of methods that access a variable (attribute)*
sum(mA)	*sum of mA over attributes of a class*

There are several with a mean of zero, so I'll choose one at random and say EventImpl.java. It has the highest as it's mean is zero. More than that however, there are no class level variables, only local variables contained within the methods. I believe this to be the reason why.

Complexity:

1. The cyclomatic complexity of the package is 1.746

2. The class with the worst complexity, on average, would be the EventsManager.java with a mean of 2.5

3.) I changed the event manager method getRepeatableEventsForDate() and extracted some of it's contents into a new method. In the new method, which is getRepeatableEventsForDateHelper(), I consolidated all the if checks that resulted in the same action being taken. This reduced the complexity score from 2.5 to 2.394. This reduced complexity as it is a method which contains many if and else if statements, which creates a lot edges. Essentially, it is a method trying to do too much on its own.

Package-level coupling:

1.) The easiest way to describe it would be that Efferent is a measure of the number of classes a class is using from other packages. Afferent would be the opposite as in the numbr ofother Classes that use it's class. For example:

```
Class Foo {

SomeClass x;

}

Class SomeOtherFoo {

SomeClass y;

}


Class SomeClass {

//..

}
```

Class Foo and class SomeOtherFoo are efferent as they use an outside class within their class. SomeClass would be considered Afferent as it Is being used in other classes.

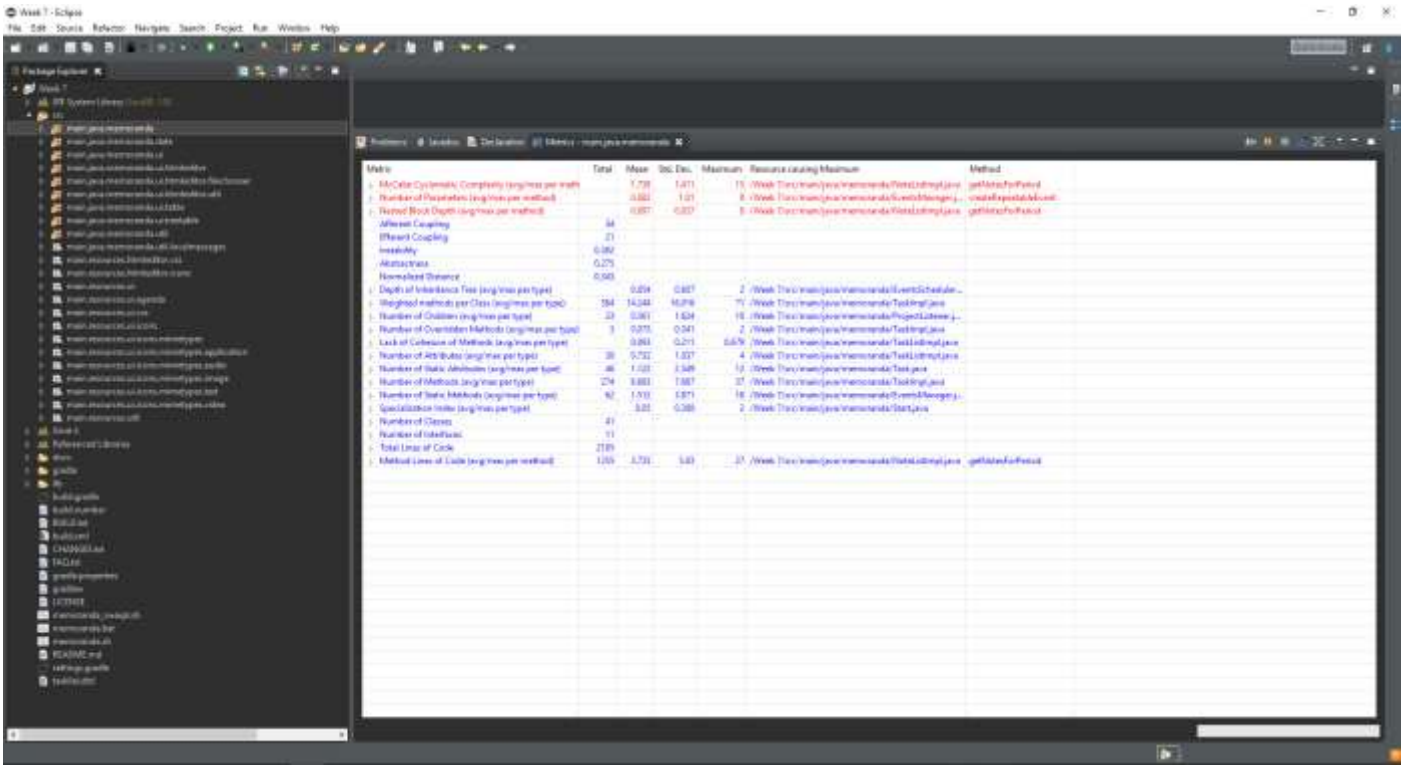
2.) The package with the most afferent coupling would be the main.java.memoranda.util package with a value of 57;

3.) The package with the most efferent coupling is the main.java.memoranda.ui package with a value of 49.

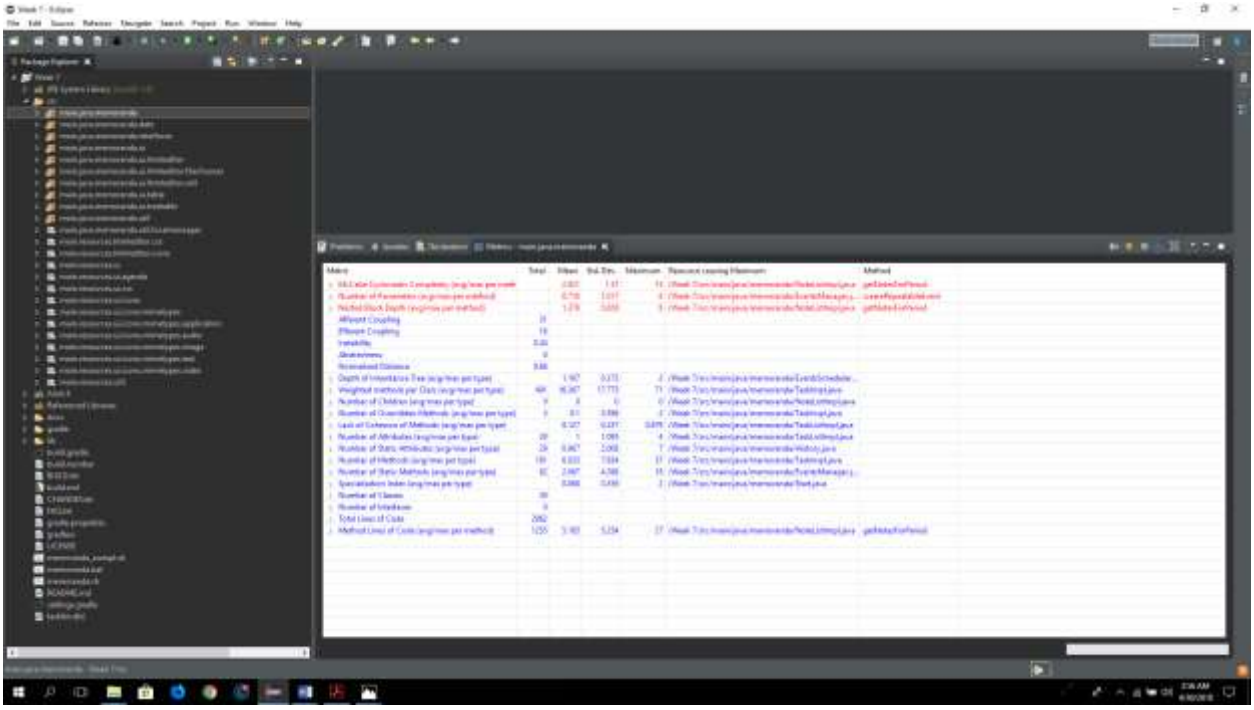
Worst Quality

I think the class with the worst quality would be the EventManager.java class. It has a high cyclomatic complexity as well as the highest number of parameters per method. In addition, it has the highest LOC count which makes it one of the more complicated classes in the package. It seems to me this class is doing more work than it likely needs to and that some of it could be broken down and split up. I know there are a lot of conditional statements in there that could be simplified in some way.

Task 2:



After



Step 8

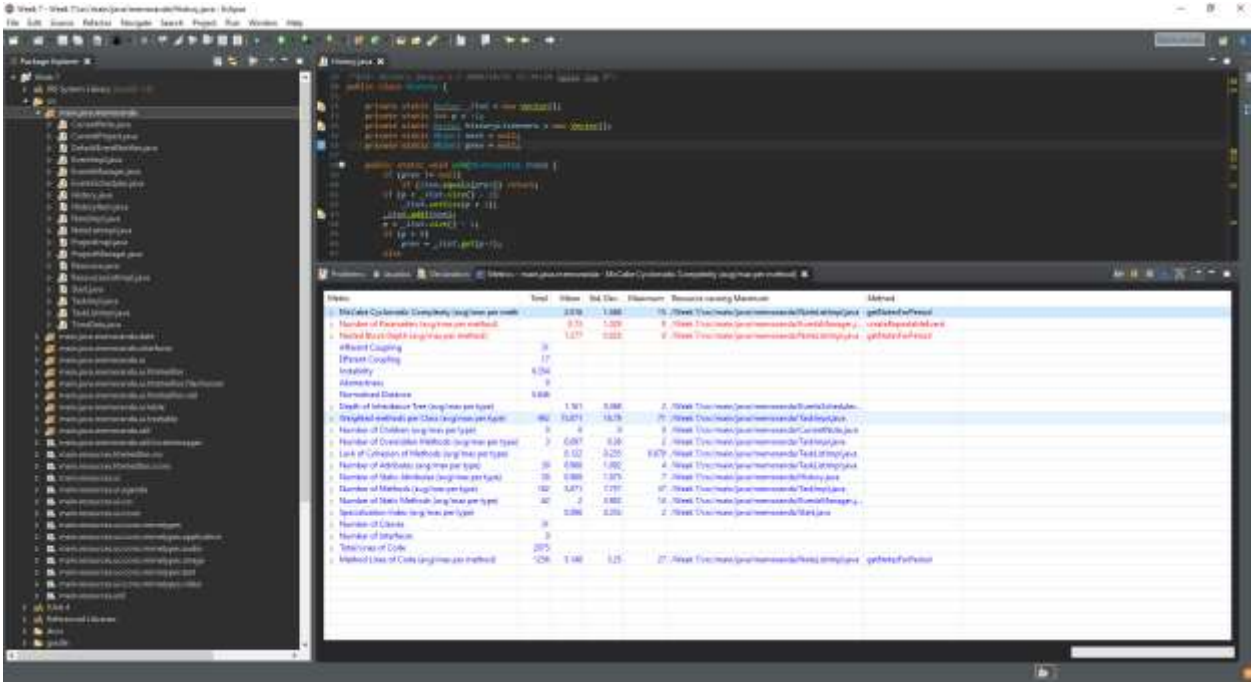
Quite a number of metrics increased for the worse actually. Cyclomatic complexity, number of parameters and nested block depth averages all increased further into the red. For most of these, I believe it Is due to the fact that the interfaces had very low scores in these metrics, so while included, they reduced the overall average. With these removed however, I think the values are more where they are supposed to be. This is especially true in the complexity score as each interface had a mean of 1 which impacted the scores rather significantly.

Task 3:

Step 1: Refactored main.java.memoranda.EventManager.java. This class was overly large and it contained a number of static classes that dealt with time that I felt could be in their own, outside class. I took out the static classes (Year, Month and Day), and the methods that were used to get their data, and put it into a new class called TimeData. This cut down the LOC for EventManager quite a bit and helped to decouple some of the workings.

Step 2: I found one code smell between classes where a particular class, History.java, was overly visible to the rest of the package. Too many of it's internals where open for any other class to call, and none of them have a need for that much access. That said, I went through and modified the internals to be private except for those ones that other classes needed to call. This resulted in a diminished public surface.

Step 3:



Step 4:

It would appear that, as a result of the refactoring, a few metrics have actually become worse. One would be the cyclomatic complexity value which changed from 1.738 to 2.016. This could perhaps be a result of extracting some methods rom the EventManager class into the TimeData class and having calls between them. The visibility changes to History shouldn't have had an effect on this. It also appears the number of parameters has increased from 0.683 to 0.73. This would be the result of having to add additional parameters to the methods I extracted to TimeData to make them still work with EventManager. Nested depth also increased from 0.997 to 1.377. This too likely due to the changes made to EventManager. A few metrics however did improve. Afferent coupling reduced from 34 to 31. Efferent from 21 to 17. Abstractness also went from 0.275 to zero. All of which, again, would be contributed to the extractions of classes from EventManager into TimeData.