Task 1:

Size:

1. 22539 LOC
2. In the main.java.memoranda package, EventsManager.java is the longest, with 329 LOC
3. It counts physical LOC, excluding whitespace and comments (Method 1 in the slides)

Cohesion:

|  |  |
| --- | --- |
| **M** | be the set of methods defined by the class |
| **F** | be the set of fields defined by the class |
| *r(f)* | be the number of methods that access field *f*, where *f* is a member of **F** |
| *<r>* | be the mean of *r(f)* over **F**. |

1. Henderson-Sellers defines Lack of Cohesion in Methods as follows. Let:

Then:

|  |  |  |
| --- | --- | --- |
| Lack of Cohesion in Methods | = | *<r>* - |**M**| |
| 1 - |**M**| |
|  |  |  |

Found at: <http://eclipse-metrics.sourceforge.net/descriptions/pages/cohesion/HendersonSellers.html>

1. There are 21 classes that tie for the highest cohesion (all have 0 lack of cohesion). The reason these classes have the highest cohesion varies depending on the class. Several of the classes are interfaces which do not declare any class variables. Some of them declare only static variables and methods. Some of them are “normal” implementation classes, but are just very cohesive.

Complexity:

1. 1.746
2. Start.java has 3.5 cyclomatic complexity
3. I was able to reduce the complexity of Start.java to 2.66… by moving the launch of the app into a separate method that gets called from main

Package-level Coupling:

1. Afferent coupling is a number of classes from other packages that depend on classes in the measured package. Efferent coupling is a number of classes in other packages that classes in the analyzed package depend on.
2. Main.java.memoranda.util has an afferent coupling of 57
3. Main.java.memoranda.ui has an efferent coupling of 49

Worst Quality:

I think NoteListImpl.java is the worst quality class in the main.java.memoranda package. It ranks:

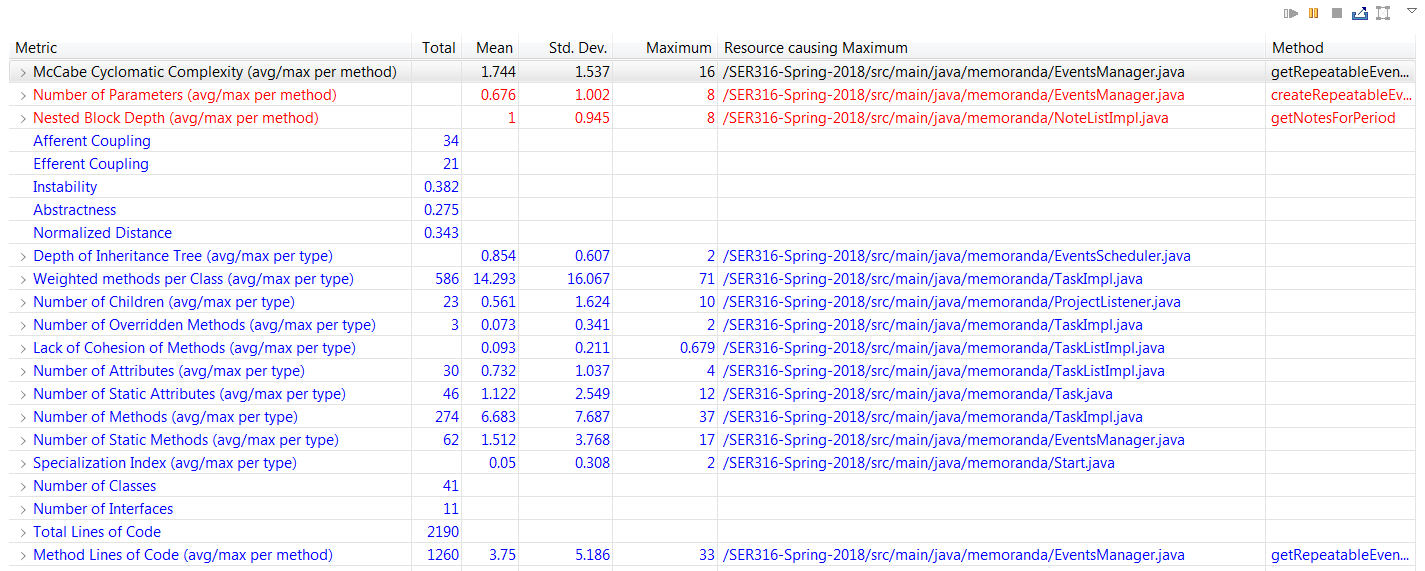
1st highest weighted methods per class

1st highest number of attributes

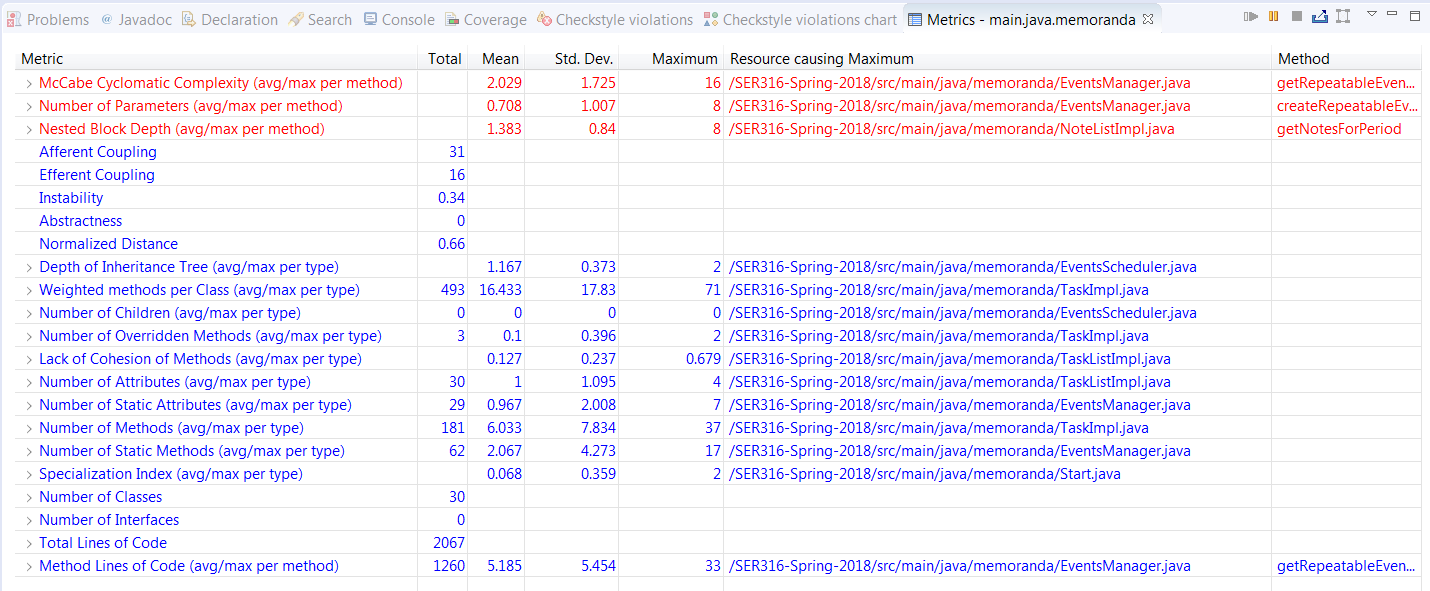
2nd highest Cyclomatic Complexity

3rd highest nested block depth

Task 2:

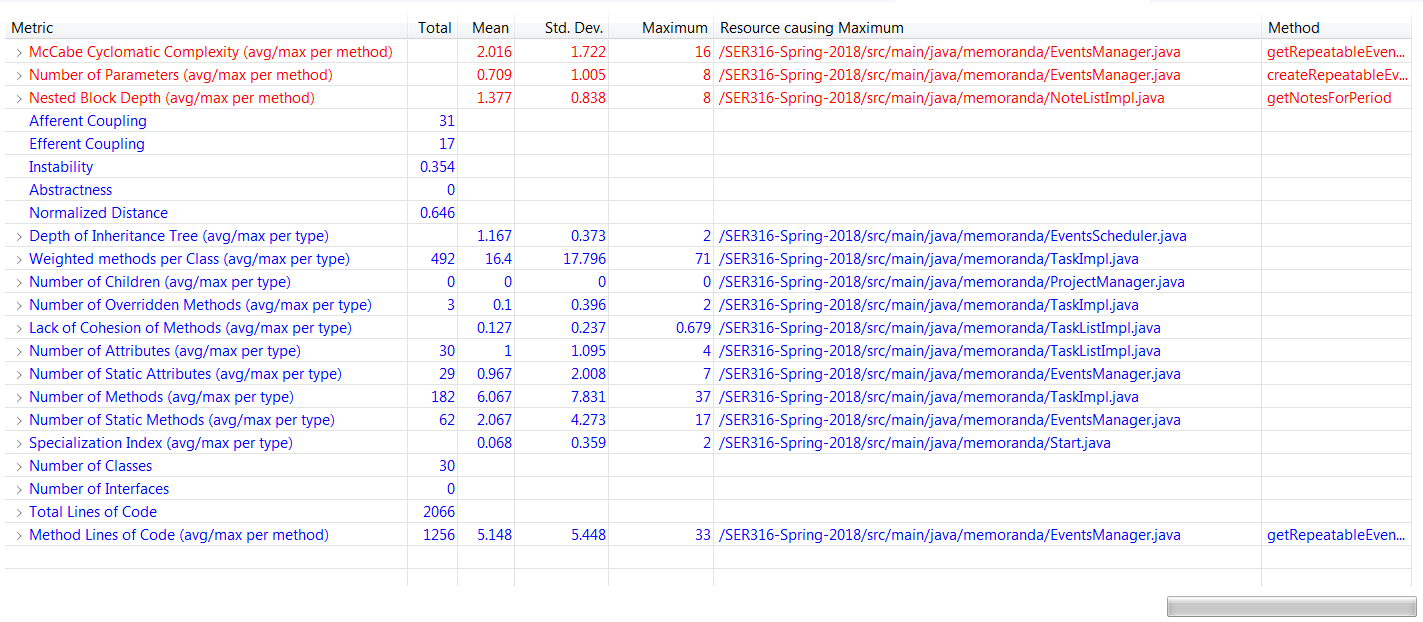


After:



8) Both Afferent and Efferent coupling improved.

Task 3:

1. In ProjectManager.java, the methods getActiveProjects and getActiveProjectsNumber are almost exact duplicates of each other. Since the purpose of getActiveProjectsNumber is to count the number of active projects, and the purpose of getActiveProjects is to return a list of active projects, the duplication is easily refactored by changing getActiveProjectsNumber. I modified this method to call getActiveProjects and then return the size of the list produced.
2. ResourceListImpl.java had a very long message chain when instantiating a Resource. I shortened it by overloading the Resource constructor to accept a Xom element directly, rather than making the calls to fetch the data inside the ResourceListImpl class.
3. 
4. The metrics did not change significantly… Efferent coupling increased by 1, since ResourceListImpl.java now depends on Xom Elements. However Cyclomatic Complexity and Instability both slightly improved.