**Metrics**

**Task 1:**

**Size**1. 2187 LOC in main.java.memoranda)  
2. EventsManager.java, 329  
3. Method 1, Every line that was not an empty (white space) line was counted including lines that only consisted of closing brackets.

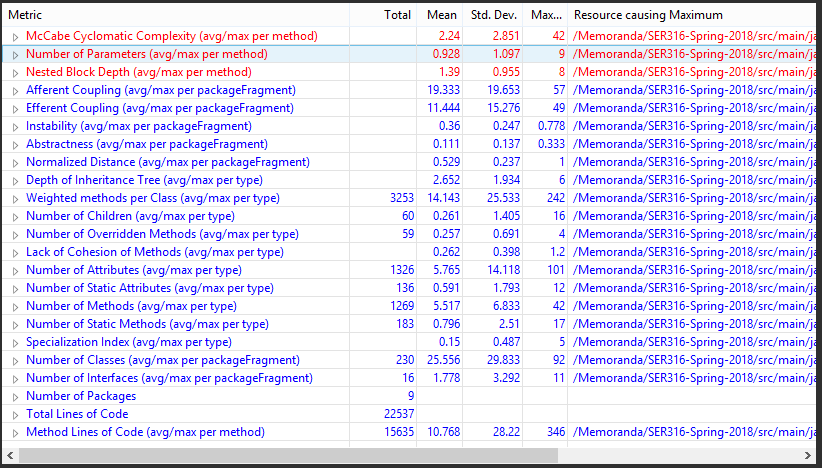
**Cohesion**  
1. LCOM2 = 1 – sum(mA)/(m\*a)  
 m = number of methods in class  
 a = number of variables in class  
 mA = number of methods that access a variable  
 sum(mA) = sum of mA over variables of a class,  
2. TaskListImpl has the highest Cohersion, my theory is due to the constant use of the 3 global variables \_project, \_doc, \_root throughout the methods

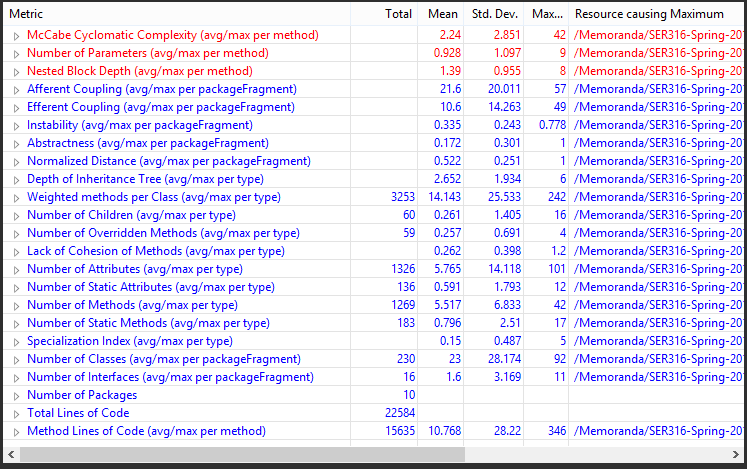
**Complexity**  
1. The complexity of mian.java.memoranda is 584  
2. NoteListImpl.java has the worst average complexity score at 2.485 if we weren’t going with averages then it would be EventsManager that has 1 method with a complex score of 3.353!  
3. To reduce the cyclomatic complexity I went into eventsmanager.java down to the method CreateEvent and commented out an If statement and replaced with a statement that creates a new day object using a calanderdate object. This made the complexity of the createEvent go from 2 to 1 because I got rid of the if statement, if statements are composed of a condition, and conditionals always add to the complexity of a class. (Changes found in eventsmanager.java lines 125 to 129

**Coupling**1 .Afferent couplings are when classes/packages call upon other classes/packages, and Efferent couplings are when classes/packages are called upon by other classes/packages. For example class Bark { Bite b;} class Bite { // .. }, Bark has 1 efferent coupling while bite has 1 afferent coupling  
2. Main.java.memoranda.util has the worst coupling at 57  
3. Main.java.memoranda.ui has the worst coupling at 49

**Worst Quality**  
I think the worst class in main.java.memoranda would have to be EventsManager.java it is listed as the most complex, with over 1000 parameters, and 329 lines of code showing its very long, perhaps this class could have been broken down into smaller simpler classes that handled only what they needed to using design patterns.

**Task 2:**

1. 

**7.**

**8.**  The total lines of code went up which I don’t think really affects the program to much, the afferent coupling went up while the efferent went down

**Task 3.**

1. Main.java.memoranda.TaskImpl.java had 2 methods (getSubtask, hassubtasks) that did the same thing but returned different values (either a Boolean or an object) the rest of the code was duplicate code so I removed the duplicate code and made it its own method below those 2 methods and then made both previous classes call that method and return what they need to.

3.a

