**LINES OF CODE**

Purpose of usage:

The chosen metric is a common basis when it comes to estimate either the size or the amount of time/effort that was required to produce a certain software project. This metric can give us an overall idea of the complexity of some of the classes/methods presented in the project. Usually there are big classes/methods which are commonly undesirable as they often times are quite difficult to read and understand. Despite some of the classes/method being large by default, there are others that can be refactored or even reduced (removing non-functional lines of code). With this metric we can constantly measure the length of certain classes/method and carefully plan countermeasures when certain threshold is met.

Looking at the *class metrics* we can see that the average of the LOC far exceeds the average of the CLOC and JLOC which indicates that there was a moderate amount of documentation produced. After further analyzing the *class metrics* table we can state the number of CLOC and JLOC presented in some of the classes are unbalanced, in example, the “*GanttProject”* class is a very dense and complex class that should’ve been more documented. We have to make sure that there is a balance between the class size and its documentation.

As mentioned before, this metric gives us a rough idea of the overall complexity of a system. If we take 2 projects, let’s call them A and B, respectively, project A being about 10 thousand LOC and project B around 15 thousand LOC, we can’t really say much about the comparison of their complexity. On other hand, if we compare project A to a project C that has 300 thousand LOC, in general, the second one is more complex. Other thing that we need to consider is the level/experience of the software developer, a more experienced developer tends to write less LOC compared to less experienced which can lead to different values shown in the *class metrics* table.