Chidamber-Kemerer Metrics

REPORT

In this report, I will be explaining the CK. CK is used for measuring the design of object-oriented programs. I will focus on the maintainability, understandability, and modifiability of classes. Using the MetricsReloaded plugin, we are presented with many parameters used to study the classes, such as CBO, DIT, LCOM, NOC, RFC, and WMC.

CBO, which stands for coupling between objects, is a count of the number of classes that are coupled to a particular class. In other words, where the methods of one class call the methods or access the variables of the other. In our project, some classes have numbers as high as 135 and as low as 0. A value of 0 indicates that a class has no relationship to any other class. A small value is good and indicates that a class is loosely coupled. A high number may indicate that a class is tightly coupled, this is not good and would complicate modifications. In our project, we have an average of 11.15, and even being a large project, this number seems to be not ideal.

DIT, which stands for depth of inheritance tree, is the number of ancestor classes that can affect a class. It is the maximum length from the node to the root. In our project, we have numbers as high as 7.0 and as low as 0. The higher the number, the lower is the maintainability and readability. In our project, we have an average of 1.62, which seems to be a good value.

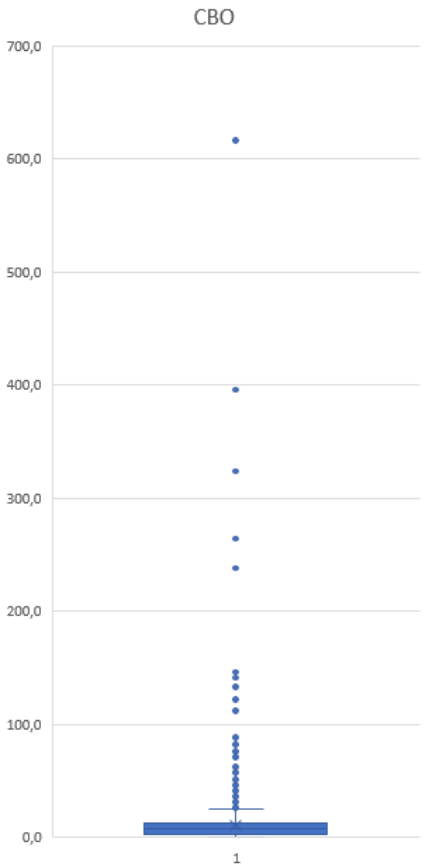
LCOM, which stands for lack of cohesion methods, is used to measure the cohesion of each class. In this project, the numbers get as high as 12 and as low as 0. A high LCOM value could indicate that the design of the class is poor. A suggestion would be to divide the class. The average in this project is 2.23, which seems to be a good value.

NOC, which stands for the number of children, is used to measure the number of classes that derive directly from the current class. In this project, the numbers get as high as 80 and as low as 0. The higher the number of children, the greater the likelihood of improper abstraction of the parent and may indicate a misuse of subclassing. The average is 0.23, which seems to be a good value, where the average range is close to 1.

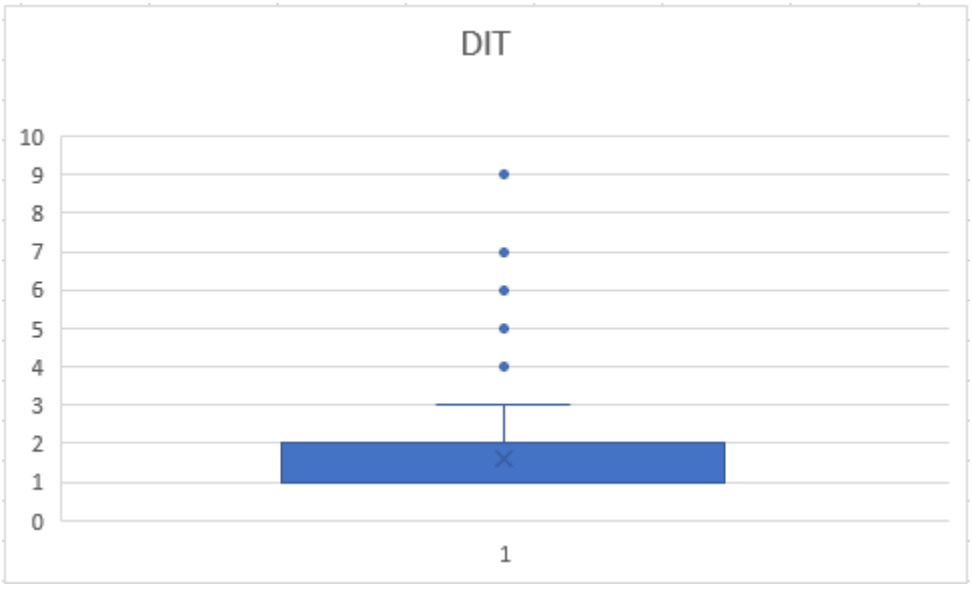
RFC, which stands for the response of class, is the total number of methods that can potentially be executed in response to a message arriving at an object of a class. It is a measure of the potential interaction of a given class with other classes, it allows us to judge the dynamics of the behavior of the corresponding object in the system. This metric characterizes the dynamic component of the external links of classes. The higher the value of RFC, the longer takes to debug and test, and the complexity of the class increases because it is. The average in this project is 23.86 which seems to be in the mean range.

WMC, which stands for weighted method complexity, corresponds to the sum of the complexity of all methods in a class. Higher values for the WMC, mean large complexity. In this project, the total complexity is equal to 20543.0 and the average corresponds to 10.89, which seems to be good values.

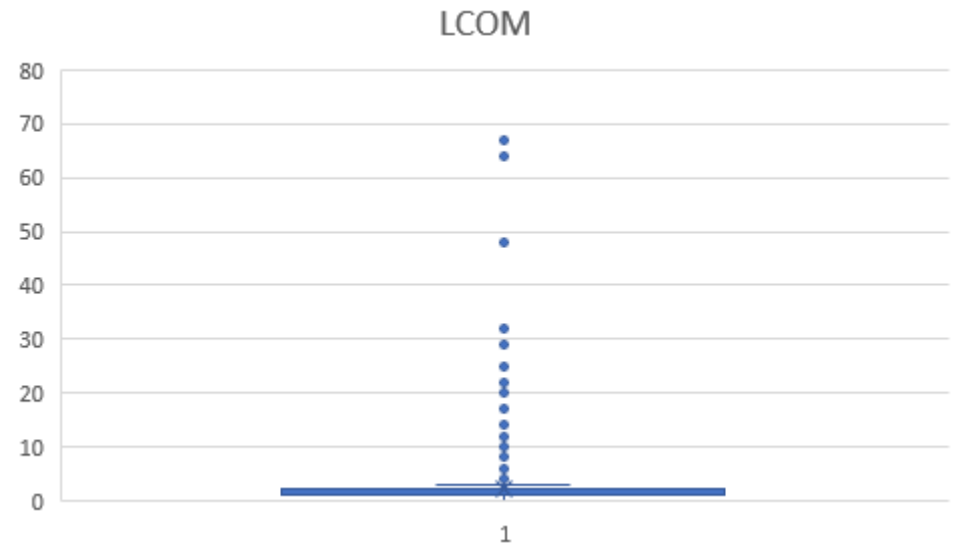
IDENTIFICATION OF POSSIBLE TROUBLE SPOTS IN THE CODEBASE



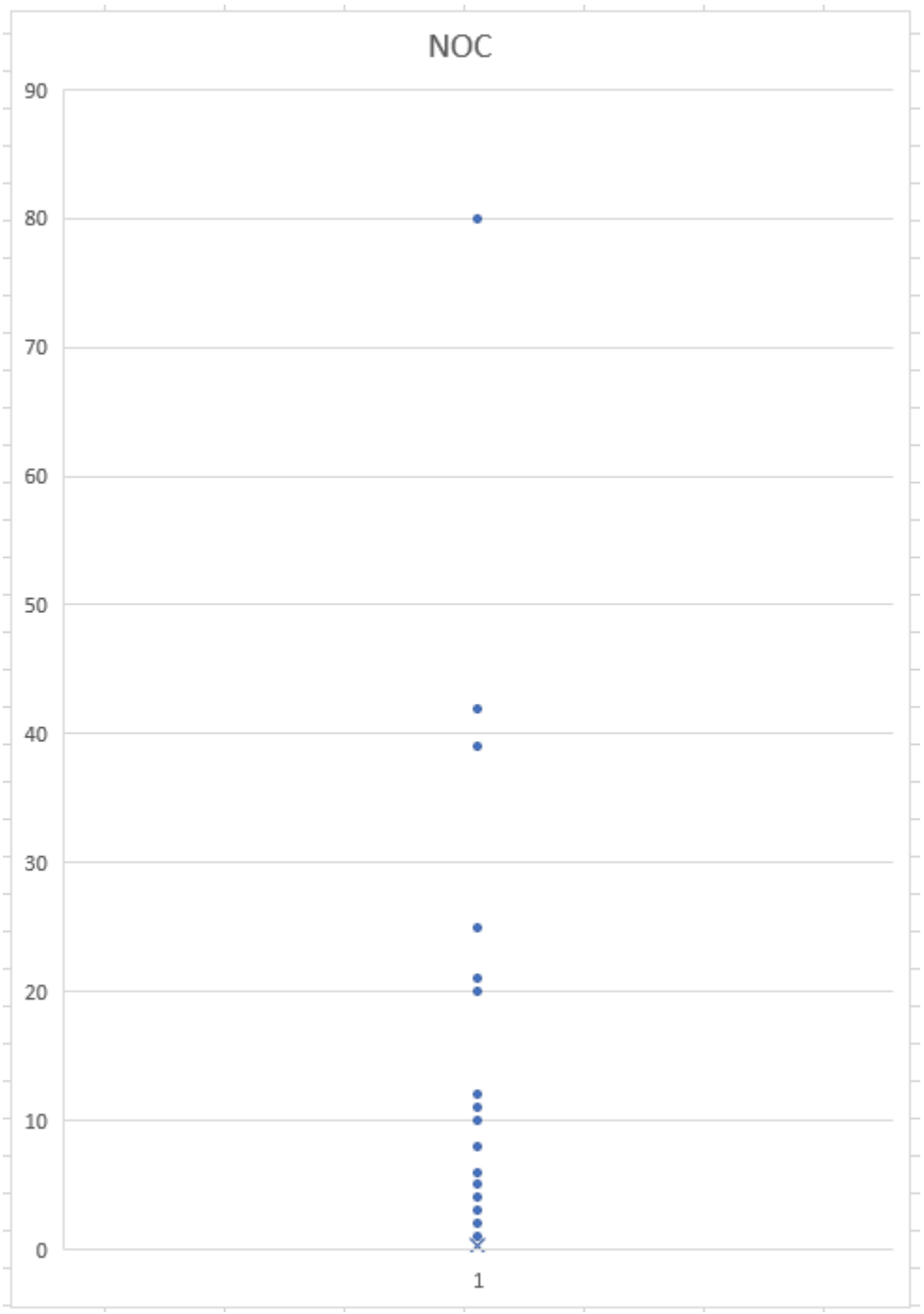
Using the CBO, I could detect two particular situations in the codebase where the value gained is much higher than the obtained average. While the mean is around 11.15, these values were 616 and 396, they correspond to the following classes: BibEntry.java and Localization.java. These situations are just an example. Many other classes have high values as well. This indicates that these classes are very tightly coupled. Not ideal.



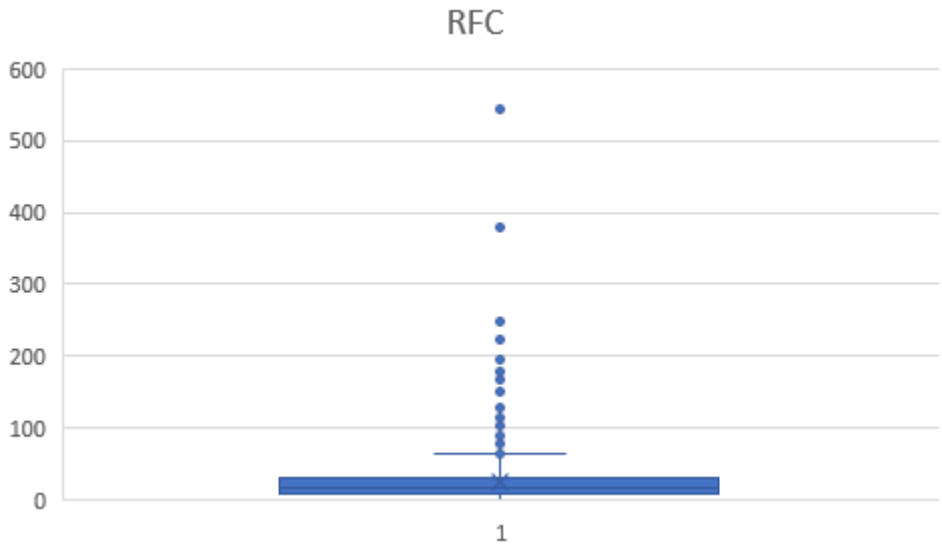
Using DIT, I could conclude that some classes exceeded the average by a lot. Two of those classes (CitationKeyPatterns.java and IconCell.java) had enormous values for the number of ancestors class, meaning that these classes will be arduous to analyze and read. While the average is 1.62, the value for both of these classes is 9.0.



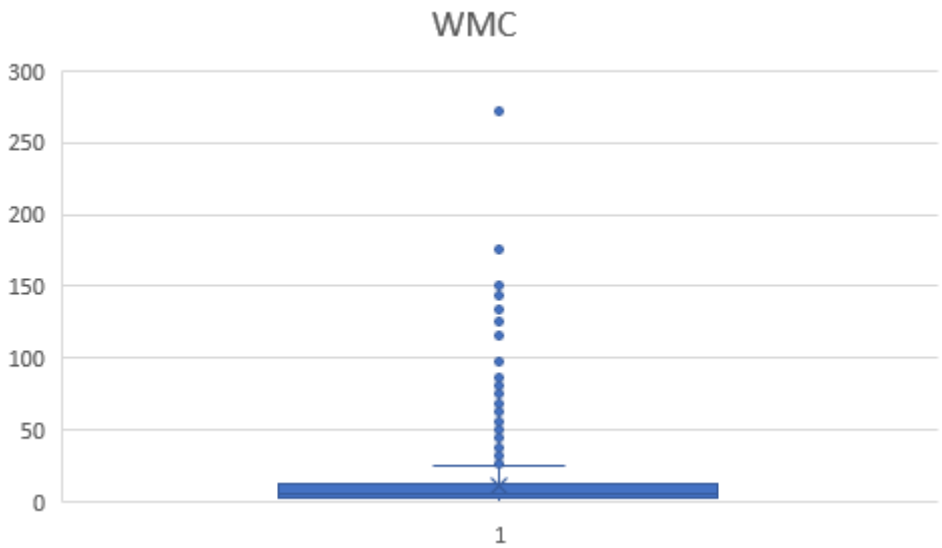
In the LCOM column, the same happens. There are particular situations where there are values considerably large. Two of those situations involve the following classes: AuthorListTest.java and TreeNodeTest.java. While the average is 2.23, the values for these classes are 67 and 64. Although these are Test classes, such values indicate there is space for improvement.



Using NOC, there is one situation in particular that differentiates from the rest. That would be the following class: SimpleCommand.java. The value for this class is 80, while the average is 0.23. An extremely important class in the project as a whole, but with an enormous number of children.



Once again, using RFC, there are two classes whose values are way over the average. These classes are JabRefPreferences.java and JabRefFrame.java. Their values are respectively 543 and 378, while the mean is 23.86. This may indicate that these classes are highly complex.



Finally, using the WMC, there is again a particular situation where the class's value exceeds the average considerably. This class is JabRefPreferences.java. Its value corresponds to 272, while the average is 10.89. Meaning that the sum of the complexity of all methods in this class is immense.In conclusion, this is a very complex class as a whole.

RELATION WITH CODE SMELLS

The classes that stood out negatively were the following: BibEntry.java, Localization.java, CitationKeyPatterns.java, IconCell.java, AuthorListTest.java, TreeNodeTest.java, SimpleCommand.java, JabRefPreferences.java and JabRefFrame.java.

Of the above mentioned, the only class our group found code smells in is JabRefFrame.java.

The two code smells found by the group are Long Method and Large Class.

This class has the following values from the metrics.

CLASS	CBO	DIT	LCOM	NOC	RFC	WMC
JabRefFrame	146	6	2	0	378	115

The Large Class code smell is proved by the values of the CBO and RFC.

The CBO indicates that many classes are coupled to JabRefFrame. As said by my colleague, this class is very diverse in terms of functionalities.

This class, having so many classes coupled proves that this class has a great responsibility to the entire system. The RFC value shows that this class has numerous methods that can potentially be executed in response to a message arriving at an object of this class. Once again demonstrates that this class has an enormous amount of responsibilities.

The Long Method is proved by the value of the WMC.

This large number indicates that the methods of this class are very complex.

This proves that some changes are welcomed, as proposed by my colleague.

Francisco Freire 58667