**Chidamber & Kemerer Metrics Analysis**

This group of metrics is composed by 6 metrics per class: CBO, DIT, LCOM1, NOC, RFC, WMC.

**CBO - Coupling between Object Classes:**

CBO is the number of classes one given class interacts with inside its own methods, be it via calling other classes methods, instance variables, constants, etc. High CBO is an indicator of excessive inter-class dependency, which is an obstacle to modular design and impedes reuse.

In JabRef’s case, most classes stay well within the limit of what is considered a good CBO (CBO <=14), with some notable exceptions like gui.LibraryTab (CBO = 19) and gui.fieldeditors.FieldEditors (CBO = 20).

**DIT - Depth of Inheritance Tree:**

DIT is how deep a class is in its inheritance tree. High DIT indicates greater design complexity, as a class inherits more methods and variables from its predecessors, but also promotes method reuse. Since excessive DIT is found to increase faults, the recommended DIT for any given class is 5 or less.

In JabRef’s case, the highest DIT that can be found is 4 and the most common is 1, so it does well in this metric. It may, however, point to instances of various classes implementing methods already implemented elsewhere. If so, abstract classes should be considered in such cases.

**LCOM4 – Lack of Cohesion of Methods:**

LCOM4 uses “connected components” to determine whether a class is “good” or if it should be split into smaller classes. A connected component is composed by all the methods and class variables related with each other.

If LCOM4 = 1, then the class is fine as is, as all its methods and variables are related.

If LCOM4 >= 2, then the class should be split into several classes, as there is more than one connected component.

If LCOM4 = 0, then the class implements no new methods and is considered a “bad” class.

In JabRef’s case, the majority of classes have LCOM = 1 but a considerable amount fall under the LCOM4 >= 2 category, with notable cases like gui.actions.ActionHelper (LCOM = 8) and gui.StateManager (LCOM = 12) greatly surpassing what is acceptable by this metric. There is also not an insignificant number of classes where LCOM = 0. The best way to improve this metric would be a severe restructuring of the program’s classes, starting with the biggest outliers.

**NOC – Number of Children:**

NOC is a class’s number of immediate children. In contrast to DIT, NOC measures the breadth of a class hierarchy and is in general less preferable to DIT, as it does not promote method reuse through inheritance like DIT does. A high NOC indicates, for example, a high reuse of the base class, which leads to fewer faults, but it can also indicate excessive sub-classing. In addition, changes to classes with high NOC and high WMC may influence a high number of child classes, and as such, may not be considered good design.

In JabRef’s case, most classes have low NOC, with the exception of classes like gui.actions.SimpleCommand (NOC = 80), gui.util.BaseDialog (NOC = 42) and gui.AbstractViewModel (NOC = 21), which provide the base methods and variables for a considerable number of children. This may, once again, be a sign of excessive sub-classing and perhaps some some of these classes would be better off as subclasses, with fewer children each.

**RFC – Response for a Class:**

RFC measures the number of methods of a class that are called from outside itself. High RFC has been found to indicate more faults, since it makes classes more complex and harder to understand.

In JabRef’s case, a big number of classes have RFC in the double digits, with preferences.JabRefPreferences (RFC = 342) and preferences.PreferencesService (RFC = 112) being examples of prime offenders. A possible solution would be the introduction of structural design patterns to allow clearer and more segmented interfacing between classes.

**WMC – Weighted Methods Per Class:**

WMC is the number of methods defined in a class. High WMC means a possible higher impact per change on classes derived from the current class due to inheritance and may indicate that a class could be transformed into various smaller classes. As a rule of thumb, good overall WMC would see at most 10% of classes with WMC >= 24, since it allows for larger classes while also guaranteeing that most of them remain small and concise.

In JabRef’s case, that rule is followed, although some highly notable exceptions like preferences.JabRefPreferences (WMC = 272) and logic.citationkeypattern.BracketedPattern (WMC = 176) should still be considered for sub-classing.