**Chidamber and Kemerer metrics**

The chidamber-kemerer metrics suit are composed by six metrics, these being: coupling between objects (CBO), depth of inheritance tree (DIT), lack of cohesion of methods (LCOM), number of children (NOC), response for class (RFC) and weighted method complexity (WCM).

For this analysis the metrics chosen were the CBO, DIT and WCM metrics.

**CBO metric**

The CBO metric, measures the amount of accesses that one class has with the other classes, counting multiple accesses as one. A higher CBO means there is an excessive coupling between object classes, therefore the class is more dependent leading to a way more harder maintenence of said class.

Classes with the highest CBO

|  |  |  |
| --- | --- | --- |
| **Class** | **CBO** | **Source path** |
| BibEntry | 616 | src/main/java/org/jabref/model/entry/BibEntry |
| Localization | 396 | src/main/java/org/jabref/logic/l10n/Localization |
| StandardField | 324 | src/main/java/org/jabref/model/entry/field/StandardField |
| BibDatabaseContext | 264 | src/main/java/org/jabref/model/database/BibDatabaseContext |
| BibDatabase | 238 | src/main/java/org/jabref/model/database/BibDatabase |
| JabRefFrame | 146 | src/main/java/org/jabref/gui/JabRefFrame |

As it shows, in the table, the BibEntry class has highest CBO in the whole project, which can only mean there is an exceeding coupling between classes, with the same happening to the other six classes.

Relative to code smells, some of the potential code smells that could be found by analysis of this metric would be innapropritate intimacy or feature envy, in the project none of these code smell were identified.

**DIT metric**

The DIT metric measures the maximum inheritance path from the class to the root class. As the inheritance tree deepens, more methods and variables are going to be inherited making the resulting class really complex, although it makes the class more reusable due to method inheritance. It can be concluded that a high DIT number means a higher chance of faults occurring in the classes of the hierarchy being more evident in the middle classes because these are the ones that receive less checks.

|  |  |  |
| --- | --- | --- |
| **Class** | **DIT** | **Source path** |
| HighlightTableRow | 9 | src/main/java/org/jabref/gui/commonfxcontrols/CitationKeyPatternPanel/HighlightTableRow |
| IkonliCell | 9 | src/main/java/org/jabref/gui/groups/GroupDialogView/IkonliCell |
| RadioButtonCell | 9 | src/main/java/org/jabref/gui/util/RadioButtonCell |
| DataBaseChangePane | 7 | src/main/java/org/jabref/gui/collab/DataBaseChangePane |
| EditorTextArea | 7 | src/main/java/org/jabref/gui/fieldeditors/EditorTextArea |
| EditorTextField | 7 | src/main/java/org/jabref/gui/fieldeditors/EditorTextField |

By analysis of the table, the top three classes all share the same DIT number, 9, this indicates that the classes in the hierarchy of these are the most fault-prone, being recommended a review.

Relative to code smell identification, some of the potential code smells that could be found by analisys of this metric would be the refused request and speculative generality code smell. In the project none of them were identified.

**WCM metric**

The WCM metric measures/counts the number of methods in a class. A class with high WMC number tends to be more application specific limiting the possibility of reuse. This metric can also helps us predict the amount of time that the class is going to cost to mantain and develop.

|  |  |  |
| --- | --- | --- |
| **Class** | **WMC** | **Source path** |
| JabRefPreferences | 272 | src/main/java/org/jabref/preferences/JabRefPreferences |
| OOBibStyle | 179 | src/main/java/org/jabref/logic/openoffice/style/OOBibStyle |
| OOBibBase | 176 | src/main/java/org/jabref/gui/openoffice/OOBibBase |
| BracketedPattern | 176 | src/main/java/org/jabref/logic/citationkeyPattern/BracketedPattern |
| MedlineImporter | 153 | src/main/java/org/jabref/logic/importer/fileformat/MedlineImporter |
| BibtexParser | 151 | src/main/java/org/jabref/logic/importer/fileformat/BibtexParser |

Analysing the table, the JabRefPreferences class is the one with biggest number of methods in the whole project, so it would be better to separate several of this methods to smaller classes, with the same being applied to the others 5 classes.

Relative to code smells, some of the potential code smells that would be identified by analysis of this metric the long method code smell, large class. In the project, both of them were identified.