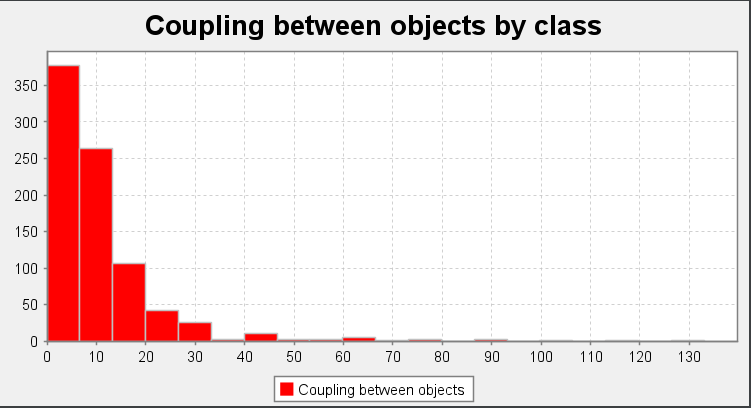
**Chidamber-Kemerer metrics** - Guilherme Franco, nº 60226:

**Coupling between classes:** Calculates the number of classes or interfaces which each class is "coupled" with. A class is declared to be coupled with another if it depends on that class or is depended on by that class. Dependencies due to inheritance are not counted.

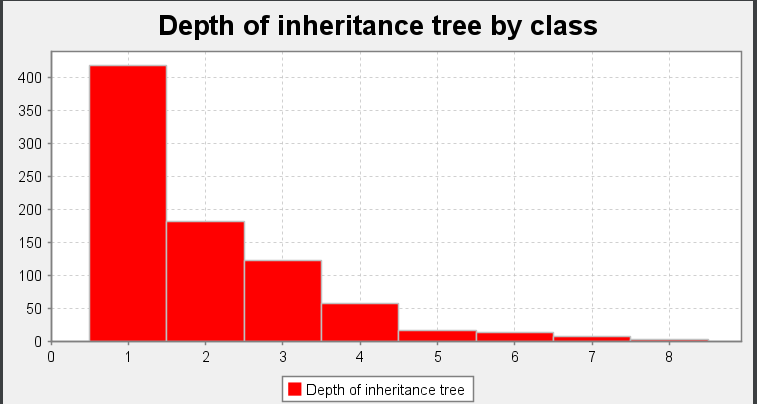
Abbreviation: CBO



Overview: As we can see above there are a couple of classes that have more than 40 dependencies. This shows that the classes depend too much on each other and it’s very likely that this means there are Inappropriate Intimacy code smells in these classes!

**Depth of inheritance tree:** Calculates the depth of the inheritance tree for each class. The depth is calculated as the number of inheritance steps between the class and java.lang.Object.

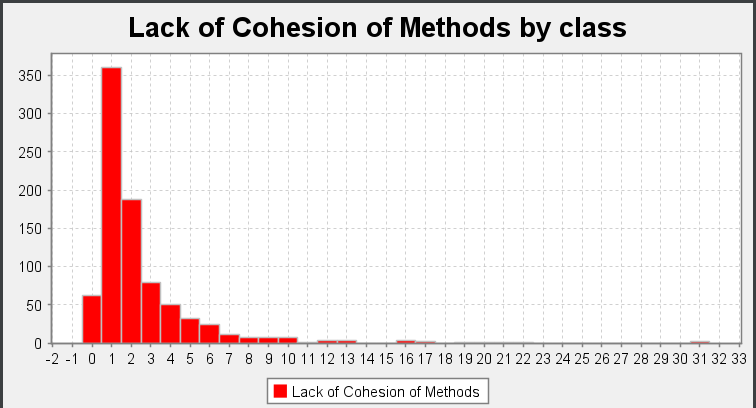
Abbreviation: DIT



Overview: Other than a few classes that have a high amount of DIT score (very specialized classes), which is not very desirable since it creates confusing code, overall the DIT score of most classes shows that there aren’t many troubled areas.

**Lack of Cohesion of Methods:** Calculates on the degree of cohesiveness of a class. We use a variant of the LCOM metric designed by Hitz and Montazeri, which is more appropriate for Java. The metric says that two methods of a class are related if they share a variable use, or one method calls another. The metric is then the count of the number of components of the method relation graph. A value of 1 indicates a highly cohesive class, which can not easily be split into smaller classes. Higher values may indicate that the class may be "doing too much", and should be split. Note that constructors, equals(), hashCode(), toString(), clone(), finalize(), readObject(), and writeObject() methods are not considered, as these scaffolding methods often touch all variables in a class, and would thus result in metrics values indicating more cohesiveness than is actually apparent in the design.

Abbreviation: LCOM



* LCOM=1 indicates a cohesive class, which is the "good" class.
* LCOM>=2 indicates a problem. The class should be split into so many smaller classes.
* LCOM=0 happens when there are no methods in a class. This is also a "bad" class.

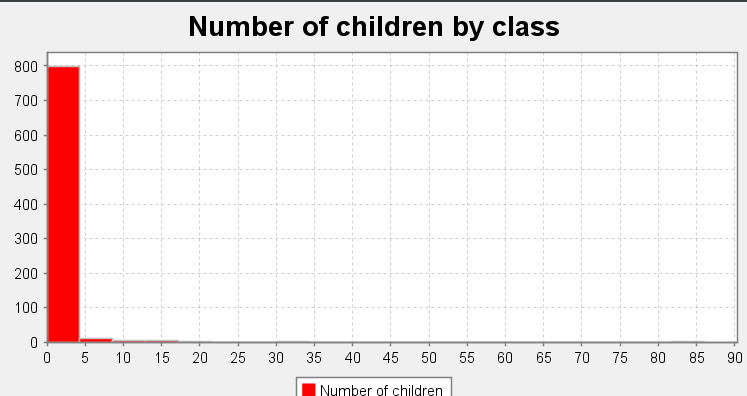
As we can see in the histogram, the majority of classes have a LCOM score of 1 which indicates they are cohesive.

On the other hand there are more than 50 classes with a LCOM score of 0, meaning we have over 50 classes where the only method they have implemented is one of the following: constructors, equals(), hashCode(), toString(), clone(), finalize(), readObject(), or writeObject(); This is obviously not the desired behavior of a class.

Finally there are over 200 classes with a LCOM score of >=2. This means that most of those classes could and should be split into smaller classes so as to avoid Large Class code smells and God Object code smells.

**Number of Children:** Calculates the total number of direct subclasses of each class that occur in the project.

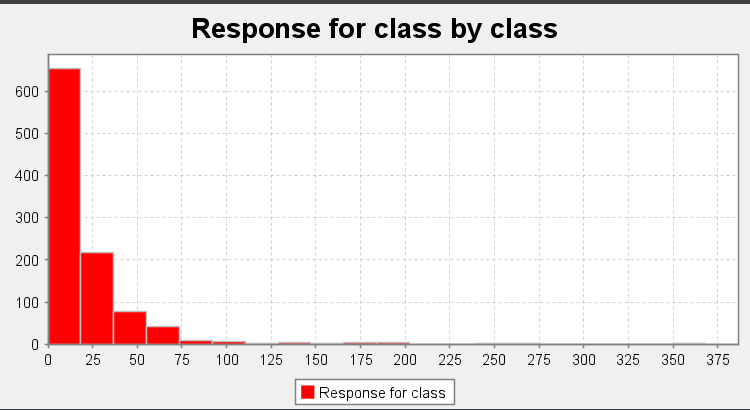
Abbreviation: NOC



**Response for class :** Calculates the Response For Class value of the class. This is defined as the total number of methods that can potentially be executed in response to a message received by an object of a class. In practice, this is the sum of the number of methods and constructors in the class, plus the number of methods and constructors that the class may directly call. Methods from or called by superclasses are not counted.

Classes with high Response For Class have a higher complexity, may be less stable, and require higher amounts of integration testing.

Abbreviation: RFC



Overview: On the histogram above we can see that there are over 100 classes with a RFC score above 50. This isn’t desired, not only because, as mentioned above, classes with a high RFC score have a higher complexity but because there is a very high chance of these classes having “law of Demeter”/”message chains” code smells!

**Weighted Method Complexity:** Calculates the total cyclomatic complexity of the methods in each class.

Abbreviation: WMC

