S-Team’s Renato Viola

**CHIDAMBER AND KEMERER METRICS**

The Chidamber and Kemerer suite of metrics was designed to measure the characteristics of object-oriented systems, by quantifying attributes deemed valuable in analyzing the difficulty and complexity of implementating changes during code maintenance, but also the effects that could derive from such alterations. Every metric is calculated for each class.

The metrics which comprise the studied suite are as follows:

* **Coupling Between Objects (*CBO*)** – Number of classes the particular class is coupled with. A high degree of coupling might prove to be detrimental, as it harms modularity and lessens reuse. With more coupling, other parts of the design become increasingly sensitive to change, thus becoming more prone to faults and difficulting maintenance.
* **Depth of Inheritance Tree (*DIT*)** – Maximum inheritance path from the class to its root. Deep inheritance trees indicate higher design complexity, as the deeper a class is in the hierarchy, the more complex it is likely to be, since it inherits more methods and variables. On a positive note, due to method inheritance, deeper trees promote reuse. However, a higher DIT has been found to increase the density of bugs and decrease quality, as deep class hierarchies are complex to develop.
* **Lack of Cohesion in Methods (*LCOM*)** – Measures how well the methods of a class are related to each other. It’s important to note that this metric has received a fair deal of criticism. In spite of its shortcomings, a higher LCOM generally indicates inappropriate design and high complexity, as well as a higher likelihood of errors, suggesting the class should be split into smaller classes.
* **Number of children (*NOC*)** - Number of immediate sub-classes derived from a class. A higher NOC may indicate: a high reuse of the base class; that the base class may require further testing; an improper abstraction of the parent class; misuse of sub-classing. Possibly due to high reuse, high NOC has been found to indicate fewer faults.
* **Response For a Class (*RFC*)** - Number of different methods that can potentially be executed by the class in response to a message. In other words, a function of the number of methods in the class and the number of different methods called within the methods of the class. A high RFC generally indicates more bugs and less quality, as classes with a high RFC are more complex and tougher to understand. Testing and debugging might prove quite troublesome as well.
* **Weighted Methods per Class (*WMC*)** – Number of methods defined in the class. The more methods a class has, the more likely it is to be more application specific, and thus less reusable. Derived classes also tend to be more impacted by changes in the base class, since they inherit a great number of methods. High WMC generally indicates more faults and could mean that the class could be restructured into smaller classes.

POTENTIAL TROUBLE IN THE CODEBASE

Upon analyzing the values of this set of metrics in the GanttProject codebase, there are several things to note. But the most interesting cases to look at are the ones with the most extreme values.

Some classes do present metrics with interesting values, potentially indicating something worth looking into.

But none were remotely as noteworthy as the *GanttProject* class, with its metrics taking values of {**CBO:** 118; **DIT:** 7; **LCOM:** 9; **NOC:** 0; **RFC:** 365; **WMC:** 156}.

As we can see, this class has very high CBO, RFC and WMC, which indicates that it is very complex, lengthy and heavily dependent on other classes.

This doesn’t immediately equate to faulty code though, as it presents a fairly low LCOM value, suggesting that its code is rather coherent. The high value for DIT does imply high complexity, but it also indicates reusability.

It’s also relevant to note that this class is used as sort of system class, which is inherently bound to make it somewhat more complex and larger than other classes.

At a first glance, it wouldn’t seem hard to infer that this class is very likely to be problematic. But with further context, we really can’t be so sure.

RELATED CODE SMELLS

The values of these metrics are able to reflect the existence of certain smells within the codebase.

2 of these smells were identified by the S-Team during phase 1 of the project. These are as follows:

* Message Chain – The code has long message chains, where you call to get an object back, where you call another object back, call an object on it again, and so on… This sort of programming practice affects the value of the CBO and RFC metrics directly.
* Long parameters list – A method who receives many arguments can be difficult to use and understand. The more arguments a method receives, the more methods it invokes, thus increasing the value of the CBO and RFC metrics.

BIBLIOGRAPHY:

[*https://www.aivosto.com/project/help/pm-oo-ck.html*](https://www.aivosto.com/project/help/pm-oo-ck.html)

[*https://www.hindawi.com/journals/isrn/2013/276105/*](https://www.hindawi.com/journals/isrn/2013/276105/)

[*https://www.aivosto.com/project/help/pm-oo-cohesion.html#LCOM1*](https://www.aivosto.com/project/help/pm-oo-cohesion.html#LCOM1)