**Complexity Metrics**

Martim Costa 64901

Brief Introduction

In this report I will analyze the Complexity Metrics of the GanttProject, in methods, classes, packages, modules and in the entire project. The metrics are:

- Cogc: Cognitive Complexity

- ev(G): Essential Cyclomatic Complexity

- iv(G): Design Complexity

- v(G): Cyclomatic Complexity

- OCavg: Average Operation Complexity

- OCmax: Maximum Operation Complexity

- WMC - Weighted Method Complexity

- v(G)avg - Average Cyclomatic Complexity

- v(G)tot - Total Cyclomatic Complexity

I will give more importance to the **Cognitive Complexity** and **Cyclomatic Complexity** since the others are strongly related to these two.

Cognitive, Essential Cyclomatic, Design and Cyclomatic Complexities

- Cognitive Complexity (Cogc) is a measure of how difficult a unit of code is to intuitively understand or in other words tells you how difficult your code will be to read and understand. At a method level, 15 for a Cognitive Complexity value is a recommended maximum.

A method's cognitive complexity is based on a few simple rules:

* Code is not considered more complex when it uses shorthand that the language provides for collapsing multiple statements into one.
* Code is considered more complex for each "break in the linear flow of the code". For example loops, switch or case statements, conditionals and others.
* Code is considered more complex when "flow breaking structures are nested". For example conditionals, loops and try/catch blocks.

All of this increase the value of **Cognitive Complexity**.

- Cyclomatic Complexity (v(G)) effectively measures the number of possible independent paths through a method or function. This tells us how complex the method is to test and debug for example. Higher values of **Cyclomatic** **Complexity** result in the need for a higher number of test cases to comprehensively test a part of code or a function, and if you add that to a more complex part of code, it increases the chances of you adding a defect to the code. Some examples of what increases the **Cyclomatic** **Complexity** are if/else statements, switch case statements ‘try-catch’ statements among others. Usually this type of complexity is divided by 4 levels:

* 1 - 10 Simple procedure, little risk
* 11 - 20 More complex, moderate risk
* 21 - 50 Complex, high risk
* > 50 Untestable code, very high risk

The risk is related to how hard it is to test it and how high are the chances of committing a defect.

All the methods start with value 1.

- Essential Cyclomatic Complexity (ev(G)) tells how much complexity is left once we have removed the well-structured complexity**. Essential Cyclomatic Complexity** and **Cyclomatic Complexity** are related.

- Design Complexity (iv(G)) is related to how interlinked a method’s control flow is with calls to other methods. It also represents the minimal number of tests necessary to exercise the integration of the method with the methods it calls.



**Table 1-Methods**

As you can see, at the level of methods (more than 6500) the average for all complexities is under 2, which is a very good average, way below the values ​​considered minimum.

Below are the tables that are related to Packages, Modules and the whole Project. As you can see the values of **Cyclomatic Complexity** ​​ referred to the methods are represented in the other tables, which have similar values. **v(G)avg** is **Average Cyclomatic Complexity** and **v(G)tot** is the **Total Cyclomatic Complexity**.



**Table 2- Packages**

 

**Table 3 – Modules Table 4- Whole Project**

Average Operation, Maximum Operation, Weighted Method Complexities

- Average Operation Complexity (OCavg) is the Average Cyclomatic Complexity of all non-abstract methods in each class. Inherited methods are not counted.

- Maximum Operation Complexity (OCmax) is the Maximum Cyclomatic Complexity of all non-abstract methods in each class. Inherited methods are not counted.

- Weighted Method Complexity (WMC) is the total cyclomatic complexity of the methods of each class.

Uma imagem com mesa

Descrição gerada automaticamente

**Table 5 – Classes**

As this table shows, the Average of the Average **Cyclomatic Complexity** of the methods of each class don’t differ much to those shown above in **Table 1**. Even the Average of the Maximum **Cyclomatic Complexity** value is around 3 so even the methods with higher values have values that are considered in the acceptable range.

Relation between Code Smells and Complexity Metrics

Of course Complexity Metrics are related directly with code smells.

When a method has high values of **Cognitive Complexity** it means that it is a method that is hard to read and understand. That is a code smell because, maybe for the method’s creator it is easy to read it, it’s hard for others to understand it. And this increases the chances of others that want to work in that part of the code to commit defects and errors. This also makes the code harder to be expandable which is a very important part of a good coding practice.

Same problem with **Cyclomatic Complexity**. Hard testing code makes it harder to work on, and increases the chances of defects and errors. For example, in a code smell detected by me( Martim Costa 64901) we can see a **Huge function** with a lot of if/else statements and try/catch statements which means that there are a high number of possible independent paths through the function. To test that would take a lot of time and patience. Here is a print of part of that function.

Uma imagem com texto

Descrição gerada automaticamente

I also want to mention that, although that's not always the case, **Cyclomatic** **Complexity** can be one of the factors driving up **Cognitive** **Complexity** and vice versa.

Conclusion

Analyzing and according to this data, I can conclude that this Project in general in terms of understanding and reading it is easy to do so. I can also say that about the difficulty to test this Project the values of **Cyclomatic Complexity** are below the value that are considered acceptable (usually 10 is the maximum value to be considered that).

Of course some improvements can be done, some methods have high values of **Cognitive Complexity** and **Cyclomatic Complexity**. Some ways to reduce **Cognitive Complexity** are :

-Avoiding multiple if-else or nested if-else statements.

-Move repeated Code/nested if else to a separate function.

-Reduce the number of parameters of the method. It is always good to have max 2–3 parameters. If it exceeds you can wrap all of them into a class and pass the object.

About **Cyclomatic Complexity**:

- Use small methods. Try reusing code wherever possible and create smaller methods which accomplish specific tasks.

- Reduce if/else statements.