**MOOD METRIC SET**

**Explanation of the collected metrics**

The aim of this report is to produce a metrics-based overview of the codebase. To accomplish this, I selected metrics sets from those available in the MetricsReloaded plugin for IntelliJ IDEA. MetricsReloaded offers many metrics sets and I chose **MOOD Metrics**. The MOOD metric set is used to measure the properties of the system in which the design of the system is according to the concepts of Object-Oriented Design that is Encapsulation, Coupling, Inheritance, Information Hiding, and Polymorphism. The set contains six main metrics to measure the design of the system. The information which is analysed by this metric set are attribute hiding factor (AHF), attribute inheritance factor (AIF), coupling factor (CF), Method Hiding Factor (MHF), Method Inheritance Factor (MIF), and Polymorphism Factor (PM).

Diagram

Description automatically generated

**Metrics**

**Method Hiding Factor (MHF) and Attribute Hiding Factor (AHF)**

To measure how the attributes and methods of one class are encapsulated, Method and Attribute hiding factors are being used. MHF and AHF show the average amount of how the members of a class are hidden in the system. MHF and AHF are 100% of all the members hidden, but in the real world, it is next to impossible because a class should communicate with other classes to provide different functionality of the system. It is not possible if the members are hidden.

To better understand how the percentage is made, let´s assume:

C = number of Classes

MV = number of other classes where a method is visible

AV = number of other classes where an attribute is visible

The MHF result (in percentage) is made as follows:

(Visible Methods) = (sum (MV) / (C–1)) / Number of Methods

MHF = 1 – (Visible Methods) 🡪 to get the hiding methods

The AHF result (in percentage) is made as follows:

(Visible Attributes) = (sum (MV) / (C–1)) / Number of Methods

AHF = 1 – (Visible Attributes) 🡪 to get the hiding attributes

If we have low MHF, then the methods are visible to almost every class. The methods are unprotected and can be subjected to changes by any class. The methods have a high tendency of reusability. In another case, if we have high MHF, then methods are not visible to many classes, and they cannot understand the working of it. These methods can be used only by the class containing it and it has less tendency of reusability. The ideal range should be around 8% to 25%. In this project, the MHF has a value of 36.93%, so it is way above the ideal range. In the case of AHF, the attributes of a class should be hidden from other classes so 100% is the ideal value of AHF. Generally, a high value of AHF is advised. In this project, the AHF has a percentage of 78.33%, it is not near 100%, but it is a high value.

**Method Inheritance Factor (MIF) and Attribute Inheritance Factor (AIF)**

In Inheritance, the child or subclass inherits the properties (Attribute and Methods) of the parent or superclass. The extent to which these methods and attributes are inherited is defined by Method Inheritance Factor (MIF) and Attribute Inheritance Factor (AIF).

To better understand how the percentage is made, let´s assume:

TM = total methods available in classes

TA = total attributes available in classes

The MIF result (in percentage) is made as follows:

MIF = inherited methods / TM

The AIF result (in percentage) is made as follows:

AIF = inherited attributes / TA

A child class that inherits many methods and attributes from its parent class contains a large value of MIF and AIF. As anything in a very small amount of very high amounts is harmful, the same case is with MIF and AIF. These values should be in a reasonable range. Generally, the range of MIF is between 20% to 80%. In this project, the MIF has a percentage of 18.28%, so a little under the ideal range. However, the ideal range of the AIF is between 0% and 48%. About this project, the AIF has a value of 23.20%, so inside the ideal range. The low value of AIF shows that the class should not inherit the attributes but rather the attributes should be private to the class. The child class can make some changes (for example, override) in the inherited methods.

### **Polymorphism Factor (PF)**

To measure the degree or extent of method overriding by the child class from the parent or superclass, Polymorphism Factor (PF) is used. In polymorphism, the child class can implement the method in a different way. The same method can be implemented in different ways in the child and parent classes.

The PF result (in percentage) is made as follows:

PF = actual number of methods overrides / max number of total method overrides.

To keep the code clean and clear and to provide the high quality we can use high PF, but it increases the complexity of the system. In this project, the PF value is 49.59%. Polymorphism factor is integrally associated with method overriding.

### **Coupling Factor (CF)**

### If two or more clauses are related to each other by means of inheritance or aggregation or association then, in that case, these classes are said to be coupled. Many functionalities of the system can be done with the help of coupled classes. It is not advisable to have too many independent classes. The high value of CF shows that the classes of the system are more inter-connected and inter-dependent. This leads to the problem that sometimes it is very hard to change or repair the system in case of any bugs or problems because the functionality in which the bug is, could be implemented by more than two classes and we must make changes in all the related classes.

The CF result (in percentage) is made as follows:

CF = Actual coupling between different classes / maximum possible coupling that can happen in the system

If a class can access the method and attributes of the second class, then it is said that the first class is coupled with the second class. Generally, coupling happening due to inheritance is not considered while calculating the CF. The high value of coupling in the system leads to larger complexity. So, it is advisable to have a high value of cohesion and a low value of coupling. In this project, the CF has a value of 0.68%, which means the project has a good design architecture because contains less amount of coupling.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| FACTOR | IDEAL  MINIMUM (%) | IDEAL  MAXIMUM (%) | MINIMUM  TOLERANCE (%) | MAXIMUM  TOLERANCE (%) | PROJECT (%) |
| MHF | 12 | 22 | 9 | 36.9 | **36.93** |
| AHF | 75 | 100 | 67.7 | 100 | **78.33** |
| MIF | 66 | 78 | 61 | 84 | **18.28** |
| AIF | 53 | 66 | 37 | 75 | **23.20** |
| PF | 2 | 9 | 1 | 15 | **49.59** |
| CF | 0 | 11 | 0 | 24 | **0.68** |

With the help of the values, the conclusion is that the project is mostly out of the range accepted by the MOOD metrics set, which means the code and the way the project was made have problems, as detected by these metrics. According to the MOOD metrics, the overall quality of this objected-oriented project could be better.

**How these metrics relate to the identified code smells**

Code Metrics is a tool that analyzes our project, measures the complexity, and provides us better insight into the code. The advantages of Code Metrics are identifying code smells (it means identifying “the design flaws or bad practices, which might require attention. Some of the common code smells are Long Method, Duplicate Code, Large Class, and Dead Code), identifying the complexity and maintainability of our code (it will give an insight into our code maintainability and complexity) and increasing our Code Review efficiency.