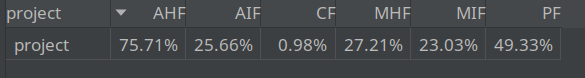
MOOD Metrics

This metrics are used to measure the quality of some characteristics of OO(object oriented) programs, and this are: AHF, AIF, CF, MHF, MIF and PF.  


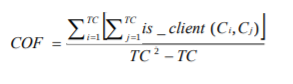
* **AHF(Atribute Hiding Factor)**: All attributes should be hidden, only being accessed by the corresponding class methods. Sum of all hidden attributes (in all classes) divided by the sum of visible attributes and hidden attributes(in all classes). Should be 100%. In these case it is 75.71%.

Importance: Low(1)

* **AIF(Attribute Inheritance Factor)**: Should be used but not too extensively. Sum of the inherited attributes divided by the sum of defined attributes and inherited attributes. In this case it was 25.66%.

Importance: Low(1)

* **CF(Coupling Factor)**: Measure of coupling between classes. It is desirable that the classes communicate with as few other classes as possible, so the lower the better. In this project it was 0.98%.



TC = Total number of Classes.

is\_client(Cc, Cs) = if((Cc => Cs) and (Cc != Cs)) return 1 else return 0

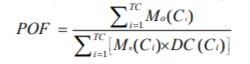
Importance: High(3)

* **MHF(Method Hiding Factor)**: All methods should be hidden, only being accessed by the corresponding classes. Sum of all hidden methods (in all classes) divided by the sum of visible methods and hidden methods(in all classes)(should be 100%). In these case it is 27.21%.

Importance: Medium(2)

* **MIF(Method Inheritance Factor)**: Should be used but not too extensively. Sum of the inherited methods divided by the sum of defined methods and inherited methods. In this case it was 23.03%.

Importance: Medium(2)

* **PF(Polymorphism Factor)**: Represents the number of possible different polymorphic situations. Should be used but not too extensively. In this project it was 49.33%.

Mo = methods that override other methods

Mn = new methods

DC = classes that derivate from another class

TC = total number of classes

Importance: Medium(2)

**Factor Evaluation:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Factor** | **MIN** | **MAX** | **MIN Tolerance** | **MAX Tolerance** |
| **AHF** | 75.2% | 100% | 67.7% | 100% |
| **AIF** | 52.7% | 66.3% | 37.4% | 75.7% |
| **CF** | 0% | 11.2% | 0% | 24.3% |
| **MHF** | 12.7% | 21.8% | 9.5% | 36.9% |
| **MIF** | 66.4% | 78.5% | 60.9% | 84.4% |
| **PF** | 2.7% | 9.6% | 1.7% | 15.1% |

* **AHF:** 75.71% is a good value.
* **AIF:** 25.66% is a really bad value, even with the minimum tolerance the value should be higher, so the project should have a higher attribute inheritance factor, but not too high.
* **CF:** 0.98% is perfect, the lower the better.
* **MHF:** This value should be lower, it’s a little bit too high. To fix we should hide more methods, to do this we could make some of them private.
* **MIF:** 23.03% is a really bad value, even with the minimum tolerance the value should be way higher, so the project should have a higher method inheritance factor.
* **PF:** 49.33% is a really bad factor, even with the maximum tolerance the value should be lower. To fix this we could create less superclasses and subclasses.

**Conclusion:**

We can conclude with the evaluation that we should really fix the AIF, MIF and PF factors, the project has too much polymorphism but the inheritance(methods and attributes) is too low, this could be due the **Large Class** code smells that we found where the programmers mostly used a large class instead of using subclasses, making the AIF/MIF factors too small. This could be due other code smells that we didn’t identify, the **Speculative Generality** code smell, where the programmers added more than the necessary subclasses thinking on the future and ended not using them as much as they tought they would, or the **Switch Statements** code smell, where they could have used the subclasses more instead of the switch statements. The MHF factor it’s a little too high but not too overwhelming for now. The AHF, CF have both good values.