Mood Metrics

Bernardo Carvalho 60012

Metrics

* Method Hiding Factor (MHF)
* Attribute Hiding Factor (AHF)
* Method Inheritance Factor (MIF)
* Attribute Inheritance Factor (AIF)
* Polymorphism Factor (PF)
* Coupling Factor (CF)

Introduction

In this report I’ll be studying the data collected from the MOOD metrics regarding our project to see if there are any anomalies or if everything is between the expected ranges for each category.

Method Hiding Factor and

Method hiding factor measures how methods are encapsulated in a class. Visibility is counted in respect to other classes. MHF represents the average amount of hiding among all classes in the system. A private method is fully hidden.

MHF = 1 − MethodsVisible

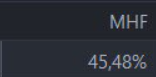
MethodsVisible = sum(MV) / (C − 1) / Number of methods

MV = number of other classes where method is visible

C = number of classes

For each method, MV is counted.

If all methods are private, MHF=100% . If all methods are public, MHF=0%



A low MHF indicates insufficiently abstracted implementation. A large proportion of methods are unprotected and the probability of errors is high. To contrast, a high MHF indicates very little functionality. It may also indicate that the design includes a high proportion of specialized methods that are not available for reuse.

It is considered that a MHF between 10% and 25% is Ideal so as we can see ours is a considerable amount over those values which may indicate less functionality than we would want.

Attribute Hiding Factor.

Attribute hiding factor is the same as MHF, but with attributes, and so, it measures how variables are encapsulated in a class. Visibility is also counted in respect to other classes. AHF represents the average amount of hiding among all classes in the system. A private attribute is fully hidden.

AHF = 1 − AttributesVisible

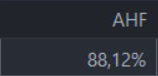
AttributesVisible = sum(AV) / (C − 1) / Number of attributes

AV = number of other classes where attribute is visible

C = number of classes

For each attribute, AV is counted.

If all attributes are private, AHF=100%. If all attributes are public, AHF=0%.



Ideally, most attributes should be hidden, and thus AHF should always be very high, nothing under 75%. Low values of AHF should trigger attention.

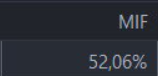
As we can see, although our AHF isn’t 100%, it is quite high and thus it looks like everything is in order.

Method Inheritance Factor

Once again both MIF and AIF work in similar ways, but (again) one with methods and the other with attributes. For this reason, both will have similar explanations.

A class that inherits lots of methods from its ancestor classes contributes to a high MIF. A child class that redefines its ancestors' methods and adds new ones contributes to a lower MIF. An independent class that does not inherit and has no children contributes to a lower MIF.

MIF = inherited methods / total methods available in classes



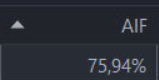
It is said that MIF should be in a reasonable range, not too low nor too high. Too high of a value indicates either superfluous inheritance or too wide member scopes. A low value indicates lack of inheritance or heavy use of Overrides/Shadows.

As we can observe, our MIF is within a good balance between too much inheritance and too little.

Attribute Inheritance Factor

A class that inherits lots of attributes from its ancestor classes contributes to a high AIF. A child class that redefines its ancestors' attributes and adds new ones contributes to a lower AIF. An independent class that does not inherit and has no children contributes to a lower AIF.

AIF = inherited attributes / total attributes available in classes

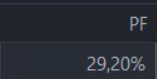


Although AIF behaves very similarly to MIF, it is believed that its range should be significantly lower than MIF, most variables should be declared private and so we can conclude that ours is quite a bit higher than it should be.

Polymorphism Factor

Polymorphism Factor measures the degree of method overriding in the class inheritance tree. It equals the number of actual method overrides divided by the maximum number of possible method overrides. Simply put, PF is the "Overrides factor". The more you use the Overrides keyword, the higher PF.

PF = overrides / sum for each class(new methods \* descendants)



There is some conflict between the accepted range for PF, but either way this measurement usually sits at low percentages close to 10% or lower so it is safe to assume that ours is, once again, above usual.

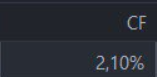
Coupling Factor

Coupling Factor measures the actual couplings among classes in relation to the maximum number of possible couplings.

CF = Actual couplings / Maximum possible couplings

Class A is *coupled to* class B if A calls methods or accesses variables of B.

If no classes are coupled, CF = 0%. If all classes are coupled to all other classes, CF=100%.



An acceptable range for CF is usually between 0 to 11% so ours is well within limits. This means the project does not couple too many classes without reason.

Conclusion

According to MOOD metrics our project has a bit too many overrides and inherited attributes. Furthermore it also has too many private methods leading to less functionality. Although these bad qualities have no specific relation to the code smells we found, they may be harmful in other ways.

The positive side is that the rest of the measurements seem to be within according ranges. The project hides most attributes, as it should, inherits a good balance of methods and couples a good percentage of classes.

Bibliography

* Mood Factors to Assess a Java Program - javatpoint
* MOOD and MOOD2 metrics - Aivosto