A picture containing text, clipart

Description automatically generated

**Engenharia de Software**

**2023/2024**

Code Metrics

**“MOOD Metrics”**

Diogo Lemos, 56837

Index

Understanding the MOOD Metrics………………………………………………………………………………………….3,4,5

MOOD results…………………………………………………………………………………………………………………………………6

References………………………………………………………………………………………………………………………………………7

Understanding the MOOD Metrics

The **MOOD (Metrics for Object Oriented Design)** metrics set includes 6 metrics:

1. **MHF (Method Hiding Factor) –** MHF measures how variables are encapsulated in a class. It represents the average amount of hiding among all classes in the system.

A private method is fully hidden. Basically, hiding decreases in the following order: Protected, Friend, Protected Friend, Public.

It is calculated using the following formula:

**MHF = 1 – MethodsVisible**

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

**MV = number of other classes where method is visible**

**C = number of classes**

If all methods are private, then MHF=100%. That would imply very little functionality as methods wouldn’t be able to be reused by other classes.

If all methods are public, MHF=0%. That would imply insufficiently abstracted implementation. A large proportion of methods would be unprotected and the probability of errors high.

So, the ideal value is somewhere in between.

1. **AHF (Attribute Hiding Factor) –** AHF works similarly to MHF, but it is applied to attributes instead.

It is calculated using a similar formula as well:

**AHF = 1 – AttributesVisible**

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

**AV = number of other classes where attribute is visible**

**C = number of classes**

If all attributes are private, then AHF=100% and that would be the ideal value.

If all attributes are public, AHF=0%. Very low values of AHF should trigger attention.

Understanding the MOOD Metrics

1. **MIF (Method Inheritance Factor) –** MIF measures the average between inherited and total methods in classes.

It is calculated using the following formula:

**MIF = inherited methods / total methods available in classes**

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.

1. **AIF (Attribute Inheritance Factor) –** AIF works similarly to MIF, but it is applied to attributes instead.

It is calculated using a similar formula as well:

**AIF = inherited attributes / total attributes available in classes**

The ideal value for AIF would be 0% since all attributes should be private.

1. **PF (Polymorphism Factor) –** PF 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.

It is calculated using the following formula:

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

PF varies between 0% and 100%. As mentioned above, when PF=100%, all methods are overridden in all derived classes. A PF value of 0% may indicate one of the following cases:

* project uses no classes or inheritance.
* project uses no polymorphism.
* full class hierarchies have not been analyzed (child Overrides unknown)

Understanding the MOOD Metrics

1. **CF (Coupling Factor) -** measures the actual couplings among classes in relation to the maximum number of possible couplings.

It is calculated using the following formula:

**CF = Actual couplings / Maximum possible couplings**

Class A is coupled to class B if A calls methods or accesses variables of B.  
In turn, B is coupled to A only if B calls methods or accesses variables of A. B is not coupled to A if there is no call/access from B to A.

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

Couplings due to the use of the Inherits statement are not included in CF.

Coupling relations increase complexity, reduce encapsulation and potential reuse, and limit understandability and maintainability. Very high values of CF should be avoided.

However, classes must cooperate somehow, and CF is expected to be lower bounded.

MOOD Results

Uma imagem com captura de ecrã, texto, file

Descrição gerada automaticamente

Figure 1. Project Metrics using IntelliJ’s extension - MetricsReloaded.

Analyzing the results, we can assume certain aspects about the project’s code:

* **MHF = 26,03% -** The value seems reasonable. We can assume that most methods must be private or maintain an acceptable degree of encapsulation.
* **AHF = 69,05% -** The target value is 0%, so the presented result is quite above the intended. It is alarming that most attributes are visible to other classes other than their own. Code smells associated with this behavior might be *Feature Envy* or *Inappropriate Intimacy.*
* **MIF = 72,78% -** The value is acceptable, although quite high. We can assume that many methods are being inherited by subclasses, which will raise the project’s overall complexity.
* **AIF = 46,46% -** Although the value is acceptable it is still quite high. We can take into consideration that while ideally having AHF at 0%, AIF should also tend to lower values to keep encapsulation in check.
* **PF = 7,06% -** We can expect a decent usage of overridden methods. A higher value of such methods could increase the code’s clarity but would also increase the project’s overall complexity.
* **CF = 3,34% -** The value seems reasonable since it should be a lower end number. Higher coupling induces a bigger complexity and lower encapsulation which compromise other factors such as understandability and maintainability.

References

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

(consulted in 07/11/2023)