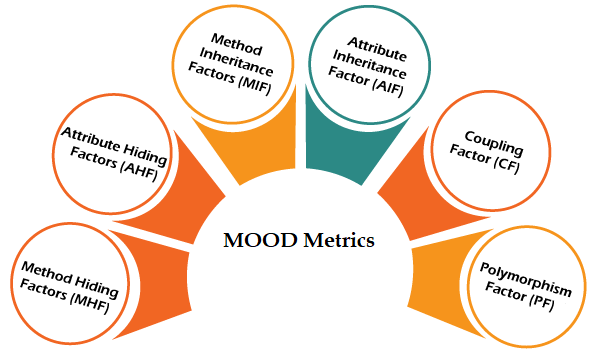
**MOOD Metrics**

João Miguel Lopes Romão Esteves - 47994

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

**Attribute Hiding Factor (AHF)**

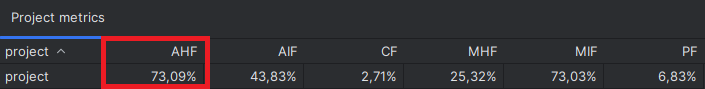
Attribute Hiding Factor (AHF) measures the degree to which the attributes (instance variables) of a class are encapsulated and hidden from external classes. The higher the AHF, the better the encapsulation and hiding of attributes, which is generally considered beneficial for code maintenance and extensibility. This factor can be calculated using the following formula:

Uma imagem com texto

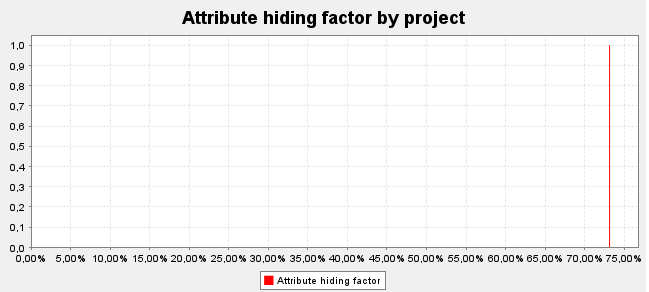
Descrição gerada automaticamente**Ah(Ci) =** Hidden attributes in the class Ci

**Ad(Ci)** = Av(Ci) + Ah(Ci): Attributes defined in Ci

**Av(Ci):** Attributes visible in the class Ci

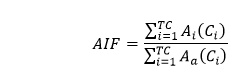
**TC:** Total number of Classes.

Generally, a high AHF value is advisable, as the attributes of a class should be hidden from other classes, making 100% the ideal AHF value. Regarding our project, we have an Attribute Hiding Factor (AHF) of 73.09%, which indicates that the classes in this project follow a good practice of encapsulation. This metric suggests that the majority of attributes (instance variables) in the classes are well protected and not directly accessible by external classes.



**Attribute Inheritance Factor (AIF)**

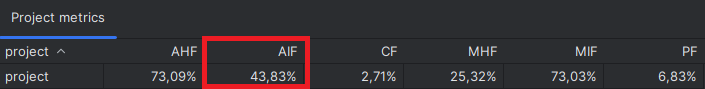
The Attribute Inheritance Factor (AIF) assesses the inheritance of attributes from a parent class to a child class. A high AIF indicates a high inheritance of attributes, which can increase complexity and coupling between classes. A low AIF is generally preferable as it reduces the dependency between classes. This factor can be calculated using the following formula:

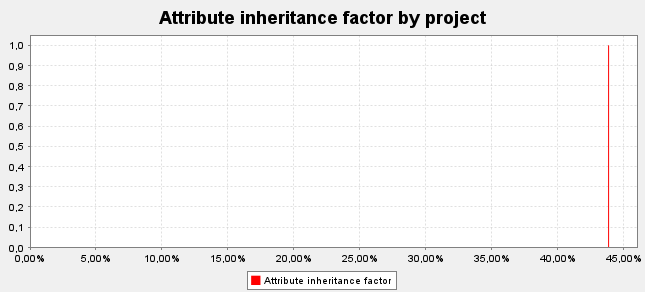
**Ah(Ci) =** Inherited attributes

**Aa(Ci)** = Ad(Ci) + Ah(Ci): Attributes defined in Ci

**Ad(Ci):** Attributes defined in the class Ci

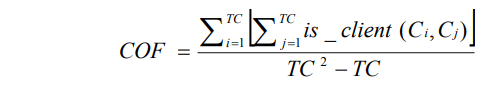
**TC:** Total number of Classes.

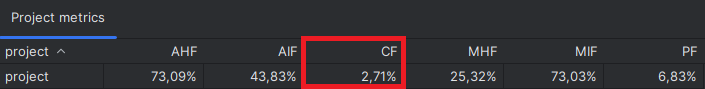


Generally, the range for AIF is between 0% and 48%. According to our program where we have a percentage of 43.83%, we can conclude that the inheritance of attributes between classes in the project is not very extensive, resulting in lower coupling between classes and reduced complexity.

**Coupling Factor (CF)**

The Coupling Factor (CF) measures the dependency between classes in the source code. A low CF indicates that classes are loosely coupled, which is desirable to facilitate code maintenance and reusability. A high CF indicates that classes are tightly coupled and may be difficult to modify without affecting other parts of the system. This factor can be calculated using the following formula:

**is\_client**(Cc,Cs) =| 1 **if** (Ci⇒Cj)^(Ci≠Cj) , **else** 0

**TC:** Total number of Classes.

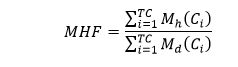
Uma imagem com texto, captura de ecrã, Gráfico, número

Descrição gerada automaticamente A high CF value indicates that the classes in the system are more interconnected and interdependent, leading to the problem that sometimes it's very difficult to change or fix the system in case of any bug or issue because the functionality where the bug resides could be implemented by more than two classes, and we have to make changes in all related classes. In our program analysis, the CF value is only 2.71%, and with such a low CF, the classes in the project are independent from each other, meaning that changes in one class tend to have minimal or no impact on other classes. This is positive as it facilitates code maintenance and modification.

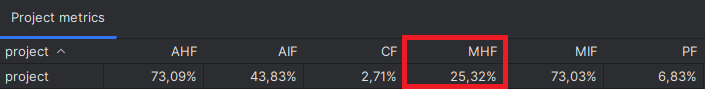
**Method Hiding Factor (MHF)**

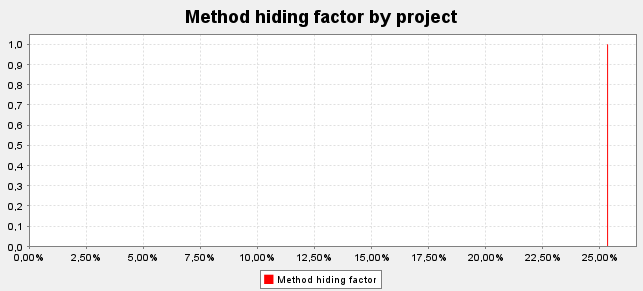
The Method Hiding Factor (MHF) assesses the degree of encapsulation and hiding of methods (functions) within a class. A high MHF indicates that methods are well encapsulated, which is generally preferable to prevent external classes from accessing and modifying methods inappropriately. This factor can be calculated using the following formula:

**Mh(Ci) =** Hidden methods in the class Ci

**Md(Ci)** = Mv(Ci) + Mh(Ci): Methods defined in Ci

**Mv(Ci):** Visible methods in the class Ci

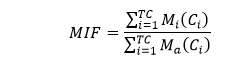
**TC:** Total number of Classes.

A low MHF indicates an insufficiently abstract implementation. A large proportion of methods are unprotected, and the likelihood of errors is high. A high MHF indicates too little functionality. It may also indicate that the design or model includes a high proportion of specialized methods that are not available for reuse. An acceptable MHF value ranges from 8% to 25%. In alignment with our program, a Method Hiding Factor (MHF) of 25.32% in a software project indicates a moderate level of method encapsulation and hiding, allowing us to use and reuse a substantial number of methods while maintaining a sufficiently abstract implementation, resulting in good program functionality.

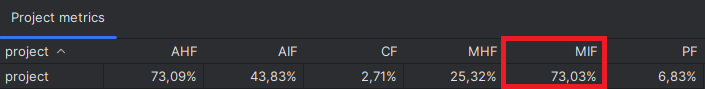
**Method Inheritance Factor (MIF)**

The Method Inheritance Factor (MIF) measures the inheritance of methods from parent classes to child classes. A high MIF indicates a high inheritance of methods, which can increase complexity and coupling between classes. A low MIF is generally preferable to reduce the dependency between classes. This factor can be calculated using the following formula:

**Mi:** Inherited methods

**Ma(Ci)** = Md(Ci) + Mi(Ci): Methods defined in Ci

**Md(Ci):** Methods defined in the class Ci

**TC:** Total number of Classes.

Uma imagem com texto, captura de ecrã, Tipo de letra, Gráfico

Descrição gerada automaticamenteAt first glance, we might be tempted to think that inheritance should be used extensively. However, composing multiple inheritance relationships builds a directed acyclic graph (a hierarchy tree of inheritance) whose depth and width can quickly erode comprehensibility and testability. Generally, the MIF range falls between 20% to 80%. According to our values, a Method Inheritance Factor (MIF) of 73.03% in a software project indicates a high inheritance of methods from parent classes to child classes. This suggests strong functionality reuse, providing us with good program comprehensibility and testability.

**Polymorphism Factor (PF)**

The Polymorphism Factor (PF) assesses the use of polymorphism in the code. Polymorphism allows objects of different classes to be treated uniformly, which can make the code more flexible and extensible. In polymorphism, the child class can implement the method differently. The same method can be implemented differently in the child class and the parent class. It is defined by the ratio between the actual number of method substitutions and the maximum total number of method substitutions. This factor can be calculated using the following formula:

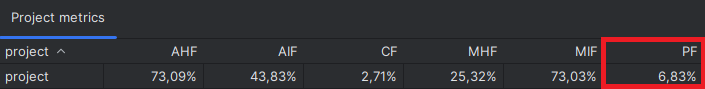
**Uma imagem com texto

Descrição gerada automaticamenteMo(Ci):** Overridden methods in the class Ci

**Mn(Ci):** New Methods in Ci

**DC(Ci):** Number of descendants of class Ci (derived classes)

**TC:** Total number of Classes.



Uma imagem com texto, captura de ecrã, Gráfico, file

Descrição gerada automaticamentePolymorphism arises from inheritance and has its pros and cons. Intuitively, we might expect that polymorphism (overrides) can be used to a reasonable extent to keep the code clear, but excessively polymorphic code can be very complex to understand (as several alternative methods can be executed for a single method call). The PF should be within a reasonable range with both lower and upper limits. When analyzing the PF in our program, we have a Polymorphism Factor (PF) of 6.83%, indicating low use of polymorphism in the project. We can conclude that we have a system that is sufficiently clear and clean with reasonable complexity, allowing for better understanding.