METRICS FOR ASSESSING THE QUALITY OF MODULARIZATION OF OBJECT-ORIENTED SOFTWARE

P.R.THERASA*  
DR.P.VIVEKANANDAN**  

*Research Scholar, College of Engineering, Anna University, Chennai, India  
**Professor & Head, A.C. Tech, Computer Centre, Anna University, Chennai, India

ABSTRACT  
The goal of Object Oriented Software (OOS) is to develop applications that have high quality. Applications that have high quality are more stable and maintainable. In order to assess and improve the quality of an application during the maintenance phase, a developer uses several modularization metrics to analyze the quality of modularization. These metrics estimate the quality of coupling and complexity of the OO System. The cohesion of a module refers to the relatedness of the module components. The module that has high cohesion performs one basic function and cannot be split into separate modules easily. Coupling is a measure of the degree of interaction between two software components (methods, classes, modules, packages). Reusability and maintainability are related to coupling. If a software component is relatively independent, that is, if there are only a few dependencies of this component on other components, it would be easy to understand, maintain, and reuse this component. Coupling reflects the modifiability, maintainability, and reusability of a software product. Software cohesion can be defined as a measure of the degree to which element of a module belong together. This thesis proposes the modularization quality of OO software by measuring the extent in which a class in a module uses another class in some other module and the extent of inter-module call traffic created by inheritance and association relationship.

KEYWORDS: Metrics, Coupling, Cohesion, Structural Metrics

1. INTRODUCTION  
This Metric analyzer is a tool, which would measure structural metrics of the given OOS which is developed in java. It consists of three major kinds of metrics coupling based module Encapsulation, Inheritance based Coupling, Inheritance based Inter-module coupling. The software is tested with different aspect of quality by finding the above metrics value. This structural metrics is very useful for measuring the modularization quality of OO system i.e. maintainability. These metric values would be calculated by means of some standard procedure and newly developed algorithms.
2. STRUCTURAL METRICS:

The relationship and structure of the module will be tested by the structural metrics. These metrics value are very useful to explore the reusability of the given code. In this, the relationships between each module with other modules are explored. All the direct and indirect relationship of the modules would be evaluated. If the relationship of a particular module is high then it can’t be reused frequently. The value of the metrics should be between 0 to 1. If it is near to 0 means it is low relationship with other module and it is a very good system. These systems have very good modularization quality. If it is near to 1 then it is poor modularization quality and cannot be reused. The metrics covered by the tool are classified as General metrics, Coupling based Module Encapsulation and Inheritance based coupling. It is described in the following section.

2.1 General Metrics

- Number of modules
- Number of methods in each module
- Number of cross reference methods for each module
- Number of cross referenced variable for each module
- Relationship diagrams
- Number of direct relationship

![Figure 2.1 Architectural Design](image)

2.2 Coupling Based Module Encapsulation

Coupling

The coupling or dependency is the degree to which each program module relies on each one of the other modules. Coupling is usually contrasted with cohesion. Low coupling
often correlates with high cohesion, and vice versa. The software quality metrics of coupling and cohesion were invented by Larry Constantine, an original developer of Structured Design.

**Low-Coupling**

Coupling can be "low" or "high" (i.e. "weak" or "strong"). Low coupling refers to a relationship in which one module interacts with another module through a stable interface and does not need to be concerned with the other module's internal implementation. With low coupling, a change in one module will not require a change in the implementation of another module. Low coupling is often a sign of a well-structured system, and when combined with high cohesion, supports the general goals of high readability and maintainability.

Systems that do not exhibit low coupling might experience the following developmental difficulties:

- Change in one module forces a ripple of changes in other modules.
- Modules are difficult to understand in isolation.
- Modules are difficult to reuse or test because dependent modules must be included.

**Module Interaction Index**

This metric calculates how effectively a module's Application Programming Interface (API) functions are used by the other modules in the system.

There are \( m \) modules in the system \( S \). Module \( M \) has \( n \) functions \( f_1, f_2, ..., f_n \). \( \text{Nint} \) - number of calls made to a function \( f \) from other functions internal to module \( M \). \( \text{Next} \) - number of calls made to \( f \) from other functions external to module \( M \).

For a method \( f \):

\[
\text{MII}(f) = \frac{\text{Next}}{\text{Next} + \text{Nint}}
\]  

(1)

For a module \( M \):

\[
\text{MII}(M) = \frac{1}{n} \sum_{i=1}^{n} \text{MII}(f_i)
\]  

(2)

For a system

\[
\text{MII(System)} = \frac{1}{m} \sum_{i=1}^{m} \text{MII}(M_i)
\]  

(3)

For example, if the System \( S \) has 2 modules and each module has 5 functions, assume that each function has 2 internal calls and 4 external calls. \( \text{MII}(f_i) = 4/6, \text{MII}(M) = 0.666; \)
2.3 Inheritance Based Coupling

**Inheritance**

Inheritance is the most important feature of Object Oriented Programming. It is the concept that is used for reusability purpose. Inheritance is the mechanism through which it can derive classes from other classes. To derive a class in java the keyword extends is used. The base class may be from the same module or from different module. When a module B extends a class in module A, would naturally create the dependences between those two modules. It is very difficult to maintain the code. A change in a module A may cause B to break. This is called as “Fragile Base-Class” problem.

### 2.3.1 Base Class Fragility Index (BCFI)

The metric is used to measure the extent of base-class-derived-class relationship when the two classes are from different modules.

To measure the BCFI, first it has to measure the BCV (Base Class Violation) value for each method in the module.

\[
BCVio(m) = \frac{\text{no of m overridden outside the module}}{\text{Total no of Overridden}} \tag{4}
\]

**Measure the violation at the module level**

For m of class c,

- Up Callby(m) - set of concrete methods called by m that are either defined in a class c or inherited from its ancestor classes but not overridden.
- BCV Max(m) - the maximum of BCVio values of all of the concrete methods in UpCallby(m) for a method m.
- BCV Set(p) - a given module p, is the set of classes that contains at least one method with nonempty Up Calledby set.

\[
BCV Set(p) = \{ c \in C_p : \exists \text{m in c} (\text{Up Callby(m) \neq \emptyset}) \} \tag{5}
\]

The value of BCFI(p) will range from 0 to 1 with the worst case being 0 and best case (no base class violation) being 1.

\[
BCFI(p) = 1 - \frac{1}{\text{BCV Set}(p)} \sum_{c \in C_p} \text{BCV Max}(c) \tag{6}
\]

The base class violation of the entire system BCFI(S) is the average of the BCFI (p) for all the module in the system.
The value of BCFI(S) will range from 0 to 1, with the worst case being 0 (worst Base Class Violation) and 1 indicating the system completely free from Base class violation case (no base class violation).

\[
BCFI(S) = \frac{1}{|P|} \sum_{n \in N} BCFI(p) \tag{7}
\]

2.3.2 Inheritance-Based Inter-module Coupling Metrics

Measure by module level or class level

P representing the set of modules and C(p) denoting the set of all classes in the module p. Chld(c,d) is true when d is the child of class c.

Metric measures such dependencies at the module level.

Number of other modules derives at least one class from module p

\[
I_{c_1}(p) = 1 - \frac{|\{c \in C(P) : \exists d \in P \text{ such that } \text{Child}(c,d) \land p \neq p\}|}{|P| - 1}
\tag{8}
\]

when \(|P| = 1\)

Number of classes in other modules extends any classes in module p

\[
I_{c_2}(p) = 1 - \frac{|\{d \in C(P) : \text{Child}(c,d) \land \text{Module}(d) \neq p\}|}{|C(P)| - |C|}
\tag{9}
\]

when \(|P| = 1\)

3. IMPLEMENTATION

The metric values are calculated as per the formulas described by the software engineering methodologies. The following section describes the implementation details of all the metric values calculated by the tool.

This system consists of the three main blocks. First the java project is analyzed and extracts the structured information of the overall project and also each module of the project.

3.1 Number of User Defined Packages

The number of packages from the java project is calculated by read the java project. Assign the path of java project to file abject.

Check for is File(). If it is false, get list of files and check is Directory(). If it is true, count the list of files as package. If is file is true, then the previous one is the package. Get the name by method get Name().

www.jiarm.com
3.2 Number of User Defined Files

From java package the list of files are extracted by File object using the method list Files() and assign the list of files to file array object.

The list of files is in array and gets the files by increment till the length and get the name by get Name() method.

3.3 Number of User Defined Methods

There is a Java API called java. lang. reflect.* is available. It has number of predefined classes and interfaces to extract the structural information from the java project. First the java project is analysed and get the number of package and number of files by the above methods. For each java class create the object and get the class by using the method get class(). Assign it to the Class object.

Create the method array and get the declared method from files by using get Declared Method() and all methods by get Methods() . Get the name by get Name method and get return type by get Return type of the object of the method array.

3.4 Cross References calculation

The call references are calculated by the pattern matching algorithm. The declared method is first calculated by the java reflect API and Tokenizes each java files and converted it into a string of tokens. The token string file is matched with the array of declared methods and parameters. The pattern are selected like this (method name&&(|()|) and increment the count of particular method reference attribute.

3.5 Modularization metrics

Structural metrics are very useful in context of reusability of the code and maintenance of the code. Various relationships between modules are explored. The structural metrics covered by the tool are as follows.

Module Interaction Index

The system S can be partitioned into a set of P modules.

- Number of calls made to a function m from other functions internal to module P.
- Number of calls made to m from other functions external to module P.
Base Class Fragility Index (BCFI)
The metric that measure the extent of base-class/derived-class relationship when the two classes are in different modules

- Set of methods inherited from other class’s c but not overridden in class c.
- Set of methods called by m that are either defined in class c or inherited from other (ancestor) class but not overridden.
- Measure the extent to which the methods overridden outside the module

Inheritance-Based Inter-module Coupling
Number of other modules, in which at least one of their classes is derived from at least one class of module p.

4. CONCLUSION
This system would be a useful tool for finding modularization quality of any system developed in java. The other important aspect of this project is re-engineering. It is used as re-engineering tool to explore the legacy system which is developed in java. It is also called as analysis tool. It is used as analysis tool for the no-voice programmer in the organization to study the system modularization. From the above, the aspect of “examining and analyzing” subject system can be easily done by this system in a metric oriented manner. It is concluded that the system is very powerful for re-engineering and finding modularization quality.

Future enhancement
- The system has been designed in scalable manner, so new functionality can be added easily.
- Also, this system can be extended to other languages.
- It has only structural metrics, it can be extended with other metric which is related to the modularization quality and conceptual coupling and cohesion metrics.
- The system has no tree structure view of the relationship. If it added, it will be very useful to view relationship as a diagram.
- This tool calculates the dependency in terms of structure of the system; this is extended with conceptual similarity of each modules of the system.
- It will be used to find more accurate results.
REFERENCES


4. Denys Poshyvanyk, Andrian Marcus”Conceptual Coupling Metrics for Object-Oriented Systems,” 22nd IEEE International Conference on Software Maintenance (ICSM'06)


