ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 2, Issue 2, February 2013

# "Identifying the Quality of Object Oriented Software System using Modularization Approach"

[1]Ms. Pragati D. Chowhan, [2]Prof. S. S. Dhande

[1]Department of Information Technology Sipna COET, SGBAU, Amravati (MH), India
[2]Department of Information Technology Sipna COET, SGBAU, Amravati (MH), India

***Abstract*-Software developers develop their software with some standard specification, but important issue is how to measure the quality of software modularization. In this paper advanced set of metrics are proposed which measure quality of modularization of an Object-Oriented Software (OOS) System. Metrics are designed with universal standard principles to measure the quality of Object-Oriented software which has been developed. Proposed metrics are coupling based structural metrics, these metrics measures the function-call traffic through the API's of the modules in relation to the overall function-call traffic. It is universally accepted that quality of the software modularization is improved when interaction between modules of system is through published APIs only. The metrics can be validated from the results obtained on human-modularized versions of the software.**

**Keywords—Metrics measurement, Software Modularization, API (Application Programming Interface), Coupling.**

## I. INTRODUCTION

The development of any software system is quite a difficult task, so modularization can be done for the development purpose. Modularizing the software system can also complicate the development process. It is challenge to measure the quality of object oriented software modularization. Modularization of object oriented code is distribution of the software into modules and these modules should communicate with each other through some application programming interface that is API. A modularized software is easy to maintain and can also help the developer. In our work we consider the object oriented technology that is the java language code for defining metrics and modularization. We consider the various java files as modules. A java file may consist of number of classes but one public class which is accessible by other modules also. We know that classes contain elements such as methods, interfaces variables, etc. The java code contains multiple files with thousands of line of code so we will use code parser to analyze the modularity of the code.

The earliest contributions to software metrics deal with the measurement of code complexity and maintainability based on the complexity measures. Some of the earliest software metrics are based on the two factors that are coupling and cohesion. Important attributes of any modularized software is low intermodule coupling, high intramodule cohesion, and low complexity. Coupling, cohesion, and complexity metrics measure the various qualities attributes of Object Oriented Software. Cohesion is measured as the ratio of the number of internal function-call dependencies that actually exist to the maximum possible internal dependencies, and Coupling is measured as the ratio of the number of actual external function-call dependencies between the two subsystems to the maximum possible number of such external dependencies.

According to several principles and laws of object oriented design, designers should always strive for low coupling and high cohesion. This software will measure the quality of any object oriented software or we can say the java program. It checks the internal details of object oriented software and gives information about the quality of software that is to be tested. This system can reorganize legacy software, consisting of millions of line of object oriented code that was never modularized or poorly modularized. User can test their object oriented software to check modularization quality. User can check their software to test whether there is any improvement from previous version of that same object oriented software. This system provides guidelines if there is any fault in software that is being tested.

## II. PRELIMINARIES

### *Getting Input*

User or tester will import file/project to our tool. The desired project which is to be tested is given as an input to this module. This is done in Java using File Input Stream. For Example: *public FileInputStream(String name)*
*throws FileNotFoundException*
This creates a *FileInputStream* by opening a connection to an actual file, the file named by the path name *name* in the file system. A new *FileDescriptor* object is created to represent this file connection.

### *Code Parsing*

The tool will partition the source code by its file type. Also in this module the file will be partition in to the form which is easy for applying our metrics. String Tokenizer and File Name Filter are used for this purpose. One of the actions the java compiler does, when it compiles the source code is to parse your .java file and create tokens that match the java grammar. When you fail to write the source code properly (for instance forget to add a ";" at the end of a statement), is the parser who identify the error. In computing, a parser is one of the components in an interpreter or compiler that checks for correct syntax and builds a data structure implicit in the input tokens. The parser often uses a separate lexical analyser to create tokens from the sequence of input characters. Parsers may be programmed by hand or may be (semi-)automatically generated (in some programming languages) by a tool. The most common use of a parser is as a component of a compiler or interpreter. This parses the source code of a computer programming language to create some form of internal representation. Programming languages tend to be specified in terms of a context-free grammar because fast and efficient parsers can be written for them. Parsers are written by hand or generated by parser generators. Context-free grammars are limited in the extent to which they can express all of the requirements of a language. Informally, the reason is that the memory of such a language is limited. The grammar cannot remember the presence of a construct over an arbitrarily long input; this is necessary for a language in which, for example, a name must be declared before it may be referenced. More powerful grammars that can express this constraint, however, cannot be parsed efficiently.

*Finding Application metadata*

In this module the tool will find the size/total number of lines in the project. After that calculate what are the functions/methods are involved in this project. How many methods call from other modules, how many modules call other modules and what are all the functions from other module and find how many classes and modules in a given file. Metadata is structured information that describes, explains, locates, or otherwise makes it easier to retrieve, use, or manage an information resource. Metadata is often called data about data or information about information.

*Storing into Database*

The output from the above module is stored in the database. It can be done using JDBC-ODBC Driver. A JDBC driver is a software component enabling a Java application to interact with a database. The JDBC type 1 driver, also known as the JDBC-ODBC bridge, is a database driver implementation that employs the ODBC driver to connect to the database. The driver converts JDBC method calls into ODBC function calls.

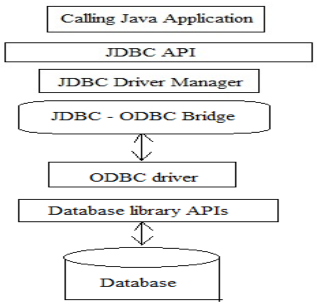

Figure1: Schematic of the JDBC-ODBC Bridge

*Applying Metrics*

This module is heart of our project. Here we are going to calculate the quality of software based on the modules, function and size. Functions used: input (), apply_metrics (), apply_formulae (), output (). There are three types of metrics used to calculate the quality of software. IFAC (Index factor for API function calls), IFNC (Index factor for non API function calls), IFMC (Index factor for module communication). IFAC measures the extent to which a software system adheres to the module encapsulation principles that are maximization of API-Based Intermodule call traffic and minimization of non-API-Based Intermodule call traffic. Since these API functions are meant to be used by the other modules, the internal functions of a module typically would not call the API functions of the module. Ideally, all the external calls made to a module should be routed through the API functions only and the API functions should receive only external calls. Ideally, the non-API functions of a module should not expose themselves to the external world. In reality, however, a module may exist in a semi modularized state where there remain some residual inter module function calls outside the API's. In this intermediate state, there may exist functions that participate in both inter module and intra module call traffic. We measure the extent of this traffic using a metric that we call Index factor for non API function calls, or IFNC. This index works on the principle of maximization of module coherence on the basis of Commonality of Goals.

IFMC or Index factor for module communication, determines what fraction of the API functions exposed by a module is being used by the other modules. Any single other module may end up using only a small part of the API. The intent of this index is to discourage the formation of a large, monolithic module offering services of disparate nature and encourage modules that offer specific functionalities. This index works on the principles of minimization of non-API-Based Intermodule Call Traffic.

## III. Proposed System

In our proposed system we will be giving a modularized object oriented code as an input. Then the code will be parsed to get the details of the elements present in it. Then the coupling-based structural metrics which we are giving in this paper will be applied. Finally with the help of the code analyzer we will get the output chart. Now next time another modularized version of the code will be given as an input and the same steps will be followed to get the new output chart. Hence, we can compare the two charts to identify the results of metrics implementation on the code, which will help the developer to understand the quality of modularized code and can take the decision of which modularized code can be used for the software system. Particularly in object oriented software development developer needs to use a lots of object oriented concepts which may lead to the inter dependency of the various units of the software like Inheritance. Therefore, software metric is a measure of some property of a piece of software or its specifications, so software metric is essential. Considering this same issue we are providing the software metrics for this modularized object oriented code.

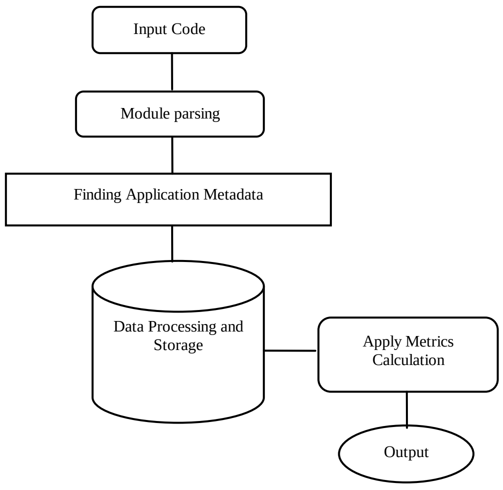
Input Code
Module parsing
Finding Application Metadata
Data Processing and Storage
Apply Metrics Calculation
Output

Figure 2: Flow Diagram of the proposed system

*Input Code:* The desired project which is to be tested is given as an input to this module by the user or tester by importing the file. It can done using File Input Stream in Java language.

*Module Parsing:* Then the source code will be partitioned by its file type. Here the file will be partition in to the form which is easy for applying the Coupling based Structural metrics. For this purpose String Tokenizer and File Name Filter are used.

*Finding Application Metadata:* Here the tool will find the size or total number of lines in the project. Then will calculate all the functions or methods involved in this project, number of methods call from other modules, how many modules call other modules and what are all the functions from other module and also how many classes and modules in a given file.

*Database Processing and Storage:* The output from the above module is stored in the database which can be done using JDBC-ODBC Driver.

*Apply Metrics Calculation:* This module is the main part of our project. Here we will actually calculate the quality of software based on the modules, functions and size using the coupling-based structural metrics.

*Output:* Finally with the help of the code analyzer we will get the output chart. Now next time another modularized version of the code will be given as an input and the same steps will be followed to get the new output chart. Now here we can compare the two charts to identify the results of metrics implementation on the code. From which the developer will be able to understand the quality of modularized code and decision can be taken which modularized code to be used for the software system.

## IV. Proposed Metrics

In the Proposed system we considered the leaf nodes of the directory hierarchy of the original source code to be the most fine-grained functional modules. All the files (and functions within) inside a leaf level directory were considered to belong to a single module. In this manner, all leaf level directories formed the module set for the software. After that we apply Coupling-based Structural Metrics as follows:

*Coupling-Based Structural Metrics:* We will begin with coupling-based structural metrics that provide various measures of the function-call traffic through the API's of the modules in relation to the overall function-call traffic. For that we have find the following four factors.

### *A. IFAC (Index factor for API function calls):*

This metric calculates how effectively a Module's API functions are used by the other Modules in the system. This metric measures the extent to which a software system adheres to the module encapsulation principles that are maximization of API-Based Intermodule call traffic and minimization of non-API-Based Intermodule call traffic. Since these API functions are meant to be used by the other modules, the internal functions of a module typically would not call the API functions of the module. Ideally, all the external calls made to a module should be routed through the API functions only and the API functions should receive only external calls. Suppose module has n API functions and let's say that nj number of API functions are called by another module mj. Also assume that there is z number of modules from module 1 to module z that calls one or more API functions of module.

IFAC (module) = (n1+n2+…nz) / (n * z)
= 0, if n i. e. number of API function is zero.
If we assume that moduleapi is the total number of modules having more than zero API functions. Then
IFAC (system) = SUM [IFAC (module)i] / moduleapi
where i = 1 to module api……………….................... (1)
The maximum value of this metric IFAC (system) will be 1, depending on the focus and nature of the modules with similar purpose.

### *B. IFNC (Index factor for non API function calls):*

This metrics works on the principle of maximization of module coherence on the basis of Commonality of Goals. In the ideal condition, the non-API functions of a module should not expose themselves to the external world. In reality, however, a module may exist in a semi modularized state where there remain some residual inter module function calls outside the API's. In this intermediate state, there may exist functions that participate in both Intermodule and Intermodule call traffic. We measure the extent of this traffic using this metric. Let us represent API function as functionapi and non API functions as functionnapi for given module.Then total function will be, function = function api + functionnapi
Total number of modules is M.

IFNC (module) = functionnapi / (function - function api)
= 0, if the non API functions are zero.
IFNC for the entire software system S by,

IFNC (system) = SUM [IFNC (module)i] / M,
where i =1 to M……………………………………. (2)
In good modularized object oriented software, functions will be either API or non API type of functions. And non API functions are not used outside the module.
Then function - functionapi will be equal to functionnapi. So that IFNC (module) = 1. Here sometimes the value of the IFNC (module) can be between 0 and 1.

### *C. IFMC (Index factor for module communication):*

This metric calculates the index factor for module communication and how well API functions of modules are used by the other modules in the system for communication. Assume that a module has n functions from 1 to n, of which the n1 API functions are given by the subset {$f_1$ api …$f_{n1}$ api}. Cext is used to denote the total number of external calls coming from the other modules. It is a java file as module. Also assume that system has $m_1$ to $m_i$ modules. Total number of modules is M. Index Factor for module communication (IFMC) for a given module and for the entire software system is given by,

IFMC (module) = {SUM [Cext (fapi)]} / Cext (module)

where fapi is in range from $f_1$ api to $f_{n1}$ api = 0, when there are no external calls made to the particular module
IFMC (system) = {SUM [IFMC (module)i ]} / M,
where i is in range from 1 to M………………………… (3)

## V. Conclusion

This paper gives a set of design principles for code modularization and proposed a set of metrics which characterize software in relation to those principles. Structural metrics are driven by the notion of API - a notion central to modern software development. These metrics are essential because otherwise it would be possible to declare a malformed software system as being well-modularized. In some extreme case in point there is putting all of the code in a single module would yield high values for some of the API-based metrics, therefore the modularization achieved would be functionally correct.

## Future Scope

In future other metrics can be introduced which are based on notions such as size-boundedness, size-uniformity, operational efficiency in layered architectures, and similarity of purpose play important supporting roles. Modules in any one layer are only allowed to call the modules in layers below them in horizontal layering. By the software design community such layered organization of the modules is considered to be important architectural principle. Based on Information-Theoretical principles, we can characterize each module on the basis of the similarity of purpose of the services which is offered by the module.

## References

[1] E. Arisholm, L.C. Briand, and A. Foyen, “Dynamic Coupling Measurement for Object-Oriented software,” IEEE Trans. Software Eng., vol. 30, no. 4, pp. 491-506, Aug. 2004.
[2] L.C. Briand, J.W. Daly, and J.K. Wust, “A Unified Framework for Coupling Measurement in Object-Oriented Systems,” IEEE Trans. Software Eng., vol. 25, no. 1, pp. 91-121, 1999.
[3] S.R. Chidamber and C.F. Kemerer, “A Metrics Suite for Object Oriented Design,” IEEE Trans. Software Eng., vol. 20, no. 6, pp. 476-493, June 1994.
[4] N. Churcher and M. Shepperd, “Comments on a Metrics Suite for Object-Oriented Design,” IEEE Trans. Software Eng., vol. 21, no. 3,pp. 263-265, Mar. 1995.
[5] M.H.Halstead, Elements of Software Science, Operating and Programming Systems Series, vol. 7, Elsevier, 1977.
[6] B. Kitchenham, S. Pfleeger, and N. Fenton, “Towards a Framework for Software Validation Measures,” IEEE Trans. Software Eng., vol. 21, no. 12, pp. 929-944, Dec. 1995.
[7] T.J.McCabe and A.H. Watson, “Software Complexity,” Crosstalk, J. Defense Software Eng., vol. 7, no. 12, pp. 5-9, Dec. 1994.
[8] Briand, L. C., Wüst, J., Daly, J. W., and Porter, V. D., "Exploring the relationship between design measures and software quality in object-oriented systems", Journal of Systems and Software , vol. 51, no. 3, May 2000, pp. 245-273.
[9] El-Emam, K. and Melo, K., "The Prediction of Faulty Classes Using Object-Oriented Design Metrics", NRC/ERB-1064, vol. NRC 43609, November 1999.
[10] Gall, H. Jazayeri, M. Krajewski, J, "CVS Release History Data for Detecting Logical Couplings", 6th International Workshop on Principles of Software Evolution (IWPSE'03) Sept.1- 2, 2003, pp.13-23.
[11] Gyimóthy, T., Ferenc, R. , and Siket, I., "Empirical validation of object-oriented metrics on open source software for fault prediction", IEEE Trans. on Software Engineering , vol. 31, no. 10, October 2005, pp. 897-910.
[12] Kramer, S. and Kaindl, H., "Coupling and cohesion metrics for knowledge-based systems using frames and rules", ACM Trans. on Soft. Engineering and Methodology (TOSEM) , vol. 13, no. 3, July 2004, pp. 332-358.
[13] Kuhn, A., Ducasse, S., and Girba, T., "Enriching Reverse Engineering with Semantic Clustering", in Proc. of 12th Working Conference on Reverse Engineering, Nov. 7-11 2005, pp. 133-142.
[14] Lee, J. K., Jung, S. J., Ki m, S. D., Jang, W. H., and Ham, D. H., "Component identification method with coupling and cohesion", in Proc. of 8th Asia -Pacific Software Engineering Conference (APSEC'01), Dec. 2001, pp. 79-86.
[15] Lee, Y. S., Liang, B. S. , Wu, S. F., and Wang, F. J., "Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow", in Proceedings of International Conference on Software Quality, Maribor, Slovenia, 1995.
[16] Li, W. and Henry, S., "Object-oriented metrics that predict maintainability", Journal of Systems and Software, vol. 23, no. 2, 1993, pp. 111-122.
[17] Marcus, A. and Poshyvanyk, D., "The Conceptual Cohesion of Classes", in Proceedings of 21st IEEE International Conference on Software Maintenance (ICSM'05), Budapest, Hungary, Sept. 2005, pp. 133-142.

[1]**Ms. Pragati D. Chowhan** student at Sipna College of Engineering & Technology, Amravati. B.E. (IT), M.E. (IT) pursuing, Sipna College of Engineering & Technology, Amravati. Sant Gadge Baba Amravati University, Amravati (MH). India.
[2]**Prof. S. S. Dhande** Associate Professor at Sipna College of Engineering & Technology, Amravati. B.E. (CSE) M.E. (CSE) from Sant Gadge Baba Amravati University, Amravati. Currently pursuing PHD in faculty of Engineering & Technology in the area of OODatabase Management System from Sant Gadge Baba Amravati University, Amravati (MH), India. She has published many papers at National as well as International level. She is a member of ISTE, IE, and IETE India.