IJETST- Volume||01||Issue||04||Pages 550-553||June||ISSN 2348-9480 2014

Open access Journal International journal of Emerging Trends in Science and Technology

Object Oriented Software Modularization Quality Measurement Based On API and Information Theoretic Metrics

Authors
Ruchi Kulkarni[1], Samidha Diwedi Sharma[2]

[1] M-TECH *,Department of Information Technology,
NRI Institutions, University of RGPV, Bhopal (MP)
India
E-mail:*ruuchik@gmail.com*
[2]Prof, Department of Information Technology,
NRI Institutions, University of RGPV, Bhopal (MP)
India
E-mail: *samidhad2000@gmail.com*

ABSTRACT

In this research project we are enhancing the software quality measurement using API and theoretic information metrics. Along with the new set of metrics that measure the quality of modularization of a non-object-oriented software system, we are presenting metrics that also measures the quality of modularization of object oriented software system. We have proposed a set of design principles to capture the notion of modularity and defined metrics centered on these principles. These metrics characterize the software from a variety of perspectives: structural, architectural, and notions such as the similarity of purpose and commonality of goals. (By structural, we are referring to inter module coupling-based notions, and by architectural, we mean the horizontal layering of modules in large software systems.) We employ the notion of API (Application Programming Interface) as the basis for our structural metrics.

In addition to this, we proposed a System to measure the quality of modularization of object-oriented software system. Here our work is proposed in three Parts as follows:

Module 1: DEFINING METRICS FOR OBJECT ORIENTED SOFTWARE AND ALGORITHM. Module 2: CODE PARSER and Finally Module 3: CODE ANALYZER.

Index Terms*— Application programming interface; modularization; dependency; communication.*

INTRODUCTION

The software development process is a difficult and modularization can makes it more complicated. This is the challenge to measure the quality of objects oriented software modularization. Modularization of object oriented code is distribution of the software in to modules and these modules should communicate with each other through some application programming interface (API). More properly modularized software is also easy for maintenance work and it can help the developer. In our work we are considering the object oriented java language code for defining metrics and modularization. In java the code is residing in directories of some modules and java files are considered as modules. One java file may consist of number of classes but one public class [4]. Classes contain the elements such as methods, interfaces, variables etc. This code contains many files with thousands of line of code so we are using code parser to analyze the modularity of the code.

Now days lot of software's are developed by the developers. Many of the software's are very big in code size.

So generally to maintain the quality of the code, developers need to distribute the code in small pieces or parts. But how to divide the software is also an important task as it can lead to various problem of inter module communication therefore this modularized code should also be checked for the quality. There are problems in removing the errors of non modularized code. Particularly in object oriented software development developer needs to use a lots of object oriented concepts which may introduced the inter dependency of the various units of the software e.g. Inheritance. Software metric is a measure of some property of a piece of software or its specifications. Therefore

software metrics suite is needed [2]. We are concentrating on the same issue and providing the software metrics for this modularized object oriented code.

PREVIOUS WORKS

In this previous works, we studied the metrics which are developed only for the non-objective oriented software systems.

PROPOSED APPROACH

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 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.

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.

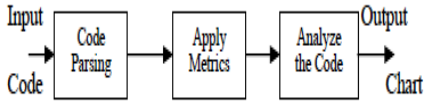

Figure 1. Three important stages of proposed system

PROPOSED METRICS

The following metrics are proposed based on object oriented programming concepts which are largely used for the software development. The non-object oriented metrics given by Sarkar et. al. [1] is a base for our work. Application programming interface (API) is the important term which we are going to use. API functions are the functions only which can be get called outside the module and non API functions are not called outside the module. In our implementation we are going to check if a function calling is found in another module or class then it will be API function and if not found then such functions will be considered as isolated and non API functions. The measurement technique is applying the metrics [3]. The proposed metrics for object oriented code are as follows:

A. Index Factor for module communication (IFMC)

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 {f1api …….f n1api}. 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 m1 to mi modules. Total number of modules is M. Index Factor for module communication (IFMC) for a given module and for the entire software system by required

$$IFMC\ (module) = \{SUM\ [C_{ext}(f^{api})]\} / C_{ext}(module),\ \text{where } f^{api} \text{ is in range from } f_1^{api} \text{ to } f_{n1}^{api}$$

$$= 0, \text{ when there are No external calls made to the particular module}$$

$$IFMC\ (system) = \{SUM\ [IFMC\ (module)_i\ ]\} / M,$$

$$\text{where i is in range from 1 to M} \qquad (1)$$

IFMC measures the degree of module encapsulation i. e. here should be encapsulation of the module code and communication of external function should be through available APIs. These API functions represent the services that the module has to offer. Since these API functions are destined to be used by the other modules, the internal functions of a module usually would not call the API functions of the module. Therefore, a non-API function of a module should not get external calls to the maximum extent possible. In object oriented software, objects can call methods from another class.

Here Numerator of the IFMC (*module*) for a particular module increases as more and more inter module communication is done through the API functions and the ideal value of IFMC (*module*) will be tends to 1. Therefore, IFMC largely measures the degree to which encapsulation related concepts have been followed. Hence in many object oriented software systems it should be tried to maintain a big value for the IFMC.

B. Index Factor for API function Calls (IFAC)

This index factor determines the usefulness of calling the API function by the other modules. Some times in one java file (module) may consists of various classes and API functions with different functionalities. If any other single module is calling the API but need only small part of it then it is unnecessarily calling the big API.

Hence to avoid the formation of such module this index factor is used. Suppose module has n API functions and let's say that nj numbers of API functions are called by another module mj. Also assume that there are z numbers of modules from *module1* to *modulez* that calls one or more API functions of module.

$$IFAC\ (module) = (n_1+n_2+\ldots n_z) / (n * z)$$

$$= 0, \text{ if n i. e. number of API function is zero}$$

If we assume that *moduleapi* is the total number of modules having more than zero API functions. Then

$$IFAC\ (system) = SUM\ [IFAC\ (module)_i] / module_{api},$$

$$Where\ i = 1\ to\ module_{api} \quad (2)$$

The maximum value of this metric IFAC (*system*) will be 1, depending on the focus and nature of the modules with similar purpose.

C. Index Factor for Non API function Calls (IFNC)

If the big software system is not modularized fully then there can be the use of non API functions. This is not preferable. As there should not be a use of non API function outside the module or a java file.
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.

$$IFNC\ (module) = function^{napi} / (function - function^{api})$$

$$= 0, \text{ if the non API functions are zero.}$$

$$IFNC\ (system) = SUM\ [IFNC\ (module)_i] / M,$$

$$Where\ i = 1\ to\ M.$$

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 - function api* will be equal to *functionnapi*. So that IFNC (*module*) = 1.
Here sometimes the value of the IFNC (*module*) can be between 0 and 1.

D. Index Factor for Implied Dependency (IFID)

When function in one module is writing to a global variable that is in use by another module then there is indirect dependency. There can be many events where this kind of dependency occurs in program. Generally in large enterprise application made in object oriented language may have complex source for the hidden dependency between the modules.

Let us say that dependency is denoted by Dglobal (modulea, moduleb) where a≠b. In which dependency will be there when modulea tries to write in to global entity (e.g. files, variables etc) at the same time moduleb is also trying to work on the same entity.
Let us say that Dfunction (modulea, moduleb) where a≠b. In which the calls are made by the functions in modulea to functions in moduleb. Then the Index factor for the implied dependency for module will be given by,

$$IFID\ (module) = SUM\ [D_{function}(module_a, module_b)] / SUM\ [D_{function}(module_a, module_b) + D_{global}(module_a, module_b)],$$
$$\text{for all implied dependencies}$$

$$= 1, \text{ when } D_{global}(module_a, module_b) = 0$$

$$IFID\ (system) = IFID\ (module) / M,$$

$$Where\ M\ is\ total\ number\ of\ module\ from\ 1\ to\ M. \quad (4)$$

From this metric we can say that there should be very less or none implied dependencies in the system.
Generally the value of IFID (*system*) is equal to 1.

ALORITHM FOR METRICS

We are providing here the customized algorithm for the proposed metrics as per the need of the project for implementation.

A. Algorithm for Index Factor for module communication (IFMC)

Algorithm for IFMC
Step 1: To find API functions of modules
Step 2: To find external calls made to the API functions
Step 3: Addition of all external calls made to the API functions
Step 4: To make total of all external calls to module
Step 5: Divide Step 3 by Step 4
Step 6: Repeat Step 1 to 5 for each module
Step 7: Add IFMC (*module*) for all modules
Step 8: Divide Step 7 by total number of modules
Step 9: Output is IFMC (*system)*

B. Algorithm for Index Factor for API function Calls (IFAC)

Algorithm for IFAC
Step 1: To find the each of the API functions called by other modules
Step 2: Add all those API functions
Step 3: Multiply total number of API functions available in module with number of modules calling one or more API functions of the module
Step 4: Divide Step 2 by Step 3
Step 5: To find the total number of modules having more than zero API functions i.e. *moduleapi*
Step 6: Add IFAC (*module*)i for value of i from 1 to *moduleapi*
Step 7: Divide Step 6 by Step 5
Step 8: Output is IFAC (*system*)

C. Algorithm for Index Factor for Non API function Calls (IFNC)

Algorithm for IFNC

Step 1: To find the total number of API functions of given module
Step 2: To find the total number of non API functions of given module
Step 3: Total number of functions in module = Add Step 1and Step 2
Step 4: Divide Step 2 by (Step 3 – Step 1), output is IFNC (*module*)
Step 5: Add IFNC (*module*) of all the modules
Step 6: Divide Step 5 by Total number of modules
Step 7: Output is IFNC (*system*)

D. **Algorithm for Index Factor for Implied Dependency (IFID)**

Algorithm for IFID

Step 1: To find the sum of functional dependencies Dfunction (modulea,moduleb)
Step 2: To find the sum of functional dependencies [Dfunction (modulea, moduleb)] and global dependencies [Dglobal (modulea, moduleb)]
Step 3: Divide Step 1 by Step 2
Step 4: Output is IFID (*module*)
Step 5: Divide Step 4 by M
Step 6: Output is IFID (*system*)

TABLE 1. POPOSED METRICS

Sr. No.	Proposed Metric
1	Index Factor for module communication (IFMC)
2	Index Factor for API function Call (IFAC)
3	Index Factor for Non API function Calls (IFNC)
4	Index Factor for Implied Dependency (IFID)

WORK DONE

In this work we have given the metrics for object oriented software which is made in java for measuring the quality of the modularization. We have provided the algorithm for the metrics based on which implementation is done. We have implemented the metrics using java language. We are using the java Swing for the creation of GUI for the module parser and code analyzer. Module parser is basically separating the elements of the input code module to the system and will be saving them in database. Then code analyzer will be analyzing the code with the defined metrics respect to the metrics applied and will be showing the output chart for comparing

CONCLUSION

Here more emphasis is given on Module 1 of the project "Defining metrics for object oriented software modularization and algorithm for it". Here we have proposed four metrics for calculating the index factor for module communication, Index Factor for API and non API function Calls and Index factor for implied dependency of the object oriented module.

REFERENCE

[1] Sarkar S., Kak A. C. and Rama G. M, "API-Based and Information-Theoretic Metrics for measuring the Quality of Software Modularization" IEEE Trans. Software Eng., vol. 33, no. 1, pp.14-30.

[2] Chidamber S. R. and Kemerer C. F.,"A Metrics Suite for Object Oriented Design," IEEE Trans. Software Eng., vol. 20, no. 6, pp. 476-493, June 1994.

[3] Pfleeger S. and Fenton N., Software Metrics: A Rigorous and Practical Approach. Int'l Thomson Computer Press, 1997.

[4] Schildt H., The Complete Reference Java 2, Fifth Edition, TATA McGRAW HILL, 2002, pp 13-54.

[5] Pressman R. S., Software Engineering: A Practitioners Approach, 6/e, TATA McGRAW HILL, 2005, pp 461-670.

[6] E. Weyuker, "Evaluating Software Complexity Measures," IEEE Trans. Software Eng., vol. 14, no. 9, pp. 1357-1365, Sept. 1988.

[7] 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.

[8] L.C. Briand, S. Morasca, and V.R. Basili, "Property-Based Software Engineering Measurement," IEEE Trans. Software Eng., vol. 22, no. 1, pp. 68-85, Jan. 1996.

[9] N. Sharma, P. Joshi, and R.K. Joshi, "Applicability of Weyuker's Property 9 to Object Oriented Metrics," short note, IEEE Trans. Software Eng., vol. 32, no. 3, pp. 209-211, Mar. 2006.

[10] 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.