www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume - 3 Issue -9 **September,** *2014 Page No. 8175-8178*

A Metric Base Calaculation for Object Oriented Software Modularization Quality Measurement

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

[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

Software development and maintenance is major concern to adopting modularization. Measuring the design quality early during software development has been regarded as a prominent way to assure the quality of software products. Several models has been proposed to estimate the quality of software systems.
This system proposed an approach for determining the design quality of an Object Oriented software using software metrics. The metrics for object oriented design focus on measurements that are applied to the class and design characteristics. To validate the proposed methodology, we have chosen Open source software project. We extracted a set of chosen software metrics that play a definite role in software design quality. Our metrics characterize the quality of modularization with respect to the APIs of the modules. The percentile average values calculated by these metrics formulate a straight forward approach to assign a design quality for any software systems.

For this work simulation, an application is developed in java. This system examines 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 Experiments are carried out by means of different software version and the result show that it works properly. The outcomes of the experimental study provide a strong base for the effectiveness of our system for metric based design quality measurement of object-oriented software.

Introduction

The fundamental issue faced by the developers in today's environment is how to measure the quality of software. Over the past couple of decades, the speed of computer hardware development has far exceeded software productivity development. As computers are used in al1 sorts of everyday activities, the demand for sophisticated and flexible software also increases

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 as modules.

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 . 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. Time constraint was also a major issue as different metrics were calculated individually and results was also based on that metrics so it was difficult in respect to complexity also.

Proposed approach

The work reported here is an experiment to check the modularization quality of the object oriented java code using the following four metrics. These metrics are the base for this experiment which is referred from Sarkar et al. Details of the metrics or formulas are given in their work. The list of referred metrics and their relationship with the assumptions is provided. The referred metrics are as follows:

- Module Interaction index
- API function usage index
- Non API function usage index
- Implicit Dependency index

In this system MOOD metrics are discussed in the context of encapsulation, inheritance. The MOOD metrics, defined by Fernando Brito e Abreu[2], are designed to provide a summary of the overall quality of an object-oriented project. The MOOD metrics referred here as

- Method Inherited Factor
- Method Hiding Factor

After applying this selection of metrics we will then identify OMI by the values of this metrics.OMI will tell the modularity of any software with reference to this metrics. By OMI the modularization achieved would be functionally correct.

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 . The proposed metrics for object oriented code are as follows:

Module Interaction Index(MII)

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.....fn1api}. 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.

$$MII(m) = \frac{\sum f^a \in \{f_1^a \ldots f_{n1}^a\} K_{ext}(f^a)}{K_{ext}(m)}$$

In ideal case when all the module calls are routed through the function calls only, value of MII should be 1.

Application Programming Usage Index(APUI)

This index determines what fraction of the API functions exposed by a module is being used by the other module. 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. The maximum value of this metric should be 1.

$$APIU(m) = \frac{\sum_{j=1}^{k} n_j}{n * k}$$

Non API Function Closedness Index(NC)

Ideally,the non-API functions of a module should not expose themselves to the external world. 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. For a well designed module value of NC will be 1. otherwise the value will be between 0 and 1.

$$NC(m) = \frac{\left|F_m^{na}\right|}{\left|F_m\right| - \left|F_m^{a}\right|}$$

Implied Dependency Index(IDI)

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. The number of dependencies must be few and far between.This is based on principle of module encapsualtion P2.The ideal value should be 1.

$$IDI(m) = \frac{\sum_{m_j \in C(m)} D_f(m_i, m_j)}{\sum_{m_j \in C(m)} (D_g(m_i, m_j) + D_f(m_i, m_j))}$$

Method Inherited Factor(MIF)

MIF is defined as the ratio of the sum of the inherited methods in all classes of the system under consideration to the total number of available methods(locally defined plus inherited) for all classes.MIF measure directly the number of inherited methods as a proportion of the total number of methods. Method hiding factor measure how variables and methods are encapsulated in a class. Visibility is counted in respect to other classes. MHF represent the average amount of hiding among all classes in the system. A private method is fully hidden. In JAVA, hiding is as following: Protected, Public,Private.

MHF = 1 –Methods Visible

Methods Visible = sum(MV) / (C-1) / Number of methods

MV = number of other classes where method is visible

C = number of classes

Method Hiding Factor(MHF)

MHF is defined as the ratio of the sum of the invisibilities of all methods defined in all classes to the total number of methods defined in the system under-consideration. The invisibility of a method is the percentage of the total classes from which this method is not visible. In other words, MHF is the ratio of hidden methods –protected or private methods. If all methods are private, MHF=100% .If all methods are public, MHF=0%

MIF = inherited methods/total methods available in classes

Implementation Method

- Read stored meta-data from the database and calculate various metrics values
- Apply following formula
- OMI = [Σ^n_1 (m_i * w_i)] / n, where
  m: metrics value
  w: Weight Value
  n: total metrics

Conclusion

The proposed system is capable to estimate the software quality of the modules of the software. The referred metrics are implemented for object oriented code modularization and it is also proposed that use of metrics with few assumptions can be done for Object-Oriented Software System. The implemented metrics are based on the concept of API. The output values lies in the range 0 to 1 after applying metrics on code. This dissertation also deals with the role of code analyzer. This will help the developer to provide a quality modularized code.

References

[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] Santonu Sarkar, Member, IEEE, Avinash C. Kak, and Girish Maskeri Rama,"Metrics for Measuring the Quality of Modularization of Large-Scale Object-Oriented Software" , IEEE Trans. Software Eng.VOL. 34, NO. 5, SEPTEMBER/OCTOBER 2008

[3] Abreu, Fernando B: "The MOOD Metrics Set," *Proc. ECOOP'95 Workshop on Metrics,* 1995IEEE Trans. Software Eng,

[4] R. Harrison, S. Counsell and R Nithi. An evaluation of the MOOD Set of Object-Oriented Software Metrics. IEEE Transaction on Software Engineering, Vol. 24, No. 6,June 1998

[5] 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.

[6] Fenton, N., and Pfleeger, S. L. "Software Metrics - A Rigorous and Practical Approach", 2ed. International Thomson Computer Press, London, 1996.

[7] L. A. Laranjeira, “Software Size Estimation of Object-Oriented Systems”, IEEE Transaction on SoftwareEngineering, Vol. 16, No. 5, May 1990, pp. 510-522.

[8] W. Li and S. Henry, “Object-oriented Metrics whichPredict Maintainability”, The Journal of Systems andSoftware, Vol. 23, Issue 2, November 1993, pp. 111-122.

[9] V. L. Basili, L. Briand and W. L. Melo, “A validationof object-oriented Metrics as Quality Indicators”,IEEE Transaction Software Engineering. Vol. 22, No. 10, 1996, pp. 751-761.