Volume 6, Issue 5, May 2016 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Modularity Index for Open Source Software Systems

Harmanreet Kaur*
CSE, Guru Nanak Dev University, Punjab, India

Parneet Kaur
CSE, Desh Bhagat University, Punjab, India

Deepinderjeet Kaur
CSE, Desh Bhagat University, Punjab, India

Abstract— Modularity has been identified as one of the primary factors contributing to the success of Open Source Projects. Ample number of tools are available which used to calculate different size metrics such as LocMetrics to count number of lines of code but no tool to calculate Modularity Index metric and other metrics under one roof. The present literature depicted the quantitative measure of very limited number of metrics for a few Object Oriented Open Source Software Systems. There is no tool yet which calculates the Modularity Index of a Software System, only theoretical formulae exist till now. Not much effort has been focused on the metrics of high modular system metrics like WMC (Weighted Method Count), CBO (Coupling between Object Classes), RFC (Response for a Class), LCOM4 (Lack of Cohesion Metrics).This paper focuses on developing of a tool which calculates Modularity Index metric for open source java based projects. The study is done to enhance the previous Modularity Index metric which was based only on the number of packages of a project. The study emphasize on calculating Modularity Index based on the number of packages as well as the size of each package. This leads to the increased efficiency of MI metric. The new tool for calculating MI can be simulated in object oriented software project such as C++, C#, VB or PHP. The results can be clearly proven with the help of quantitative analysis and the graphical representation of Modularity Index value based on various projects.

Keywords— Open Source Software Projects; Modularity; Modularity Index; Java; Package Cohesion, Package Coupling, Complexity, Fan In, Fan Out.

I. INTRODUCTION

Modular design is a design approach that subdivides a system into smaller parts called modules or skids, that can be independently created and then used in different systems. A modular system can be characterized by functional partitioning into discrete scalable, reusable modules. Modularity is directly related to software architecture, since modularity is separation of a software system in independent and collaborative modules that can be organized in software architecture. It has been identified by many studies as one of the key factors of the success of Open Source Projects, but how modularity should be achieved and measured systematically is not yet known. Modularity has been identified by many researchers as one of the success factors of Open Source Software (OSS) Projects.

These modularity traits are influenced by some aspects of software metrics such as size, complexity, cohesion, and coupling/dependency, cohesion, fan in, and fan out.

A. The Intuitive View of Quality

Intuitively, software quality is about a computer program working “as it is supposed to”. Customer satisfaction is a key ingredient [Roy90, MäM06]. In other words, there is an element of expectancy that the program does not act in ways which surprises the user.

An intuitive view of quality concerns not only the user, but also the designer and writer of software. To the designer, conceptual elegance is sometimes considered high quality. A well designed class hierarchy does not break or lose its logical composition when introducing new classes as the program evolves. The task of programming according to specifications becomes possible, and there is little need to redesign the program. Programming tasks are easily divided among the programmers.

To the writer of software, quality goes beyond the design and into the lowest levels of the program. Each line of code can have an aesthetic quality. It can be easy to understand, making the code fluent to read and thus easy to modify. However, other constraints, such as efficiency requirements, might lead to code that is hard to read but performs exceptionally well. This is also an aspect of quality, as is the ability to write terse, compact code – including as much functionality as possible in as few statements as possible. It is impossible to dismiss any such view of quality without first establishing what is to be accomplished.

To both designers and writers of software, it seems that ease of change, or flexibility, is a key quality criterion on the intuitive level [Roy90]. It is a prerequisite for ongoing development. Unless the software can be practically changed, there is a real risk of financial loss to users and developers.

Since quality is such a multi-faceted concept, a rigorous definition is needed to remove the ambiguities and allow a group of people to work toward the same, known quality goals [Sto90]. This view of quality must be measurable, otherwise it is impossible to know whether or not quality has been achieved.

This paper explains the need of developing of a tool which calculates Modularity Index metric for open source projects based on the number of packages as well as the size of each package.

B. Software Metrics

Table I Description of Parameters and their Symbols

Parameter	Symbol	Description	Minimized/ Maximized
Modularity Index	M	Parameter indicating the quality of a software.	Maximized
Size	S	Size of each individual module, may be stated as LOC or FP(Function Point)	Minimized
Number of Modules	N	Number of Modules in the system,may be stated as the number of objects,headers etc.	Maximized
Complexity	X	The Internal Structure of the module, such as the decision structure, number of operators etc.	Minimized
Cohesion	H	Internal–interdependency inside module, which measures the semantic strength of relationships between components within a functional module is usually stated as high cohesion or low cohesion only.	Maximized
Fan in	F_1	One form of dependency/coupling, in which the class or object is being used by many classes or objects, which is the number of modules that call a given module.	Minimized
Fan out	F_0	The other form of dependency/coupling, in which it shows the ability of a class or object to be reused, which is the number of modules that called by a given module.	Maximized

C. Modularity Index:

A single quantitative measure of modularity level, which is proposed to be called Modularity Index, is needed to unify all of those parameters and it may also give insight about how certain level of modularity can be achieved. The value of the modularity metrics always increases without upper border, since some of the parameters which correlate to the Modularity Index are also having no upper bound, such as size, complexity, number of modules, etc.

1) *Class Level Modularity:* There are three software metrics that determine the level of modularity in class level, which are:

Size Metrics which consists of: NCLOC, Lines, and Statements. NCLOC is the number of non-commenting lines of code.. The LOCQ is a normalized value that determines the quality of a class based on the number of Non-Commenting Lines of Code (NCLOC) in the class

$LOC_Q = 0.0138\ NLOC + 0.310$ for $NCLOC \leq 50$

$LOC_Q = 1 / (NCLOC - 50)^{1.969}$ for $NCLOC > 50$

Where: LOCQ = LOC Quality

NCLOC = Non-Commenting Lines of Code

The number of functions / methods in the class. This may indicate the complexity.

$F_Q = 0.1836\ F + 0.0820$ for $F \leq 5$

$F_Q = 1/ (F - 4.83)^{2.691}$ for $F > 5$

Where:

F_Q = Function Quality

F = Function/Method

Cohesion refers to the degree to which the elements of a module belong together.[1] Thus, it is a measure of how strongly related each piece of functionality expressed by the source code of a software module is.LCOM4 or Lack of Cohesion Method version 4, Cohesion is determined by the value of LCOM4. The ideal value is 1 which means that the class is highly cohesive. Higher value of LCOM4 indicates the degree of needed separation of classes into smaller classes.

$H_Q = 1/ LCOM4^{2.216}$

Where:

H_Q = Cohesion Quality

LCOM4 = Lack of Cohesion Metrics 4

$c_Q = 0.25\ LOC_Q + 0.25\ F_Q + 0.5\ H_Q$

Where:

$_{CQ}$ = Class Quality

LOC_Q =LOC Quality

F_Q = Function Quality

H_Q =Cohesion Quality

2) *Package Level Modularity:* A package is a namespace that organizes a set of related classes and interfaces. Conceptually you can think of packages as being similar to different folders on your computer. Package Quality or PQ is the quality of individual package. Since in a single package there are many classes and there is no similarities

found the the optimal number of classes in each package, so the Package Quality is determined by the average Class Quality.

$$P_Q = \frac{\sum_{i=1}^{j} C_{Qi}}{\sum_{i=1}^{j} C_{Qi}}$$

Where :
P_Q = Package Quality
c_{Qi} = i-th class Quality
c_i = i-th class

3) *System Level Modularity:* S is a normalized value (with maximum value of 1) which determine the value of software architecture. The factors that influence this value are Package Cohesion (relationship among classes within package) and Package Coupling (relationship among classes from different packages). The principle used here is "Maximize Cohesion and Minimize Coupling" which becomes a widely known principle in building a good software system.

$$S_A = \frac{\sqrt{\sum_{i=1}^{d} c_{ii}^{2}}}{\sqrt{\sum_{i=1}^{d} \sum_{j=1}^{d} c_{ij}^{2}}}$$

Where :
S_A = System Architecture
C_{ii} = Package Cohesion
C_{ij} = Package Coupling (if i=1)

4) *Formulation of Modularity Index*

Modularity Index is the product of SA and the sum of all package quality in the software system as stated in the following formula:

$$M_I = S_A \frac{\sum_{i=1}^{j} P_{Qi}}{\sum_{i=1}^{j} P}$$

Where:
M_I = Modularity Index
S_A = System Architecture
P_{Qi} = i-th Package Quality
P_Q = i-th Package

II. LITERATURE SURVEY

A growing number of studies contribute to our understanding of the new software metric for open software systems (Emanuel; Wardoyo, 2011; Surjawan, 2012; Sharma, 2014).Sharma, et al. (2014) [1] in the paper "Empirical Analysis of Object Oriented Metrics using Dimensionality Reduction Techniques" suggested Quality is a critical factor of software, as its absence results in financial loss and can also endanger lives. In this study, they evaluated twelve object-oriented metrics proposed by various researchers. This study used an open source data to evaluate the metrics and two dimension reduction techniques namely, Principal Component Analysis and Principal Axis Factoring to eliminate the metrics providing redundant information. Patidar, et al. (2013) [22] in the paper "Coupling and Cohesion Measures in Object Oriented Programming" have discussed an idea on how to reduce coupling in object oriented programming. It is helpful for the developers to check which concept is best between inheritance and interface. It was reported that relating the measures is a difficult task in most of the cases and especially to conclude for which applications they can be used.
There Algorithm consist of four phases

1) Authentication
2) Select two Object Oriented Programming Files
3) Count no of Classes, Object and Inheritance
4) Based on the analysis provided in the database we deduce that which programming approach is better in the current situation.

Perez ,et.al (2013) [9] in the paper "Intensive Metrics for the Study of the Evolution of Open Source Projects: Case Studies from Apache Software Foundation Projects" presented the concept of invariant metrics in the domain of software projects, in particular with to software evolution. They discussed three simple metrics that have the property of being scale invariant regarding to project size, community size, global activity and project age, thus being good candidates for an intensive metrics. The output is based on the empirical evidence that the ratio of email messages in public mailing lists to versioning system commits has remained relatively constant along the history of the Apache Software Foundation (ASF).
Shatnawi, et.al (2013) [7] in the paper "Empirical study of fault prediction for open-source systems using the Chidamber and Kemerer metrics" proposed a new dependent variable is to improve the quality of the data sets.The dependent variable has four different categories of faults instead of the commonly used categories.Successful software projects

require implementing quality assurance plans that lead to successful software. However,poor-quality software implies a higher software failure,especially in system operation. Software faultdetection is a major factor that leads to a successful project. Surjawan, et al. (2012) [3] in the paper "Revised Modularity Index to Measure Modularity of OSS Projects with Case Study of Freemind" proposed to measure the modularity level of a java- based OSS Projects. To show its effectiveness in analysing OSS Project, the Modularity Index and its supporting software metrics are then used to analyse the evolution of Freemind mind mapping OSS Project. The analysis using Modularity Index and its supporting metrics shows the strength and weaknesses of the Freemind OSS Projects.

Formula

$$M_I = S_A \frac{\sum_{i=1}^{j} P_{Qi}}{\sum_{i=1}^{j} P}$$

Where:

M_I = Modularity Index

S_A = System Architecture

P_{Qi} = i-th Package Quality

P_Q = i-th Package

Santos, et al (2012) [8] in the paper "The attraction of contributors in free and open source software projects" they have developed a theoretical model to explore the contextual and causal factors of project attractiveness in inducing activities such as source code contribution, software maintenance, and usage. The main findings include that set of conditions such as license restrictiveness and their available resources provide the context that directly influence the amount of work activities observed in the projects. It was also found that indirect and unintended contributions such as recommending software, despite of being non-technical, cannot be ignored for project activeness, diffusion and sustainability. Finally, the analysis provided evidence that higher attractiveness leads to more code-related activities with the downside of slowing down responsiveness to address projects' tasks, such as the implementation of new features and bug fixes.Wardoyo, et al. (2011) [3] in the paper "Modularity Index Metrics for Java-Based Open Source Software Projects" presents the first quantitative software metrics to measure modularity level of Java-based OSS Projects called Modularity Index. This software metrics is formulated by analysing modularity traits such as size, complexity, cohesion, and coupling. These OSS Projects are selected since they have been downloaded more than 100K times. The software metrics related to modularity in class, package and system level of these projects are extracted and analysed. The similarities found are then analysed to determine the class quality, package quality, and then combined with system architecture measure to formulate the Modularity Index.

Formula

$$M_I = S_A \times \sum_{i=1}^{j} P_{Qi}$$

Where:

M_I = Modularity Index

S_A = System Architecture

P_{Qi} = i-th Package Quality

Istiyanto, et al. (2011) [6] in the paper "Statiscal Analysis On Software Metrics Affecting Modularity In Open Source Software " have identified modularity as one of the success factors of Open Source Software (OSS) Projects. In the research, they analysed the software metrics such as Size Metrics (NCLOC, Lines, and Statements), Complexity Metrics (McCabe's Cyclomatic Complexity), Cohesion Metrics (LCOM4), and Coupling Metrics (RFC, Afferent coupling and Efferent coupling) of Java-based OSS Projects. The software metrics reflecting the modularity of these projects are collected and then statistically analysed. It can be shown that there are only three independent metrics reflecting modularity which are NCLOC, LCOM4, and Afferent Coupling, whereas there is also one inconclusive result regarding Efferent Coupling.Olbrich ,et .al(2010) [10] in the paper "The Evolution and Impact of Code Smells: A Case Study of Two Open Source Systems" identifed different phases in the evolution of code smells during the system development and that code smell infected components exhibit a different change behavior. Code smells are design flaws in object-oriented designs that may lead to maintainability issues in the further evolution of the software system.This information is useful for the identification of risk areas within a software system that need refactoring to assure a future positive evolution.Zhu , et.al (2011) [24] in the paper "Monitoring Software Quality Evolution by Analyzing Deviation Trends of Modularity Views " discussed an approach for monitoring design quality evolution of software systems in long-term evolution by analyzing the deviation trends of different modularity views. The study largely indicated the status of design quality evolution, especially the trends of quality degradation. And the continuous monitoring of deviation trends provides useful feedback for the future evolution decisions.Farooq,et.al (2011) [17] presented insights into quality practices of open source software projects which affects the quality of OSS in a negative manner. Avoiding such practices and using proven quality management practices can result in high quality OSS. OSS quality is an open issue and it should continue striving for even better quality levels if it has to outperform traditional, closed source development and target corporate and safety critical systems.Emanuel, et al. (2006) [7] has explored a new Software Metrics called Modularity Index and analysed for the first time. This metrics could be used as a single measure to the modularity level of Open Source Projects. The result of the analysis is the general formula for Modularity Index in a form of matrix relation. It can be shown from the analysis that the possibility of uniting many of these attributes into this single measure called Modularity Index is highly feasible.

III. RESULT

$M_d = H. P$

Matrix H: Matrix of interest to be solved.

Stamelos ,et.al (2002) [4] in the paper “Code quality analysis in open source software development” have discussed Proponents of open source style software development claim that better software is produced using this model compared with the traditional closed model. They have measured quality characteristics of 100 applications written for Linux, using a software measurement tool, and compared the results with the industrial standard that is proposed by the tool and investigated the issue of modularity in open source of open source for this type of software development. Empirically assessed the relationship between the size of the application components and the delivered quality measured through user satisfaction.

IV. METHODOLOGY

The study shows that the Modularity Index can be calculated in three levels namely class level, package level and system level which is described below with the help of flow chart in Fig.1. These steps can be followed to calculate Modularity Index of Open Source Software Systems considering all important software metrices including Cohesion, Coupling, number of packages as well as the size of each package.

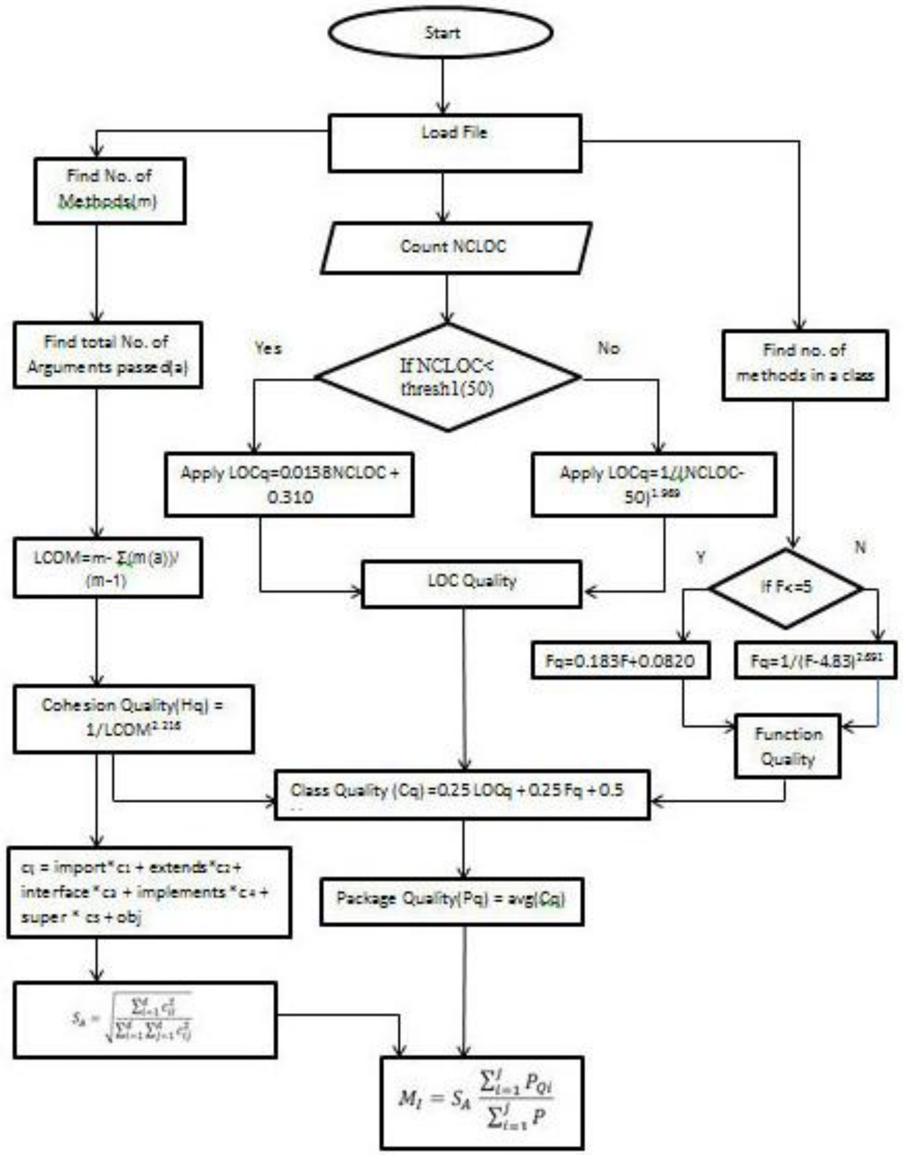
Start
Load File
Find No. of Methods(m)
Count NCLOC
Find total No. of Arguments passed(a)
Yes
If NCLOC< thresh1(50)
No
Find no. of methods in a class
Apply LOCq=0.0138NCLOC + 0.310
Apply LOCq=1/(NCLOC-50)^1.989
LCOM=m- Σ(m(a))/(m-1)
LOC Quality
Y
If F<=5
N
Fq=0.183F+0.0820
Fq=1/(F-4.83)^2.691
Cohesion Quality(Hq) = 1/LCOM^2.216
Function Quality
Class Quality (Cq) =0.25 LOCq + 0.25 Fq + 0.5
cl = import*c1 + extends*c2 + interface*c3 + implements*c4 + super*c5 + obj
Package Quality(Pq) = avg(Cq)
$S_A = \sqrt{\frac{\sum_{i=1}^{d} c_{ii}^2}{\sum_{i=1}^{d}\sum_{j=1}^{d} c_{ij}^2}}$
$M_I = S_A \frac{\sum_{i=1}^{j} P_{Qi}}{\sum_{i=1}^{j} P}$

Fig. 1 Methodology of the proposed scheme

V. CONCLUSION

This research focused on developing of a tool which calculates Modularity Index metric for open source java based projects and has enhanced the previous Modularity Index metric which was based only on the number of packages of a project.The proposed version calculates Modularity Index based on the number of packages as well as the size of each package. This leads to the increased efficiency of MI metric. The new tool for calculating MI can be implemented in any of the Object Oriented Languages such as C++, C#, VB, PHP etc. The results can be clearly proven with the help of quantitative analysis and the graphical representation of Modularity Index value based on various projects.

REFERENCES

[1] Zhu, Tianmei, Yijian Wu, Xin Peng, Zhenchang Xing, and Wenyun Zhao. "Monitoring software quality evolution by analyzing deviation trends of modularity views." In Reverse Engineering (WCRE), 2011 18th Working Conference on, pp. 229-238. IEEE, 2011.

[2] Wardoyo, Emanuel and J.Istiyanto. "Modularity Index Metrics for Java-Based Open Source Software Projects." , (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 2, No. 11, 2011.

[3] Surjawan D.J, (2012), "Revised Modularity Index to Measure Modularity of OSS Projects with Case Study of Freemind", International Journal of Computer Applications (IJCA), Vol. 59, No. 12, , pp 105 - 118, December 2012.

[4] Stamelos, Ioannis, Lefteris Angelis, Apostolos Oikonomou, and Georgios L. Bleris. "Code quality analysis in open source software development."Information Systems Journal 12, no. 1 (2002): 43-60.

[5] Sharma, Ritu, Sangeeta Sabharwal, and Shruti Nagpal. "Empirical analysis of object oriented metrics using dimensionality reduction techniques." In Recent Advances and Innovations in Engineering (ICRAIE), 2014, pp. 1-5. IEEE, 2014.

[6] Sharma, Rashmi, Sangeeta Sabharwal, and Sushma Nagpal. "Empirical analysis of object oriented metrics using dimensionality reduction techniques." In Recent Advances and Innovations in Engineering (ICRAIE), pp. 1-5. IEEE, 2014.

[7] Shatnawi, Raed. "Empirical study of fault prediction for open-source systems using the Chidamber and Kemerer metrics." Software, IET 8, no. 3 (2014): 113-119.

[8] Santos, Carlos, George Kuk, Fabio Kon, and John Pearson. "The attraction of contributors in free and open source software projects." The Journal of Strategic Information Systems 22, no. 1 (2013): 26-45.

[9] Roy, Sudipta, Sanjay Nag, Indra Kanta Maitra, and Samir K. Bandyopadhyay. "International Journal of Advanced Research in Computer Science and Software Engineering." International Journal 3, no. 6 (2013).

[10] Olbrich, Steffen, Daniela S. Cruzes, Victor Basili, and Nico Zazworka. "The evolution and impact of code smells: A case study of two open source systems." In Proceedings of the 2009 3rd international symposium on empirical software engineering and measurement, pp. 390-400. IEEE Computer Society, 2009.

[11] Nakagawa E.Y, de Sousa E.P.M de Britto Murata K. (2008), "Software architecture relevance in open source software evolution: a case study", Annual IEEE International Computer Software and Application Conference, pp 1234 – 1239, 2008.

[12] Midha, Vishal, and Prashant Palvia. "Retention and quality in open source software projects." AMCIS 2007 Proceedings (2007): 25.

[13] Mustofa K**,"** Modularity Index to Quantify Modularity of Open Source Software Projects." 2009. 5th International Conference on Information & Communication Technology and Systems, pp. 1-5. IEEE, 2009.

[14] Istiyanto J.E ,Wardoyo R , Mustofa K. (2010), "Success factors of OSS projects from sourceforge using datamining association rule", Proceeding of 2010 International Conference on Distributed Frameworks for Multimedia Applications (DFmA), pp 141 – 148, 2010.

[15] Gyimothy, Tibor, Rudolf Ferenc, and Istvan Siket. "Empirical validation of object-oriented metrics on open source software for fault prediction." Software Engineering, IEEE Transactions on 31, no. 10 (2005): 897-910.

[16] Ganpati, Anita, Arvind Kalia, and Hardeep Singh. "A Comparative Study of Maintainability Index of Open Source Software." International Journal of Software and Web Sciences 3, no. 2 (2012): 69-73.

[17] Farooq, Sheikh Umar, and S. M. K. Quadri. "Quality Practices in Open Source Software Development Affecting Quality Dimensions." Trends in Information Management 7, no. 2 (2012).

[18] Emanuel A.W.R, Wardoyo R., Istiyanto J.E., Mustofa K. (2011), "Statistical analysis on software metrics affecting modularity in open source software", International Journal of Computer Science and Information Technology (IJCSIT), Vol. 3, No. 3, pp 105 – 118, June 2011.

[19] Emanuel, Andi Wahju Rahardjo, Khabib Mustofa, and Jl Prof Drg Suria Sumantri. "Modularity Index to Quantify Modularity of Open Source Software Projects." In Proceedings of the 5th International Conference on Information & Communication Technology and Systems (ICTS),Gurbani, VK, Garvert, A., and Herbsleb, JD, pp. 1-6. 2006.

[20] Emanuel A.W.R., Wardoyo R., Istiyanto J.E., and Mustofa K., "Modularity Index Metrics for Java-Based Open Source Software Projects", International Journal of Advanced Computer Science and Applications (IJACSA) Vol. 2 No. 11, November 2011.

[21] Di Maio, Paola. "An open ontology for open source emergency response system." Open Source Research Community (2007).

[22] Capra E., Francalanci C., Merlo F. (2008), "An Empirical Study on the Relationship among Software Design Quality, Development Effort, and Governance in Open Source Projects", IEEE Transactions on Software Engineering, Vol. 34, No. 6, pp 765 - 782, Nov/Dec 2008.

[23] Aberdour, Mark. "Achieving quality in open-source software." Software, IEEE24, no. 1 (2007): 58-64.

[24] Aruna M., M.P. Suguna Devi M.P, Deepa M. (2008), "Measuring the Quality of Software Modularization using Coupling-Based Structural Metrics for an OOS System", Proceeding of the First International Conference on Emerging Trends in Engineering and Technology 2008.1