Volume 2, Issue 8 August 2016 ISSN NO: 2454 - 7514

North Asian International Research Journal Consortium

North Asian International Research Journal

Of

Science, Engineering and Information Technology

IRJIF IMPACT FACTOR: 3.821

NAIRJC JOURNAL PUBLICATION

North Asian International Research Journal Consortium

Welcome to NAIRJC

ISSN NO: 2454 -7514

North Asian International Research Journal of Science, Engineering & Information Technology is a research journal, published monthly in English, Hindi. All research papers submitted to the journal will be double-blind peer reviewed referred by members of the editorial board. Readers will include investigator in Universities, Research Institutes Government and Industry with research interest in the general subjects

Editorial Board

Address: -North Asian International Research Journal Consortium (NAIRJC) 221 Gangoo, Pulwama, Jammu and Kashmir, India - 192301, Cell: 09086405302, 09906662570, Ph. No: 01933-212815,
Email: nairjc5@gmail.com, nairjc@nairjc.com, info@nairjc.com Website: www.nairjc.com

IRJIF IMPACT FACTOR: 3.821

AN IMPROVED TECHNIQUE TO MEASURE MODULARITY INDEX FOR OPEN SOURCE SOFTWARE SYSTEMS

PARNEET KAUR
Department of Computer Science and Engineering, Desh Bhagat University

HARMANREET KAUR
Department of Computer Science and Engineering, Guru Nanak Dev University

DEEPINDERJEET KAUR
Department of Computer Science and Engineering, Desh Bhagat University

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	Maximized

		module is usually stated as high cohesion or low cohesion only.	
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. 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.

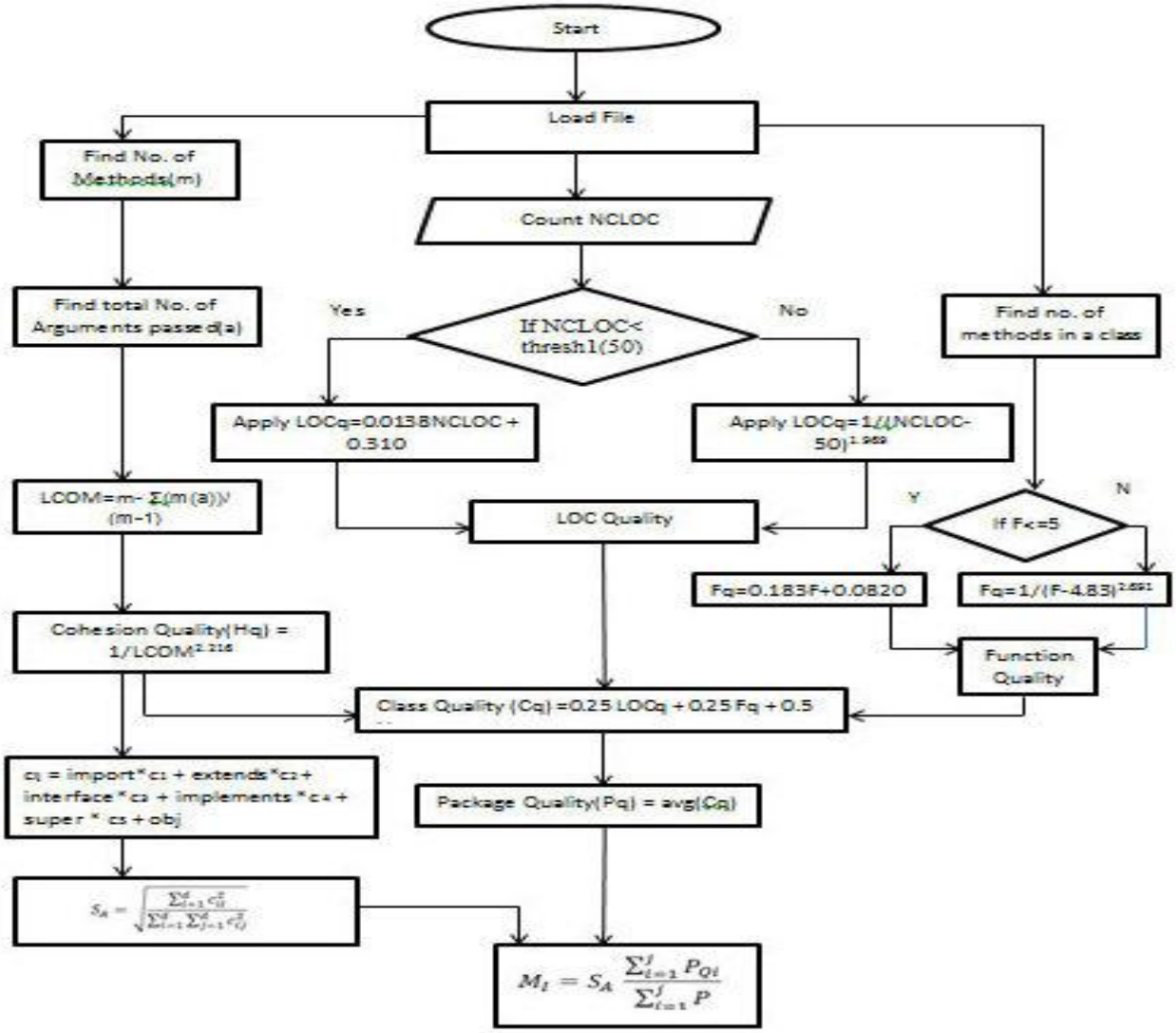

Fig. 2.1 Methodology of the proposed scheme

III. RESULTS

Quantitative Results

Quantitative results of three different open source software java based projects have been depicted. Each having variable file size. The Class quality, Package Quality, Cohesion, System Level Modularity and Modularity Index has been shown in the following table:

Sample for Metrics Evaluation- Project Neuroph

Table 3.1 Details of Metrics of Neuroph

333 Sno.	File Name	Cq	Pq	Cohesion	Sa	MI	Size (IN KB)
1	Adaline.java	0.367886033	0.367886033	0.808431041	0.911116306	0.335187	4
2	And.java	1.42235	0.711175	1	1.04	0.739622	3
3	AndTest.java	0.032771949	0.032771949	0.916025826	0.980436098	0.0321308	4
4	AnimalsClassifiactionSample.java	0.206877633	0.206877633	0.996411489	1.012670858	0.209499	9
5	Apptest.java	0.497352225	0.248676112	0.668462485	0.849041534	0.2111363	1
6	BackPropagation.java	0.417294326	0.417294326	0.96808154	1.013726843	0.4230225	4
7	BalanceScaleSample.java	0.206877633	0.206877633	0.996411489	1.014304198	0.2098369	5
8	BAM.java	0.51361112	0.25680556	0.634854799	0.807401464	0.2073452	4
9	Benchmark.java	0.451794326	0.451794326	0.96808154	1.147896572	0.5186132	3
10	BiasNeuron.java	0.4379	0.4379	1	0.962962963	0.4547423	2
11	BinaryDeltaRule.java	0.417294326	0.417294326	0.96808154	1.035696155	0.4321901	4
12	BinaryHebbianLearning.java	0.289342397	0.289342397	0.934151643	1.017384427	0.2943724	2
13	BufferedDataSet.java	0.601662371	0.300831186	0.988160959	1.005757712	0.3025633	8
14	CharExtractor.java	0.024325435	0.024325435	0.948602608	0.997717502	0.0242699	15
15	CompetitiveLayer.java	0.251879699	0.251879699	1	1.03030303	0.2595124	4
16	ConcreteStrenghtTestSample.java	0.208287408	0.208287408	0.996411489	1.014304198	0.2112668	5
17	Connection.java	0.00437391	0.00437391	0.990324202	1.071700368	0.0046875	7
18	ConnectionFactory.java	0.002252002	0.002252002	0.998778101	1.023764202	0.0023055	8

19	DecimalScaleNormalizer.java	0.205225828	0.205225828	0.996411489	1.074989065	0.2206155	4
20	DelayedConnection.java	0.382140181	0.382140181	0.998040236	1.03744347	0.3964488	3
21	DelayedNeuron.java	0.296242397	0.296242397	0.934151643	0.990678086	0.2934809	3
22	Difference.java	0.323842397	0.323842397	0.934151643	0.995803546	0.3224834	2
23	DifferenceTest.java	0.375011213	0.375011213	0.994762025	1.021703856	0.3831504	2
24	Dimension.java	0.42065	0.42065	1	1.166666667	0.4907583	2
25	DistortRandomizer.java	0.336894326	0.336894326	0.96808154	1.013726843	0.3415188	3
26	DynamicBackPropagation.java	0.004045982	0.004045982	0.984717593	1.044557239	0.0042263	8
27	ElmanNetwork.java	0.339661403	0.339661403	0.993963527	1.011638624	0.3436146	3
28	FileInputAdapter.java	0.281636033	0.281636033	0.808431041	0.923428689	0.2600708	2
29	FileOutputAdapter.java	0.281636033	0.281636033	0.808431041	0.91956266	0.258982	2
30	FileUtils.java	0.156820175	0.156820175	0.934151643	0.986650939	0.1547268	3

Sample for Metrics Evaluation- Project Smark

Table 3.2 Details of Metrics of Smark

Sno.	File Name	Cq	Pq	Cohesion	System Level Modularity	MI	Size of File
1	BasicBlock.java	0.07164266	0.03582133	0.8328334	1.09511646	0.03922853	7
2	Edge.java	0.31805129	0.31805129	0.8943436	1.1348369	0.36093634	1
3	EmptyMethodException.java	0.2410424	0.2410424	0.9341516	1.15981825	0.27956537	1
4	PositiveIntSynthesizer.java	0.14086139	0.14086139	0.7243378	1.02129641	0.14386123	21
5	NullNENullCallGenerator.java	0.21214818	0.21214818	0.9680815	1.18069362	0.250482	13
6	CallingCallGenerator.java	0.15951581	0.15951581	0.994762	1.19685309	0.19091699	12
7	CodeContext.java	0.14170442	0.14170442	0.7243378	1.13477379	0.16080246	3
8	FallthroughEdge.java	0.23905983	0.23905983	0.7243378	1.02129641	0.24415094	1
9	Annotator.java	0.4379	0.4379	1	1.33333333	0.58386667	3
10	TracePoint.java	0.2504417	0.2504417	1	1.2	0.30053004	5

11	ArrayDummyExpr.java	0.31268603	0.31268603	0.808431	1.07895352	0.3373737	1
12	AllAlgsOnce.java	0.30460983	0.30460983	0.7243378	1.13477379	0.34566325	2
13	BranchExpr.java	0.30460983	0.30460983	0.7243378	1.02129641	0.31109692	2
14	CatalanNumbers.java	0.30805983	0.30805983	0.7243378	1.13477379	0.34957822	2
15	CaughtExceptionData.java	0.32999433	0.32999433	0.9680815	1.18069362	0.38962219	1
16	DFA.java	0.06694703	0.06694703	0.8616982	1.11393242	0.07457446	20
17	MonitorExpr.java	0.4517	0.4517	1	1.2	0.54204	2
18	Overseer.java	0.25003881	0.25003881	1	1.2	0.30004657	7
19	HashHolder.java	0.00033062	0.00033062	0.9995445	1.33302965	0.00044072	12
20	IfExpr.java	0.01685013	0.01685013	0.9454566	1.16681514	0.01966099	7
21	Folder.java	0.00076073	0.00076073	0.9989388	1.33262567	0.00101377	18
22	StoreExpr.java	0.48965	0.48965	1	1.2	0.58758	2
23	IncExpr.java	0.39659433	0.39659433	0.9680815	1.18069362	0.46825639	2
24	ThrowExpr.java	0.38969433	0.38969433	0.9680815	1.18069362	0.4601096	1
25	StringConstantExpr.java	0.32654433	0.32654433	0.9680815	1.18069362	0.3855488	1

Sample for Metrics Evaluation- Library Management System

Table 3.3 Details of Metrics of Library Management System

Sno.	Name of File	Cq	Pq	Cohesion	System Level Modularity	MI	Size of File
1	AddBooks.java	0.20427567	0.20427567	0.99940837	1.01508420	0.207357001	13
2	AddMembers.java	0.25002619	0.25002619	1	1.019607843	0.254928669	10
3	Books.java	0.1078089	0.1078089	0.61630217	0.84543754	0.091145693	5
4	Borrow.java	0.03263319	0.03263319	0.99483059	1.074135942	0.035052487	3
5	Center.java	0.2479424	0.2479424	0.93415164	1.0408625	0.258073951	1
6	JLibrary.java	0.00655894	0.00655894	0.99527049	1.017193862	0.006671712	9
7	ListSearchBoks.java	1.3640934	0.6820467	1	1.01388888	0.691519569	6
8	Main.java	0.24595983	0.24595983	0.72433775	1.134773793	0.279108766	1
9	Members.java	0.10834171	0.10834171	0.61772222	0.84641099	0.091701616	4
10	Menubar.java	0.14273953	0.14273953	0.72433775	0.872357353	0.124519876	4

11	PrintingBooks.java	0.15955708	0.15955708	0.99476203	1.0112300	0.161348917	6
12	PrintingMembers.java	0.15956072	0.15956072	0.99476203	1.01123004	0.161352591	5
13	ReturnBooks.java	0.9999805	0.49999025	8.75E-08	0.00029884	0.000149422	13
14	SearchBooksAndMembers.java	0.16189772	0.16189772	0.98531671	1.015191006	0.164357112	12
15	StatusBar.java	0.25285983	0.25285983	0.72433775	0.87687065	0.221725363	1
16	Toolbar.java	0.31840983	0.31840983	0.72433775	0.895874047	0.285255101	2
17	ListMembers.java	0.9408993	0.47044965	0.02217706	0.15098798	0.071032244	6
18	BorrowBooks.java	0.20428858	0.20428858	0.99940837	1.016940417	**0.207749315**	9

Sample Projects Summary

In the following table below we compare all the three different open source java based projects:

Table 3.4 Summary of Sample Projects

Sno.	Project NO.	Cq	Pq	Cohesion	System Level Modularity	MI
1	Neuroph	9.75793876	8.2404509	28.1365642	30.1230341	8.34237962
2	Smark	6.16143781	6.12561648	22.456594	29.1169931	7.12694616
3	Lib.management	5.90783332	4.25534672	14.0514628	15.858376	3.3130494

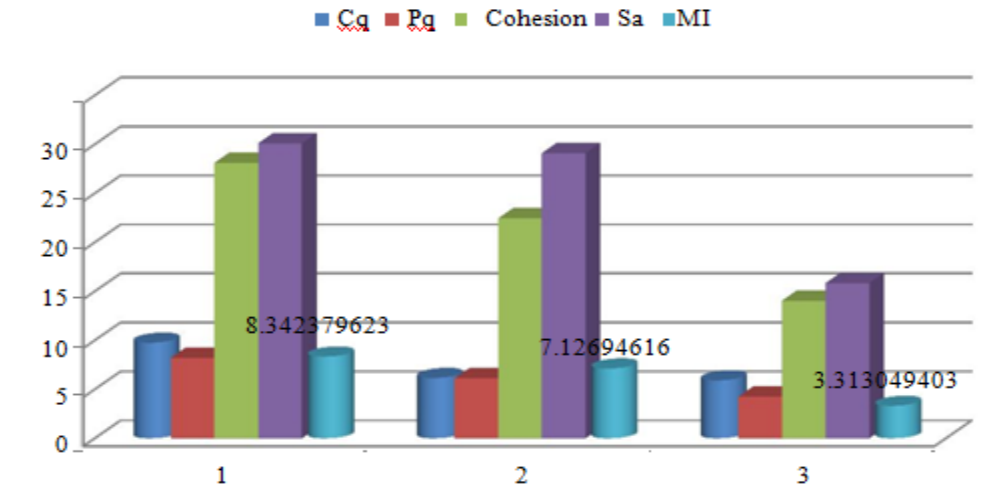

Figure 3.1 Graphical Representation of various parameters of projects

Comparison between Modularity Index:

In this we compare modularity index metric of different projects based on the number of packages and the size of package.

Case 1: While considering the number of Packages

Table 3.5 MI with respect to no. of packages

Sno.	No of Packages	MI	MI/No of Packages
1	30	8.342379623	0.278079321
2	25	7.12694616	0.285077846
3	18	3.31304903	0.184058279

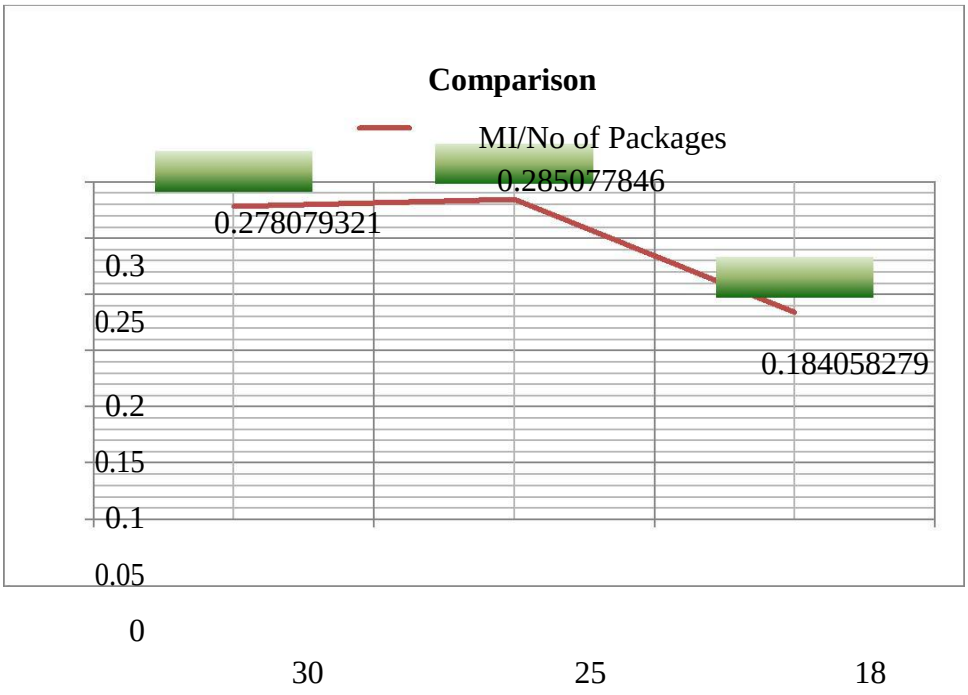

Figure 3.2 MI with respect to no. of packages

Case 2: When we increase the number of packages by introducing packages having bad smells or blank classes.

Table 3.6 MI with respect to increased no. of packages

Sno.	No. of Packages	MI	MI/total no of Packages
1	38	8.342379623	0.219536306
2	27	7.12694616	0.263960969
3	24	3.31304903	0.138043725

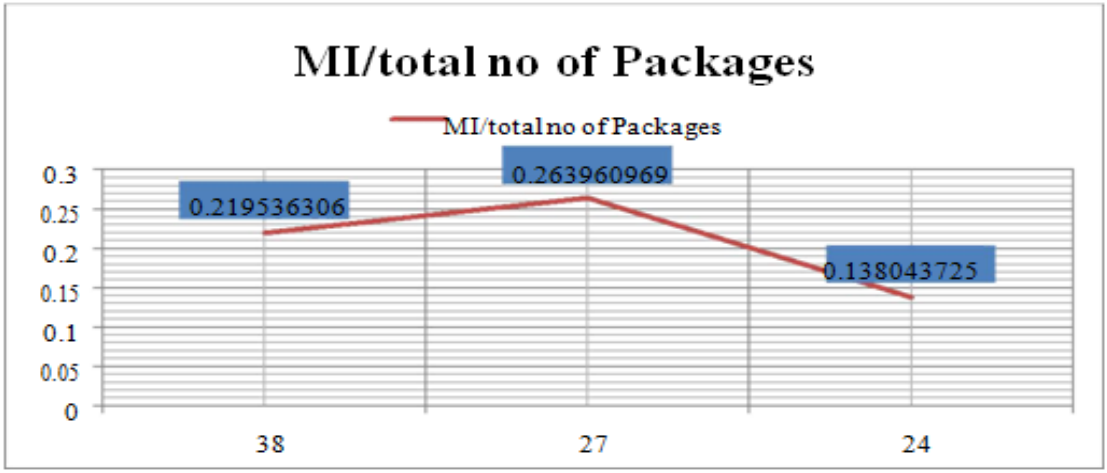

Figure 3.3 MI with respect to increased no. of packages

Case 3: When we further increased the number of packages, effect on Modularity Index.

Table 3.7 Overall effect on MI

Sno.	No. of Packages	MI/total no of Packages
1	30	0.278079321
2	25	0.285077846
3	18	0.184058279
4	50	0.166847592
5	35	0.203627033
6	27	0.12270552

MI/Total no. of packages

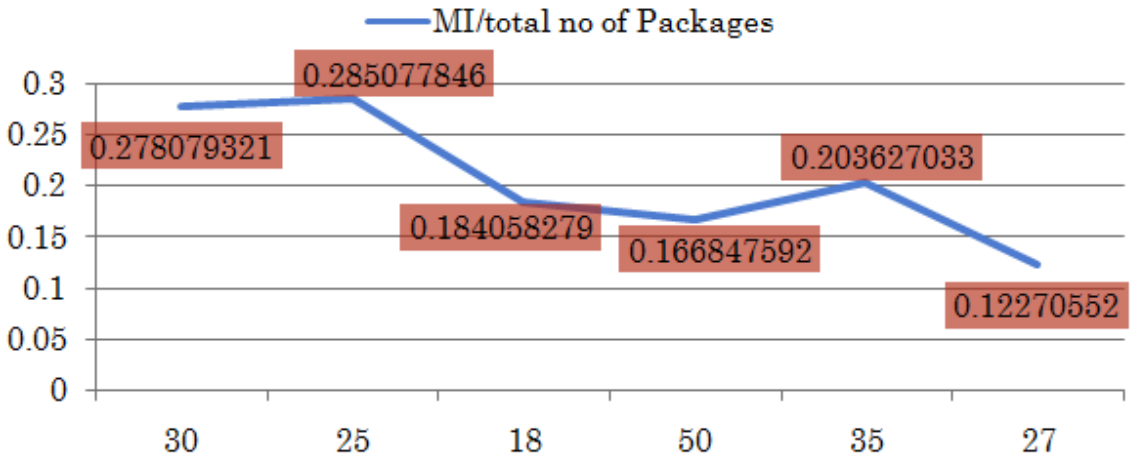

Figure 3.4 Overall effect on MI

Case 4: While considering number of packages and size of package.

Table 3.8 Effect on MI based on size and no. of Packages

No. of Packages	MI/total no of Packages
159	0.052467796
172	0.041435733
128	0.025883196
181	0.046090495
185	0.038524033
138	0.024007602

MI/total no of Packages

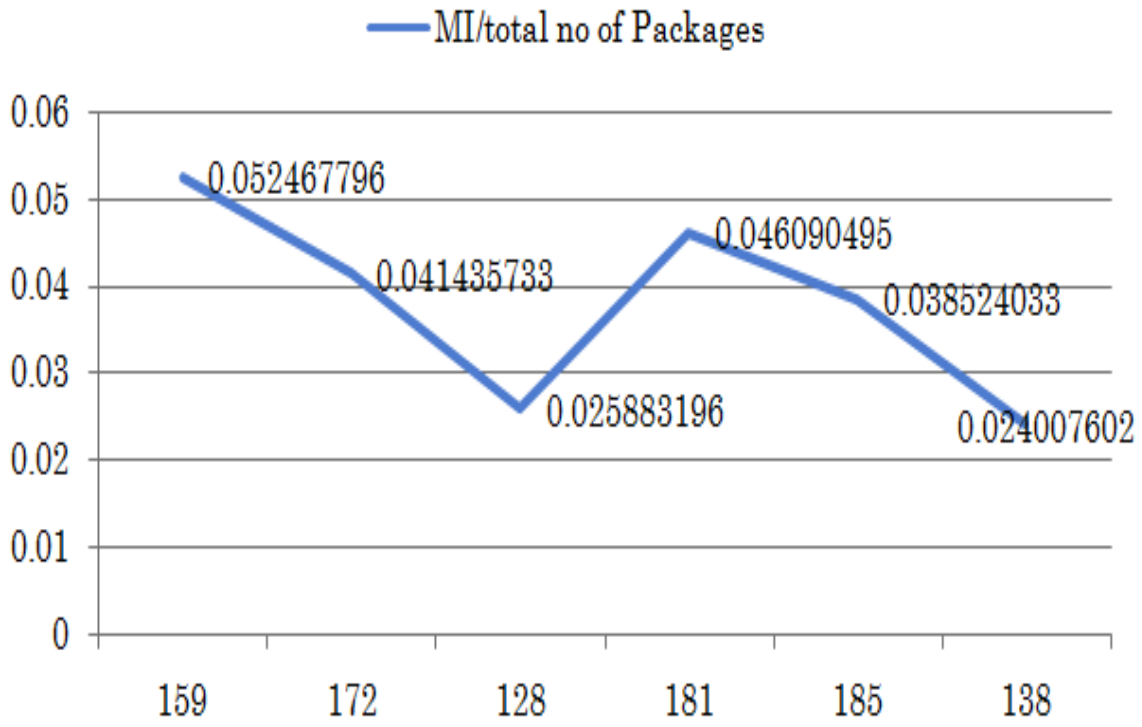

Figure 3.5 Effect on MI based on size and no. of Packages

Case 5: Comparison between MI while considering no. of packages and size.

Table 3.9 Values of MI

MI/No. of Packages	MI/(Packages +Package size)
0.111231728	0.006377301
0.081450813	0.0029117
0.06135276	0.001875594

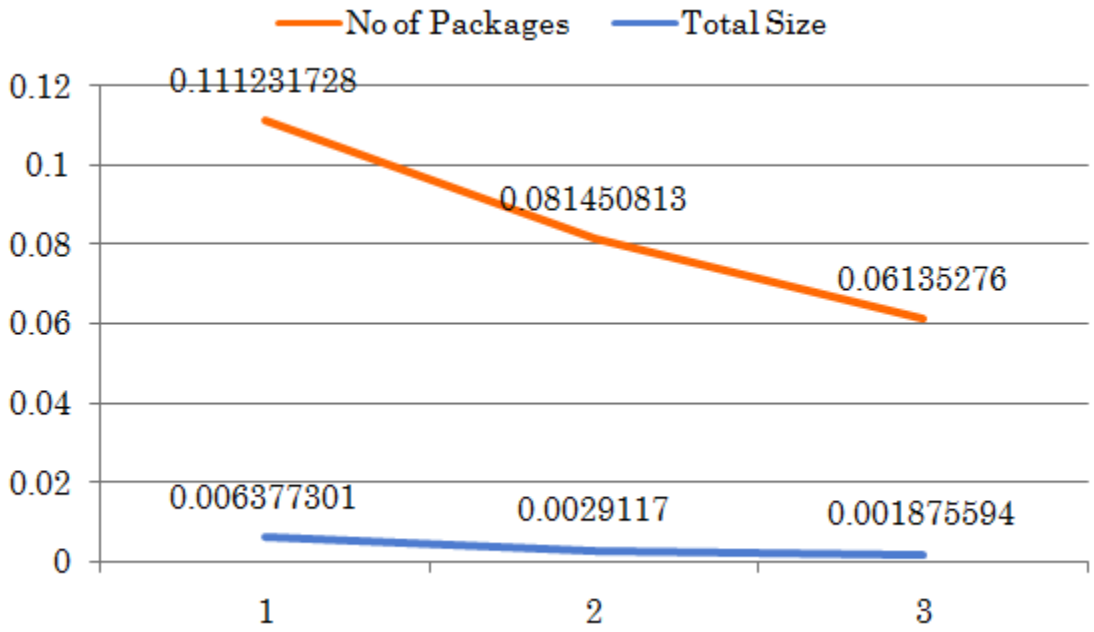

Figure 3.9. Comparison between Modularity Index

IV. 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] 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.

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

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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).

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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).

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

Publish Research Article

Dear Sir/Mam,

We invite unpublished Research Paper,Summary of Research Project,Theses,Books and Book Review for publication.

Address:- North Asian International Research Journal Consortium (NAIRJC)
221, Gangoo Pulwama - 192301
Jammu & Kashmir, India
Cell: 09086405302, 09906662570,
Ph No: 01933212815
Email:- nairjc5@gmail.com, nairjc@nairjc.com , info@nairjc.com
Website: www.nairjc.com