STATISTICAL ANALYSIS ON SOFTWARE METRICS AFFECTING MODULARITY IN OPEN SOURCE SOFTWARE

Andi Wahju Rahardjo Emanuel[1], Retantyo Wardoyo[2], Jazi Eko Istiyanto[2], Khabib Mustofa[2]

[1]Bachelor Informatics, Faculty of Information Technology, Maranatha Christian University, Indonesia
awreman@gmail.com
[2]Department of Computer Science and Electronics, Faculty of Math and Natural Sciences, Gadjah Mada University, Indonesia
{rw,jazi,khabib}@ugm.ac.id

ABSTRACT

Modularity has been identified by many researchers as one of the success factors of Open Source Software (OSS) Projects. This modularity trait are influenced by some aspects of software metrics such as size, complexity, cohesion, and coupling. In this research, we analyze 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 59 Java-based OSS Projects from Sourceforge.net. By assuming that the number of downloads can be used as the indication of success of these projects, the OSS Projects being selected are the projects which have been downloaded more than 100,000 times. The software metrics reflecting the modularity of these projects are collected using SONAR tool and then statistically analyzed using scatter graph, Pearson r product-moment correlation, and least-square-fit linear approximation. 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.

KEYWORDS

Open Source Software Project, Modularity, Software Metrics, Statistical Analysis, Java

1. INTRODUCTION

Open Source Software (OSS) Projects nowadays are gaining momentum worldwide and they are getting more attention not only from large corporations, but also from researchers. Once only considered an *ad hoc* software development process performed mainly by academics and freelance developers in their leisure times, they are now considered one major alternative in developing software challenging the already developed 'software engineering' methodology used in commercial or proprietary software projects. OSS Projects are attracting many large corporations such as Oracle, IBM, Sun Microsystem, etc. in which they are supporting some large OSS Projects such as Java, Eclipse, and many more. Some of the initially small OSS Projects are also evolved into large and complex projects and they eventually set up their own companies or foundations to solidify their organization, such as Red Hat, Apache, Mozilla, etc. Moreover, many studies are also being conducted using OSS Projects as the research object in order to find their success factors, evolutions, community developments, and their problem areas.

Some studies are able to discover the key success factors of the OSS Projects, and one of the important finding is that one of the key success factors in developing high quality OSS is the modular architecture of the system [10][14][17][26]. Even though modularity has been identified as one of the key success factor, how to apply the modularity principles in the early phase of OSS Projects is not yet clearly understood. The study of the modularity properties of small to medium sized OSS Projects that considered successful should give some understanding about this matter. This paper presents the statistical analysis of 59 Java-based OSS Projects from Sourceforge.net Portal that has been downloaded more than 100k times. These projects are samples of successful small-to-medium-sized OSS Projects developed using Java programming languages. The similarities of these OSS Projects in terms of their modularity properties (size, complexity, cohesion, and coupling) should give better understanding about the measurement and applications of modularity principles during early phase which will be the next target of the research.

This paper is organized as follows: section 2 will explain some current studies relating to the success and also the potential problem in OSS Projects. Section 3 provides some theoretical background relating to OSS Projects, Software Metrics, and Modularity in OSS Projects. Section 4 will present the result of the statistical analysis of the 59 OSS Projects. Section 5 will discuss the result with their interpretations. Lastly, section 6 is the conclusion which includes short summary, conclusion, and future studies.

2. CURRENT STUDIES ON OSS PROJECTS

Many studies have been conducted to identify the key success factors of OSS Projects and they can be categorized into three approaches. First approach of the study is by studying several successful OSS Projects, i.e. the study on Debian GNU/Linux [28], FreeBSD [11], Apache [23], and OpenBSD [18]. The second approach is trying to find the similarity in the processes in many successful OSS Projects i.e. the study of Apache and Mozilla [22], fifteen OSS Projects [29], and Arla and Mozila Projects [5]. The third approach focuses on process aspects such as team communication across for projects with with at least 7 developers and 100 reported bugs [9], the bug arrival rate of 8 OSS Projects [31], activities from CVS Repository [8], and OSS Projects' dependability [17]. All of these approaches have the same weakness since they involve only relatively small number of OSS Projects, leading to the consequence that they may not give good representation of other OSS Project that already numbered in hundreds of thousands [13] The authors have conducted research of more than 135 k OSS Projects using Datamining Association Rule to find the success factors of OSS Projects and able to propose 9 Success Factors [13].

Some other researches are able to identify some alarming potential problems in the OSS Projects and their communities. In certain phase of the projects, the software system will increase in complexity that makes it more difficult to be managed by the communities [14][28]. The ad hoc development of the Open Source System also creates possible decline in quality [28], coupling explosions [3], and creating poorly coded source code [12], lack of formal process [8], high entry barrier for new developer to contribute [2], the poor architectural design and lack of supporting tools which is comparable to modern software development methodology [11], and lack of documentation which prohibits new developers to immediately join in the projects [12]. As for the Open Source Communities themselves, the alarming problems are the frequent / rapid turnovers of volunteers [12] and the fact that only very few Open Source Projects attract enough support to develop properly [17]. The study of the cause of failure in OSS Projects is also useful, but finding the problem areas do not necessarily imply the solutions to the problem.

The current research are the continuation of the previous research [13] in which the subject area are focused on Java-based OSS Projects with high numbers of downloads (more than 100K

times). The high number of downloads may indicate the success of these OSS Projects, and by studying their commonalities in terms of software metrics reflecting modularity (size, complexity, cohesion, and coupling), some common properties may be found. These common properties may be implemented in the future Java-based OSS Projects to increase their possibility of success.

3. THEORETICAL BACKGROUND

The theoretical background of this paper includes short explanations about OSS Projects as well as the modularity in OSS Projects.

3.1. Open Source Software Project

OSS Project is a software development methodology based on several distinct characteristics not commonly found in commercial or proprietary software development:

- The source code is freely available for everybody to download, improve and modify [27]. The licensing scheme of the OSS System such as GPL will ensure the continual improvement of the applications by requiring everybody who improves the application to share them to everyone else.
- Person who contribute to the development of the OSS Projects is usually forming a group called Open Source Communities [8][9]. This community will share information to each other electronically using email, mailing list, forum etc., and they are seldom or may be never meet each other face to face. The recruitment process of the developers are completely voluntary and the hierarchy of the communities are determined by their loyalty to the project and their technical capabilities.
- The development methods of the OSS Projects are lacking of formal methodology found in commercially developed software applications. Their primary concern during the development are adding new features and fixing bugs[8]

Currently, many portals have been developed as a incubator for OSS developers to develop and host their projects. These portals are equipped with many development tools (version control, bug tracking, wikis, etc.) and statistics to assist the project initiator or administrator in improving their OSS Projects and other interested contributors to join the projects. Some of the popular portals are Sourceforge.net, freshmeat.net, launchpad.net, and Google Code (http://code.google.com).

3.2. Modularity in OSS Projects

Modularization involves breaking up of an software system into smaller, more independent elements known as module. Booch has defined modularity as the property of a system whose modules are cohesive and loosely-coupled [21]. Fenton stated that Modularity is the internal quality attribute of the software system [21]. It is also known that 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 a software architecture [25]. Modular software has several advantages such as maintainability, manageability, and comprehensibility [24], as well as ensuring the legitimate peripheral participation of new members [16].

There are five attributes closely related to modularity in software system which are size, coupling / dependency, complexity, cohesion, and information hiding. The first attribute is the size of the module as well as the system in which each module should not be to large in size and additional features in the system should be translated as the addition in the module of the system. The second attribute is coupling / dependency which consist of direct / syntactic which can be achieved through composition, method signatures, class instantiations, and inheritance[6]; and semantic or indirect coupling [19]. The third attribute is complexity that can

be measured by using software metrics such as McCabe's Cyclomatic Complexity or Halstead's Software Metrics [30]. The fourth attribute is cohesion which measure the integrity of the code inside each of the module. The term used to qualitatively measure cohesion are high cohesion or low cohesion. The last attribute is information hiding [8] which involves hiding the details of implementation from external modules.

Relating to the modularity property of a software system, in order to have an ideal modular software system, the software system should have the following attributes:

- Small size in each module (package) and many modules in the system. Each module / package should only responsible for simple feature, and the more complex features should be composed of many of these simple features. The possible software metrics to measure size are NCLOC, Lines, or Statements.
- Low coupling / dependency [4]: minimization or standardization of coupling / dependency e.g. through standard format i.e. published APIs [1], elimination of semantic dependencies, etc. The possible software metrics to measure coupling are Afferent Coupling, Efferent Coupling, or RFC (Response for a Class).
- Low complexity: hierarchy of modules that prefers flatter than taller dependency [21][1]. The most popular software metrics to measure complexity is cyclomatic complexity by McCabe [20].
- High cohesion: high integrity of the internal structure of software modules which is usually stated as either high cohesion or low cohesion. The better measure of cohesion in object oriented programming such as Java is LCOM4 or Lack of Cohesion Metrics version 4 proposed by Hitz and Montazeri [15]
- Open for extension and close to modification[4]: capability of the existing module to be extended to create a more complex module. And avoid changing already debugged code. The creation of new modules should be encourage using available extension and not modifying the already tested module.

This paper will address all of above characteristics except for the last one. The first four characteristics can be found in the module level (package level in Java-based OSS Projects) and they can be extracted from SONAR tool. Where as the last characteristics exist in system level which will be addressed in the next research.

4. STATISTICAL ANALYSIS

4.1. Data Collection Process

There are several consideration and assumption to select which OSS Projects to be analyzed:

- Small to medium size and Java-based OSS Projects. The limitation of the size (NCLOC) of OSS Projects being evaluated are 170K. Moreover, modularity will be a lot easier to comprehend in object-oriented programming (C++, Java, etc.) compared to procedural programming (C, Fortran, etc.), since the concept of module, coupling, cohesion, etc. are more straightforward. Java-based OSS Projects are selected since they are among the mostly popular object oriented programming for developing Open Source Software.
- The Projects should already be downloaded more than 100,000 times. This high number of downloads may indicate the 'success' of the projects, which in turn may imply modularity traits that already identified as the success factor of OSS Project.
- The source code of the OSS Project are free of error and compile-able. The SONAR tool requires that the source code should be compiled first using compile tool such as

maven, make, or ant. Many of the OSS Projects provides separate binary and source code and it is difficult to create binary directly from the source code due to several reasons such as compile error, build tool configuration error, syntax error, etc.

There are many metrics that are able to be collected using SONAR tool, but only several metrics are selected since they may indicate the level of modularity of the OSS Projects. These metrics are:

- *Size Metrics* which consists of :
  - *NCLOC*: the number of non-commenting lines of code. This metrics is the count of the source code's lines excluding the comment , empty lines, and white spaces.
  - *Lines*: the number of lines including white spaces, empty lines and comments in the source code.
  - *Statements*: the number of statements. In Java, a single statement will be ended with a semi colon.
- *Complexity Metrics*: Cyclomatic Complexity proposed by McCabe. This metrics measures the number of decisions caused by conditional statements in the source code [20].
- *Cohesion Metrics: LCOM4* or Lack of Cohesion Method version 4, this version is better for object oriented programming such as Java as proposed by Hitz and Montazeri [15] which is the improvement of LCOM1 Chidamber and Kemerer [7].
- *Coupling Metrics* which consists of*:*
  - *RFC*: Response for a Class. It is a set of method that may be executed in response to a message received of an object by that class. This metrics is first proposed by Chidamber and Kemerer [7].
  - *Afferent (incoming) Coupling*: the number of packages in which depend on classes within the package. Afferent coupling indicates package's responsibility.
  - *Efferent (outgoing) Coupling*: the number of packages in which the package depend upon. Efferent coupling indicates package's independence.

Table 1. shows the list of OSS Projects as a subject for this research. The initial OSS Projects to be evaluated are 209 projects, but only 59 which are suitable to be evaluated using SONAR due to the compile-ability consideration. There are total 1885 modules / packages being measured from these 59 OSS Projects.

Table 1. List of 59 OSS Projects

No	Project Name	No	Project Name	No	Project Name
1	FreeMind	21	Jajuk	41	FreeGuide TV Guide
2	jEdit	22	FreeTTS	42	Eteria IRC Client
3	TV-Browser - A free EPG	23	A Java library for reading/writing Excel	43	MeD's Movie Manager
4	JFreeChart	24	checkstyle	44	subsonic
5	JasperReports - Java Reporting	25	httpunit	45	kXML
6	OpenProj - Project	26	JMSN	46	Jaxe

No	Project Name	No	Project Name	No	Project Name
	Management				
7	HyperSQL Database Engine	27	PDFBox	47	The JUMP Pilot Project
8	yura.net	28	JBidwatcher	48	Aglet Software Development Kit
9	JabRef	29	JTidy	49	Antenna
10	FreeCol	30	Jena	50	CBViewer
11	jTDS - SQL Server and Sybase JDBC driver	31	Jin client for chess servers	51	Sunflow Rendering System
12	Torrent Episode Downloader	32	SAX: Simple API for XML	52	Thingamablog
13	FindBugs	33	jKiwi	53	BORG Calendar
14	PMD	34	Data Crow	54	Directory Synchronize Pro (DirSync Pro)
15	JGraph Diagram Component	35	Wicket	55	Java Treeview
16	ANts P2P	36	Cewolf - Chart TagLib Project	56	Java Network Browser
17	Paros	37	DrawSWF	57	Red Piranha
18	ProGuard Java Optimizer and Obfuscator	38	c3p0:JDBC DataSources / Resource Pools	58	Cobertura
19	TripleA	39	JavaGroups	59	Jake2
20	JSch	40	OmegaT - multiplatform CAT tool	-	-

4.2. Statistical Analysis of Software Metrics

The relationship among these metrics are evaluated and analyzed statistically. There are several steps in performing this analysis:

- Mapping the values of two metrics in scatter graph to show their relationship to each other. The tool being used to show the scatter graph is using JPGraph (http://jpgraph.net).
- Measuring the Pearson r product-moment correlation of each of the correlation in the scatter graph. The value of Pearson r may indicate the level of correlation between the two metrics.
- Measuring the values of least-square-fit linear approximation which are the gradients and constants. These values are important only for relationship which is categorized as high (shown in the value of Pearson r).

Pearson r product-moment correlation shows the possible relationship of two variables. The possible values of Pearson r (or shortened as r only) varies between -1 (means perfectly inversely proportional), 0 (no correlation), to 1 (perfectly proportional). In this research, the possible values of Pearson r ranges from 0 to 1 only, and they are classified into three categories as shown in Table 2.

Table 2. Classification of *Pearson r* Value

Pearson r Classification	Ranges	Interpretation
Low	$0 \leq r < 0.5$	There are very small or no correlation between the two variables.
Mid	$0.5 \leq r < 0.8$	The are possibility of indirect correlation between the two variables. This correlation may be caused by the third variable which are unknown.
High	$0.8 \leq r < 1$	There are correlation which can be assumed as direct correlation between the two variables. The values of gradient and constant from least-square-fit approximation can be used as the linear formulation of these variables.

The following are the result of the statistical analysis of each of the metrics.

4.2.1 Size Metrics (NCLOC, Lines, Statements)

The following is the scatter graph showing the relationship between NCLOC and Lines. Each of the points in the scatter graph representing packages of OSS Projects. Each dot in the scatter graph represents a single package of the OSS Projects (total 1885 packages).

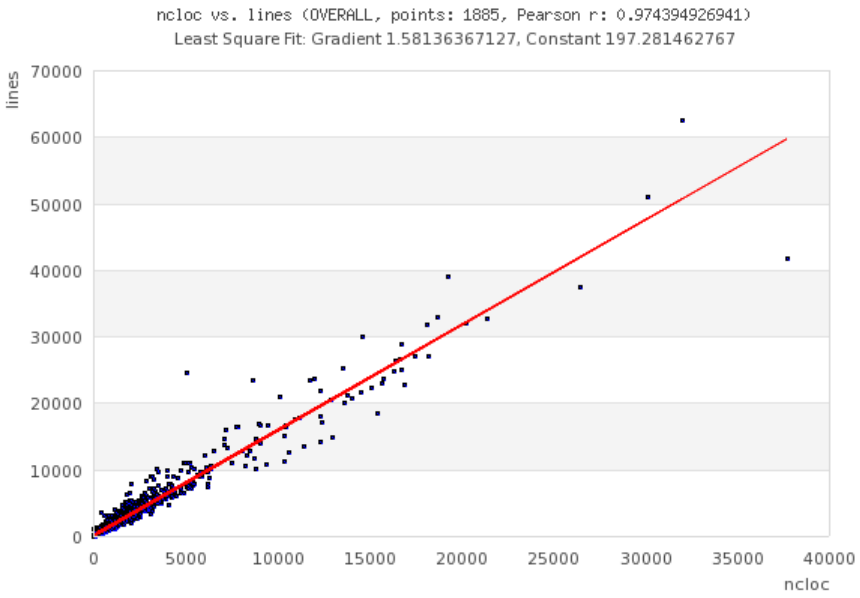

Figure 1. Scatter Graph of NCLOC vs. Lines

The scatter graph shows that between NCLOC and Lines have high correlation (r = 0.974), meaning that there are direct correlation between the two Size Metrics. The same trends also shown in the correlation with the third size metrics which is statements. Table 3 shows the correlation of the tree size metrics. The first three column (Metrics, Pearson r, and Least Square Fit) are aggregate data from 59 OSS Projects (grouped in package or package-wise), while the last column (r (Count Per Project)) are for each individual OSS Projects (grouped in class or class-wise).

Table 3. Correlation of Size Metrics with Other Metrics

Metrics	Pearson r	Least Square Fit		r (Count Per Project)		
		Gradient	Constant	low	mid	high
NCLOC vs. Lines	**0.97439**	**1.58136**	**197.28146**	0	0	**59**
NCLOC vs. Statements	**0.97282**	**0.54400**	**-50.34333**	0	1	**58**
NCLOC vs. Complexity	**0.94855**	**0.25596**	**-30.50913**	0	2	**57**
NCLOC vs. LCOM4	0.75473	0.01023	5.08766	53	5	1
NCLOC vs. RFC	**0.90815**	**0.18544**	**49.18084**	1	12	**46**
NCLOC vs. Afferent_Coupling	0.50813	0.03093	13.47167	56	2	1
NCLOC vs. Efferent_Coupling	**0.81999**	**0.03502**	**8.07098**	9	40	10

Above table shows that all three Size Metrics (NCLOC, Lines, and Statements) have high correlation (r more than 0.9) which indicates direct correlation among them. This observation also confirmed by counting the Pearson r classification for each OSS Projects which shows that almost all projects having the same trend. Choosing only one of the Size Metrics will also implies to the other two. Table 3 also shows that the Size Metrics has high correlation with Complexity Metrics.

The Size Metrics also have high correlation with RFC and Efferent Coupling. In the correlation with RFC, it also shows that the trends are similar in each OSS Projects. In the correlation with Efferent Coupling, the table shows that the trends is different in each project since the correlation is mostly (40 out of 59 projects) are in 'mid' category.

4.2.2 Complexity Metrics (McCabe's Cyclomatic Complexity)

Figure 2 shows the scatter graph NCLOC versus McCabe's Cyclomatic Complexity. This figure shows that Size Metrics has high correlation with the Complexity Metrics with r = 0.949.

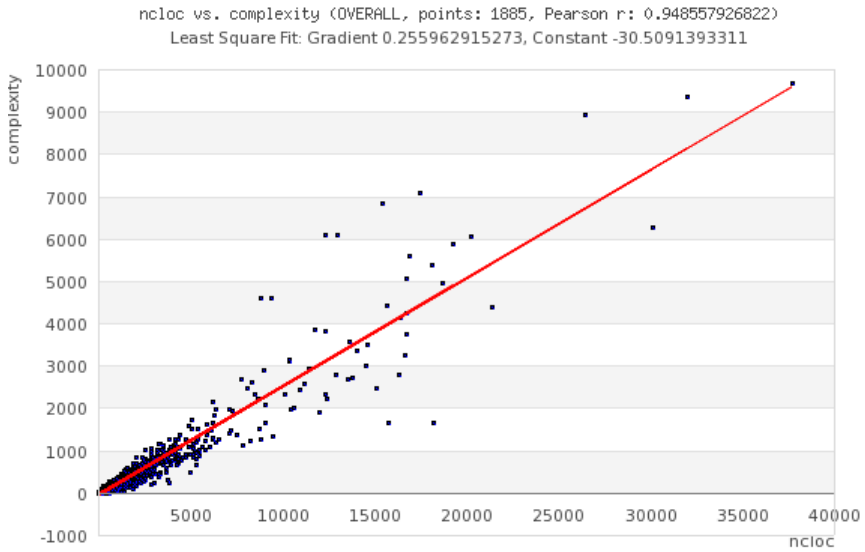

Figure 2. NCLOC vs. Cyclomatic Complexity

Table 4 shows the correlation of Completely Metrics with other metrics (both aggregate or each individual project).

Table 4. Complexity Metrics versus Other Metrics

Metrics	Pearson r	Least Square Fit		r (Count Per Project)		
		Gradient	Constant	low	mid	high
Complexity vs. NCLOC	**0.94855**	**3.51520**	**239.43100**	0	2	**57**
Complexity vs. Lines	**0.91629**	**5.51086**	**590.63047**	0	2	**57**
Complexity vs. Statements	**0.96435**	**1.99843**	**53.45208**	0	2	**57**
Complexity vs. LCOM4	0.71734	0.03604	7.51613	55	3	1
Complexity vs. RFC	**0.85871**	**0.64982**	**94.21622**	1	14	**44**
Complexity vs. Afferent_Coupling	0.51157	0.11540	18.82907	55	3	1
Complexity vs. Efferent_Coupling	0.75947	0.12022	17.34922	15	35	9

Table 4 shows that Complexity Metrics has high correlation with Size Metrics and RFC metrics and has medium to low correlation with other metrics. There are direct relationship between Completely Metrics with Size Metrics. Similarly, there is also direct relationship between Complexity Metrics and RFC.

4.2.3 Cohesion Metrics (LCOM4)

Table 5 shows the correlation of Cohesion Metrics (LCOM4) with other metrics in aggregate data or in individual projects.

Table 5. Cohesion Metrics versus Other Metrics

Metrics	Pearson r	Least Square Fit		r (Count Per Project)		
		Gradient	Constant	low	mid	high
LCOM4 vs. NCLOC	0.75473	55.65486	284.39032	53	5	1
LCOM4 vs. Lines	0.76716	91.81064	576.38322	54	4	1
LCOM4 vs. Statements	0.68180	28.11471	144.53654	56	2	1
LCOM4 vs. Complexity	0.71734	14.27406	41.75480	55	3	1
LCOM4 vs. RFC	0.74158	11.16689	86.20232	50	8	1
LCOM4 vs. Afferent_Coupling	0.50617	2.272227	12.03519	57	2	0
LCOM4 vs. Efferent_Coupling	0.75182	2.36831	10.24952	51	7	1

Above table shows that Cohesion Metrics (LCOM4) doesn't have high correlation with other metrics. The similar trends is also shown in counting the categories of the Pearson r for each projects in which most of the the values are in ‘low’ category. Based on this result, it can be concluded that LCOM4 is an independent metrics.

4.2.4 Coupling Metrics (RFC, Efferent Coupling, Afferent Coupling)

Table 6, 7, and 8 shows the correlation of three Coupling Metrics (RFC, Efferent Coupling, and Afferent Coupling) with other Metrics.

Table 6. RFC versus Other Metrics

Metrics	Pearson r	Least Square Fit		r (Count Per Project)		
		Gradient	Constant	low	mid	high
RFC vs. NCLOC	**0.90815**	**4.44731**	**12.38407**	1	12	**46**
RFC vs. Lines	**0.91075**	**7.23828**	**156.51212**	0	10	**49**
RFC vs. Statements	**0.87555**	**2.39763**	**-37.23211**	1	15	**43**
RFC vs. Complexity	**0.85871**	**1.13474**	**-26.28144**	1	14	**44**
RFC vs. LCOM4	0.74158	0.04924	4.11878	50	8	1
RFC vs. Afferent_Coupling	0.57580	0.17165	3.84236	53	5	1
RFC vs. Efferent_Coupling	**0.87406**	**0.18284**	**0.55447**	2	36	21

Table 6 shows that RFC has high correlation with Size Metrics (NCLOC, Lines, and Statements) and Efferent Coupling. As for the correlation with Efferent Coupling, the different trends are shown in each individual projects in which mostly (36 out of 59 projects) are in 'mid' category.

Table 7. Efferent Coupling versus Other Metrics

Metrics	Pearson r	Least Square Fit		r (Count Per Project)		
		Gradient	Constant	low	mid	high
Efferent_Coupling vs. NCLOC	**0.81999**	**19.19538**	**277.10299**	9	40	10
Efferent_Coupling vs. Lines	**0.82014**	**31.15840**	**591.87991**	12	41	6
Efferent_Coupling vs. Statements	0.74694	9.77773	136.46194	16	36	7
Efferent_Coupling vs. Complexity	0.75947	4.79746	46.70467	15	35	9
Efferent_Coupling vs. LCOM4	0.75182	0.23866	5.63360	51	7	1
Efferent_Coupling vs. RFC	**0.87406**	**4.17824**	**67.00795**	2	36	21
Efferent_Coupling vs. Afferent_Coupling	0.52533	0.74863	13.63980	58	0	1

Table 7 shows that Efferent Coupling has similar correlation with RFC in table 6 with one exception which are the Statements which is slightly lower than the boundary of “high” classification (r >= 0.8). In counting the r category for each projects, the trends are different in which most of the projects have 'mid' category. The different results in the correlation of Efferent Coupling with other Metrics in aggregate data and each individual projects shows that there are inconclusive result for this metrics.

Table 8. Afferent Coupling versus Other Metrics

Metrics	Pearson r	Least Square Fit		r (Count Per Project)		
		Gradient	Constant	low	mid	high
Afferent_Coupling vs. NCLOC	0.50813	8.34703	865.77399	56	2	1
Afferent_Coupling vs. Lines	0.56893	15.16749	1459.60496	55	3	1
Afferent_Coupling vs. Statements	0.46271	4.25045	436.39265	58	0	1
Afferent_Coupling vs. Complexity	0.51157	2.26765	183.98163	55	3	1
Afferent_Coupling vs. RFC	0.57580	1.93149	188.92495	53	5	1
Afferent_Coupling vs. LCOM4	0.50617	0.11275	12.46603	57	2	0
Afferent_Coupling vs. Efferent_Coupling	0.52533	0.36864	34.25975	58	0	1

Table 8 shows that Afferent Coupling doesn't have high correlation with other metrics. The count of the Pearson r values for each projects also shows that all of the correlation are in 'low' category. It can be concluded that Afferent Coupling is an independent metrics.

5. DISCUSSION

The statistical analysis performed on 1885 modules / packages of 59 OSS Projects has provide some interesting results. One of the result is the close correlation among three Size Metrics (NCLOC, Lines, and Statements) despite the different methodology in measuring them. It means that selecting one of the Size Metrics already represents the other metrics. These relationships can be stated in linear relationship (from table 3) as follows:

$$Lines = 1.58136 * NCLOC + 197.28146 \quad \text{….....(1)}$$

$$Statements = 0.54400 * NCLOC – 50.34333 \quad \text{….....(2)}$$

The second result is the close correlation of McCabe's Cyclomatic Complexity and Size Metrics. This may be caused by the level of maturity in the source code of the OSS Projects being analyzed are already high level so that the complexity of the code is reflected from the size of the code. The relationship between the McCabe's Cyclomatic Complexity and Size Metrics (from table 4) is:

$$NCLOC = 3.51520 * Complexity + 239.43100 \quad \text{….....(3)}$$

The third result is the close correlation of Coupling Metrics' RFC with Size Metrics (NCLOC, Lines, Statements) and McCabe's Cyclomatic Complexity. The relationship of RFC with the Size Metrics may be expressed (from table 6) as follows:

$$NCLOC = 4.44731 * RFC + 12.38407 \quad \text{….....(4)}$$

$$Complexity = 1.13474 * RFC - 26.28144 \quad \text{….....(5)}$$

The correlation of Efferent Coupling with Size Metrics and Complexity metrics in aggregate data (grouped in package for all 59 projects) and for each projects (grouped in class for each project) show different result. The correlation of Efferent Coupling to the other Metrics (Size and Complexity metrics) are remain inconclusive and this 'anomaly' should be investigated further in the next research.

The fourth result is the fact that Cohesion Metrics which is LCOM4 and the Coupling Metrics' Afferent Coupling are independent metrics. These metrics should be considered separately in terms of modularity since they are not correlated with any other metrics.

6. CONCLUSIONS

This paper presents the statistical analysis of small to medium sized, Java based OSS Projects. The selection of the projects are based in the number of downloads in which each projects already being downloaded more than 100K times. The high number of download indicates the success of these projects, and their modularity properties (size, complexity, cohesion, and coupling) are analyzed using Pearson r product-moment correlation, scatter graph, and least-square fit linear approximation. The limitation of project's size in terms of NCLOC are less than 170K. Out of 209 OSS Projects as the candidate for evaluation, there are only 59 OSS Projects that are able to be analyzed.

There are some interesting findings from the analysis. The first finding is the close correlation of the three Size Metrics (NCLOC, Lines, Statements) which indicates the direct relationship among these metrics. The second finding is the close correlation of Size Metrics and McCabe's Cyclomatic Complexity. The third finding is also the close correlation of one of the Coupling Metrics (RFC) with Size Metrics and McCabe's Cyclomatic Complexity and the inconclusive

result for Efferent Coupling. The last findings is the independence of Cohesion Metrics (LCOM4) and one Coupling Metrics (Afferent Coupling). These results indicates that there are only three dimensions that should be considered when analyzing modularity in these OSS Projects which are Size, Cohesion, and Afferent Coupling.

There are some cautions should be considered when applying these results in broader scope. These four findings are may be only applicable to small to medium sized (NCLOC < 170K) and Java-based OSS Projects collected from Sourceforge.net. It can be observed from the scatter graphs that the deviation from the measured least-square-fit approximation are getting greater if the package is getting bigger in size. The applications of the results for other object oriented programming such as C++ may requires adjustment and re-confirmation.

Future study related to this research is further evaluation of these OSS Projects in order unify the modularity-related metrics into a single metrics called Modularity Index. These index may serve as a measure quantify the modularity of Java-based OSS Projects. Based in this Modularity Index, a Software Framework called Modularity Framework may be constructed to address the last properties of high modular system (see Section 3.2) which is Open for extension and close to modification. This Software Framework that may be used as guidelines of OSS Project's Administrator or other developers in developing their projects so that the chance of success of their projects is higher.

ACKNOWLEDGEMENTS

The authors would like to thank Maranatha Christian University (http://www.maranatha.edu) that provides funding for the research, and the Department of Computer Science and Electronics Gadjah Mada University (http://mkom.ugm.ac.id) that provides technical support for the research.

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] Bird C., Gourley A., Devanbu P., Swaminathan A., Hsu G. (2007), "Open Borders? Immigration in Open Source Projects", Fourth IEEE International Workshop on Mining Software Repositories 2007, pp 6.

[3] Bouktif S., Antoniol G., Merlo E. (2006), "A Feedback Based Quality Assessment to Support Open Source Software Evolution: the GRASS Case Study", 22nd IEEE International Conference on Software Maintenance 2006, pp 155 - 165

[4] Cai Y., Huynh S. (2007), "An Evolution Model for Software Modularity Assessment", Proceeding of the Fifth International Workshop on Software Qualty 2007 (WoSQ'07).

[5] Capiluppi A., Ramil J.F. (2004), "Studying the Evolution of Open Source Systems at Different Levels of Granularity: Two Case Studies", Proceeding on the 7th IEEE International Workshop of Principles of Software Evolution, 2004, pp 113 - 118.

[6] 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, Nov/Dec 2008, pp 765 - 782.

[7] Chidamber S.R., Kemerer C.F. (1994), "Metrics suite for Object Oriented Design", IEEE Transaction on Software Engineering, Vol. 20 No. 6 June 1994, pp 476 – 493.

[8] Christley S., Madey G. (2007), "Analysis of Activity in the Open Source Software Development Community", Proceeding of the 40th IEEE Annual Hawaii International Conference on System Sciences, 2007, pp 166b.

[9] Crowston K., Wei K., Li Q., Howison J. (2006), "Core and Periphery in Free / Libre and Open Source Software Team Communications", Proceeding of the 39th IEEE Hawaii International Conference on System Sciences 2006

[10] DeKoenigsberg G. (2008), "How Successful Open Source Projects Work, and How and Why to Introduce Students to the Open Source World", 21st IEEE Conference on Software Engineering Education and Training, 2008, pp 274 – 276.

[11] Dinh-Trong T., Bieman J.M. (2004), "Open Source Software Development: A Case Study of FreeBSD", Proceedings of the 10th IEEE International Symposium on Software Metrics, 2004, pp 96 - 105.

[12] Ellis H.J.C., Morelli R.A. , Lanerolle T.R., Damon J., Raye J. (2007), "Can Humanitarian Open-Source Software Development Draw New Students to CS?", Proceeding of the 38th SIGCSE Technical Symposium on Computer Science Education 2007, pp 551 – 555.

[13] Emanuel A.W.R, Wardoyo R., Istiyanto J.E., Mustofa K. (2010), "Success Rules of OSS Projects Using Datamining 3-Itemset Association Rule", International Journal of Computer Science Issue (IJCSI), Vol. 7 Issue 6 Nov. 2010, pp 71 – 80.

[14] Gurbani V.K., Garvert A., Herbsleb J.D. (2005), "A Case Study of Open Source Tools and Practices in Commercial Setting", Proceeding of the fifth Workshop on Open Source Software Engineering 2005, pp 1 - 6.

[15] Hitz M., Montazeri B. (1995), "Measuring Coupling and Cohesion In Object-Oriented Systems", Proceeding International Symposium on Applied Corporate Computing, Oct. 25-27 1995, Monterrey, Mexico, 75-76, 197, 78-84

[16] Kishida K., Ye Y.(2003), "Toward an Understanding of the Motivation of Open Source Software Developers". IEEE Digital Library 0-7695-1877-X/03: 419 – 429

[17] Lawrie T., Gacek C. (2002), "Issues of Dependability in Open Source Software Development", Software Engineering Notes vol 27 no 3 of ACM Sigsoft. May 2002. Pp 34 -37

[18] Li P.L., Herbsleb J., Shaw M. (2005), " Finding Predictors of Field Defects for Open Source Software Systems in Commonly Available Data Sources: a Case Study of OpenBSD", Proceeding of 11th IEEE International Software Metrics Symposium, 2005, 32.

[19] Matos Jr P., Duarte R., Cardim I., Borba P. (2007), "Using Design Structure Matrices to Assess Modularity in Aspect-Oriented Software Production Lines", Proceeding on the First International Workshop on Assessment of Contemporary Modularization Techniques 2007 (ACoM'07).

[20] McCabe T. (1976), "A Complexity Measure", IEEE Transactions On Software Engineering, Vol. Se-2, No. 4, December 1976, pp. 308-320.

[21] Melton H., Tempero E. (2007), "Toward Assessing Modularity", Proceeding of the First International Workshop on Assessment of Contemporary Modularization Techniques 2007 (ACoM'07)

[22] Mockus A., Fielding R.T., Herbsleb J.(2002), "Two Case Studies of Open Source Software Development: Apache and Mozilla", ACM Transaction on Software Engineering and Methodology Vol. II No. 3, Juli 2002, 309 – 346

[23] Mockus A, Fielding R.T, Herbsleb J.(2000), "A Case Study of Open Source Software Development: The Apache Server", ACM ICSE, 2000, 263 – 272

[24] Munelly J., Fritsch S., Clarke S. (2007) An Aspect-Oriented Approach to the Modularisation of Context. Proceedings of the Fifth Annual IEEE International Conference on Pervasive Computing and Communication (PerCom'07)

[25] 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, 2008, pp 1234 – 1239.

[26] Paech B, Reuschenbach B (2006), "Open Source Requirements Engineering", Proceeding of 14th IEEE International Requirement Engineering Conference: 257 - 262

[27] Raymond E.S. (2000), "The Cathedral and the Bazaar", version 3, Thyrsus Enterprises (http://www.tuxedo.org/~esr/), 2000.

[28] Spaeth S., Stuermer M. (2007), "Sampling in Open Source Development: The Case for Using the Debian GNU/Linux Distribution", Proceedings of the 40th IEEE Hawaii International Conference on System Sciences, 2007, pp 166a.

[29] von Krogh G., Spaeth S., Haefliger S. (2005), "Knowledge Reuse in Open Source Software: An Exploratory Study of 15 Open Source Projects", Proceeding of 38th Hawaii International Conference on System Sciences, 2005, pp. 198b

[30] Wang Y., Shao J. (2003), "Measurement of the Cognitive Functional Complexity of Software", Proceedings of the Second IEEE International Conference on Cognitive Informatics 2003 (ICCI'03).

[31] Zhou F., Davis J. (2008), "A Model of Bug Dynamics for Open Source Software", The Second IEEE International Conference on Secure System Integration and Reliability Improvement 2008, pp 185 - 186.

Authors

Andi Wahju Rahardjo Emanuel is a Full Time Lecturer at the Bachelor Informatics Program, Faculty of Information Technology, Maranatha Christian University in Bandung, Indonesia. He is currently taking his Doctoral Program at the Department of Computer Science and Electronics, Gadjah Mada University in Yogyakarta, Indonesia

Retantyo Wardoyo is an Associate Professor at the Department of Computer Science and Electronics, Gadjah Mada University in Yogyakarta, Indonesia.

Jazi Eko Istiyanto is a Professor and Head of the Department of Computer Science and Electronics, Gadjah Mada University in Yogyakarta, Indonesia

Khabib Mustofa is an Assistant Professor at the Department of Computer Science and Electronics, Gadjah Mada University in Yogyakarta, Indonesia