Volume 5, Issue 5, May 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Prediction of Quality of Open Source Software using Metrics

Uttamjit Kaur*, Gagandeep Singh
Department of Computer Science, GIMET
Punjab, India

Abstract— Open Source Software used for study in this paper is Joda-Time (Java Date and Time API) and Find Bugs (Find Bugs in JAVA Program) and both software used for study were developed using java. The aim of this paper is to study the relationship between maintainability and metrics of 16 and 12 different versions of java open source software. The main objective of this paper is to calculate different metrics over different releases. The result illustrate that these metrics strongly related to the Quality of Open Source Software. From the result, it is concluded that Open Source Software 'Joda-Time' has less impact of these metrics on its quality as other Open Source Software 'Find Bugs' directly affect the Quality, it then the software versions may require a reengineering in the successive versions.

Keywords— Object-Oriented Metrics, OSS, Metrics, Find Bugs, Joda-Time.

I. INTRODUCTION

Object-Oriented Technology has become widely popular concepts in today's Software development environment. This technology provides software with higher quality and low maintenance cost. As now, Software development industry has focus on Software Quality. With today's Software development object-oriented design is a popular concept. It has proved its value for system that maintained and modified. It uses the system metrics to evaluate the Quality of the Object-oriented system; it is time to start investigating Object-Oriented Metrics with respect to system Quality.

Open Source Software (OSS) has been getting more interest in the last few years. Open Source Software (OSS) development scenario, software is available on internet and it allows developers to contribute to the new functionalities, improvement of existing software versions and submitting bud fixes to the current release [2].In such a software development scenario the maintenance of the open source software is a never ending task. Software metrics help to control the quality of Open Source Software (OSS). This paper presents a study of several versions of two Open Source Software (OSS) projects: Find Bugs (Find Bugs in Java Programs) and JODA-TIME (Java date and time API), both software used for study were developed using java. This research seeks to describe concepts and techniques to improve the quality of the object-oriented System. The specific objectives of this study are to investigate the quality change for 12 versions of the Find Bugs and 16 versions of the JODA-TIME software over the successive releases.

II. RELATED WORK AND BACKGROUND

Jubair J. Al-Ja'afer and Khair Eddin M. Sabri King Abdullah [3] presented how Chidamber and Kemerer (CK) introduce the concept of metrics suits that encapsulate with the Method, coupling, cohesion and inheritance. The CK and LK metrics are used to evaluate the design of object oriented programs. Author wants to define the strength and weaknesses of a java programs.

Dr. Linda H. Rosenberg [1] in this paper, the SATC (Software Assurance Technology Centres) at NASA introduce for choosing metrics by identifying the attribute related to object oriented. Author describe about the standard metrics that are used in object oriented metrics i.e: line of code and cyclomatic complexity. The objective of this paper is to support the project manager not to uses the in- built metrics, but to choose the metrics on the basis of behaviour and functionality of object oriented.

Anita Ganpati Dr. Arvind Kalia Dr. Hardeep Singh [3] This paper take a case study on PHP open source software of 50 releases and identify the relationship between various software metrics i.e. LOC, CC and Halstead Volume and Maintainability Index(MI).From the result it was observed that there is a decreases in the maintainability index of the PHP.Moreover,it is analyzer that software metrics are strongly inversely related to the maintainability of PHP Software. The CC and LOC can be considered as the most important factor for controlling the maintainability of the PHP.

III. CASE STUDY

This section represents two Open Source Software used in the study, namely Find Bugs and Joda-Time.

CASE STUDY1: JODA-TIME [5]

Joda-Time provides a quality replacement for the Java date and time classes.Joda-Time is the *de facto* standard date and time library for Java. The standard date and time classes prior to Java SE 8 are poor. By tackling this problem head-on, Joda-Time has become the de facto standard date and time library for Java. Note that from Java SE 8 onwards, users are asked to migrate to java.time (JSR-310).

The design allows for multiple calendar systems, while still providing a simple API. The "default" calendar is the ISO8601 standard which is used by many other standards. The Gregorian, Julian, Buddhist, Coptic, Ethiopic and Islamic calendar systems are also included. Supporting classes include time zone, duration, format and parsing. Joda time version 2.4 contains enhancements, bug fixes and a time zone update.

TABLE I VALUES OF METRICS FOR DIFFERENT RELEASES OF JODA-TIME

Versions	LOC	WMC	LCOM	DIT	MI	CC	RFC	CBO	NOC
0.9	7063	17.71	0.4	1.72	101.31	1.56	48.87	5.47	1.34
0.95	8619	22.14	0.48	2.41	97.21	1.28	69.96	7.02	0.89
0.98	7216	16.46	0.32	1.96	101.47	1.35	37.75	5.91	0.7
0.99	7422	17.02	0.36	1.55	103.66	1.26	37.81	5.3	0.61
1.1	9254	17.49	0.36	1.72	94.74	1.26	70.31	7.12	0.36
1.2	9413	17.75	0.37	1.72	94.41	1.26	71.56	7.26	0.36
1.2.1	9425	17.8	0.37	1.72	94.33	1.26	71.64	7.29	0.36
1.4	8582	23.24	0.47	1.67	97.25	1.35	63.8	8.44	0.42
1.5.2	8792	23.59	0.48	1.74	97.66	1.37	63.31	7.63	0.43
1.6.1	11485	16.92	0.44	1.61	87.25	1.56	47.42	8.17	0.4
1.6.2	10474	21.13	0.46	1.61	91.6	1.47	58.37	8.43	0.38
2.0	11240	9.96	0.31	0.89	55.84	0.77	27.46	5.49	0.17
2.1	11357	9.74	0.31	0.9	55.46	0.77	27.58	5.56	0.18
2.2	11308	9.59	0.3	0.9	56.27	0.76	26.94	5.4	0.18
2.3	10855	10.25	0.3	0.93	58.46	0.81	27.87	5.49	0.18
2.4	10844	10.25	0.3	0.93	58.38	0.83	27.82	5.5	0.18

A. CASE STUDY2:FINDS BUGS (Find Bugs in Java Programs) [4]

Find Bugs is an Open Source Software created by Bill Pugh and David Hove ever, Which looks for bugs in Java code. It uses static analysis to identify hundreds of different potential types of errors in java programs. Find Bugs operates on Java byte code rather than source code. Find Bugs 2.0.3 is intended to be a minor bug fix release over Find Bugs 2.0.2. Although than some improvements to existing bug detectors and analysis engines, and a few new bug patterns, and some important bug fixes to the Eclipse plug-in, no significant changes should be observed.

TABLE II VALUES OF METRICS FOR DIFFERENT RELEASES OF FIND BUGS

Versions	LOC	WMC	LCOM	DIT	MI	CC	RFC	CBO	NOC
1.2.1	3076	7.29	0.53	1.18	93.39	1.94	27.9	7.05	0.1
1.3.4	3212	8.27	0.58	1.39	92.95	1.8	21.49	5.91	0.16
1.3.5	3110	7.8	0.52	1.38	80.72	1.88	26.44	6.39	0.22
1.3.6	3405	7.56	0.51	1.48	89.37	1.96	25.9	7.26	0.17
1.3.7	3068	7.71	0.61	1.37	100.96	1.78	21.42	5.44	0.12
1.3.8	3183	6.96	0.53	1.33	96.92	2.05	26.09	6.69	0.15
2.0.0	4305	5.55	0.58	1.9	78.81	2.37	26.49	8.98	0.27

2.0.1	4351	6.41	0.53	1.45	85.42	1.92	27.84	9.96	0.14
2.0.2	4718	5.03	0.63	1.77	81.35	2.17	25.35	8.93	0.25
2.0.3	4441	5.74	0.46	1.26	75.14	1.39	26.19	4.65	0.21
3.0.0	4618	6.34	0.54	1.8	80.03	1.88	24.51	8.33	0.3
3.0.1	4673	6.79	0.49	1.6	73.06	2.09	28.05	8.36	0.2

IV. ANALYSIS

This section presents various object-oriented metrics and their relationship with the quality of the Open Source Software (OSS).A comparison of the measured values of the metrics is made with an increase in the value of Cyclomatic Complexcity (CC) and Lines of Code (LOC) represent it is harder to understand and maintain the software. It clearly represents a decrease in the quality of the software.There are many metrics that are traditional functional development. The metrics that are applicable to object oriented development: Complexcity, Size and readiability.To measure the complexcity, the Cyclomatic complexcity is used.

A. METRIC 1: Cyclomatic Complexcity (CC)

Cyclomatic Complexity (CC) metrics [6] used to identify the complexity of a software program which is based on a directed graph. The formula for calculating the cc is the Number of edges minus the number of nodes plus 2. A high value of Cyclomatic Complexity indicates high complexity of a program which is harder to test, understand, modify and maintain [8]. To reduce the complexcity of software development, it is suggested to limit the Cyclomatic Complexcity to 10[10]. From Table1 it is clear that cyclomatic Complexcity of Joda-Time decreases as releases over its lifecycle and Cyclomatic Complexcity of Find Bugs increases shown in Table2. As Figure1and 2 shows the Cyclomatic Complexcity of Find Bugs and joda-Time.

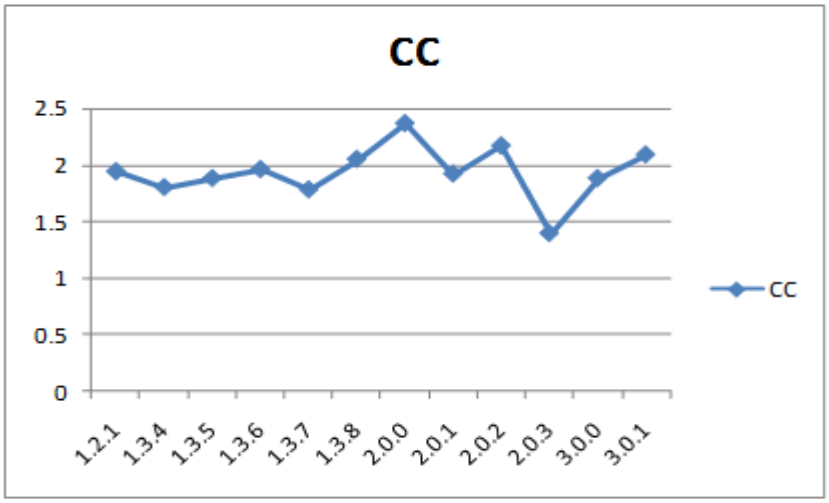

Fig. 1 Cylomatic Complexity for Find Bugs

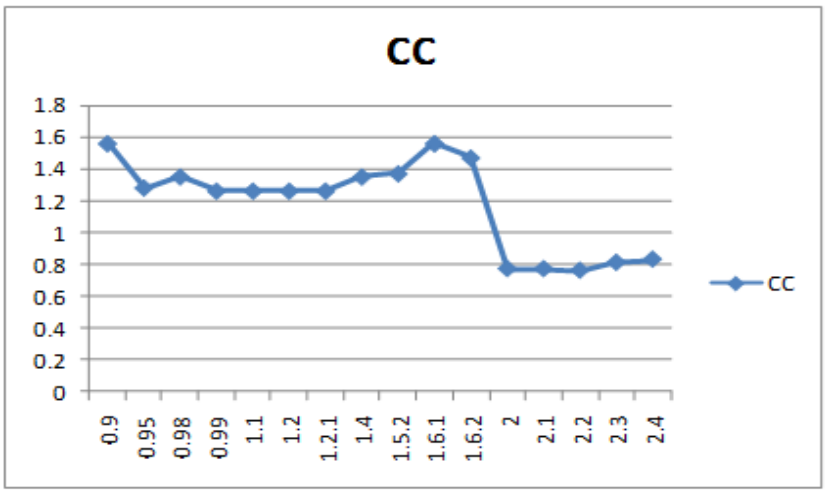

Fig. 2 Cylomatic Complexity for Joda-Time

B. METRIC 2: Lines of Code (LOC)

Lines of Code represent the size of a class that is used to evaluate the ease of understanding of code by developers and maintainers. LOC measures vary depending on the coding language used and the complexity of the method. However, since size affects ease of understanding by the developers and maintainers, classes and methods of large size will always pose a higher risk. High value of LOC indicates less understandability of the software and also affected the quality of the software [8].From Table1 and Table2, it is clear that LOC increases as releases over its lifecycle.Figure3 and Figure4 shows the LOC of Find Bugs and Joda-Time.

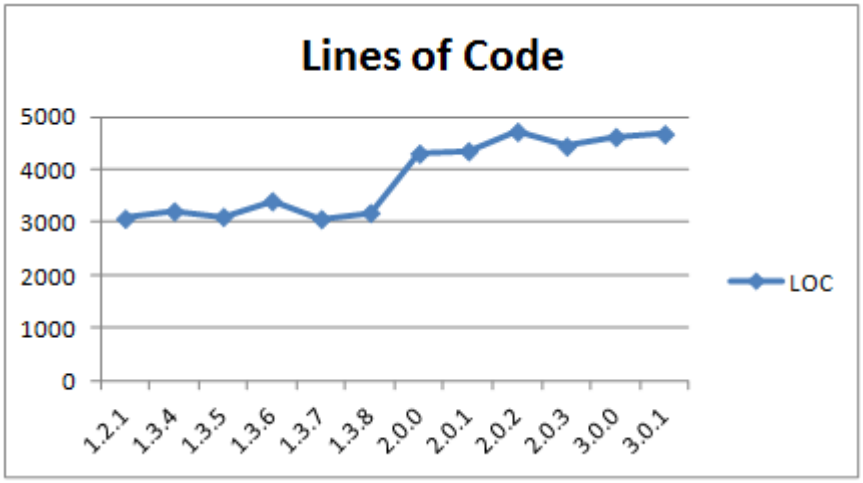

Fig. 3 LOC for Find Bugs

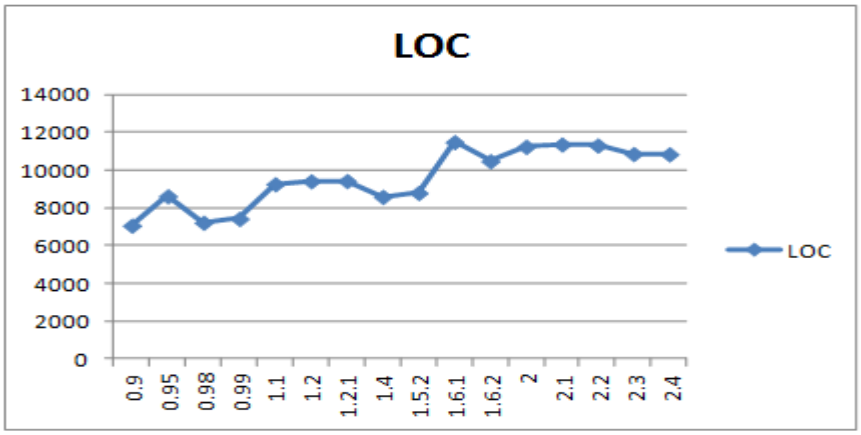

Fig. 4 LOC for Joda-Time

C. METRICS4: Weighted Methods per Class (WMC)

The WMC indicates the total complexity of an object-oriented class, measured by the number of methods defined in the class when all methods complexity is considered unity. Smaller amount of methods will reduce program complexity and increases readability of program. High value of WMC, will increases the complexity and decreases the quality of the software. Larger the value of complexity, less the quality of the software. So, low value of WMC is considered [10].From Table 1 and 2, it is clear that WMC decreases as releases over its lifecycle. Fig 5 and 6 shows the Weighted Method per Class of Find Bugs and Joda Time.

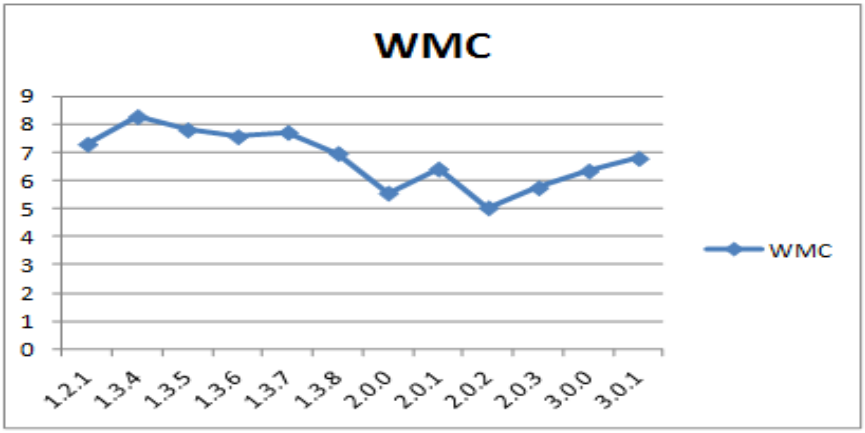

Fig. 5 WMC for Find Bugs

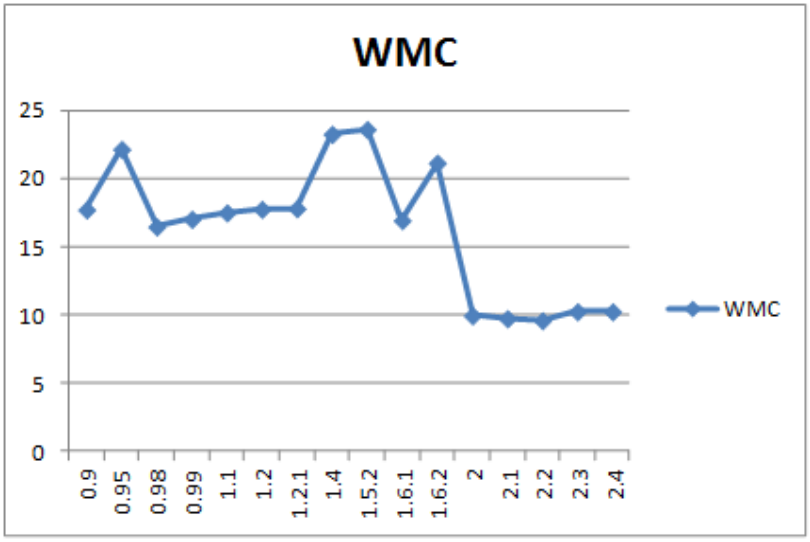

Fig. 6 WMC for Joda-Time

D. METRICS5: Response for a Class (RFC)

RFC is defined as "[8] a set of methods that can potentially be executed in response to a message received by an object of that class". It indicates the number of methods in the response set of the class. High value of RFC will make the classes more complex and hard to understand, test and debug. So low value of RFC is considered. This metrics measure the understabability and complexity of the software.From Table 1 and 2, it is clear that RFC decreases in Joda-Time and increases in Find Bugs. As fig 7 and 8 shows the RFC of Find Bugs and Joda-time.

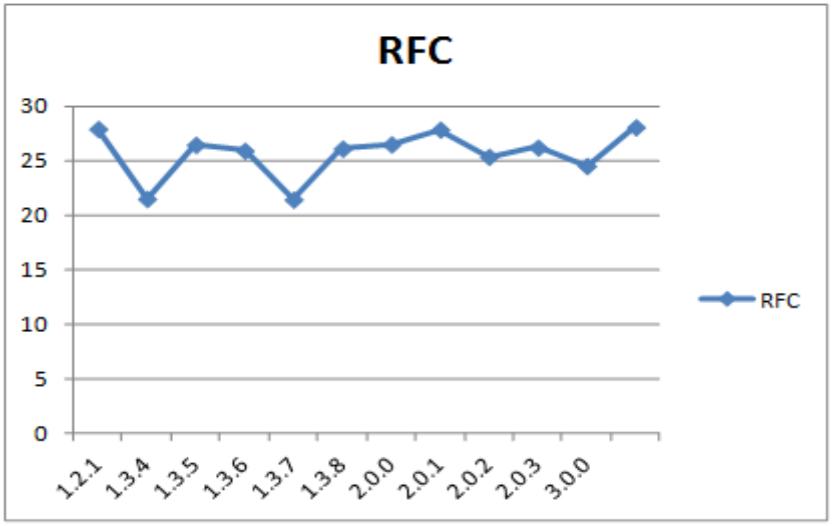

Fig. 7 RFC for Find Bugs

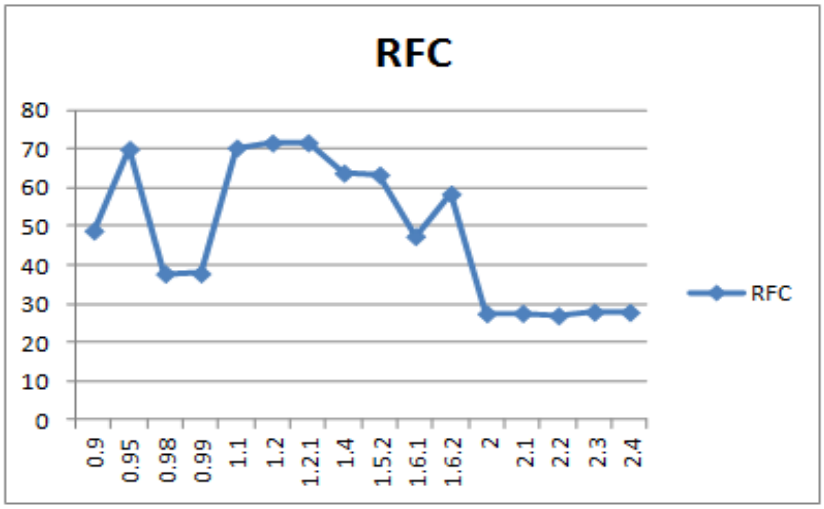

Fig. 8 RFC for Joda-Time

E. METRICS6: Lack of Cohesion (LCOM)

In object oriented programming, cohesion refers to the degree to which the elements of a module belong together. When the cohesion is high, the abstraction of a module is well managed and readability of the program is increased. High value of LCOM can be affected the complexity, reusability, efficiency and quality of the software. Low value of LCOM is preferred. From Table 1 and 2. it is clear that LCOM decreases over the releases of its lifecycle. As from fig 9 and 10 shows the LCOM of Find Bugs and Joda-Time.

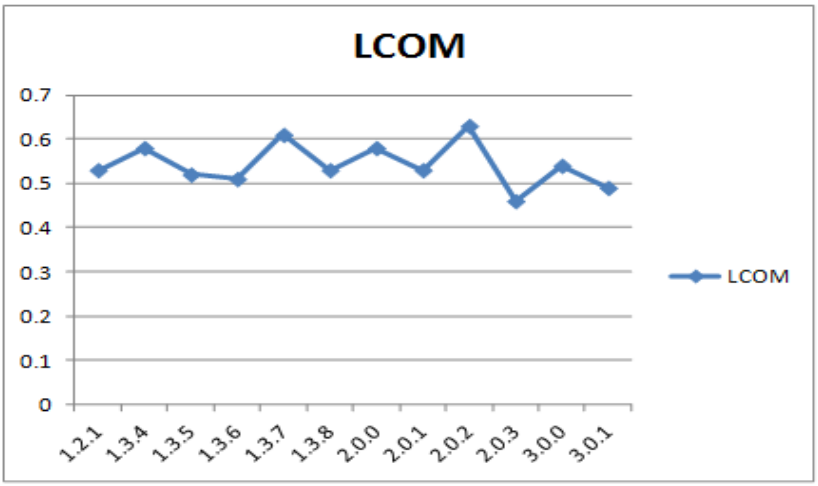

Fig. 9 LCOM for Find Bugs

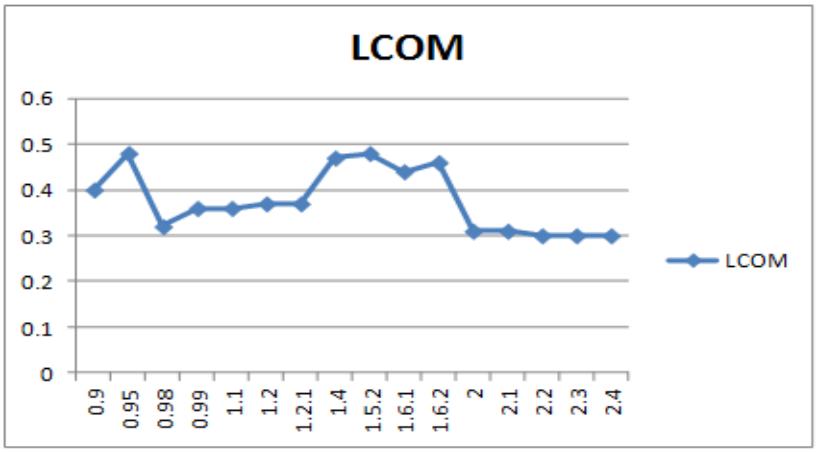

Fig. 10 LCOM for Joda-Time

F. METRICS7: Coupling between Object Classes (CBO)

Coupling is defined as "the measure of the strength of association established by a connection from one module to another". It measures the interdependence of two classes. When coupling between classes is high, it would be more time consuming for programmers to understand the codes and make changes. High value of CBO, will difficult to understand, correct and more complex. To design high Quality software, low coupling is considered [8].From Table 1 and 2, it is clear that CBO increases in Find Bugs and decreases in Joda-Time. Fig 11 and 12 shows CBO of Find Bugs and Joda-Time.

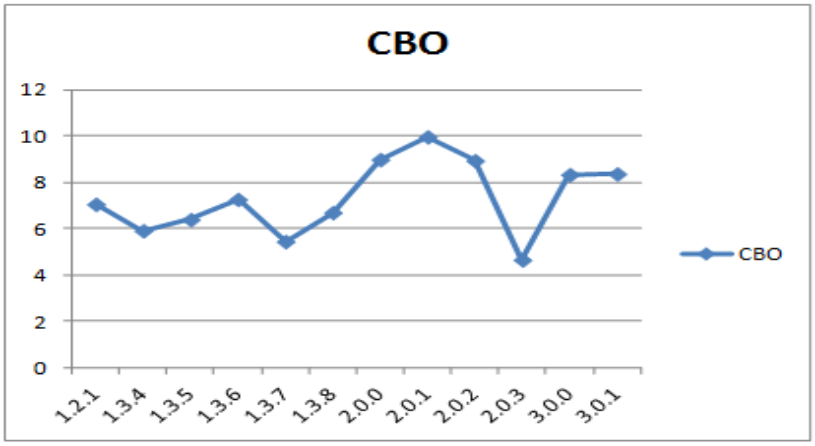

Fig. 11 CBO for Find Bugs

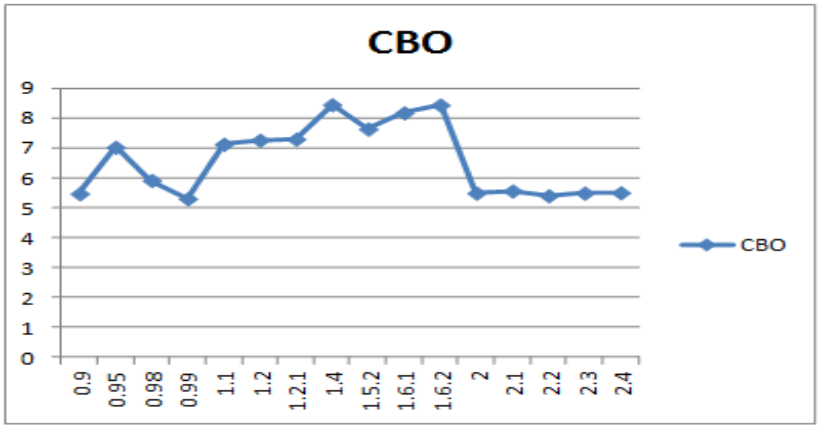

Fig. 12 CBO for Joda-Time

G. METRICS8: Depth of Inheritance Tree (DIT)

DIT is the maximum length from the node to the root of the tree. The deeper a class in the hierarchy is, the greater the number of methods is likely to inherit making it more complex to predict its behavior. The DIT indicates the maximum length from a class to its root class, which is the depth of the hierarchy. High value of DIT will make higher the level of inheritance. Due to larger DIT, Quality of the software will decreases and increases the complexity. So low value of DIT is considered. From Table 1 and 2, it is clear that DIT increases in Find Bugs and Decreases in Joda Time. Fig 13 and 14 shows the DIT of Find Bugs and Joda-Time.

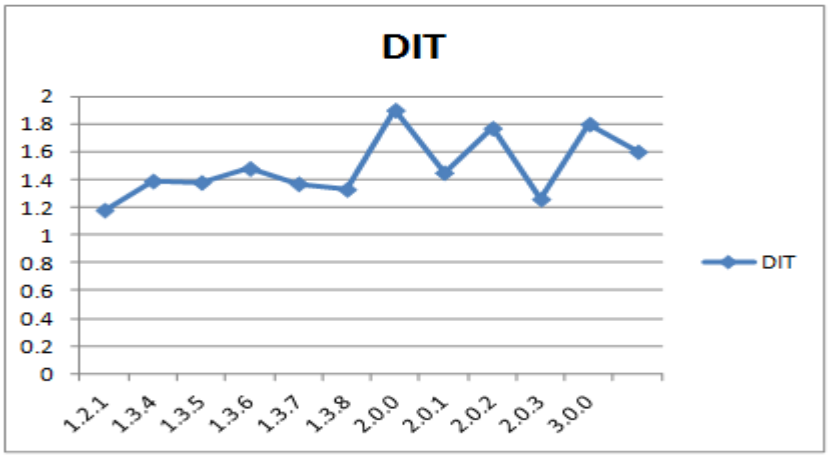

Fig. 13 DIT for Find Bugs

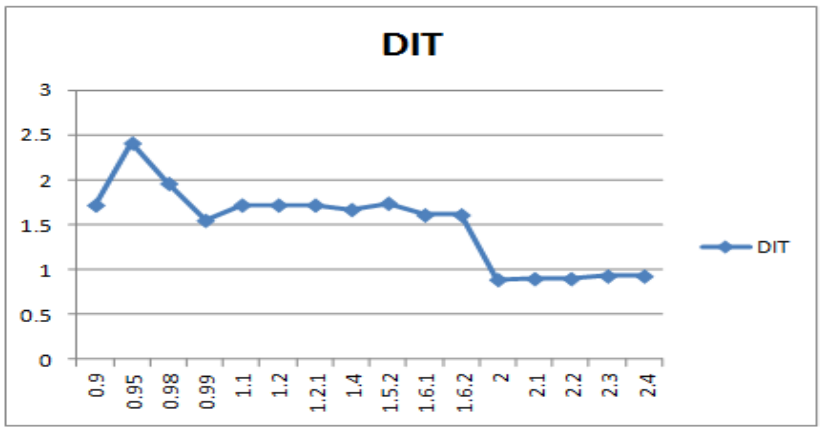

Fig. 14 CBO for Joda-Time

H. METRICS9: Number of Children (NOC)

NOC is the number of immediate sub-classes derived from a base class. The NOC represent the width of the class. The greater the NOC, the greater the likelihood of improper abstraction of the parent and may be case of misuse of sub-classing. High value of NOC indicates high reuse of the base class, But the greater NOC, the greater the reuse since inheritance is a form of reuse. High value of NOC increases the testing time and Quality of the software decreases with the higher value of NOC. So low NOC is considered. From Table 1 and 2, it is clear that NOC increases in Find Bugs and decrease in Joda-Time. As fig 15 and 16 shows the NOC of Find Bugs and Joda-Time.

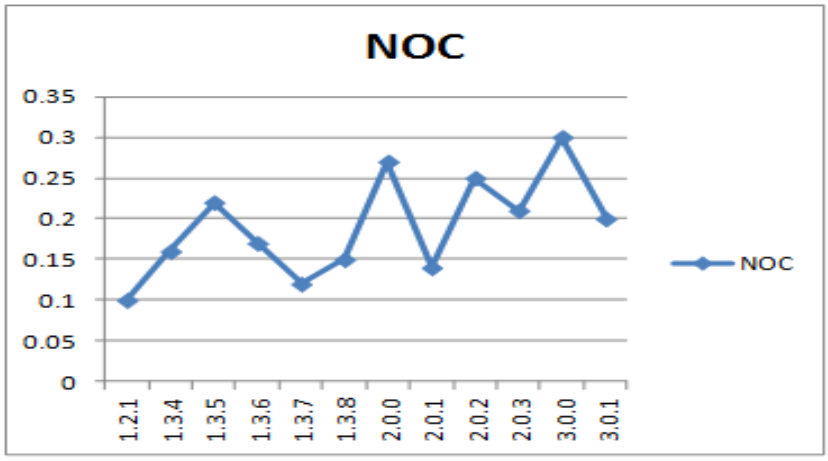

Fig. 15 NOC for Find Bugs

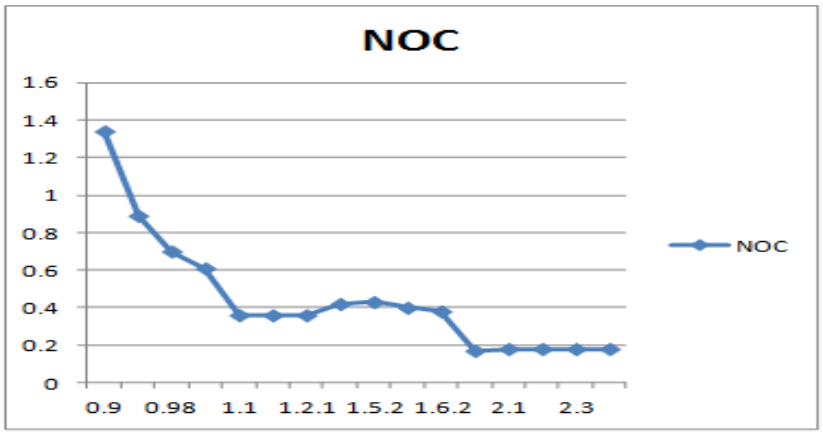

Fig. 16 NOC for Joda-Time

I. METRICS10: Maintainability Index (MI)

MI is very useful in improving software system's maintainability. It makes system more maintainable and cost effective during the software development life cycle. Also, it can be used to study and assess different systems by comparing their MI values. It gives an outstanding approach into the source code of a system for direct manual analysis to emphasize areas of the code which require human intervention. The formula for maintainability index is:

MI = 171 - 5.2 * In(aveV) - 0.23 * aveV(g') - 16.2 *In (aveLOC)

Where

aveV = average Halstead Volume V per module

aveV(g') = average cyclomatic complexity per module

aveLOC = the average count of lines of code (LOC) per module

According to Coleman a MI value above 85 indicates that the software is highly maintainable, a value between 85 and 65 suggests moderate maintainability, and a value below 65 indicates that software is difficult to maintain. The result for the MI for 16 versions of JODA-TIME and 12 versions of Find Bugs is graphically depicted below in fig 17 1nd 18. It is clear from the values of the MI that there is a decrease in the MI of JODA-TIME. The decrease in the MI is an indicator that the maintenance of the JODA-TIME and Find Bugs has increased over the releases.

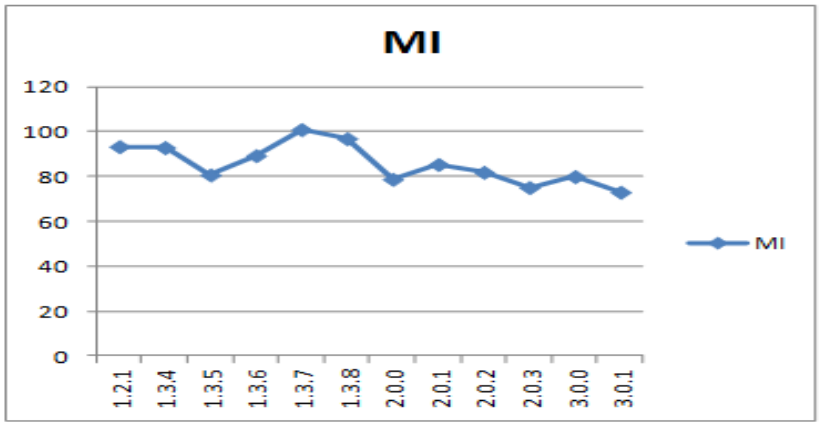

Fig. 17 MI for Find Bugs

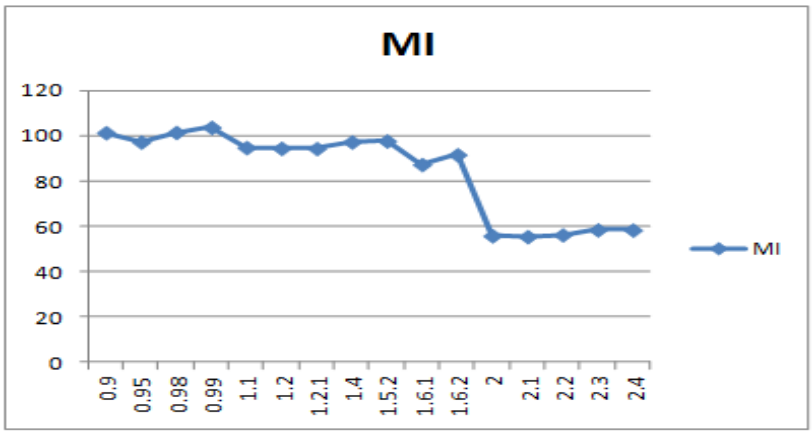

Fig. 18 MI for Joda-Time

V. CONCLUSIONS

In this paper, a case study of two Open Source Software (OSS) namely, Find Bugs and Joda-Time was analyzed over the 12 and 16 versions respectively. From the result it was observed that the Joda-Time increases in LOC and decreases in CC,WMC,RFC,LCOM,CBO,DIT,NOC,MI and Find Bugs increases in CC,LOC,RFC,CBO,DIT,NOC,but deceases in WMC,LCOM,MI. From the result, it is concluded that Joda-Time increases the quality as Find Bugs decreases the Quality it then the software versions may require a reengineering in the successive versions.

REFERENCES

[1] "Applying and Interpreting Object Oriented Metrics"Presenter: Dr.Linda H. Rosenberg Track: Track 7 – Measures/Metrics.

[2] "An Overview of Object-Oriented Design Metrics" DanielRodriguez Rachel HarrisonRUCS/2001/TR/A March 2001.

[3] CHIDAMBER-KEMERER (CK) AND LORENZE-KIDD (LK) METRICS TO ASSESS JAVA PROGRAMS Jubair J. Al-Ja'afer and Khair Eddin M. Sabri King Abdullah II School for Information Technology, University of Jordan, Jordan.

[4] http://sourceforge.net/projects/findbugs/?source=typ_redirect

[5] http://sourceforge.net/projects/joda-time/?source=typ_redirect

[6] https://github.com/JodaOrg/joda-time/releases

[7] http://findbugs.sourceforge.net/

[8] J. Hayes, S. atel, L. Zhao, "A MtricsBased Software Maintenance Effort Model". Proceedings 8th European Conference on Software Maintenance and Reengineering, IEEE Computer Society Press, Pages 254-260, Los Alamitos, Calfornia, 2004.

[9] McCabe, T.J. 1`976. A Complexity Measure, IEEE Transaction on Software Engineering, vol.2, No.4, pp.308-320.

[10] Watson, A.H, McCabe, T.J and Wallace, D.R.1996.Structured testing: A testing methodology using the Cyclomatic Complexity metrics.National Institute of Standards and Technology Special Publication 500-235.

