Using Metrics To Measure The Complexity, Understandability And Readability Of Open Source Software Over Multiple Releases

Uttamjit Kaur

Department of Computer Science, GIME, Amritsar, India
Email: er.uttamjitkaur@gmail.com

ABSTRACT: Open-Source Software (OSS) is becoming very popular in today’s software development environment. In OSS, the development and maintenance of the software is decentralized due to which maintainability is a core issue in OSS development. The maintenance of OSS is a never-ending task. Software metrics help to control the quality of OSS. This paper reviews popular object-oriented metrics to analyze LOC, CC RCC and CLC of the JACOB Open Source Software (OSS) of Twenty-two releases over a successive versions. The terms LOC, CC, RCC and CLC was computed for all the Twenty-Two successive versions of OSS. The software metrics were calculated using Understand4Java tool.

Keywords: Open-Source Software, Software Metrics, Versions and Software Complexity.

1 INTRODUCTION

In a changing environment, all software’s must change with time to meet the need of its users [2]. Therefore it is important for software development and software developers to keep software operational and to enhance the functionality of the software [2] such that the problems arising from changes are reduced. Open Source Software (OSS) has been getting more interest in the last few years. Open Source Software (OSS) is usually developed by volunteers from all over the world working co-operatively [2]. In 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 version and submitting bug fixes to the current release [2]. In the OSS, the development as well as maintenance of the software is decentralized due to which maintainability is core issue in OSS development [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.

2 LITERATURE REVIEW

Jubair J. Al-Ja’afer and Khair Eddin M. Sabri King Abdullah [1] 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. Mrinal Singh Rawat, Arpita Mittal and Sanjay Kumar Dubey [7] in this paper, author describe about the software quality can be viewed differently according to factors i.e.; User view. Manufacturing view, product view, value based view. Author defines the software metrics as a valuable entity and used to measure the standard that helps to evaluate the quality, efficiency, design and product. It describes the merits and demerits of object oriented metrics. Hayes et al. [9] derived a model for estimating adaptive maintenance effort. It was concluded that the number of LOC changed and the number of operators changed are strongly correlated with the maintenance effort. Denis et al. [2] investigated the relationships between software maintainability and other internal software quality attributes. The source code characteristics of five java-based open-source software products were analyzed.

3 OBJECTIVE OF THE STUDY

The objective of this study is to analyze Line of Code (LOC), Cyclomatic Complexity (CC), Count Line Code (CLC) and Ratio Comment to Code (RCC) for Twenty-Two versions of the Open Source Software.

4 RESEARCH METHODOLOGY

The source code of Open Source Software (OSS) i.e JACOB was used in the study as the data source. JACOB is a JAVA-COM Bridge that allows you to call COM Automation components from java. It uses JNI to make native calls to the COM libraries. JACOB runs on x86 and x64 environments supporting 32 bit and 64 bit JVM’s. As of versions 1.8, the following things are true about JACOB:
- The project license changes from the LGPL to BSD.
- JACOB is now compiled with java 1.4.2.
- The project is hosted on SourceForge.

Understand 4 Java tool provides information regarding code. All information on functions, classes, variables, etc. how they are used, called, modified and interacted with. Understand is very efficient at collecting metrics about the code and providing different ways for you to view it. There is a substantial collection of standard metrics quickly available as well as options for writing metrics cover exactly what you need.

<table>
<thead>
<tr>
<th>TABLE 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUES OF METRICS FOR DIFFERENT RELEASES OF JACOB</td>
</tr>
<tr>
<td>Versions</td>
</tr>
<tr>
<td>1.9</td>
</tr>
<tr>
<td>1.9.1</td>
</tr>
<tr>
<td>1.10.0</td>
</tr>
<tr>
<td>1.10.1</td>
</tr>
</tbody>
</table>

Copyright © 2015 IJTEE.
gram which is harder to test, understand, modify and maintain [9]. To reduce the complexity of software development, it is suggested to limit the Cyclomatic Complexity to 10[11]. From Table 1 it is clear that cyclomatic Complexity increases as releases over its lifecycle. Figure 1 shows the Cyclomatic Complexity of JACOB.

**Fig:** Cyclomatic Complexity

![Cyclomatic Complexity](image)

5.2 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 [9]. From Table 1 it is clear that LOC increases as releases over its lifecycle. Figure 2 shows the LOC of JACOB.

**Fig 2:** LOC of JACOB

![LOC of JACOB](image)
5.3 METRIC 3: Count Line Code (CLC)
In many programming language, Count Line Code (CLC) counts blank lines, comment lines and physical lines of source code. CLC is known to run on many Windows as Linux, FreeBSD, OpenBSD, Mac OS X or Higher. From Table 1 it is clear that CLC value increases over the successive versions. Figure 3 shows the Count Line Code of JACOB.

![Count Line Code](image1)

**Fig 3: Count Line Code of JACOB**

5.4 METRIC 4: Ratio Comment to Code (RCC)
Comment Percentage is defined as a ratio of the Number of comment lines to the Number of Non-Blank LOC [9]. In any stage of the life cycle, comments will help developers and maintainers to better understand the programs. Since comments assist developers and maintainers, higher comment percentage increases understandability and maintainability. From Table 1 it is clearly defined that RCC decreases. So readability of code is difficult to understand by software developers and development. Figure 4 Shows the RCC of JACOB.

![Ratio Comment To code](image2)

**Fig 4: Ratio Comment To Code of JACOB**

6 CONCLUSION
The Lines of Code, Cyclomatic Complexity, Ratio Comment to Code and Count Line Code of the JACOB software was observed over Twenty-Two Successive versions. From the result it was observed that JACOB has the higher Cyclomatic Complexity and Lines of Code value. The lower values of Ratio Comment to Code in case of JACOB indicates that the versions of this software are harder to understand. Increases in the values of Cyclomatic Complexity, Lines of Code and decreases in the value of Ratio comment to Code indicates that JACOB software must contain complex class, packages and events that harder to understand and reuse.

REFERENCES


