International Journal of Multimedia and Ubiquitous Engineering
Vol.9, No.7 (2014), pp.403-414
http://dx.doi.org/10.14257/ijmue.2014.9.7.34

A New Measure of Code Complexity during Software Evolution: 'A Case Study'

Vinay Singh[1] and Vandana Bhattacherjee[2]

Usha Martin Academy, Ranchi, India[1]
Birla Institute of Technology, Ranchi, India[2]
mailtovsingh@yahoo.co.in[1]
bhattacherjeev@yahoo.in[2]

Abstract

This paper first computes the Complexity increment by taking four complexity metrics WMC (CK), CMC (Li), CC (BS) and CCC (S&B). The maintainability index of the successive version has been computed at the system level. The tracking of the number of classes added and deleted has also been obtained for the archaeology of successive versions. The understandability and the maintainability of software are then mapped with the trends of complexity increment, change in number of classes added and deleted and the Maintainability index. The complexity increments between successive versions give an indication towards the maturity level of software. These metrics are empirically evaluated with 38 versions of JFree Chart and nine versions of three live project data at the system level.

Keywords*: Complexity Metric, maintainability index, maturity, software evolution, understandability*

1. Introduction

The IEEE glossary [15] defined Complexity as the degree to which a system or a component has a design or implementation that is difficult to understand and verify. There are many attributes that directly contribute to software complexity. Thomas McCabe proposed a measure of software called Cyclomatic complexity [9]. Making use of graph theory, McCabe postulated that software with a large number of possible control path would be more difficult to understand, maintain, and test. However, the Cyclomatic number presents only a partial view of complexity. An alternative approach to Cyclomatic complexity presented by Du and Wang [4] defines a Software power (SP) by expanding the information entropy theory. A high value of SP indicates that the software is more complex.

Several researchers have studied the evolution of software across several versions of the same software system [5-7]. The retrieval of essential details about an existing software system is called software archaeology [11, 14] that can aid to realize the organization history, make the main form of the development of the organization, identify major factors influencing software change and predict possible software development trends. Keeping track of this information is all important for estimating the cost of maintenance. Erlikh reported that cost devoted to system maintenance and evolution now accounts for more than 90% of the total cost [12].

Software Metrics are important for software evolution community, because credible methods to assess, observe, model and analyze software evolution process are required [13].

ISSN: 1975-0080 IJMUE

The maintainability index is applied to quantify the effort needed for keeping change in software. It was first proposed by Oman and Hagemeister [16] used Halsted effort, Cyclomatic complexity and the Line of code metrics for calculating maintainability index which was measured at procedural language. Their primary aim is to determine how easy it will be maintaining a particular body of code. Later in virtual machinery [17] proposed a maintainability index for Java systems by taking an extra parameter the average number of comments in the program as it increases code understandability. They also redefine the LOC by counting the number of Java statements.

In this paper, an effort has been made to examine the code understandability with the change of complexity increment and the maintainability index. The complexity increments are obtained by taking four complexity metrics such as Weighted method per class (WMC) of Chidamber and Kemerer [3], Class method complexity (CMC) of Li [8], Class complexity metric (CC) of Balasubramanian [1] and complete class complexity metric (CCC) of Vinay and Bhattacherjee [10].

The rest of the paper is organised as follows: The study of the existing complexity metrics is presented in Section 2, tools and process are presented in Section 3. Empirical study is presented in section 4 whereas metrics and quality is presented in Section 5. The validation is given in Section 6. In the end, the conclusion is given in Section 7.

2. Existing Complexity Metrics

Table 1. Existing Complexity Metrics

Metric	Definition	Reference
WMC (CK)	Consider a class C_i with methods M_1, M_2, $M_3 \ldots M_n$ that are defined in the class. Let c_1, c_2, $c_3 \ldots c_n$ be the complexity of the methods. Then, $$WMC(\text{Weighted Method per Class}) = \sum_{i=1}^{n} c_i$$ If all method complexities are considered to be unity, then WMC= n, the number of methods.	[3]
CMC (Li)	Class Method Complexity (CMC) is the summation of the internal structural complexity of all local methods, regardless of whether they are visible outside the class or not (e.g.. all the private, protected and public methods in class).	[8]
CC (BS)	Class Complexity (CC) metric, is calculated as the sum of the number of instance variables in a class and the sum of the weighted static complexity of local methods in the class. To measure the static complexity Balasubramanian uses McCabe's Cyclomatic Complexity [McCabe, 1976] where the weighted result is the number of nodes subtracted from the sum of the number of edges in a program flow graph and the number of connected components.	[1]

CCC (S&B)	Complete Class Complexity Metric) is used for measuring the complexity of class in Object-Oriented Design. The CCC metric measures the classes at the method level, attribute level and their relationships. CCC metric is calculated as the sum of the nine properties at code level, such as methods, Cyclomatic complexity, external method called, the message sent to other methods, reference variables, super classes, subclasses, interface implemented and the package imported.	[10]

3. Tools and Processes

3.1 Flow of modules

Figure 1 provides an overview of the tools and processes used. The JHawk metric tool is used to extract metric values of Java source code. An interface has been developed in C# .Net that takes a Java source file as input and generate an XML file. To measure the metric at system level two macros has been designed. Summary macro is applied to examine the XML file and generate the Excel sheet which holds the needed attribute to design the metrics as discussed in table 1. The Avg-Calculator macro is applied to compute the measured value at the system level. To compute the average, an average by quartile method is recursively used to find, first at the package level and finally at the system level as presented in Table 2.

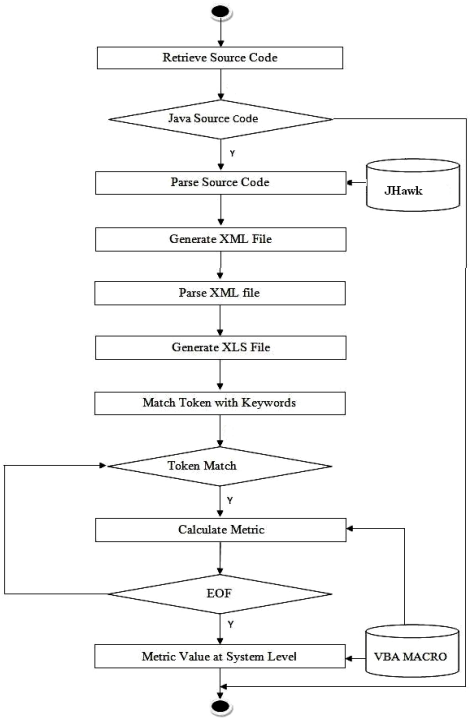

Figure 1. Program Flow Chart of the Complexity Metric

3.2 Algorithm for Average by Quartile

The obtained value of complexity metrics at system level is calculated through the Average by Quartile process. The algorithm defines as follows.

Table 2. Average by Quartile Process

Step 1: for i=1 to Pck_n, where, Pck_n is the number of packages in system X
Step 2: for k=1 to Cl_n , where, Cl_n is the number of classes or interfaces in package i
Step 3: Arrange the attributes of Cl_k in ascending order
Step 4: find the average by quartile of $\overline{Pck_i} = \frac{Q1+Q2+Q3}{3}$

Where, Pck_i is a package with classes Cl_n

$Q1 = \frac{n}{4}$ term, $Q2 = \frac{n}{2}$ term , $Q3 = \frac{3n}{4}$

Step 5: end of inner for loop
Step 6: store each value of Pck_i
Step 7: end of outer for loop
Step 8: Retrieve each value of Pck_i and calculate in the same way for all package value, first, arrange in ascending order and then find the average by quartile similarly from step 3 and step4, finally, the single value is obtained for each metric for system X.

4. Empirical Study

The software used in the experiment was JFreeChart, which is a powerful and flexible open-source charting library. We chose JFreeChart as the target software system because it is a long-term open source library with a rich set of release notes and documents to confirm our observations. An empirical study on the complexity metrics throughout the software evolves and their consequence on the quality attribute understandability and maintainability is discussed. We summarized the collected data by calculating the complexity metrics WMC, CMC, CC, and CCC for the 38 versions of JfreeChart starting from versions 0.9.0 to 1.0.14 with 1658 packages, 28277 classes and 265457 methods at the system level.

The complexity increment of different complexity metrics WMC, CMC, CC and CCC are calculated from

$$Incr(WMC)_i = WMC_i - WMC_{i-1}$$

Where, WMC_i is the complexity value of current version i and WMC_i-1 is the complexity value of the previous version i-1. Likewise the value of CMC, CC and CCC metrics are calculated. Figure 2 shows the complexity increments of WMC, CMC, CC and CCC for the different versions of JFreeChart at the system level. It has been noted that the significant complexity growth occurs in the initial versions as the systems evolve.

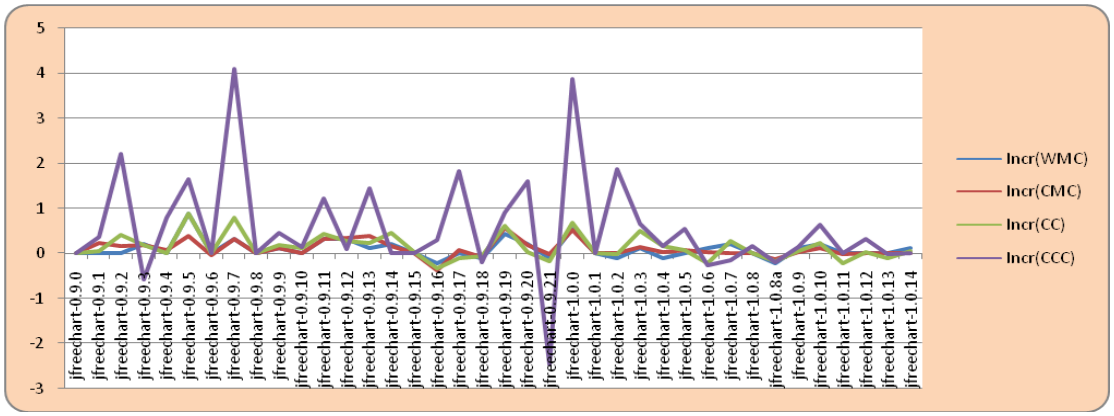

Figure 2. Complexity Increment

Software complexity is inversely proportional to understandability. In order to obtain the measure of the validity of the complexity metrics, it was expected that the complexity of each version of the system should increase from one version to the next which is shown in Figure 2.

The positive growth of complexity comes in the early releases of systems which point out that the changes in the system is difficult tends to sharpen out the fact that the understandability of the system diminishes. The complexity increment values of WMC, CMC, CC and CCC becomes negative in JfreeChart version 0.9.18, version 0.9.21 and version 1.0.8a as shown in system number 19, 22 and 32 shown in Table 6 in Appendix A. The reverse trends in complexity increment increases code understandability.

5. Metric and Quality

To assess the quality attributes understandability and programmer's effort, this paper computes maintainability index and percentage of additions and subtractions of classes for the successive versions of JfreeChart. Programmer's effort is related to the effort required for the additions of functionalities in the successive version. The Maintainability index(MI) value represents relative of ease maintaining the code. A higher value means better maintainability. The MI [17] is being calculated by the formula given in (i)

$$MI = 171 - 3.42\ln(aveE) - 0.23\,aveV(g') - 16.2\ln(ave\ loc) + 50 * \sin(sqrt(2.46 * ave(cm)) \quad \text{(i)}$$

Where, aveE is the average Halsted volume per module.

aveV(g') is the average extended Cyclomatic complexity per module.

ave(loc) is the average number of Java statement per module.

ave(cm) is the average number of comment line per module.

It has been considered that MI less than 65 is considered as poor maintainable, MI greater than 65 is considered reasonable maintainable whereas MI greater than 85 is considered as excellent maintainable [17].

To keep track of the number of changes in the subsequent versions, Percentage of Added and removed classes have been obtained from the following

Let,

$C_i = Number\ of\ classes\ in\ version\ i$

$C_i = C_{i-1} + A_i - R_i$

Where,

$C_{i-1} = Number\ of\ classes\ in\ previous\ version\ (i-1)$

$A_i = Number\ of added\ classes\ in\ version\ i$

$R_i = Number\ of\ classes\ removed\ in\ version\ i$

$A_i\% = {}^{A_i}/_{C_{i-1}} \times 100$

$R_i\% = {}^{R_i}/_{C_{i-1}} \times 100$

The following four cases have been observed to measure the understandability of the software during system evolution.

Case I: Positive growth of complexity increment and low maintainability index causes decrease in understandability and the increase in programmer's effort.

Case II: Positive growth of complexity increment and high maintainability index causes decrease in understandability and also decrease in programmer's effort.

Case III: Negative growth of complexity increment and low maintainability index causes an increase in understandability and also increase in programmer's effort.

Case IV: Negative growth of complexity and high maintainability index causes an increase in understandability and decrease in programmer's effort.

Table 3. Metric and Quality growth (↑ increases ↓decreases)

Case	Complexity	MI	Understandability	Programmer' Effort
I	↑	↓	↓	↑
II	↑	↑	↓	↓
III	↓	↓	↑	↑
IV	↓	↑	↑	↓

It has been noted from Figure 2 and Figure 3 that in early release, there is a positive growth of complexity increments and low value of maintainability is observed that shows the understandability of system decreases and similarly more effort is required to add new functionalities in change the new release(complete results of MI is given in Table 7 in Appendix A). It has been found that 11 versions are under case I, similarly, 11 versions are under case II, 4 versions are under case III whereas, 9 versions are under case IV. It has been noted that 9 versions under case IV are both highly understandable and require less programmer's effort for the addition of new functionality.

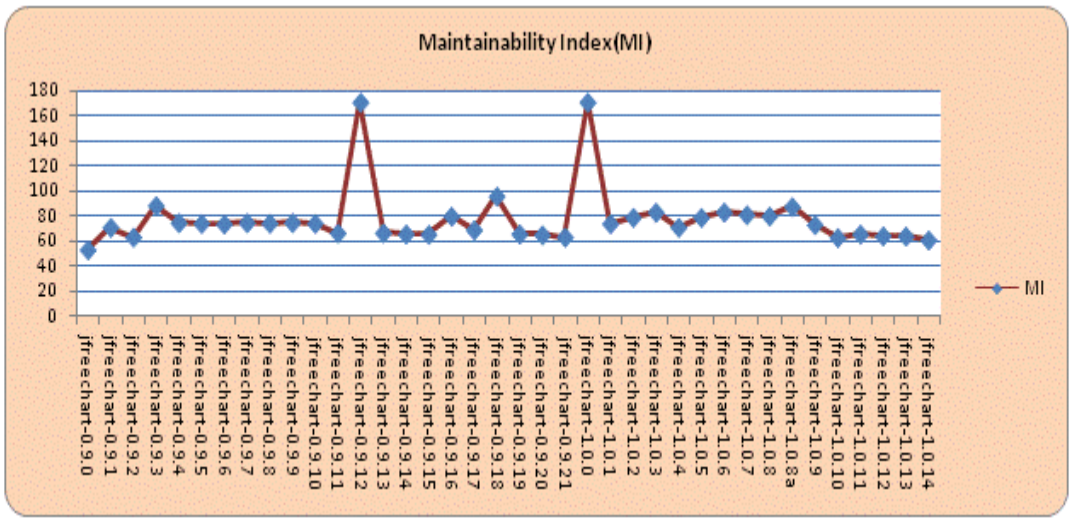

Figure 3. Maintainability Index during Software Evolution

Later on, after several initial releases, a version is expected to mature, having incorporated most required capabilities by removing insignificant classes from the previous version. In this study the major revision of the system is located in version JfreeChart 1.0.0 with a very high maintainability index (MI =171). This means that the system is more maintainable hence require less effort for addition of new functionalities. In this version, 25.4% new classes are added.

It has been calculated by the percentage of added and removed classes (complete results are given in Table 8 in appendix A). This information is used to find the programmer's effort to make the new version.

It has been observed that about 25 % of the classes were added in version 0.9.5, 0.9.7 and 1.0.0 and similarly the system was more complex in the same version 0.9.5 with WMC, CMC, CC and CCC values as (0.39, 0.89, 0.89 and 1.63), version 0.9.7 with (0.33, 0.32, 0.81 and 4.9) and version 1.0.0 with (0.56, 0.52, 0.68 and 3.89). Similarly, the MI value of 0.9.5 is 74.16, whereas, the MI value of version 0.9.7 is 74.73.

A mature version needs more maintenance and programmer's mental effort since a large number of changes are required [2]. This eventually results in a decrease in complexity and an increase in understandability. The mature version 1.0.0 has been obtained after removing 28.8% classes from the previous version 0.9.21 whereas 3.7% classes are added to the low MI value 63.17. Similarly the MI value of the mature version is 171 which indicate that the code is more understandable and less effort was required to add new functionality. The above results on deviation trends in complexity increments, maintainability index and the changes in size (number of added and removed classes) could measure the code understandability.

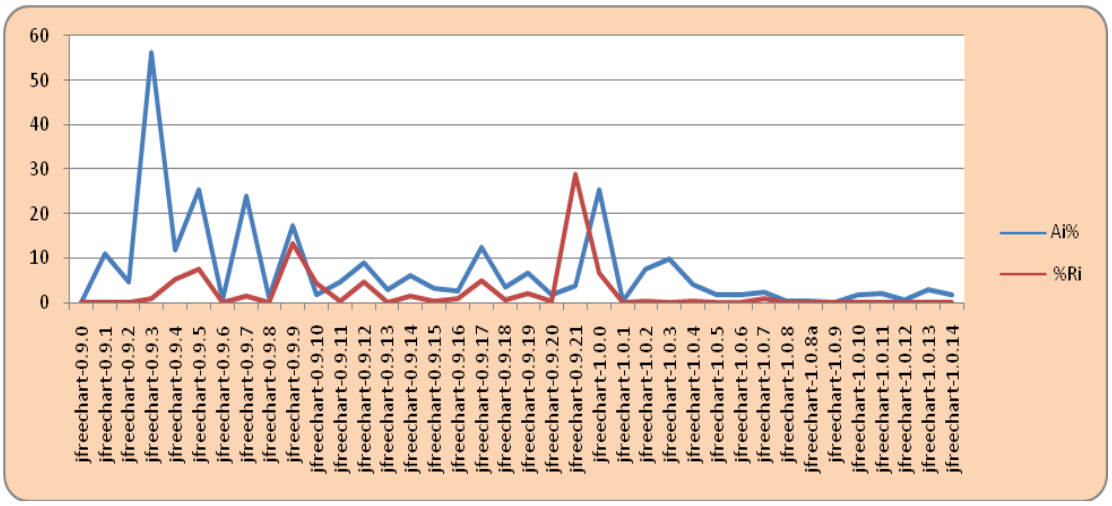

Figure 4. Percentage Number of Classes Removed and Added

6. Validation

In order to obtain a measure of the validity of the quality attribute understandability, it was expected that the understandability should initially decrease from one release to the next because the early structure of the system may also be unstable and undergo significant reworks during the initial releases. To corroborate the above fact, three medium size Java projects, namely, Skywar, LMS and Java Parser have been taken from Master's level students. The Skywar is the gaming software; Library management system uses Oracle as a backend where as Java parser is used to parse the Java code and returns the lower level program attributes. Each project has an initial set of requirements and was provided at the start in version 1.0 and was followed by version 1.1 and version 1.2. The projects were independently completed by the Master's level student over a period of 3 months. The deliverables include the design document and an implementation that came across with all the versions requirements. The projects were parsed by the tool for obtaining the complexity metric value and the maintainability index as shown in Table 4.

Table 4. Complexity Increments and Maintainability Index

Project	WMC	CMC	CC	CCC	MI
SKYWAR 1.0	0.0	0.0	0.0	0.0	64
SKYWAR 2.0	0.35	0.37	0.40	2.61	59
SKYWAR 3.0	0.2	-0.11	0.39	-0.6	85
LMS 1.0	0.0	0.0	0.0	0.0	67
LMS 2.0	0.27	0.46	0.74	1.75	61
LMS 3.0	0.12	0.21	0.85	-0.27	93
Java Parser 1.0	0.0	0.0	0.0	0.0	63
Java Parser 2.0	0.29	0.31	0.61	2.1	51
Java Parser 3.0	0.03	0.5	-0.8	-1.56	91

It has been observed from Table 4 that the complexity increment is initially increasing as the number of changes made in the project is difficult and similarly the maintainability index values are decreasing. The ranking of the projects was determined on a scale of 1-10, from least to most understandings.

Table 5. Programmers Ranking

Project	Version 1.0	Version 2.0	Version 3.0
SKYWAR	4	3	8
LMS	7	5	9
Java Parser	5	3	8

It has been observed from Table 5 that the project SKYWAR was the gaming software in which the changes made to it are difficult and the understandability of the versions are initially decreases similarly, the value of complexity increments increase and the maintainability indexed decreases. The similar trend is also found with other projects.

7. Conclusion and Future Scope

This work demonstrated the outcomes of an experiment where the impact of complexity increment on resulting software quality attributes (Understandability) was empirically evaluated. Weighted method per class, class method complexity, class complexity and complete class complexity metrics were adopted in order to measure the code complexity during software development. The outcomes attained thus far allow us to infer that, increasing trends in complexity shows that the understandability decreases. The complexity increments and the maintainability index can be used for forecasting how much effort would be needed to make the new version of the system, easy to understand. Through the increase in complexity, information about the number of functionalities that are added and deleted, and maintainability index one can choose to measure the understandability and the maintainability of software.

References

[1] N. V. Balasubramanian, "Object Oriented Metrics", Asian Pacific Software Engineering, Conference (APSEC-96), (**1996**) December, pp. 30-34.
[2] J. Bansiya and C. G. Davis, "A hierarchical model for Object-Oriented design quality assessment", IEEE Trans. on Software Engineering, vol. 28, no. 1, **(2002).**

[3] S. R. Chidamber and C. F. Kemerer, "A Metric Suite for Object-Oriented Design", IEEE Trans. on Software Engineering, vol. 20, no. 6, (**1994**), pp. 476-493.
[4] Q. Du and F. Wang, "Software Power: A New Approach to Software Metrics", 2010 Second WRI World Congress on Software Engineering.
[5] C. F. Kemerer and S. Slaughter, "An empirical approach to studying software evolution", IEEE Trans. on Software Engineering, (**1999**), vol. 25, no. 4, pp. 493-509.
[6] J. F. Girard, M. Verlage and D. Ganesan, "Monitoring the Evolution of an OO system with Metrics: An Experience from the Stock Market Software Domain", IEEE Int. conference on Software Maintenance, (**2004**) September 11-17, Chicago USA, pp. 360-367.
[7] J. Kothari, T. Denton, A. Shokoufandeh, S. Mancoridis and A. E. Hassan, "Studying the Evolution of Software System Using Change Clusters Proc. Of ICPC 2006 Int. Conf on program comprehension", Athens, Greece, (**2006)** June 14-16**.**
[8] W. Li, "Another metric suite for Object-Oriented programming", The Journal of Systems and Software, 1998, vol. 44, no. 2, (**1998**), pp. 155-162.
[9] T. J. McCabe, "A Complexity Measure ", IEEE Transaction on Software Engineering, vol. 2, (**1976**), pp. 308-320.
[10] V. Singh and V. Bhattacherjee, "New Complete Class Complexity Metrics" International Journal of Soft Computing and software Engineering, vol. 3, no. 9, (**2013**) September, ISSN: 2251-7545 Prefix DOI: 10.7321/jscse
[11] G. Booch, "Software Archaeology Rational users conference, (**2004).**
[12] Erlikh "Leveraging legacy system dollars for e-business" IEEE IT Pro May/June 2000, 17-23
[13] W. Li and J. Talburt, "Empirically Analysing Object-Oriented Software Evolution; Journal of Object-Oriented Programming", vol. 11, no. 5, (**1998**), pp. 15-19.
[14] G. Robles, J. M. Gonzalez-Barahona and I. Herraiz, "An Empirical Approach to Software Archaeology", Proc. Of 21st Int. Conf. on Software Maintenance (ICSM 2005), Budapest, Hungary, September 25-30, pp. 47-50.
[15] IEEE standard glossary of software engineering terminology. IEEE Std 610.12-1990.
[16] P. W. Oman, J. Hagemeister and D. Ash, "A Definition and Taxonomy for Software Maintainability", Technical Report #91-08-TR, Software Engineering Test Laboratory, University of Idaho, Moscow, ID, (**1991).**
[17] http://www.virtualmachinery.com/sidebar4.htm

Appendix A

Table 6. Complexity Increment Value of Different Version of JfreeChart

System Number	**System Name**	**Incr(WMC)**	**Incr(CMC)**	**Incr(CC)**	**Incr(CCC)**
1	jfreechart-0.9.0	0.00	0.00	0.00	0.00
2	jfreechart-0.9.1	0.00	0.24	0.06	0.35
3	jfreechart-0.9.2	0.00	0.17	0.41	2.21
4	jfreechart-0.9.3	0.22	0.19	0.20	-0.59
5	jfreechart-0.9.4	0.00	0.07	0.00	0.78
6	jfreechart-0.9.5	0.89	0.39	0.89	1.63
7	jfreechart-0.9.6	0.00	-0.03	0.00	0.00
8	jfreechart-0.9.7	0.33	0.32	0.81	4.09
9	jfreechart-0.9.8	0.00	0.00	0.00	0.00
10	jfreechart-0.9.9	0.11	0.12	0.19	0.44
11	jfreechart-0.9.10	0.00	0.00	0.13	0.13
12	jfreechart-0.9.11	0.33	0.33	0.44	1.22
13	jfreechart-0.9.12	0.33	0.34	0.29	0.09
14	jfreechart-0.9.13	0.11	0.38	0.23	1.44
15	jfreechart-0.9.14	0.22	0.17	0.45	0.00
16	jfreechart-0.9.15	0.00	0.00	0.04	0.00
17	jfreechart-0.9.16	-0.22	-0.37	-0.33	0.29
18	jfreechart-0.9.17	0.00	0.08	-0.11	1.83
19	jfreechart-0.9.18	-0.11	-0.09	-0.05	-0.21

20	jfreechart-0.9.19	0.44	0.57	0.62	0.89
21	jfreechart-0.9.20	0.22	0.18	0.04	1.59
22	jfreechart-0.9.21	-0.11	-0.02	-0.17	-2.47
23	jfreechart-1.0.0	0.56	0.52	0.68	3.87
24	jfreechart-1.0.1	0.00	0.00	0.00	0.00
25	jfreechart-1.0.2	-0.11	0.02	-0.02	1.86
26	jfreechart-1.0.3	0.11	0.14	0.51	0.65
27	jfreechart-1.0.4	-0.11	0.03	0.17	0.16
28	jfreechart-1.0.5	0.00	0.07	0.07	0.55
29	jfreechart-1.0.6	0.11	0.04	-0.22	-0.27
30	jfreechart-1.0.7	0.22	0.01	0.27	-0.15
31	jfreechart-1.0.8	0.00	0.02	0.00	0.16
32	jfreechart-1.0.8a	-0.22	-0.13	-0.17	-0.20
33	jfreechart-1.0.9	0.11	0.04	0.05	0.13
34	jfreechart-1.0.10	0.22	0.13	0.24	0.62
35	jfreechart-1.0.11	0.00	-0.02	-0.22	0.00
36	jfreechart-1.0.12	0.00	0.02	0.04	0.31
37	jfreechart-1.0.13	0.00	0.01	-0.11	-0.01
38	jfreechart-1.0.14	0.11	0.02	0.05	0.01

Table 7. Maintainability Index of Different Version of JfreeChart

System Number	System Name	Maintainability Index
1	jfreechart-0.9.0	53
2	jfreechart-0.9.1	71
3	jfreechart-0.9.2	63.25
4	jfreechart-0.9.3	88.35
5	jfreechart-0.9.4	74.69
6	jfreechart-0.9.5	74.16
7	jfreechart-0.9.6	73.69
8	jfreechart-0.9.7	74.73
9	jfreechart-0.9.8	74.29
10	jfreechart-0.9.9	74.82
11	jfreechart-0.9.10	74.15
12	jfreechart-0.9.11	66.53
13	jfreechart-0.9.12	171.00
14	jfreechart-0.9.13	66.60
15	jfreechart-0.9.14	65.65
16	jfreechart-0.9.15	65.45
17	jfreechart-0.9.16	79.85
18	jfreechart-0.9.17	69.05
19	jfreechart-0.9.18	95.87
20	jfreechart-0.9.19	65.68
21	jfreechart-0.9.20	65.00
22	jfreechart-0.9.21	63.17
23	jfreechart-1.0.0	171.00
24	jfreechart-1.0.1	74.00
25	jfreechart-1.0.2	79.00
26	jfreechart-1.0.3	83.01
27	jfreechart-1.0.4	71.07
28	jfreechart-1.0.5	78.85

29	jfreechart-1.0.6	83.19
30	jfreechart-1.0.7	81.09
31	jfreechart-1.0.8	80.00
32	jfreechart-1.0.8a	87.44
33	jfreechart-1.0.9	73.13
34	jfreechart-1.0.10	62.83
35	jfreechart-1.0.11	65.65
36	jfreechart-1.0.12	64.19
37	jfreechart-1.0.13	64.00
38	jfreechart-1.0.14	60.85

Table 8. Percentage Added and Removed Classes of Different Version of JfreeChart

System No i	System Name	Ai	Ai%	Ri	%Ri	Ci
1	jfreechart-0.9.0	0	0.0	0	0.0	210
2	jfreechart-0.9.1	23	11.0	0	0.0	233
3	jfreechart-0.9.2	11	4.7	0	0.0	244
4	jfreechart-0.9.3	137	56.1	2	0.8	379
5	jfreechart-0.9.4	45	11.9	20	5.3	404
6	jfreechart-0.9.5	102	25.2	30	7.4	476
7	jfreechart-0.9.6	3	0.6	0	0.0	479
8	jfreechart-0.9.7	115	24.0	7	1.5	587
9	jfreechart-0.9.8	7	1.2	0	0.0	594
10	jfreechart-0.9.9	102	17.2	79	13.3	617
11	jfreechart-0.9.10	11	1.8	26	4.2	602
12	jfreechart-0.9.11	28	4.7	2	0.3	628
13	jfreechart-0.9.12	56	8.9	28	4.5	656
14	jfreechart-0.9.13	19	2.9	0	0.0	675
15	jfreechart-0.9.14	41	6.1	10	1.5	706
16	jfreechart-0.9.15	22	3.1	2	0.3	726
17	jfreechart-0.9.16	19	2.6	6	0.8	739
18	jfreechart-0.9.17	91	12.3	36	4.9	794
19	jfreechart-0.9.18	27	3.4	5	0.6	816
20	jfreechart-0.9.19	55	6.7	16	2.0	855
21	jfreechart-0.9.20	15	1.8	2	0.2	868
22	jfreechart-0.9.21	32	3.7	250	28.8	650
23	jfreechart-1.0.0	165	25.4	42	6.5	773
24	jfreechart-1.0.1	3	0.4	1	0.1	775
25	jfreechart-1.0.2	58	7.5	2	0.3	831
26	jfreechart-1.0.3	82	9.9	0	0.0	913
27	jfreechart-1.0.4	36	3.9	4	0.4	945
28	jfreechart-1.0.5	15	1.6	0	0.0	960

29	jfreechart-1.0.6	16	1.7	1	0.1	975
30	jfreechart-1.0.7	21	2.2	10	1.0	986
31	jfreechart-1.0.8	2	0.2	0	0.0	988
32	jfreechart-1.0.8a	3	0.3	0	0.0	991
33	jfreechart-1.0.9	0	0.0	0	0.0	991
34	jfreechart-1.0.10	17	1.7	0	0.0	1008
35	jfreechart-1.0.11	21	2.1	1	0.1	1028
36	jfreechart-1.0.12	5	0.5	1	0.1	1032
37	jfreechart-1.0.13	31	3.0	0	0.0	1063
38	jfreechart-1.0.14	17	1.6	0	0.0	1080

Authors

Vinay Singh, he has received his Master of Computer Application degree from IGNOU, New Delhi, India in the year 2003 and Master of Technology in Computer science and Engineering from BITs Mesra, Ranchi, India in 2009. He is pursuing Ph.D from BIT's Mesra, India. Presently he is working as an Associate Dean of Information Technology in UMESL, Kolkata, India since 2008. He is also empanelled with Wipro Technologies as a corporate trainer. He has published twelve papers in the International Journal and Conference. His Research area is Software Metrics and Quality.

Vandana Bhattacherjee, she is working as a Professor, Department of Computer Science and Engineering, Birla Institute of Technology, Ranchi. She completed her B. E. (CSE) in 1989 and her M. Tech and Ph. D in Computer Science from JNU New Delhi in 1991 and 1995 respectively. She has over 100 National and International publications in Journal and Conference Proceedings. She is a member of IEEE Computer Society and Life Member of Computer Society of India. Her research areas include Software Process Models, Software Cost Estimation, Data Mining and Software Metrics.