International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Evaluation of OO Software Quality by Using Predictive Object Points (POP) Metric

Vijay Yadav[1], Raghuraj Singh[2], Vibhash Yadav[3]

Submitted: 25/10/2022 **Revised:** 24/12/2022 **Accepted:** 21/01/2023

Abstract— Software quality assessment is essential as it reduces costs associated with the allocation of testing and maintenance resources. Software quality metrics based on object-oriented technologies will become increasingly important as object-oriented technologies become more prevalent. This paper examines how Predictive Object Point software sizing metrics can help designers assess an OO system's quality during the planning phase. By combining high-level quality attributes such as extendibility, reusability, flexibility, understandability, functionality, and effectiveness, POPs Metric makes it easy to make quality decisions. Different versions of three OO software were tested for the exact necessities and aimed to evaluate the proposed quality assessment model. Various design metrics have been measured using a quality metric tool to assess the quality of the projects under study. We compare POP counts with these quality attributes to see what's trending. Analysis and presentation of the results demonstrate that the POP Count can be used to assess the quality of OO software.

Keywords**—***Quality measurement, quality model, quality attributes, Object Orientation, Object-Oriented Software Metrics, Predictive Object Point Metric.*

1. Introduction

With the increasing demand for quality software in today's software development environment, Software engineering practices have increasingly been dominated by object-oriented technologies (OO).

Software developers and managers were forced to rethink the parameters or elements used to estimate the size of and assess software's quality as object-oriented analysis and design methodologies gained popularity. Because different metrics measure different parameters, the way they measure, and the application of those metrics varies, the results vary from metric to metric.

The introduction and subsequent application of metrics for predicting software quality is an area that is particularly well suited for predictive analysis. [2].

It's hard to test the practicality and effectiveness of some metrics in an industrial setting because most have not been validated or tested with small data sets. An external quality attribute and a measured metric value should be mapped together to ensure the software develops a high level of external quality. OO software analysis models tend to be applicable only during the implementation phase of a project and therefore do not assist in improving the software characteristics before the project's completion. To ensure that software end products are of high quality, it is imperative to assess software quality early in the development process.

A fundamentally different concept embodied in object orientation is inheritance, encapsulation, and polymorphism, which should be emphasized instead of traditional metrics evaluating product characteristics such as size, performance, quality, and complexity. [1].

With an Object-Oriented approach, it is naturally possible to evaluate and assess the project at an early stage [1]. As a result, different scientists have developed and used a variety of metrics related to product quality in order to meet the requirements [[5][6][7][8]].

However, it is essential to note that the results can vary from metric to metric due to the difference in parameters, the method by which they measure, and the timeframe in which they are applied.

A quality model is generally applicable to the analysis and development of OO software, but it is not applicable to the improvement of its characteristics prior to its completion. Numerous researchers have suggested that

[1]*Department of CSE,*
Dr. A. P. J Abdul Kalam Technical University, Lucknow, Uttar Pradesh, India vijayyadavuiet@gmail.com
[2]*Department of C.S.E,*
Harcourt Butler Technical University, Kanpur, Uttar Pradesh, India.
raghurajsingh@rediffmail.com
[3]*Department of I.T,*
Rajkiya Engineering College,
Banda, Uttar Pradesh, India.
vibhashds10@gmail.com

different methods can be used to measure software quality. For example, object-oriented design assessment can be assessed using a hierarchical model called QMOOD [1].

This tool maps source code metrics to abstract quality attributes such as understandability, extendibility, flexibility, reusability, functionality and effectiveness,

As long as these purposes and aims do not overlap, we can use a variety of quality attributes to represent them. It is not possible to directly observe these quality attributes since they are abstract concepts. [1].

Using the QMOOD quality model, Table 1 summarizes the computation formula for quality attributes according to design properties.

Table 1: Defining Quality Attributes based on Formulas [1]

Quality Attribute	Equation for calculating the index
Reusability	minus (25/100) x coupling plus (25/100) x cohesion plus (5/10) x messaging plus (5/10) x design size
Flexibility	minus (25/100) x encapsulation minus (25/100) x coupling plus (5/10) x composition plus (5/10) x polymorphism
Understandability	minus (33/100) x abstraction plus (33/100) x encapsulation minus (33/100) x coupling plus (33/100) x cohesion minus (33/100) x polymorphism minus (33/100) x complexity minus (33/100) x design size
Functionality	(12/100) x cohesion plus (22/100) x polymorphism plus (22/100) x messaging plus (22/100) x design size plus (22/100) x hierarchies
Extendibility	(5/10) x abstraction minus (5/10) x coupling plus (5/10) x inheritance plus (5/10) x polymorphism
Effectiveness	(2/10) x abstraction plus (2/10) x encapsulation plus (2/10) x composition plus (2/10) x inheritance plus (2/10) x polymorphism

In addition, it is important to note that the metrics employed to assess messaging, coupling, cohesion, encapsulation, complexity, polymorphism, hierarchies, abstraction, and design size are not fixed and can instead be replaced with their alternatives. [1].

A comparison of QMOOD design metrics and corresponding replacement metrics is presented in Table 2.

Table 2: A Description of QMOOD Design Metrics and some alternatives.

Design Properties	Metrics in QMOOD[1]	Equivalent Metric Computed
Encapsulation	(DAM)	-
Polymorphism	NOP	NMO [14]
Abstraction	ANA	TLC [2]
Cohesion	CAM	-
Coupling	DCC	Ce [13]
Inheritance	MFA	-
Complexity	NOM	WMC [4]
Messaging	CIS	NPM [16]
Composition	MOA	-
Design Size	DSC	NOC [15]
Hierarchies	NOH	DIT [4]

In the above table, DAM stands for Data Access Metrics. NOP stands for Number Of Polymorphic Methods, NMO stands for the Number Of Method Overridden, ANA stands for Average Number Of Ancestors, TLC stands for Top Level Class, CAM stands for Cohesion Among Method In Class, DCC stands for Direct class coupling, Ce stands for Efferent Coupling, MFA stands for Measure Of Functional Abstraction, NOM stands for Number Of Methods, WMC

stands for Weighted Method per Class, CIS stands for Class Size Interface, NPM stands for Number Of Public Methods, MOA stands for The measure of Aggregation, DSC stands for Design Size In Class, NOC stands for Number Of Class, NOH stands for Number Of Hierarchies, DIT stands for Depth Of Inheritance respectively.

I. ANALYSIS OF QUALITY USING PREDICTIVE OBJECT POINT METRICS

A method for determining the effort needed to develop an object-oriented software system was introduced by Minkiewicz in 1998[3]. A POP is designed to improve upon FPs. It consists of counting two metrics: the number of TLC and the WMC, along with adjustments for the average DIT and the average NOC.

Based on the MOOSE metrics suite developed by Chidamber and Kemerer, the WMC, DIT, and NOC are derived [6].

It is important to note that POPs [3] incorporate three dimensions of OO systems: the number of functions provided by the software, communication between objects, and reuse through inheritance. To indicate how much work is involved in the development of a software system, these aspects were combined into one metric.

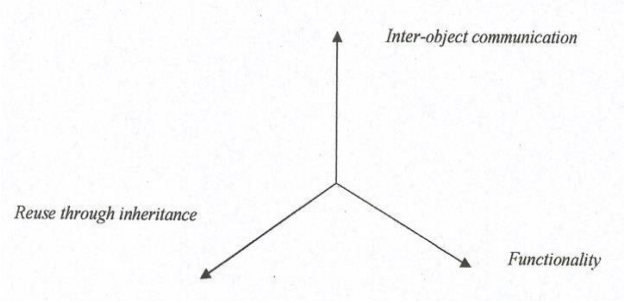

Fig 1 - Object-oriented systems: aspects and characteristics [3]

Since POPs are based on objects and their features, they fulfill most of the criteria that must be met by OO concepts.

Methodology for measuring: As a method of calculating the Overall system size, the following formula was proposed [2].

$$f1(TLC, NOC, DIT) = TLC * (1 + ((1 + NOC) * DIT)^{1.01} + (|NOC - DIT|)^{.01})$$

$$f2(NOC, DIT) = 1.0$$

$$POPs(WMC, NOC, DIT, TLC) = \frac{WMC * f1(TLC, NOC, DIT)}{7.8} * f2(NOC, DIT)$$

(1)

F1 attempts to estimate the overall system size, while F2 considers how inheritance affects reuse.

In most cases, estimation begins with the creation of a size estimate for the software to be developed. A reasonable size estimate is essential to accurately estimating the amount of effort, schedule, and quality. The POP metrics validated through the APA tool provide a good indication of the size of the software to be produced. [4].

POP Count's software measurement metrics incorporate almost all design metrics used in QMOOD's assessment of high-level quality attributes [1]. As part of the POP count formula, WMC consists of functionality and inter-object communication [3]. WMC analyzes the class structure, and the results have a bearing on how understandable, maintainable, and reusable the system is as a whole [12]. Generally, the reuse of a system is accomplished through inheritance and overall system size by the average DIT, TLC, and NOC. [3]. It also evaluates efficiency, reuse, and testability. DIT has an emphasis on efficiency and reuse and also relates to testability and understandability. [12].

As part of the quality assessment through the POP count, the data set may be viewed as a group of projects with identical requirements and objectives to ascertain if the POP metrics can predict the quality of software across object-oriented languages.

2. Details of the Empirical Study

POP Count is a model for quality measurement based on POPs. Therefore, the validation designs were selected based on similar requirements and objectives.

A. Set of projects taken

In this study, several versions of three projects, JaimBot[9], JCommon[10], and proguard[11], are analyzed. JaimBot [9] provides a generic AIM library that can be used by a bot to provide services such as Lists, Stock Quotes, Weather, Headlines, Offline Messaging, and Artificial Intelligence chatterbots. JFreeChart, Pentaho Reporting, and a few other projects use JCommon [10], a Java class library containing packages such as date, IO, layout, resources, and user interface. ProGuard[11] is also a command-line utility that shrinks, optimizes, obfuscates, and pre-verifies Java classes. Java class files are also pre-verified for Java 6 or higher or Java Micro Edition in addition to detecting and removing unused classes, fields, methods, attributes, and instructions. Each of these designs has multiple versions that are widely used in real-life software-based development, and they all work. Based on the metrics of design in Table 2 and the attributes for quality in Table 1, four versions of JaimBot [9], three versions of JCommon

[10], and three versions of proguard [11] were evaluated.

B. A Method for Normalizing Measured Metric Values

The QMOOD quality attribute values are calculated by combining actual metric values in different ranges. Therefore a normalization of the results of various metrics is performed in the first version of the specification. However, as the comparison is between different versions of the same project, this is acceptable as it is calculated by dividing the metric values by the metric value in the first version. As per consideration, if a metric value is zero before normalization, then that metric value is not normalized as it falls between [min, max], where zero is considered the min value, avoiding the 0/0 form. In the case of projects of different types, the above normalization technique cannot be implemented.

C. A Tool For Automating The Process

A tool has been developed to analyze the above designs. A total of eleven metrics from Table 2 have been collected for each of the four versions of JaimBot[9], three versions of JCommon[10], and three versions of Proguard[11].

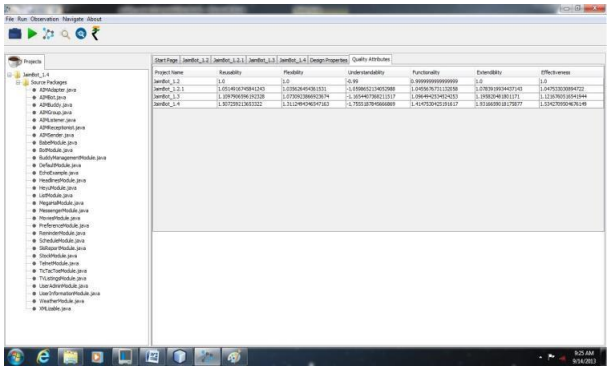

Fig 2: The values of the sample quality attributes

Figure 2 illustrates the snapshot of the quality tool, which evaluates the quality attributes and POP count of the software.

3. Results Of Analysis

New versions of existing software usually add new features or fix bugs found in previous versions. Most software is modified early on to improve functionality that meets additional needs or add new functionalities.

As a result, Software that is released early may be more user-friendly and easier to use. In terms of quality, they're way better than the previous generation. The improvements aren't that big for higher versions.

For several designs, QMOOD quality attributes are computed using an automation tool, and the trends of these attributes are compared with POP counts computed with APA. [4].

It's important that the evaluated quality characteristics of the three projects match the overall trends derived from QMOOD's half-a-dozen extraordinary-level quality attributes for validation of the proposed model.

It's estimated that reusable, flexible, functional, extendible, and effective features will increase with each release, while understandability should decrease with an increase in complexity.

D. Jaimbot Project Versions Evaluation Results

Using an automated tool, we have gathered the values for eleven metrics in Table 2 for the four versions of JaimBot [9]. Then, we normalized the values and presented them in Table 3.

Table 3: Metric Values for JaimBot Project Versions: Actual and Normalized.

Project	**Actual Metric Values**				**Normalized Metric Values**			
Versions / Metric	**1.2**	**1.2.1**	**1.3**	**1.4**	**1.2**	**1.2.1**	**1.3**	**1.4**
Design Size	162	173	182	249	1	1.07	1.12	1.54
Hierarchies	10	10	10	0	1	1	1	1
Coupling	84	88	94	120	1	1.05	1.12	1.43
Cohesion	10.28	10.71	10.88	12.39	1	1.04	1.06	1.21
Abstraction	20	21	23	33	1	1.05	1.15	1.65
Encapsulation	13.17	13.61	14.46	18.17	1	1.03	1.09	1.37
Messaging	153	164	178	251	1	1.03	1.12	1.58

Polymorphism	51	55	59	84	1	1.07	1.16	1.64
Complexity	210	219	239	345	1	1.04	1.14	1.64
Composition	2	2	2	2	1	1	1	1
Inheritance	0	0	0	0	0	0	0	0
POP COUNT	**458.60**	**522.50**	**584.58**	**968.71**	**1**	**1.07**	**1.20**	**1.99**

For the different versions of JaimBot, Table 4 shows the computed values of the six quality attributes based on normalization.

Table 4: Versions of JaimBot Projects with QMOOD Quality Attribute Values and POP Counts.

Version / Quality Attribute	**1.2**	**1.2.1**	**1.3**	**1.4**
Reusability	1	1.05	1.10	1.50
Flexibility	1	1.03	1.07	1.31
Understandability	-0.99	-1.05	-1.16	-1.75
Functionality	0.99	1.04	1.09	1.41
Extendibility	1	1.07	1.19	1.93
Effectiveness	1	1.04	1.12	1.53
POP COUNT	**1**	**1.07**	**1.20**	**1.99**

Based on the values listed above for all four versions of Jaimbot, it is evident that the attributes which measure quality, such as that reusable, flexible, functional, extendible, and effective metrics, will increase with each release, whereas understandability decreases as complexity increases in higher versions.

Furthermore, based on the graph below, it is evident that effectiveness, reusability, flexibility, functionality, and extendibility factors increase with higher versions while understandability factors decrease.

There was also an increase in the POP count of all four versions of JaimBot.

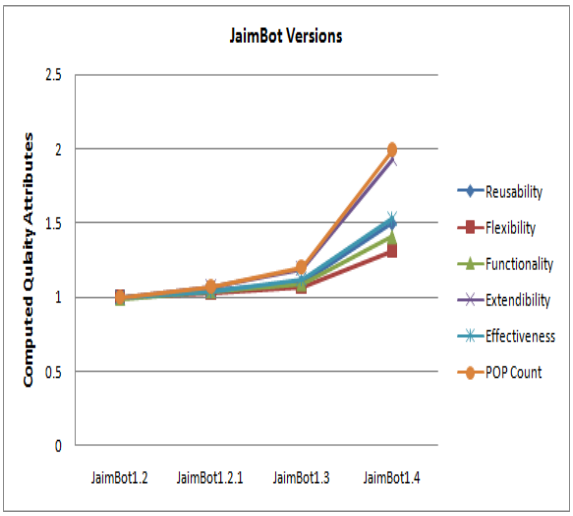

Fig 3: Plot of JaimBot Project Version's computed quality attributes & POP count

E. Results Of The Evaluation Of Jcommon Project Versions

The automated tool collects eleven metrics for three different versions of Jcommon [10]. The values measured are normalized and presented in Table 5.

Table 5: Jcommon Projects Metric Values: Actual and Normalized

Project	**Actual Metric Values**			**Normalized Metric Values**		
Versions / Metric	**0.8.0**	**0.9.0**	**1.0.0**	**0.8.0**	**0.9.0**	**1.0.0**
Design Size	357	419	483	1	1.17	1.35
Hierarchies	184	201	214	1	1.09	1.16
Coupling	74	81	93	1	1.09	1.26
Cohesion	41.09	45.85	51.22	1	1.12	1.25
Abstraction	75	84	99	1	1.12	1.32
Encapsulation	42.33	48.23	59.66	1	1.14	1.41
Messaging	378	428	525	1	1.13	1.39
Polymorphism	5	7	7	1	1.4	1.4
Complexity	457	531	647	1	1.16	1.42
Composition	7	43	55	1	6.14	7.86
Inheritance	35.53	36.89	37.75	1	1.04	1.06
POP COUNT	**1792.79**	**2082.55**	**2560.36**	**1**	**1.16**	**1.43**

As a result of normalizing the POP counts, Table 6 shows the six quality attributes for each version of Jcommon.

Table 6: Jcommon Project Versions with QMOOD Quality Attribute Values.

Version / Quality Attribute	**0.8.0**	**0.9.0**	**1.0.0**
Reusability	1	1.16	1.36
Flexibility	1	3.78	4.67
Understandability	-0.99	-1.21	-1.34
Functionality	1	1.18	1.32
Extendibility	1	1.24	1.26
Effectiveness	1	2.17	2.61
POP COUNT	**1**	**1.16**	**1.43**

For all three versions of Jcommon, the values listed above indicate an increase in effectiveness, reusability, flexibility, functionality, and extendability and a decrease in understandability.

The graph below indicates that effectiveness, reusability, flexibility, functionality, and extendability increase with higher versions, but the understandability factors decrease. In addition, all three versions of Jcommon increased their POP count.

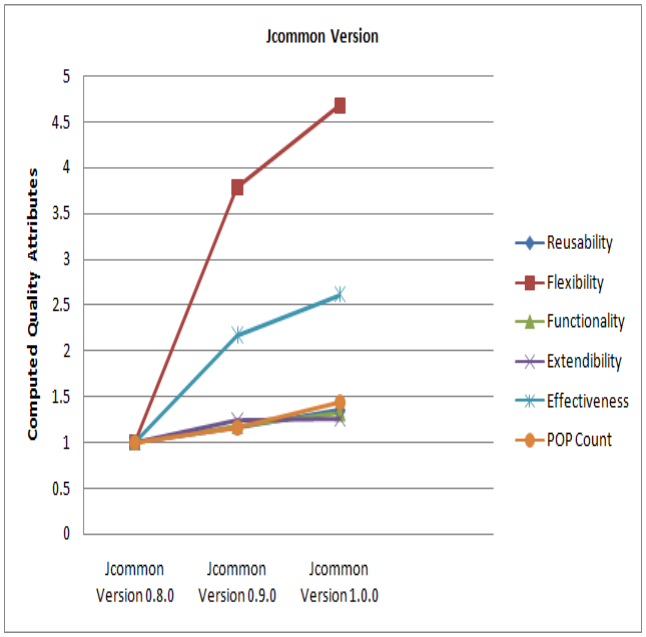

Fig 4: Plot of computed quality attributes & POP Count for Jcommon Project Versions

F. Results Of Evaluation for Proguard Project Versions

The values of the 11 metrics of Table 2 for the three varieties of Proguard[11] are measured by the automated tool. The measurements are normalized and presented in Table 7.

Table 7: Proguard Project Version Metric Values: Actual and Normalized

Project	**Actual Metric Values**			**Normalized Metric Values**		
Version / Metrics	**1.7.2**	**4.0**	**4.9**	**1.7.2**	**4.0**	**4.9**
Design Size	257	497	556	1	1.93	2.16
Hierarchies	23	45	44	1	1.96	1.91
Coupling	284	594	690	1	2.09	2.43
Cohesion	8.06	26.03	29.26	1	3.23	3.63
Abstraction	33	89	107	1	2.69	3.24
Encapsulation	16.8	46.75	50.75	1	2.78	3.02
Messaging	167	299	331	1	1.79	1.98
Polymorphism	4	8	8	1	2	2
Complexity	253	405	482	1	1.60	1.91
Composition	4	5	8	1	1.25	2
Inheritance	1	1	1	1	1	1
POP COUNT	**1142.71**	**2102.89**	**2531.81**	**1**	**1.84**	**2.22**

In Table 8, the six quality attributes, along with POP Count based on the normalization, have been computed for each version of the project Proguard.

Table 8: Proguard Project Versions with QMOOD Quality Attributes and POP Counts

Version / Quality Attribute	1.7.2	4.0	4.9
Reusability	1	2.14	2.37
Flexibility	1	1.79	2.15
Understandability	-0.99	-1.42	-1.68
Functionality	1	2.07	2.21
Extendibility	1	1.8	1.9
Effectiveness	1	1.94	2.25
POP COUNT	**1**	**1.84**	**2.22**

In all three versions of Proguard, the attributes which measure quality such as that reusable, flexible, functional, extendible, and effective metrics will increase with each release, whereas understandability decreases as complexity increases in higher versions.

According to the graph below, with higher versions, the factors of effectiveness, reusability, flexibility, functionality, and extendability increase while those related to understandability decrease.

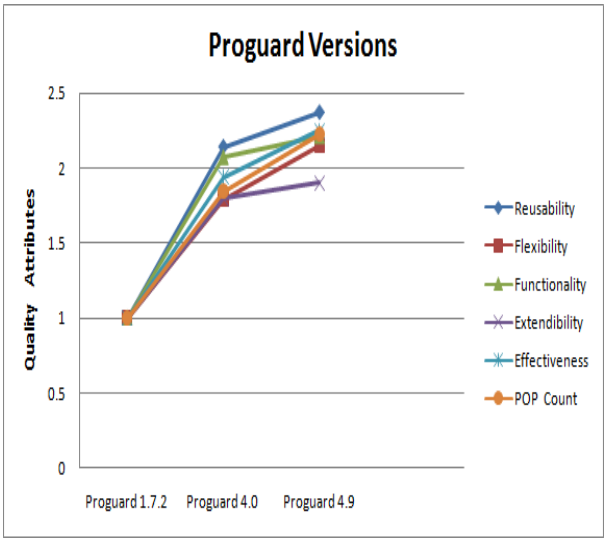

Fig 5: A Plot depicting the computed quality attributes and POP count for each version of the Proguard project.

The above results demonstrate that as different versions are released, the effectiveness, reusability, flexibility, functionality, and extendability factors increase, and the understandability factor decreases.

POP counts for all three versions of the projects were also found to have increased.

5. Conclusion And Future Work

A tool for measuring software quality has been developed. With this tool, the trend of QMOOD quality attributes was measured for three Java projects and compared to the trend shown by the POP count values for all the identical versions. Quality was analyzed.

From this study, it is evident that the POP metric is a good indicator of Software Quality, as demonstrated by comparing the results obtained.

Based on the results of the study, we can conclude that the POP count is an appropriate method for estimating Effectiveness, Reusability, Flexibility, Functionality, and Extendability quality attributes. As a result of the POP count, however, understandability is indirectly proportional to the POP count.

In this manner, it is possible to estimate the quality of projects by comparing their POP counts. The increase in POP count value corresponds to an increase in the quality

attributes of effectiveness, reusability, flexibility, functionality, and extendability, as well as a decrease in understandability.

Predictive Object Point Metrics and Quality are related to the data studied. However, the data studied has not covered many software projects exhaustively.

Further research is required to determine if this relationship exists by obtaining additional projects developed for similar requirements and objectives. Validation is key to making sure such predictions are accurate, and software quality assessment is effective.

Since it has already been demonstrated that the POP metrics set is also a good predictor of size, it is possible to follow up this study with another in which the POP metrics will be mapped to measure the cost and schedule of software development.

References

[1] J.Bansiya and C.G.Davis, "A Hierarchical Model for Validation of Object Oriented Design Quality Assessment", IEEE Transactions on Software Engineering, Vol.28, No 1, January 2002.

[2] N. Kayarvizhy, S. Kanmani," Analysis of Quality of Object Oriented Systems using Object Oriented Metrics", in proceedings of 3rd International Conference on Electronics Computer Technology (ICECT), Volume 5, pg 203-206, April 2011.

[3] Arlene F. Minkiewicz, "Measuring Object Oriented Software with Predictive Object Points" PRICE Systems, LLC 609-866-6576.

[4] Jain Shubha, Yadav Vijay and Singh Raghuraj. "OO Estimation through Automation of the Predictive Object Point Sizing Metric", International Journal Of Computer Engineering and Technology(IJCET), Volume 4, Issue 3, May-June (2013), pp. 410-418.

[5] J. Bansiya," A Hierarchical Model for Quality Assessment of Object Oriented Designs", Ph.D. Dissertation, Univ. Of Alabama in Huntsville, 1997.

[6] S. R. Chidamber and C. F. Kemerer. A Metrics Suite for Object-Oriented Design. IEEE Transactions on Software Engineering, vol.20, No 6, pp. 476 - 493, June 1994.

[7] M. Hintz and B. Montazeri," Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective ", IEEE Trans. Software Eng, vol. 22, No. 4, pp. 67-271, Apr.1996.

[8] W. Li and S. Henry, "Object Oriented Metrics that Predict Maintainability," The J. Systems and software vol.23, no. 21, pp. 929-994, Dec. 1995.

[9] JaimBot : URL http://sourceforge.net/projects/jaimbot/

[10] Jcommon : URL http://www.jfree.org/jcommon/download

[11] Proguard : URL http://proguard.sourceforge.net

[12] Software Quality Metrics for Object Oriented System Environments, National Aeronautics and Space Administration Goddard Space Flight Center, Greenbelt Maryland 20771, June 1995.

[13] Mandeep K. Chawla, Dr. Indu Chhabra," Capturing OO Software Metrics to attain Quality Attributes – A case study", International Journal of Scientific & Engineering Research, Volume 4, Issue 6, June-2013 ISSN 2229-5518

[14] Objecteeering Metrics User Guide available at: http://support.objecteering.com/objecteering6.1/help/us/metrics/toc.htm [accessed 25 June 2013].

[15] User Guide for CCCC available at : http://www.stderr.org/doc/cccc/CCCC%20User%20Guide.html [accessed 25 June 2013]

[16] CKJM extended manual," An extended version of Tool for Calculating Chidamber and Kemerer Java Metrics (and many other metrics), " available at: http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/intro.html [accessed 3 May 2013]