ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645
Indian Journal of Science and Technology, *Vol 11(20), DOI: 10.17485/ijst/2018/v11i20/48215, May 2018*

Predictive Object Points (POP) Sizing Metric: A Good Predictor of Quality of OO Software

Shubha Jain[1]*, Shantanu Pant[1] and Raghuraj Singh[2]

[1]Department of Computer Science and Engineering, Kanpur Institute of Technology, Kanpur – 208001, Uttar Pradesh, India; shubhj@rediffmail.com, shantanupant03@gmail.com

[2]Computer Science and Engineering Department, Harcourt Butler Technological Institute, Kanpur – 208002, Uttar Pradesh, India; raghurajsingh@rediffmail.com

Abstract

Measuring the quality of software is an essential task as it leads to the minimization of cost in allocation of resources for testing or maintenance effort. With the emergence of Object Oriented (OO) technologies as a dominant software engineering practice today, it is required to investigate object-oriented metrics with respect to the software quality. This paper is an attempt for measuring the quality attributes of an OO system during the design phase using Predictive Object Point software sizing metrics set in. This paper relates high-level quality attributes such as reusability, flexibility, understandability, functionality, extendibility and effectiveness to Predictive Object Point Count and hence shows that Predictive Object Point Metrics set can be used to make quality decisions. The proposed model of assessment of quality through POP Count at the design phase has been studied on the several separate versions of three object oriented software which are developed for the same types of requirements and objectives. A quality metric tool has been developed to measure the various design metrics and hence the quality attributes of the projects under study. The trend observed through these quality attributes is compared with the corresponding POP Count values. The results have been analyzed and presented to show that the POP Count can be used to assess the quality of an object oriented system.

Keywords: Automation, Object Orientation, Predictive Object Point, Quality Attributes, Quality Measurement, Quality Model, Software Metrics,

1. Introduction

Object Oriented (OO) technologies have become a dominant software engineering practice today. The demand for quality software grows day by day in today's software development environment. With object-oriented analysis and design methodologies gaining popularity, the software developers and managers had to rethink about the parameters or elements used to estimate the size as well as to assess the software quality. However, the results vary from metric to metric due to the different parameters they measure, the way they measure and when they are applicable.

Software quality is especially a favored area when it comes to prediction based on metrics. The introduction and subsequent use of metrics as a means to evaluate the software quality has had deep and useful impact on the overall system[2].

Most of the metrics have not been validated or validated with small data sets therefore their practical applicability and effectiveness in an industrial environment is not known. There should be a way to map the external quality attributes of the software developed with the measured metrics values. Most of the quality models for analyses of OO software are applicable during their implementation and hence does not help in improving the software characteristics before the completion of the project. Thus there is a need to assess the quality of software in early stages of development to ensure quality end products.

Traditional software product metrics that evaluate product characteristics such as size, complexity, per-

*Author for correspondence

formance, and quality must be changed to rely on some fundamentally different notions such as encapsulation, inheritance, and polymorphism which are inherent in object-orientation[1]. The object oriented approach naturally lends itself to an early assessment and evaluation[1]. Many metrics relating to product quality have been developed and used by various scientists in order to meet the requirement[5–8]. However, the results may vary from metric to metric due to the different parameters they measure, the way they measure and when they are applicable.

Most of the metrics have not been validated or validated with small data sets therefore their practical applicability and effectiveness in an industrial environment is not known. Most of the quality models for analyses of OO software are applicable during their implementation and hence does not help in improving the software characteristics before the completion of the project. There are various models suggested by various researchers to measure software quality. One such model is QMOOD[1] which is a hierarchical model for object oriented design assessment. It Implements a way to map source code metrics to higher abstract quality attributes like reusability, functionality, effectiveness, understandability, extendibility and flexibility. However, this set of quality attributes is not exclusive, and it can be easily changed to represent different objectives and goals. These quality attributes are abstract concepts and are therefore not directly observable[1].

QMOOD quality model describes the way to compute the quality attributes in terms of design properties through computational formulas mentioned in Table 1.

The metrics used for the measurement of the above design properties such as messaging, coupling, cohesion, encapsulation, complexity, polymorphism, hierarchies, abstraction and design size are not fixed and hence may be replaced with their corresponding replacement metrics[1].

Table 2 shows QMOOD design metrics and corresponding replacement metrics.

Table 1. Computation formulas for quality attributes[1]

Quality Attribute	Index Computation Equation
Reusability	- 0.25 * coupling + 0.25 * cohesion + 0.5 * messaging + 0.5 * design size
Flexibility	0.25 * encapsulation - 0.25 * coupling + 0.5 * composition + 0.5 * polymorphism
Understandability	- 0.33 * abstraction + 0.33 * encapsulation - 0.33 * coupling + 0.33 * cohesion - 0.33 * polymorphism - 0.33 * complexity - 0.33 * design size
Functionality	0.12 * cohesion + 0.22 * polymorphism + 0.22 * messaging + 0.22 * design size + 0.22 * hierarchies
Extendibility	0.5 * abstraction - 0.5 * coupling + 0.5 * inheritance + 0.5 * polymorphism
Effectiveness	0.2 * abstraction + 0.2 * encapsulation + 0.2 * composition + 0.2 * inheritance + 0.2 * polymorphism

Table 2. QMOOD design metrics and some substitute metrics

Design Properties	Metrics in QMOOD[1]	Equivalent Metric Computed
Coupling	Direct class coupling (DCC)	Efferent Coupling (Ce)[13]
Cohesion	Cohesion Among Method In Class (CAM)	-
Encapsulation	Data Access Metrics (DAM)	-
Abstraction	Average Number Of Ancestors (ANA)	Top Level Class (TLC)[2]
Hierarchies	Number Of Hierarchies (NOH)	Depth Of Inheritance (DIT)[4]
Polymorphism	Number Of Polymorphic Methods (NOP)	Number Of Method Overridden (NMO)[14]
Complexity	Number Of Methods (NOM)	Weighted Method per Class (WMC)[4]
Messaging	Class Size Interface (CIS)	Number Of Public Methods (NPM)[16]
Design Size	Design Size In Class (DSC)	Number Of Class[15]
Composition	Measure of Aggregation (MOA)	-
Inheritance	Measure Of Functional Abstraction (MFA)	-

2. Quality Assessment through Predictive Object Point (POP) Metrics

Predictive Object Points (POPs) was proposed by Minkiewicz in 1998[3] for predicting effort required for developing an object oriented software system. POPs are intended as an improvement over FPs, and are based on counts of metrics: number of Top Level Classes (TLC) and weighted methods per class (WMC), with adjustments for the average Depth of the Inheritance Tree (DIT) and the average Number Of Children per class (NOC).

WMC, DIT, and NOC are taken from the MOOSE metrics suite of Chidamber and Kemerer[6]. POPs[3] incorporate three dimensions of OO systems: the amount of functionality the software delivers communication between objects and reuse through inheritance. These aspects used to give rise to a single metric in order to indicate the amount of effort involved in the production of a software system.

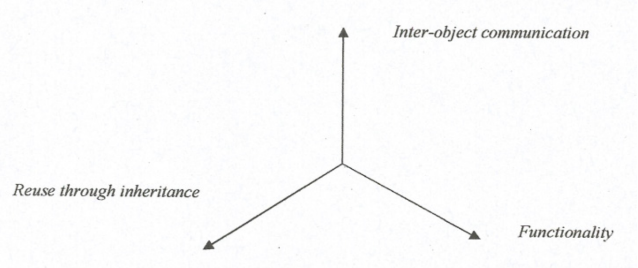

Figure 1. Aspects of an object-oriented system[3].

POPs are based on objects and their characteristics. It fulfilled almost all the criteria of OO concepts.

Measurement process: The following formula was proposed to calculate the size of the overall system[2].

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

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

$$POPs(WMC, NOC, DIT, TLC) = \frac{WMC * f1(TLC, NOC, DIT)}{7.8} * f2(NOC, DIT) \quad (1)$$

where, f1 attempts to size the overall system, and f2 applies the effects of reuse through inheritance.

Typically, estimation begins by projecting the amount of software to be produced. Getting a good size estimate is essential to getting good estimates of effort, schedule, and quality. The POP metric is a good indicator of software size validated through APA tool[4].

The metrics used in POP Count for the measurement of software incorporate almost all the design metrics required for the assessment of high-level quality attributes suggested by QMOOD[1]. WMC used in POP count formula encompass both functionality and inter-object communication in POPs count[3]. WMC analyzes the class structure and the result has a bearing on the understandability, maintainability, and reusability of the system as a whole[12]. The average DIT, TLC and Average NOC establishes reuse through inheritance and overall system size[3]. It also evaluates efficiency, reusability, and testability. DIT also evaluates efficiency and reuse and also relates to understandability and testability[12].

For Quality assessment through POP count, data set may be taken as projects with identical requirements and objectives. This would help to ascertain that the POP metrics are capable of predicting the quality of software across the object oriented language.

3. Empirical Study Description

The validation of the proposed model for quality measurement through POP Count was carried out. Designs chosen for validation are developed for similar requirements and objectives.

3.1 Project Set Taken

Several versions of three projects, JaimBot[9], JCommon[10] and proguard[11] are chosen for this study. JaimBot[9] is a modular architecture for providing services through an AIM client. It contains a generic AIM library and a Bot which uses this library to provide such services as Offline Messaging, Lists, Weather, Headlines, Stock Quotes, AI chatterbot. JCommon[10] is a Java class library contains packages such as date, io, layout, resources, ui etc. that is used by JFreeChart, Pentaho Reporting and a few other projects. ProGuard[11] is a command-line tool with an optional graphical user interface. It is a free Java class file shrinker, optimizer, obfuscator, and pre-verifier. It detects and removes unused classes, fields, methods, attributes and instructions. It also pre-verifies the processed code for Java 6 or higher, or for Java Micro Edition. All are commercial successful object- oriented designs that are extensively used in real-world software development and several versions of designs exist for comparison.

Four versions of JaimBot[9], three versions of JCommon[10] and three versions of proguard[11] were evaluated using the suite of design metrics in Table 2 and the quality attributes in Table 1.

3.2 Procedure for Normalizing Measured Metric Values

For computation of the QMOOD quality attribute values, actual metric values of different ranges are combined, hence normalization is done with respect to the metrics' values in the first version. This is obtained by dividing the metric values with the metric value in the first version. This is acceptable as the comparison is made between the different versions of the same project. If a metric value is zero prior to normalization, then that metric values are not normalized as per consideration the normalized value fall between [min, max] where min value considered is zero, thus avoiding 0/0 form. The above normalization technique cannot be implemented if the projects considered are of different types.

3.3 Automated Tool

An automation tool has been built to analyze the above designs. The metric values are collected for eleven metrics of Table 2 for the four versions of JaimBot[9], 3 versions of JCommon[10] and 3 versions of Proguard[11] and then are normalized.

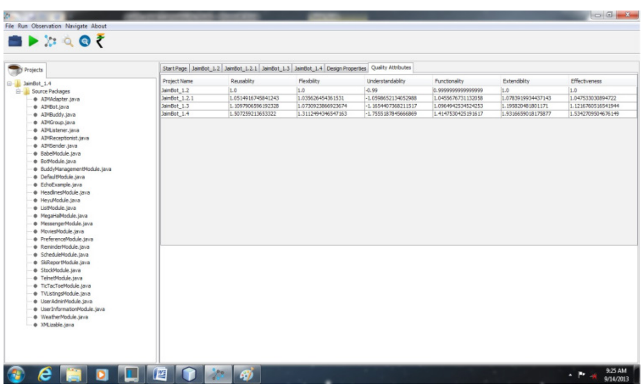

Figure 2. Sample quality attributes values.

Figure 2 shows the snapshot of the quality tool which assesses the quality of the software by evaluating the quality attributes and POP Count.

4. Analysis Results

Generally new versions of an existing software product add new features or eliminate errors discovered in previous version. In early versions generally software is modified to enhance capabilities and to add new features or incorporate additional requirements. Thus early releases may improve the usability and user friendliness of the software product. Their quality has generally been observed to be significantly better than their predecessors. After that the improvements are small for higher versions.

The quality attributes proposed in QMOOD are computed for several designs through an automation tool and compared with the trend observed through POP Count computed through APA Tool[4].

For the validation of the proposed model, it was expected that the evaluated quality characteristics for each version of the three projects through POP Count should match with the generally expected trends obtained of the six high-level quality attributes in the QMOOD model.

The Expected trend is that the quality attributes reusability, flexibility, functionality, extendibility, and effectiveness should increase from one release to the next and understandability should decrease with increase in complexity in higher versions.

4.1 Evaluation Results for Jaimbot Project Versions

The metric values are collected for eleven metrics of Table 2 for the four versions of JaimBot[9] through the automated tool. The values measured are normalized and presented in Table 3.

Table 3. Actual and normalized metric values for JaimBot projects versions

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

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**
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**

Table 4 shows the computed values of the six quality attributes for the different versions of the project JaimBot along with POP Count based on the normalization.

Table 4. QMOOD Quality attribute values with POP count for JaimBot project versions

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**

The values listed above for all four versions of Jaimbot indicate that the quality attributes reusability, flexibility, functionality, extendibility, and effectiveness increase from one release to the next and understandability decrease due to increase in complexity in higher versions. The graph below indicates that for higher versions, the reusability, flexibility, functionality, extendibility and effectiveness factors increases and the understandability factors decreases.

The POP count of all four versions of JaimBot also found to be increases.

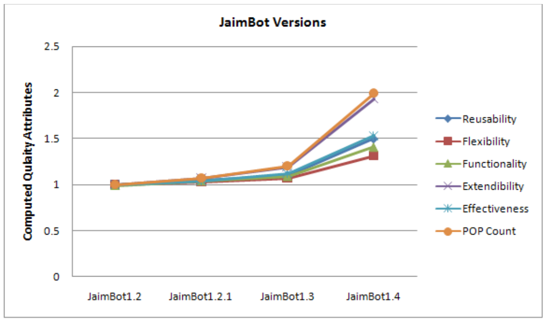

Figure 3. Plot of computed quality attributes and POP Count for JaimBot project versions

4.1.1 Evaluation Results for Jcommon Project Versions

The metric values are collected for eleven metrics of Table 2 for the three versions of Jcommon[10] through the automated tool. The values measured are normalized and presented in Table 5.

Table 5. Actual and normalized metric values for Jcommon projects versions

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**

Table 6 shows the computed values of the six quality attributes for the different versions of the project Jcommon along with POP Count based on the normalization.

Table 6. QMOOD quality attribute values with pop count for Jcommon project versions

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

Version / Quality Attribute	0.8.0	0.9.0	1.0.0
Extendibility	1	1.24	1.26
Effectiveness	1	2.17	2.61
POP COUNT	**1**	**1.16**	**1.43**

The values listed above for all three versions of Jcommon indicate that the quality attributes reusability, flexibility, functionality, extendibility, and effectiveness increase for higher versions and understandability decrease. The graph below also indicates that with higher versions, the reusability, flexibility, functionality, extendibility and effectiveness factors increases however the understandability factors decreases.

The POP count of all three versions of Jcommon also found to be increases.

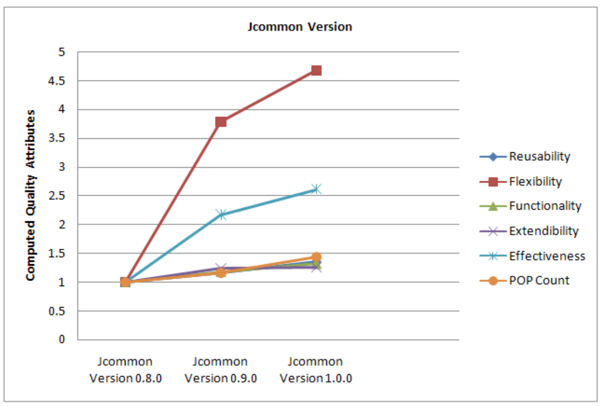

Figure 4. Plot of computed quality attributes and POP Count for Jcommon project versions.

4.2 Evaluation Results for Proguard Project Versions

The metric values are collected for eleven metrics of Table 2 for the three versions of Proguard[11] through the automated tool. The values measured are normalized and presented in Table 6.

Table 7. Actual and normalized metric values for Proguard projects versions

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**

Table 8 shows the computed values of the six quality attributes for the different versions of the project Proguard along with POP Count based on the normalization.

Table 8. QMOOD Quality attribute values with POP count for Proguard project versions

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**

The values listed above for all three versions of Proguard indicate that the quality attributes reusability, flexibility, functionality, extendibility, and effectiveness increase from one release to the next and understandability decrease.

The graph also indicates that with higher versions, the reusability, flexibility, functionality, extendibility and effectiveness factors increases and the understandability factors decreases.

From the results it is clear that as the different versions roll out, the reusability, flexibility, functionality, extendibility and effectiveness factors increases and the understandability factor decreases.

The POP count of the different versions of all three projects also found to be increases.

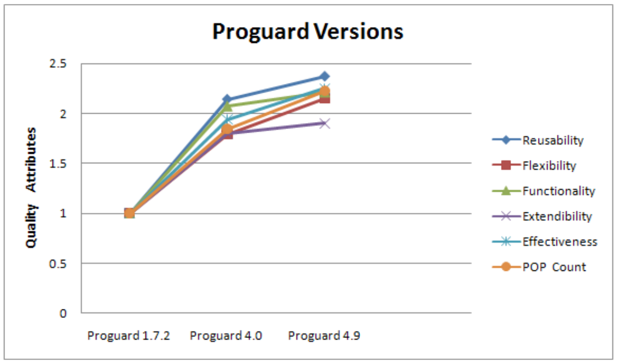

Figure 5. Plot of computed quality attributes and POP Count for Proguard Project Versions.

5. Conclusion and Future Work

Here an Automatic Software Quality Measurement Tool has been made to access the quality of OO software by measuring its Predictive Object Point Metrics. QMOOD quality attributes have been measured for several versions of three java projects through this tool and trend is compared with the trend shown by POP count values for all same versions of these Projects. The results were analyzed in terms of quality.

The conclusion that could be drawn from this study is that the POP metric is a good predictor of software Quality which can be easily seen through the comparisons of results obtained. After seeing the trend observed during study we can ascertain that Reusability, Flexibility, functionality, Extendibility and Effectiveness quality attributes can be estimated directly with the value of POP count. However, Understandability goes indirect in proportionality with the POP count. Hence by comparing the POP count values of projects, comparison in their quality can be estimated. An increase in POP count value reflect corresponding increase in Reusability, Flexibility, functionality, Extendibility and Effectiveness quality attributes and decrease in Understandability.

Research till date for the data studied shows relation between Predictive Object Point Metrics and Quality. But the data studied has been, by no means, exhaustive in coving various software projects. We need to collect additional projects developed for similar requirements and objectives and continue to check this relation. Always there is a need for constant validation to ensure the accuracy of such predictions for the success of software quality assessment through metrics.

Lastly, since this has already been proven that POP metrics set is a also a good predictor of size hence study can be followed up with another through which POP metrics can be mapped to measure software cost and schedule also.

6. References

1. Bansiya J, Davis CG. A hierarchical model for validation of object oriented design quality assessment. IEEE Transactions on Software Engineering. 2002 Jan; 28(1);4–7. Crossref.
2. Kayarvizhy N, Kanmani S. Analysis of quality of object oriented systems using object oriented metrics. Proceedings of 3rd International Conference on Electronics Computer Technology; 2011 Apr. p. 203–6. Crossref.
3. Minkiewicz AF. Measuring object oriented software with predictive object points. PRICE Systems, L.L.C.
4. Shubha J, Vijay Y, Raghuraj S. OO estimation through automation of the predictive object points sizing metric. International Journal Of Computer Engineering and Technology. 2013 May–Jun; 4(3):410–18.
5. Bansiya J. A hierarchical model for quality assessment of object oriented designs. PhD Dissertation, University of Alabama in Huntsville; 1997.
6. Chidamber SR, Kemerer CF. A metrics suite for object-oriented design. IEEE Transactions on Software Engineering. 1994 Jun; 20(6):476–93. Crossref.
7. Hintz M, Montazeri B. Chidamber and Kemerer's metrics suite: A measurement theory perspective. IEEE Transactions on Software Engineering. 1996 Apr; 22(4): 67–271.
8. Li W, Henry S. Object oriented metrics that predict maintainability. Journal of Systems and Software. 1995 Dec; 23(21):929–94.
9. JaimBot [Internet]. Available from: http://sourceforge.net/projects/jaimbot/
10. Jcommon [Internet]. Available from: http://www.jfree.org/jcommon/download
11. Proguard [Internet]. Available from: 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; 1995 Jun.
13. Chawla MK, Chhabra I. Capturing OO software metrics to attain quality attributes – A case study. International Journal of Scientific and Engineering Research. 2013 Jun; 4(6):2229–5518.

14. Objecteering Metrics User Guide [Internet]. [cited 2013 Jun 25]. Available from: http://support.objecteering.com/objecteering6.1/help/us/metrics/toc.htm.
15. User Guide for CCCC.
16. CKJM extended manual. An extended version of Tool for Calculating Chidamber and Kemerer Java Metrics (and many other metrics) [Internet]. [cited 2013 May 3]. Available from: http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/intro.html.