ISSN No. 0976-5697

Volume 3, No. 2, March-April 2012

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Maintainability Measurement in Object Oriented Paradigm

Yajnaseni Dash*
Department of Computer Science & Engineering,
Amity School of Engineering & Technology,
Amity University, Sec-125, Noida, U.P., 201301, India
yajnasenidash@gmail.com

Sanjay Kumar Dubey
Department of Computer Science & Engineering,
Amity School of Engineering & Technology,
Amity University, Sec-125, Noida,U.P., 201301, India
skdubey1@amity.edu

Ajay Rana
Department of Computer Science & Engineering, Amity
School of Engineering & Technology,
Amity University, Sec-125, Noida, U.P., 201301, India
ajay_rana@amity.edu

Abstract: Software quality is an integral aspect of development scheme that determines the degree to which the software in use will meet the expectations of the customer. Maintainability has obtained its significance as an attribute of software quality. However in spite of the importance, there are no definite criteria to measure it. Hence great research interest is required for developing and applying sophisticated techniques to estimate software maintenance effort. As the object-oriented systems use a large number of small methods, a unique maintenance problem is associated with it. The relationship between OO metrics and software maintenance effort is complex and non linear. This paper surveys the different studies regarding software maintainability on object-oriented paradigm which provide further assistance in succeeding researches.

Keywords: maintainability; maintenance; object-oriented; measurement; metrics; software quality

I. INTRODUCTION

Generally it is very difficult to accomplish a good software design because of its error prone nature. The presence of defects in software compromises its quality. According to McCall's [1] there are 11 quality attributes and among which maintainability has its own significance. However, maintainability is the most costly activity in the whole lifecycle of software development and much effort is needed to complete this phase. Design errors during development have a negative impact on maintainability. Hence identification of these blemishes and solving these issues is an essential concern for enhancing the software quality.

Object-oriented (OO) systems are also associated with bugs. So production of good software by large scale legacy system written in OO language is a cumbersome process. Although the syntaxes and the concepts of this language are known, these legacy systems are monolithic, not flexible and difficult to maintain. Either perfect OO system follows the rules and design heuristics or requires proper quantification for controlling the quality. De Marco [2] justified that good design rules cannot be expressed in a quantifiable manner. Assessing maintainability accurately not only helps the developer to improve the design and coding but also increases the performances of the system by reducing the complexity. Hence estimating maintainability with novel techniques leads to enhancement of quality of software system.

II. SOFTWARE MAINTENANCE

A. *Maintenance:*

Maintenance is the act of keeping an entity in an accessible state of repair or validating it to protect from failures. Maintenance process is needed to modify the software product after the delivery procedure for rectification of the existing faults, enhancement of the system performance and adaption of the software to the newer environment. Boehm [3] suggested three steps of maintenance process (Fig. 1).

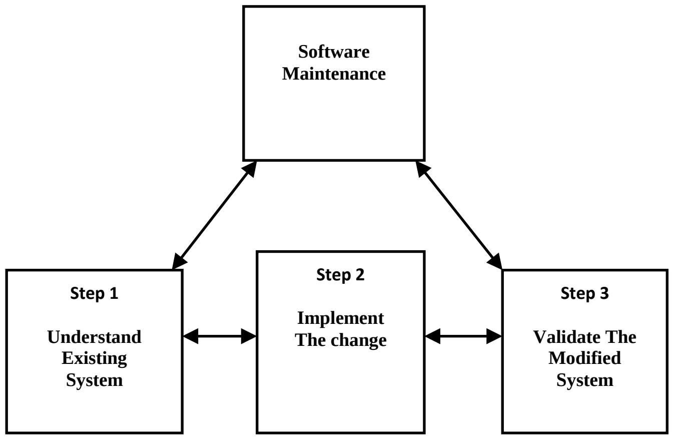
Software Maintenance
Step 1 Understand Existing System
Step 2 Implement The change
Step 3 Validate The Modified System

Figure 1: Steps of Maintenance

B. Maintenance Types:

Most researchers categorize maintenance as adaptive, corrective, perfective and preventive [4-5].

a. Adaptive Maintenance:

The necessity of using this environment-driven maintenance occurs if there are alterations in hardware, operating systems, files, or compilers which has impact on the system.

b. Corrective Maintenance:

This error-driven activity is similar to the debugging process and takes place after the system is placed in operation. During the complete life cycle of a system corrective maintenance is required as software programs are error prone in nature. requirements. The bulk of maintenance behaviors are of this kind.

c. Preventative Maintenance:

It is used to make the software more maintainable by updating the documents. The changes made to modify the software product after delivery for detection and correction of latent faults.

According to the recent studies 90% of the maintenance is carried out in either corrective or perfective ways whereas corrective alone corresponds to 70% of all modifications [6]. Thus corrective maintenance is also called as 'traditional maintenance' and the other forms are referred as 'software evolution'.

III. MAINTAINABILITY

Maintainability is the ease with which the process of maintenance is carried out. It has previously been described in two ways, either informally or as a function of directly measurable attributes. There are many text descriptions available, which are in essence very similar.

According to the IEEE standard glossary of software engineering terminology, maintainability is defined as "the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment" [7].

The other way of defining maintainability is "the capability of the software product to be modified". Modifications may include corrections, improvements or adoptions of the software to changes in environment, and in requirements and functional specifications [8].

A. Maintainability Measurement:

Regardless of the immanency of any exertion for measuring maintainability, immense effort has been exercised to construct formulas for depicting maintainability. Maintainability can be described as a function of instantly computable attributes from A1 to An, such as, M = f (A1, A2, …, An). Informally this approach is somewhat interesting, as it is instinctive that a maintainable system must be simple and reliable. Conversely, there may be complications to measure the attributes, to weigh them opposing to each other and to merge them in a function f. However such an aspiration is relatively inadequate to certain contexts namely kind of system, category of project, programming language, the skill and knowledge of people concerned for drawing conclusions. According to IEEE-1219 [9] explicit measures of software maintenance are analyzability, changeability, stability and testability.

B. Usage of Software Metrics in Measuring Maintainability:

The need of measurement arises not only to reduce cost, effort and schedule but also to amplify the system performance. Certain measures of the maintainability of software can be obtained by the use of available commercial tools. However maintainability can be measured by using software metrics. Software metrics is defined as, "The continuous application of measurement-based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products" [10].

Many researchers have tried to enumerate maintainability by using different types of measures. Chidamber & Kemerer (C&K) [11-13] also explored some of the OO design metrics for computing maintainability effort. These metrics includes WMC, DIT, NOC, CBO, RFC and LCOM. 'Syntactic complexity family of metrics' statically analyzes source code to measure maintenance effort and constitutes Mccabe's cyclomatic complexity (CC) [14] and Halstead Volume (HV) metric [15]. HV in turn is a combined metric based on the distinct number of operators and operands in source code. Effort metrics is commonly used to estimate the maintenance effort by calculating the Mean Time To Change (MTTC) [16].

Among all measurement methods, probably the most appreciable is the Maintainability Index (MI) [17-18]. They have proposed to objectively determine the maintainability of software systems based on the status of the corresponding source code. MI is a composite number, based on several different metrics for a software system, such as the HV metric, the CC metric, the average number of lines of code per module (LOC), and optionally the percentage of comment lines per module (COM). Software maturity index is based on measuring the stability of the product [19]. The metrics used to measure maintainability are described in Table 1.

Table 1: Measuring metrics

Measuring Metrics	***Author***	***Year***	***Reference***	***Description***	***Drawbacks***
Mccabe's cyclomatic complexity number	Mccabe	1976	[14]	Graphically measure the number of independent paths in a program. V(G) = E – N + 2P	It is impossible to specify the total paths. Higher cyclomatic complexity V(G) has a negative impact on changing the system.
Halsted volume	M.H. Halsted	1977	[15]	Used to predict maintenance effort and mean number of faults in programs.	Between the lexical complexities of code, there exist weak logical associations.
Effort metrics	Pressman	1982	[16]	The lower value of Mean Time To Change (MTTC) leads to more maintainable product.	Lack of prediction capability and dependency on skill of maintainer.
Maintainability Index (MI)	Oman et al, and Coleman et al	1992, 1994	[17-18]	Derivation is from source code. Effectively analyzes many systems by making MI comparisons.	Identification of high risk modules provides outstanding insight into the source code. Hence there is no major drawback.
Software Maturity Index (SMI)	IEEE 982.1	1998, 2005	[19]	If SMI becomes 1.0 then the product does not require modifications and stabilized. SMI= $(M_T - (F_a + F_c + F_d))/M_T$	Module measurement is not associated with product stability and difficult interpretation of negative values.

IV. LITERATURE SURVEY OF MAINTAINABILITY AND OO SYSTEM

Rombach [20] reported the results of a controlled experiment for studying the maintainability of the distributed software partially in the object-oriented language LADY, a Language for Distributed systems. He summarized the findings by concluding that the software that was written in object-oriented language is more maintainable than software written in a conventional language. The author also explored that complexity metrics could estimate maintenance and understand the consequent code measures. Some of the measures were applicable as early as the end of architectural design. Johnson and Foote found the use of numerous small methods which influence excellent programming style of object-oriented system [21]. According to Moreau [22-24] conventional metrics are not suitable for OO systems for many causes. Firstly, the hypothesis related to program size and programmer productivity in structured systems need not be applied directly to OO systems. Secondly, the conventional metrics not accomplish the structural features of OO systems. Thirdly, the calculation of complexity of the system which is the same as the summation of the complexity of the components is not proper for OO systems. Moreau [23] also stated that the existing traditional software metrics might be efficient in a specific method within an object. However, there is no empirical evidence present to support these statements.

Moreau [24] again employed the traditional metrics for comparing two implementations of a graphics editor. One of them is a traditional implementation in C and other one is the object-oriented implementation in C++. Mancl and Havanas [25] presented a case study of the impact of the C++ programming language and object oriented design on the maintenance phase of a software development project. The result had shown that while using the object-oriented design, there is variation in terms of improvement in reusability and decrease in complexity of software. This study attempted to assess the differences between object-oriented programming and conventional structured programming. The measurements identified some of the places where object-oriented programming had a significant impact on increasing productivity. They have applied 'the number of lines of code modified per maintenance task' as a measure of the maintenance effort. Inheritance and polymorphism methods not only strengthen the object-oriented systems but also make the process of program understanding and analysis difficult. Wilde and Huitt [26] described several maintenance issues which are unique to object-oriented systems. They have taken instances from a PC smalltalk environment and two real world systems. They also have analyzed difficulties associated to dynamic binding, object dependencies, dispersed program structure, control of polymorphism, high-level understanding and detailed code understanding. Li and Henry [27] have studied the applicability of OO complexity metrics (proposed by C&K) with reference to the maintenance in two commercial systems. They concluded that these metrics in general can be used as predictors of maintenance effort; however, two of the metrics were not as good as expected.

C&K [11-13] experimentally analyzed the OO design metrics for the purpose of evaluating whether or not these metrics are useful for predicting the probability of detecting faulty classes. It is clear that the definitions of these metrics are not language independent. Basili *et al.* [28] as a consequence, had slightly adjusted some of C&K's metrics in order to reflect the specificities of C++ language. Although, the weakness related to this study is that, the authors had used 'the number of lines changed per classes' as a measure of the maintenance effort. Li and Henry [27] also defined some additional metrics. Abreu and Melo [29] also experimentally studied that they have got their MOOD metrics for correlating with the system reliability and maintainability. Harrison *et al.* [30] defined the usefulness of a design metric (the number of test cases) to predict the testing time. Binkley and Schach [31] applied class coupling method to validate the coupling dependency metrics as a predictor of run time failures and maintainability measures by the usage of C++ system (patient core management), 113cls, 82KLOC, file transfer facility, 29 java classes and 6 KLOC measures. Tang *et al.* [32] empirically studied the object-oriented metrics by using C & K metrics, but they have not considered the LCOM and LOC methods as variables.

Muthanna *et al.* [33] investigated the use of software design metrics to statistically estimate the maintainability of large software systems, and to identify error prone modules. The fact that the usage of metrics in the analysis and design of OO software can help designers make better decisions in gaining relevance in software measurement arena. Polo *et al*. [34] reported code metrics for prediction of maintenance of legacy programs as a case study by applying logistic regression, MANTEMA a methodology for maintenance. Moreover, the necessity of having early indicators of external quality attributes depends on maintainability. In addition to this, the aim is to show how early metrics, which measure internal attributes, such as structural complexity and size of UML class diagrams, can be used as early class diagram maintainability indicators. For this purpose, Genero *et al*. [35] presented a controlled experiment and its replication, which was carried out to gather the empirical data. Subramanyam and Krishnan [36] conducted an experimental analysis on subset of C & K metrics suite in determining software defects. Hayes *et al*. [37] proposed a metrics-based software maintenance effort model by the use of validated datasets. They applied COCOMO (constructive cost estimation Model), SLIM, AMEffMo(Adaptive Maintenance Effort Model) and Regression Analysis methods for computing the effort. Kiewkanya *et al*. [38] constructed maintainability model of object oriented design by applying methods like association, aggregation, generalization and classification.

Alain *et al*. [39] proposed prediction of maintainability by the use of regression analysis and DC ratio methods. Bocco *et al*. [40] assessed the capability of internal metrics as early indicators of maintenance effort through experimentation. The usefulness of measures for the analysis and design of object-oriented software is increasingly being recognized in the field of software engineering research. In particular, recognition of the need for early indicators of external quality attributes is increasing. Genero *et al*. [41] investigated through experimentation, whether a collection of UML class diagram measures could be excellent predictors of two main sub distinctiveness of the maintainability of class diagrams namely understandability and modifiability. Results obtained from a controlled experiment supports valuable prediction models for building these features as an early measure. Particularly, the measures were capturing structural complexity through associations and generalizations. Likewise, these measures have been clearly correlated with the subjective perception of the subjects about the complexity of the diagrams. This fact showed that to some extent, objective measures capture the same features as the subjective ones. Breesam [42] empirically formalized a set of metrics with the intention of using them for the quality measurement of an OO design with consideration of class inheritance. Neelamegan and Punithavalli [43] surveyed four OO quality metrics and they found that these metrics focuses on measurements when apply them to the class and design characteristics.

Sastry and Saradhi [44] attempted to implement software metrics with assistance of GUI and also examined relationships of metrics for determination of quality. The quantity of software attributes estimated by observing the life cycle of object oriented software development. Amjan Shaik *et al*. [45] have statistically analyzed the OO software metrics on C&K metric collections by verifying the data put together from the projects of some students. Fast feedback for software designers and managers were provided by applying metrics data. They examined minutely that proper use of these metrics directly reduce the cost of the overall execution and enhancement of quality of the ultimate product. Rizvi and Khan [46] proposed MEMOOD model that caused an improvement of the maintainability or understandability of class diagram and also enhanced the maintainability in final software. Thapaliyal and Garima [47] have empirically analyzed the software defects and object oriented metrics. They evaluated two metrics weighted method per class (WMC) and coupling between object classes (CBO) of C&K metrics suite. They investigated whether the metrics taken are linked to defects or not by the usage of 50 samples of Java classes of different projects. Dubey and Rana [48] used metrics approach to assess utility of object-oriented software to develop successful software applications. Gautam and Kang [49] described that the compound MEMOOD model is better to determine the maintainability of class diagram in terms of their understandability, modifiability, scalability and level of complexity. Malhotra and Jain [50] reviewed software fault prediction for OO system and used logistic regression method. Shaik *et al.* [51] suggested a metric approach for evaluating the system test cases in OO system. They have used regression CR tool architecture method for the estimation process. The detailed survey is summarized in Table 2.

Table 2: Studies on maintainability and object-oriented systems

Sr. No.	***Author***	***Year***	***Reference***	***Description***
1	Rombach	1987	[20]	Controlled experiment on LADY, a Language for Distributed systems to study maintainability. Summarized that the software written in object-oriented language is more maintainable than software written in a traditional language.
2	Johnson and Foote	1988	[21]	Found utilization of ample number of tiny methods persuades outstanding programming style of object-oriented system.
3	Moreau	1987, 1989, 1990	[22] [23] [24]	Unsuitability of traditional metrics for OO systems are due to program size and programmer productivity, conventional metrics and complexity of the system.
4	Mancl and Havanas	1990	[25]	Explained the impact of maintainability in OO design as a case study. Measure of the maintenance effort is computed by the number of lines of code modified per maintenance task. Found improved software reuse and decreased complexity of software differs in object-oriented design.

5	Wilde and Huitt	1992	[26]	Illustrated some unique maintenance issues of OO systems and evaluated the difficulties linked to it.
6	Li and Henry	1993	[27]	Considered the applicability of OO complexity metrics in two commercial systems and summarized that these metrics can predict of maintenance effort.
7	Chidamber and Kemerer	1994	[12]	Explored some of the OO design metrics for computing maintainability effort such as WMC, DIT, NOC, CBO, RFC and LCOM.
8	Basili *et al.*	1996	[28]	Slightly adjusted a few of Chidamber & Kemerer's metrics to imitate the specialness of C++ language.
9	Abreu and Melo	1996	[29]	Studied the MOOD metrics to correlate it with the system reliability and maintainability.
10	Harrison and Samaraweera	1996	[30]	Explained the usefulness of a design metric (the number of test cases) to predict the testing time.
11	Chidamber *et al.*	1998	[13]	Evaluated whether or not the above metrics are useful for predicting the probability of detecting faulty classes
12	Binkley and Schach	1998	[31]	Validated class coupling method and found the coupling dependency metrics as a forecaster of run time failures and maintainability measures of C++ system.
13	Tang *et al*.	1999	[32]	Studied the OO metrics by using C & K metrics but not used the LCOM and LOC methods as variable.
14	Muthanna *et al*.	2000	[33]	Statistically estimated the maintainability of large software systems, and identified error prone modules. OO metrics can assist to gain relevance in software measurement arena.
15	Polo *et al.*	2001	[34]	Used code metrics for prediction of maintenance of legacy programs as a case study.
16	Genero *et al.*	2003	[35]	Obtained a controlled experiment and its replication to assemble the empirical data.
17	Subramanyam and Krishnan	2003	[36]	Performed an empirical analysis on subset of C & K metrics suite to resolve software defects.
18	Hayes *et al*	2004	[37]	Computed maintenance effort by using COCOMO, SLIM, AMEffMo and Regression Analysis methods.
19	Kiewkanya *et al*.	2004	[38]	Constructed maintainability model of object oriented design by applying methods like association, aggregation, generalization and classification.
20	Alain *et al*.	2005	[39]	Predicted maintainability using regression analysis and DC ratio methods.
21	Bocco *et al*.	2005	[40]	Found that internal metrics are early indicators of maintenance effort. Recognized the usefulness of the analysis and design of OO software.
22	Genero *et al*.	2007	[41]	Investigated a set of UML class diagram measures could be excellent predictors of two main sub distinctiveness of the maintainability of class diagrams namely understandability and modifiability.
23	Breesam	2007	[42]	Formalized a set of metrics with the intention of using them for the quality measurement of an object oriented design with consideration of class inheritance.
24	Neelamegan and Punithavalli	2009	[43]	Surveyed four object oriented quality metrics and they found that these metrics focuses on measurements when apply them to the class and design characteristics.
25	Sastry and Saradhi	2010	[44]	Attempted to implement software metrics with assistance of GUI. Examined relationships of metrics for determination of quality.
26	Amjan Shaik *et al*.	2010	[45]	Statistically analyzed the object-oriented software metrics on CK metric collections by verifying data. Examined appropriate use of these metrics to reduce the cost of execution and improvement of quality of the ultimate product.
27	Rizvi and Khan	2010	[46]	Proposed MEMOOD model which improves the maintainability of class diagram and consequently the maintainability of final software.
28	Thapaliyal *et al.*	2010	[47]	Analyzed the software defects and object oriented metrics. Evaluated two metrics WMC and CBO of C and K metrics Suite.
29	Dubey and Rana	2010	[48]	Metrics approach was used to precisely define the qualitative characteristics of the software system.
30	Gautam and Kang	2011	[49]	Observed compound MEMOOD model is better than MEMOOD model to determine the maintainability of class diagram in terms of their understandability, modifiability, scalability and level of complexity.
31	Malhotra and Jain	2011	[50]	Predicted software inaccuracy for object oriented system by applying logistic regression method.
32	Shaik *et al.*	2011	[51]	Evaluated the system test cases in OO system by regression CR tool architecture method. Applied carving and replaying methods to find the degree of differences in unit test cases.

V. CONCLUSION

It is concluded by various researchers that maintainability is a critical concern in the OO design as it uses a large number of small methods. Therefore OO paradigm must be chosen to solve its complexity of measuring the maintainability. This survey is the expected outcome of different researches that are carried out to overcome the issues of maintainability estimation in object-oriented systems. The multitude methods of maintainability summarized in this paper will be a key guide to develop an efficient and cost effective way of maintaining the object-oriented systems leading to the betterment of software quality.

VI. REFERENCE

[1] J.A. Mccall, P.K. Richards, and G.F. Walters, Factors in software quality, Vols I-III, Rome Air Development Centre, Italy, 1977

[2] T. DeMarco Controlling Software Projects; Management, Measurement and Estimation. Yourdan Press, New Jersey, 1982.

[3] B. Boehm, Software Engineering Economics, Englewood Cliffs, NJ:Prentice-Hall, ISBN 0-13-822122-7, 1981.

[4] ISO/IEC 14764:2006 Software Engineering — Software Life Cycle Processes — Maintenance.

[5] E. Burt Swanson, The dimensions of maintenance. Proceedings of the 2nd international conference on Software engineering, San Francisco, 1976, pp. 492 — 497.

[6] S. R. Schach, B. Jin, D.R. Wright, G.Z. Heller, J. Offutt Quality Impacts of Clandestine Common Coupling, Software Quality Journal, 11, 2003, pp. 211-218.

[7] IEEE, IEEE Standard Glossary of Software Engineering Terminology, report IEEE Std 610.121990, IEEE, 1990.

[8] ISO/IEC, Information technology - Software product quality - Part 1: Quality model, report ISO/IEC FDIS 9126-1:2001 (E), ISO, 2001.

[9] IEEE, "IEEE Standard for Software Maintenance", IEEE Std. 1219, The Institute of Electrical and Electronics Engineers, 1998.

[10] P. Goodman, "Practical Implementation of Software Metrics", McGraw Hill, London, 1993.

[11] S. R. Chidamber and C. F. Kemerer ,"Towards a Metrics Suite for Object Oriented design". Proc. Conference on Object-Oriented Programming: Systems, Languages and Applications (OOPSLA'91), Published in SIGPLAN Notices, vol 26 no. 11, 1991, pp.197-211.

[12] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design." IEEE Trans. Software Eng., vol. 20, no. 6, 1994, pp. 476–493.

[13] S. R. Chidamber, D. Darcy and C. F. Kemerer, "Managerial use of Metrics for Object-Oriented Software: An Exploratory Analysis", IEEE Transactions on Software Engineering, vol.24 no.8, 1998, pp. 629-639.

[14] T. J. McCabe, "A complexity measure." IEEE Trans. Software Eng., vol. 2, no. 4, 1976, pp. 308–320.

[15] M. H. Halstead, Elements of Software Science, ser. Operating, and Programming Systems. New York, NY: Elsevier, vol. 7, 1977.

[16] R. S. Pressman, Software engineering: a practitioner's approach, New York: McGraw-Hill, 1982.

[17] D. M. Coleman, D. Ash, B. Lowther, and P. W. Oman, "Using metrics to evaluate software system maintainability." IEEE Computer, vol. 27, no. 8, 1994, pp. 44–49.

[18] P. W. Oman and J. R. Hagemeister, "Construction and testing of polynomials predicting software maintainability." Journal of Systems and Software, vol. 24, no. 3, 1994, pp. 251–266.

[19] (IEEE 982.1-88) IEEE Std 982.1-1988, IEEE Standard Dictionary of Measures to Produce Reliable Software, 1988.

[20] H. D. Rombach, "A controlled experiment on the impact of software structure on maintainability", Software Engineering, IEEE Transactions on, SE-13(3):344–354, March 1987.

[21] R. E. Johnson and B. Foote Designing Reusable Classes. Journal of Object-Oriented Programming. 1988, vol. 1, no. 2, pp. 22-35.

[22] D. R. Moreau, "A Programming Environment Evaluation Methodology for Object-Oriented Systems", Ph.D. Dissertation, University of Southwestern Louisiana, Sep. 1987.

[23] D. R Moreau and W. D. Dominick, "Object-Oriented Graphical Information Systems: Research Plan and Evaluation Metrics," Journal of Systems and Software, vol. 10, 1989, pp. 23-28.

[24] D. R. Moreau and W. D. Dominick, "A programming environment evaluation methodology for object oriented systems: part I - the methodology," Journal of Object-Oriented Programming, vol. 3, May/Jun. 1990, pp. 38-52.

[25] D. Mancl and W. Havanas, "A Study of the Impact of C++ on Software Maintenance", Conference on Software Maintenance, San Diego, CA, USA, IEEE Computer Society Press, 1990.

[26] N. Wilde and R. Huitt, "Maintenance support for object-oriented programs", Software Engineering, IEEE Transactions, vol 18, no. 12, 1992, pp. 1038 – 1044.

[27] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability", Journal of Systems and Software, vol 23, no.2, 1993, pp.111-122.

[28] V. Basili, L. Briand and W. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators", IEEE Transactions on Software Engineering, vol. 22, no.10, 1996, pp. 751-761.

[29] B. F. Abreu, and W.L. Melo "Evaluating the Impact of Object-Oriented Design on Software Quality", Third International Software Metrics Symposium, Berlin, Germany, March 1996.

[30] R. Harrison, L. Samaraweera, M. Dobie, P. Lewis, "An Evaluation of Code Metrics for Object-Oriented Programs," Information and Software Technology, vol 38, 1996, pp.443-450.

[31] Binkley and S. Schach, "Validation of the Coupling Dependency Metric as a risk Predictor", Proceedings in ICSE 98, 1998, pp. 452-455.

[32] M.H. Tang, M.H. Kao, and M.H. Chen, "An Empirical Study on Object Oriented Metrics," Proc. Sixth Int'l Software Metrics Symp., 1999, pp. 242-249.

[33] S. Muthanna, K. Kontogiannis, K. Ponnambalaml and B. Stacey, "A Maintainability Model for Industrial Software

Systems Using Design Level Metrics”, In Working Conference on Reverse Engineering (WCRE’00), 2000.

[34] M. Polo, M. Piattini and F. Ruiz, “Using code metrics to predict maintenance of legacy programs: a case study”, 2001.

[35] M. Genero, M. Piattini, E. Manso, G. Cantone, “Building UML class diagram maintainability prediction models based on early metrics”, Proceedings 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry, , IEEE, 2003, pp. 263-275.

[36] R. Subramanyam and M. S. Krishnan, “Empirical Analysis of CK metricsfor Object Oriented Design Complexity: Implications of Software defects” IEEE transactions on Software Engineering, vol 29, no 4, 2003.

[37] J.H. Hayes, S.C. Patel and L. Zhao, “A Metrics-Based Software Maintenance Effort Model,” Proc. 8th European Conference on Software Maintenance and Reengineering (CSMR'04), 24 – 26 Mar. 2004, IEEE Computer Society, 2004, pp. 254 – 258.

[38] M. Kiewkanya, N. Jindaswat and P. Muenchaisri, “A Methodology for constructing Maintainability Model of Object Oriented Design”, Proc. 4th International Conferences on Quality Software”, IEEE Computer 8-9 Sept, Society, 2004, pp.206-213.

[39] Alain, J. H. Hayes, A. Abran and R. Dumke, “Software Maintenance Maturity Model (SMmm)-The software maintenance process model”, Journal of Software Maintenance and Evolution Research and Practice, vol. 17, no. 3, 2005, pp. 197-223.

[40] M. Bocco, D. Moody and M. Piattini, “Assesing the capability of internal metrics as early indicators of maintenance effort through experimentation”, Journal of system maintenance and evolution, vol.17, 2005, pp. 225-246.

[41] M. Genero, E. Manso, A. Visaggio, G. Canfora and M. Piattini Building measure-based prediction models for UML class diagram maintainability, Published online: Springer Science, 21 March 2007

[42] K. M. Breesam, “Metrics for Object Oriented design focusing on class Inheritance metrics”, 2nd International conference on dependability of computer system IEEE, 2007.

[43] C. Neelamegan and M. Punithavalli, “A Survey- Object Oriented quality metrics”, Global journal of Computer Sc. And Technology, Vol 9, no 4, 2009.

[44] B.R. Sastry and M. V. V. Saradhi “Impact of software metrics on Object Oriented Software Development life cycle”, International Journal of Engineering Science and Technology, Vol 2, no.2, 2010, pp. 67-76.

[45] Amjan Shaik, C. R. K. Reddy, A. Damodaram, “Statistical Analysis for Object Oriented Design Software Security Metrics” International Journal of Engineering Science and Technology Vol. 2, no.5, 2010, pp. 1136-1142.

[46] S.W.A. Rizvi and R.A. Khan, “Maintainability Estimation Model for Object-Oriented Software in Design Phase (MEMOOD)”, Journal of Computing, Volume 2, Issue 4, April 2010.

[47] M. P. Thapaliyal and G. Verma, “Software Defects and Object Oriented Metrics–An Empirical Analysis”, International Journal of Computer Applications Volume 9– No.5, November 2010, pp. 0975 – 8887.

[48] S.K. Dubey and A. Rana, “A comprehensive assessment of object oriented software system using metrics approach”, International journal of computer science and engineering (IJCSE), 2010, pp. 2726-2730.

[49] C. Gautam and S.S. kang, “Comparison and Implementation of Compound MEMOOD MODEL and MEMOOD MODEL”, International journal of computer science and information technologies, 2011, pp. 2394-2398.

[50] R. Malhotra and A. Jain, “Software fault prediction for object oriented System: A Literature Review”, ACM SIGSOFT Software Engineering Notes, 2011.

[51] Shaik, H. Bhadriraju, K. Vikram, N. Shaik and S.V.A. Rao, “A Suggestive evaluation of system test cases in OO system trough carving and replaying differential unit test cases: A Metric Approach”, International journal of Computer Science and Technology, 201, pp. 1345-1353.