IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 8, Issue 4, April 2019

A Consolidated and Comparative Analysis of Software Metrics Tools for Systems Performance Evaluation: A Survey

Er. Ashish Gupta [1], Dr. Rajat Kumar Mehta [2], Er. Mahendra Rai [3]

Ex-Guest Faculty, Department of Computer Sc. & Engineering, MCAET, Ambedkar Nagar, Uttar Pradesh, India[1]

Professor, Department of Irrigation & Drainage Engineering, MCAET, Ambedkar Nagar, Uttar Pradesh, India[2]

Associate Professor, Department of Soil & Water Conservation Engineering, MCAET, Ambedkar Nagar, U.P., India[3]

Abstract: Software metrics refers to a broad range of measurements for computer software. Software metric is a mathematical definition mapping the entities of a software system to numeric measurement values. Measure the software and the software development processes both are very important for organization. As we know when we are calculate software metrics by different-different tools the result will differ due to metrics tools use different methodology. The results are thus tools dependent and are in question for validations. Here an attempt is made to integrate eight different object oriented free metric tools. A study has been done to measure the metrics values using the same set of standard metrics for a software projects. The results have been discussed before and showed the variations in results from different tools for same metrics. Measurements show that, for the same software system and metrics, the metrics values are tools dependent. In this study we have considered comparison of CCCC, CKJM, Dependency Finder, Semmle, VizAnalyzer, JHawk Tool, Analyst 4j and OOMeter Tools.

Keywords: Software Metrics, Software Metric Tools, Measurement, Verification, Maintainability, Coupling and Object Oriented.

I. Introduction

Software metric can be defined as measure that reflects some property of a software product or its specification. Software metric value can also be related to only one unit of a software product. Although metric values could be calculated manually, nowadays software metric tools are being used for calculation of metric values and for their further processing and analysis. Software metrics and software metric tools are wide research areas and improvements in these fields may bring higher success of software projects in general.

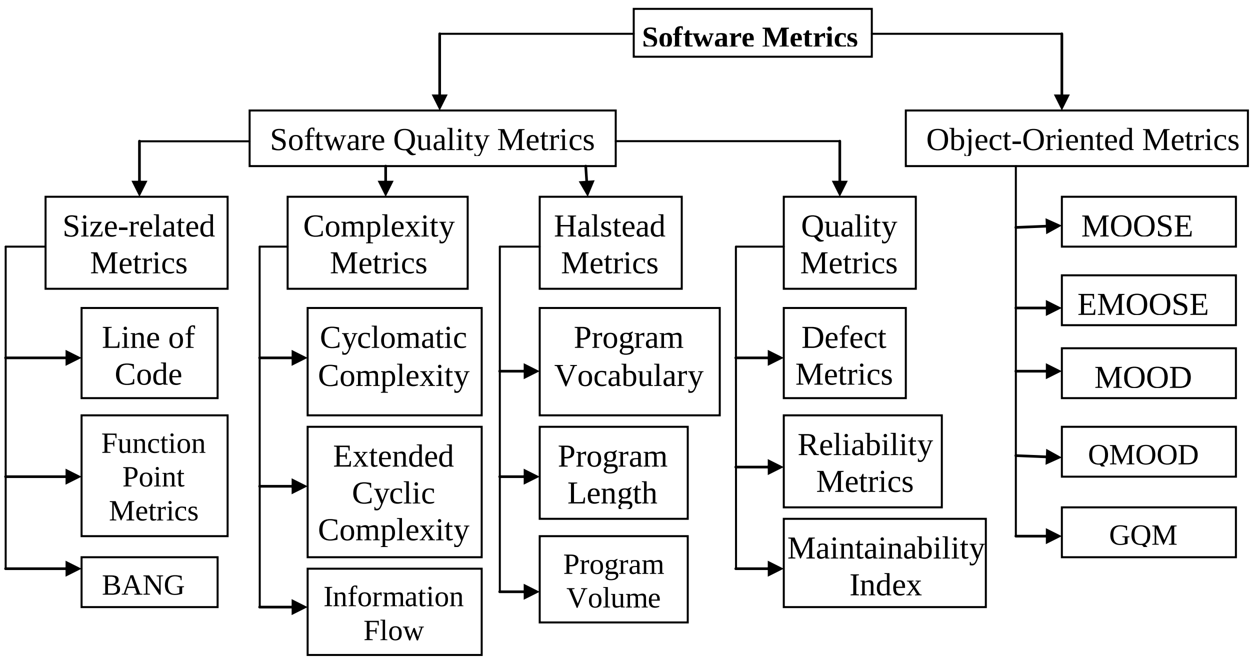

Figure1: classification of the software metrics

Figure 1 shows classification of the software metrics. In object-oriented design, the class is the basic term of concern, not the procedure or statement. Hence, the metrics used to measure such software should be class-oriented. In order to analyze the software which are coded by object oriented language java. We can easly use conventional metrics to measure some parts of characteristics of object-oriented programs (e.g. the number of classes, the number of methods, the number of variables and the complexity of methods). But of course these metrics cannot use to measure the main characteristics for those software which has written in java. A large body of software quality metrics have been developed, and numerous tools exist to collect metrics from program representations. This large variety of tools allows a user to select the tool best suited, e.g., depending on its handling, tool support, or price. This paper will show that different metrics tools show different metrics values for same measurement and same project to overcome with this problem. We came with the integration of metric tools to get the optimized metric value. Option to select those tools whose license type is free.

The remainder of this paper is structured as follows: Section 2 describes various research papers in literature survey. Section 3 describes OBJECT ORIENTED METRICS which led to the questions raised in this paper. It supports the practical relevance of the conducted experiments. Section 4 discusses Software Metrics Tool Selection, some practical issues and sharpens the research questions. Section 5 presents the setup of our experiments including METRIC SELECTION FOR OPTIMIZATION. In Section 6, we discuss threats to the validity of our study. In Section 7, we conclude our findings and finally, in Section 8, we discuss future work.

II. LITERATURE SURVEY

Several maintainability models/methodologies were proposed to help the designers in calculating the maintainability of software so as to develop better and improved software systems. Several groups proposed **metrics-based methods** to measure attributes of software systems which are believed to affect maintenance [1]. Typically, these methods use a set of well-known metrics like *lines of code*, *Halstead volume* [10], or McCabe's Cyclomatic Complexity [3] and combine them into a single value, called *maintainability index*. Bohem [4] presented quality model giving maintainability as one of its important attribute. Berns [5] concluded that maintenance factor depends on the level of difficulty to understand the software program. In 1985 Bowen put forward the equation to find out corrective maintainability. Sneed-Mercy Model [6], Robert Grady Model (at HP) [7], Geoferry and Kemerer Model [8].
Oman et.al [9] demonstrated that how software maintainability analysis can be used to guide software related decision making. Welker K. et.al [10] concluded that MI should not be interpreted in a vacuum rather it should be used as an indicator to direct human investigation. Muthanna et al. [11], developed a maintainability model using polynomial linear regressions. Polo et al. [15] used number of modification requests, mean effort per modification request and type of correction to examine maintainability. M. Dagpinar et .al [12] concluded that size and import direct coupling metrics are significant predictors for measuring maintainability of classes while inheritance, cohesion and indirect/export coupling measures are not. Hayes J.H et.al [13] maintainability model categorized software modules as “easy to maintain” and “not easy to maintain”. The model helps the developers to identify the modules those are not easy to maintain, before integrating them.
et.al [4] Chidamber and Kemerer present theoretical work that builds a suite of metrics for object-oriented design. Further, et.al [1] Chidamber and Kemerer presented the empirical data to demonstrate that these metrics could be used in both C++ and Smalltalk environments. et.al [9] R. Kolewe confirms (based on a field study) that two of the metrics (Class Coupling, and Response for Class) correlate with the defect densities. et.al [2] Abreu and Carapuca, have proposed a suite of six metrics called the MOOD metrics (Metrics for Object-Oriented Design). These metrics include Method Hiding Factor, Attribute Hiding Factor, Method Inheritance Factor, Attribute Inheritance Factor, Polymorphism Factor, and Coupling Factor. et.al [16] Abreu and Melo report that in an experimental study they found these metrics to correlate with the system reliability and maintainability. et.al [9] Chen and Lu, Abott have worked combine on validation of metrics. et.al [7] B.Murgante mentioned that WMC, RFC, CBO, LCOM metrics are good for the good indicators of the functional correctness of OO classes.

III. OBJECT ORIENTED METRICS

The metrics presented here are: method related metrics, class related metrics, inheritance metrics, metrics measuring coupling and metrics measuring general (system) software production characteristics. In this paper fourteen metrics are considered for optimization. These metrics are: DIT (Depth of Inheritance), CBO (Coupling between Objects), LCOM-CK (Lack of Cohesion of Methods) as originally proposed by Chidamber & Kemerer, LCOM-HS, NOC, NOM, RFC, WMC (Weighted Methods per Class), TCC (Tight Class Cohesion), MI (Maintainability Index),UWCS, MPC, Fan Out, Fan In.

IV. SOFTWARE METRICS TOOL SELECTION

With the selection of software metrics tools, we limited ourselves to test systems written in Java (source and byte code) and Eclipse based plug-in. SourceForge.NET provides a large variety of open source software projects and metric tools.

TABLE I
REQUIREMENTS AS A BASIS FOR THE SELECTION OF TOOLS

S.NO.	REQUIREMENT	REQUIREMENTS TYPE TO SUITE
1	Supporting language	Java
2	Measuring metrics	Object Oriented metrics
3	license type	Freely available
4	characteristics	command line tools

The selected tools are listed below:

I. **Analyst4j** is based on the Eclipse platform and available as a stand-alone Rich Client Application or as an Eclipse IDE plug-in. It features search, metrics, analyzing quality, and report generation for Java programs.

II. **CCCC** is an open source command-line tool. It analyzes C++ and Java files and generates reports on various metrics, including Lines of Code and metrics proposed by Chidamber & Kemerer and Henry & Kafura.

III. **Chidamber & Kemerer Java Metrics** is an open source command-line tool. It calculates the C&K object-oriented metrics by processing the byte-code of compiled Java files.

IV. **Dependency Finder** is open source. It is a suite of tools for analyzing compiled Java code. Its core is a dependency analysis application that extracts dependency graphs and mines them for useful information. This application comes as a command line tool, a Swing-based application, a web application, and a set of Ant tasks.

V. **OOMeter** is an experimental software metrics tool developed by Alghamdi et al. It accepts Java/C# source code and UML models in XMI and calculates various metrics.

VI. **Semmle** is an Eclipse plug-in. It provides an SQL like querying language for object-oriented code, which allows searching for bugs, measure code metrics, etc. Understand for Java is a reverse engineering, code exploration and metrics tool for Java source code.

VII. **VizzAnalyzer** is a quality analysis tool. It reads software code and other design specifications as well as documentation and performs a number of quality analyses.

VIII. **JHawk** is a Java based open source framework, it compile Java files and calculate maintainability index and other metrics.

V. METRIC SELECTION FOR OPTIMIZATION

The metrics we selected are basically the "least common denominator", the largest common subset of the metrics assessable by all selected software metrics tools. We created a list of all metrics which can be calculated. Six software metrics have been selected for this study. These metrics work on different program entities, e.g., method, class, package, program, etc. The tools and metrics are shown in Table II. The hash "#" marks that a metrics can be calculated by the corresponding metric tool. It follows a brief description of the metrics finally selected:

A. **CBO** (Coupling between Object classes) is the number of classes to which a class is coupled.

CBO = Number of links/ Number of classes

Numbers of links are number of classes used associations, use links for all the package's classes. A class used several times by another class is only counted once. Numbers of classes are number of classes of the package, by recursively processing sub-packages and classes, for the UML modelling project, this variable represents, therefore, the total number of classes of the UML modelling project.

B. **LCOM-CK** (Lack of Cohesion of Methods) (as originally proposed by Chidamber & Kemerer) describes the lack of cohesion among the methods of a class.

$$LCOM(C) = \begin{cases} P - Q & \text{if } P > Q \\ 0 & \text{otherwise} \end{cases}$$

• P = #pairs of distinct methods in C that do not share variables.
• Q = #pairs of distinct methods in C that share variables.

C. LCOM-HS (Lack of Cohesion of Methods) (as proposed by Henderson-Sellers) describes the lack of cohesion among the methods of a class.

D. NOC (Number of Children) is the number of immediate subclasses subordinated to a class in the class hierarchy.

E. NOM (Number of Methods) is the methods in a class.

F. DIT (Depth of Inheritance Tree) is the maximum inheritance path from the class to the root class.

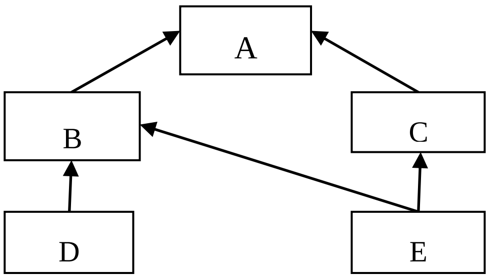

Figure2: Sample measurement of DIT

G. RFC (Response for a Class) is the set of methods that can potentially be executed in response to a message received by an object of the class.

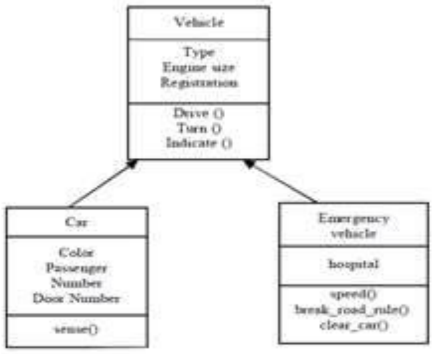

Figure3: Sample Measurement of RFC

H. WMC (Weighted Methods per Class) (using Cyclomatic Complexity as method weight) is the sum of weights for the methods of a class. It is an indicator of how much effort is required to develop and maintain a particular class. A class with a low WMC usually points to greater polymorphism. A class with a high WMC, indicates that the class is complex (application specific) and therefore harder to reuse and maintain. The lower limit for WMC in Refactor IT is default 1 because a class should consist of at least one function and the upper default limit is 50.

I. TCC Tight Class Cohesion metric measures the cohesion between the public methods of a class.

Tight class cohesion TCC = NDP/NP

- NDP–number of pairs of methods directly accessing the same variable.
- NIP–number of pairs of methods directly or indirectly accessing the same variable.
- NP – number of pairs of methods: n (n-1)/2.

J. MI (Maintainability Index**)** is a composite metric that incorporates a number of traditional source code metrics into a single number that indicates relative maintainability. The MI is comprised of weighted Halstead metrics (effort or volume) HV, McCabe's cyclomatic complexity (CC), Lines of codes (LOC). It is used in several automated software metric tools, including the Microsoft Visual Studio 2010 development environment, which uses a shifted scale (0 to 100) derivative. Maintainability Index to calculate MI value of Cumulative Halstead Length, Effort and Volume is to be calculated. Cumulative Halstead Length is the sum of total number of operators and total number of operands present in the given code.

$$N=N1+N2$$

Cumulative Halstead Volume N is Cumulative Halstead Length

$$V = N \times \log n$$

$$MI= 171-5.2\ln (V)-0.23V (g)-16.2\ln (LOC)$$

Where LOC is Line of Code, ln (V) is Halstead Volume and (g) is Cyclomatic Complexity.

Preserving a consistent model that adequately described the interdependencies between the various quality criteria became soon very hard. The reason for this was that our model just like other well-known other models mixed up nodes of two very different kinds: *activities* and *characteristics of the system*. An example for this problem is shown in figure 4 which shows the *maintainability* branch of Boehm's *Software Quality Characteristics Tree*.

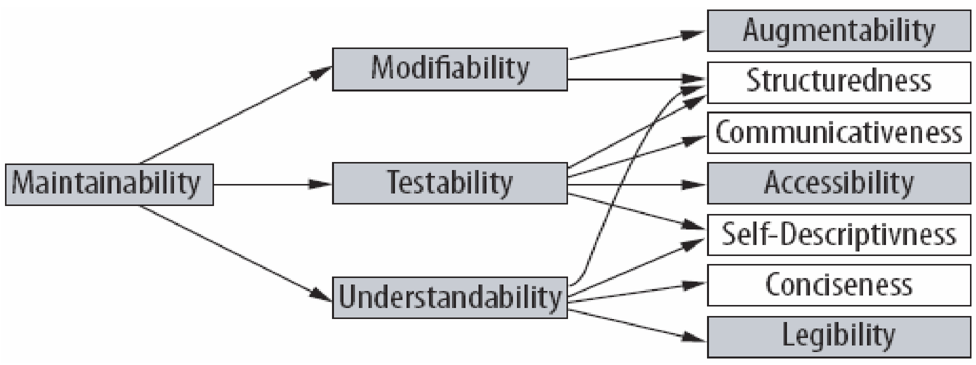

Figure4: Software Quality Characteristics

Though adjectives are used as descriptions the nodes in the gray boxes refer to activities whereas the uncolored nodes describe system characteristics. So the model should rather read as: When we *maintain* a system we need to *modify* it and this activity of *modification* is somehow influenced by the *structuredness* of the system. While this may not look important at first sight we claim that this mixture of activities and characteristics is at the root of most problems encountered with known quality models. The semantics of the edges of the tree is unclear or at least ambiguous because of this mixture. And since the edges do not have a clear meaning they neither indicate a sound explanation for the relation of two nodes nor can they be used to aggregate values!

As the actual maintenance efforts strongly depend on both, the type of system and the kind of maintenance activity it should be obvious that the need to distinguish between activities and characteristics is imperative.

Acts facts matrix: - The separation of activities from facts leads to a two-dimensional model that regards activities and facts as rows and columns of a matrix with explanations for their interrelation as its elements. The separation of activities from facts leads to a two-dimensional model that regards activities and facts as rows and columns of a matrix with explanations for their interrelation as its elements. The selection of activities depends on the particular development and maintenance process of the organization that uses the quality model. Here, we use the IEEE 1219 standard maintenance [12] as an example. An excerpt from its activity breakdown structure is shown in figure 5a. Now, the edges of the activity tree have the clear meaning of task composition. The 2nd dimension of the model, the facts about the situation, are modelled similar to an FCM model but without activity-based nodes like *readability* (5b). Again, the semantics of the edges within this tree is unambiguous though different from the activity tree.

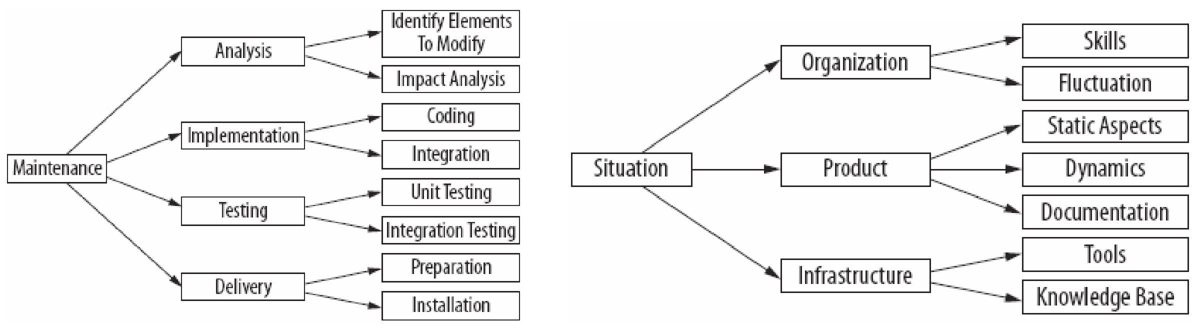

Figure5: Example activity (a) and fact (b) trees

Obviously, the granularity of the facts shown in the example is too coarse for proper evaluation of the facts. In practice, we refine the situation tree stepwise down to detailed, tangible facts that we call *atomic*. Since many important atomic facts are semantic in nature and inherently not computable, we carefully distinguish three fact categories.

Vol. 8, Issue 4, April 2019

1. Computable facts that can be extracted or measured with a tool.
2. Facts that require manual inspection.
3. Facts that can be computed to a limited extent requiring additional manual inspection. One example for this is dead code analysis.

K. **UWCS** (Unweighted Class Size) is calculated from the number of methods plus the number of attributes of a class. Smaller class sizes usually indicate a better designed system reflecting better distributed responsibilities. In other words, all the functionalities were not just stuffed into one big class. It's difficult to set hard and fast rules about this but the classes should be looked at carefully where UWCS is above 100.

L. **MPC** (Message Passing Coupling) measures the numbers of messages passing among objects of the class. A larger number indicates increased coupling between this class and other classes in the system. This makes the classes more dependent on each other which increases the overall complexity of the system and makes the class more difficult to change.

M. **Fan Out** is defined as the number of other classes referenced by a class. Most differences in interpretation hang on the definition of 'references'.

N. **Fan In** is the number of other classes that reference a class. JHawk provides Fan In and Fan Out measures and takes the view that CBO is equivalent to Fan Out.

Efferent Coupling is viewed as equivalent to Fan Out and **Afferent Coupling** to Fan In. Definitions of Afferent and Efferent Coupling tend to be stricter for those for Fan In and Fan Out.

TABLE II: Tools and metrics used in evaluation

Tool	**Metrics**													
Name	**CBO**	**DIT**	**LCOM-CK**	**LCOM-HS**	**NOC**	**NOM**	**RFC**	**TCC**	**WMC**	**MI**	**UWCS**	**MPC**	**Fan In**	**Fan Out**
Analyst4j	#	#	#		#	#	#		#		#	#		
CCCC	#	#		#	#	#		#		#	#			
CKJM	#	#	#		#	#	#		#			#	#	#
Dependency Finder		#		#	#	#		#		#		#		
OOMeter	#	#	#		#			#		#			#	#
Semmle		#	#	#	#	#	#		#		#		#	#
VizzAnalyzer	#	#	#		#	#	#	#	#		#	#		
JHawk				#				#		#			#	#

VI. VALIDITY EVALUATION

We have followed the design and methods recommended by Robert Yin [9]. For supporting the validity, we now discuss possible threats to:

Construct Validity is about establishing correct operational measures for the concepts being studied. To ensure construct validity, we assured that there are no other varying factors than the software metrics tools, which influence the outcome of the study. We selected an appropriate set of metrics and brought only those metrics into relation where we had a high confidence that other experienced software engineers or researchers would come to the same conclusion, given that metrics expressing the same concept might have different names. We assured that we ran the metrics tools on identical source code. Further, we assumed that the limited selection of three software projects of the same programming language posses still enough statistical power to generalize our conclusions. We randomized the test system selection.

Internal Validity is about establishing a causal relationship, whereby certain conditions are shown to lead to certain other conditions, as distinguished from spurious relationships. We believe that there are no threats to internal validity, because we did not try to explain causal relationships, but rather dealt with an exploratory study. The possibility for interfering was limited in our setting. There were no human subjects which could have been influenced, which could have led to different results depending on the time or person of the study. The influence on the provided test systems and the investigated software metrics tools was limited. The variation points like data extraction and analysis allowed only for very small room for changes.

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 8, Issue 4, April 2019

External Validity deals with the problem of knowing if our findings are generalizable beyond the immediate case study. We included the most obvious software metrics tools available on the internet. These should represent a good deal of tools used in practice. We are aware that there is likely a much larger body of tools, and many companies might have developed their own tools. It was necessary to greatly reduce the number of tools and metrics considered in order to obtain results that could allow for reasonable comparisons. Four tools and five metrics applied to three different systems is frankly spoken not very representative for the space of possibilities. Yet, we think the selection and problems uncovered are representative enough to indicate a general problem, which should stimulate additional research including tests of statistical significance. The same holds for the selection of software projects measured. We see no reason why other projects should allow for different conclusions than the three systems we analyzed, and the programming language should have no impact. The selected metrics could include a potential threat. As we have seen in Section 5, some metrics, like NOC, tend to be rather stable over the used tools. We only investigated object-oriented metrics. Other metrics, like the Halstead metrics [10] implemented by some of the tools, might behave differently. Yet, object-oriented metrics are among the most important metrics in use nowadays. The imaginary task and the software quality model used for abstracting the metrics values could be irrelevant in practice. We spent quite some thought on defining our fictive task, and considering the experiences we had, e.g., with Euro control, and the reengineering tasks described by Bar et al in the FAMOOS Handbook of Re-engineering [3], we consider it as quite relevant. The way we applied software quality models is nothing new, it has been described in one or another form in literature.
Reliability assures that the operations of a study, such as the data collection procedures, can be repeated yielding the same results. The reliability of a case study is important. It shall allow a later investigator to come to the same findings and conclusions when following the same procedure. We followed a straight forward design, thus simplicity should support reliability. We documented all important decisions and intermediate results, like the tool selection, the mapping from the tool specific metrics names to our conceptual metrics names, as well as the procedures for the analysis. We minimized our impact on the used artefacts and documented any modifications. We described the design of the experiments including the subsequent selection process.

VII. CONCLUSION

Today a large number of software metrics tools exist. But give different values for the same projects and hence none of them have been validated experimentally for the software metric values they measure. Most tools computed different values for the same metrics on the same projects. From the study it is observed that a new metric tool can be developed which covers metrics values which were emitted before. For more accurate values manual investigation can be done. Since metrics results are strongly dependent on the implementing tools, a validation in terms of manual investigation only supports the applicability of some metrics as implemented by a certain tool. All eight different object oriented metrics measured by them have been optimized by investigating the results manually.

Our final conclusions are that, from a practical point of view, software engineers need to be aware that the metrics results are tool-dependent, and that these differences change the advice the results imply. Especially, metrics based results cannot be compared when using different metrics tools. From a scientific point of view, validations of software metrics turn out to be even more difficult. Since metrics results are strongly dependent on the implementing tools, a validation only supports the applicability of some metrics as implemented by a certain tool. More effort would be needed in specifying the metrics and the measurement process to make the results comparable and generalizable.

VIII. FUTURE SCOPE

Regarding future work, more case studies should repeat our study for additional metrics, e.g., Halstead metrics, and for further programming languages. Moreover, a larger base of software systems should be measured to increase the practical relevance of our results. Additionally, an in-depth study should seek to explain the differences in the measurement results, possibly describing the metrics variants implemented by the different tools. Furthermore, with the insights gained, metrics definition should be revised.

Finally, we or other researchers should revise our experimental hypotheses, which have been stated very narrowly. We expected that all the tools provide the same metrics values and same results for client analyses, so that they can be literally interpreted in such a way that they do not require tests of statistical significance. Restating the hypotheses to require such tests, in order to get a better sense of how bad the numbers for the different tools really are, is additional future work supporting the generalization of our results.

ACKNOWLEDGMENT

The author wishes to thank the reviewers and editors for their valuable suggestions and expert comments that help improve the quality as well as quantity of the paper. We would like to thank **Dr. R.D.S. Yadav**, Ex-Dean, MCAET, Ambedkar Nagar for providing continuous support.

REFERENCES

[1]. Shubha Jain, Viplav Srivastava, Preeti Katiyar, "Integration of Metric Tools for Software Testing" International Journal of Enhanced Research in Science Technology & Engineering, ISSN: 2319-7463 Vol. 3 Issue 5, May-2014, pp: (445-447) .
[2]. Shubha Jain, Preeti Katiyar and Prof. Raghuraj Singh," An integration and optimization of software metric tools", international conference on advance computing and communication technologies -2013.
[3]. Shubha Jain, Preeti Katiyar, Prof. Raghuraj Singh," An integration and optimization of software metric tools", international conference on advance computing and communication technologies -2013.
[4]. Jarallah S. Alghamdi, Raimi A. Rufai, Sohel M. Khan "OOMeter: A Software Quality Assurance Tool" Proceedings of the Ninth European Conference on Software Maintenance and Reengineering (CSMR'05).
[5]. Rüdiger Lincke, Jonas Lundberg and Welf Löwe, "Comparing software metrics tools", software technology group school of mathematics and systems engineering växjö university, Sweden issue, July 20–24, 2008, Seattle, Washington, USA.
[6]. G. V. Bard. Spelling-error tolerant, order-independent pass-phrases via the damerau-levenshtein string-edit distance metric. In ACSW '07: Proc. of the 5th Australasian symposium on ACSW frontiers, pages 117-124, Darlinghurst, Australia, 2007. ACS, Inc.
[7]. R. Lincke. Validation of a Standard and Metric-Based Software Quality Model {Creating the Prerequisites for Experimentation}. Licentiate thesis, växjö university, Sweden, Apr 2007.
[8]. J. Alghamdi, R. Rufai, and S. Khan. Oometer: A software quality assurance tool. Software Maintenance and Reengineering, 2005. CSMR 2005. 9th European Conference on, pages 190-191, 21-23 March 2005.
[9]. 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.
[10]. Hayes, J. Huffman, Mohamed, N., Gao, T. The Observe-Mine-Adopt Model: An agile way to enhance software maintainability. Journal of Software Maintenance and Evolution: Research and Practice, Volume 15, Issue 5, Pages 297 – 323, October 2003.
[11]. C.V. Koten, A.R. Gray, "An application of Bayesian network for predicting object-oriented software maintainability", Information and Software Technology Journal, vol: 48, no: 1, pp 59-67, Jan2006.
[12]. Rizvi S.W.A. and Khan R.A. (2010) "Maintainability Estimation Model for Object-Oriented Software in Design Phase (MEMOOD)", Journal of Computing, Volume 2, Issue 4, April 2010.
[13]. Gautam C, kang S.S (2011), "Comparison and Implementation of Compound MEMOOD MODEL and MEMOOD MODEL", International journal of computer science and information technologies, pp 2394-2398.
[14]. Malhotra et.al," Software Maintainability Prediction using Machine Learning Algorithms." Software Engineering: An International Journal (SEIJ), Vol. 2, No. 2, SEPTEMBER 2012.
[15]. Alisara Hincheeranan and Wanchai Rivepiboon,"A Maintainability Estimation Model and Tool." International Journal of Computer and Communication Engineering, Vol. 1, No. 2, July 2012.
[16]. Dubey et.al."Maintainability Prediction of Object Oriented Software System by Using Artificial Neural Network Approach." International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, Issue-2, May 2012.

BIOGRAPHIES

Er. Ashish Gupta received the **B.Tech** degree in Computer Science and Engineering from Inderprastha Engineering College Ghaziabad, affiliated with GAUTAM BUDDH TECHNICAL UNIVERSITY, Lucknow, India. After qualifying GATE, He completed **M.Tech** in Computer Science and Engineering from Kanpur Institute of Technology, Kanpur, affiliated with UTTAR PRADESH TECHNICAL UNIVERSITY, LUCKNOW, India. His research interest is Mobile Computing and Software Engineering. He worked as an In-charge in department of CS&E at Mahamaya College of Agricultural Engineering and Technology, Ambedkar Nagar.

Dr. Rajat Kumar Mehta is working as a Professor/HOD, Department of Irrigation & Drainage Engineering, MCAET, Ambedkar Nagar, Uttar Pradesh. Currently, he is also the Dean of Mahamaya College of Agricultural Engineering and Technology (MCAET), a constituent college of N.D.University of Agricultural and Technology, Kumarganj, Ayodhya. MCAET was established in 2002 by the elevation of Department of Agricultural Engineering of College of Agriculture. His research interest is Irrigation & Drainage Engineering and Soil & Water Conservation Engineering.

Er. Mahendra Rai is working as an Associate Professor/HOD, Department of Soil & Water Conservation Engineering, MCAET, Ambedkar Nagar, Uttar Pradesh. He was former Dean of Mahamaya College of Agricultural Engineering and Technology (MCAET), a constituent college of N.D.University of Agricultural and Technology, Kumarganj, Ayodhya. MCAET was established in 2002 by the elevation of Department of Agricultural Engineering of College of Agriculture. His research interest is Soil & Water Conservation Engineering and Irrigation & Drainage Engineering.