International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-1 Issue-4, April 2012

Empirical Analysis of Object Oriented Quality Suites

Aman Kumar Sharma, Arvind Kalia, Hardeep Singh

Abstract*— As object oriented paradigm is becoming more pervasive, it becomes necessary that the software engineering methodologies have quantitative measurements for accessing the quality of software at both the architectural and component level. These measures allow the designer to access the software in early stages of the development process and making changes, that will reduce complexity and improving the quality of the product at the development phase. Object Oriented Design metrics is an essential part of software engineering. An empirical study of applying the design measures to asses the software quality is presented. The two metrics suites namely Chidamber and Kemerer (CK) Metrics and Metrics for Object-Oriented Design (MOOD) Metric Suite are used in this study. Using these metrics suite, various design metrics namely Depth of Inheritance Tree (DIT), Coupling Between Object Classes (CBO), Response for a Class (RFC), Number of Children (NOC), Method Inheritance Factor (MIF), Attribute Inheritance Factor (AIF), Method Hiding Factor (MHF), Attribute Hiding Factor (AHF), Polymorphism Factor (POF) for three latest versions of JfreeChart software have been analyzed to predict the software quality. The results of this empirical analysis will help the software developers and academicians in improving the software quality while developing software products using the Object-oriented (OO) approach.

Index Terms*— CK Suite, MOOD Suite, Object-Oriented Software Metrics, Software Quality.

I. INTRODUCTION

Software engineering is commonly misunderstood to be mainly dealing with process oriented activities, i.e. requirements, design, testing, process improvement and project management [1]. Software quality is no more a benefit but has become a necessity because software error can have effects in terms of life, financial loss or time delays. With the ever increasing number of software projects and increasing concern for quality of software systems, practitioners and designers need a quantified and experience-based view on how to make best use of object-oriented mechanisms at the time of development. An empirical analysis on the implementation of object-oriented paradigm on some standard software product can reveal some of the common industry practices about the quality of the software in terms of object oriented software design

metrics. Once designers have such empirical data, they can use it for benchmarking their own application designs and can highlight possible problem areas which are crucial for the software quality. Object-oriented software metrics provide such quantitative view about the implementation of object-oriented constructs in software design for the improvement of the software quality.

The necessary features of the preferred object oriented metrics are recognized as capability of covering all quality and design related features, capability of representing diverse system aspects that are measured, capability of obtaining measurement values for a given system at different time, capability to have an experimental validation and capability of reliable operation [2].

Object Oriented Design (OOD) and development are famous concepts in software development environment. Chidamber & Kemerer (CK) metrics suite [3] is the most widely referenced set of object oriented metrics at class level. The six structural design metrics proposed by CK suite are explained as follows:

1. Weighted Method per Class (WMC): Weighted Methods per Class is the summation of the complexity of each method of the class.

2. Depth of Inheritance Tree (DIT): Depth of Inheritance Tree is the maximum length of the path from the class to the root of the hierarchical tree.

3. Number Of children (NOC). Number Of Children is the number of immediate subclasses of a class.

4. Coupling Between Objects (CBO): CBO for a class is count of the number of other classes to which it is coupled. Two classes are coupled together if methods of one use methods or instance variables of other. Excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application.

5. Response For a Class (RFC): This is a set of methods that can potentially be executed in response to a message received by an object of that class. Since it specifically includes methods called from outside the class, it is also a measure of the potential communication between the class and other classes.

6. Lack of COhesion in Method (LCOM): Lack of cohesion in methods is a measure of how poorly the methods and the variables are related in a class. The larger the number of similar

Manuscript published on 30 April 2012.
***** Correspondence Author (s)
Aman Kumar Sharma, Department of Computer Science, Himachal Pradesh University/, Shimla, India, (e-mail: sharmaas1@gmail.com)
Arvind Kalia, Department of Computer Science, Himachal Pradesh University, Shimla, India, (e-mail: arvkalia@gmail.com).
Hardeep Singh, Computer Science and Engineering Department, Guru Nanak Dev University, Amritsar, India, (e-mail: hardeep_gndu@rediffmail.com).

methods, the more cohesive the class, which is consistent with traditional notions of cohesion that measure the inter-relatedness between portions of a program. However, studies [4] [5] have concluded that LCOM is insignificant in evaluation of quality of a software product.

Abreu's MOOD metrics [6] is a system level metrics set which quantifies the use of four main mechanisms of object-oriented design that is encapsulation, inheritance, polymorphism and message-passing. The metrics used by MOOD are namely MIF, AIF, MHF, AHF, POF and CF. The metrics related to encapsulation are MHF and AHF. Information hiding is a way of designing routines such that only a subset of the module's properties is known to users of the product. Information hiding leads to higher encapsulation [7]. Information hiding is a theoretical technique that indisputably proven its value in practice. Software products with smaller values of MHF and AHF have been found to be easier to modify, than programs that don't follow encapsulation [8] [9].

1. Attribute Hiding Factor (AHF): It measures the invisibilities of attributes in classes. The invisibility of an attribute is the percentage of the total classes from which the attribute is not visible. An attribute is called visible if it can be accessed by another class or object. Attributes should be "hidden" within a class by being declared as private. The numerator is the sum of the invisibilities of all attributes defined in all classes. The denominator is the total number of attributes defined in the project [10]. It is desirable for the Attribute Hiding Factor to have a large value.

2. Method Hiding Factor (MHF): The Method Hiding Factor measures the invisibilities of methods in classes. The invisibility of a method is the percentage of the total classes from which the method is not visible. The Method Hiding Factor is a fraction where the numerator is the sum of the invisibilities of all methods defined in all classes. The denominator is the total number of methods defined in the project. Methods should be encapsulated (hidden) within a class and not available for use to other objects. Method hiding increases reusability in other applications decreases complexity and also reduces modifications to the code [10].

3. Method Inheritance Factor (MIF) & Attribute Inheritance Factor (AIF): Inheritance decreases complexity by reducing the number of operations and operators, but abstraction of objects can make maintenance and design difficult. MIF and AIF measure directly the number of inherited methods and attributes respectively as a proportion of the total number of methods/attributes. It is intuitively clear that there is a relationship between the relative amount of inheritance in a system and the number of methods/attributes which have been inherited [11].

4. POlymorphism Factor (POF): This metric is defined as the ratio of the methods inherited from base classes but overridden to all methods taken from base classes not necessarily overridden or inherited.

5. Coupling Factor (CF): The CF is defined as the ratio of number of non-inheritance couplings to the maximum number of couplings in a system. The maximum number of couplings includes both inheritance and non-inheritance related coupling. Inheritance-based couplings arise as derived classes (subclasses) inherit methods and attributes form its base class (superclass). The metric is related to coupling and is not included in this study as another metric CBO pertaining to coupling is applied.

Using the results of this study, practitioners and designers can identify potential problem areas and can improve their software during the development of the software. This paper is organized as follows: brief reviews of some of the existing literature are presented in Section 2. In Section 3, description about the objective of this study is given. The research methodology opted for this research is discussed in the section 4. In the Section 5 results and empirical analysis of object-oriented software metrics used in this study are discussed and finally in section 6 conclusions and future scope are summed up.

II. REVIEW OF LITERATURE

A majority of researchers have concentrated on theoretical and empirical validation of object-oriented software metrics. Chidamber and Kemerer metrics have been empirically validated in many studies.

Erik Arisholm et al. defines coupling and its measurement based on dynamic analysis of systems. An empirical evaluation of the proposed dynamic coupling measures was conducted in which the author studied the relationship of these measures with the change proneness of classes. The static analysis results suggested that some dynamic coupling measures are significant indicators of change and hence the quality of the software [6].

Khaled El Emam et al. performed a validation study on a telecommunication system developed in C++ language. The results of the study indicated that from among the many metrics covering coupling, cohesion, inheritance and complexity only four are related to fault proneness after controlling the class size, and only two out of these are useful predictors: CBO and Ancestor class coupling Class Method Import Coupling (ACMIC). It was also suggested that perhaps many of the contemporary object-oriented metrics may not be associated with fault-proneness and good prediction accuracy may be attained by careful selection of a small number of metrics [13].

Harrison et al. indicated that the systems without inheritance were easier to modify than the corresponding systems containing three or four levels of inheritance. Also, it was easier to understand the system without inheritance than a corresponding version containing three levels of inheritance [14].

V.R. Basili et al. empirically investigated the suite of OOD metrics introduced by Chidamber and Kemerer. The goal of the study was to assess the CK suite metrics as predictors of fault-prone classes and therefore determine whether they can be used as early quality indicators. As a result of the study it appeared that several of CK suite OO metrics can be useful to predict class fault-proneness during the early phases of the software development [8].

F. Brito Abreu et al. in their study experimentally evaluated the impact of OOD on software quality characteristics. A suite for the OO design namely MOOD was adopted to measure the use of OOD mechanisms. The study showed that OOD mechanisms such as inheritance, polymorphism, information hiding and coupling can influence quality characteristics

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

like reliability or maintainability [2].

L.C. Briand et al. in their study showed that many coupling and inheritance measures are strongly related to the probability of fault detection in a class. In their view coupling induced by method invocations, the rate of change in a class due to specialization and the depth of a class in its inheritance hierarchy appear to be important quality factors [10].

J. Bansiya et al. discusses a hierarchical model for the assessment of high-level design quality attributes in object-oriented designs. The model provides the evaluation of structural and behavioural design properties of classes and objects and their relationship using a suite of object-oriented design metrics. The design properties such as encapsulation, modularity, coupling, and cohesion are related to high-level quality attributes such as reusability, flexibility and complexity were empirically validated. The relationship between the design properties and quality attributes were weighted in accordance with their influence and importance which can provide an insight to the practical quality assessment in the OOD [7].

Ramanath Subramanyam et al. discusses that the design metrics play an important role in helping developers understand design aspects of software hence improve software quality during development. The study provides empirical evidence supporting the role of OOD complexity metrics in determining software defects. They believed that these results have significant implications for designing high-quality software products using the OO approach [21].

Daly et al. [12] investigate the effects of DIT on the maintainability of object oriented software, by means of formal experiments on student populations. The results suggest that software systems with three levels of inheritance require less time for maintenance, with respect to systems with no inheritance. However, systems with five levels of inheritance require more effort for the same purpose.

Abreu's MOOD metrics have been validated but not much thoroughly empirically validated. Abreu et al. have studied relationship of the metrics with defect density and rework [3].

Abreu et al. [4] have derived some design heuristics for MOOD metrics using a collection of C++ libraries.

Abreu et al. in their paper [5] analyzed the metrics for some of the Eiffel libraries and derived some design heuristics. This study evaluates these metrics for very large set of Java standard libraries and projects and presents some design heuristics.

Harrison et al. [15] also provided a theoretical validation of MOOD metrics set.

III. Objectives of the Study

The main objective in this study is to empirically validate the object oriented design metrics namely CK suite metrics and MOOD metrics. The results from this empirical investigation may provide an insight into the relationship between the object oriented design metrics and the software quality.

IV. Scope of the Study

Java is a common object-oriented programming language, a metrics based measurement and analysis of Java based software product can reveal particular design trends that can be considered as guidelines for other object oriented based user applications. Thereby, in this study the selected component is written in java, an object oriented language. Jfreechart, an open source free library for creating various kinds of charts and graphs, is used. Three versions of JfreeChart are used for measurement and analyses are: JfreeChart 1.0.13, JfreeChart 1.0.12 and JfreeChart 1.0.11.

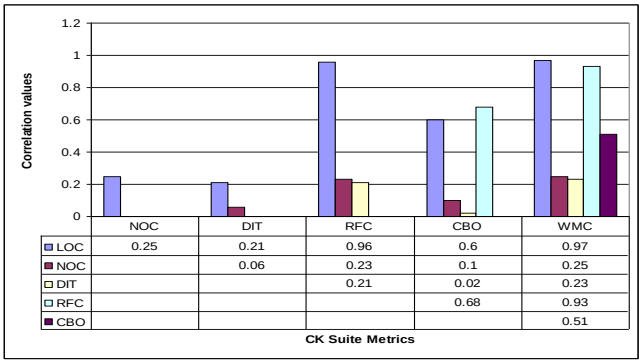
V. Research Methodology

As discussed in Section 4, the research object for the empirical study is JfreeChart with three latest versions. A software metrics tool Chidamber Kemerer Java Metrics (CKJM) software tool was used on the research object to compute CK metrics namely WMC, NOC, DIT, RFC and CBO. The empirical values for these metrics were obtained in a text file. For computing the MOOD suite metrics namely AHF, MHF, AIF, MIF and POF the Essential Metrics software tool was used. The output of the essential metrics was stored in a comma separated value (.csv) file format. These tools have been used to evaluate the CK suite metrics as well as MOOD suite metrics for the different versions of JfreeChart components.

The data collected as result of experiment supplied data from which useful information was to be extracted. Correlation Statistical technique was used to analyse the data, for which software tool, PSPP (Public Social Private Partnership), a free Open Source Software (OSS) was used. It produces tabular output in ASCII, Hyper Text Markup Language, or PostScript format. The study used bivariate correlation. The correlation values range between -1 to +1. A negative correlation value implies that the metrics between which the correlation is evaluated are negatively related to each other. A zero correlation denotes that the metrics are not related to each other. Correlation value of less than -0.5 or more than + 0.5 indicates a strong correlation between the metrics either negatively or positively.

VI. Analysis of Results

The CKJM tool computes the values of WMC, NOC, DIT, RFC and CBO for each class. The raw data gathered during the process of experiment is huge as the number of classes in each version of the components runs into hundreds. The correlation results for the component JfreeChart for the three evaluated versions are represented below:



	NOC	DIT	RFC	CBO	WMC
LOC	0.25	0.21	0.96	0.6	0.97
NOC		0.06	0.23	0.1	0.25
DIT			0.21	0.02	0.23
RFC				0.68	0.93
CBO					0.51

Fig. 1: Correlation results of CK suite Metrics for JfreeChart 1.0.11

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

Fig. 1 shows the correlation values between different CK metrics such as the value of 0.25 depicts the correlation between NOC and LOC. The same analogy is followed for the remaining values in Fig. 1, Fig. 2, Fig. 3 and Fig. 4. The results have shown a similar trend across all the three versions of JfreeChart as depicted in the Fig. 1, Fig. 2 and Fig. 3. It is evident from the result that WMC correlates more strongly with RFC and CBO. DIT had minor to moderate correlation with RFC and WMC. Moreover RFC metric is highly correlated with WMC. The NOC in this study had either trivial or minor correlation with RFC and WMC for all the three versions of the software. It was further observed that except NOC and DIT other design metrics significantly correlated to the LOC. The low correlation between the LOC and (DIT and NOC) are because of the reason that there may be small classes with few lines of code inside them but are designed in such a way that a long hierarchy of the inheritance exists which gives high values of

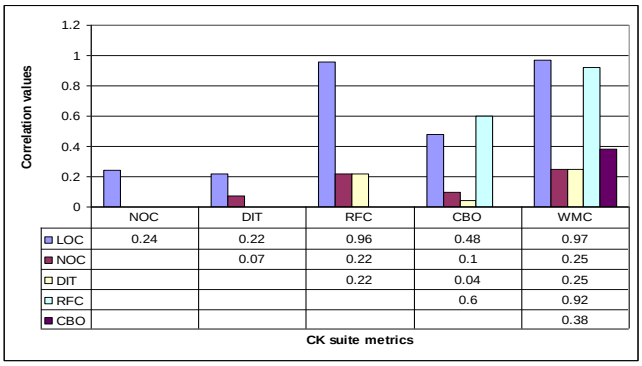


Fig. 2 Correlation results of CK suite Metrics for JfreeChart 1.0.12

DIT and NOC and vice versa. It is evident from the results except NOC and DIT the other three metrics WMC, RFC and CBO have more or less notable correlations with each other. The high correlation value between RFC and WMC can be explained in the manner that with increased functionality provided by more methods i.e. with the increase in the WMC value, there is potential of increased method calls, thus increased RFC. No statistically significant relationships were found between DIT and NOC. Moderate correlation values between WMC and DIT is due to the fact that classes with larger number of methods are likely to be more application specific limiting the possibility of reuse.

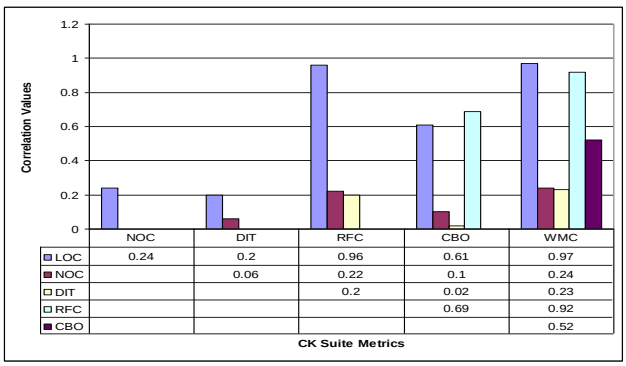


Fig. 3 Correlation results of CK suite Metrics for JfreeChart 1.0.13

The above results also suggests that although WMC, RFC, and CBO measure different aspects of class design, there is a significant statistical reason to believe that classes with high WMC also have high values of RFC and CBO and vice versa.

The results generated for MOOD metrics are presented in Fig. 4. It is clear from the correlation results that there is no correlation between the MIF and AIF it is because of the reason that accessing attributes can be accomplished by two means, firstly, if attributes are not hidden, access them directly without concerning MIF value. Secondly, if attributes hidden, access them using unhidden methods. In case the attribute is to be made inaccessible then attributes and as well as methods both are made hidden. Hence, there is no relation between MIF and AIF. The above analogy can be applied on AHF and MHF to conclude a weak or negligible correlation between them.

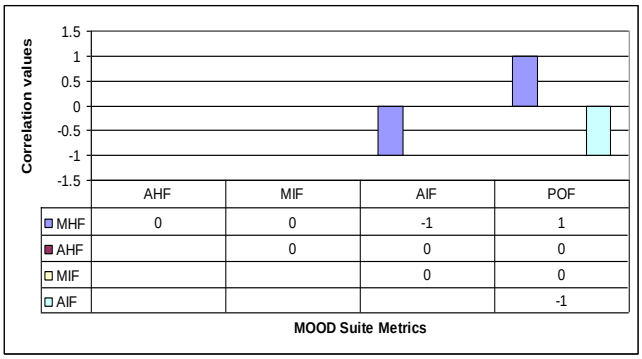


Fig. 4 Correlation results of MOOD suite metrics for all versions of JfreeChart

It is evident from the results that AIF and MHF have a negative strong correlation. It may be due to addition of methods with public access modifier to the code for each visible attribute. Thus, making the software product more reusable but indicates insufficiently abstracted implementation and a large proportion of unprotected methods means the product is more prone to errors but is flexible as it may be inherited in any way.

The MOOD suite measures POF due to method overriding and not by method overloading or polymorphism due to interface implementation. The results indicate a positive relation between the POF and MHF. It is due to a large number of methods are made visible (MHF has low value) but only a few of such methods are overridden in sub classes (POF is restricted to a low value). We can intuitively say that polymorphism (overrides) was kept low to make the code understandable as more excessive polymorphic code may be too complex to understand.

Further the results showed a strong negative correlation between the POF and AIF. The results are on the same analogy as the relation between POF and MHF. It appears that a considerable amount of attributes are visible (AIF has high value) nevertheless hardly any such attribute or method of such attribute is applied for POF (POF has minimum value). For the component it may concluded on the basis of the relation between POF and (MHF & AIF) that methods and as well as attributes are available for inheritance but these are not used. It implies that the versions of JfreeChart considered in this study are flexible, extendable and usable but are error prone and lack security. Either one of them is not enough; especially AIF individually may not control POF value.

VII. CONCLUSION AND FUTURE SCOPE

This study is focused on the empirical investigation of the object oriented design metrics for a component showed that the results across the different versions of JfreeChart have similar trend with respect to correlation among CK metrics. The analysis of the software metrics is done by providing the empirical evidence of such metrics through case studies. This study provides the results of three versions of JfreeChart for the CK suite metrics and MOOD metrics suite. This empirical analysis provides the practitioner with some empirical evidence demonstrating that most of these metrics can be useful quality indicators. The CK metrics are correlated among themselves. WMC – RFC, CBO – RFC, WMC – CBO are highly correlated. Similarly, the MOOD metrics are correlated AIF – MHF and POF – AIF and negatively very strongly correlated, whereas POF – MHF is very strongly positively correlated. In future, validation of CK and MOOD suites of object oriented design metrics across different OO languages can be performed to further strengthen the opinion of this study.

REFERENCES

[1] Abreu F B and Carapuca R, "Object-Oriented Software Engineering: Measuring and Controlling the Development Process", Proceedings of the 4th International Conference on Software Quality, ASQC, McLean, VA, USA, 1994.
[2] Abreu F B and Melo W, "Evaluating the Impact of Object- Oriented Design on Software Quality", Proceeding of Third International Software Metrics Symposium, pp. 90-99, 1996.
[3] Abreu F B and Melo Walcelio, "Evaluating the Impact of Object-Oriented Design on Software Quality", METRICS'96: Proceedings of the 3rd International Symposium on Software Metrics: From Measurement to Empirical Results, IEEE Computer Society Washington, DC, USA, 1996.
[4] Abreu F B, Coulao Miguel and Esteves Rita, "Toward the Design Quality Evaluation of Object-Oriented Software Systems", Proceedings of the 5th International Conference on Software Quality, Austin, Texas, USA, 1995.
[5] Abreu F B, Esteves R and Goulao M, "The Design of Eiffel Programs: Quantitative Evaluation Using the Mood Metrics", Proceedings of TOOLS'96, Santa Barbara, CA. USA, 1996.
[6] Arisholm Erik, Briand Lionel C and Foyen Audun, "Dynamic Coupling Measurement for Object-Oriented Software", IEEE Transaction Software Engineering, 30(8), pp. 491–506, 2004.
[7] Bansiya J and Davis C, "A Hierarchical Model for Object-Oriented Design Quality Assessment", IEEE Transaction Software Engineering, 28(1), pp. 4-17, 2002.
[8] Basili V R, Briand L C and Melo W L, "A Validation of Object-Oriented Design Metrics as Quality Indicators", IEEE Transactions on Software Engineering, 22, pp. 751–761, 1996.
[9] Boehm B W, "Improving Software Productivity", IEEE Computer, pp. 43-57, 1987.
[10] Briand L C, Wust J, Daly J W and Porter D V, "Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems", Journal of Systems and Software, 51, pp. 245–273, 2000.
[11] Chidamber S R and Kemerer C F, "A Metrics Suite for Object Oriented Design", IEEE Transaction Software Engineering, 20(6), pp. 476–493, 1994.
[12] Daly J, Brooks A, Miller J, Roper M and Wood M, "Evaluating Inheritance Depth on the Maintainability of Object-Oriented Software", Empirical Software Engineering 1(2): 109– 132, 1996.
[13] El-Emam Khaled, Benlarbi S, Goel N and Rai S, "A Validation of Object-Oriented Metrics", Technical Report, NRC/ERB 1063, National Research Council Canada, 1999.
[14] Harrison R, Counsell S and Nithi R, "Experimental Assessment of the Effect of Inheritance on the Maintainability of Object-Oriented Systems", Journal of Systems and Software, 52(2-3), pp. 173–179, 2000.
[15] Harrison R, Counsell Steve J and Nithi Reuben V, "An Evaluation of the Mood Set of Object-Oriented Software Metrics", IEEE Transaction Software Engineering, 24(6), pp. 491–496, 1998.
[16] Hitz M and Montazeri B, "Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective", IEEE Transactions On Software Engineering, 22(4), pp. 267-271, 1996.
[17] Jacobson I, Christerson M, Jonsson P and Overgaard G, "Object-Oriented Software Engineering: A Use Case Driven Approach", Wokingham, England, Addison-Wesley, 1992.
[18] Khan R A, Mustafa K and Ahson S I, "An Empirical Validation of Object Oriented Design quality Metrics, Journal of KSU, KSA, 19, 1427H, pp. 1-16.
[19] Korson T D and Vaishnavi V K, "An Empirical Study of Modularity on Program Modifiability", Empirical Studies of Programmers, pp. 168-86, 1986.
[20] Meyer B, "Software Engineering in the Academy", Computer IEEE, 34(5), pp. 28-35, 2001.
[21] Subramanyam R and Krishnan M S, "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects", IEEE Transactions on Software Engineering, 29(4), pp. 297-310, 2003.
[22] Gill Nasib Singh, "Software Engineering", Khanna Book Publishing Co. Ltd., New Delhi, 2000, 1st ed.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)