Futuristic Trends in Object Oriented Languages and Metrics

[1] Mahesh Kumar,[2] Dr.Jeeetendra Sheethlani

[1]Department of Computer Science
SSSUTMS, Sehore

Abstract— In this paper we provide an detailed overview of existing research in the _eld ofsoftware restructuring and refactoring, from a formal as well as a practical point of view. Next, we propose an extensive list of open questions that indicate future research directions, and we provide some partial answers to these questions. An intrinsic property of software in a real-world environment is its need to evolve. As the software is enhanced, modi_ed and adapted to new requirements, the code becomes more and more complex and drifts away from its original design. Because of this, the major part of the total software development cost is devoted to software maintenance [26,49,61]. Better software development methods and tools do not solve this problem, because their increased capacity is used to implement more new requirements within the same time frame [44], making the software more complex again. To cope with this spiral of complexity there is an urgent need for techniques that reduce software complexity by incrementally improving the internal software structure.

IndexTerms —Back Propagation Neural Network, Form Processing, Histogram of Gradients, Kannada Script, Principal Component Analysis.

I. INTRODUCTION

Several researchers have proposed a variety of criteria [1-11] for evaluation and validation to which a proposed software metric should adhere. Amongst them, we can mention validation through measurement theory [2, 4, 6], IEEE standards [5] Kaner's framework [3], and Weyuker's properties [1]. However, most of the existing evaluation and validation criteria were proposed when procedural languages were dominant. After the adaptation of OO languages by the software industry, not too much effort has been made to develop a model/ framework for evaluating software complexity measures in the OO domain. There are some proposals for OO languages [2,3,4]; however, most of them cover only specific features of evaluation. For example, Z use's properties [5] for OO metrics are mathematical in nature and based on principles of measurement theory. The lack of proper guidelines for evaluation and validation of OO metrics motivate us to develop a new evaluation criterion which includes all the features required for evaluation of the OO metrics. For achieving this goal, first we have analyzed the available validation and evaluation criteria, extracted their important features, suggested additions/modifications (if required), then presented them in a formal way. The validite of the proposed model is evaluated by applying eleven different well-known OO metrics. OO metrics are measurement tools to achieve quality in software process and product. However, in general, software measurement has not yet achieved the needed degree of maturity [9] and it needs standardization [16]. Existing proposals, such as Weyuker's properties [1] and the application of measurement theory in software engineering [2, 4, 6, 7, 8], are a topic of discussion [10,1]. We have also worked in the related area of software measurement and presented several papers. We have presented a paper on the usefulness of Weyuker's properties for procedural languages [24]. In another work, we have analysed how Weyuker's properties are used by the developers of three OO metrics [5]. We have previously performed experimentations to analyze the current situation of standard measurement activities in small and medium software companies [26]. We have also performed a study on the situation of the empirical validation of software complexity measures in practice, and we accordingly proposed a model [7]. The applicability of measurement theory on software complexity measures is also investigated in one of our previous works [2]. In the present paper we analyze the present practices used for evaluation and validation of OO metrics, and we accordingly present a model for evaluating OO metrics. We also propose a framework for evaluating software complexity measures but, the present paper is specifically for OO metrics, since OO languages do not share the same features with procedural languages.

II. LITERATURE SURVEY

Most of the software maintainability assessment model have been proposed and compared with other molds. Zhuo F. et al. (1993) proposed maintainability index (MI) that determine the maintainability of software system based uponthe status of the source code, which show high correlation between assessments automated model and some expertevaluation [28]. Binkley A. et al. (1998) collect the data ofmaintenance for the development of project written in anylanguage like C, C++, COBOL etc and produce a level of interaction between modules, which show low coupling were subjected for fewer maintenance effort and fewer maintenancefault and failures [9]. Muthana S. et. al. (2000) proposed that the linear prediction model which is being evaluated by some industrial software system to estimate the maintainability of larg system and to identified some fault prone models to defineimpact rate, effort and error rate [30]. Kiewkanya M. et al. (2004) prescribed that object-oriented (OO) is ease ofmaintenance to provide better understandability and modifiability. It describes three technique discriminant techniques (correlation between maintainability and structural complexity), weighted score level technique (combination of understanding and modifiability) and weighted predicate level

technology (combination of predicate understandability and modifiability). Rizvi S.W.A. et al. (2010) propose a MEMOOD model, which provide an opportunity to improve the maintainability or understandability of class diagram and consequently the maintainability in final software [31]. Gautam C. et al. (2011), describe that the compound MEMOOD model is better the MEMOOD model to determine the maintainability of class diagram in terms of their understandability, modifiability, scalability and level of complexity [32]. Abreu et al. [33] provides a new classification framework for the TAPROOT. This framework was defined with the other two independent vectors these are category and granularity. Six categories of Object-Oriented metrics were defined are design metrics, complexity metrics, size metrics, quality metrics, productivity metrics and reuse metrics and also proposed three Levels of granularity are software, class and methods but no empirical/theoretical base for the metrics was provided. M.Alshayeb et al. [34] have given two iterative procedures for the pragmatic study of OO metrics. They include the short-cycled agile process and the long cycled framework evolution process. By bserving the results, it can be seen that the design efforts and source lines of code added, changed, and deleted were triumphantly predicted by objectoriented metrics in short-cycled agile process where as in the case long-cycled framework process the same features were not successfully predicted by it. This has shown that the design and implementation changes during development iterations can be predicted by object-oriented Metrics,

III. GOAL QUESTION METRICS (GQM):

V. L. Basili [43] developed GQM approach. This approach was originally defined for evaluating defects for a set of projects in the NASA Goddard Space Flight Center environment. He has also provided the set of sequence which are helpful for the designers. The goal of GQM is to express the meaning of the templates which covers purpose, perspective and environment; a set of guidelines also proposed for driving question and metrics. It provides a framework involving three steps: (i) List major goals of the development or maintenance project. (ii) Derive from each goal the questions that must be answered to determine if the goals are being met. (ii) Decide what must be measured in order to be able to answer the questions adequately.

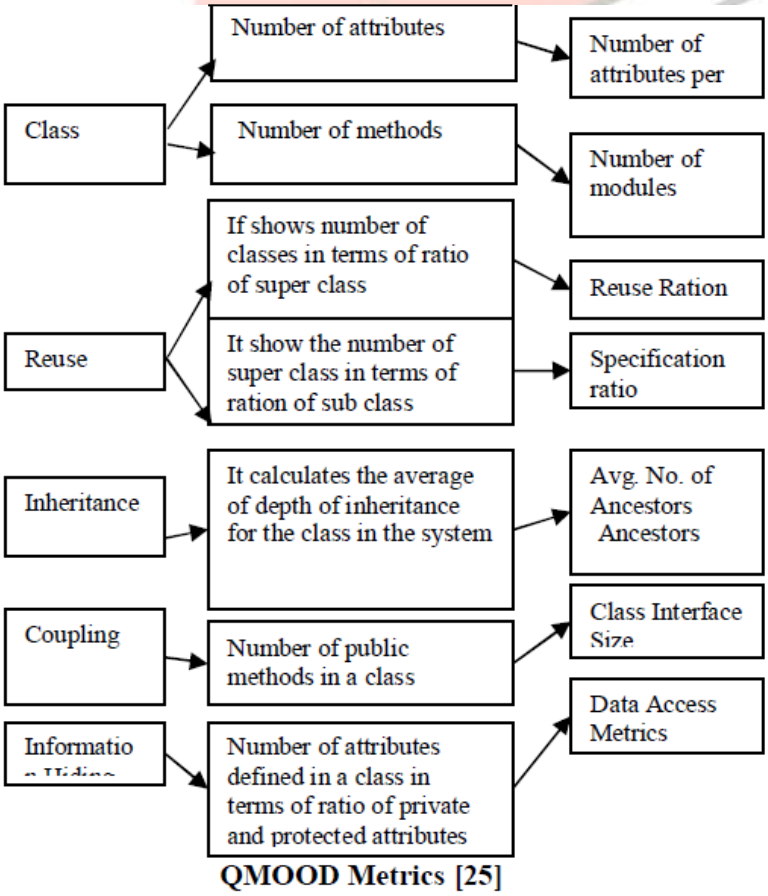
Goal (Conceptual level): A goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, relative to a particular environment. Objects of measurement are products, processes and resources.

Question (Operational level): A set of questions is used to characterize the way the assessment/achievement of a specific goal is going to be performed based on some characterizing OUTPUT.

Metric (Quantitative level): A set of data is associated with every question in order to answer it in a quantitative way. This data can be objectives and subjective, if they depend only on the objects that can be measured and not on the viewport from which they may be taken. For example, number of versions of a document, staff hours spent on a task, size of a program.

Quality Model for Object-Oriented Design (QMOOD):

The QMOOD [44] is a comprehensive quality model that establishes a clearly defined and empirically validated model to assess object-oriented design quality attributes such as understandability and reusability, and relates it through mathematical formulas, with structural object-oriented design properties such as encapsulation and coupling. The QMOOD model consists of six equations that establish relationship between six object-oriented design quality attributes (reusability, flexibility, understandability, functionality, extendibility, and effectiveness) and eleven design properties



QMOOD Metrics [25]

IV. SATC'S METRICS

Rosenberg Linda [46] proposed to select OO metrics that supports the goal of measuring the code, quality, result and they proposed many object-oriented metrics due to lack of theoretical basis and that can be validated. These metrics may be used to evaluate the OO concepts like methods, coupling and inheritance and mostly focus on both of the internal and external efficiency measures of the psychological complexity factors that affect the ability of the programmer. It proposed three traditional metrics and six new metrics for the objectoriente system metrics.

Traditional Metrics

(i) Cyclomatic Complexity (CC): Cyclomatic Complexity is used to measure the complexity of an algorithm in a method of class. Cyclomatic Complexity of methods can be combine with other methods to measure the complexity of the class. Generally, this is only used for the evaluation of quality attribute complexity.

(ii)Line of Code: It is a method used to evaluate the ease of understandability of the code by the developer and the maintainer. It can easily be counted by the counting the number of lines for the code and so on. Generally, used to measure the reusability and maintainability.

New OO Metrics

The six new OO metrics are may be discussed as:

(i) Weight Method per Class (WMC): It is used to count the methods implemented within a class. The number of methods and complexities involved as predictors, how many time and effort is required to develop and maintain the class.

(ii) Response for a Class (RFC): It is used to the combination of the complexity of a class through the number of methods and the communication of methods with other classes. This is used to evaluate the understandability and testability.

(iii) Lack of Cohesion of Method (LCOM): Cohesion is a degree of methods through which all the methods of the class are inter-related with one another and provide a well bounded behavior. It also measures the degree of similarity of methods by data inputs variables and attributes. Generally, ii is used to evaluate the efficiency and reusability.

(iv) Depth of Inheritance Tree (DIT): Inheritance is a relationship between the class that enables the programmer to use previously defined object including the operators and variables. It also helps to find out the inheritance depth of the tree from current node to the ancestor node. It is used to evaluate the reusability, efficiency, understandability and testability.

(v)Number of Children (NOC): This is used to measure the subclass subordinate to a class in the hierarchy. Greater the number of children means greater reusability and inheritance i.e. in the form of reuse. Generally, it is used to measure efficiency, testability and reusability.

SATC focused on some selected criteria for the OO metrics as:

(i) Efficiency of constructor design to decrease architecture
(ii) Specification of design and enhancement in testing structure
(iii) Increase capacity of psychological complexity.

LI W. METRICS

Li et al. [45] proposed six metrics are Number of Ancestor Classes (NAC), Number of Local Methods (NLM), Class Method Complexity (CMC), Number of Descendent Classes (NDC), Coupling Through Abstract data type (CTA), and Coupling through Message Passing (CTM).

(i)Number of Ancestor Classes (NAC): The Number of Ancestor classes (NAC) metric proposed as an alternative to the DIT metric measures the total number of ancestor classes from which a class inherits in the class inheritance hierarchy. The theoretical basis and viewpoints both are same as the DIT metric. In this the unit for the NAC metric is "class", justified that because the attribute that the NAC metric captures is the number of other classes' environments from which the class inherits.

(ii)Number of Local Methods (NLM): The Number of Local Methods metric (NLM) is defined as the number of the local methods defined in a class which are accessible outside the class. It measures the attributes of a class that WMC metric intends to capture. The theoretical basis and viewpoints are different from the WMC metric. The theoretical basis describes the attribute of a class that the NLM metric captures. This attribute is for the usage of the class in an object-oriented design because it indicates the size of a class's local interface through which other classes can use the class. They stated three viewpoints for NLM metric as following: 1) The NLM metric is directly linked to a programmer's effort when a class is reused in an Object-Oriented design. More the local methods in a class, the more effort is required to comprehend the class behavior.

2) The larger the local interface of a class, the more effort is needed to design, implement, test, and maintain the class.

3) The larger the local interface of a class, the more influence the class has on its descendent classes.

V. CONCLUSION

This paper introduces the basic metric suite for object-oriented design. The need for such metrics is particularly acute when an organization is adopting a new technology for which established practices have yet to be developed. It is unlikely that universally valid object-oriented quality measures and models could be devised, so that they would suit for all languages in all development environments and for different kind of application domains. Therefore measures and models should be investigated and validated locally in each studied environment. It should be also kept in mind that metrics are only guidelines and not rules. They are guidelines that give an indication of the progress that a project has made and the quality of design.

VI. REFERENCES

[1] Fenton N. (1993) New Software Quality Metrics Methodology Standards Fills Measurement Needs', IEEE Computer, April, pp. 105-106

[2] Briand L. C., Morasca S., Basili V. R. (1996) Propertybased Software Engineering Measurement, IEEE Transactions on Software Engineering, 22(1), pp. 68-86

[3] Kaner C. (2004) Software Engineering Metrics: What do They Measure and How Do We Know?' In Proc. 10th Int. Software Metrics Symposium, Metrics, pp. 1-10
[4] Fenton N. (1994) Software Measurement: A Necessary Scientific Basis', IEEE Transactions on Software Engineering, 20(3), pp. 199-206
[5] IEEE Computer Society (1998) Standard for Software Quality Metrics Methodology. Revision IEEE Standard, pp. 1061-1998
[6] Kitchenham B., Pfleeger S. L., Fenton N. (1995) Towards a Framework for Software Measurement Validation. IEEE Transactions on Software Engineering, 21(12), pp. 929-943
[7] Vengatesan K., and S. Selvarajan"Improved T-Cluster based scheme for combination gene scale expression data" International Conference on Radar, Communication and Computing (ICRCC), pp. 131-136. IEEE (2012).
[8] Kalaivanan M., and K. Vengatesan." Recommendation system based on statistical analysis of ranking from user. International Conference on Information Communication and Embedded Systems (ICICES), pp.479-484, IEEE, (2013).
[9] K. Vengatesan, S. Selvarajan: The performance Analysis of Microarray Data using Occurrence Clustering. International Journal of Mathematical Science and Engineering, Vol.3 (2) .pp 69-75 (2014).
[10] K Vengatesan, V Karuppuchamy, S Pragadeeswaran, A Selvaraj," FAST Clustering Algorithm for Maximizing the Feature Selection in High Dimensional Data", Volume – 4, Issue-2, International Journal of Mathematical Sciences and Engineering (IJMSE), December 2015