Synergy of Classical and Model-Based Object-Oriented (OO) Metrics in Reducing Test Costs

M. Raviraja Holla

Abstract--- Software testing and maintenance being interleaved phases span more in software life cycle. The efforts to minimize this span rely obviously on testing when maintenance is natural. The features of Object-Oriented (OO) software systems, when compared to the classical systems, claim much reducing the maintenance cost without necessarily the possibility of maintenance itself. It is natural that even such systems evolve due to many reasons. Though the specific reasons leading to the maintenance differ, the general rationale behind maintenance is to enhance the life-cycle and possibly the value of the existing system. Hence testing effort is more natural and significant even then. Moreover, the salient features of OO software systems further enhance the testing span despite their claim on maintenance. However, the availability of classical OO software metrics aid better early quality testing of OO systems. They exploit the critical parts of OO software systems thereby offering timely, thorough, and effective assurance. However there is not yet a common metric model in this regard. On the other hand, it is expected that the evolved model-based OO software metrics help define the subjective features more objectively facilitating users to perform metrics activities. The conflation of both classical and model-based metrics mutually alleviates their limitations and brings more synergy in reducing the test costs of OO software systems.

Keywords--- OO metrics, Object-Orientation, Model-based OO metrics, Software metrics.

I. INTRODUCTION

OBJECT-orientation (OO) and the associated OO metrics are the two prominent outcomes in the continuum of software process improvement. Despite their advantages, the systems developed with OO also undergo changes due to numerous reasons. Each modification each time mandates thorough testing. Testing of OO systems consumes more effort in terms of time and resource due to their refined and unique features. Therefore finding the more likely error-prone or critical part of the OO software well ahead may alleviate the cost of testing coping prudential schedule of effort for testing. Those critical parts can be determined by OO software quality models. The intuition is that high quality parts of software are less error prone and easy to maintain.

The object oriented software metrics derive design quality and therefore the software engineers can take their help to choose critical parts. However there is no consensus on a common OO software quality model that is acceptable widely despite long time research in the field. Further the existing OO metrics subject to serious criticisms obvious in progressive research community. One such criticism is the lack of theoretical base. There are other reasons also. The quality models attempt to establish relationship between external and internal attributes.

However the verification of reasonableness of such quality models themselves varies from model to model. As a consequence the theoretical validation of those relationships elicited in a model is difficult to achieve. Moreover the definitions of external attributes of a software find vary and lack consensus. The verification and validation of the models themselves are biased to the researcher-concerned principles. The attributes and their relationship involved in the metrics models need to be adapted to various metrics domains and metrics users. In order to make the models apt to the requirements of specific domains and specific users, those models are empirically validated, adjusted and refined. This is not an effective approach of using quality models. Therefore, an effective approach suitable for this measurement process is highly desired.

The evolved model-based approach to object-oriented software metrics clearly defines the artifacts of the measurement process. Though there are comments on such contemporarily available metrics, certainly the same cannot be viewed the rebuttals rather the opportunities to develop those established model-based OO metrics. This survey intends to explore possible conflation methodology to determine better the critical or change-prone parts of software using model-based OO design metrics. This blending even supplements the model-based-testing in particular.

II. SOFTWARE METRICS AND OO SOFTWARE METRICS
Quantifying the observations eases managing the underlying causes therein. Thus quantifying metrics help understand and control the reasons for those observations. Alike the software engineering discipline evolved the concept of software metrics. The software metrics evolved as a byproduct of software quality management. Software metrics measuring software engineering products, processes and professionals proved significant improvement in software quality. Thus incorporating software metric in the early stages of software development life cycle deviates from otherwise possible inferior path. It supplements both reducing the overall cost of

M. Raviraja Holla, Assistant Professor-I, Department of Information Technology, Manipal University Jaipur, Rajasthan, India, E-mail: raviraja.holla@jaipur.manipal.edu

DOI:10.9756/BIJSESC.4813
software development process, and increasing the software quality level.

The evolved OO software metrics promotes more the engineering aspects than the aspects of craft of software environment. They supplemented not only the technical requirements as well the managerial needs like measuring training for current and new software engineers to adapt to the object oriented principles. Even then the OO software metrics need to be validated to reap its practical benefits. Further the OO-software metrics are specific to the OO paradigm where as other software metrics are generic as they can be applied to the software development under any paradigms. Despite their specificity, however, OO-software metrics give specific practical benefits such as in evaluating the degree of object orientation of an implementation given the relative newness of the OO approach. This measure is a degree of object-orientation in the newly trained OO community. In addition, OO metrics can assure a common quality model with its obvious objective features by alleviating the difficulty in validation due to subjective features. Thus OO metrics aid in setting design standards for an organization.

Hecht. A et al.[1] brings criticisms on conventional metrics when applied to OO systems ([2],[3]) and emphasizes the shortcomings of existing metrics and the need for new explicit OO software metrics. The class design makes OO design different than data/procedure design [4]. A measure by Pfleeger uses counts of objects and methods to develop and test a cost estimation model for OO development[2]. Three metrics for OO graphical information system suggested by Moreau and Dominick do not provide formal, testable definitions [2].

The extant software metrics literature starts with six candidate metrics [1] for measuring the size and complexity of OO design. These metrics addressed the empirical notion of software complexity using metrics with firm basis in theoretical concepts in measurement. The proposed metrics are formally evaluated against a widely accepted list of software metric evaluation criteria.

The six candidate metrics proposed in Hecht. A et al., [1] are Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Response For A Class (RFC), Coupling Between Objects (CBO), Lack of Cohesion in Methods (LCOM). Mohamed El-Wakil et al., [3] has evaluated a number of existing software metrics using a list of desired properties. These properties are: Property 1 (P1): Non-Coarseness, Property 2 (P2): Non-uniqueness (notion of equivalence), Property 3 (P3): Permutation is significant, Property 4 (P4): Implementation not function is important, Property 5 (P5): Monotonicity, Property 6 (P6): Non-equivalence of interaction. Property 7 (P7): Interaction increases complexity. Further research in moving OO development management towards a strong theoretical base should provide a basis for significant future progress [1].

Shyam R. Chidamber et al., [5] extended the work in [1] by Hecht. A et al. They developed an automated data collection tool to collect an empirical sample of these metrics at two field sites in order to demonstrate their feasibility and suggested ways in which managers may use these metrics for process improvement. In evaluating these metrics against a set of standard criteria, they are found to both a) possess a number of desirable properties, and b) suggest some ways in which the OO approach may differ in terms of desirable or necessary design features from more traditional approaches. They anticipated future research to further investigate these apparent differences. In addition to the proposal and analytic test of theoretically-grounded metrics, Shyam R. Chidamber et al., [5] has also presented empirical data on these metrics from actual commercial systems. In addition to the usual benefits obtained from valid measurements, OO design metrics should offer needed insights into whether developers are following OO principles in their designs. This use of metrics may be an especially critical one as organizations begin the process of migrating their staffs toward the adoption of OO principles.

Vessey et al., [6] surveyed four object-oriented design quality models. The work of Hecht. A. et al.,[1] and Shyam R. Chidamber et al.,[5], called the CK metric suite, resulted in Metrics for Object-Oriented Software Engineering model (MOOSEmodel). This work has been seminal in defining, and validating quality models. Lorenz and Kidd metrics are criticized for not being a part of a quality model, however, they have the advantages of being well-defined, easy to collect, and could be computed in the early phases of design. Metrics for Object-Oriented Design (MOOD) model is a well-defined by mathematical formulas and Object Constraint Language (OCL) statements. This model is empirically validated, supported by a tool, and most importantly provides thresholds that could be used to judge the metrics collected from a given design. The Quality Metrics for Object-Oriented Design (QMOOD) model enjoys similar properties as the MOOD model of being well-defined, empirically validated and supported by a tool. QMOOD distinguishes itself by providing mathematical formulas that links design quality attributes with design metrics. This allowed computing a Total Quality Index (TQI), which were already used by JagdishBansiya et al., [4] authors to compare fourteen class diagrams.

JagdishBansiya et al., [4] proposed a hierarchical QMOOD of four steps shown in the Figure 1 was proposed.

Vessey et al.,[6] proposed a group of desirable properties for OOD quality models, and then they are used to compare the presented OOD quality models. Based on this comparison, [6] concludes that the QMOOD suite is the most complete, comprehensive, and supported suite. The results of assessing the presented OOD quality models against proposed OOD quality desirable properties are summarized below in Table 1.

However, the theories, techniques and tools of OO software metrics are still under developing [13]. Forma
software metrics to measure the OO software product and to derive a metrics tool to support the OO activities is still the need of the hour. Further the application of OO metrics in the component-based software development process is also an area of research.

Table 1: Assessment of the four major OOD quality models against our proposed OOD quality models desirable properties

<table>
<thead>
<tr>
<th>Property</th>
<th>MOOSE</th>
<th>LK OO Metrics</th>
<th>MOOD</th>
<th>QMOOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependence on high level design characteristics only</td>
<td>NO¹</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Explicit quality characteristics</td>
<td>NO²</td>
<td>NO³</td>
<td>Error Density, Fault Density and Normalized Rework</td>
<td>Reusability, Flexibility, Understandability, Extendibility, and Effectiveness</td>
</tr>
<tr>
<td>Precise metrics definitions</td>
<td>Yes, except the WMC</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Formal relationships</td>
<td>NO²</td>
<td>NO³</td>
<td>YES (Pearson r correlation coefficients)</td>
<td>(Equations)</td>
</tr>
<tr>
<td>Results interpretation</td>
<td>NO²</td>
<td>NO³</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

¹Chidamber and Kemmeren metric set focuses on class level metrics and that several of them are highly dependent on low level metrics for their derivation, and as such are not ideally suited to early stage design analysis.
²The original MOOSE suite did not refer to any quality characteristic, however, later experiments linked the MOOSE with maintenance effort [8] and fault proneness [9]
³Lorenz and Kidd metrics are criticized [10] for being mere counts of class properties such as the number of public methods, the number of all instance methods, the number of all class methods and so.

The metrics research started formulating the rational relation between external and internal attributes of software products. They are termed the metrics model or quality model. As early as in 1977 and 1978, McCall quality model[14] and Boehm quality model[15] were proposed respectively. In recent years, ISO 9126 model has been proposed [16]. It improves the McCall model and defines six factors but does not elaborate on criteria and metrics layers. REBOOT model [17] is composed of one quality model and one reusability model, which give detailed definition and description for each layer.

Despite long time research on software quality continued, there has not been a common quality model yet which can be accepted widely. There are several reasons. The external attributes are not yet formal enough to formalize the models. The theoretic verification and validation of the models is also difficult as the associations between internal and external attributes are non-theoretic yet. Many software metrics researchers validate and analyze the proposed metrics models only according to some verification principles they are concerned about.

According to various metrics domains and metrics users, the attributes and their association involved in the metrics models should be adapted accordingly. By validating empirically the metrics models in practice, metrics users can adjust and improve the models, making them fulfill the requirements in specific domains and for specific users. Therefore, an effective approach suitable for this measurement process is greatly wanted to facilitate users to perform the metrics activities.

III. MODEL-BASED OO METRICS

MEI Hong et al., [13] proposed a model-based approach to OO software metrics. This approach clearly defines the activities and procedure of the measurement process. It explicitly proposes the conception of absolute normalization computation and relative normalization computation for metrics model, which can be applied to different situations respectively.

Moreover, MEI Hong et al., [13] implemented this approach in their developed generic software metrics tool Jade Bird OO Metrics Tool (JBOOMT). This tool provides a flexible user interface for metrics users to filter and tune in calculating and reviewing the metrics data. Users can customize the metrics model based on the specific requirements of a measurement task in hand. Therefore the tool can effectively support customizing, browsing and comparing the hierarchical metrics models. The proposed approach is the top-down process to develop the metrics model. JBOOMT can also be extended to support some other object-oriented languages such as Java [13]. Moreover, it is also necessary to formulate some more appropriate metrics models for components in the component library and also some reusability metrics models which can be used to extract reusable components through the model-based approach. In addition, a lot of case studies should be done to validate and improve the correctness and usability of these metrics models [13]. At the same time, the functionality of JBOOMT could be further enhanced.

IV. SYNERGY IN PREDICTING CRITICAL PARTS

OO methodology promises mitigating maintenance effort or change-management effort. This in turn indirectly promises to lessen testing effort. However, testing of OO systems lies in another dimension obvious to the salient features of OO systems. Therefore predicting source code changes has become a crucial factor to synergize both maintenance and testing.

Ali R. Sharafatet al., [18] proposed a probabilistic approach to predict changes in OO systems. This technique is applied on an OO open source project, JFlex [19]. The proposed approach uses the axis of time to define and guide the prediction process. This approach is noteworthy for two main reasons. First, it attempts to address a problem that has challenged the research community for several years, namely the maintenance of OO mission critical systems. Second, it aims to devise a workbench in which the changes to the source code do not occur in a vacuum, but can be evaluated and fine-tuned in order to address specific quality requirements for the
new target system such as enhancements in maintainability. Ali R. Sharafat et al., [18] proposed future work on the extensibility of the proposed approach on large scale projects with the scope of determining those parameters directly from the source code, the application domain, or any other related data rather than calculating empirically. There are several possible solutions discarded due to the lack of resources which can be reconsidered when new case studies are analyzed.

Varun Gupta and Dr. Jitender Kumar Chhabra [20] identified mainly two kinds of metric for OO software – static software metrics and dynamic software metrics. The static software metrics are obtained from static analysis of the software, whereas dynamic software metrics are computed on the basis of data collected during the execution of the software. Varun Gupta et al., [20] discuss some new static and dynamic software metrics for the measurement of design and complexity of OO software grouped into three categories – package level metrics, dynamic metrics and cognitive complexity metrics.

Sinan Eskiet al., [21] proposed a method that is based on OO design metrics to determine the critical and change-prone parts of software. Experimental results show that this approach can find up to 80 percent of the change-prone parts. In this study, QMOOD [4] and CK [5] metric suites are used to detect class change proneness. Sinan Eskiet al., [21] define “change cost” value for classes, which is calculated according to the differences between two versions of the class. In this calculation, to get the possible effect of the change, modifications are weighted according to their architectural impacts on design. The results of evaluation revealed that by applying this method in last phases of software development, critical classes are determined. These classes should be tested first and more deeply because they are possibly unstable or immature.

On the other hand, the model programs can change the way the software is developed. One can begin checking and testing development products earlier in the project. In general, model programs can be helpful during specification, design as well as testing. However it is not necessary to model everything; projects typically focus their modeling efforts on the system behaviors or components that are the most novel, critical, or intricate. Object oriented software development uses models consistently in all stages. The synergy of both classical and model based object oriented software metrics can reduce testing costs by early predicting the error-prone artifacts of OO software.

V. CONCLUSION

The literatures suggest ample opportunity for the research in OO metrics. On the other hand there are comments on contemporarily available model-based OO metrics. But those comments cannot be viewed as rebuttals rather the opportunities to develop those established model-based OO metrics. Therefore a conflation methodology of conventional OO metrics with model based OO metrics to find error-prone classes alleviate the shortcomings of both approaches when considered separately. This blending even supplements the model-based-testing in particular.

REFERENCES


