Capturing OO Software Metrics to attain Quality Attributes – A case study

Mandeep K. Chawla[1], Dr. Indu Chhabra[2]
[1]Research Scholar, Department of Computer Science, Panjab University, Chandigarh. Email: manu.chw@gmail.com
[2]Associate Professor, Department of Computer Science, Panjab University, Chandigarh.

Abstract— This paper is an attempt to study the role of various object oriented software metrics with regard to software quality and implementing them on an open source Java based operating system to evaluate its design traits. Captured metrics have been mapped to higher level quality attributes characterized by the QMOOD quality model. Prior to mapping, metric values have been normalized to enable us to interpret the meaning of quantitative figures calculated to indicate quality factors. This one offers a concrete model resulting in computation of measures such as reusability, flexibility, understandability, functionality and alike using composite metrics. Also it is economical to implement this type of model and can be effectively used in observing many aspects of software product quality. This would enable the project managers to foresee the problem areas and refactor the code to eliminate anomalies and undesirable complexities which may crop up post-deployment. Besides early assessment, it should considerably facilitate in better project planning and efficient resource-allocation.

Index Terms— Software quality, object-oriented software metrics, quality attributes, QMOOD model, JNode OS, metric collection tool, normalization.

1 INTRODUCTION

Software Quality evaluation entails set of processes that consume great deal of time and resources to maintain sufficient level of software quality. Besides, application of comprehensive testing suites consistently over all the components of a software is not practically feasible. Early in the development stages, project managers would like to identify segments of classes which are potentially more fault-prone and are characterized by other complexities beyond an acceptable limit, that too with minimum efforts and time. Software metrics play essentially a vital role in quantifying certain properties of a software artifact to gain insight into its qualitative aspects. The IEEE Standard Glossary of Software Engineering Terms [1] defines a metric as "a quantitative measure of the degree to which a system, component, or process possesses a given attribute." When processes are going off-track, these are the indicators that enable managers to take pro-active actions to bring them back under control. They can also ascertain the degree of success or failure of an end product by assessing their internal attributes. There exist several metrics to capture the quality of object-oriented (OO) design and processes, such as proposed by Chidamber and Kemerer [2] popularly known as CK metrics (WMC, RFC, DIT, LCOM, CBO, NOC), Lorenz and Kidd Metrics [3], Li and Henry Metrics [4], Tang et al. [5], Henderson-sellers [6], Goal Question Metrics [7], and some other lesser known also exist. In addition there exist many quality models, for instance, McCall's (1977), Boehm (1978), Dromey (1995), ISO/IEC 9126, SQuaRE (1999), QMOOD (2002) which define components for specifying and measuring quality and for assessing and aggregating the measurement results. To examine the actual implications of a quality evaluation approach, it needs to be practically tested on a specific software product with the aid of a compatible metric collection tool.

Rest of the paper is organized as follows. Section 2 discusses relevant research work in the subject. Section 3 gives a brief description of QMOOD quality model [8] followed. Section 4 identifies data set, tool and related metrics. Section 5 presents metric calculation results in the form of descriptive statistics. Section 6 describes normalization of chosen metrics & deriving higher level quality attributes. Section 7 pinpoints the threats to validity of results. Concluding remarks are given in Section 8.

2 RELATED WORK

Researchers have done various empirical studies to evaluate the effect of OO metrics on software quality and introducing models that utilize them in estimating fault-proneness and predicting quality attributes. MOOD metrics model [9] describes basic structural mechanism of the object oriented paradigm as encapsulation, inheritance, polymorphism and message passing. Briand et al. [10] empirically explore the relationships between existing object oriented measures and observes that size of classes, frequency of method invocations and depth of inheritance also affect fault proneness. Subramanyam et al. [11] uses CK Metrics suite to find software defects using two programming languages C++ and Java and results vary according to the language chosen. Nagappan [12] in his large empirical study of five Microsoft software systems found that failure prone software entities are statistically correlated with code complexity measures. Jiang [13] demonstrated that code-level metrics perform better than design level metrics and combination of them performs the best. Y Ma [14] suggests a hybrid set of complexity metrics for large scale object oriented systems. Authors [15] calculate and optimizes thresholds for set of software metrics. Authors in their paper [16] conducted an experiment to evaluate the

practical use of the proposed thresholds for a set of OO metrics. Authors in their comprehensive literature review [17], reported OO and process metrics to be more successful in finding faults compared to traditional size and complexity metrics.

3 THE QUALITY MODEL ADOPTED

A Quality Model is defined as "The set of characteristics and the relationships between them which provides the basis for specifying requirements and evaluating quality" [ISO 15498-1]. Models usually decompose quality into a hierarchy of criteria and attributes. These hierarchical models lead to metrics at their lowest level [18]. Metrics are directly measurable attributes of software and they are used to express certain aspects of the product that affect quality. Several approaches to model the quality of a software product have been recommended in the literature. In this paper we have implemented a model that focuses on the quality of the design by using source code metrics. Given by Bansiya et. al. [8], QMOOD is a hierarchical model which implements a way to map source code metrics to higher abstraction levels. Table 1 shows the computation formulas for quality attributes according to QMOOD.

TABLE 1: COMPUTATION FORMULAS FOR QUALITY ATTRIBUTE FOLLOWED BY QMOOD [8]

Quality Attribute	Index Computation
Reusability	-0.25 * Coupling + 0.25 * Cohesion + 0.5 * Messaging + 0.5 * DesignSize
Flexibility	0.25 * Encapsulation - 0.25 * Coupling + 0.5 * Composition + 0.5 *Polymorphism
Understandability	-0.33 * Abstraction + 0.33 * Encapsulation - 0.33 * Coupling + 0.33 * Cohesion -0.33 * Polymorphism -0.33 * Complexity -0.33 * DesignSize
Functionality	0.12 * Cohesion + 0.22 * Polymorphism + 0.22 * Messaging + 0.22 * DesignSize + 0.22 * Hierarchies
Extendibility	0.5 * Abstraction - 0.5 * Coupling + 0.5 * Inheritance + 0.5 * Polymorphism
Effectiveness	0.2 * Abstraction + 0.2 * Encapsulation + 0.2 * Composition + 0.2 * Inheritance + 0.2 * Polymorphism

Primarily QMOOD model was designed for evaluation with C++/Visual C++ validation suite though it can be applied on any object oriented language based software.

4 DATA SET, TOOL AND METRICS SELECTION

In order to verify the practical applicability of the adopted model, a java based operating system "JNode" (Version 0.2.7) [19] has been chosen as a case study. It is an open source multi-version software project to create a Java platform operating system. JNode source tree is divided into 10 major modules which further contain many more subdirectories and files. It spans over more than 15000 classes and beyond 1.9 million lines of code. Since it would have been very computation-intensive to assess whole OS in one go, hence JNode's key component called 'Core' is opted for evaluation. JNode-Core contains virtual machine code, the classpath java library sources and the core of the JNode OS, including the plugin manager, driver framework, resource manager and security manager. This is by far the largest and most complex module; covers 83% of total OS size and therefore the most convincing representative of whole software.

To measure the source code, Eclipse IDE's plugin 'Metrics 1.3.6' [20] has been implemented on JNode-Core module. This tool analyzes the source code, generates about 20 metrics and allows exporting the results in .xml file for further study. The metrics produced are not exactly the same as defined in QMOOD metric-suite, yet bear many resemblances. For some, replacement is straight forward; while for rest, other related metrics were used to derive them. According to Bansiya et. al. [8], at the lowest level, metrics used to assess design properties may be changed or a different set of design properties may be used to assess quality attributes. Therefore our step of replacing metrics is justified. Table 2 records QMOOD design metrics and our corresponding replacement metrics.

TABLE 2: QMOOD DESIGN METRICS & OUR SUBSTITUTE METRICS

Design Property [QMOOD]	Design Metric [QMOOD]	Equivalent metric Computed/ derived (in this paper)	Construct
Coupling	Direct Class Coupling (DCC)	Efferent Coupling (CE)	Package
Cohesion	Cohesion Among Methods of Classes (CAM)	*Lack of Cohesion of Methods (LCOM)	Class
Messaging	Class Interface Size (CIS)	Number of Methods (NOM)	Class
Design Size	Design Size in Classes (DSC)	Number of Classes (NOC)	Package
Encapsulation	Data Access Metric (DAM)	1	-
Composition	Measure of Aggregation (MOA)	Number of Attributes (NOF)	Class
Polymorphism	Number of Polymorphic Methods (NOP)	Number of Overridden Methods (NORM)	Class
Abstraction	Average Number of Ancestors (ANA)	Abstractness (RMA)	Package
Complexity	Number of Methods (NOM)	Weighted methods per Class (WMC)	Class
Hierarchies	Number of Hierarchies (NOH)	Depth of Inheritance Tree (DIT)	Class
Inheritance	Measure of Functional Abstraction (MFA)	* [∑ NORM / NOM * 100]	Class

** Equivalent Design-metric 'derived' from related metric*

Following is a brief explanation of some of the metrics replaced for the ones dictated by chosen model.

a) Cohesion: It is the degree to which the methods within a class are related to one another. High cohesion indicates good class subdivision [21]. CAM is conversely related to LCOM and can be easily derived from latter by taking its reciprocal value.
b) Messaging: Originally CIS includes number of public methods in a class. But we are constrained to consider total number of methods instead, due to unavailability of measure suggested primarily.
c) Encapsulation: DAM is the ratio of number of private (protected) attributes to the total number of attributes declared in a class. The tool selected does not provide support for this metric, neither does it provide enough information to derive this metric. Therefore, value "1" has been fixed as a neutral measure in this study.
d) Composition: MOA is the count of number of data declarations whose types are user defined classes. Since our tool does not support this metric as well, therefore this has been compensated by number of attributes (NOF) assuming that the attributes inclusive of primitive data types along with user-defined can also indicate the composition design property.
e) Complexity: According to NASA-SATC [21] review, WMC is computed as sum of complexities (cyclomatic complexity) of methods in a class or count of methods of a class. Chidamber and Kemerer [2] suggests number-of-methods as measure of class complexity in WMC metric. When all methods are weighted equally, the WMC metric has the same measure as number of methods (NOM) in a class. However, sum-of-complexities more accurately depicts this metric and our tool also makes it available, so we decided to use this metric.
f) Inheritance: Since this metric was not readily available, we could derive it through other two available metrics – as ratio of number of overridden methods (NORM) to total number of methods (NOM).

5 Computing Metrics

After these comprehensive pre-preparations, we are all set to compute metrics by applying software metric tool. Since it is an eclipse plug-in, it first required us to import JNode source in eclipse IDE and build (compile) it successfully. Moreover, for a considerable size software- build process, computing metrics through plug-in and finally exporting it to .xml file is quite a CPU & memory-intensive task and it took hours of processing before delivering the results. Table 3 shows the descriptive statistics for metrics computed through eclipse plug-in, along with the Z-score which is calculated separately.

TABLE 3 : DESCRIPTIVE STATISTICS FOR METRICS COMPUTED

Metric	MAX	AVG	STD-DEV	SUM	Z-value (normalized)
CE	260	7.138	14.864	-	17.02
*Cohesion (1/LCOM)	1.778	0.197	0.314	-	0.20
NOM	1189	8.032	17.61	126577	67.06
NOC	389	14.211	29.736	15760	12.61
* Encap-sulation =1	-	-	-	-	#
NOF	193	2.192	4.734	34545	40.34
NORM	193	0.883	3.756	13914	51.23
RMA	0.26	0.32	1	-	2.31
WMC	1785	22.421	51.662	353362	34.12
DIT	9	2.172	1.296	-	5.27
*Inheri-tance=10.99	-	-	-	-	#

- indicates statistics not-available or not-applicable
** represents fixed value or derived as mentioned in Section 4*
 # represents already-normalized metric value

As mentioned in Table 2, some of the measures computed are class-wise and some are package-wise. Since these measures further have to make up as input parameters to attain quality attributes, we need to choose one aggregate value from each metric's domain of quantities. For this purpose, maximum value obtained for each metric has been selected. It makes sense because upper-limit indicates up to what level a measure is spread out for the software. One may argue that, in many cases it may be an outlier and final outcomes should not be based on a few extreme values. The justification lies in the next step of normalization (process of computing standard scores); where standard deviation and mean for each set of measures would be used as parameters to get normalized value. Standardized scores retain the order of the values and more important, the shape, center, and spread of the distribution.

6 Normalizing & Deriving Higher Level Quality Attributes

One may observe in Table 3 that metric values are of different ranges and combining them (as raw) further to attain higher level quality attributes would lead us to the problem of interpretation. In that case, we have two choices – either we compare them with some other releases of the same software or we normalize them into a smaller, more interpretable range. Since first option is beyond the scope of this paper, so we go for the second one. For this we compute the Z-value [22] which is commonly used in statistics to convert arbitrary data into z-scores, typically with a mean of 0 and variance of 1. They are also called 'standard-scores' and are calculated for a series of data by subtracting the mean from the observed-measurement ('max' value in our case) and dividing by the standard deviation. Using standard scores or percentiles, it is also possible to compare scores from different distributions where measurement is based on a different scale.

Last column of Table 3 records normalized metric values which are further used to obtain quality attributes (in Table 1). Normalization should be performed before adding the metrics values at the bottom level in the model to calculate items at the higher level. And then the values of the higher level items in the model can be derived from the weighted sum of the values of those items associated with it [23].

Accordingly, after computation, Fig. 1 shows the plot of Reusability, Flexibility, Functionality, Extendibility and Effectiveness for JNode-Core.

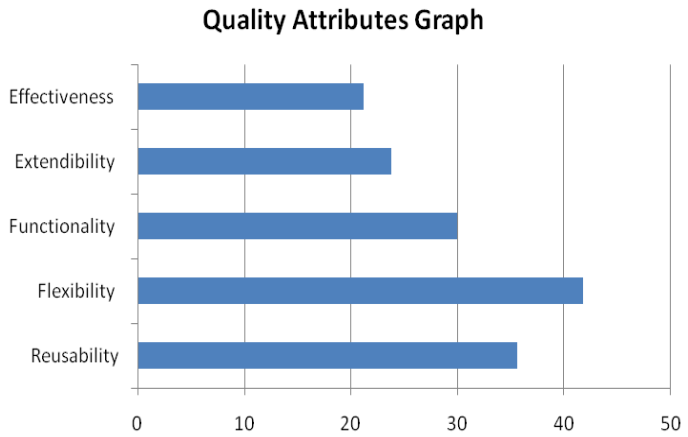

Fig. 1: Quality Attributes Graph

Understandability is the only quality attribute that is characterized with a negative value and has not been plotted in graph. It agrees with the expectation that each release portray a developing framework that takes on additional functionality in each new revision and is therefore expected to be harder to learn and understand [8]. However, all other attributes are positively associated as one can anticipate.

Additionally, there is always a trade-off among choosing the thresholds for various quality attributes. A desirable increase in one may lead to compromising decline in other. Nevertheless it should be balanced according to client's priorities or organization's strategic goals.

7 Threats to Validity

A number of threats to validity should be addressed in future:
i) Findings may be biased towards the single representative data set which was opted for case study. In next phase, we should pick multiple releases and their comparison analysis will shed more light on validity of results.
ii) Selection of replacement metrics for the original ones might have put across the things differently.
iii) Choice of metric collection tool and its measurement precision may be a source of bias.
iv) Additional metric sets, if taken into consideration, might have influenced the quality of end product other way.
v) The Quality model chosen may itself be another source of bias.
Nonetheless there is less likelihood that selection of different data set, metrics or tools would altogether change the conclusions drawn or would end up largely in conflict with the estimation accuracy. However, we encourage readers to test more data sets with additional metrics and tools.

8 Concluding Remarks

In this paper, we have conducted source code analysis for 'core' component of one version of JNode OS, an open source Java based platform and presented the preliminary results of this study. For this purpose we utilized several object oriented metrics, calculated their values through a tool 'Metrics 1.3.6' (an eclipse plugin), normalized the metric scores, and further used them as input parameters to compute higher level quality attributes dictated by QMOOD quality model. Results indicate that a concrete quality model using an OO metrics suite is useful and cost-effective medium for evaluation of a software artifact and can reveal the potential problem spots which need to be taken care off before deployment. Almost all metrics are employed over and again to obtain multiple quality attributes, though with different weights every time. Some are positively correlated to higher level entities while a few behave conversely. It would be beneficial for project managers to identify weighted contribution of these metrics in attaining qualitative aspects and allocate resources for improvement areas accordingly. There is no denying that code metrics alone are not sufficient to make concluding statements for a product's ultimate quality, yet their contribution is significant in an organization's comprehensive quality assessment program.

9 References

[1] Institute of Electrical and Electronics Engineers, "IEEE Standard Glossary of Software Engineering Terminology", IEEE Std 610.12-1990.
[2] Chidamber S. and Kemerer C., "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, vol. 2O, no. 6, pp. 476-493, 1994.
[3] M. Lorenz, J. Kidd, "Object Oriented Software Metrics", Prentice Hall, NJ, (1994).
[4] W.Li, Sallie, Henry, "Metrics for Object-Oriented system", Transactions on Software Engineering,1995.
[5] Tang M-H., Kao M-H. and Chen M-H, "An Empirical Study on Object-Oriented Metrics", Proc. of The Software Metrics Symposium, 1999, 242-249.
[6] Henderson-Sellers B., Object-Oriented Metrics, measures of Complexity", Prentice Hall, 1996.
[7] V.L.Basili, L. Briand and W. L. Melo, "A validation of object-oriented Metrics as Quality Indicators", IEEE Transaction Software Engineering. Vol. 22, No. 10,1996, pp. 751-761.
[8] Jagdish Bansiya and Carl G. Davis, "A hierarchical model for object-oriented design quality assessment", Software Engineering, IEEE Transactions on, 28(1):4–17, 2002.
[9] Abreu, F. B., "The MOOD Metrics Set", presented at ECOOP '95 Workshop on Metrics, 1995.
[10] L. Briand, J. Wüst, John W. Daly, V. Porter, "Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems", Journal of Systems and Software, 51 (2000) p 245-273.
[11] Subramanyam, R. and M.S. Krishnan, "Empirical analysis of CK metrics for object oriented design complexity", Implications for software defects. IEEE Trans. Software Eng., vol. 29, 2003, pp. 297-310.
[12] Nagappan, N., Ball, T., Zeller, A., "Mining metrics to predict component failures", In ICSE(2006) 452-461, 2006.
[13] Y. Jiang, B. Cukic, T. Menzies, N. Bartlow, "Comparing Design and Code Metrics for Software Quality Prediction", Proceedings of the Workshop on Predictive Models in Software Engineering (PROMISE'08), Leipzig, Germany, May 2008.
[14] Yutao Ma, Keqing He, Bing Li, Jing Liu, Xiao-Yan Zhou, "A Hybrid Set of Complexity Metrics for Large-Scale Object-

Oriented Software Systems", J. Comput. Sci. Technol. 25(6): 1184-1201 (2010)
[15] Steffen Herbold, Jens Grabowski, Stephan Waack, "Calculation and optimization of thresholds for sets of software metrics", Empirical Software Engineering 16(6): 812-841 (2011)
[16] K. A. M. Ferreira, M. A. S. Bigonha, R. S. Bigonha, L. F. O. Mendes, and H. C. Almeida, "Identifying thresholds for object-oriented software metrics," The Journal of Systems and Software, vol. 85, no. 2, pp. 244–257, 2012.
[17] D Radjenovic, M Hericko, Richard Torkar, Ales Zivkovic, "Software Fault Prediction Metrics: A Systematic Literature Review", Information & Software Technology 2013. (in press)
[18] Samoladas, I., G. Gousios, D. Spinellis and I. Stamelos, "The sqo-oss quality model: measurement based open source software evaluation", in: OSS'08: Proceedings of International Conference on Open Source Systems 2008.
[19] JNode team: URL www.jnode.org/
[20] Frank Sauer, Eclipse Metrics Plug-in. URL http://metrics.sourceforge.net/
[21] L. Rosenberg and L. Hyatt, "Software Quality Metrics for Object-Oriented System Environments", NASA Technical Report SATC no. 1, pp 11-58, 2OO1.
[22] Triola MF. 1995a, "Elementary Statistics (6th edition). Addison-Wesley: Reading, MA.
[23] Hong Mei, Tao Xie, Fuqing Yang, "A Model-Based Approach to Object-Oriented Software Metrics", J. Comput. Sci. Technol. 17(6): 757-769, 2002