The Analysis of Object-Oriented Metrics in C++ Programs

Normi Sham Awang Abu Bakar

Abstract—Software metrics play a very important role to develop good quality software. In today's software development, object-oriented (OO) languages are used due to their basic features like class, object, information hiding, inheritance, encapsulation, abstraction and polymorphism. In addition, a number of object-oriented metrics are available to be used to measure the quality of the OO systems. One of the most popular OO metrics suite is the Chidamber and Kemerer (CK) metrics suite. The main focus of this research is to apply the CK metrics to a group of open source systems written in C++. The results show that only two out of six CK metrics, NOC and RFC are significant in predicting defects in the systems.

Index Terms—Object-oriented language, software metrics, C++, CK metrics, open source system.

I. INTRODUCTION

Software metrics refers to a broad range of measurements for computer software. Although the terms “measure”, “measurement” and “metrics” are often used interchangeably, it is important to note the subtle differences between them. Within the software engineering context, a measure provides a quantitative indication of the extent, amount, dimensions, capacity, or size of some attribute of a product or process. Measurement is the act of determining a measure. The IEEE Standard Glossary of Software Engineering Terms [1] defines metric as “a quantitative measure of the degree to which a system, component, or process possesses a given attribute.”

There are many different types of metrics proposed in the literature for the measurement of multiple properties associated with any software system of a reasonable size. Process metrics measure facets of the software development process. Product metrics measure facets of project deliverables. This paper is concerned with product metrics related directly to application source code and its inherent design.

Design product metrics can be sub-divided into module design and structural metrics. Module design metrics measure aspects within a module, whereas structural design metrics concentrate on measuring aspects of the interactions between modules [2].

The popularity of open source systems (OSS) has made it possible to have easy access on the empirical data for research. Researchers now have access to rich repositories for large projects used by thousands of users and developed by

hundreds of developers over extended periods of time. This has catalyzed many breakthrough results in many areas of software engineering research, such as software maintenance, metrics and measurement, code quality, developers' communication, development culture and many more.

This paper describes the application of several design metrics to the analysis of 30 open source systems, and presents the results obtained. This research is mainly focused on systems written in C++, and all of them were collected from Sourceforge.net [3], the biggest open source repository.

The data collection process is assisted by a tool written to mine data from the large repository, also known as OSSGrab [4]. The usage of this tool saves a considerable amount of time which instead can be spent on data analysis and model building.

Essentially, this paper seeks to discuss the application of Chidamber and Kemerer Object-oriented metrics suite (CK OO metrics) on various open source systems. This paper is organized as follows: Section II presents several related works, and the object-oriented metrics is discussed in Section III. The discussion continues in Section IV where the research background is described in detail while Section V highlights the research results. Finally, Section VI concludes this paper and presents future work.

II. RELATED WORK

Among the object-oriented metrics being introduced, the Chidamber and Kemerer or CK metrics are the most validated by previous researchers. This research focuses on the CK OO metrics to be used as the design metrics for a group of C++ systems.

Sherif and Sanderson [5] applied the CK OO metrics to measure the metrics for two projects, UGC and SEQGEN which were written in C++. They found that the metrics gave good results and insights into comparing the complexity of the two projects and also the complexity of individual classes within a project. Tang *et al.* [6] attempted to correlate the CK metrics with defects and they found WMC is a good predictor of faulty classes and RFC is a good indicator for faults. Later, Wilkie and Harmer [7] introduced a tool, Extensible Metrics toolBench for Empirical Research (EMBER) to measure the complexity of object-oriented software systems using CK metrics.

Li and Henry used CK OO metrics plus several others including some size metrics (such as the number of attributes plus the number of local methods), to determine whether object-oriented metrics could predict maintenance effort. For the weights in WMC, they used cyclomatic numbers. On the basis of an empirical study using regression analysis, they concluded that these measures are useful. In addition, they claim that the Chidamber-Kemerer metrics “contribute to the

Manuscript received September 12, 2014; revised November 12, 2014. This work was funded by the Ministry of Higher Education Malaysia under the Research Acculturation Grant Scheme (RAGS), grant number RAGS12-041-0041.

Normi Sham Awang Abu Bakar is with the Department of Computer Science International Islamic University Malaysia Kuala Lumpur, Malaysia (e-mail: nsham@iiu.edu.my).

prediction of maintenance effort over and beyond what can be predicted using size metrics alone” [8].

El-Emam *et al.* have investigated the confounding of class size on the validity of object-oriented metrics. They studied a large C++ telecommunications framework using the CK metrics and a subset of Lorenz and Kidd metrics [9] and they found that before controlling for size, the results indicate there are correlations between the metrics and fault proneness. However, after controlling for size, none of the metrics they studied associated with fault-proneness anymore [10]. This study was supported by Subramanyam and Krishnan [11].

A recent work by Goel and Bhatia [12] investigates the reusability of object-oriented systems using CK metrics. They study three programs to test their theory, the first program uses multilevel inheritance, the second program uses multiple inheritance and the third program uses hierarchical inheritance. In the end, they concluded that multilevel inheritance has more impact on reusability. Another work by Kumari and Basin [13] attempt to compare several programs written in C++ and Java using the CK metrics and they found that Java proves to be more object-oriented than C++.

Compared to the previous research, this research focuses on finding which metrics have the most influence on defect density in the systems. Moreover, most prior work use data from commercial/industrial systems, while, this work uses data from open source projects.

III. OBJECT-ORIENTED METRICS

The measures of object-oriented systems are derived from traditional design techniques, for example, coupling and cohesion and then interpreted for object-oriented approaches. Chidamber and Kemerer have suggested measures for object-oriented systems [14], also known as Chidamber and Kemerer Object-Oriented metric suite (CK OO metrics). The basis for the empirical relations systems in the Chidamber and Kemerer’s work is the set of “ontological principles” proposed by Bunge [15] and later applied to object-oriented systems by Yand and Weber [16]. In the latter work, the world is viewed as being composed of substantial individuals that possess a finite set of properties. Collectively, a substantial individual and its properties constitute an object. A class is a set of objects that have common properties, and a method is an operation on an object that is defined as part of the declaration of the class [17].

Attributes such as coupling, cohesion, object complexity, and scope of properties are then defined in Bunge’s “ontological” terms. For instance, in Bunge’s terminology, two objects are coupled if and only if one of them acts upon the other. X is said to act upon Y if the history of Y is affected by X, where history is defined as the chronologically ordered states that a substantial individual traverses in time [17].

Chidamber and Kemerer (CK) use the object-oriented concepts to define a number of metrics that are claimed to relate to some of the attributes of Bunge’s ontology [Chidamber and Kemerer, 1994]:

Metric 1: Weighted methods per class (WMC): This metric is intended to relate to the notion of complexity. For a class C with methods $M1; M2; \dots; Mn$; weighted respectively

with “complexity” $c1; c2; \dots; cn$; the measure is calculated as:

$$WMC = \sum_{i=1}^n c_i \quad (1)$$

Metric 2: Depth of inheritance tree (DIT): In an object-oriented, the application domain is modeled as hierarchy of classes. This hierarchy can be represented as a tree, called the inheritance tree. The nodes in the tree, called the inheritance tree. The nodes in the tree represent classes, and for each such class, the DIT metric is the length of the maximum path from the node to the root of the tree. This measure relates to the notion of scope of properties. DIT is a measure of how many ancestor classes can potentially affect this class.

Metric 3: Number of children (NOC): This metric relates to a node (class) of the inheritance tree. It is the number of immediate successors of the class.

Metric 4: Coupling between object classes (CBO): For a given class, this measure is defined to be the number of other classes to which the class is coupled.

Metric 5: Response for class (RFC): This measure captures the size of the response set of a class. The response set of a class consists of all the methods called by local methods. RFC is the number of local methods plus the number of methods called by local methods.

Metric 6: Lack of cohesion metric (LCOM): The cohesion of a class is characterized by how closely the local methods are related to the local instance variables in the class. LCOM is defined as the number of disjoint (non-intersecting) sets of local methods.

IV. RESEARCH BACKGROUND

The main objective of this research is to investigate the general quality of OSS programs in correlation with defect density of the systems. This research mainly focuses on the application of CK OO metrics suite to OSS written in C++. For this purpose, 30 C++ programs were downloaded from SourceForge.net, which is the largest OSS repository. In order to download the systems from SourceForge, a tool, OSSGrab, was developed to automate the search and retrieval of the systems.

A. Research Questions

The research questions to be investigated are:

- 1) Which object-oriented metrics correlate with defect density in the open source systems under investigation?
- 2) What is the general quality of the open source systems written in C++?

B. Research Hypotheses

H1: The increasing value of CBO correlates with defect density.

H2: The increasing value of LCOM correlates with defect density.

H3: The increasing value of DIT correlates with defect density.

H4: The increasing value of NOC correlates with defect density.

H5: The increasing value of RFC correlates with defect density.

H6: The increasing value of WMC correlates with defect density.

C. Dependent Variable

In spite of recent advances in programming technology, it is not yet possible for developers to produce error-free code consistently. A software product is considered defective when it does not perform its functions according to the user's expectations.

In this research, defects refer to faults in the system, that later will cause failures in the system. The dependent variable used in this research is post-delivery defects collected from the bug tracking report in SourceForge.

D. Independent Variables

The independent variables selected for this research are a collection of metrics known as Chidamber and Kemerer Object-Oriented metric suite (CK OO metrics).

In order to get the metrics data, the source code of the system which have been downloaded from SourceForge was analyzed using a metric extraction tool called Understand C++. After all the metrics were obtained, they were analyzed using SPSS statistical package.

E. Data Collection

In order to ease the data collection process from SourceForge.net, a tool was developed using Python and Regular Expressions techniques, known as OSSGrab [4].

Fig. 1. OSSGrab parsing techniques.

The parsing techniques are shown in Fig. 1. The application receives a query from the user that specifies the criteria to search along with the repository. The query is then passed to the web-crawler engine that starts crawling the pages from the respective online repository's API. After loading the pages web-crawler engine hands it down to the parsing engine, which then retrieves the queried data from the mass of text. Once the parsing is done the program writes the collected data in HTML and CSV format for research use. CSV format

allows the user to further manipulate the data using rich functions of spreadsheets. Java scripts are added in the HTML to make the data more interactive and useful.

Firstly, users can choose to search the systems in SourceForge. The search process is shown in Fig. 2 and Fig. 3. Fig. 2 exhibits a simple search, where users need to specify the name of the system they want to look for. The parser will search through the SourceForge repository and will return the result to the users.

Fig. 2. OSS repository simple search.

Fig. 3 shows the advanced search option where users can select systems based on Categories, Programming Language, Development Status and Number of Downloads. The Number of Pages keyword means that the users can choose the number of systems that will be displayed on the results page, if the users choose bigger number of pages, the search time will be longer.

Fig. 3. OSS repository advanced search.

The outputs were generated in both CSV and HTML format. The users can sort the output based on the header of the column. The column Download Link will connect the users to the system download in SourceForge. This will

provide fast access to the system and the users can directly download the system. From the research point of view, this facility will provide the researchers many options of systems to choose from. In empirical software engineering, researchers need to find as many data as possible, especially when they want to build prediction models, to ensure that the models can be more generalized to the population at large.

Later, the source code of the selected systems will be downloaded and executed through a tool called Understand [18], which will produce a set of metrics i. e., CBO, WMC, DIT, LCOM, NOC and RFC. The results later are analysed using a statistical package, SPSS.

V. RESULTS DISCUSSION

The descriptive statistics for the independent variables are shown in Table I. The number of observations or sample size for this research is 30 systems. Columns 'Mean', 'Std. Dev.', 'Min', 'Max' represents the mean value, standard deviation, minimum and maximum values for each metric considered, respectively.

TABLE I: DESCRIPTIVE STATISTICS

Metrics	Mean	Std. Dev.	Min	Max
CBO	0.63	48.33	5.90	8.28
WMC	0.24	21	12.47	4.71
LCOM	14.50	367.67	54.83	60.25
DIT	0.18	24	1.55	4.26
NOC	0.00	27.33	1.31	4.92
RFC	3.33	88.65	23.48	15.67

The initial analysis on the data shows that the distribution is not normal. Therefore, for correlation analysis, Spearman correlation analysis was done. The overall results of the Spearman analysis are shown in Table I.

TABLE II: SPEARMAN CORRELATION BETWEEN VARIABLES

Metrics	Defects	CBO	RFC	NOC	WMC	LCOM
Defects	1	0.038 (0.84)	0.495 (0.005)	0.36 (0.05)	0.17 (0.369)	0.058 (0.76)
CBO	-	1	0.528 (0.003)	0.306 (0.10)	0.278 (0.137)	0.423 (0.02)
RFC	-	-	1	0.38 (0.038)	0.533 (0.002)	0.244 (0.195)
NOC	-	-	-	1	-0.305 (0.101)	-0.032 (0.867)
WMC	-	-	-	-	1	0.544 (0.002)
LCOM	-	-	-	-	-	1

In Table II, the top row represents the values for the Spearman correlation coefficient between the two variables, while the bottom row (in parenthesis) represents the p -value for the correlation. The results in Table II shows that only RFC and NOC are significant in predicting defects (Spearman corr. = 0.495, p -value = 0.005) and (Spearman corr. = 0.36, p -value = 0.05). Based on the result, we can conclude that H4 and H5 are supported. However, H1, H2, H3 and H6 are not supported by the results.

The correlation between Defect Density and CBO is depicted in Fig. 4. The data for all variables were transformed into log 10 values to normalize the values. Only WMC values were not transformed because the data are already normal.

Fig. 4. Defect density vs CBO.

Fig. 5. Defect density vs DIT.

Fig. 5 shows the relationship between Defect Density and DIT, while Fig. 6 shows the relationship between Defect Density and RFC.

Fig. 6. Defect density vs RFC.

Fig. 7 presents the relationship between Defect Density and NOC, while Fig. 8 shows the correlation between Defect Density and LCOM. Lastly, Fig. 9 depicts the relationship between Defect Density and WMC.

Fig. 7. Defect density vs NOC.

Fig. 8. Defect Density vs LCOM.

Fig. 9. Defect density vs WMC.

VI. CONCLUSION AND FUTURE WORK

The application of the object-oriented (OO) design metrics to measure the quality of open source systems has helped to understand the relationships of the metrics with defect density of the systems. This paper seeks to investigate the metrics which can be used to predict defects in OO systems, especially the ones written in C++. The results show that only RFC and NOC are significant in predicting defects.

This research can be further extended to compare the metric values for systems written in C++ and Java systems. The comparison between these languages may shed some light into the differences between both object-oriented languages.

ACKNOWLEDGMENT

This research is funded by the Ministry of Higher Education Malaysia under the Research Acculturation Grant Scheme (RAGS), grant number RAGS12-041-0041.

REFERENCES

- [1] IEEE Standard 1990, *IEEE Standard Glossary of Software Engineering Terminology*, No. STD 610-12-1990, 1990.
- [2] F. G. Wilkie and B. Hylands, "Measurement complexity in C++ application software," *Software-Practice and Experience*, vol. 28, no. 5, pp. 513–546, 1998.
- [3] Sourceforge. [Online]. Available: www.sourceforge.net.
- [4] N. S. A. A. Bakar and I. Mahmud, "OSSGrab: Software repositories and app store mining tool," *Lecture Notes in Software Engineering (LNSE)*, vol. 5, pp. 55–61, 2013.
- [5] J. S. Sherif and P. Sanderson, "Metrics for object-oriented software projects," *Journal of Systems and Software*, vol. 44, pp. 147–154, 1998.
- [6] M. H. Tang, M. H. Kao, and M. H. Chen, "An empirical study on object-oriented metrics," in *Proc. the 6th International Symposium on Software Metrics*, 1999, pp. 242–249.
- [7] F. G. Wilkie and T. J. Harmer, "Tool support for measuring complexity in heterogeneous object-oriented software," in *Proc. the International Conference on Software Maintenance (ICSM'02)*, 2002, pp. 152–161.
- [8] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *Journal of Systems and Software*, vol. 23, no. 2, pp. 111–122, 1993.
- [9] M. Lorenz, and J. Kidd, *Object-Oriented Software Metrics: A Practical Guide*, New Jersey: Prentice Hall, 1994.
- [10] K. El-Emam, S. Benlarbi, N. Goel, and S. N. Rai, "The confounding effect of class size on the validity of object-oriented metrics," *IEEE Transactions on Software Engineering*, vol. 27, no. 7, pp. 630–650, 2001.
- [11] R. Subramanyam and M. S. Krishnan, "Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects," *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 297–310, 2003.
- [12] B. M. Goel and P. K. Bhatia, "Analysis of reusability of object-oriented systems using object-oriented metrics," *ACM Sigsoft Software Engineering Notes*, vol. 38, no. 4.
- [13] U. Kumari and S. Bhasin, "Application of object-oriented metrics to C++ and Java: A comparative study," *ACM SIGSOFT Software Engineering Notes*, vol. 36, no. 2, March 2011.
- [14] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object-oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.
- [15] M. Bunge, *Treatise on Basic Philosophy: Ontology II: The World of Systems*, Riedel, 1979.
- [16] Y. Yand and R. Weber, "The ontological model of an information system," *IEEE Transactions on Software Engineering*, vol. 16, pp. 1282–1292, 1990.
- [17] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, PWS Publishing Company, Boston, 1997.
- [18] Scitools. [Online]. Available: <http://www.scitools.com/index.php>

Normi Sham Awang Abu Bakar is an assistant professor in the Department of Computer Science, International Islamic University Malaysia. She obtained her PhD in computer science at the Australian National University. Her research interests are in the area of empirical software engineering, open source quality, agile methodology, mining software repositories and software engineering education.