International Journal of Computer Applications (0975 – 8887)
Volume 28– No.8, August 2011

An Investigation of the Relationships between Software Metrics and Defects

Pradeep Singh
Deptt of Computer Sc & Engg.
National Institute of Technology, Raipur

K. D. Chaudhary
Deptt. of Electrical Engg.
National Institute of Technology, Raipur

Shrish Verma
Deptt. of I.T.
National Institute of Technology, Raipur

ABSTRACT
Open source software systems are becoming more popular today, and are playing important roles in many scientific and business software applications. Many companies are investing in open source projects and lots of them are also using such software in their own work. But, because open source software is often developed with a different management style than the industrial ones, the quality and reliability of the code needs to be investigated. Hence, more projects need to be measured to obtain information about the characteristics and nature of the source code.

In this paper we have evaluated the object-oriented metrics given by Chidamber and Kemerer, and few other static code metrics for two open source projects. We have carried out an empirical study and tried to find out the nature of relationship between these metrics and defects. In other words, it has been investigated whether these metrics are significantly associated with defects or not. For this we have extracted source code, defects processed it for metrics .Two open source java projects of emulators have been taken from source forge and used for this purpose.

Keywords
Software metrics, Defects, Open source software, Fact extraction

1. INTRODUCTION
Open source software systems are becoming more popular today, and are important components of many business software applications. Many large companies are investing in open source projects and many of them are also using this kind of software in their own work. As a consequence, many of these projects are being developed rapidly and are quickly becoming very large. But, because open source software is usually developed outside companies—mostly by volunteers—and the development methodology employed is quite different from the usual methods applied in commercial software development, the quality and reliability of the code needs to be investigated. Various kinds of code measurements can be quite helpful in obtaining information about the quality of the code.

In this paper, we describe how we extracted the relevant data from source forge, and how we calculated the object-oriented metrics suite given by Chidamber and Kemerer and some other static code metrics for the investigation of relationship between software metrics and defects. The main objective of analyzing these metrics is to improve the quality of the software. The rest of this paper is organized as follows. Section II summarizes relevant background information and related work. Section 3 provides an overview of open source projects and extraction process of metric and defect data. Section 4 discusses metrics analyzed in this paper. Section 5 deals with the results and analysis. Section 6 deals with limitations. Conclusions are given in Section 7.

2. RELATED WORK
This section briefly describes the various studies done in the field of the relationships between OO design metrics and defect. Metrics played a pivotal role in the prediction of fault proneness. El Eman et al. analyzed a large C++ telecommunication application and found that the size (i.e. SLOC) of classes confounded the effect of most OO design metrics on faults [4]. In their experiment, WMC, RFC, CBO, and LCOM were found to be significant without size control but none of these metrics was significant after controlling for the size of the system. NOC was not investigated. To further examine El Eman et al.'s findings, Subramanyam and Krisnan analyzed an e-commerce application developed in C++ and Java [2]. They performed an experiment based on 405 C++ and 301 Java classes to study the effect of the size along with the WMC, CBO, and DIT on fault-proneness of classes. Their results indicated, however, that even after controlling for the size (i.e. SLOC) of classes, some of the CK metrics were significantly associated with faults. Tang et al. [3] analyzed CK OO metrics suite on three industrial applications developed in C++. They found none of the metrics examined to be significant except RFC and WMC. Olague et al. [6] validated object oriented metrics on versions of open source agile software. They found WMC, CBO, RFC and LCOM to be very significant. The MOOD metrics is direct measure of size when used over large classes.

Briand et al. carried out an industrial case study on quality factors in OO designs [7]. They found that CBO, RFC, and LCOM were statistically related to the fault proneness of classes. Later, Birand et al. investigated the relationships based on a set of size metrics and a more complete set of OO design metrics [7]. They found that CBO, RFC, DIT, and SLOC were significant predictors of fault-proneness, and NOC was also significantly related to fault-proneness but in an inverse direction. From the review of literature, we found that earlier fault relationship analysis have only considered OO design metric and most of the studied systems were implemented in C++. Keeping these points in mind, a relationship analysis between defects in an emulator developed in java is proposed using few other static metrics other than OO design attributes. No previous work done on emulators defect data analysis.

3. MINING OBJECTIVE SOFTWARE

Our objective is to find the error prone files based on metric data, so we need metric data and bug data. In order to do this we have taken two projects whose target domain , size and the languages in which they developed are similar. So we selected following two emulator projects from sourceforge[8] developed in java. An emulator is hardware and/or software that duplicates (or emulates) the functions of a first computer system in a different second computer system, so that the behavior of the second system closely resembles the behavior of the first system. This focus on exact reproduction of external behavior is in contrast to some other forms of computer simulation, in which an abstract model of a system is being simulated.

3.1 JaC64 *:* JaC64 are completely written in Java and can be run from a modern web browser like Firefox, Internet Explorer or Netscape Navigator. A Java-based Commodore 64 emulator runs most of C64 games and many demos and emulates CPU (6510), VIC-II (on a cycle level), SID (including filters, combined waveforms and bugs) and CIA. This emulator is easy to add to any web-page using Java/JavaScript.

3.2 JMStella Atari 2600 Emulator for J2ME :

JMStella is a J2ME Atari 2600 VCS emulator based upon JStella 0.95. It allows one to play Atari 2600 games on a java-enabled mobile phone.

3.3 Metric data and bug data collection process

We selected these projects in part because they had a source code version archive (SVN), a bug database. Most of the software development system uses subversion or cvs for version control. Subversion is a free/open source version control system. i.e. Subversion manages files and directories, and the changes made to them, over time. This allows us to recover older versions of your data or examine the history of how your data changed. Subversion can operate across networks, which allows it to be used by people on different computers. At some level, the ability for various people to modify and manage the same set of data from their respective locations fosters collaboration. Some version control systems are also software configuration management (SCM) systems. These systems are specifically tailored to manage trees of source code and have many features that are specific to software development—such as natively understanding programming languages, or supplying tools for building software. Subversion is a centralized system for sharing information. At its core is a repository, which is a central store of data. The repository stores information in the form of a file system tree—a typical hierarchy of files and directories. Any numbers of clients connect to the repository and then read or write to these files. By writing data, a client makes the information available to others; by reading data, the client receives information from others.
Emulator JaC64 and JMStella source repositories were well documented, and were of modest size. Computation of the number of bugs by using SVN repositories of each project is done by identification of the project with release point. Extraction of the log data from each file (java file) in the projects is done by using log subcommand of svn command. Finally searching the word (bug or fixed) and counted the frequency of appearances. For the java project we computed the bug count per file and assigned that bug count to the public class. To be precise we computed the number of bugs fixed and not all other bugs. The metrics data is computed based on the source files of one release and the bug data is computed based on the log data. Our objective is to find the error prone files based on metric data, so we need metric data and bug data. Source code of both projects obtained from sourceforge.net. The metric data needs to be computed based on the predefined methods, so we used the metric tool family [9].These tools compute 63 metrics and we selected the eleven metrics including C&K object oriented metrics [1].

4. METRICS ANALYSED IN THIS STUDY

In this section, we define the eleven metrics that we extracted from the source code of emulators and used for relationship analysis. Six of these metrics were first presented by Chidamber and Kemerer [1]. We also used other metrics such as Lorenz & Kidd object-oriented metric named class variables and added few more Base class , Class Methods and the well-known lines of code metric Lines with Comments (CLOC) , Source Lines of Code (SLOC) because we were also interested in comparing object-oriented metrics with the traditional code-size metric. The details of metrics we investigated were the following:

4.1 C.K. Metrics Model

Six metrics were first presented by Chidamber and Kemerer [1], these metric suites offers informative insight into when developers are following object oriented principles in their design. CK metrics have generated a significant amount of interest and are adopted by practitioners [7] and are also being incorporated into industrial software development tools such as Rational Rose and Together. Chidamber and Kemerer proposed six metrics; the following discussion shows their metrics.

4.1.1 Weighted Method per Class (WMC)

WMC measures the complexity of a class. The WMC is the number of methods defined in each class. More precisely, this is a weighted sum of all the methods defined in a class. High value of WMC indicates the class is more complex than that of low values. So class with less WMC is better.

4.1.2 Depth of Inheritance Tree (DIT)

The depth of a class within the inheritance hierarchy is the maximum number of steps from the class node to the root of the tree and is measured by the number of ancestor classes So this metric calculates how far down a class is declared in the inheritance hierarchy. This metric also measures how many ancestor classes can potentially affect this class. DIT represents the complexity of the behavior of a class, the complexity of design of a class and potential reuse. If DIT increases, it means that more methods are to be expected to be inherited, which makes it more difficult to calculate a class's behavior. Thus it can be hard to understand a system with many inheritance layers. On the other hand, a large DIT value indicates that many methods might be reused.

4.1.3 Number of children (NOC)

This metric measures how many sub-classes are going to inherit the methods of the parent class. The size of NOC approximately indicates the level of reuse in an application. If NOC grows it means reuse increases. On the other hand, as NOC increases, the amount of testing will also increase because more children in a class indicate more responsibility. So, NOC represents the effort required to test the class and reuse.

4.1.4 *Coupling between objects (CBO)*
A class is coupled to another one if it uses its member functions and/or instance variables. The CBO gives the number of classes to which a given class is coupled. An increase of CBO indicates the reusability of a class will decrease. Thus, the CBO values for each class should be kept as low as possible.

4.1.5 *Response for a Class (RFC)*
This is the number of methods that can potentially be executed in response to a message being received by an object of that class.

4.1.6 *Lack of Cohesion in Methods (LCOM)*
This metric uses the notion of degree of similarity of methods. LCOM measures the amount of cohesiveness present, how well a system has been designed and how complex a class is . This is the number of pairs of member functions without shared instance variables, minus the number of pairs of member functions with shared instance variables

4.1.7 *OTHER METRICS*
Source Lines of Code (LOC): The SLOC of a class is the number of all nonempty, non comment lines of the body of the class and all of its methods.

Base Class: Number of Immediate Base Classes.
Class Methods: Number of Class Methods.
Lorenz & Kidd's Class Variables: Number of Class Variables.
Lines with Comments (CLOC): Number of lines containing comment.

Table 1 and 2 shows brief statistics of various metrics of emulator projects jac64 and table 3 and 4 shows brief statistics of various metrics of project jmstella.

TABLE 1 Descriptive Statistics of the Classes jac64

	CBO	NOC	WMC	RFC	DIT	LCOM
Max	64.00	2.00	48.00	645.00	5.00	100.00
Min	0.00	0.00	0.00	12.00	1.00	0.00
Mean	8.114	0.045	8.898	82.807	1.534	47.227
St. dev	10.863	0.259	10.654	178.306	1.072	39.251

TABLE 2 Descriptive Statistics of the Classesjac64

	Base Classes	Class Methods	Class Variables	CLOC	SLOC
Max	4.00	9.00	108.00	566.00	1566.00
Min	1.00	0.00	0.00	0.00	4.00
Mean	1.511	0.386	5.634	40.716	174.625
St. dev	0.711	1.208	13.616	104.428	278.189

TABLE 3 Descriptive Statistics of the Classes jmstella

	CBO	NOC	WMC	RFC	DIT	LCOM
Max	21.00	0.00	74.00	86.00	3.00	99.00
Min	0.00	0.00	0.00	12.00	1.00	0.00
Mean	4.309	0.00	13.255	37.836	1.455	59.145
St. dev	5.314	0.00	15.577	23.40	0.633	39.117

TABLE 4 Descriptive Statistics of the Classes jmstella

	Base Classes	Class Methods	Class Variables	CLOC	SLOC
Max	3.00	13.00	128.00	524.00	1833.00
Min	1.00	0.00	0.00	0.00	4.00
Mean	1.291	0.945	6.291	58.236	156.382
St. dev	0.567	2.592	18.357	111.785	320.417

5. ANALYSIS
To analyze the relationship between defect and metrics chosen for this work, we have chosen Spearman rank correlation analysis. The Spearman rank correlation is a commonly-used robust correlation technique [5] because it can be applied even when the association between elements is non-linear. The resulting standard Spearman correlation coefficients are shown in Table 5 and table 6.

We computed spearman's rank correlation and the statistical significance of correlation. Presented metrics are significant at p value < 0.001 using rank correlation analysis i.e. associated metrics are correlated with number of defects. We found that in object oriented design metrics NOC and DIT for both projects are highly significant.

TABLE 5 RESULT OF SPEARMAN'S TEST OF LINEAR CORRELATION

Project Name	CBO	NOC	WMC	RFC	DIT	LCOM
Jmstella	0.569	0.947	0.554	0.527	0.539	0.544
Jac64	0.527	0.787	0.579	0.54	0.655	0.576

TABLE 6 RESULT OF SPEARMAN'S TEST OF LINEAR CORRELATION

Project Name	Base Classes	Class Methods	Class Variables	CLOC	SLOC
Jmstella	0.7	0.721	0.49	0.545	0.543
Jac64	0.43	0.657	0.684	0.581	0.584

The following observations are made based on the results given in Tables 5 and 6.

CBO metrics that count the both import and export coupling are related to defects for both emulator projects. Metric NOC counting number of children of a class is highly related to defects. We found that NOC (Number of Children) was very significant. In accordance with findings of [11] NOC was not significant .Basili et al. [13] also found NOC to be very significant and they noticed that, the larger the value of NOC, the lower the probability of fault detection. For our analysis of project Jmstella it is 0.947 and for jac64 it is 0.787 significant (p-value <0.001).

WMC (Weighted Methods per Class) was found to be significant in our analyses. On the other hand, Basili et al. [13] found it less significant, but, for extensive modified classes and for UI classes, it was more significant. In the study by Yu et al. [14], WMC was found significant which is similar to our results. Subramanyan and Krishnan [2] found WMC to be significant for C++ (but not significant for Java). All of our two analysis of WMC yielded the same result; hence, we accepted WMC having significant relation with defect.

DIT (Depth of Inheritance Tree) was found to be significant in our analysis .This finding is similar to those given by Basiliet al. [13].

We found RFC (Response for a Class) to be significant—which is the same as the result of Basili et al. [13]. Yu et al. [14] found RFC to be significant as well but they calculated the RFC in a different way. We examined two RFCs in the case of emulators and we found high and significant (p-value < 0.001) correlation between defects.

LCOM show positive coefficients. This indicates that the probability of defect increases as the cohesion of a class decreases. The results indicate that inheritance metric DIT measuring depth of inheritance tree is positively related to faults SLOC (Source Lines of Code) was found to be significant in our study. Subramanyan and Krishnan [2] also found that LOC was significant, but neither Basili et al. [13] nor Yu et al. [14] examined this metric. Class method was found significant for the both of the projects in our study but neither Basili et al. [13] nor Yu et al. [14] examined this metric.

6. THREATS TO VALIDITY

The study has a number of limitations that are not unique to our study but are common with most of the empirical relationship analysis in the literature. The degree to which the results of our study can be generalized to other research settings is questionable. The reason is that the systems developed are small-sized. We identified defect fixes from the SVN logs entered by developers. Surely, some potential issues can be raised, such as some defects may not surface, some defects may surface but not get fixed, and some defect fixes may not be recorded in SVN logs. Still, in many studies of software quality, defect fix data have served useful purposes. Since we know that a large community of users and developers used the open-source products studied here, we are confident that the defects were revealed and fixed adequately. In this study the severity of faults is not taken into account. There can be number of faults which can leave the system in various states e.g. a failure that is caused by a fault may lead to a system crash or an inability to open a file. The former failure is more severe than latter, thus the types of fault is not the same. Though these results provide guidance for future research on the impact of OO metrics on fault proneness, further validations are needed with different systems to draw stronger conclusions.

7. CONCLUSIONS

We presented a method and process with which metrics (and also other data) can be automatically calculated from the source code of real-size software. By processing the data from source forge, we associated the bugs with classes found in the source code. We employed statistical methods to assess the applicability of the well-known object-oriented metrics to the number of bugs in classes. We have conducted an empirical validation of eleven metrics. The systems under study are medium sized systems written in Java and have a testing record including number of faults found in each class. In this study we found NOC is highly correlated with defects. The NOC metric seems to be the best in this analysis . The SLOC, CLOC and Class methods metric performed fairly well and, they can be easily calculated. More similar type of studies must be carried out with large data sets to get an accurate measure of performance. We plan to replicate our study on large data set for different types of open source software system.

8. REFERENCES

[1] Chidamber, Shyam , Kemerer, Chris F. "A Metrics Suite for Object- Oriented Design." M.I.T. Sloan School of Management E53-315, 1993

[2] R. Subramanyan and M.S. Krisnan, "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects," IEEE Trans. Software Eng., vol. 29, no. 4, pp 297-310, Apr. 2003.

[3] Tang, M. H., Kao, M. H., & Chen, M. H. An empirical study on object-oriented metrics. In Proceedings of 6th IEEE International Symposium on Software Metrics. 1999, pp.242–249

[4] 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 Trans. Software Eng., vol. 27, no. 7, pp. 630-650, July 2001

[5] Fenton, N., S.L. Pfleeger: "Software Metrics: A Rigorous and Practical Approach", PWS Publishing Co.

[6] Olague, H., Etzkorn, L., Gholston, S., & Quattlebaum, S. Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. IEEE Transactions on Software Engineering, 33(8), 2007, pp.402–419.

[7] L.C. Briand, J. Wu¨ st, J.W. Daly, and D.V. Porter, "Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems," J. Systems and Software, vol. 51, no. 3,pp. 245-273, 2000

[8] JaC64 and JMStella Atari 2600 Emulator for J2ME, available at www.sourceforge.net/projects/

[9] Scitools. http://www.scitools.com/index.php.

[10] K..El Emam, S. Benlarbi, N.Goel , S. Rai, "A Validation of Object-Oriented Metrics", Technical Report ERB-1063, National Research Council of Canada *(NR*C), 1999.

[11] T. Gyimothy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open software for fault prediction. IEEE Trans. on Soft. Eng., 31(10):897–910, 2005.

[12] S. Watanabe, H. Kaiya, K. Kaijiri, Adapting a Fault Prediction Model to Allow Inter Language Reuse, PROMISE'08, May 12-13, Leipzig, Germany, 2008Nov. 1999.

[13] V.R. Basili, L.C. Briand, and W.L. Melo, "A Validation of Object- Oriented Design Metrics as Quality Indicators," IEEE Trans. Software Eng., vol. 22, no. 10, pp. 751-761, Oct. 1996.

[14] P. Yu, T. Systa¨ , and H. Mu¨ ller, "Predicting Fault-Proneness Using OO Metrics: An Industrial Case Study," Proc. Sixth European Conf. Software Maintenance and Reeng. (CSMR 2002), pp. 99-107, Mar. 2002.