Validation of Object Oriented Metrics Using Open Source Software System: An Empirical Study

Kalpana Johari
Centre for Development of Advanced Computing,
CDAC, NOIDA
NOIDA, India
kalpanajohari@cdacnoida.in

Arvinder Kaur
Guru Gobind Singh Indraprastha University
Sector-16 Dwarka
Delhi, India
arvinderkaurtakkar@yahoo.com

ABSTRACT
In today's environment the relevance of Free Open Source Software Systems is understood and appreciated both in academia and research. The debate about the pros and cons of the open source vis-à-vis proprietary software has been raging from ages ever since Richard Stallman founded the Free Software Foundation in 1985. With the changing trends in the domain of Object Oriented Systems there is a need to measure the fault predictability of software metrics on open source software systems. In this paper we present the results of empirical study which was conducted using open source software, JHotDraw 7.5.1. We computed the object oriented metrics, proposed by Chidamber and Kemmerer, and performed bug- class mapping for the software under study. We also studied the relationship between the revisions made to open source software and its software metrics measure.

Categories and Subject Descriptors
D.2.8 [**Software Engineering**]: Metrics – *Complexity measures, Performance measures, Process metrics, Product metrics, Software science*.

General Terms
Measurement, Verification.

Keywords
object oriented metrics, open source software system, bug mapping, empirical study, revision.

1. INTRODUCTION
The importance of open source software systems has been felt both in software industry and research. Large number of software is being developed using open source tools and more companies are investing in open source software. Much of the research is being done on or using open source software systems because such software is not monopolized and they are free from licensing issues. Since open source software are developed following a style different form the conventional one, there arises a need to measure the quality of open source software.

In this paper we are presenting the empirical study of open source software system namely JhotDraw 7.5.1. The study was conducted to validate the effectiveness of object oriented metrics suite, proposed by Chidamber and Kemerer [6], in detecting the fault proneness in open source software systems. Similar studies, meant to validate the metrics, have been conducted by many researchers [1][2][3][4][5][7]. We understand that studies conducted on wide variety of software are important for providing the industrial acceptability to software metrics [8].

We have computed class level object oriented metrics using ckjm-1.9 : A tool for calculating Chidamber and Kemerer Java Metrics [14] and Metrics 1.3.6 [15], an eclipse plugin. A SCM (software configuration management) repository [13] maintains the details of revisions made to open source software. We studied the description for each revision, identified with a specific id, and filtered out those revisions that were made as a result of bug fixture. Each revision contains the details of classes being affected it. These were the classes that contained fault. Using the said approach we mapped bugs with classes that were modified due to bug fixture. The main contributions of this paper are as follows. Firstly, since our analysis is based on medium level open source software (JhotDraw 7.5.1) that has been used in number of research [10][11], the results will have better acceptability. Moreover due to the usage of open source, the details can be published. Secondly, we have also studied the relationship between the number of revision made to a class and the measure of software metrics suite. Software may be revised in response to post release bug, a feature request, for preventive activity or during the process of adapting the software to some new environment, and therefore, such study of number of revisions and the measure of software metrics can be used to predict the maintainability of the software.

The paper is organized as follows. Section 2 presents a brief overview of software metrics used in the study and describes the hypothesis formulation. Section 3 gives an introduction to open source software. Section 4 presents the analysis and results and the last section presents conclusions and future scope.

2. OVERVIEW OF SOFTWARE METRICS USED IN THE STUDY
We have applied class level metrics on open source software. This section presents the definition of seven metrics used in the study. Of the seven, six metrics are the one that were originally proposed by Chidamber and Kemerer (C&K). We have used two different variants of weighted method per class(WMC) metrics. One that simply counts the number of methods in the class and the other one that counts the sum of cyclomatic complexity of each method of a class. We have also used token count of a class as a size metrics. This will help us in comparing object oriented metrics with traditional size oriented metrics.

CBO - Coupling between object classes

The CBO metric for a class is the count of all those classes with which the given class is coupled. Two classes may be coupled due to method call, arguments, return type, field access, inheritance and exception.

DIT - Depth of Inheritance Tree

The (DIT) metric of a class is the measure of its inheritance level from the top of hierarchy. In Java every class inherits from Object class therefore the minimum DIT for a class is 1.

LCOM - Lack of cohesion in methods

A LCOM metric of a class is the count of set of methods in a class that are disjoint with respect members of a class being accessed by them. The original definition of this metric as presented in [6](which is the one used in ckjm) considers all pairs of a class's methods.

NOC - Number of Children

NOC metrics measure the number of direct descendents of a class.

RFC- Response for a Class

The RFC metrics of a class is the measure of number of methods that can be invoked in response to a message received by an object of the class. Ideally RFC should measure the transitive closure of the call graph for each method. The ckjm tool measures the count of methods being called by the method of a class under study.

WMC - Weighted methods per class

WMC metric for a class is the sum of complexities of its methods. The complexity of an individual method can be measured as cyclomatic complexity or simply we can assign 1 as the complexity value .The ckjm tool assigns 1 as the complexity measure to each method and

DOI: 10.1145/2088883.2088893 http://doi.acm.org/10.1145/2088883.2088893

therefore the WMC for any class is equal to number of methods in that class. The cyclomatic complexity variant of WMC (WMC using CC) has also been used in the study. WMC using CC was calculated using metric 1.3.6 [15].

Token Count- The token count metrics is the measure of number of tokens in a class. For example, in java an import statement in class definition (import java.util.Date;) contains 3 tokens. The token count does not include comments but it includes annotations.

2.1 Hypothesis Formulation

We drew up following 7 hypotheses to be tested in order to establish the relationship between object oriented metrics(OOM) & fault proneness of classes and OOM and revision count of a class. Similar hypothesis were also drawn by Basili et al.[1] and Gyimo´thy et al. [7].

H1(CBO): A class that is highly coupled is more prone to errors than it peer and may require more number of revisions.

H0(CBO): Coupling of class does not contribute to its fault proneness and require no more revisions than its peer.

H1(DIT): Classes lower in class hierarchy are more error prone and need to be revised more often than the one that are located up in the hierarchy.

H0(DIT): Location of a class in class hierarchy does not affect number of revisions and its fault proneness.

H1(LCOM): The class which shows poor cohesion are more fault prone and require more revisions than its peer.

H0(LCOM): The classes with high value of lack of cohesion are no more fault prone and does not require more revisions than the peer.

H1(NOC): A class with more children is less error prone than its peer. This is mainly because a class with many child classes has to be very simple and flexible in order to allow many different implementations thus making it less fault prone and requiring fewer revisions. Our hypothesis regarding NOC is similar to Gyimo´thy but is different from the one drawn by Basili.

H0(NOC): The number of children for a class does not affect the fault proneness of the class. This means that a class with any number of children is as fault prone as its peer and requires as many revisions as its peer.

H1(RFC): A class with high value of RFC is more fault prone and need to be revised more often than its peer.

H0(RFC): A class with high value of RFC is no more fault prone and requires no more revisions than its peer.

H1(WMC): A class with more number of methods requires more number of revisions and more fault prone.

H0(WMC): A class with high WMC is no more fault prone and does require as many revisions as any peer class.

H1(Token count): A class with large number of token is more fault prone that its peer. This mainly due to the fact that large size makes a class complex, requiring more revisions and the fault proneness of a class may increase.

H0(Token count): A class with large token count is no more fault prone and requires no more revisions than its peer.

3. INTRODUCTION TO CASE STUDY

In this section, we give a summary of software that we have investigated, namely JhotDraw 7.5.1[12]. The software studied had the following characteristics:

- Open source: in order to ensure availability of source code, revision history and bug details.
- Implemented in java: Limitation of metrics measuring tools.

3.1 Case study: JhotDraw 7.5.1 [9]

JhotDraw 7 is a framework for 2-dimensional drawing editors and for document-oriented applications. It is implemented in java. It is based on Erich Gamma's JhotDraw which is an adaption of HotDraw. JhotDraw 7 is a result of major revisions applied to previous versions of JhotDraw. It was developed by Werner Randelshofer. So far about 7 different versions of JhotDraw7 has been released. The latest version is JhotDraw7.6.1. Our case study is based on JhotDraw 7.5.1. The latest version with availability of all the revisions is the reason behind choosing the 7.5.1 version of JhotDraw 7. Table I provides the details of JhotDraw 7.5.1. There are 613 classes and interfaces. The source code and the revision history for JhotDraw 7.5.1 is available on sourceforge.net. Each revision is identified by an id, contains the description about the revision, revision date and it also contains the details about the changes made during the revision. One revision may affect more than one class. We considered revisions that were made after the release of JhotDraw 7.5.1 and before the release of JhotDraw7.6.1. We assume that these revisions were made to JhotDraw 7.5.1. There were 47 such revisions. The bug report for the software is also available on sourceforge.net. Bug details are maintained using bug tracker system. Each bug is assigned an id. The bug report date and a brief description about the bug are also maintained. The bugs that were reported before the release of JhotDraw 7.5.1 and fixed after the release date are the one that existed in the software being investigated. For bug mapping, we studied the revisions made after the release of JhotDraw7.5.1 and out of the total 59 revisions we identified 42 revisions as the bug fixtures. Table II presents the bug distribution.

Table 1. Summary of JhotDraw

JHotDraw 7	
Size (KLOC) approx	52KLOC
Language	Java
Version control tool	SVN
Version	7.5.1
Number of classes	613
No. of revisions before the next release	57
Number of bugs mapped	42

Table 2. Bug Distribution

	JHotDraw7.5.1	
Number of Bugs	**Number of classes**	**%**
0	250	40.72%
1	236	38.44%
2	92	14.98%
3-5	35	5.70%

4. DATA ANALYSIS

This section presents an empirical assessment of object oriented metrics in predicting the revision count and fault proneness of classes. Regression analyses being widely used technique, we have used it to measure dependent variable on the basis of independent variables. We employed linear regression analysis approach. This analysis technique has been used in [7]. This section consists of two subsections: the first one gives the descriptive statistics of software studied and the second section presents the results of univariate linear regression analysis.

4.1 Descriptive Statistics and Correlation Analysis

Table III presents the descriptive statistics of software metrics calculated for an open source software. The results indicate that the usage of inheritance in the software was low. Number of children per class was also observed to be very few. The results are similar to the one found in [1][2][7]. For LCOM, the maximum value is generally high. This is mainly due to the fact the LCOM is square of the number of methods in the class. The same observation has been noted in [7]. The token count metrics gives the measure of size and the size varies over a wide range.

We applied correlation analysis to compute coefficient of correlation(r) between software metrics measure and bug count per class as given in table IV. This is a commonly used statistical measure to determine dependency of dependent variable on explanatory variables. The technique has been used in number of research. The value of r ranges between -1 and +1. The value of r closer to 1 indicates a positive correlation whereas value of r closer to -1 indicates negative correlation. Correlation coefficient with 0 values indicates no correlation between the variant. That is the two variants are independent of each other. The

coefficient of correlation cannot provide the prediction of bug count in the class. For that we need to do regression analysis. The results show a strong correlation between bug count and some of the metrics. WMC and WMC(CC) showed almost same correlation with the bug count. A higher value token count shows the relationship between bug count and size. Similar results for size has been observed by [7][9]. The results of correlation analysis also indicate that the bug count per class is not dependent on all the metrics of object oriented metrics suite. The bug count displayed strong positive correlation with WMC (Both) and RFC. The results also show that bug count is independent of DIT and NOC and is moderately dependent of CBO and LCOM.

We also computed correlation coefficient for software metrics and count of revision per class, where measure of metrics was independent variable and count of revision per class was dependent variable. The results showed similar values as the one observed for bug count. Considering the correlation results presented in table IV, we can conclude that number of revision per class can be predicted using metrics suite. A very high value of r for bug count and revision count, as mentioned in table V also supports the same. Since revisions are made in response to any maintenance related activity. The software metrics can predict the maintenance effort required for the software system.

4.2 Regression Analyses

This section describes the analyses we performed to find out the relationship between measure of metrics and bug count. We also performed regression analysis to find out relationship between measure of software metrics and revision count. We used linear regression analysis to assess the validity of software metrics as predictor of number of faults in the class. We applied univariate linear regression technique. Univariate regression analyses establish the relationship between dependent variable (bug count and revision count) and explanatory variable (software metrics). Table VI shows the results of univariate linear regression analysis for bug count of JhotDraw and table VII shows the result for univariate linear regression analysis revision count.

Table 3. Descriptive Statistics for JhotDraw

	CBO	DIT	LCOM	NOC	RFC	WMC	WMC (CC)	Token count
Mean	6.18	1.23	97.18	.31	36.67	11.72	21.03	795.56
Median	4	1	15	0	24	8	13	453
tdDev	7.61	1.61	277.30	1.27	36.55	11.65	31.3	1176.62
Max	66	7	3602	14	285	92	461	16309
Min	0	0	0	0	0	0	0	9

Table 4. Result of Correlation analysis between revision count and bug count

	Revision Count
Bug Count	0.8809

We also computed correlation coefficient for software metrics and count of revision per class, where measure of metrics was independent variable and count of revision per class was dependent variable. The results showed similar values as the one observed for bug count. Considering the correlation results presented in table IV, we can conclude that number of revision per class can be predicted using metrics suite. A very high value of r for bug count and revision count, as mentioned in table V also supports the same. Since revisions are made in response to any maintenance related activity. The software metrics can predict the maintenance effort required for the software system. This needs to be assessed empirically.

4.3 Validation of Hypothesis

The results of regression analysis clearly indicate that the null hypothesis pertaining to WMC, token count, RFC and CBO can be rejected. The study indicate high significance of WMC, Token count, RFC and CBO in predicting the fault proneness and number of revisions made to a class.

The results related to NOC shows no significance of number of children in predicting revision count and fault proneness. Therefore we can reject the null hypothesis and accept the alternate hypothesis.

The regression analysis for DIT (Depth of Inheritance) and LCOM showed least significance in predicting the revision count and bug count. The results are similar to the one observed in [7].

5. CONCLUSION AND FUTURE WORK

We presented the empirically study to assess the validity of object oriented metrics in predicting the revision count and fault proneness of a class. The major contributions of this study are:

The study shows the applicability of object oriented metrics in predicting the number of revisions that a class under goes. Since the revision made to a class are a part of maintenance activity. This study can be used as a motivating factor for exploring the applicability of software metrics in estimation of maintenance effort.

Since the study is conducted using open source tools applied on open source software system, it will help in supporting the industrial acceptability of software metrics.

We used an open source case study therefore the data and results can be easily distributed. The applicability of software metrics in predicting the revision count per class need to empirically established using more case studies. We are currently focusing on study of more open source software system and shall formulate a model for predicting the maintenance effort required in open source software.

Table 5. Results Of Correlation Analysis For Software Metrics & Bug Count And Software Metrics & Revision Count

	CBO	DIT	LCOM	NOC	RFC	WMC	WMC(CC)	Token count
Bug count	0.2409	-0.0089	0.2359	0.0654	0.3641	0.3186	0.3037	0.3218
Revision count	0.2948	-0.0127	0.2370	0.0639	0.3902	0.2988	0.3211	0.3314

Table 6. Results For Univariate Linear Regression Analysis Bug Count Of Jhot Draw 7.5.1

	CBO	DIT	LCOM	NOC	RFC	WMC	WMC(CC)	Token count
Intercept	0.69434	0.88579	0.80110	0.86399	0.53393	0.57633	0.83621	0.67317
coefficient	0.02992	-0.00526	0.00080	0.04829	0.00942	0.02584	0.03152	0.05912
R-Squared	0.0580	0.000	0.0556	0.0043	0.1326	0.1015	0.0922	0.1036
p-value	0.0000	0.0000	0.0000	0.0632	0.0001	0.0000	0.0000	0.0000

DOI: 10.1145/2088883.2088893 http://doi.acm.org/10.1145/2088883.2088893

Table 7. Results For Univariate Linear Regression Analysis Revision Count Of Jhot Draw 7.5.1

	CBO	DIT	LCOM	NOC	RFC	WMC	WMC(CC)	Token count
Intercept	0.59834	0.93162	1.00763	0.60210	0.70336	0.54831	0.8692	0.77213
coefficient	0.08326	-0.00381	0.00731	0.02185	0.02769	0.01451	0.03872	0.04011
R-Squared	0.0869	0.0002	0.0562	0.0041	0.1523	0.0891	0.1031	0.1098
p-value	0.0000	0.0000	0.0000	0.0756	0.0001	0.0000	0.0000	0.0000

6. REFERENCES

[1] V. R. Basili, L. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," IEEE Transactions on Software Engineering, vol. 22, pp. 751–761, 1996.

[2] R. Subramanyam, M.S. Krishnan, "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects," IEEE Transaction on Software Engineering, vol. 29, no. 4, pp. 297-310, 2003.

[3] S.R. Chidamber, D.P. Darcy, C.F. Kemerer, " Managerial Use of Metrics for Object Oriented Software: An Exploratory Analysis," IEEE Transacion on Software Engineering, vol. 24 no. 8, pp. 629-639, 1998.

[4] L.C. Briand, W.L. Melo, J. Wu¨st, "Assessing the Applicability of Fault-Proneness Models Across Object-Oriented Software Projects," IEEE Transaction on Software Engineering, vol. 28 no. 7, pp. 706-720, 2002.

[5] H.M. Olague, L.H. Etzkorn ,S. Gholston, S. Quattlebaum, "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 Transaction on Software Engineering, vol. 33 no. 6, pp. 402-419, 2007.

[6] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," IEEE Trans. Software Eng., vol. 20, no. 6, pp. 476–493, 1994.

[7] T. Gyimo´thy, R. Ferenc, I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," IEEE Transaction on Software Engineering, vol. 31 no. 10, pp. 897-910, 2005.

[8] T. Zimmermann, R. Premraj, A. Zellar, "Predicting Defects for Eclipse," In: 3rd International Workshop on Predictor Models in Software Engineering, PROMISE'07. 2007.

[9] N.E. Fenton, M. Neil, "Software Metrics: Successes, failures and new directions," Journal of System and Software, vol. 47, issue 2-3, pp. 149-157, 1999.

[10] M. Marian, L. Moonen, A. van Deursen, "A Classification of Crosscutting Concern," In: 21st IEEE International Conference on Software Maintenance, ICSM'05, pp. 673-676, 2005.

[11] G. Canfora, L. Cerulo, M. Di Penta, "On the use of Line Co-change for Idenifying Crosscutting Concern Code," In: 22nd IEEE International Conference on Software Maintenance, ICSM'06, pp. 213-222, 2006.

[12] JhotDraw 7, http://www.randelshofer.ch/oop/JhotDraw/Documentation/index.html

[13] Open source software, www.sourceforge.net

[14] Ckjm-1.9, http://www.spinellis.gr/sw/ckjm/doc/index.html

[15] Metrics 1.3.6, http://metrics.sourceforge.net/

DOI: 10.1145/2088883.2088893 http://doi.acm.org/10.1145/2088883.2088893