Investigating the Strength of Relationships among Software Metrics

Mandeep K. Chawla
Assistant Professor
Department of Computer Science
MCM DAV College for Women, Chandigarh.

Indu Chhabra, Ph.D
Associate Professor
Department of Computer Science
Panjab University, Chandigarh.

ABSTRACT
Software metrics provide desirable means to measure design traits of an application under development as well as quality of end product. These are beneficial at various stages to enhance developer productivity and to make the software more manageable post-deployment. Investigating the strength of relationships among these metrics can offer more meaningful insights than analyzing them in isolation. This paper carries out a case study on an open source java based web server to identify correlations between several metrics from well known OO metrics suites. Quantitative distributions of classes over different metric values have also been observed. Results have been compared with similar past studies to verify the findings.

Keywords
Software Quality, CK metrics, Apache Tomcat, ckjm, data distribution, correlation.

1. INTRODUCTION
Incorporating software metrics for measurement purposes promise better supervision and handling of project complexities, and a potential improvement in target product quality. Projects are measured so as to quantify structural properties of source code and to recognize interdependencies of attributes of interest. 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." According to authors of CK metrics suite [2], these measures can identify areas of application which may require more rigorous testing, potential flaws, and areas that are candidates for redesign. However, for enhanced understanding of these quality determinants, investigating the relationship among them can offer more meaningful insights than analyzing them in isolation. We calculate correlations among each set of metrics chosen to determine which pairs hold strong connections than others, indicating positive or negative influence on each other. Also, quantitative distributions of classes over different metrics' ranges have been established to observe the statistical trends of each metric domain.
The objects of this study are source code of an open source popular web server and, some software metrics from CK suite [2] and a few others which are computed through a freeware robust metric calculation tool. Rest of the paper is organized as follows. Section 2 discusses prior research and related work in OO metrics. Section 3 identifies the tool, software to be analyzed and metrics under consideration. Section 4 implements the chosen tool on selected software to compute metrics, results are produced and correlation coefficients are calculated for each pair of metrics. Section 5 deals with analysis and interpretation of results. Section 6 pinpoints the threats to validity of results. Concluding remarks are given in Section 7.

2. RELATED WORK
Evaluating internal structure of the products through metrics provides objective and economical solution to quality assessment. Researchers have used them widely to gain clarity in software design and to attain stability and maturity in software engineering processes. Nagappan [3] empirically studied five Microsoft systems to find that failure prone software entities are statistically correlated with code complexity measures. Jiang [4] found that source code metrics perform better than design level metrics and combination of them performs the best. Y Ma [5] proposed a hybrid set of complexity metrics for large scale object oriented systems. Authors [6] suggested new software metrics based on coding standards violations to capture latent faults in a development. Authors [7] carried out an experiment to evaluate the practical use of the proposed thresholds for some OO metrics including DIT, coupling, NPM etc. A. Meneely et al. [8] provided a comparative analysis of the metrics validation criteria found in the academic literature. Authors [9] performed measurements on the source code using CK and LOC metrics which are proven to be correlated with software quality for various phases of an agile project. Based on search-based refactoring [10], five cohesion metrics were assessed to explore relationships between them using java systems. In their analytical study [11], managers rated data and metrics as the most important factor to their decision making and developers are also more interested in source code metrics. In their Industrial Experience Report [12], usefulness of two architecture level metrics were evaluated to quantify the analyzability and encapsulation within a software system. In a comprehensive literature review [13], OO and process metrics were reported to be more successful in finding faults compared to traditional size and complexity metrics. A set of extended OO metrics were proposed in [14] to quantify and measure difficulty in implementing changes during maintenance, as well as the possible effects.

Table 1. Description of metrics under consideration according to CKJM[2]

Metric	Full Name	Category of Metric-suite	Description
WMC	Weighted methods per class	CK [2]	It is equal to the number of methods in the class, assuming the complexity value of 1 for each method.
DIT	Depth of Inheritance Tree	CK [2]	It provides for each class a measure of the inheritance levels from the object hierarchy top.
CBO	Coupling between object classes	CK [2]	It represents the number of classes coupled to a given class.
RFC	Response for a Class	CK [2]	It measures the number of different methods that can be executed when an object of that class receives a message.
LOC	Lines of Code	Size metric	It is the sum of number of fields, number of methods and number of instructions in every method of given class.
MFA	Measure of Functional Abstraction	QMOOD [16]	This metric is the ratio of the number of methods inherited by a class to the total number of methods accessible by member methods of the class.
IC	Inheritance Coupling	Coupling metric	This metric provides the number of parent classes to which a given class is coupled.
CBM	Coupling Between Methods	Coupling metric	The metric measure the total number of new/redefined methods to which all the inherited methods are coupled.

3. SELECTION OF DATA SOURCE, TOOL AND METRICS ANALYZED

In this paper, the measurements and observations have been conducted on the source code of Apache Tomcat[1] web server (version 7.0.39). It is an open source software implementation of the Java Servlet and Java Server Pages technologies, and coded in pure java. Apache Tomcat powers numerous large-scale, mission-critical web applications across a diverse range of industries and organizations.
To calculate the metric values for the software selected, CKJM-extended 2.0[2] has been used which is an enhanced version of the original ckjm tool [15] for calculating CK and many other metrics. It is freely downloadable command line tool which processes the bytecode of compiled Java files and delivers the results in plain text or .XML form. Optionally, an XSL transformation may be used to convert the output to html. Since it is further needed to analyze the outcomes of the tool, hence in this case, metric values would be captured in XML file. Moreover, for the tool, a command line batch script had to be written to specify all the directories containing .class files to be measured.

The chosen tool is capable of computing 19 size and structure metrics for each class; however following 8 metrics have been shortlisted for this study. Table 1 records a brief description of them, as given in tool's manual. Since the tool takes java bytecode as input, it is one of the prerequisite that the project compiles successfully to be able to start analysis. To meet this requirement, Apache Tomcat source code was compiled with Apache Ant[3] (version 1.9.0) which is a Java library and command-line utility that helps building software.

4. SOFTWARE MEASUREMENTS

Once this comprehensive groundwork is accomplished, the batch script is invoked from command line to run CKJM for Tomcat. Metrics are computed and collected in XML format for all java classes. At this point, it is straightforward to import the XML file in any spreadsheet program to make it ready for further examination. Table 2 shows the descriptive statistics for metrics under consideration.

Table 2. Descriptive statistics of metrics

Metric	MIN	MAX	AVG	STD-DEV
WMC	0	319	9.32	17.04
DIT	1	6	1.77	1.01
CBO	0	72	3.34	5.94
RFC	0	639	22.86	39.04
LOC	0	8986	206.50	521.45
MFA	0	1	0.352	0.41
IC	0	4	0.325	0.61
CBM	0	26	0.832	2.35

Though these statistics provide a good overview of the nature of data one is dealing with, yet some interesting facts are inhibited. For instance, one would like to know how many values fall within a certain range to have an idea of distribution of data in each metric domain. This information is revealed in Fig. 1 which illustrates the histograms for each metric of interest having count of classes on y-axis and corresponding metric values on x-axis.

[1] http://tomcat.apache.org/
[2]http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/
[3]http://ant.apache.org/

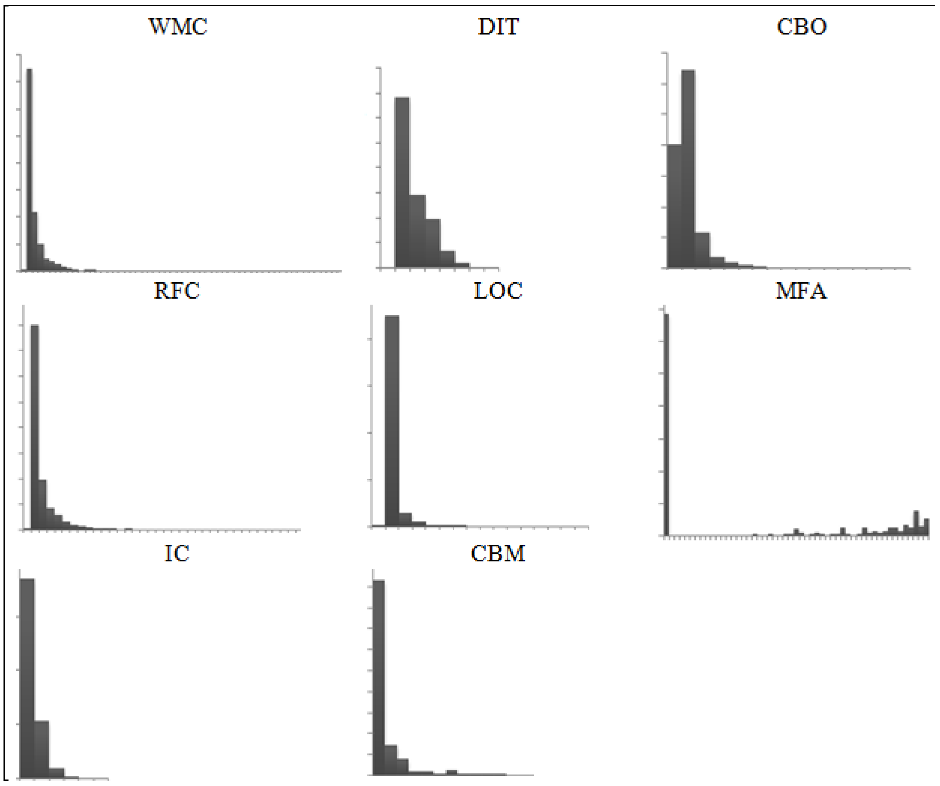

Fig 1: Histograms

One may notice in Fig. 1 that most of metrics are right-skewed and none of the metrics are normally distributed. This guides the next step where key goal is to find out interdependence among all by calculating correlation coefficient for each pair of metrics. To achieve this, Spearman's rank correlation [17] test has been applied which is not sensitive to non-normally distributed data. The correlation coefficient calculated takes value from +1 to -1. A value close to -1 indicates negative correlation, 0 implies no relation and +1 is designated as positive correlation. Correlations are obtained in Table 3 to show entities which signify the strength of relationship among their counterparts.

Table 3. Correlation values between OO metrics

	WMC	DIT	CBO	RFC	LOC	MFA	IC	CBM
WMC	1							
DIT	0.035	1						
CBO	0.418	0.110	1					
RFC	**0.850***	0.065	0.597	1				
LOC	0.747	0.064	0.551	**0.918***	1			
MFA	-0.249	**0.819***	-0.048	-0.174	-0.160	1		
IC	0.195	0.568	0.188	0.259	0.263	0.407	1	
CBM	0.242	0.571	0.227	0.295	0.285	0.375	**0.955***	1

** indicates strong relationships*

5. ANALYSIS AND RESULT INTERPRETATION

Histograms, in addition, might allow us to have an understanding of realistic metric thresholds. Thresholds are highly important in interpreting values of a metric. Knowing reference values of software metrics might strongly contribute to make them useful in practice [7]. Thresholds for software metrics are often used in the context of fault-proneness. This means that a measured entity is more fault-prone, if it violates a threshold [18]. Hence, they can act as guidelines for developers to keep them at optimum levels to reduce potential risks.

In Fig. 1, for instance, DIT values are 1 for 54% of the classes and 99.8% classes lie between 0 to 5 which is also an acceptable limit according to NASA technical report SATC [19]. In case of CBO, upper limit for 83% of classes is 5, again in agreement with SATC thresholds.

It is observed in Table 3 that (RFC, WMC), (LOC, RFC), (DIT, MFA), (IC, CBM) hold strong connections having correlation coefficient greater than 0.80. RFC inspects method calls within the class's method bodies and WMC implies number of methods in a class. It is apparent that both are significantly associated because if the count of method increases, logically it will have a positive influence on number of different methods that can be executed in response to a message. For similar reasons, RFC and LOC are also fairly correlated to a higher degree.

DIT is a measure of how many ancestor classes can potentially affect a certain class [2]; while MFA is the ratio of the number of methods inherited by a class to the total number of methods accessible. Consequently, more the number of inheritance levels in object hierarchy, more the likelihood of having higher MFA ratio.

IC carries highest degree of relationship with CBM as sound as 0.95. This makes sense because a class is coupled to its parent class (in case of IC) if one of its inherited methods are functionally dependent on the new or redefined methods; while CBM is the total number of new/redefined methods as mentioned earlier in Table 1.

In an endeavor to verify these findings, outcomes produced have been compared with results of [20][21][22][23][24] which carried out similar studies in the past on different set of software and various other metrics along with subset of metrics undertaken by us. Results are motivating as RFC, WMC, LOC have been found quite significant in their case studies as well, along with CBO. Interestingly, CBO showed off no significant association with any of the metrics in our case. DIT is found to be weak in relation with other significant threesome stated above and results are in accordance with [20][21][22][23][24] as well.

6. THREATS TO VALIDITY AND FUTURE WORK

Like any empirical study, there are number of issues which should be addressed in future to build more confidence in the results:

a) Software as data set was randomly chosen based on their huge customer base and usage in previous research studies. Our conclusions may be biased according to what representative data set (source code) was used to produce them. Also, data extracted and employed is from a single version of a project. Future research might lead us to verify the results across multiple versions and similar other projects.

b) Our findings are subject to set of metrics undertaken by us. There exist many other which may have crucial influence on inferences drawn. Moreover, pairs of metrics show strong association in one case, but it may not be the same with different set of software. Change of software may indicate the same metrics pair to have weak connection as shown in various past studies also. So without taking into consideration other factors such as type, platform and size of software, generalizations to other research settings is questionable. Metrics that were either insignificant or were not mentioned here require further study. Their measured values may be low as a result of the distinct nature of the software.

c) There are many tools that can measure quality metrics for the Java code base. This study is limited by the assumptions and accuracy of the CKJM in producing metrics values. Comparison of results with other measurement tools is an additional future course of action.

7. CONCLUSION

In this paper, a set of metrics for Apache Tomcat server have been calculated with the aid of CKJM, a metrics measurement tool, after building the source code (generating java byte code) with Apache ant. Degree of strength of relationship between each pair of metrics was derived using spearman correlation technique. Histograms were generated to show trends of distribution of values in each metric domain. Results reveal that (RFC, WMC), (LOC, RFC), (DIT, MFA), (IC, CBM) have strong connections having correlation coefficient greater than 0.80. Nevertheless, the relationship established here may be valid for only a specific population of systems.

There are various ways in which metrics can be interpreted that is why metrics should be placed in a context in order to maximize their benefits [25]. It would be beneficial for project managers to use metrics to identify extreme values, isolate problem areas, and stay well-versed about the trade-off between various software attributes of interest. As significance of metrics is related to nature of projects and goals of the organization, so these should be appropriately chosen with the aim to reduce cost and foster ease of development, implementation and maintenance over the life cycle of project.

8. 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. 1994. A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering, vol. 2O, no. 6, pp. 476-493.

[3] Nagappan, N., Ball, T., Zeller, A. 2006. Mining metrics to predict component failures. In ICSE, 452-461.

[4] Y. Jiang, B. Cukic, T. Menzies, N. Bartlow 2008. Comparing Design and Code Metrics for Software Quality Prediction. In Proceedings of the Workshop on Predictive Models in Software Engineering (PROMISE'08), Leipzig, Germany.

[5] Yutao Ma, Keqing He, Bing Li, Jing Liu, Xiao-Yan Zhou 2010. A Hybrid Set of Complexity Metrics for Large-Scale Object-Oriented Software Systems. J. Comput. Sci. Technol. 25(6): 1184-1201.

[6] Yasunari Takai, Takashi Kobayashi, Kiyoshi Agusa. 2011. Software Metrics based on Coding Standards Violations. In Proc. the Joint Conference of the 21th International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement (IWSM/MENSURA2011) pp.273-278, Nara, Japan.

[7] K. A. M. Ferreira, M. A. S. Bigonha, R. S. Bigonha, L. F. O. Mendes, and H. C. Almeida 2012. Identifying thresholds for object oriented software metrics. The Journal of Systems and Software, vol. 85, no. 2, pp. 244–257.

[8] A. Meneely, B. Smith, and L. Williams 2012. Validating software metrics: A spectrum of philosophies. ACM Transactions on Software Engineering and Methodology (TOSEM).

[9] Giulio Concas, Michele Marchesi, Giuseppe Destefanis, Roberto Tonelli 2012. An Empirical Study of Software Metrics for Assessing the Phases of an Agile Project. International Journal of Software Engineering and Knowledge Engineering 22(4): 525-548.

[10] Mel Ó Cinnéide, Laurence Tratt, Mark Harman, Steve Counsell, Iman Hemati Moghadam 2012. Experimental assessment of software metrics using automated refactoring. ESEM, 49-58, ACM.

[11] Raymond P. L. Buse, Thomas Zimmermann 2012. Information needs for software development analytics.. ICSE, 987-996, ACM.

[12] Eric Bouwers, Arie van Deursen, and Joost Visser, 2013. Evaluating Usefulness of Software Metrics – an Industrial Experience Report. In Proceedings International Conference on Software Engineering (ICSE), Software Engineering in Practice (SEIP) track, ACM/IEEE.

[13] D Radjenovic, M Hericko, Richard Torkar, Ales Zivkovic, 2013. Software Fault Prediction Metrics: A Systematic Literature Review. Information & Software Technology 55(8): 1397-1418.

[14] John Michura, Miriam A. M. Capretz, and Shuying Wang, 2013. Extension of Object-Oriented Metrics Suite for Software Maintenance. ISRN Software Engineering,

[15] D. Spinellis 2005. Tool Writing: A Forgotten Art?, IEEE Software, Vol. 22, No. 4, pp 9-11.

[16] Jagdish Bansiya and Carl G. Davis 2002. A hierarchical model for object-oriented design quality assessment. Software Engineering, IEEE Transactions on, 28(1):4–17.

[17] C. Spearman 1987. The proof and measurement of association between two things. The American Journal of Psychology, 100(3/4):441–471.

[18] S. Herbold, J. Grabowski, and S. Waack 2011. Calculation and optimization of thresholds for sets of software metrics", Journal of Empirical Software Engineering, vol. 16, no. 6, pp. 812–841.

[19] L. Rosenberg and L. Hyatt 2001. Software Quality Metrics for Object-Oriented System Environments. NASA Technical Report SATC no. 1, pp 11-58.

[20] Olague, H., Etzkorn, L., Gholston, S., & Quattlebaum, S. 2007. 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), pp.402–419.

[21] Jie Xu, Danny, Ho and Luiz, Fernando Capretz. 2008. An empirical validation of object-oriented design metrics for fault prediction. J. of Comp. Science 4, 7, 571—577.

[22] Jureczko M. 2011. Significance of Different Software Metrics in Defect Prediction. Software Engineering: An International Journal. No 1(1), 86-95.

[23] Okutan, A., O. T. Yildiz 2012. Software Defect Prediction using Bayesian Networks. Empirical Software Engineering, doi: 10.1007/s10664-012-9218-8.

[24] M. Badri and F. Toure 2012. Empirical Analysis of Object-Oriented Design Metrics for Predicting Unit Testing Effort of Classes. Journal of Software Engineering and Applications, Vol. 5 No. 7, pp. 513-526.

[25] Eric Bouwers, Arie van Deursen, Joost Visser 2013. Software metrics: pitfalls and best practices. ICSE 1491-1492.