Improving Object-Oriented Lack-of-Cohesion Metric by Excluding Special Methods

JEHAD AL DALLAL
Department of Information Science
Kuwait University
P.O. Box 5969, Safat 13060
KUWAIT
j.aldallal@ku.edu.kw

Abstract: - Classes are the basic units in object-oriented programs, and therefore, their quality has impact on the overall quality of the software. Class cohesion is a key quality factor, and it refers to the degree of relatedness of class attributes and methods. Software developers use class cohesion measure to assess the quality of their products and to guide the restructuring of poorly designed classes. Several class cohesion metrics are proposed in the literature, and the impact of considering the special methods (i.e., constructors, destructors, and access methods) in cohesion calculation is not empirically studied for most of them. In this paper, we address this issue for one of the most popular class cohesion metrics, referenced as Lack of Cohesion (LCOM). Our empirical study involves applying the metric with and without considering special methods on classes of two open source Java applications and statistically analyzing the results. The empirical study results show that the ability of LCOM in indicating class quality slightly improves when excluding special methods from the LCOM computation.

Key-Words: - object-oriented class, software quality, class cohesion metric, class cohesion, special methods.

1 Introduction

A popular goal of software engineering is to develop the techniques and the tools needed to develop high-quality applications that are more stable and maintainable. In order to assess and improve the quality of an application during the development process, developers and managers use several metrics. These metrics estimate the quality of different software attributes, such as cohesion, coupling, and complexity.

The cohesion of a module refers to the relatedness of the module components. A module that has high cohesion performs one basic function and cannot be split into separate modules easily. Highly cohesive modules are more understandable, modifiable, and maintainable [1].

Since the last decade, object-oriented programming languages, such as C++ and Java, have become widely used in both the software industry and research fields. In an object-oriented paradigm, classes are the basic modules. The members of a class are its attributes and methods. Therefore, class cohesion refers to the relatedness of the class members.

Researchers have introduced several metrics to indicate class cohesion during high or low level design phases. Lack of Cohesion (LCOM) [3] is proposed by Chidamber and Kemerer, and it counts the number of method pairs that do not directly share attributes. Higher LCOM value indicates low cohesion and vice versa. LCOM is widely applied and theoretically and empirically compared to other metrics (e.g., [1, 2, 3, 12, 13, 14, 21]). In these empirical studies, the goodness of the metric in indicating cohesion is indirectly measured by statistically analyzing the relation between the cohesion values and the values of external software attributes such as the fault proneness of the class (i.e., the extent to which a class is prone to faults). Most of the reported empirical results show that LCOM is relatively weakly capable in predicting faulty classes. As a result, LCOM is suggested not to be a good cohesion indicator. Originally, LCOM do not differentiate between the different types of methods. Some researchers (e.g., [3]) theoretically analyzed the impact of including/excluding special methods in LCOM measurement. However, none of the researchers studied this issue empirically. In this paper, we consider three different versions of each class: (1) a version including all special methods, (2) a version excluding access methods (i.e., setter and getter methods), and (3) a version excluding all types of special methods. We perform an empirical study to investigate which version is the best when LCOM is applied to indicate class quality. The empirical study is applied on classes of two open source Java systems that have available fault data repositories. The empirical study results show that excluding all types of special methods, in the computation of LCOM, slightly improves its goodness in predicting faulty class. This indirectly, indicates that excluding special methods improves LCOM's goodness in indicating class cohesion.

This paper is organized as follows. Section 2 provides an overview of the class cohesion metrics proposed in literature. Section 3 reports the empirical study setting, and Section 4 overviews and discusses the empirical study results. Finally, Section 5 concludes and discusses future work.

2 Related Work

Researchers have proposed several class cohesion metrics in the literature. These metrics can be applicable based on high-level design (HLD) or low-level design (LLD) information. HLD class cohesion metrics rely on information related to class and method interfaces. The more numerous LLD class cohesion metrics require an analysis of the algorithms used in the class methods (or the code itself if available) or access to highly precise method postconditions. Class cohesion metrics are based on the use or sharing of class attributes. For example, the LCOM metric counts the number of method pairs that do not share instance variables [15]. Chidamber and Kemerer [16] propose another version of the LCOM metric, which calculates the difference between the number of method pairs that do and do not share instance variables. Li and Henry [17] use an undirected graph that represents each method as a node and the sharing of at least one instance variable as an edge. They define lack-of-cohesion in methods as the number of connected components in the graph. The graph is extended in [18] by adding an edge between a pair of methods if one of them invokes the other. Hitz and Montazeri [18] introduce a connectivity metric to apply when the graph has one component. In addition, Henderson-Sellers [19] proposes a lack-of-cohesion in methods metric that considers the number of methods referencing each attribute.

Bieman and Kang [4] describe two class cohesion metrics, Tight Class Cohesion (TCC) and Loose Class Cohesion (LCC), to measure the relative number of directly connected pairs of methods and the relative number of directly or indirectly connected pairs of methods, respectively. TCC considers two methods to be connected if they share the use of at least one attribute. A method uses an attribute if the attribute appears in the method's body or the method invokes another method, directly or indirectly, that has the attribute in its body. LCC considers two methods to be connected if they share the use of at least one attribute directly or transitively. Badri [5] introduces two class cohesion metrics, Degree of Cohesion-Direct (DC_D) and Degree of Cohesion-Indirect (DC_I), that are similar to TCC and LCC, respectively, but differ by considering two methods connected also when both of them directly or transitively invoke the same method. Briand et al. [3] propose a cohesion metric (called Coh) that computes cohesion as the ratio of the number of distinct attributes accessed in methods of a class. Fernandez and Pena [6] propose a class cohesion metric, called Sensitive Class Cohesion Metric (SCOM), that considers the cardinality of the intersection between each pair of methods. In the metric presented by Bonja and Kidanmariam [7], the degree of similarity between methods is used as a basis to measure class cohesion. The similarity between a pair of methods is defined as the ratio of the number of shared attributes to the number of distinct attributes referenced by both methods. Cohesion is defined as the ratio of the summation of the similarities between all pairs of methods to the total number of possible pairs of methods. The metric is called Class Cohesion (CC). Al Dallal and Briand [1] propose a metric based on measuring the degree of similarity between each pair of methods in terms of the number of shared attributes.

Cohesion Among Methods in a Class (CAMC), Normalized Hamming Distance (NHD), Scaled NHD (SNHD), Distance Design-based Direct Class Cohesion (D_3C_2), and Similarity-based Class Cohesion (SCC) are examples of HLD metrics. CAMC [8], NHD, and SNHD [9] use the types of the method parameters to predict the interactions between the methods and attributes. D_3C_2 [10] uses the relation between the types of the parameters and the types of the attributes to predict the interactions between the methods and attributes. SCC [13] extends D_3C_2 by considering more types of interactions including the interactions caused by method invocations modelled in UML diagrams. Related work in the area of software cohesion can be found in [2, 11, 12, 14, 20, 21, 29].

3 Empirical Study Setting

We chose two Java open source software systems from two different domains: Art of Illusion v.2.5 [22] and JabRef v.2.3 beta 2 [23]. Art of Illusion consists of 481 classes and about 88 thousand lines of code (KLOC), and is a 3D modeling, rendering, and animation studio system. JabRef consists of 569 classes and about 48 KLOC, and is a graphical application for managing bibliographical databases. We chose these two open source systems randomly from http://sourceforge.net.The restrictions taken into account in choosing these systems were that they (1) are implemented using Java, (2) are relatively large in terms of the number of classes, (3) are from different domains, and (4) have available source code and fault repositories.

We considered three versions of each class: (1) a version that includes all types of special methods, (2) a version that excludes access methods and includes all other types of methods, and (3) a version that excludes all types of special methods. Consequently, the third version has the least number of methods. We excluded all classes that their third version has less than two methods because LCOM value is not defined for such classes. This implies excluding 177 classes from the first system and 310 classes from the second system. We applied the LCOM to the rest of the classes including the three considered versions. We developed our own Java tool to automate the cohesion measurement process for Java classes using LCOM. The resulting values for the three versions are referenced here as $LCOM_1$, $LCOM_2$, and $LCOM_3$, respectively. The tool analyzed the Java source code, extracted the information required to build the models that represent the cohesive interactions, and calculated the cohesion values for the three LCOM versions. Tables 1 and 2 show descriptive statistics for each cohesion measure including the minimum, 25% quartile, mean, median, 75% quartile, maximum value, and standard deviation. Note that the following analyses do not take into account class inheritance. The impact of inheritance on the study results is left as a subject for further research.

Table 1: Descriptive statistics for the cohesion measures applied on classes of Art of Illusion system

Statistic	$LCOM_1$	$LCOM_2$	$LCOM_3$
Min	0	0	0
Max	98	91	86
25%	2	0	0
Med	11	4	2
Mean	21.4	10.6	7.9
75%	36	14	11
Std. Dev.	25.1	15.0	12.3

The descriptive statistics results show that excluding special methods reduces LCOM values. This is because excluding special methods decreases the number of pairs of methods that do not share common attributes, and consequently decreases LCOM values.

Table 2: Descriptive statistics for the cohesion measures applied on classes of JabRef system

Statistic	$LCOM_1$	$LCOM_2$	$LCOM_3$
Min	0	0	0
Max	92	58	49
25%	0	0	0
Med	5	1	0
Mean	11.3	5.0	3.1
75%	17	5	2
Std. Dev.	15.8	9.4	7.4

4 Empirical Study Results

To study the relationship between the values of the collected metrics and the extent to which a class is prone to faults, we applied logistic regression [24], a standard and mature statistical method based on maximum likelihood estimation. This method is widely applied to predict fault-prone classes (e.g., [3, 25, 26, 27]). In logistic regression, explanatory or independent variables are used to explain and predict dependent variables. A dependent variable can only take discrete values and is binary in the context where we predict fault-prone classes. The logistic regression model is univariate if it features only one explanatory variable and multivariate when including several explanatory variables. In this case study, the dependent variable indicates the presence of one or more faults in a class, and the explanatory variables are the cohesion metrics. Univariate regression is applied to study the fault prediction of each metric separately, whereas multivariate regression is applied to study the fault prediction of different combinations of metrics to determine the best model. In this paper, we focus on comparing the results for the metrics in terms of their individual fault prediction power, and therefore, we consider only univariate regression.

We collected fault data for the classes in the considered software systems from publicly available fault repositories. The developers of the considered systems used an on-line Version Control System (VCS) to keep track of the changes performed on the source code of the system. The changes, called revisions, are due to either detected faults or required feature improvements. Each revision is associated with a report including the revision description and a list of classes involved in this change. Two research assistants, one with a B.Sc. in computer science and

ISBN: 978-960-474-277-6

six years of experience in software development activities and another with a B.Sc. and Master both in computer science; each alone, manually traced the description of each revision and identified the ones performed due to detected faults. Author of this paper compared the manual results and rechecked the results in which the two assistants differ to choose the correct one. Finally, we used the lists of classes involved in changes due to detected faults to count the number of faults in which each class is involved. We classified each class as being fault-free or as having at least one fault. Ideally, class cohesion should be measured before each fault occurrence and correction, and used to predict this particular fault occurrence. However, not only this would mean measuring cohesion for dozens of versions (between each fault correction) for each system, but we would not be able to study the statistical relationships of a set of faults with a set of consistent cohesion measurements for many classes. Our cohesion measurement is based on the latest version of the source code, after fault corrections, and is therefore an approximation. This is however quite common in similar research endeavors (e.g., [3,25,26,27]) and is necessary to enable statistical analysis.

The results of the univariate regression study are reported in Tables 3 and 4. Estimated regression coefficients are reported. The larger the absolute value of the coefficient is, the stronger the impact (positive or negative, according to the sign of the coefficient) of the metric on the probability of a fault being detected in a class. The considered metrics have different standard deviations as shown in Tables 1 and 2. Therefore, to help compare the coefficients, we standardized the explanatory variables by subtracting the mean and dividing by the standard deviation and, as a result, they all have an equal variance of 1 and the coefficients reported in Tables 3 and 4 are also standardized. These coefficients represent the variation in standard deviations in the dependent variable when there is a change of one standard deviation in their corresponding independent variable. The p-value is the probability of the coefficient being different from zero by chance, and is also an indicator of the accuracy of the coefficient estimate. We use a typical significance threshold (α=0.05) to determine whether a metric is a statistically significant fault predictor.

To evaluate the performance of a prediction model regardless of any particular threshold, we used the receiver operating characteristic (ROC) curve [28]. In a fault prediction context, the ROC curve is a graphical plot of the ratio of classes correctly classified as faulty versus the ratio of classes incorrectly classified as faulty at different thresholds. The area under the ROC curve shows the ability of the model to correctly rank classes as faulty or non-faulty. A 100% ROC area represents a perfect model that correctly classifies all classes. The larger the ROC area, the better the model in terms of classifying classes. The results for all the coefficients and for all considered metrics are reported in Tables 3 and 4.

Table 3: Univariate logistic regression results for classes of Art of Illusion system

Metric	$LCOM_1$	$LCOM_2$	$LCOM_3$
Std. Coeff.	0.50	0.56	0.57
p-value	<0.0001	<0.0001	<0.0001
ROC area	65.4%	66.6%	68.1%

Table 4: Univariate logistic regression results for classes of JabRef system

Metric	$LCOM_1$	$LCOM_2$	$LCOM_3$
Std. Coeff.	0.98	1.26	1.38
p-value	0.001	0.002	0.013
ROC area	69.7%	69.8%	71.6%

The results in Tables 3 and 4 lead to the following conclusions:

1. All versions of LCOM (i.e., with and without considering special methods) are statistically significant at α=0.05 (i.e., their coefficients are significantly different from 0).
2. As expected, the estimated regression coefficients for the three versions of the inverse cohesion measure LCOM are positive. This indicates an increase in the predicted probability of fault detection as the lack of cohesion of the class increases.
3. In both systems, the results of the standard coefficient and the area under the ROC curve are slightly improved when excluding access methods. In addition, they are further slightly improved when excluding all types of special methods.

As a result, the empirical results above show that the versions of LCOM that exclude special methods predict faulty classes more accurately than the original LCOM that accounts for all types of special methods. These results indirectly indicate that the ability of LCOM in indicating class cohesion improves when excluding special methods.

5 Conclusions and Future Work

This paper investigates versions of LCOM, a widely referenced class cohesion metric. The versions of LCOM consider the inclusion and exclusion of special methods from LCOM computation. The original and other versions of the metric are empirically compared by applying them on classes of two open source systems. The results show that the versions of the metric, when excluding special; methods, predict faulty classes, and thus indicate cohesion, better than the original version of the metric.

In the future, we plan to extend the empirical study by including classes from other systems of different programming languages. In addition, we plan to carefully study the impact of including/excluding special methods on the cohesion values of other cohesion metrics. Finally, we intend to empirically study the impact of considering other factors when applying cohesion metrics such as inheritance.

Acknowledgment

The author would like to acknowledge the support of this work by Kuwait University Research Grant WI03/07.

References

[1] J. Al Dallal and L. Briand, A precise method-method interaction-based cohesion metric for object-oriented classes, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, in press, 2010.

[2] J. Al Dallal, Mathematical validation of object-oriented class cohesion metrics, *International Journal of Computers*, 2010, Vol. 4, No. 2, pp. 45-52.

[3] L. C. Briand, J. Daly, and J. Wuest, A unified framework for cohesion measurement in object-oriented systems, *Empirical Software Engineering - An International Journal*, Vol. 3, No. 1, 1998, pp. 65-117.

[4] J. M. Bieman and B. Kang, Cohesion and reuse in an object-oriented system, *Proceedings of the 1995 Symposium on Software reusability*, Seattle, Washington, United States, pp. 259-262, 1995.

[5] L. Badri and M. Badri, A Proposal of a new class cohesion criterion: an empirical study, *Journal of Object Technology*, Vol. 3, No. 4, 2004..

[6] L. Fernández, and R. Peña, A sensitive metric of class cohesion, *International Journal of Information Theories and Applications*, Vol. 13, No. 1, 2006, pp. 82-91.

[7] C. Bonja and E. Kidanmariam, Metrics for class cohesion and similarity between methods, *Proceedings of the 44th Annual ACM Southeast Regional Conference*, Melbourne, Florida, 2006, pp. 91-95.

[8] J. Bansiya, L. Etzkorn, C. Davis, and W. Li, A class cohesion metric for object-oriented designs, *Journal of Object-Oriented Program*, Vol. 11, No. 8, pp. 47-52. 1999.

[9] S. Counsell , S. Swift , and J. Crampton, The interpretation and utility of three cohesion metrics for object-oriented design, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 15, No. 2, 2006, pp.123-149.

[10] J. Al Dallal, A design-based cohesion metric for object-oriented classes, *International Journal of Computer Science and Engineering*, 2007, Vol. 1, No. 3, pp. 195-200.

[11] J. Al Dallal, Software similarity-based functional cohesion metric, *IET Software*, 2009, Vol. 3, No. 1, pp. 46-57.

[12] J. Al Dallal, Theoretical validation of object-oriented lack-of-cohesion metrics, proceedings of the *8th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems (SEPADS 2009)*, Cambridge, UK, February 2009.

[13] J. Al Dallal and L. Briand, An object-oriented high-level design-based class cohesion metric, *Information and Software Technology*, 2010, Vol. 52, No. 12, pp. 1346-1361.

[14] J. Al Dallal, Measuring the discriminative power of object-oriented class cohesion metrics, *IEEE Transactions on Software Engineering*, In press, 2010.

[15] S.R. Chidamber and C.F. Kemerer, Towards a Metrics Suite for Object-Oriented Design, Object-Oriented Programming Systems, *Languages and Applications (OOPSLA), Special Issue of SIGPLAN Notices*, Vol. 26, No. 10, 1991, pp. 197-211.

[16] S.R. Chidamber and C.F. Kemerer, A Metrics suite for object Oriented Design, *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, 1994, pp. 476-493.

[17] W. Li and S.M. Henry, Maintenance metrics for the object oriented paradigm. *In Proceedings of 1st International Software Metrics Symposium*, Baltimore, MD, 1993, pp. 52-60.

[18] M. Hitz and B. Montazeri, Measuring coupling and cohesion in object oriented systems, *Proceedings of the International Symposium on Applied Corporate Computing*, 1995, pp. 25-27.

[19] B. Henderson-Sellers, *Software Metrics*, Prentice Hall, Hemel Hempstaed, U.K., 1996.

[20] J. Al Dallal, Efficient program slicing algorithms for measuring functional cohesion and parallelism, *International Journal of Information Technology*, Vol. 4, No. 2, 2007, pp. 93-100.

[21] J. Al Dallal, Fault prediction and the discriminative powers of connectivity-based object-oriented class cohesion metrics, submitted for publication in *IEEE Transactions on Software Engineering*, 2010.

[22] Illusion, http://sourceforge.net/projects/aoi/, July 2010.
[23] JabRef, http://sourceforge.net/projects/jabref/, July 2010.
[24] D. Hosmer and S. Lemeshow, *Applied Logistic Regression*, Wiley Interscience, 2000, 2nd edition.
[25] L. C. Briand, J. Wüst, and H. Lounis, Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs, *Empirical Software Engineering*, 6(1), 2001, pp. 11-58.
[26] T. Gyimothy, R. Ferenc, and I. Siket, Empirical validation of object-oriented metrics on open source software for fault prediction, *IEEE Transactions on Software Engineering*, 3(10), 2005, pp. 897-910.
[27] A. Marcus, D. Poshyvanyk, and R. Ferenc, Using the conceptual cohesion of classes for fault prediction in object-oriented systems, *IEEE Transactions on Software Engineering*, 34(2), 2008, pp. 287-300.
[28] J. A. Hanley and B. J. McNeil, The meaning and use of the area under a receiver operating characteristic (ROC) curve, *Radiology*, 143(1), 1982, pp. 29-36.
[29] J. Al Dallal, Improving the applicability of object-oriented class cohesion metrics, *Information and Software Technology*, 2011, in press.