International Journal of Business, Human and Social Sciences
ISSN: 2517-9411
Vol:8, No:3, 2014

Theoretical Analysis of the Effect of Accounting for Special Methods in Similarity-Based Cohesion Measurement

Jehad Al Dallal

***Abstract*—**Class cohesion is an important object-oriented software quality attributes, and it refers to the degree of relatedness of class attributes and methods. Several class cohesion measures are proposed in the literature, and the impact of considering the special methods (i.e., constructors, destructors, and access and delegation methods) in cohesion calculation is not thoroughly theoretically studied for most of them. In this paper, we address this issue for three popular similarity-based class cohesion measures. For each of the considered measures we theoretically study the impact of including or excluding special methods on the values that are obtained by applying the measure. This study is based on analyzing the definitions and formulas that are proposed for the measures. The results show that including/excluding special methods has a considerable effect on the obtained cohesion values and that this effect varies from one measure to another. The study shows the importance of considering the types of methods that have to be accounted for when proposing a similarity-based cohesion measure.

***Keywords*—**Object-oriented class, software quality, class cohesion measure, class cohesion, special methods.

I. 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 measures. These measures 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], [2].

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 measures to indicate class cohesion during high or low level design phases. These measures follow different approached to estimate the cohesion of a class. For example, some of the measures are based on counting the number of pairs of methods that share common attributes [3], [4]. Some others are more precise and they are based on measuring the similarity between each pair of methods in terms of the ratio of the shared common attributes [2], [5], [6]. In this paper, we consider three class cohesion measures: LSCC [2], SCOM [5], and CC [6]. We selected these cohesion measures because they are similarity based measures and they are widely theoretically [7], [8] and empirically [9]-[20] studied.

Classes include special types of methods, such as constructors, destructors, and access and delegation methods. Constructors are used to initialize most or all of the attributes in the class and destructors are used to deinitialize most or all of the attributes. Access methods are classified as either setters or getters. A setter method initializes a single attribute and a getter method returns the reference/value of a single attribute. Finally, a delegation method is used to inquire about the status of a single attribute. Each of these special methods has its own characteristics, which can artificially affect the class cohesion value. Incorrectly determining whether to include or exclude the special methods in cohesion measurement can lead to improper re-designing decisions and actions based on the misleading class cohesion values that are obtained. However, the original definitions for the considered measures do not differentiate between the different types of methods, which make these measures ill-defined. The impact of including/excluding special methods in cohesion measures on the obtained values and refactoring and fault prediction activities is empirically studied by Al Dallal [15]. However, this impact is not thoroughly theoretically studied yet.

In this paper, we analyze the definitions and formulas of the considered cohesion measures to study the impact of including/excluding each type of special methods on the values that can be obtained by the measures. Based on the analysis, a recommendation is provided for each measure for whether to include or exclude special methods in cohesion measurement.

This paper is organized as follows. Section II provides an overview of the class cohesion measures proposed in literature. Section III reports the theoretical analysis and results. Finally, Section IV concludes the paper and discusses future work.

The author would like to acknowledge the support of this work by Kuwait University Research Grant WI01/12.

Jehad Al Dallal is with Department of Information Sciences, Kuwait University, P.O. Box 5969, Safat 13060, Kuwait (e-mail: j.aldallal@ku.edu.kw).

II. Related Work

Researchers have proposed several class cohesion measures in the literature. These measures can be applicable based on high-level design (HLD) [10], [21]-[23], [12] or low-level design (LLD) information [1], [3]-[6]. HLD class cohesion measures rely on information related to class and method interfaces. The more numerous LLD class cohesion measures 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 measures are based on the use or sharing of class attributes. For example, Bieman and Kang [3] describe two class cohesion measures, 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 [4] introduces two class cohesion measures, Degree of Cohesion-Direct (DCD) and Degree of Cohesion-Indirect (DCI), 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. [8] propose a cohesion measure (called Coh) that computes cohesion as the ratio of the number of distinct attributes accessed in methods of a class. Fernández and Peña [5] propose a class cohesion measure, called Sensitive Class Cohesion Measure (SCOM), which considers the cardinality of the intersection between each pair of methods. In the measure presented by Bonja and Kidanmariam [6], 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 measure is called Class Cohesion (CC). Al Dallal and Briand [1] propose a different formula for the similarity between a pair of method calculated as the ratio of the number of shared attributes to the total number of attributes in the class. They defined their Low-level design Sensitive Class Cohesion measure, LSCC, to be the relative similarities among the pairs of methods in the class. Related work in the area of software cohesion can be found in [24], [21], [5].

III. Theoretical Analysis

Here, we theoretically study the impact of including or excluding special methods on the values that are obtained by applying each of the eight considered measures. This study is based on analyzing the definitions and formulas that are proposed for the measures. In addition, this study is based on the following typical observations:

1. Potentially, constructors and destructors can reference most if not all of the attributes of the class. As a result, there is a higher chance that each one of the constructors and destructors references more distinct attributes than any other method in the class.
2. Each one of the access or delegation methods references a single attribute. Peer access and delegation methods are the setter, getter, and delegation methods that reference the same attribute. Non-peer access and delegation methods reference different attributes.
3. A non-special method can reference any number of distinct attributes; however, typically, it references a lower number of distinct attributes in comparison to constructors and destructors. A non-special method may not reference any attributes, although, theoretically, this is an unusual case.
4. Constructors and destructors as well as access and delegation methods have almost the same characteristics in terms of the number of referenced attributes. That is, each constructor or destructor potentially references most or all attributes, whereas each access or delegation method references a single attribute. Therefore, for the rest of this section, the discussion regarding the impact of including the constructors is applicable to destructors as well. Similarly, the discussion regarding the access methods also applies to delegation methods.

For each of the considered measures, the cases in which the inclusion of constructors and access methods causes the measure value to increase or decrease are identified and analyzed. Based on this analysis, a recommendation to include or exclude the special methods is given. Analytically, special methods have no influence on the cohesion of the class. Therefore, if the inclusion of the special methods usually causes the measure value to increase or decrease, the recommendation will be to exclude them from the cohesion measurement. On the other hand, the recommendation will be to include the special methods if this inclusion slightly changes or does not usually change the obtained measure value and if the inclusion does not increase the cohesion computational complexity.

A. LSCC

LSCC can be calculated by obtaining the ratio of the summation of the similarities between all pairs of methods to the total number of pairs of methods. The similarity between a pair of methods is determined by dividing the number of shared attributes between the two methods by the total number of attributes in the class. The similarity between a constructor and any other method m is expected to be higher than the similarity between method m and any other method in the class. As a result, the inclusion of a constructor method artificially increases the LSCC value.

The similarity between an access method and any other non-peer access method is zero, and it is low (i.e., has a value

of 1/l) between peer access methods, where l is the number of attributes. In addition, the similarity between an access method and any other method has a maximum value of 1/l, whereas, generally, the maximum possible similarity between a pair of methods is one. Therefore, the inclusion of access methods is expected to artificially reduce the LSCC value. As a result, when applying LSCC, the recommendation is to exclude constructors and access methods from the cohesion measurement.

B. CC

The similarity definition of CC differs from that of LSCC in the sense that the similarity between a pair of methods is the ratio of the number of shared attributes between the two methods to the total number of distinct attributes that are referenced by either of the two methods. As a result, the similarity between a constructor method and another method depends on the percentage p of attributes that are accessed by the other method. That is, if the value of p is relatively high, then the similarity will be relatively high, and vice versa. Typically, p is not high for methods other than the constructor, and, therefore, the inclusion of a constructor is expected to decrease the value of CC from this perspective; however, from a different point of view, because the constructor references most or all of the attributes, the chance that the constructor is similar in some degree to other methods is higher than that for a pair of non-special methods. It is important to remember that methods that do share common attributes have a similarity of degree zero. Therefore, from this perspective, including a constructor can cause the CC value to increase. As a result, these two points of view oppositely impact the CC value, and they depend on factors that vary from one class to another. Therefore, the impact of including constructors is not generally determined. That is, in some typical cases, the value of CC considerably increases or decreases.

The similarity between an access method and any (1) non-peer access method is zero, (2) peer access method is one (i.e., maximum), and (3) other method depends on the attributes that are accessed by that other method. If the other method shares the same attribute with the access method, then the similarity will be one in the event that the other method accesses only an attribute, and it will linearly decrease as the number of attributes that are accessed by the other method increases. This means that the impact of including the access methods is not generally determined; the inclusion of access methods causes the CC value to increase in some typical cases and to decrease in some other typical cases. As a result, the recommendation is to exclude constructors and access methods from the CC measurement.

C. SCOM

Typically, the similarity definition for SCOM is more complicated than that for LSCC and CC. Here, the similarity between two methods depends on three factors. The similarity is proportional to (1) the number of shared attributes and (2) the relative number of distinct attributes that are accessed by either of the two methods, and it is reversely proportional to the minimum number of attributes that are accessed by each method. When measuring the similarity between a constructor and another method, the first and third factors will eliminate the effects of one another if the constructor references all of the attributes that are referenced by the method. As a result, the similarity between a constructor and another method depends on the second factor (i.e., the relative number of distinct attributes that are referenced by the constructor), which is typically high. Therefore, including the constructor potentially causes the SCOM value to artificially increase.

The similarity between an access method and a non-peer access method is zero, and it is equal to 1/l for a pair of peer access methods. The similarity between an access method and another method equals zero when the pair of methods does not share a common attribute, and it is equal to the relative number of attributes that are accessed by the other method when the pair of methods share a common attribute. In other words, the similarity between an access method and another method is usually low, and it is only high in one unusual case when the other method references a high relative number of attributes that include the attribute that is referenced by the access method. As a result, the inclusion of the access method is expected to artificially decrease the SCOM value. In conclusion, the recommendation is to exclude both of the constructors and access methods from the SCOM measurement.

IV. Conclusions and Future Work

This paper provides an analysis for the impact of including several types of special methods in cohesion measurement performed using three different widely applied similarity-based cohesion measures. The definitions and formulas of the measures are analyzed to figure out the impact of including special methods on the values obtained using the considered measures. The analysis demonstrated that the impact of including special methods varies among the types of the special methods considered and among the measures themselves. Finally, the analysis showed that the values obtained using the considered measures are expected to be artificially affected by the inclusion of the special methods. This indicates the importance of considering this issue whenever a measure is introduced.

In the future, we plan to extend the analysis to include more existing measures and to study the impact of accounting for special methods in cohesion measurement on practical issues of interest for software practitioners such as reusability, maintainability, and testability.

References

[1] Al Dallal, J. and Briand, L., A Precise Method-Method Interaction-Based Cohesion Metric for Object-Oriented Classes, ACM Transactions on Software Engineering and Methodology (TOSEM), 2012, Vol. 21, No. 2, pp. 8:1-8:34.

[2] Al Dallal, J. Object-Oriented Class Maintainability Prediction Using Internal Quality Attributes, Information and Software Technology, 2013b, Vol. 55, No. 11, pp. 2028-2048.

[3] Bieman, J. and Kang, B., Cohesion and Reuse in an Object-Oriented System, Proceedings of the 1995 Symposium on Software reusability, Seattle, Washington, United States, 1995, pp. 259-262.
[4] Badri, L. and Badri, M., A Proposal of a New Class Cohesion Criterion: An Empirical Study, Journal of Object Technology, 3(4), 2004, pp. 145-159.
[5] Fernández, L., and Peña, R., A Sensitive Metric of Class Cohesion, International Journal of Information Theories and Applications, 13(1), 2006, pp. 82-91.
[6] Bonja, C. and Kidanmariam, E., Metrics for Class Cohesion and Similarity between Methods, Proceedings of the 44th Annual ACM Southeast Regional Conference, Melbourne, Florida, 2006, pp. 91-95.
[7] Al Dallal, J., Mathematical Validation of Object-Oriented Class Cohesion Metrics, International Journal of Computers, 2010, Vol. 4, No. 2, pp. 45-52.
[8] Briand, L. C., Daly, J., and Wüst, J., A Unified Framework for Cohesion Measurement in Object-Oriented Systems, Empirical Software Engineering - An International Journal, Vol. 3, No. 1, 1998, pp. 65-117.
[9] Al Dallal, J., A Design-Based Cohesion Metric for Object-Oriented Classes, International Journal of Computer Science and Engineering, 2007, Vol. 1, No. 3, pp. 195-200.
[10] Al Dallal, J. and Briand, L., An Object-Oriented High-Level Design-Based Class Cohesion Metric, Information and Software Technology, 2010, Vol. 52, No. 12, pp. 1346-1361.
[11] Al Dallal, J., Improving Object-Oriented Lack-of-Cohesion Metric by Excluding Special Methods, Proceedings of the 10th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems (SEPADS 2011), Cambridge, UK, February 2011a.
[12] Counsell, S., Swift, S., and Crampton, J., 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.
[13] Briand, L. C., Wüst, J., and Lounis, H., Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs, Empirical Software Engineering, 6(1), 2001, pp. 11-58.
[14] Marcus, M., Poshyvanyk, D., and Ferenc, R., Using the Conceptual Cohesion of Classes for Fault Prediction in Object-Oriented Systems, IEEE Transactions on Software Engineering, 34(2), 2008, pp. 287-300.
[15] Al Dallal, J., The Impact of Inheritance on the Internal Quality Attributes of Java Classes, Kuwait Journal of Science, 2012d, Vol. 39, No. 2A, pp. 131-154.
[16] Al Dallal, J., Incorporating Transitive Relations in Low-Level Design-Based Class Cohesion Measurement, Software: Practice and Experience, 2013a, Vol. 43. No. 6, pp. 685-704.
[17] Al Dallal, J., Fault Prediction and the Discriminative Powers of Connectivity-Based Object-Oriented Class Cohesion Metrics, Information and Software Technology, 2012b, Vol. 54, No. 4, pp. 396-416.
[18] Al Dallal, J., Constructing Models for Predicting Extract Subclass Refactoring Opportunities Using Object-Oriented Quality Metrics, Information and Software Technology, 2012a, Vol. 54, No. 10, pp. 1125-1141.
[19] Al Dallal, J. and Morasca, S., Predicting Object-Oriented Class Reusability Using Internal Quality Attributes, Empirical Software Engineering, in press, 2012.
[20] Al Dallal, J., The Impact of Accounting for Special Methods in the Measurement of Object-Oriented Class Cohesion on Refactoring and Fault Prediction Activities, Journal of Systems and Software, 2012c, Vol. 85, No. 5, pp. 1042-1057.
[21] Al Dallal, J., Measuring the Discriminative Power of Object-Oriented Class Cohesion Metrics, IEEE Transactions on Software Engineering, 2011c, Vol. 37, No. 6, pp. 788-804.
[22] Al Dallal, J., Improving the Applicability of Object-Oriented Class Cohesion Metrics, Information and Software Technology, 2011b, Vol. 53, No. 9, pp. 914-928.
[23] Al Dallal, J., Transitive-Based Object-Oriented Lack-of-Cohesion Metric, Procedia Computer Science (Elsevier), Volume 3, 2011d, pp. 1581-1587.
[24] Al Dallal, J., Software Similarity-Based Functional Cohesion Metric, IET Software, 2009, Vol. 3, No. 1, pp. 46-57.