Qualitative Analysis for the Impact of Accounting for Special Methods in Object-Oriented Class Cohesion Measurement

Jehad Al Dallal
Kuwait University/Department of Information Sciences, Kuwait
Email: jehad@ku.edu.kw

Abstract— **Class cohesion is a key object-oriented software quality attribute. It refers to the degree of relatedness of class attributes and methods. Several class cohesion metrics are proposed in the literature. However, 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 the existing cohesion metrics. An incorrect determination of whether to include or exclude the special methods in cohesion measurement can lead to improper refactoring decisions according to the misleading class cohesion values that are obtained. In this paper, we qualitatively analyze the impact of including or excluding the special methods in cohesion measurement on the values that are obtained by applying 19 popular class cohesion metrics. The study is based on analyzing the definitions and formulas that are proposed for the metrics. The results show that including/excluding special methods has a considerable effect on the cohesion values that are obtained and that this effect varies from one metric to another. The study shows the importance of considering the types of methods that must be accounted for when proposing a cohesion metric.**

Index Terms—**object-oriented design, class quality, class cohesion, cohesion metric, special methods.**

I. Introduction

Proposing the techniques and developing the tools needed to develop high-quality applications that are more stable and maintainable is an important goal of software engineering. During the software development process, developers and managers use several metrics to assess and improve the quality of an application. 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. These metrics follow different approached to estimate the cohesion of a class. For example, some of the metrics are based on counting the number of distinct attributes accessed by the methods (e.g., [2]). Some others are based on counting the number of pairs of methods that share common attributes (e.g., [3]).

In object-oriented programming, some methods are typically provided to support the encapsulation key feature of the object-oriented paradigm. Some of these methods do not provide any functionality that contributes to the problem for which the class was developed. Instead, they are used to initialize attributes or inquire about their values. We refer to these methods as special methods, and we classify them into four different types, including constructors, destructors, access methods, 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. The 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. An incorrect determination of whether to include or exclude the special methods in the cohesion measurement can lead to improper re-designing decisions and actions that are based on the misleading class cohesion values that are obtained. However, the original definitions for most of the existing class cohesion metrics do not differentiate between the different types of methods; therefore, these metrics are ill-defined. The impact of including/excluding the special methods in cohesion measurement is empirically studied by Al Dallal [4], although a solid theoretical discussion that is based on the formulas applied by the metrics is not provided. Such a theoretical qualitative analysis for the formulas of the metrics helps in interpreting the empirical results. In addition, the qualitative analysis provides a more general view for the possible scenarios in which the cohesion values are affected when the special methods are either included or excluded. In contrast, the empirical analysis is limited to those scenarios that exist in the selected samples.

In this paper, we qualitatively analyzed the definitions and formulas of 19 existing cohesion metrics to investigate the impact of including/excluding each type of special methods on the values that can be obtained by the metrics. The study was based on identifying typical cases that are featured by the special methods and analyzing the changes in the measured cohesion values for these cases. The 19 cohesion metrics that were selected apply different cohesion measurement approaches. The results of the qualitative analysis are compared to the empirical results that were reported by Al Dallal [4].

This paper is organized as follows: Section 2 provides an overview of the class cohesion metrics that are proposed in literature. Section 3 summarizes the related work. Section 4 explores the cohesive relations of the special methods. Section 5 reports the qualitative analysis of the impact of accounting for the special methods in the cohesion measurement. Finally, Section 6 concludes the paper and discusses future work.

II. Class Cohesion Metrics

Several metrics have been proposed in the literature to measure cohesion in object-oriented systems at different abstraction levels including method (e.g., [5]) and class. The class cohesion metrics can be classified according to different perspectives, such as the types of considered interactions, the development phase during which they are applicable, and the types of considered methods. In this paper, we consider 19 metrics including LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, LSCC, CC, SCOM, Coh, TCC, LCC, DC_D, DC_I, CBMC, PCCC, OL_n, CAMC, NHD, and MMAC, as defined in Table 1. The last three metrics are applicable during the high-level design phase, whereas the rest are applicable during the low-level design phase.

TABLE 1:
DEFINITIONS OF THE CONSIDERED CLASS COHESION METRICS.

Class Cohesion Metric	**Definition/Formula**
The Lack of Cohesion in Methods (LCOM1) [6]	LCOM1= Number of pairs of methods that do not share attributes.
LCOM2 [7]	*P*= Number of pairs of methods that do not share attributes. *Q*= Number of pairs of methods that share attributes. $LCOM2 = \begin{cases} P-Q & if\ P-Q \geq 0 \\ 0 & otherwise \end{cases}$
LCOM3 [8]	LCOM3= Number of connected components in the graph that represents each method as a node and the sharing of at least one attribute as an edge.
LCOM4 [9]	Similar to LCOM3, and additional edges are used to represent method invocations.
LCOM5 [10]	LCOM5= (*kl*-*a*)/(*kl*-*l*), where *l* is the number of attributes, *k* is the number of methods, and *a* is the summation of the number of distinct attributes that are accessed by each method in a class.
Low-level design Similarity-based Class Cohesion (LSCC) [1]	$LSCC(C) = \begin{cases} 0 & \text{if } l = 0 \text{ and } k > 1, \\ 1 & \text{if } (l > 0 \text{ and } k = 0) \text{ or } k = 1, \\ \frac{\sum_{i=1}^{l} x_i(x_i - 1)}{lk(k-1)} & \text{otherwise.} \end{cases}$, where *l* is the number of attributes, *k* is the number of methods, and x_i is the number of methods that reference attribute *i*.
Class Cohesion (CC) [11]	CC= The ratio of the summation of the similarities between all pairs of methods to the total number of pairs of methods. The similarity between methods *i* and *j* is defined as: $Similarity(i,j) = \frac{\lvert I_i \cap I_j \rvert}{\lvert I_i \cup I_j \rvert}$, where I_i and I_j are the sets of attributes that are referenced by methods *i* and *j*, respectively.
Class Cohesion Metric (SCOM) [12]	SCOM= The ratio of the summation of the similarities between all pairs of methods to the total number of pairs of methods. The similarity between methods *i* and *j* is defined as: $Similarity\,(i,j) = \frac{\lvert I_i \cap I_j \rvert}{\min(\lvert I_i \rvert, \lvert I_j \rvert)} \cdot \frac{\lvert I_i \cup I_j \rvert}{l}$, where *l* is the number of attributes.
Coh [2]	Coh=*a/kl*, where *a*, *k*, and *l* have the same definitions above.
Tight Class Cohesion (TCC) [3]	TCC= The relative number of directly connected pairs of methods, wherein two methods are directly connected if they are directly connected to an attribute. A method *m* is directly connected to an attribute when the attribute appears within the method's body or within the body of a method that is directly or transitively invoked by method *m*.
Loose Class Cohesion (LCC) [3]	LCC= The relative number of directly or transitively connected pairs of methods, wherein two methods are transitively connected if they are directly or indirectly connected to an attribute. A method *m* that is directly connected to an attribute *j* is indirectly connected to an attribute *i* when there is a method that is directly or transitively connected to both attributes *i* and *j*.
Degree of Cohesion-Direct (DC_D) [13]	DC_D= The relative number of directly connected pairs of methods, wherein two methods are directly connected if they satisfy the condition mentioned above for TCC or if the two methods directly or transitively invoke the same method.

TABLE 1 (CONTINUATION):
DEFINITIONS OF THE CONSIDERED CLASS COHESION METRICS.

Class Cohesion Metric	**Definition/Formula**
Degree of Cohesion-Indirect (DC_I) [13]	DC_I= The relative number of directly or transitively connected pairs of methods, wherein two methods are transitively connected if they satisfy the same condition mentioned above for LCC or if the two methods directly or transitively invoke the same method.
Cohesion Based on Member Connectivity (CBMC) [14]	CBMC(G)=F_c(G)×F_s(G), where F_c(G)=|M(G)|/|N(G)|, M(G)=the number of glue methods in graph G, N(G)=the number of non-special methods in graph G, $F_s(G)=[\sum_{i=1}^{n} CBMC(G^i)]/n,$ n=the number of child nodes of G, and glue methods is the minimum set of methods for which their removal causes the class-representative graph to become disjointed.
Path Connectivity Class Cohesion (PCCC) [15]	$PCCC(C)=\begin{cases} 0 & \text{if } l=0 \text{ and } k>1, \\ 1 & \text{if } l>0 \text{ and } k=0, \\ \frac{NSP(G_c)}{NSP(FG_c)} & \text{otherwise.} \end{cases}$, where NSP is the number of simple paths in graph G_c, FG_c is the corresponding fully connected graph, and a simple path is a path in which each node occurs once at most.
OL_n [16]	OL_n= The average strength of the attributes, wherein the strength of an attribute is the average strength of the methods that reference that attribute. The strength of a method is initially set to 1 and is computed, in each iteration, as the average strength of the attributes that it references, where *n* is the number of iterations that are used to compute OL.
Cohesion Among Methods in a Class (CAMC) [17, 18]	CAMC= *a/kl,* where *l* is the number of distinct parameter types, *k* is the number of methods, and *a* is the summation of the number of distinct parameter types of each method in the class. Note that this formula is applied on the model that does not include the self parameter type that is used by all methods.
Normalized Hamming Distance (NHD) [18]	$NHD = 1 - \frac{2}{lk(k-1)}\sum_{j=1}^{l} x_j(k-x_j)$, where *k* and *l* are as defined above for CAMC, and x_j is the number of methods that have a parameter of type *j*.
Method-Method through Attributes Cohesion (MMAC) [19]	$MMAC(C)=\begin{cases} 0 & \text{if } k=0 \text{ or } l=0, \\ 1 & \text{if } k=1, \\ \frac{\sum_{i=1}^{l} x_i(x_i-1)}{lk(k-1)} & \text{otherwise.} \end{cases}$, where x_i is the number of methods that have a parameter or a return of type *j* (the type must match one of the attribute types).

LCOM1, LCOM2, TCC, LCC, DC_D, and DC_I are based on counting the number of method pairs that share or do not share common attributes, wherein two methods share an attribute when both of them reference the same attribute. LCOM3, LCOM4, CBMC, PCCC, and OL_n are based on measuring the connectivity of the class-representative graph. Coh is based on counting the number of attributes that are referenced by the methods in a class. LSCC, CC, and SCOM are based on measuring the degree of similarity between each pair of methods. Finally, CAMC, NHD, and MMAC use information about the relationship between the types of method parameters and the types of attributes in a class in order to estimate the degree of cohesion. These cohesion measures are well theoretically and empirically studied [4, 15, 20, 21, 22, 23, 24, 25, 26, 27, 28, 31, 32].

III. RELATED WORK

The problem of whether to include or exclude special methods is theoretically addressed for some of the considered metrics. The original definitions for LCOM1, LCOM2, LCOM3, LCOM4, and LCOM5 do not differentiate between the different types of methods, whereas TCC and LCC were originally defined to exclude constructors. Briand et al. [2] have suggested the exclusion of constructors and destructors when applying LCOM1, LCOM2, LCOM3, and LCOM4 because these metrics are based on counting the number pairs of methods that share common attributes; hence, the inclusion of constructors artificially increases their values. In addition, Briand et al. identified that the inclusion of access methods artificially influences the cohesion values that are obtained by using LCOM1, LCOM2, LCOM3, TCC, and LCC. Therefore, they have suggested either accounting for method invocations or excluding the access methods (in the case of applying TCC or LCC). Etzkorn et al. [29] have provided examples showing that when constructors are included, the LCOM2 value either decreases or remains the same, whereas the LCOM3 value either increases or remains the same. Badri and Badri [13] has suggested the application of both DC_D and DC_I when considering all public methods, which typically include all special methods. However, they did not justify their choice regarding the inclusion of special methods or study the effect of this inclusion on the obtained cohesion values. Chae et al. [14] have argued that special methods do not contribute to the cohesiveness of the class and, therefore, must be excluded when measuring cohesion so as to prevent any impact on the obtained cohesion values. Accordingly, they have suggested the exclusion of special methods when applying CBMC. Similarly, OL_2 was defined to measure cohesion while excluding special methods. Without providing any justifications, Counsell et al. [18] have suggested the application of CAMC and NHD when including special methods, and they left the analysis of the effect of these methods on their proposed cohesion measures open for further research. Fernandez and Pena [12] have argued that constructors and destructors do not always reference all attributes in a class. Therefore, these types of methods must be included in cohesion measurements in order to differentiate between the constructors/destructors that cause the cohesion values to increase or decrease. However, they argued that access methods must be excluded because they always reference a single attribute.

None of these works has qualitatively addressed in detail the effect of including special methods by discussing the cases in which the inclusion/exclusion of special methods causes cohesion values to increase, decrease, or remain the same. These works have merely discussed a general trend of change that might not always be true.

Al Dallal [4] empirically investigated the impact of including/excluding the constructors and access methods in cohesion measurement on (1) the cohesion values that were obtained, (2) refactoring decisions, and (3) the abilities of the metrics to predict faulty classes. The results of the empirical study regarding the effect of including/excluding the constructors and access methods on the cohesion values that were obtained by using the metrics under consideration are summarized in Table 2. In addition, the results showed that the cohesion values and the corresponding refactoring decisions that use most of the cohesion metrics that are under consideration are significantly changed when the special methods are included. It is suggested that the access methods must be excluded and the constructor methods must be included when using cohesion metrics to predict the classes that require refactoring. Finally, the results demonstrated that including/excluding special methods in cohesion measurement insignificantly changes the abilities of the cohesion metrics that are considered to predict faulty classes. This paper extends the work proposed by Al Dallal [30] in which only lack-of-cohesion metrics were considered.

TABLE 2:
SUMMARY OF THE EMPIRICAL RESULTS REGARDING THE IMPACT OF INCLUDING CONSTRUCTORS AND ACCESS METHODS IN COHESION MEASUREMENT ON COHESION VALUES

Metric	Impact of including constructors on metric values	Impact of including access methods on metric values
LCOM5, LSCC, CAMC, MMAC	Increase	Increase
LCOM3, LCOM4, CBMC, OL_2, PCCC	Increase	Decrease
CC, SCOM, Coh, TCC, DC_D	Decrease	Increase
LCOM1, LCOM2, LCC, DC_D, NHD	Decrease	Decrease

IV. COHESIVE RELATIONS OF SPECIAL METHODS

The cohesion of a class is determined by the extent to which the attributes and methods of the class are directly or indirectly related. Typically, the existence of constructors and destructors in a class increases the number of direct and indirect relations in a class. Such an outcome occurs for two reasons: First, the constructors and destructors reference most, if not all, of the class attributes, which introduces additional direct relations to the class between each of the constructors and destructors and the class' attributes. Second, the attributes that are referenced by the constructors and destructors are typically referenced by some other methods in the class. Therefore, the existence of the constructors and destructors in the class creates indirect relations between them and the methods that reference the attributes. For the rest of this paper, the discussion regarding the impact of including the constructors is applicable to the destructors as well because both types of methods are expected to reference the majority or all of the attributes.

The existence of access and delegation methods in a class increases the number of direct and indirect relations in a class due to the same two reasons that are mentioned above. However, since each access and delegation method typically references a single attribute, more relations are introduced by the constructors and destructors than by the access and delegation methods. Increasing the number of relations in a class by incorporating more attributes or methods does not necessarily increase a class's cohesion level. When assessing the cohesion level, both the existing relations and the missed possible relations must somehow be considered. For example, an access method has relations with the methods that reference the same accessed attribute, and it has no relations with the other methods. An access method references a single attribute; therefore, the number of methods that are related to the access method is typically less than the number of unrelated methods. As a result, the process of incorporating the access methods into the cohesion measurement is expected to reduce the obtained cohesion value.
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. Because access and delegation methods have the same characteristics, in terms of the number of referenced attributes (i.e., each access or delegation method references a single attribute), the discussion regarding the access methods also applies to delegation methods throughout the remainder of this paper.

The impact of including all types of special methods in cohesion measurement on the cohesion values that are obtained is determined by the strength of the impact of each type of special method. In addition, it depends on the number of constructors, destructors, and access and delegation methods that are under consideration. That is, if the inclusion of the constructors and destructors increases the cohesion value that is obtained using a specific metric by an amount that is greater than the decrease in the cohesion value which is caused by including the access and delegation methods, then the cohesion value increases and vice versa.

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.

The existing cohesion metrics use different approaches to measure cohesion. For example, some of the metrics count the number of existing and missed relations in a class, and some of them are based on averaging the number of relations in a class. Some of the metrics account only for direct relations, whereas others consider both direct and indirect relations. Some of the metrics measure cohesion, and others measure the lack of cohesion in a class. Therefore, the effect of including special methods in a cohesion measurement varies according to the cohesion metric itself and the type of special method that is being considered (e.g., a constructor or an access method).

V. Qualitative Analysis

Here, we qualitatively study the impact of including or excluding special methods on the values that are obtained by applying each of the 19 metrics that are under consideration. This study is based on analyzing the definitions and formulas that are proposed for the metrics. For each of the considered metrics, the cases in which the inclusion of constructors and access methods causes the metric value to increase or decrease are identified and analyzed as follows.

A. LCOM1

Unless a method shares an attribute with each of the other methods, including a method in the LCOM1 measurement causes the LCOM1 value to increase. This increase occurs because such an inclusion causes the number of pairs of methods that do not share attributes to increase. Ideally, the constructor shares an attribute with each other method, and, therefore, in this case, including the constructor does not affect the LCOM1 value. There are two cases in which including the constructor affects the value of LCOM1: (1) when the constructor references most of the attributes but not all of them, and some of the methods simultaneously only reference attributes that are not referenced by the constructor, or (2) when some of the methods do not reference any attributes. In either of these two unusual cases, including the constructor causes the LCOM1 value to increase because the constructor does not share attributes with the indicated methods.

Non-peer access methods do not share attributes, and an access method is not expected to share attributes with all other methods. Therefore, including the access methods causes the number of method pairs that do not share attributes and, consequently, the LCOM1 value to artificially increase. As a result, the recommendation is to include constructors and exclude access methods from the LCOM1 measurement.

B. LCOM2

When including a method in the LCOM2 measurement, the change in the LCOM2 value depends on the difference between the number of methods that do not share attributes with the included method and the number of methods that share attributes with the included method. Based on the earlier discussion in Section III.A, expectedly, this difference has a considerable negative value for a constructor method and a considerable positive value for an access method. Therefore, the inclusion of constructors or access methods is expected to artificially decrease or increase the LCOM2 value, respectively. For the unusual cases that were described in Section III.A, the inclusion of a constructor can increase the LCOM2 value. As a result, when applying LCOM2, the recommendation is to exclude constructors and access methods from the cohesion measurement.

C. LCOM3 and LCOM4

Typically, including the constructor makes the class representative graph connected, which causes the LCOM3 value to decrease to one. This is based on the typical expectation that each method in the class is referencing at least an attribute. However, when the class-representative graph is already connected, the inclusion of the constructor does not change the LCOM3 value. In addition, when a method in the class does not share an attribute with any other method and also does not share an attribute with the constructor because of the same reasons that were indicated in Section III.A, the inclusion of the constructor does not change the isolation situation of that method, although it may change the isolation situations of other methods that do not feature the same unusual cases. As a result, the inclusion of constructors typically decreases the LCOM3 value of the class.

The inclusion of an access method causes the LCOM3 value to increase only when, unusually, the access method does not share attributes with any other method in the class. Otherwise, typically, the LCOM3 value remains the same when including access methods. Note that the inclusion of an access method does not complicate the LCOM3 computation because this metric is based on simply counting the number of disjointed components. LCOM3 and LCOM4 share the same arguments because constructors and access methods do not typically invoke other methods. As a result, when applying either LCOM3 or LCOM4, the recommendation is to include access methods and exclude constructors from the cohesion measurement.

D. LCOM5

Including the constructor does not change the numerator part of the LCOM5 formula, which is given in Section II, when the constructor accesses all attributes.

This is because the value of *a* (i.e., the summation of the number of distinct attributes that are referenced by each method in the class) in the formula is increased by *l* (i.e., the number of attributes in the class), and the value of *k* (i.e., the number of considered methods) is increased by one; hence, $(k+1)\times l-(a+l)=kl-a$. The inclusion of a constructor slightly increases the numerator when the constructor accesses most of the attributes. However, the inclusion of a constructor causes the denominator part to increase by *l*. Therefore, the inclusion of a constructor potentially causes the LCOM5 value to artificially decrease.

The inclusion of an access method causes the numerator and denominator parts of the formula to increase by the values of (l-1) and *l*, respectively. This means that the inclusion of access methods causes the LCOM5 to slightly increase or decrease depending on the values of *k*, *l*, and *a*. Note that including the access methods does not complicate the LCOM5 computation because LCOM5 is based on simply counting the number of distinct attributes that are referenced by each method. As a result, the recommendation is to include access methods and exclude constructors from the LCOM5 measurement.

E. LSCC

LSCC can be calculated it two ways, either by using the direct formula given in Section II or 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. 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.

F. 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. As a result, the impact of including the constructor in the CC computation does not follow a specific trend. Instead, it depends on the relations between the nonspecial methods. For example, assume that a class *C* consists of three attributes: a constructor and two other methods. If each of the two methods references the same single attribute, then the similarity between the two methods will have a value of 1. If the constructor is excluded from the CC computation, then the CC value will equal the value of the similarity between the two methods (i.e., a value of 1). If the constructor references the three class attributes, then the similarity between the constructor and each of the two methods will have a value of 0.33, as defined by the CC metric. In this case, the value of the CC metric will be 0.55 (i.e., [0.33+0.33+1]/2); hence, the value of the CC decreases from the value of 1 to the value of 0.55. In another hypothetical example, assume that a class has the same number of attributes and methods as class *C*; the constructor references all the attributes, and each of the attributes references two attributes, such that one of the attributes is commonly referenced by the two methods. In this case, the similarity between the two methods is 0.33, and the similarity between the constructor and each method is 0.67. If the constructor is excluded from the CC computation, then the CC value will equal the value of the similarity between the two nonspecial methods (i.e., 0.33), whereas the CC value increases from 0.33 to 0.56 (i.e., [0.67+0.67+0.33]/2) when the constructor is considered in the CC computation.

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. In addition, 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.

G. SCOM

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. In addition, 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.

H. Coh

Typically, including the constructor artificially increases the average number of distinct attributes that are accessed by each method in the class, which consequently artificially increases the Coh value. In contrast, access methods artificially decrease the Coh value because they decrease the average number of distinct attributes that are accessed by each method in the class. As a result, when applying Coh, the recommendation is to exclude both constructors and access methods from the cohesion measurement.

I. TCC, LCC, DC_D, and DC_I

The constructor potentially references a number of attributes to a greater extent than any other non-special methods. Therefore, the chance for a constructor method to be directly or transitively connected to other methods is higher than that for non-special methods. As a result, the inclusion of the constructor artificially increases the TCC and LCC values. In this case, the LCC value is expected to increase at a rate that is higher than that for TCC. This is because the constructor is not only expected to be directly connected to each other method, but it also causes some other methods to be transitively connected to one another.

Pairs of access methods are not directly connected, except for peer access methods. In addition, the number of methods to which an access method is directly connected is potentially fewer than that for other non-special methods. Therefore, including access methods is expected to artificially decrease the TCC value.

In usual cases, the attributes are not supposed to be only referenced by access methods but also by other non-special methods. In this case, the access method is directly connected to the peer access method and the special methods that directly reference the same attribute. In addition, it can be transitively connected to other non-special methods and non-peer access methods in a manner that greatly depends on how the other methods are connected to one another. For example, assume that a class has an access method *m1*, two other methods *m2* and *m3*, and two attributes *a1* and *a2*. If *a1* is only referenced by *m1*, and if *a2* is referenced by both *m2* and *m3*, then including the access method in the LCC measurement decreases the LCC value from 1 to 0.33. In another hypothetical scenario, if *a1* is referenced by *m1* and *m2*, and if *a2* is only referenced by *m3*, then including the access method in the LCC computation increases the LCC value from zero to 0.33. As a result, the effect of including the access methods in the LCC value depends on how the other methods are related to each other and to the access methods. Therefore, the access method can considerably increase or decrease the value of the LCC.

The difference between DC_D and TCC as well as DC_I and LCC is in the consideration of the references of attributes that derive from method invocations. As previously indicated, constructors and access methods typically do not invoke other methods. Therefore, the above discussion regarding TCC and LCC similarly applies to DC_D and DC_I, respectively. As a result, when applying TCC, LCC, DC_D, or DC_I, the recommendation is to exclude both constructors and access methods from the cohesion measurement.

J. CBMC

The CBMC value is proportional to the minimum number of glue methods (i.e., the minimum number of methods for which removal causes the class-representative graph to become disjointed) and is inversely proportional to the number of considered methods. When the constructor is considered, it will be the best candidate method to be included in the minimum set of glue methods because, typically, the constructor references all attributes. In this case, if the graph, without including the constructor, is connected and then the constructor is included, the minimum set of glue methods will include the constructor and the glue methods of the original graph. Unless the original graph is fully connected, the minimum number of glue methods of the original graph will be fewer than the number of methods that are represented in the graph. Therefore, the CBMC value increases when a method is added, and that method increases the number of glue methods by one, which is the case when a constructor that references all attributes is added. It is important to note that when one or more of the methods do not reference attributes, and, thus, the class-representative graph is disjointed and the CBMC value of the class is zero, the inclusion of the constructor does not change the CBMC value even if the constructor references all of the attributes.

The inclusion of nodes that represent access methods to a connected class-representative graph does not change the minimum set of glue methods. This is because, in this case, the attribute that is referenced by the access method is also referenced by some other methods. Therefore, the removal of the access method does not make the connected graph disjointed. However, the inclusion of access methods increases the number of considered methods and, thus, decreases the CBMC value. As a result, the recommendation is to exclude both constructors and access methods from the CBMC cohesion measurement.

K. PCCC

PCCC is based on counting the number of simple paths from each node to each other node in the class-representative graph. The number of simple paths that are initiated from a node that represents a method greatly depends on the number of attributes that are referenced by that method. Therefore, the inclusion of the constructor increases the average number of simple paths and consequently increases the PCCC value because the constructor references most or all of the attributes in the class. Oppositely, the inclusion of access methods decreases the average number of simple paths and, consequently, decreases the PCCC value because each access method references a single attribute. As a result, the recommendation is to exclude both constructors and access methods from the PCCC measurement.

L. OL_n

The OL_n value depends on the average strength of the class attributes. The strength of an attribute depends on the total strengths of the methods that reference this attribute. Recursively, the strength of each method depends on the total strengths of the attributes that are referenced by the method. Consequently, the constructor potentially has the highest strength because it references all or most of the attributes, and, therefore, when the constructor is considered, the attributes that are connected to the constructor have an average strength that is higher than that when the constructor is excluded. In other words, including the constructor potentially increases the OL_n value.

An access method is connected to one attribute, and, therefore, it has a relatively low strength, which consequently lowers the strength of the accessed attribute. On average, when the access methods are considered, the attributes that are referenced by access methods are expected to have lower strengths than those when the access methods are excluded. Therefore, the inclusion of access methods potentially decreases the OL_n value. Similar to CBMC, the OL_n value for a class with a disjointed representative graph is zero. Therefore, OL_n has effects that are similar to those that are specified for CBMC when the nodes that represent the constructors or access methods are added to an already disjointed graph. As a result, the recommendation is to exclude both constructors and access methods from the OL_n measurement.

M. CAMC

CAMC is based on an assumption that the types of method parameters are potentially the same as the types of the attributes that are referenced by the methods. Based on this assumption, the inclusion of a constructor that has parameters that are used to initialize all of the attributes increases the average number of distinct types of parameters of all methods, which increases the CAMC value.

A setter method has one parameter type, whereas a getter method does not have any parameter type. Therefore, the inclusion of access methods potentially decreases the CAMC value. As a result, the recommendation is to exclude both constructors and access methods from the CAMC measurement.

N. NHD

NHD is based on averaging the agreement degree between each pair of methods, wherein the agreement degree of a pair of methods is inversely proportional to the number of parameter types that are used by only one of the methods. The assumption about the relationship between the parameter and attribute types is similar to that used in CAMC. The constructor is expected to have many more parameter types than any of the other methods. Therefore, the agreement between the constructor and other methods is expected to be low, which consequently reduces the NHD value.

Similarly, the matching between the parameter types of the access methods and other methods is expected to be low. However, the agreement does not only consider the types of the parameters of the two considered methods, but it also considers all of the types of parameters of all methods. Access methods and other methods agree in that they do not have other parameter types aside from their own parameters. In addition, if the number of such parameters is relatively high, as expected, the agreement between the access methods and other methods is expected to be relatively high, which increases the NHD value. As a result, the recommendation is to exclude both constructors and access methods from the NHD measurement.

O. MMAC

This metric has the same assumption as that for CAMC and is based on measuring the similarities between methods in terms of parameter type matches if these types also match the attribute types. The average similarity between a constructor and each other method is expected to be higher than that between each pair of other methods.

Therefore, including constructors potentially increases the MMAC value.

Opposite to LSCC, which is based on the same similarity concept, setter methods that have the same parameter types have some degree of similarity, although they reference different attributes. MMAC does not only consider the types of the parameters but also considers the return type of the method. Therefore, getter methods that have the same return type feature some degree of similarity among themselves and between themselves and other setter methods that have the same parameter types, including peer access methods. Finally, access methods for which the parameter types match the parameter types of other methods have some degree of similarity, although they may reference different attributes as long as the attributes have the same types. All of these factors potentially cause the MMAC value to increase if the average similarity that is caused by these factors when the access method is included is higher than the average similarity that is featured in the class before including the access methods. Otherwise, the MMAC value increases or remains the same. As a result, the impact of including the access methods on the MMAC value cannot be generally determined, and such an inclusion considerably changes the value of MMAC in some typical cases. In conclusion, the recommendation is to exclude both constructors and access methods from the MMAC measurement.

VI. Conclusions And Future Work

This paper qualitatively explores the impact of including or excluding special methods in cohesion measurement with respect to the cohesion values that are obtained. The study is based on analyzing the impact of including the special methods on the definitions and formulas that are applied via 19 existing cohesion metrics. Table 3 summarizes the expected changes in the cohesion values when the special methods are included in cohesion measurement. The qualitative analysis that is reported in this section leads to the following observations:

1. When using most of the considered metrics, the cohesion (lack of cohesion) value of a class increases (decreases) when the constructors and destructors are considered in the cohesion measurement.
2. When using most of the considered metrics, the cohesion (lack of cohesion) value of a class decreases (increases) when the access and delegation methods are considered in the cohesion measurement.

These two observations are consistent with the intuition that was discussed in Section IV. However, the qualitative results that are reported in the current paper and the empirical analysis results that are reported by Al Dallal [4] and summarized in Section III do not always identically match. The reason behind this difference is the fact that qualitative results are based on the ideal cases in which each constructor and destructor references all of the attributes in the class of interest, which is not always true in practice. Al Dallal [4] reported that, for the classes that were selected in the empirical study, the constructors and the access methods reference an average of 61.7% and 37.1% of the attributes, respectively.

The recommendation to include or exclude the special methods in the cohesion measurement greatly depends on the application of interest for which the cohesion value will be used. For example, when using the cohesion values to predict whether the class requires refactoring, the constructors and destructors must be included. This is because the constructors and destructors are used to partially provide the class' intended functionality. On the other hand, the access and delegation methods must be excluded because they are used not to provide functionalities, but instead, to support encapsulation. Therefore, the access and delegation methods are not expected to weaken the class design and cause the class to be refactored. Changing the cohesion value of a class because of the inclusion or exclusion of access or delegation methods can alter the value and mislead the refactoring decision.

TABLE 3:
EXPECTED CHANGES IN COHESION VALUES BASED ON THEORETICAL GROUNDS.

Metric	Impact of including constructors and destructors on metric values	Impact of including access and delegation methods on metric values
LCOM1	Increases for unusual cases but, otherwise, remains the same	Increases
LSCC, SCOM, Coh, TCC, DC_D, CBMC, PCCC, OL_n, and CAMC	Increases	Decreases
LCOM2 and NHD	Decreases	Increases
LCOM3 and LCOM4	Decreases	Increases for unusual cases but, otherwise, remains the same
LCOM5	Decreases	Slightly increases or decreases
LCC, DC_I, and MMAC	Increases	Depends on other factors
CC	Depends on other factors	Decreases

In the future, we plan to qualitatively study the impact of including/excluding directly- and indirectly-inherited attributes and methods on cohesion measurements that apply the same set or larger sets of metrics. Similarly, we plan to qualitatively investigate the impact of considering method invocations on cohesion measurement. Our final goal is to qualitatively explore the best scenario in which to apply each specific metric, in terms of including/excluding special methods, inherited attributes and methods, and method invocations.

Acknowledgment

The author would like to acknowledge the support of this work by Kuwait University Research Grant WI04/08.

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)*, 2012, Vol. 21, No. 2, pp. 8:1-8:34.

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

[3] J. Bieman and B. Kang, 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] J. Al Dallal, 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*, 2012, Vol. 85, No. 5, pp. 1042-1057.

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

[6] 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, 1991, Vol. 26, No. 10, pp. 197-211.

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

[8] 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.

[9] 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.

[10] B. Henderson-sellers, *Object-Oriented Metrics Measures of Complexity*, Prentice-Hall, 1996.

[11] 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.

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

[13] L. Badri and M. Badri, A Proposal of a new class cohesion criterion: an empirical study, *Journal of Object Technology*, 2004, Vol. 3, No. 4, pp. 145-159.

[14] H. S. Chae, Y. R. Kwon, and D. Bae, A cohesion measure for object-oriented classes, *Software—Practice & Experience*, 2000, Vol. 30, No. 12, pp.1405-1431.

[15] J. Al Dallal, Fault prediction and the discriminative powers of connectivity-based object-oriented class cohesion metrics, *Information and Software Technology*, 2012, Vol. 54, No. 4, pp. 396-416.

[16] X. Yang, *Research on Class Cohesion Measures*, M.S. Thesis, Department of Computer Science and Engineering, Southeast University, 2002.

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

[18] 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)*, 2006, Vol. 15, No. 2, pp.123-149.

[19] 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.

[20] L. C. Briand, J. Wust, J. Daly, and V. Porter, Exploring the relationship between design measures and software quality in object-oriented systems, *Journal of System and Software*, 2000, Vol. 51, No. 3, pp. 245-273.

[21] L. C. Briand, J. Wüst, and H. Lounis, Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs, *Empirical Software Engineering*, 2001, Vol .6, No. 1, pp. 11-58.

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

[23] J. Al Dallal, Improving the applicability of object-oriented class cohesion metrics, *Information and Software Technology*, 2011, Vol. 53, No. 9, pp. 914-928.

[24] J. Al Dallal, Measuring the discriminative power of object-oriented class cohesion metrics, *IEEE Transactions on Software Engineering*, 2011, Vol. 37, No. 7, pp. 788-804.

[25] J. Al Dallal, Incorporating transitive relations in low-level design-based class cohesion measurement, *Software: Practice and Experience*, in press, 2012.

[26] J. Al Dallal, Constructing models for predicting extract subclass refactoring opportunities using object-oriented quality metrics, *Information and Software Technology*, 2012, Vol. 45. No. 10, pp. 1125-1141.

[27] J. Al Dallal, The impact of inheritance on the internal quality attributes of java classes, *Kuwait Journal of Science and Engineering*, in press, 2012.

[28] J. Al Dallal and S. Morasca, Predicting object-oriented class reusability using internal quality attributes, submitted for publication in *Empirical Software Engineering*, 2012.

[29] L. Etzkorn, C. Davis, and W. Li, A practical look at the Lack of Cohesion in Methods metric, *Journal of Object-Oriented Programming*, 1998, Vol. 11, No. 5, pp. 27-34.

[30] J. Al Dallal, Theoretical analysis for the impact of including special methods in lack-of-cohesion computation, *1st World Conference on Innovation and Software Development*, Istanbul, Turkey, 2011.

[31] J. Al Dallal, Object-oriented class maintainability prediction using internal quality attributes, submitted for publication in *Empirical Software Engineering*, 2012.

[32] J. Al Dallal, The impact of incorporating special methods and transitive relations into cohesion measurement on object-oriented class reusability prediction, submitted for publication in *Journal of Systems and Software*, 2012.

Jehad Al Dallal received his PhD in Computer Science from the University of Alberta in Canada and was granted the award for best PhD researcher. He is currently working at Kuwait University in the Department of Information Science as an Associate Professor. Dr. Al Dallal has completed several research projects in the areas of software testing, software metrics, and communication protocols. In addition, he has published more than 60 papers in conference proceedings and ACM, IEEE, IET, Elsevier, Wiley, and other journals. Dr. Al Dallal was involved in developing more than 20 software systems. He also served as a technical committee member of several international conferences and as an associate editor for several refereed journals.