The Impact of Accounting for Special Methods in the Measurement of Object-Oriented Class Cohesion on Refactoring and Fault Prediction Activities

Jehad Al Dallal
Department of Information Science
Kuwait University
P.O. Box 5969, Safat 13060, Kuwait
j.aldallal@ku.edu.kw

Abstract
Class cohesion is a key attribute that is used to assess the design quality of a class, and it refers to the extent to which the attributes and methods of the class are related. Typically, classes contain special types of methods, such as constructors, destructors, and access methods. Each of these special methods has its own characteristics, which can artificially affect the class cohesion measurement. Several metrics have been proposed in the literature to indicate class cohesion during high- or low-level design phases. The impact of accounting for special methods in cohesion measurement has not been addressed for most of these metrics. This paper empirically explores the impact of including or excluding special methods on cohesion measurements that were performed using 20 existing class cohesion metrics. The empirical study applies the metrics that were considered to five open-source systems under four different scenarios, including (1) considering all special methods, (2) ignoring only constructors, (3) ignoring only access methods, and (4) ignoring all special methods. This study empirically explores the impact of including special methods in cohesion measurement for two applications of interest to software practitioners, including refactoring and predicting faulty classes. The results of the empirical studies show that the cohesion values for most of the metrics considered differ significantly across the four scenarios and that this difference significantly affects the refactoring decisions, but does not significantly affect the abilities of the metrics to predict faulty classes.

Keywords: object-oriented design, class quality, class cohesion, cohesion metric, special methods, refactoring, fault prediction.

1. Introduction

Over the past decade, object-oriented programming languages, such as C++ and Java, have been widely used in the software industry. The basic unit of design in object-oriented programs is the class. The members of a class include its attributes and methods. Several factors have to be addressed to evaluate the quality of class design, including cohesion, coupling, and complexity (Fenton and Pfleeger 1998). Class cohesion refers to the extent to which the methods and attributes in a class are related (Briand et al. 1998).

In object-oriented programming, some methods are typically provided to support the encapsulation key feature of the object-oriented paradigm. 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 to inquire about their values (Deitel 2005). We refer to these methods as special methods, and we classify them into four different types, including constructors, destructors, access methods, and delegation methods. A constructor is invoked when creating an object of the class, and the constructor typically initializes most or all of the attributes in the class. In contrast, a destructor is invoked at the end of the object life cycle and typically deinitializes most or all of the attributes. In contrast to constructors, destructors are not supported by some object-oriented programming languages, such as Java. Access methods are classified as either setters or getters. A setter method initializes a single attribute to the reference/value that is passed through the method's parameter. 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 (e.g., whether the value is positive/negative or whether a stack/queue is full/empty). As a result, special methods feature extreme scenarios regarding the percentage of the referenced attributes in a class. That is, constructors and destructors potentially reference a relatively high percentage of attributes, whereas access and delegation methods potentially reference a relatively low percentage of attributes.

All existing cohesion metrics are directly or indirectly based on observations of the attributes referenced by the methods, although they apply different approaches to measure cohesion. Therefore, in some cases, the inclusion of special methods may influence the cohesion values so strongly that they become clearly incorrect cohesion indicators, in comparison to human intuition. In such cases, the obtained cohesion values could lead to incorrect class quality assessments and refactoring decisions. To solve this problem, special methods must be excluded if the obtained cohesion values indicate class quality incorrectly or lead to incorrect refactoring decisions. Therefore, after a cohesion metric is defined and before the metric is used in practice, the impact of including or excluding special methods on the values obtained using the metric must be theoretically and empirically investigated. Guidelines based on the obtained results must be provided to the practitioners to direct them toward the best scenario in which the metric can be applied.

Several class cohesion metrics that apply different approaches, use different pieces of information, and measure different cohesion aspects have been proposed in the literature. The problem of including/excluding special methods in cohesion measurement has been qualitatively addressed for some of the existing metrics, albeit in brief (e.g., Bieman and Kang 1995, Briand et al. 1998, Etzkorn et al. 1998, Chae et al. 2000, Zhou et al. 2004). As far as we know, the impact of including/excluding special methods has not been empirically studied for any metric, although this impact can affect the use of the cohesion metrics in the applications of interest for software practitioners. In practice, cohesion is among the quality attributes that are useful only when they are shown to be related to other quality attributes of interest for software users and practitioners, such as fault proneness, reusability, and maintainability (Morasca 2009). In addition, measuring cohesion is useful when it is demonstrated to be related to design enhancement tasks such as refactoring (Czibula and Serban 2006, De Lucia et al. 2008, Al Dallal and Briand 2010b). As a result, addressing the problem of including/excluding special methods is not

of interest for software practitioners unless the impact of including/excluding special methods in cohesion measurement on other quality attributes and design improvement tasks is investigated, which is the main goal of this paper.

Refactoring is the process of changing an existing object-oriented software code to enhance its internal structure while preserving its external behavior (Fowler 1999). A class is predicted to require refactoring when a value that is based on design metrics which measure quality attributes, such as cohesion, is found to be less than a certain threshold (e.g., De Lucia et al. 2008, Czibula and Serban 2006, Al Dallal and Briand 2010b). Although the impact of including/excluding special methods in cohesion measurement can have a crucial impact on the use of cohesion metrics in refactoring activities, as far as we know, this impact has not been empirically studied.

In this paper, we qualitatively discuss the expected impact of including/excluding different types of special methods on cohesion measurement. The goal of this discussion is to demonstrate, with the aid of examples, that the impact of including special methods on class cohesion values greatly depends on the type of the special method considered. In addition, we empirically investigated the impact of including/excluding special methods from 2,245 classes of five open-source Java systems when applying several cohesion metrics to the cohesion values obtained. In this empirical study, 20 cohesion metrics that apply different cohesion measurement approaches are considered. In addition, we address the four possible scenarios of (1) excluding all special methods, (2) including constructors and excluding access methods, (3) excluding constructors and including access methods, and (4) including all special methods. Destructors are not considered because they are not supported by Java, and delegation methods are not accounted for because they are difficult to detect automatically. To empirically investigate whether the changes in the cohesion values affect refactoring decisions, several thresholds are selected to examine the percentage of classes for which the refactoring decisions are changed when the special methods are included in the cohesion measurement. A statistical technique was applied to explore whether the changes in the refactoring decisions were significant. The results indicate that a high percentage of the methods in a class are special, and it is therefore important to address the problem of including/excluding special methods. The results also demonstrate that the cohesion values and the corresponding refactoring decisions using most of the cohesion metrics considered are significantly changed when the special methods are included. According to the above discussion, these results suggest 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.

Highly cohesive classes are expected to be less prone to faults. This expectation has been confirmed by several empirical studies (e.g., Briand et al. 1998, Briand et al. 2001, Gyimothy et al. 2005, Aggarwal et al. 2007, Marcus et al. 2008) in which cohesion metrics were involved in models used to predict faulty classes. The impact of including/excluding special methods on the abilities of the cohesion metrics to predict faulty classes has not previously been thoroughly studied or addressed. In this paper, we

empirically investigated the impact of including/excluding special methods from classes of the same five open-source Java systems when applying the same 20 cohesion metrics on the abilities of the metrics to predict faulty classes. To perform this study, we collected fault data for the classes in the considered software systems from publicly available fault repositories, obtained the cohesion values for the considered classes using the 20 considered cohesion metrics and the four considered scenarios, statistically analyzed the relationship between the cohesion values and the presence of faults in the classes, and statistically compared the fault prediction results across the four scenarios. The results show that including/excluding special methods in cohesion measurement insignificantly changes the abilities of the cohesion metrics considered to predict faulty classes.

In summary, the major contributions of this paper are as follows:

1. It empirically explores the importance of addressing the problem of including/excluding special methods and the impact of including/excluding special methods on the cohesion values that are obtained from using 20 different metrics on five open-source systems.
2. It empirically investigates the effects of including/excluding special methods from classes of five open-source systems on refactoring decisions based on applying 20 different cohesion metrics.
3. It empirically explores the impact of including/excluding special methods from classes of five open-source systems on the ability of 20 different cohesion metrics to detect faulty classes.

This paper is organized as follows. Section 2 reviews related work. Section 3 qualitatively discusses the effects of including/excluding special methods from cohesion measurements on changing the cohesion values. Section 4 demonstrates the empirical effects of including/excluding special methods from cohesion measurements on the cohesion values obtained. Sections 5 and 6 report and discuss the results of the empirical studies that investigate the impact of including/excluding special methods on the refactoring decisions and the abilities of the metrics to predict faulty classes. Section 7 lists validity threats to the empirical studies. Finally, Section 8 concludes the paper and discusses future work.

2. Related Work

Several metrics have been proposed in the literature to measure cohesion in object-oriented systems at different abstraction levels, including method metrics (e.g., Al Dallal 2009) and class metrics (Briand et al. 1998). The class cohesion metrics can be classified according to different perspectives, such as the types of interactions considered, the development phase during which they are applicable, and the types of methods considered. In this paper, we consider 20 metrics, including LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, LSCC, CC, SCOM, Coh, TCC, LCC, DC_D, DC_I, ICBMC, 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. 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, ICBMC, 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 to estimate the degree of cohesion. The theoretical validation for most of these metrics has been studied by Al Dallal (2010).

Class Cohesion Metric	Definition/Formula
The Lack of Cohesion in Methods (LCOM1) (Chidamber and Kemerer 1991)	LCOM1= Number of pairs of methods that do not share attributes.
LCOM2 (Chidamber and Kemerer 1994)	*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 (Li and Henry 1993)	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 (Hitz and Montazeri 1995)	Similar to LCOM3, and additional edges are used to represent method invocations.
LCOM5 (Henderson-Sellers 1996)	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) (Al Dallal and Briand 2010a)	$LSCC(C) = \begin{cases} 0 & \text{if } l = 0 \text{ and } k > 1, \\ 1 & \text{if } (l > 0 \text{ and } k = 0) \text{ or } k = 1, \\ \dfrac{\sum_{i=1}^{l} x_i(x_i - 1)}{lk(k-1)} & \text{otherwise.} \end{cases}$, 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) (Bonja and Kidanmariam 2006)	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: $Similarity(i,j) = \dfrac{\lvert I_i \cap I_j \rvert}{\lvert I_i \cup I_j \rvert}$, where I_i and I_j are the sets of attributes that are referenced by methods *i* and *j*, respectively.
Class Cohesion Metric (SCOM) (Fernandez and Pena 2006)	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: $Similarity(i,j) = \dfrac{\lvert I_i \cap I_j \rvert}{\min(\lvert I_i \rvert, \lvert I_j \rvert)} \cdot \dfrac{\lvert I_i \cup I_j \rvert}{l}$, where *l* is the number of attributes.
Coh (Briand et al. 1998)	Coh=*a*/*kl*, where *a*, *k*, and *l* have the same definitions above.
Tight Class Cohesion (TCC) (Bieman and Kang 1995)	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*.

Table 1: Definitions of the class cohesion metrics considered.

Class Cohesion Metric	Definition/Formula
Loose Class Cohesion (LCC) (Bieman and Kang 1995)	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) (Badri 2004)	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.
Degree of Cohesion-Indirect (DC_I) (Badri 2004)	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) (Chae et al. 2000)	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.
Improved Cohesion Based on Member Connectivity (ICBMC) (Xu and Zhou 2001, 2003)	ICBMC(G)=F_c(G)×F_s(G), where F_c(G)=|M(G)|/|N(G)|, M(G)=the number of edges in the cut set of G, N(G)=the number of non-special methods represented in graph G multiplied by the number of attributes, $F_s(G)=[\sum_{i=1}^{2} ICBMC(G^i)]/2$, and cut set is the minimum set of edges such that their removal causes the graph to become disjointed.
Path Connectivity Class Cohesion (PCCC) (Al Dallal 2011d)	$PCCC(C)=\begin{cases}0 & \text{if } l=0 \text{ and } k>1,\\ 1 & \text{if } l>0 \text{ and } k=0,\\ \frac{NSP(G_c)}{NSP(FG_c)} & \text{otherwise.}\end{cases}$, 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 (Yang 2002)	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) (Bansiya et al. 1999, Counsell et al. 2006)	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) (Counsell et al. 2006)	$NHD=1-\frac{2}{lk(k-1)}\sum_{j=1}^{l} x_j(k-x_j)$, 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) (Al Dallal and Briand 2010b)	$MMAC(C)=\begin{cases}0 & \text{if } k=0 \text{ or } l=0,\\ 1 & \text{if } k=1,\\ \frac{\sum_{i=1}^{l} x_i(x_i-1)}{lk(k-1)} & \text{otherwise.}\end{cases}$, 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).

Table 1 (continuation): Definitions of the class cohesion metrics considered.

The problem of whether to include or exclude special methods is qualitatively addressed for some of the metrics considered. 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. (1998) have suggested the exclusion of constructors and destructors when applying LCOM1, LCOM2, LCOM3, and LCOM4 because these metrics are based on counting the number of pairs of methods that share common attributes; hence, the inclusion of constructors artificially increases their values. In addition, Briand et al. observed that the inclusion of access methods artificially influences the cohesion values that are obtained by using LCOM1, LCOM2, LCOM3, TCC, and LCC; therefore, they suggested either accounting for method invocations or excluding the access methods (in the case of applying TCC or LCC). Etzkorn et al. (1998) 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 (2004) 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 cohesion values obtained. Chae et al. (2000) have argued that special methods do not contribute to the cohesiveness of the class and that they must therefore be excluded when measuring cohesion to prevent any impact on the cohesion values obtained. Accordingly, they have suggested the exclusion of special methods when applying CBMC. Similarly, ICBMC and OL_2 were defined to measure cohesion while excluding special methods. Without providing any justifications, Counsell et al. (2006) have suggested the application of CAMC and NHD when including special methods, and they left the analysis of the effects of these methods on their proposed cohesion measures open for further research. Fernandez and Pena (2006) have argued that constructors and destructors do not always reference all attributes in a class and must therefore be included in cohesion measurements to differentiate between the constructors/destructors that cause the cohesion values to increase or decrease; however, access methods must be excluded because they always reference a single attribute. Al Dallal and Briand (2010b) and Al Dallal (2011c) have empirically studied the impact of including special methods in LSCC and LCOM1 measurement, respectively, on the abilities of the metrics in predicting faulty classes. However, they did not consider all the scenarios considered in this paper and they considered less than half of the number of the classes considered in the empirical study reported in this paper. In addition, none of the reviewed works have empirically investigated the impact of including/excluding special methods either on the cohesion values obtained or on the practical applications of cohesion metrics, such as refactoring. All of these issues are qualitatively and empirically addressed in this paper in the following sections.

3. Qualitative Analysis

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, 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 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 example, 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. LCOM2 is an example for a metric that its lack-of-cohesion value decreases when including constructors in cohesion measurement. LCOM2 is defined as the difference between the number of method pairs that do not share common attributes and those pairs that do share common attributes. 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. This difference has a considerable negative value for the constructor method because such a method typically shares attributes with most or all of the other methods that access attributes. Therefore, the inclusion of constructors is expected to artificially decrease the LCOM2 value.

The existence of access and delegation methods in a class increases the number of direct and indirect relations in a class, for the same two reasons mentioned above for the constructors and destructors. 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 necessary increase a class's cohesion level. When assessing the cohesion level, both the existing relations and the missed possible relations must be considered somehow. 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 related to the access method is typically less than the number of unrelated methods. As a result, incorporating access methods into the cohesion measurement is expected to reduce the obtained cohesion value.

For example, including the access methods causes the number of method pairs that do not share attributes and, consequently, the LCOM1 value, to artificially increase (i.e., the cohesion indicated by LCOM1 decreases). LSCC is another example for a metric that its cohesion value decreases when including access methods in cohesion measurement. LSCC can be calculated in two ways, either by using the direct formula given in Section 2 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 an access method and any other nonpeer access method is zero, and is low (i.e., has a value of $1/l$) between peer access methods (i.e., access methods that reference the same attribute). In addition, the similarity between an access method and any other method has a maximum value of $1/l$. Therefore, the inclusion of access methods is expected to artificially reduce the LSCC value.

The impact of including all types of special methods in cohesion measurement on the cohesion values obtained is determined by the strength of the impact of each type of special method and the number of constructors, destructors, and access and delegation methods considered. That is, if the inclusion of the constructors and destructors increases the cohesion value obtained using a specific metric by an amount that is greater than the decrease in the cohesion value caused by including the access and delegation methods, then the cohesion value increases, and vice versa. The next section empirically examines the amount and significance of changes in the cohesion values for 20 cohesion metrics when including/excluding special methods to explore whether the changes can have impacts on the applications for which the cohesion metrics are useful.

4. Empirical Analysis for the Cohesion Values

We empirically studied the impact of including special methods on the values that were obtained using the 20 cohesion metrics summarized in Table 1. The goal of this study was to empirically determine whether the changes in the cohesion values are significant, and consequently, to empirically determine whether it is important for software practitioners to pay attention to the problem of including/excluding special methods when using cohesion metrics in supporting software quality decisions. Four scenarios were considered: (1) excluding all special methods, (2) including constructors and excluding access methods, (3) including access methods and excluding constructors, and (4) excluding all special methods.

Five Java open-source software systems from different domains were selected, including Art of Illusion v.2.5 (Illusion 2010), FreeMind v.0.8.0 (FreeMind 2010), GanttProject v.2.0.5 (GanttProject 2010), JabRef v.2.3 beta 2 (JabRef 2010), and Openbravo v.0.0.24 (Openbravo 2010). Art of Illusion consists of 463 classes and is a 3-D modeling, rendering, and animation studio system. FreeMind consists of 379 classes and is a hierarchical editing system. GanttProject consists of 435 classes and is a project-scheduling application that features resource management, calendaring, and importing and exporting (MS Project, HTML, PDF, spreadsheets). JabRef consists of 540 classes and is a graphical application for managing bibliographical databases. Openbravo consists of 428 classes and is a point-of-sale application that is designed for touch screens. We selected these five open-source systems from http://sourceforge.net. The restrictions that were placed on the choice of these systems were that they (1) be implemented using Java, (2) be relatively large in terms of the number of classes, (3) be from different domains, and (4) have available source code and fault repositories. Except for FreeMind, the other systems were used by Al Dallal and Briand (2010a, 2010b) and Al Dallal (2011a, 2011b, 2011d) in empirical validation studies for newly proposed metrics and cohesion aspects but without accounting for the inclusion/exclusion of the special methods. From this perspective, the current study is unique and does not overlap or replicate any of the previous studies. As will be detailed in Section 6, for each of the classes, we considered the version of the code before correcting the first detected fault. This consideration was not necessary for the analysis in this section, but it was necessary to perform the fault prediction analysis reported in Section 6. Therefore, we considered

this version of the classes to consistently apply all of the analyses reported in this paper on the same version of the classes.

We developed our own Java tool to automate the cohesion measurement process for Java classes using the 20 cohesion metrics that are considered in this paper. The tool (1) analyzed the Java source code, (2) extracted the source code information that is required by the 20 metrics to calculate cohesion values, (3) identified the constructors and access methods, and (4) calculated cohesion values for each of the four versions of each class. Each version features one of the scenarios that were indicated earlier in this section. The tool was applied to each class in the five systems, which implied applying the tool on the 2,245 considered classes. The tool only measures cohesion of Java classes, and therefore, the empirical studies reported in this paper do not investigate the impact of including destructors, as they are not supported by Java. In addition, it is difficult to automate the detection of delegation methods because delegation methods are highly dependent on the semantics of the problem. The empirical studies consider both constructors and access methods because these two types of methods can be detected automatically. The tool considers a method to be a constructor if the method does not have a declared return type and the name of the method matches the name of the class. An access method can be a getter or a setter. The tool considers a method to be a setter for a variable *var* if the method has a name set*Var*, has a void return type, and assigns the value passed as a parameter to the variable *var*. A method is considered as a getter for a variable *var* if the method has the name get*Var*, has a return type that matches the type of the variable *var*, and returns the value of *var*. Because destructors and delegation methods were not considered in the empirical studies, the rest of this section and Section 5 will use the term "special methods" to refer only to the considered types of special methods (i.e., constructors and access methods).

For each of the four considered scenarios, Table 2 lists the descriptive statistics for the number of methods in the classes considered, including the minimum, the 25% quartile, the mean, the median, the 75% quartile, the maximum value, and the standard deviation. In addition, Figures 1, 2, and 3 depict the relative frequencies of the number of constructors, access methods, and special methods considered, respectively, that were included in the classes considered.

Scenario	Min	Max	25%	Med	75%	Mean	Std. Dev.
Excluding all considered special methods	0	125	1	3	7	4.93	6.51
Including constructors	0	126	2	4	8	5.90	6.68
Including access and delegation methods	0	125	1	4	8	6.20	7.95
Including all considered special methods	1	126	2	5	9	7.17	8.15

Table 2: Descriptive statistics for the methods of the classes considered.

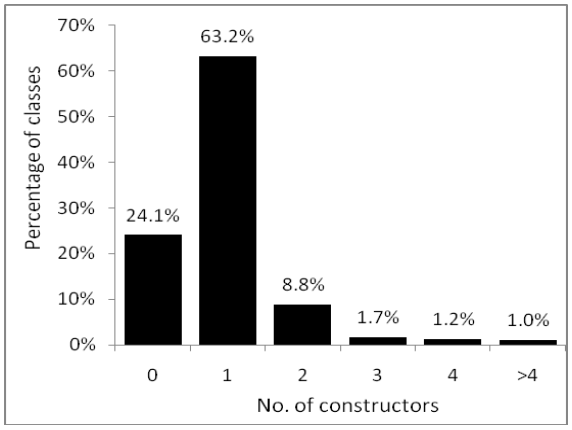

Figure 1: The distribution of constructors.

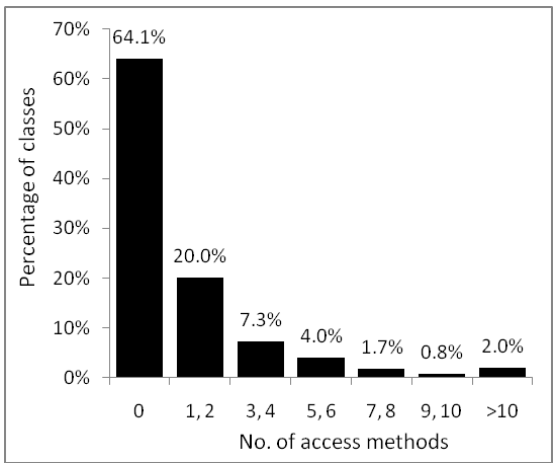

Figure 2: The distribution of access methods.

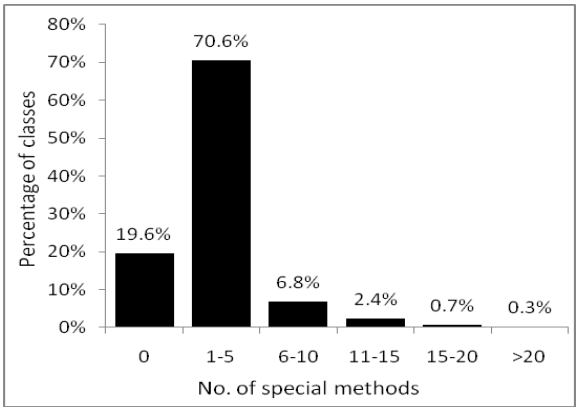

Figure 3: The distribution of the considered special methods.

Figure 1 demonstrates that some of the classes (24.1%) do not have constructors, most of the classes (63.2%) have a single constructor, and each of the rest of the classes (12.7%)

has multiple constructors. When considering constructors, Table 2 demonstrates that the average number of methods in a class increases by 19.7%, from 4.93 to 5.90.

Figure 2 demonstrates that more than half of the classes do not include access methods. When considering access methods, Table 2 demonstrates that the average number of methods in a class increases by 25.8%, from 4.93 to 6.2. Finally, Figure 3 demonstrates that most of the considered classes (80.4%) include special methods, and Table 2 shows that when considering all special methods, the average number of methods in a class increases considerably, by 45.4%, from 4.93 to 7.17. This high percentage for special methods indicates the importance of studying their impact on the cohesion values that are obtained by the metrics. In addition, the high percentage of special methods potentially indicates that a metric will incorrectly indicate the degree of cohesion when all of the methods are considered if that metric is supposed to only consider nonspecial methods.

Some of the metrics considered are undefined for some classes. For example, TCC, LCC, DC_D, DC_I, CC, SCOM, LCOM5, and NHD are undefined for classes that consist of fewer than two methods. In addition, LCOM5, SCOM, Coh, and NHD are undefined for classes without any attributes. The cohesion metric applicability problem has been thoroughly studied by Al Dallal (2011a). Our empirical studies do not attempt to compare the results across the 20 metrics. Instead, the studies seek to compare the results across the four scenarios for each metric alone. Therefore, when considering each metric, we obtained the cohesion results for all of the considered classes in all of the four scenarios, excluding the classes for which at least one of the scenarios resulted in an undefined cohesion value when the considered metric was applied. We then compared the respective results of the remaining classes. The classes for which at least one of the scenarios resulted in an undefined cohesion value were excluded so that we could perform the same analysis on the same set of classes and, therefore, could compare the respective results in an unbiased manner.

For each of the four scenarios considered, we obtained descriptive statistics for each cohesion measure, including the minimum, the 25% quartile, the mean, the median, the 75% quartile, the maximum value, and the standard deviation. Because of the space limitation, Table 3 depicts only the number of classes for which each metric is defined and the means of the cohesion values of the metrics considered when they were applied in the four aforementioned scenarios. The complete set of descriptive statistics is listed in Appendix A.

Metric	No. of considered classes	Scenario 1: Excluding special methods	Scenario 2: Including constructors			Scenario 3: Including access methods			Scenario 4: Including special methods		
		Mean	Mean	Mean Diff.	Z	Mean	Mean Diff.	Z	Mean	Mean Diff.	Z
LCOM1	1651	34.20	38.61	-4.41	-14.21	51.79	-17.59	-8.02	57.27	-23.1	-9.5
LCOM2	1651	27.31	29.56	-2.25	-8.48	40.42	-13.11	-6.70	42.73	-15.4	-7.4
LCOM3	2245	1.65	1.55	0.10	7.53	1.79	-0.14	-6.50	1.62	<0.01	**1.7**
LCOM4	2245	1.64	1.54	0.10	7.13	1.77	-0.13	-6.13	1.62	<0.01	**1.3**
LCOM5	1281	0.88	0.77	0.11	18.44	0.83	0.05	9.85	0.75	0.1	18.1
LSCC	2245	0.32	0.32	0.01	**0.85**	0.31	0.02	3.19	0.30	<0.01	4.8
CC	1281	0.20	0.22	-0.02	-4.43	0.20	<0.01	**0.58**	0.21	<0.01	-2.9
SCOM	1281	0.23	0.26	-0.03	-7.43	0.23	<0.01	**1.85**	0.26	<0.01	-6.0
Coh	1579	0.39	0.43	-0.03	-11.11	0.39	<0.01	**0.43**	0.42	<0.01	-6.7
CBMC	1579	0.18	0.17	0.01	**1.24**	0.16	0.02	7.41	0.16	<0.01	4.5
ICBMC	1579	0.16	0.14	0.02	4.03	0.14	0.02	6.53	0.13	<0.01	6.7
OL_2	1579	0.17	0.17	<0.01	**0.85**	0.16	0.02	7.19	0.15	<0.01	4.6
PCCC	2245	0.39	0.28	0.11	17.17	0.34	0.05	11.62	0.25	0.1	19.6
TCC	1651	0.34	0.36	-0.01	-4.01	0.33	0.01	3.36	0.35	<0.01	**-1.7**
LCC	1651	0.38	0.41	-0.03	-6.47	0.40	-0.02	-6.58	0.43	-0.1	-11.8
DC_D	1651	0.35	0.36	-0.01	-3.89	0.34	0.01	3.45	0.35	<0.01	**-1.5**
DC_I	1651	0.38	0.41	-0.03	-6.37	0.40	-0.02	-6.50	0.44	-0.1	-11.6
CAMC	2245	0.71	0.59	0.12	39.28	0.67	0.04	18.95	0.55	0.2	45.1
NHD	1372	0.52	0.58	-0.06	-15.77	0.57	-0.05	-14.62	0.62	-0.1	-21.2
MMAC	2245	0.16	0.10	0.06	6.58	0.15	0.01	**1.73**	0.09	0.1	8.7

Table 3: Paired z-test results and partial descriptive statistics for the metrics considered.

To investigate whether the changes in the cohesion values were statistically significant, we applied the paired z-test (Hines et al. 2003), a standard statistical technique applied to compare two independent sets of samples. We applied this technique because the cohesion values before and after considering the special methods are independent and the number of selected classes is large enough. In our context, to apply the paired z-test for a metric, (1) the difference between the cohesion values before and after the change has to be calculated for each class, (2) the means and standard deviation values of these differences have to be obtained, and (3) the Z value has to be calculated by multiplying the mean of the differences by the square root of the number of classes and dividing the result by the standard deviation of the differences. We applied the typical significance threshold ($\alpha = 0.05$) to decide whether the differences between the cohesion values were significant. The corresponding critical Z value is 1.96. The differences were considered significant when the absolute value of Z was greater than that of the critical Z. We applied this analysis to compare the values of Scenarios 2, 3, and 4 to the values of Scenario 1. The analysis results, including the mean of the differences and the Z values, are listed in Table 3.

The results in Table 3 lead to the following observations:

1. The cohesion values for most of the metrics considered changed significantly when any type of special method was included in the cohesion measurement. The few insignificantly changed values are highlighted in boldface in Table 3. The next two sections will explore the effect of the changes in cohesion values on applications of interest so that software practitioners can determine whether the special methods must be excluded when cohesion metrics are intended to be used in supporting these applications.
2. The value and sign of the mean of the differences vary from one metric to another depending on the definition of the metric, as discussed in Section 3. However, the qualitative and empirical analysis results do not always identically match. For example, qualitatively, when constructors are included in LCOM1 measurement, the cohesion values will not be changed. This relationship is based on the ideal case in which each constructor references all of the attributes in the class of interest, which is not always true in practice. For the selected classes, we found that the constructors, the access methods, and the nonspecial methods reference an average of 61.7%, 37.1%, and 39.3% of the attributes, respectively.
3. The results for a few metrics demonstrate that the cohesion values were not significantly changed when some types of special methods were added. The z-test analysis considers the signs of the changes in the cohesion values. For example, if the value changes positively for a class and negatively by the same amount for another class, the mean difference will be zero, and the value will be considered to be changed insignificantly. Therefore, in a z-test analysis, the values are considered to be insignificantly changed if the magnitudes of the changes are small or if the changes have different signs such that the changes are eliminating each other when considering all of the samples together. Therefore, to explore the reason why a few of the metrics show insignificant change when including special methods, we calculated the percentage of the average absolute change in the cohesion values for all of the classes. For example, to obtain the percentage of the average absolute change for LSCC in Scenario 2, we (1) calculated, for each class considered, the absolute difference between the LSCC values in Scenarios 1 and 2, (2) obtained the average of these differences over all classes, and (3) found the percentage of this average to the mean value for Scenario 1 (0.32). To obtain the percentage of average absolute changes for Scenario 3, we considered the absolute differences between the values obtained in Scenarios 1 and 3. Similarly, to obtain the percentage of average absolute changes for Scenario 4, we considered the absolute differences between the values obtained in Scenarios 1 and 4. Because of space limitations, the results are listed in Appendix A and show that for the few changed values whose changes are insignificant, the average absolute change ranges from 11.4% to 44.4%. These changes are greater than the corresponding values for some other metrics that have significant changes. This observation demonstrates that the insignificant changes in the values shown for some of the metrics in Table 3 are because of the sign difference of the change in the cohesion values across the classes considered.

4. The results of the fourth scenario are affected by the strength and significance of the impact of each type of special method. If the obtained values of a specific metric in both the second and third scenarios significantly change with the same trend (i.e., if the mean values in both scenarios either increase or decrease), then the obtained cohesion values in the fourth scenario will significantly change with the same trend of both the second and third scenarios. For example, the results show that including each of constructors and access methods causes the LCOM1 values to increase significantly. Therefore, in the fourth scenario, when both types of special methods are considered together, the LCOM1 values follow the same trend of increasing significantly. The same observation applies for LCOM2, LCOM5, ICBMC, PCCC, LCC, DC_I, CAMC, and NHD. When the cohesion values for both the second and third scenarios follow the same trend of change, but one of them is changed significantly and the other changes only slightly, then the values in the fourth scenario will change significantly, along with the same trend as the other two scenarios. This observation applies for LSCC, CBMC, OL_2, and MMAC. On the other hand, if the second and third scenarios have opposite effects on the cohesion values (i.e., if one of them causes the cohesion values to increase and the other one causes the values to decrease), then in the fourth scenario, the two opposite effects will partially eliminate each other. Therefore, if the two opposite effects have close strengths, then the change of the cohesion values in the fourth scenario can be insignificant (e.g., as for the LCOM3, LCOM4, TCC, and DC_D metrics). Otherwise, the change in the cohesion values will follow the trend of the scenario that has the stronger effect (e.g., as for CC, SCOM, and Coh metrics).

In summary, the empirical results reported in this section demonstrate the importance of addressing the problem of including or excluding special methods because these results show that a high percentage of classes include special methods. In addition, the results indicate that including special methods in cohesion measurement significantly changes the cohesion values for most of the metrics considered, which might affect the use of the cohesion metrics in practice. The next two sections investigate the impact of including the special methods in cohesion measurements on two applications of interest for software practitioners. Based on the discussion provided in Section 1, if the inclusion of constructors in the cohesion measurement significantly affects the results of an application to which the cohesion was empirically determined to be related, then the constructors should be included in the cohesion measurement. Otherwise, the application will be missing a factor that is related to the cohesion aspects. On the other hand, if the inclusion of access methods in the cohesion measurement significantly affects the results of an application to which the cohesion was empirically determined to be related, then the access methods should be excluded from the cohesion measurement. Otherwise, the application will be incorrectly influenced by factors that are unrelated to the cohesion aspects. Two practical applications have been empirically determined to be related to cohesion and will be considered in the following two sections: (1) predicting classes that require refactoring, and (2) predicting classes that include faults.

5. Impact of including special methods on refactoring decisions

Refactoring aims to improve code maintainability and understandability, and it refers to the process of changing an existing object-oriented software code to enhance its internal structure while preserving its external behavior (Fowler 1999). Automating design changes, reducing testing efforts, simplifying designs, assisting validation, and experimenting with new designs are among the refactoring benefits identified by Tokuda and Batory (2001). Researchers use metrics that measure quality attributes, such as cohesion, to guide refactoring activities (e.g., De Lucia et al. 2008, Czibula and Serban 2006, Al Dallal and Briand 2010b). These studies investigated several refactoring methods, such as *Extract Class*, which refers to creating a new class to include related attributes and methods that currently exist in the original class (Fowler 1999). In addition, these studies compared values based on factors (including cohesion) with specific thresholds to decide whether a class of interest required *Extract Class* refactoring (i.e., to decide whether a class should be split into several classes to enhance overall software cohesion).

By their natures, some types of special methods, such as access methods, have simple implementations because these special methods are used not to provide functionalities, but instead, to support encapsulation. Therefore, these special methods are not expected to weaken the class design and cause the class to be refactored. The refactoring decision must not be based on methods that include these types of special methods. Changing the cohesion value of a class because of the inclusion or exclusion of access methods can alter the value, compared with the refactoring threshold; therefore, it can mislead the refactoring decision. On the other hand, developers typically do not use other types of special methods, such as constructors and destructors, solely to initialize or deinitialize attributes; they also use them to perform some required functionalities when the object is created or destroyed. Therefore, there is no difference between these special methods and the nonspecial methods, in terms of being incorporated into cohesion indication and refactoring decisions. Typically, all these methods provide functionalities, and they can have any degree of implementation complexity. As a result, these types of special methods should be included in cohesion measurements if their exclusion causes the cohesion values to change significantly.

As reported in Section 4, including special methods in cohesion measurement was empirically found to significantly change the cohesion values obtained using most of the metrics considered. Therefore, the values compared with the refactoring threshold and, consequently, the decisions for whether to refactor a class are changed when special methods are included in a cohesion measurement. Here, we empirically investigate whether the refactoring decisions significantly change when special methods are included. Although we realize that there are other factors to consider in addition to cohesion, using multiple criteria for class feature assignment is out of the scope of this paper.

In this empirical study, we considered the same classes, cohesion metrics, and scenarios regarding the inclusion/exclusion of special methods as those considered in the empirical

study illustrated in Section 4. For each scenario and for each cohesion metric, we counted the number of classes that had cohesion values below the threshold (above the threshold for the lack of cohesion metrics). These classes were predicted to require *Extract Class* refactoring. For each metric, we selected the threshold to decide whether the class required *Extract Class* refactoring to be the same as the best experimentally selected threshold found by De Lucia et al. (2008) (i.e., one-fifth of the mean value reported in Table 3). In addition, we selected another arbitrary threshold to be the cohesion mean value, reported in Table 3, to investigate whether the refactoring decisions based on some metrics were unaffected by the inclusion of special methods regardless of the selected refactoring threshold. For each of the two thresholds and for each scenario, we recorded whether each of the considered classes required refactoring.

We applied a paired z-test analysis to investigate whether the differences were significant between the refactoring decisions based on the cohesion values obtained in Scenario 1 and those based on the cohesion values obtained in each of Scenarios 2, 3, and 4. Similar to the paired z-test analysis reported in Section 4, we applied the typical significance threshold ($\alpha = 0.05$) to decide whether the differences between the refactoring decisions were significant. The corresponding critical Z value was 1.96. The differences were considered significant when the absolute value of Z was greater than the absolute value of the critical Z. We applied this analysis to compare the refactoring decisions that are based on Scenarios 2, 3, and 4 with the decisions based on Scenario 1. The analysis results, including the percentage of classes predicted to require *Extract Class* refactoring, the mean of the differences, and the Z values for both thresholds, are listed in Tables 4 and 5. In these tables S_1, S_2, S_3, and S_4 refer to Scenarios 1, 2, 3, and 4, respectively. The changed refactoring decisions that were insignificant are highlighted in boldface in Tables 4 and 5.

The results in Tables 4 and 5 show that the *Extract Class* refactoring decisions significantly changed when including any type of special methods in cohesion measurement using most of the metrics considered, regardless of the selected refactoring thresholds. The results for the two selected thresholds show that although the refactoring decisions based on some of the metrics were insignificantly affected by the inclusion of some types of special methods for some specific thresholds, these refactoring decisions were significantly changed using the same metrics when using other refactoring thresholds. Based on the discussion provided earlier in this section, this observation indicates that the refactoring decisions that are based on a cohesion measurement that does not consider constructors are unreliable, whereas the refactoring decisions that are based on a cohesion measurement that considers access methods are unreliable. It is important to note that the refactoring decisions for both selected thresholds using CBMC and ICBMC were changed insignificantly when including constructors, as reported in Tables 4 and 5. The same observation applies to OL_2 when including both constructors and access methods. However, this observation does not mean that the refactoring decisions for these metrics are always insignificantly changed when including special methods, regardless of the thresholds. We tried several other threshold values, such as

three times the mean value, and we found that the refactoring decisions were changed significantly for these metrics when selecting these other refactoring thresholds.

Metrics	Percentage of classes predicted to require refactoring				Paired z-test results for refactoring decisions					
					S_1 vs. S_2		S_1 vs. S_3		S_1 vs. S_4	
	S_1	S_2	S_3	S_4	Diff	Z	Diff	Z	Diff	Z
LCOM1	46.9	53.8	54.9	60.1	-0.07	-3.94	-0.08	-4.57	-0.13	-7.67
LCOM2	44.5	47.1	50.6	50.5	-0.03	**-1.54**	-0.06	-3.56	-0.06	-3.49
LCOM3	96.2	97.1	98.1	97.9	-0.01	**-1.66**	-0.02	-3.77	-0.02	-3.36
LCOM4	96.2	97.1	98.1	97.9	-0.01	**-1.66**	-0.02	-3.77	-0.02	-3.36
LCOM5	93.7	95.2	95.3	96.3	-0.02	**-1.73**	-0.02	**-1.82**	-0.03	-2.99
LSCC	51.2	44.7	50.7	44.2	-0.07	-4.37	-0.01	**-0.36**	-0.07	-4.73
CC	39	26.9	30.1	20.3	-0.12	-6.57	-0.09	-4.76	-0.19	-10.56
SCOM	41.5	27.8	36	23.3	-0.14	-7.38	-0.06	-2.88	-0.18	-10.07
Coh	17.5	11.9	12.7	10.4	-0.06	-4.44	-0.05	-3.79	-0.07	-5.73
CBMC	76.5	73.6	76.6	73.8	-0.03	**-1.89**	0.00	**0.04**	-0.03	**-1.73**
ICBMC	77.6	75.5	78.6	76.4	-0.02	**-1.43**	0.01	**0.65**	-0.01	**-0.85**
OL_2	76.6	73.7	77.1	74.2	-0.03	**-1.85**	0.01	**0.34**	-0.02	**-1.57**
PCCC	52.6	59.2	56.3	62.3	0.07	4.43	0.04	2.46	0.10	6.61
TCC	44.8	38.2	40.8	35.1	-0.07	-3.86	-0.04	-2.36	-0.10	-5.75
LCC	44.6	38	40.5	34.6	-0.07	-3.86	-0.04	-2.36	-0.10	-5.90
DC_D	43.7	37.3	39.7	34.1	-0.06	-3.80	-0.04	-2.37	-0.10	-5.70
DC_I	43.6	37	39.5	33.7	-0.07	-3.87	-0.04	-2.40	-0.10	-5.89
CAMC	99.2	98.9	98.4	98	0.00	**1.07**	0.01	2.47	0.01	3.43
NHD	88	98	91.3	98.4	-0.10	-10.48	-0.03	-2.83	-0.10	-11.02
MMAC	75	77.1	71.2	73.7	0.02	**1.64**	-0.04	-2.83	-0.01	**-0.99**

Table 4: Paired z-test results when the refactoring threshold equals one-fifth of the mean value.

In summary, the results of this empirical study show that it is risky to rely on refactoring decisions that are based on cohesion metrics when access methods are included and constructors are excluded from the cohesion measurement. As a result, it is recommended to include constructors and exclude access methods in cohesion measurement when using cohesion metrics to predict classes that require refactoring because this inclusion/exclusion can incorrectly affect the refactoring decisions. The next section provides an example for an application of interest that will be empirically shown to be unaffected by the inclusion of the special methods in cohesion measurement, which shows that the decision of whether to include special methods in cohesion measurement depends on the application for which the cohesion metrics are used in practice.

Metrics	Percentage of classes predicted to require refactoring				Paired z-test results for refactoring decisions					
					S_1 vs. S_2		S_1 vs. S_3		S_1 vs. S_4	
	S_1	S_2	S_3	S_4	Diff	Z	Diff	Z	Diff	Z
LCOM1	19.1	21.3	25.4	28.8	-0.02	-5.99	-0.06	-10.43	-0.10	-6.57
LCOM2	20	21.4	23.9	26	-0.01	-3.37	-0.04	-7.65	-0.06	-4.01
LCOM3	30.6	25	36.4	28.1	0.05	9.20	-0.05	-8.83	0.03	**1.87**
LCOM4	30	24.4	35.5	25.7	0.06	9.07	-0.05	-8.27	0.02	**1.75**
LCOM5	54.9	37.5	49.1	33.3	0.17	15.83	0.06	7.48	0.22	11.25
LSCC	67.4	66.3	68.8	68.8	-0.01	**-1.49**	0.01	2.49	0.01	**1.02**
CC	67.8	62.5	67.5	62.8	-0.05	-5.75	0.00	**-0.31**	-0.05	-2.62
SCOM	67.1	59.1	68.7	60.2	-0.08	-8.22	0.02	2.27	-0.07	-3.66
Coh	60.4	52.4	59.9	53.5	-0.08	-9.55	0.00	**-0.82**	-0.07	-3.89
CBMC	78.7	77.3	79.7	78.5	-0.01	**-1.93**	0.01	4.14	0.00	-0.13
ICBMC	81.3	81.6	83	84	0.00	**0.49**	0.02	5.34	0.03	2.02
OL_2	79.1	77.1	80.2	78.6	-0.02	-2.83	0.01	4.14	-0.01	**-0.35**
PCCC	62.3	72.3	67.7	75.3	0.10	13.69	0.05	10.28	0.13	9.46
TCC	60.9	57.2	59.9	56	-0.04	-5.31	-0.01	**-1.75**	-0.05	-2.86
LCC	59.1	55	55.1	50	-0.04	-6.03	-0.04	-6.69	-0.09	-5.30
DC_D	60.5	56.9	59.6	55.7	-0.04	-5.26	-0.01	**-1.63**	-0.05	-2.82
DC_I	59	54.6	55.2	50	-0.04	-6.41	-0.04	-6.62	-0.09	-5.19
CAMC	49.5	29.6	43.7	24.8	0.20	23.55	0.06	11.70	0.25	17.73
NHD	55.9	64.1	65.2	72.2	-0.08	-8.47	-0.09	-11.05	-0.16	-9.04
MMAC	26.3	33.7	29.8	37.7	0.07	9.92	0.03	5.98	0.11	6.43

Table 5: Paired z-test results when the refactoring threshold equals the mean value.

6. Impact of including special methods on predicting faulty classes

Researchers have provided empirical evidence showing that cohesion metrics strongly contribute to models used in predicting the presence of faults in classes (e.g., Briand et al. 1998, Briand et al. 2001, Gyimothy et al. 2005, Aggarwal et al. 2007, Marcus et al. 2008).

Typically, some special methods, such as access methods, are less complex and smaller in size than other (nonspecial) methods. In addition, these special methods can be fully or semi-automated by many existing programming editors and tools (e.g., Websphere 2011 and Together 2011). As a result, these special methods are, ideally, less fault-prone than other (nonspecial) methods. Therefore, hypothetically, the inclusion of access methods in the cohesion measurement must not change the decisions based on the cohesion metrics regarding whether the class is predicted to be faulty. Other special methods, such as the constructors and destructors, can have complex pieces of code and large sizes. Therefore, these special methods must be dealt with just like any of the other nonspecial methods, and they can affect the fault prediction abilities. To support these hypotheses empirically, we applied logistic regression (Hosmer and Lemeshow 2000), a standard statistical technique that is based on maximum likelihood estimation and that has been widely applied to the prediction of fault-prone classes (Briand et al. 1998, Briand et al. 2001, Gyimothy et al. 2005, Marcus et al. 2008). The application of other analysis techniques,

such as those discussed by Briand and Wuest (2002), Subramanyam and Krishnan (2003), and Arisholm et al. (2009), is outside the scope of this paper. We applied this analysis to study the relationship between the values of the collected metrics when applying each of the four scenarios and the extent to which a class is prone to faults. The regression analysis here is not intended to be used to build a fault prediction model because, in this case, other factors, such as coupling and complexity, also have to be considered (Arisholm et al. 2009). Instead, our goal is to investigate whether the inclusion of special methods in cohesion measurement affects the prediction of the faulty classes, as determined by the cohesion metrics considered. In other words, the analysis is intended to explore which scenario is to be considered when including the metric in a fault prediction model.

In logistic regression, the variables are classified as either dependent and explanatory or independent. The former only takes discrete values and is binary in the context in which we predict fault-prone classes. The latter is used to explain and predict dependent variables. In our context, the dependent variable indicates the presence of one or more faults in a class, whereas the independent variables are the cohesion metrics that are used to predict the presence of faults in the class. There are two types of logistic regression analyses: univariate and multivariate. In univariate regression, the fault prediction capability of each metric is separately investigated, whereas in multivariate regression, the combined fault prediction of several cohesion metrics is explored, which is out of the scope of this paper. A comparison of the metrics in terms of their fault prediction powers is not our goal here. Instead, our analysis intends to compare, for each metric, the fault prediction results when each of the four scenarios are applied so as to investigate which scenario is the best for that metric in terms of fault prediction power. In other words, univariate regression analysis is applied to investigate, for each metric, whether the consideration of special methods in class cohesion measurement is better from a fault prediction perspective than the case in which the special methods are excluded.

We collected fault data for the classes in the software systems considered from publicly available fault repositories. The developers of the systems considered used an online version control system (VCS) to keep track of the changes, called revisions, performed on the source code of the system. Each revision is associated with a report that includes the revision description and a list of classes involved in this change. Working separately, two research assistants each manually traced the description of each revision and identified the ones performed because of detected faults. One of the research assistants has a B.Sc. in computer science and six years of experience in software development activities and the other has a B.Sc. and a Masters degree, both in computer science, and two years of experience as a research assistant. The developers of the considered systems used phrases, such as *fix bug*, *fix mistake*, *fix problem*, or *corrected*, to show that a given revision was due to a detected fault. In 21 days, the research assistants traced a total of 15,283 available online revisions. The author of this paper compared the manual results, noticed a few differences, rechecked the results in which the two assistants differed, and chose the correct ones based on the author's experience in analyzing and understanding the descriptions associated with the revisions. The lists of classes involved in changes due

to detected faults were used to count the number of faults in which each class was involved. Out of the 2,245 classes considered, 1228 classes were found to include at least one fault. Among these faulty classes, 958 classes were found to have a single fault. The total number of faults reported for all of the classes considered was 1,734. We classified each class as being fault-free or as having at least one fault because only a small percentage of classes contained two or more faults. For each revision, the VCS tracking system reported the source code of the involved classes before and after the modification. For each class involved in a revision, we used the VCS available data to collect the source code of the class before the first detected fault was corrected. We considered the version containing the faulty classes because the cohesion has to be measured for the version of the class from before the faults were corrected.

When applying univariate regression analysis, several factors can be used to compare the results, as follows. The estimated regression coefficient indicates the strength of the impact of the metric on the probability of a fault being detected in a class, where a larger absolute value of the coefficient indicates a stronger 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. As shown in Appendix A, the metrics considered had different standard deviations for different scenarios. Therefore, to help compare the coefficients, we standardized the explanatory variables by subtracting the mean and dividing by the standard deviation, which resulted in an equal variance of 1.0 for each variable. In this case, the resulting coefficients are standardized, and they represent the variation in the standard deviations of the dependent variable when there was a change of one standard deviation in their corresponding independent variables. 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; a larger p-value indicates a larger confidence interval for the coefficient. To decide whether a metric is a statistically significant fault predictor, we applied the typical significance threshold ($\alpha = 0.05$). We adjusted the significance threshold using the Bonferroni adjustment procedure (Abdi 2007) so as to avoid the typical problem of type-I error rate inflation in the context of multiple tests. The adjusted threshold is α/n, where n is the number of compared metrics. In our case, a metric was determined to not be a statistically significant fault predictor if its p-value was greater than $0.05/20 = 0.0025$.

Precision and recall are traditional evaluation criteria (Olson and Delen 2008) that are used to evaluate the prediction accuracy of logistic regression models. *Precision* measures the percentage of the faulty classes that are correctly classified as faulty and is defined as the number of classes that are correctly classified as faulty divided by the total number of classes that are classified as faulty. *Recall* measures the percentage of the faulty classes that are correctly or incorrectly classified as faulty and is defined as the number of classes that are correctly classified as faulty divided by the actual number of faulty classes. A probability threshold has to be selected to predict whether or not a class is faulty. As in Briand et al.'s study (2000), we classified a class as faulty if its predicted probability of containing a fault was higher than a selected threshold, such that the percentage of classes that are classified as faulty is roughly the same as the percentage of

classes that are actually faulty. Because precision and recall are subject to change as the selected threshold changes, we used the receiver operating characteristic (ROC) curve (Hanley and McNeil 1982). This curve evaluates the performance of a prediction model regardless of any particular threshold, and therefore, it is often considered to be a better evaluation criterion compared with standard precision and recall. In the context of fault prediction problems, the ROC curve graphically represents the ratio of classes that are correctly classified as faulty versus the ratio of classes that are incorrectly classified as faulty at different thresholds. The ability of the model to correctly rank classes as faulty or nonfaulty is represented by the area under the ROC curve. A perfect model that correctly classifies all classes has an ROC area of 100%. The larger the ROC area, the better the model is at classifying classes. To obtain a more realistic assessment of the predictive ability of the metrics, we used cross-validation, which is a procedure in which the data set is partitioned into *k* subsamples. The regression model is then built and evaluated *k* times. Each time, a different subsample is used to evaluate the precision, recall, and ROC area of the model, and the remaining subsamples are used as training data to build a regression model.

Because of space limitations, Table 6 depicts only the ROC area results for each metric applied to each of the four scenarios. The results for the other evaluation factors are reported in Appendix B. Here, we rely on the ROC area results to compare the integrity of the individual metrics in predicting faulty classes because, as mentioned earlier, the ROC area is a better evaluation criterion compared with standard precision and recall.

Metric	ROC area			
	Scenario 1: Excluding special methods	Scenario 2: Including constructors	Scenario 3: Including access methods	Scenario 4: Including special methods
LCOM1	64	65.6	65	66
LCOM2	63.3	63.8	65.2	65.2
LCOM3	52.9	53.1	51	52.6
LCOM4	53.4	53.6	51.5	53
LCOM5	63.8	65.6	66.2	67.2
LSCC	57	58.6	57.9	59.1
CC	62.8	64.3	65.7	66.8
SCOM	62.4	64.2	65.5	66.6
Coh	65.7	66.5	67.5	66.9
CBMC	58.3	59.2	58.6	59.3
ICBMC	58.8	59.2	59	59.3
OL_2	58.6	59.4	59.3	59.4
PCCC	63.8	59.9	64.7	59.7
TCC	51.6	53.6	46.1	52
LCC	53.4	53.8	52.8	54
DC_D	52.1	53.7	45.2	51.8
DC_I	54	54.9	53.1	54.7
CAMC	62.7	66.1	63.4	66.8
NHD	57.4	59.9	56.8	59.4
MMAC	49.4	47.6	48.2	46.2

Table 6: Univariate logistic regression results.

The results shown in Table 6 and Appendix B demonstrate that the considered metrics' fault prediction abilities are slightly different across the four scenarios. The slight improvement in the prediction abilities, for most of the considered metrics, is more evident in the second scenario than in the third scenario—possibly because the regression models of these metrics were able to detect some of the faults that existed in the constructors of the considered classes. We were unable to investigate this possibility because the developers, although they reported the faulty classes, did not specify the exact locations (within the classes) of most of the faults. To investigate whether the changes in the prediction results were significant, for each class, we recorded whether the analysis predicted the class to include faults. We recorded this prediction when each metric was applied and when each of the four scenarios was considered. We applied the paired z-test to explore whether the changes in the recorded prediction results were significant when the special methods were included in the cohesion measurement. Similar to the paired z-test analysis reported in Sections 3 and 4, we applied the typical significance threshold ($\alpha = 0.05$) to decide whether the differences between the fault prediction results were significant. The corresponding critical Z value was 1.96. We applied this analysis to compare the prediction results that are based on Scenarios 2, 3, and 4 with the prediction results based on Scenario 1. Except for CAMC, when each type of special methods was considered, all of the Z values obtained for all of the other metrics were below the critical Z value and all of their *p* values were above the typical significance threshold. This observation indicates that the changes in the fault prediction abilities of the metrics considered were insignificantly changed when the special methods were included in the cohesion measurement.

In conclusion, the decision about whether to include special methods when using a cohesion metric in a class fault prediction model does not depend on the change of the cohesion value caused by this inclusion.

7. Threats to validity

The reported empirical studies have several internal and external threats that may restrict the generality and limit the interpretation of the results. These threats are detailed as follows.

7.1. Internal validity

To detect faulty classes or to predict the classes that require refactoring, several factors other than cohesion must be considered (Arisholm et al. 2009). However, the present studies were not aimed at detecting faulty classes or predicting classes that require refactoring. Instead, our studies intended to investigate whether there is empirical evidence that including or excluding special methods when applying cohesion metrics can badly affect the applications for which cohesion metrics were considered useful in practice.

In the refactoring empirical study, two thresholds were selected to demonstrate the impact of including special methods in the cohesion measurement on refactoring

decisions. The first threshold was based on an existing experimental study, and the second threshold (i.e., the mean value) was based on a commonly statistical representative value. Although the first threshold is based on an experiment with subjective generalizability and the second threshold is not necessarily suitable for refactoring decisions, the selection of these thresholds did not affect the general obtained results and conclusions. The empirical study demonstrated that, at least for these two thresholds, the inclusion of special methods in the cohesion measurement led to different refactoring decisions. This observation does not mean that if other thresholds were selected, then the same results would be obtained. Instead, the observation indicates that it is risky to make refactoring decisions based on obtained cohesion values when the access methods are considered and when the constructor methods are ignored; the refactoring decisions might be misled. In other words, the results indicate that if software engineers use a metric that includes a special method in the cohesion measurement to indicate refactoring opportunities, then they must search for a threshold that is not affected by the inclusion or exclusion of the special methods and that simultaneously provides correct refactoring indications. Such a threshold requires careful investigation, and it might not always exist.

Another threat to the validity of the refactoring empirical study is the fact that we built our recommendations to include or exclude special methods in the cohesion measurement, based on the hypothesis that the constructors contribute to class functionality, whereas the access methods are provided to implement the encapsulation feature. We inspected the considered classes and found that our hypothesis was mostly valid for the constructors and always valid for the access methods. Alternatively, to decide which scenario is better, from a refactoring perspective, the considered classes could be provided to an external software engineer who could intuitively determine which classes require *Extract Class* refactoring; the results could then be compared to the decisions based on the considered metrics, both with and without considering special methods. However, this approach is affected by human intuition, which can differ from one software engineer to another.

In the fault prediction study, the faults were collected from the CVS systems available online for the four selected systems, under the assumption that all of the discovered and fixed faults were reported in the CVS systems.

7.2. External validity

The first external threat is that all of the systems considered were implemented in Java, which prevented us from empirically studying the effect of including destructors. However, constructors and destructors are expected to have similar effects, as discussed in Section 3.

There is a second external threat; our method of automatic detection for the special methods is based on the assumption that the developers of the considered systems have followed the conventional naming guidelines for the attributes and methods. For example, the name of a setter method for a variable *Var* is conventionally named as

setVar. However, deviating from this rule of thumb by using a different name for the setter method, or by naming a method other than the setter as *setVar*, does not cause a syntactic error. To increase the confidence that our developed tool correctly detected the special methods, we manually traced a large number of the selected classes and found that the developers of the selected systems consistently followed this rule of thumb. As assumed, our tool was able to detect these special methods.

Although the selected systems originate from different domains, they are open-source systems that may not be representative of all industrial domains. However, performing empirical analysis on open-source systems is a common practice within the research community. Several studies have investigated the differences in design quality and reliability between open-source systems and industrial systems (e.g., Samoladas et al. 2003, Samoladas et al. 2008, Spinellis et al. 2009).

In the selected systems, as indicated in Section 4, 31% of the methods used in the classes, on average, are special. The average percentage of the special methods in the classes of other systems could be different than the considered percentage, which could lead to different results and conclusions. The selected systems have different average percentages of special methods. We performed the same empirical analysis on the classes of each system separately. The results, which are not provided here because of space limitations, were consistently close to each other, and they led to the same conclusions drawn by the results of the empirical studies reported in this paper. This observation increases the confidence that the general conclusions drawn in this paper are highly expected to be the same for other systems. The classes that were included in the selected systems may not be representative, in terms of their numbers, sizes, and percentages of special methods included. To generalize the results, researchers should conduct similar large-scale evaluations that involve large-scale systems selected from different domains and written in different programming languages.

8. Conclusions and Future Work

This paper empirically addressed whether to include or exclude special methods from cohesion measurements. Two types of special methods were considered, constructors and access methods. The impact of including/excluding each of these special methods on the cohesion values that were obtained using 20 metrics was empirically studied. In the empirical analyses, four scenarios were considered when applying each metric. The empirical study demonstrated the importance of addressing how to deal with the special methods when proposing and applying a cohesion metric. The empirical study demonstrated that the inclusion/exclusion of the special methods caused the cohesion values to significantly change for most of the metrics, and therefore, it is important to decide which scenario to apply for each metric. We considered two applications of interest for software practitioners, including predicting the classes that require refactoring and detecting faulty classes. An empirical study was performed to investigate the effect of including special methods in cohesion measurement when predicting the classes that require *Extract Class* refactoring. The results demonstrated that including special methods in cohesion measurement caused the refactoring prediction results to

significantly change, and therefore, it is recommended that access methods be excluded and constructors be included when using cohesion metrics in the refactoring activity considered. We also empirically studied the change in the fault detection power of each metric when the special methods were included and excluded. The empirical study results showed that the abilities of the metrics considered to detect faulty classes were changed insignificantly when special methods were included and excluded. Therefore, including access methods and excluding constructors in cohesion measurement was empirically found to be harmful when predicting the classes that require refactoring and negligibly effective when predicting the faulty classes.

In the future, we plan to study the impact of including or excluding directly and indirectly inherited attributes and methods on cohesion measurements that apply the same set or larger sets of metrics. In this future study, we plan to consider four different versions of each subclass in the considered systems: (1) the original class, (2) the original class with the inherited attributes, (3) the original class with the inherited methods, and (4) the original class with both the inherited attributes and the inherited methods. The cohesion of each version of each subclass will measured using several metrics. The results will be compared to evaluate the impact of inheritance in cohesion measurement on some applications of interest, such as refactoring. Similarly, we plan to investigate the impact of considering method invocations on cohesion measurement. Method invocations are among the key sources of potential transitive attribute-method relations. That is, a method is not only related to the attributes that it references but also, possibly, transitively related to the attributes referenced by the methods that it invokes. A few of the existing class cohesion metrics capture this potential transitive cohesion aspect. We plan to extend the definition of several existing cohesion metrics to incorporate the cohesion caused by method invocations. The impact of incorporating the transitive relations due to the method invocations on the cohesion values will be studied empirically. Our final goal is to explore the best scenario in which each specific metric can be applied in terms of including and excluding special methods, inherited attributes and methods, and method invocations.

Acknowledgments
The author would like to acknowledge the support of this work by Kuwait University Research Grant WI06/09. In addition, the author would like to thank Anas Abdin and Saqiba Sulman for assisting in collecting the cohesion results.

References
H. Abdi, Bonferroni and Sidak corrections for multiple comparisons, *Neil Salkind (ed.), Encyclopedia of Measurement and Statistics*, Thousand Oaks, CA: Sage, 2007, pp. 1-9.
K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, Investigating effect of design metrics on fault proneness in object-oriented systems, *Journal of Object Technology*, 6(10), 2007, pp. 127-141.
J. Al Dallal, Software similarity-based functional cohesion metric, *IET Software*, 2009, 3(1), pp. 46-57.

J. Al Dallal, Mathematical validation of object-oriented class cohesion metrics, *International Journal of Computers*, 2010, 4(2), pp. 45-52.
J. Al Dallal, Improving the applicability of object-oriented class cohesion metrics, *Information and Software Technology*, 2011a, Vol. 53, No. 9, pp. 914-928.
J. Al Dallal, Measuring the discriminative power of object-oriented class cohesion metrics, *IEEE Transactions on Software Engineering*, 2011b, Vol. 37, No. 7, pp. 788-804.
J. Al Dallal, 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 2011c.
J. Al Dallal, Fault prediction and the discriminative powers of connectivity-based object-oriented class cohesion metrics, acepted for publication in *Information and Software Technology*, 2011d.
J. Al Dallal and L. Briand, An object-oriented high-level design-based class cohesion metric, *Information and Software Technology*, 2010a, Vol. 52, No. 12, pp. 1346-1361.
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, 2010b.
E. Arisholm, L. C. Briand, and E. B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models, accepted for publication on the *Journal of Systems and Software*, 2009.
L. Badri and M. Badri, A Proposal of a new class cohesion criterion: an empirical study, *Journal of Object Technology*, 3(4), 2004, pp. 145-159.
J. Bansiya, L. Etzkorn, C. Davis, and W. Li, A class cohesion metric for object-oriented designs, *Journal of Object-Oriented Program*, 11(8), 1999, pp. 47-52.
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.
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.
L. C. Briand, J. Daly, and J. Wuest, A unified framework for cohesion measurement in object-oriented systems, *Empirical Software Engineering - An International Journal*, 3(1), 1998, pp. 65-117.
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*, 51(3), 2000, pp. 245-273.
L. C. Briand and J. Wust, Empirical studies of quality models in object-oriented systems, *Advances in Computers*, Academic Press, 2002, pp. 97-166.
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.
H. S. Chae, Y. R. Kwon, and D. Bae, A cohesion measure for object-oriented classes, *Software—Practice & Experience*, 30(12), 2000, pp.1405-1431.

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, 26(10), 1991, pp. 197-211.
S.R. Chidamber and C.F. Kemerer, A Metrics suite for object Oriented Design, *IEEE Transactions on Software Engineering*, 20(6), 1994, pp. 476-493.
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)*, 15(2), 2006, pp.123-149.
I. Czibula and G. Serban, Improving systems design using a clustering approach, *IJCSNS International Journal of Computer Science and Network Security*, 6(12), 2006, pp. 40-49.
P. Deitel and H. Deitel, Java How to Program, 6th edition, 2005, Printice Hall.
L. Etzkorn, C. Davis, and W. Li, A practical look at the Lack of Cohesion in Methods metric, *Journal of Object-Oriented Programming*, 11(5), 1998, pp. 27-34.
N. Fenton and S. Pfleeger, Software Metrics: A Rigorous and Practical Approach, Course Technology, 2nd edition, 1998.
L. Fernández, and R. Peña, A sensitive metric of class cohesion, *International Journal of Information Theories and Applications*, 13(1), 2006, pp. 82-91.
FreeMind, http://freemind.sourceforge.net/, accessed April 2011.
M. Fowler, *Refactoring: improving the design of existing code*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1999.
GanttProject, http://sourceforge.net/projects/ganttproject/, accessed April 2011.
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.
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.
B. Henderson-sellers, *Object-Oriented Metrics Measures of Complexity*, Prentice-Hall, 1996.
W. Hines, D. Montgomery, D. Goldsman, and C. Borror, *Probability and statistics in engineering*, 4th edition, John Wiley & Sons Inc., 2003.
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.
D. Hosmer and S. Lemeshow, *Applied Logistic Regression*, Wiley Interscience, 2000, 2nd edition.
Illusion, http://sourceforge.net/projects/aoi/, accessed April 2011.
JabRef, http://sourceforge.net/projects/jabref/, accessed April 2011.
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.
A. De Lucia, R. Oliveto, and L. Vorraro, Using structural and semantic metrics to improve class cohesion, *In Proceedings of IEEE International Conference on Software Maintenance*, 2008, pp. 27-36.

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.

S. Morasca, A probability-based approach for measuring external attributes of software artifacts, *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009, USA, pp. 44-55.

D. Olson and D. Delen, *Advanced Data Mining Techniques*, Springer, 1st edition, 2008.

Openbravo, http://sourceforge.net/projects/openbravopos, accessed April 2011.

I. Samoladas, S. Bibi, I. Stamelos, and G.L. Bleris. Exploring the quality of free/open source software: a case study on an ERP/CRM system, *9th Panhellenic Conference in Informatics*, Thessaloniki, Greece, 2003.

I. Samoladas, G. Gousios, D. Spinellis, and I. Stamelos, The SQO-OSS quality model: measurement based open source software evaluation, *Open Source Development, Communities and Quality*, 275, 2008, pp. 237-248.

D. Spinellis, G. Gousios, V. Karakoidas, P. Louridas, P. J. Adams, I. Samoladas, and I. Stamelos, Evaluating the quality of open source software, *Electronic Notes in Theoretical Computer Science*, 233, 2009, pp. 5-28.

R. Subramanyam and M. Krishnan, Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects, *IEEE Transactions on Software Engineering*, 29(4), 2003, pp. 297-310.

Together software, http://www.borland.com/us/products/together/index.aspx, April 2011.

L. Tokuda and D. Batory, Evolving object-oriented designs with refactorings, *Automated Software Engineering*, 8, 2001, pp. 89-120.

Websphere software, http://www-01.ibm.com/software/websphere/, April 2011.

B. Xu and Y. Zhou, Comments on 'A cohesion measure for object-oriented classes' by H. S. Chae, Y. R. Kwon and D. H. Bae (Softw. Pract. Exper. 2000, 30: 1405-1431), *Software—Practice & Experience*, Vol. 31, No. 14, 2001, pp. 1381-1388.

B. Xu and Y. Zhou, More comments on 'A cohesion measure for object-oriented classes' by H. S. Chae, Y. R. Kwon and D. H. Bae (Softw. Pract. Exper. 2000, 30: 1405-1431), *Software—Practice & Experience*, Vol. 33, No. 6, 2003, pp.583-588.

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

Y. Zhou, J. Lu, H. Lu, and B. Xu, A comparative study of graph theory-based class cohesion measures, ACM SIGSOFT Software Engineering Notes, 29(2), 2004, pp. 13-13.

About the author

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.

Appendix A. Descriptive statistics for the considered metrics

Tables A1, A2, A3, and A4 report the descriptive statistics of the considered metrics applied on the four considered scenarios.

Metric	Min	Max	25%	Med	75%	Mean	Std. Dev.
LCOM1	0	7750	1.00	6.00	25.00	34.20	222.59
LCOM2	0	7750	1.00	3.00	18.50	27.31	215.17
LCOM3	0	20	1.00	1.00	2.00	1.65	1.66
LCOM4	0	20	1.00	1.00	2.00	1.64	1.65
LCOM5	0	2	0.73	0.91	1.00	0.88	0.36
LSCC	0	1	0.00	0.05	1.00	0.32	0.43
CC	0	1	0.00	0.09	0.28	0.20	0.27
SCOM	0	1	0.00	0.10	0.33	0.23	0.31
Coh	0	1	0.13	0.32	0.58	0.39	0.32
CBMC	0	1	0.00	0.00	0.00	0.18	0.36
ICBMC	0	1	0.00	0.00	0.00	0.16	0.35
OL_2	0	1	0.00	0.00	0.00	0.17	0.36
PCCC	0	1	0.00	0.04	1.00	0.39	0.46
TCC	0	1	0.00	0.18	0.67	0.34	0.39
LCC	0	1	0.00	0.21	0.80	0.38	0.41
DC_D	0	1	0.00	0.20	0.67	0.35	0.38
DC_I	0	1	0.00	0.24	0.80	0.38	0.41
CAMC	0.08	1	0.50	0.67	1.00	0.71	0.28
NHD	0	1	0.33	0.57	0.68	0.52	0.26
MMAC	0	1	0.00	0.00	0.03	0.16	0.34

Table A1: Descriptive statistics for the considered metrics applied with excluding all special methods.

Metric	Min	Max	25%	Med	75%	Mean	Avg. absolute mean diff.	Std. Dev.
LCOM1	0	7875	3.00	9.00	28.00	38.61	13.29	227.97
LCOM2	0	7875	1.00	4.00	21.00	29.56	11.65	219.48
LCOM3	0	20	1.00	1.00	1.00	1.55	10.53	1.58
LCOM4	0	20	1.00	1.00	1.00	1.54	10.72	1.58
LCOM5	0	2	0.63	0.81	0.96	0.77	14.73	0.29
LSCC	0	1	0.00	0.10	0.60	0.32	34.45	0.40
CC	0	1	0.04	0.14	0.30	0.22	31.40	0.25
SCOM	0	1	0.03	0.17	0.38	0.26	35.51	0.29
Coh	0	1	0.19	0.38	0.64	0.43	18.99	0.30
CBMC	0	1	0.00	0.00	0.11	0.17	29.77	0.34
ICBMC	0	1	0.00	0.00	0.03	0.14	25.95	0.32
OL_2	0	1	0.00	0.00	0.10	0.17	29.86	0.34
PCCC	0	1	0.00	0.01	0.50	0.28	31.85	0.41
TCC	0	1	0.00	0.29	0.67	0.36	19.79	0.36
LCC	0	1	0.00	0.32	0.78	0.41	18.51	0.40
DC_D	0	1	0.00	0.30	0.67	0.36	19.62	0.36
DC_I	0	1	0.00	0.33	0.78	0.41	18.35	0.40
CAMC	0.08	1	0.40	0.56	0.75	0.59	17.39	0.25
NHD	0	1	0.50	0.60	0.71	0.58	17.82	0.18
MMAC	0	1	0.00	0.00	0.02	0.10	110.47	0.27

Table A2: Descriptive statistics for the considered metrics applied with including constructors and excluding access methods.

Metric	Min	Max	25%	Med	75%	Mean	Avg. absolute mean diff.	Std. Dev.
LCOM1	0	7750	2.00	10.00	35.50	51.79	51.79	256.14
LCOM2	0	7750	1.00	6.00	26.00	40.42	48.92	238.94
LCOM3	0	33	1.00	1.00	2.00	1.79	15.91	1.82
LCOM4	0	33	1.00	1.00	2.00	1.77	15.70	1.80
LCOM5	0	2	0.70	0.88	1.00	0.83	7.79	0.31
LSCC	0	1	0.00	0.06	0.67	0.31	22.50	0.41
CC	0	1	0.02	0.11	0.28	0.20	23.38	0.25
SCOM	0	1	0.01	0.10	0.31	0.23	15.18	0.29
Coh	0	1	0.16	0.32	0.53	0.39	11.38	0.30
CBMC	0	1	0.00	0.00	0.00	0.16	9.88	0.34
ICBMC	0	1	0.00	0.00	0.00	0.14	12.94	0.33
OL_2	0	1	0.00	0.00	0.00	0.16	10.47	0.34
PCCC	0	1	0.00	0.02	1.00	0.34	15.27	0.44
TCC	0	1	0.00	0.21	0.60	0.33	14.40	0.36
LCC	0	1	0.00	0.30	0.79	0.40	11.39	0.40
DC_D	0	1	0.00	0.23	0.60	0.34	14.16	0.36
DC_I	0	1	0.00	0.30	0.79	0.40	11.20	0.40
CAMC	0.08	1	0.43	0.64	1.00	0.67	6.05	0.29
NHD	0	1	0.47	0.62	0.73	0.57	11.31	0.24
MMAC	0	1	0.00	0.00	0.05	0.15	44.41	0.32

Table A3: Descriptive statistics for the considered metrics applied with excluding constructors and including access methods.

Metric	Min	Max	25%	Med	75%	Mean	Avg. absolute mean diff.	Std. Dev.
LCOM1	0	7875	3.00	12.00	42.00	57.27	67.66	264.19
LCOM2	0	7875	1.00	6.00	28.00	42.73	58.56	244.57
LCOM3	0	20	1.00	1.00	2.00	1.62	14.43	1.64
LCOM4	0	20	1.00	1.00	2.00	1.62	14.64	1.64
LCOM5	0	2	0.64	0.80	0.92	0.75	17.77	0.27
LSCC	0	1	0.00	0.10	0.50	0.30	34.61	0.38
CC	0	1	0.05	0.15	0.29	0.21	44.07	0.22
SCOM	0	1	0.05	0.17	0.36	0.26	41.89	0.27
Coh	0	1	0.19	0.36	0.60	0.42	22.68	0.29
CBMC	0	1	0.00	0.00	0.08	0.16	32.03	0.32
ICBMC	0	1	0.00	0.00	0.02	0.13	30.35	0.31
OL_2	0	1	0.00	0.00	0.07	0.15	32.43	0.32
PCCC	0	1	0.00	0.00	0.38	0.25	40.13	0.39
TCC	0	1	0.00	0.30	0.61	0.35	27.42	0.34
LCC	0	1	0.00	0.38	0.82	0.43	25.06	0.39
DC_D	0	1	0.00	0.30	0.61	0.35	27.09	0.34
DC_I	0	1	0.00	0.38	0.82	0.44	24.74	0.39
CAMC	0.08	1	0.36	0.50	0.70	0.55	22.71	0.25
NHD	0	1	0.50	0.63	0.75	0.62	24.28	0.18
MMAC	0	1	0.00	0.00	0.04	0.09	102.12	0.23

Table A4: Descriptive statistics for the considered metrics applied with including all special methods.

Appendix B. Univariate logistic regression results

The detailed univariate logistic regression results are given in Tables B1 and B2. Table B1 shows the results in two cases: with excluding all special methods and with including constructors and excluding access methods. Table B2 shows the results in the other two cases: with excluding constructors and including access methods and with including all special methods.

Metric	1st scenario: Excluding all special methods				2nd scenario: Including constructors			
	Std. Coeff.	p-value	Precision	Recall	Std. Coeff.	p-value	Precision	Recall
LCOM1	2.73	< 0.0001	100.0	60.3	2.44	< 0.0001	100.0	60.3
LCOM2	3.24	< 0.0001	100.0	60.3	3.08	< 0.0001	100.0	60.3
LCOM3	-0.14	0.001	95.0	55.7	-0.16	0.0005	95.4	55.7
LCOM4	-0.15	0.001	94.7	55.9	-0.17	0.0002	95.4	55.7
LCOM5	0.48	< 0.0001	96.7	71.5	0.51	< 0.0001	96.1	71.2
LSCC	-0.44	< 0.0001	80.3	61.5	-0.47	< 0.0001	81.0	61.8
CC	-0.48	< 0.0001	94.3	71.2	-0.45	< 0.0001	95.4	70.5
SCOM	-0.52	< 0.0001	92.8	72.0	-0.48	< 0.0001	93.8	70.7
Coh	-0.57	< 0.0001	88.6	69.4	-0.58	< 0.0001	89.7	69.5
CBMC	-0.42	< 0.0001	89.9	68.7	-0.44	< 0.0001	91.2	68.6
ICBMC	-0.39	< 0.0001	90.1	68.6	-0.39	< 0.0001	92.3	68.3
OL_2	-0.41	< 0.0001	90.1	68.8	-0.43	< 0.0001	91.6	68.4
PCCC	-0.60	< 0.0001	76.9	65.3	-0.48	< 0.0001	81.4	61.2
TCC	0.04	0.381	100.0	60.3	0.10	0.050	100.0	60.3
LCC	0.09	0.085	100.0	60.3	0.12	0.020	100.0	60.3
DC_D	0.05	0.276	100.0	60.3	0.11	0.026	100.0	60.3
DC_I	0.10	0.055	100.0	60.3	0.13	0.010	100.0	60.3
CAMC	-0.52	< 0.0001	64.6	66.7	-0.61	< 0.0001	80.5	63.0
NHD	0.21	0.000	100.0	63.7	0.32	< 0.0001	98.6	64.1
MMAC	-0.11	0.011	88.6	55.3	-0.12	0.004	93.2	55.8

Table B1: Univariate logistic regression results for the first two scenarios.

Metric	3rd scenario: Including access methods				4th scenario: Excluding all special methods			
	Std. Coeff.	p-value	Precision	Recall	Std. Coeff.	p-value	Precision	Recall
LCOM1	1.80	< 0.0001	100.0	60.3	1.69	< 0.0001	100.0	60.3
LCOM2	2.22	< 0.0001	100.0	60.3	2.14	< 0.0001	100.0	60.3
LCOM3	-0.06	0.184	98.3	54.7	-0.11	0.009	95.7	55.1
LCOM4	-0.07	0.119	97.8	54.7	-0.12	0.006	95.5	55.1
LCOM5	0.59	< 0.0001	94.9	71.6	0.58	< 0.0001	93.9	71.3
LSCC	-0.46	< 0.0001	80.9	61.1	-0.49	< 0.0001	83.1	61.4
CC	-0.53	< 0.0001	94.7	71.1	-0.49	< 0.0001	94.7	70.5
SCOM	-0.58	< 0.0001	93.5	72.8	-0.53	< 0.0001	93.5	71.1
Coh	-0.60	< 0.0001	88.8	69.4	-0.59	< 0.0001	89.1	69.3
CBMC	-0.43	< 0.0001	91.5	68.5	-0.44	< 0.0001	92.3	68.4
ICBMC	-0.39	< 0.0001	91.8	68.2	-0.39	< 0.0001	93.2	68.0
OL_2	-0.42	< 0.0001	91.8	68.3	-0.42	< 0.0001	92.7	68.0
PCCC	-0.61	< 0.0001	79.9	64.3	-0.48	< 0.0001	83.7	60.7
TCC	-0.02	0.761	100.0	60.3	0.06	0.261	100.0	60.3
LCC	0.07	0.192	100.0	60.3	0.12	0.017	100.0	60.3
DC_D	0.00	0.922	100.0	60.3	0.07	0.169	100.0	60.3
DC_I	0.08	0.137	100.0	60.3	0.13	0.009	100.0	60.3
CAMC	-0.53	< 0.0001	71.9	64.0	-0.66	< 0.0001	78.8	65.4
NHD	0.20	0.0004	100.0	63.7	0.31	< 0.0001	98.9	64.0
MMAC	-0.12	0.006	89.4	55.6	-0.14	0.001	94.7	55.8

Table B2: Univariate logistic regression results for the last two scenarios.