Fault Prediction and the Discriminative Powers of Connectivity-Based Object-Oriented Class Cohesion Metrics

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

Abstract

Context: Several metrics have been proposed to measure the extent to which class members are related. Connectivity-based class cohesion metrics measure the degree of connectivity among the class members.
Objective: We propose a new class cohesion metric that has higher discriminative power than any of the existing cohesion metrics. In addition, we empirically compare the connectivity and non-connectivity-based cohesion metrics.
Method: The proposed class cohesion metric is based on counting the number of possible paths in a graph that represents the connectivity pattern of the class members. We theoretically and empirically validate this path connectivity class cohesion (PCCC) metric. The empirical validation compares seven connectivity-based metrics, including PCCC, and 11 non-connectivity-based metrics in terms of discriminative and fault detection powers. The discriminative-power study explores the probability that a cohesion metric will incorrectly determine classes to be cohesively equal when they have different connectivity patterns. The fault detection study investigates whether connectivity-based metrics, including PCCC, better explain the presence of faults from a statistical standpoint in comparison to other non-connectivity-based cohesion metrics, considered individually or in combination.
Results: The theoretical validation demonstrates that PCCC satisfies the key cohesion properties. The results of the empirical studies indicate that, in contrast to other connectivity-based cohesion metrics, PCCC is much better than any comparable cohesion metric in terms of its discriminative power. In addition, the results also indicate that PCCC measures cohesion aspects that are not captured by other metrics, wherein it is considerably better than other connectivity-based metrics but slightly worse than some other non-connectivity-based cohesion metrics in terms of its ability to predict faulty classes.
Conclusion: PCCC is more useful in practice for the applications in which practitioners need to distinguish between the quality of different classes or the quality of different implementations of the same class.

Keywords: object-oriented software quality, cohesive interactions, connectivity pattern, discriminative power, object-oriented class cohesion, fault prediction.

1. Introduction

A popular goal of software engineering is to develop techniques and tools that promote quality applications that are stable and easy to maintain. In order to assess and improve the quality of an application during the development process, developers and managers use several metrics. These metrics estimate different software attributes, such as cohesion, coupling, and complexity. The cohesion of a module indicates the extent to which the components of the module are related (Bieman and Ott 1994). A highly cohesive module is likely to be easier to understand, modify, and maintain in comparison to a less cohesive module (Briand et al. 2001a, Chen et al. 2002). The class is the basic unit of object-oriented software, wherein each consists of a set of methods and attributes. Class cohesion metrics measure the relatedness of the methods and attributes within a class.

Several metrics have been proposed in the literature to evaluate class cohesion based on the information that is available during high- or low-level design phases. High-level design (HLD) cohesion metrics (e.g., Briand et al. 1999, Bansiya et al. 1999, Counsell et al. 2006, Al Dallal and Briand 2010) identify potential cohesion issues early in the HLD phase. During the HLD phase, the interactions between the methods and attributes in a class are not precisely known or defined. Low-level design (LLD) cohesion metrics (e.g., Chidamber and Kemerer 1991, Chidamber and Kemerer 1994, Hitz and Montazeri 1995, Bieman and Kang 1995, Chen et al. 2002, Badri 2004, Wang 2005, Bonja and Kidanmariam 2006, Fernandez and Pena 2006, Chae et al. 2000, Chae et al. 2004, Zhou et al. 2002, Al Dallal and Briand 2012) use finer-grained information than that used by the HLD cohesion metrics. In these metrics, the referencing of an attribute by a method is considered to be an interaction. Here the interaction is graphically represented by an undirected edge that links a circular node to a rectangular node, which represent an attribute and a method, respectively. The resulting graph is called a *reference graph*. For example, Figure 1 depicts the reference graph of a hypothetical class that consists of three attributes, three methods, and four interactions. In this graph, the pattern that results from the way in which the nodes that represent attributes and methods are connected by edges is referred to as the connectivity pattern. Intuitively, a class with a more complex connectivity pattern is more cohesive than a class with a less complex connectivity pattern (Xu and Zhou 2001).

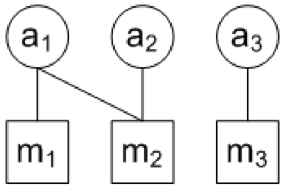

Figure 1. Sample representative graph for a hypothetical class.

LLD cohesion metrics use different measuring approaches. Some of these metrics are based simply on counting the number of interactions (e.g., Henderson-Sellers 1996, Briand et al. 1998), regardless of the relations between these interactions (i.e., whether the interactions are directly connected to the same or different methods or attributes). For example, such metrics similarly account for the interactions that link a_1 to m_1, a_1 to m_2, and a_3 to m_3, although the first two interactions are related to one another and the third is not related to either of the first two. In other words, such metrics narrowly look at the connectivity between each method–attribute pair and, therefore, do not indicate the general complexity of the connectivity pattern of the class members. Similarly, metrics based on counting the number of interactions between each pair of methods (e.g., Chidamber and Kemerer 1991, Li and Henry 1993, Chidamber and Kemerer 1994, Bieman and Kang 1995, Hitz and Montazeri 1995, Badri 2004, Bonja and Kidanmariam 2006, Fernandez and Pena 2006, Al Dallal and Briand 2012) fail to capture the cohesion between different pairs of methods. For example, such metrics individually account for the connectivity between each of the pairs of methods that are represented in Figure 1; however, they do not directly consider the overall cohesion between the three methods m_1, m_2, and m_3. In other words, these metrics narrowly consider the complexity of the connectivity pattern between each pair of methods but not that among all members of a class. Connectivity-based metrics (e.g., Li and Henry 1993, Hitz and Montazeri 1995, Chae et al. 2000, Yang 2002, Xu and Zhou 2003) view cohesion as the degree of connectivity between all of the members of a class, and, therefore, from this perspective, they more comprehensively indicate cohesion than metrics that are based on other measuring approaches. Connectivity-based metrics have several advantages over metrics that use other measuring approaches. First, connectivity metrics directly or indirectly address whether the reference graph of the class is connected or disjointed. A class with a disjoint reference graph is given a low cohesion value because such a class can be easily divided into two or more classes. Second, even when the graph is connected, some connectivity metrics precisely predict the degree of connectivity of the class members, which indicates the complexity of the connectivity pattern of the class reference graph. Therefore, cohesion conclusions that are drawn by applying such metrics are more consistent with intuition than those that are drawn by applying other metrics (Xu and Zhou 2001).

The connectivity-based metrics that have been proposed to date have one or more of the following limitations. The first limitation is that most of these metrics violate one or more key cohesion properties (e.g., Li and Henry 1993, Hitz and Montazeri 1995, Chae et al. 2000, Xu and Zhou 2003), which makes them ill defined, and their usability as cohesion indicators is questionable. It is important to note that in several publications (e.g., Xu and Zhou 2003, Zhou et al. 2004), one of these metrics (Xu and Zhou 2003) has been claimed to satisfy all key cohesion properties, whereas we provide here a counterexample proving that it does not. The second limitation is that without any supporting studies, connectivity-based metrics have been claimed to be more sensitive to changes in the connectivity pattern in comparison to other metrics, and, therefore, they have been claimed to have a higher ability to distinguish between cohesion of classes with different connectivity patterns (e.g., Xu and Zhou 2003, Zhou et al. 2004). Discriminative power is defined as the ability of a class cohesion metric to obtain different cohesion values for classes with the same number of methods and attributes but different connectivity patterns (Al Dallal 2011b); the discriminative powers of connectivity metrics have not been evaluated and compared to those of other metrics. The third limitation is that the ability of connectivity metrics to predict faulty classes has not been empirically studied and compared to that of other metrics. Note that several studies have shown that cohesion metrics can be used as good predictors of faulty classes. Finally, the impact of including or excluding special methods (i.e., constructors, destructors, access, and delegation methods) on cohesion calculation has not been empirically studied.

In this paper, we review and discuss some existing class cohesion metrics and emphasize connectivity metrics. We propose the path connectivity class cohesion (PCCC) metric, which is a connectivity metric that accounts for the degree of connectivity among the class members. This metric is based on counting the number of class reference graph paths that satisfy a certain criterion. The validity of a metric must be both theoretically and empirically studied (Kitchenham et al. 1995). Theoretical validation investigates whether the proposed metric satisfies the key properties of the measured attribute. Empirical validation investigates whether what is being measured demonstrates the expected statistical relationships with other measures, such as measures of external quality attributes. Consistent with this general validation approach, PCCC was then theoretically and empirically validated. In addition, we studied the discriminative power of the metric.

Our theoretical validation involved analyzing whether or not PCCC satisfies the cohesion properties that have been proposed by Briand et al. (1998), and our proofs demonstrate that PCCC does not violate any of these. The discriminative-power study aims to (1) compare the discriminative power of PCCC to those of other connectivity metrics and (2) compare the discriminative powers of connectivity metrics to those of other LLD metrics. This study followed the guidelines that have been proposed by Al Dallal (2011b) and involved the application of 18 metrics, including the most common LLD cohesion metrics in the literature, PCCC, and six other connectivity metrics. These metrics were applied on simulated reference graphs that covered all possible connectivity patterns for several representative models, wherein a model is defined by its number of attributes and methods. Contradictory to the claims, the results demonstrate that the connectivity metrics that have been proposed to date have worse discriminative powers in comparison to those of some other metrics. Interestingly, PCCC was found to be almost twice as good as the next best metric among those studied in terms of discriminative power. This suggests that PCCC is the best metric among those studied in distinguishing between the degrees of cohesion of classes with different connectivity patterns. Therefore, from this perspective, PCCC is the best cohesion indicator. The empirical validation involved the application of the same 18 cohesion metrics to classes that were selected from four open-source Java systems. We investigated the correlations between the 18 considered metrics, thus, determining (1) whether connectivity metrics captured cohesion information that is not covered by nonconnectivity metrics and (2) if PCCC alone captured new cohesion information that is not accounted for by using any other connectivity or nonconnectivity metrics. In addition, in the empirical validation, we investigated the fault-prediction powers of the metrics that were considered individually and in combination. In order to perform this study, we collected fault data for the classes in the considered software systems from publicly available fault repositories and statistically analyzed the relationships between cohesion values and the presence of faults in these classes. The results indicate that most of the nonconnectivity metrics were better than the existing connectivity metrics in terms of their ability to detect faulty classes. In addition, the results demonstrate that PCCC was the best connectivity metric in detecting faulty classes and, from this perspective, it was also better than some of the nonconnectivity metrics. Finally, when the metrics were studied in combination, the results demonstrate that

PCCC was included into the optimal fault-prediction model based on cohesion metrics. This confirms that PCCC is complementary to other metrics in the detection of faulty classes.

The major contributions of this paper are as follows:

1. Introducing a connectivity-based metric that satisfies key mathematical cohesion properties, is much better than other metrics in terms of discriminative power, is complementary to other metrics in terms of indicating cohesion and detecting faulty classes, and is considerably better than other connectivity metrics in predicting faulty classes.
2. Studying the discriminative powers of seven connectivity metrics, including PCCC, and comparing them to those of other metrics.
3. Empirically validating seven connectivity metrics and 11 nonconnectivity metrics by analyzing their correlations with one another and exploring their relationships to the presence of faults in classes of four open-source Java systems.
4. Empirically studying the effects of including or excluding special methods on the computation of connectivity metrics.

This paper is organized as follows. Section 2 reviews related work. Section 3 defines the proposed metric and discusses its theoretical validation. Section 4 explains the discriminative-power study and reports and discusses its results. Section 5 illustrates several empirical case studies and summarizes their results. Finally, Section 6 concludes the paper and discusses future work.

2. Related Work

In this section, we summarize a widely used set of mathematical properties that all class cohesion metrics are expected to satisfy and on which we will rely for our theoretical validation. In addition, we review and discuss several existing connectivity-based class cohesion metrics for object-oriented systems and other related works in the area of software cohesion measurement.

2.1. Necessary properties of class cohesion metrics

Briand et al. (1998) defined four properties that provide a supportive underlying theory for class cohesion metrics. These properties have been widely used to support the theoretical validation of several proposed class cohesion metrics (e.g., Briand et al. 1998, Zhou et al. 2002, Zhou et al. 2004, Al Dallal 2009, Al Dallal 2010, Al Dallal and Briand 2010, Al Dallal and Briand 2012). Metrics that do not satisfy any of these properties are considered to be ill-defined (Briand et al. 1998). The first property, which is called *non-negativity and normalization*, holds that a cohesion measure belongs to a specific interval [0, Max]. Normalization allows for easy comparison between the cohesion of different classes. The second property, which is called *null value and maximum value*, states that the cohesion of a class equals zero if the class has no cohesive interactions, whereas the cohesion of a class is equal to Max if all possible interactions within the class are present. The third property, which is called *monotonicity*, holds that the addition of cohesive interactions to the module cannot decrease its cohesion. That is, the cohesion of the modified class must be greater than or equal to that of the original class. The fourth property, which is called *cohesive modules*, states that the merging of two unrelated modules into one module does not increase the module's cohesion. More specifically, given two classes, *c1* and *c2*, the cohesion of the merged class *c'* must satisfy the following condition: cohesion(*c'*)≤max {cohesion(*c1*), cohesion(*c2*)}.

2.2. Connectivity-based class cohesion metrics

Several metrics have been proposed in the literature to measure class cohesion during the HLD and LLD phases. These metrics use different underlying models and formulas. In this subsection, we start by discussing in detail five connectivity-based cohesion metrics. Other LLD and HLD metrics and less directly relevant work are briefly discussed in the following subsection.

A. Lack-of-Cohesion in Methods

Li and Henry (1993) used an undirected graph that represents each method as a node and the sharing of at least one attribute as an edge. Lack-of-cohesion in methods, LCOM3, is measured in terms of the number of connected components in the graph. This metric was extended by Hitz and Montazeri (1995), who added an edge between two nodes that represent methods if one of the methods invokes the other. When the graph has one component, the following formula is applied:

$$connectivity = 2 * \frac{e-(k-1)}{(k-1)(k-2)}$$, where e is the number of edges and k is the number of nodes.

Al Dallal (2010) proved that neither LCOM3 nor LCOM4 satisfy the *non-negativity and normalization* and *null value and maximum value* cohesion properties but do satisfy the *monotonicity* and *cohesive modules* properties. Both LCOM3 and LCOM4 consider the connectivity of the graph in which the attributes are not represented and, therefore, they cannot capture the detailed connectivity pattern that exists between methods and attributes in their measures. LCOM3 does not differentiate between different connectivity patterns when the graph has one component. In addition, in this case, the connectivity factor that was defined for LCOM4 is based on counting the number of interactions rather than depending on the connectivity pattern. That is, the connectivity factor does not distinguish between two connected graphs that have the same number of nodes and edges but different connectivity patterns. Consequently, both metrics were found to have the weakest discriminative powers among the metrics in a following study (Al Dallal 2011b). In addition, several empirical studies (e.g., Briand et al. 2000, Briand et al. 2001b, Al Dallal and Briand 2010, Al Dallal and Briand 2012) have shown that LCOM3 and LCOM4 are relatively weak in detecting faulty classes.

B. Cohesion Based on Member Connectivity (CBMC)

Chae et al. (Chae et al. 2000) proposed CBMC, a metric that accounts for the connectivity pattern in the reference graph G_c that represents class C. This graph does not represent special methods (i.e., constructors, destructors, delegation, and access methods). The set of glue methods is the minimum set of methods by which their removal causes the graph to become disjointed. This metric defines the connectivity factor F_c of the graph that represents class C to be the ratio of the number of glue methods to the number of normal methods (i.e., nonspecial methods). When the nodes that represent glue methods and their connecting edges are removed from the graph, two or more connected subgraphs are formed. In this case, the original graph is considered to be the parent, whereas the resulting subgraphs are the children. Recursively, in the same manner, each child graph is decomposed into subgraphs until the subgraph has a single node that represents either a method or an attribute, or the subgraph has multiple nodes such that each node that represents a method is connected to each node that represents an attribute. In these cases, the connectivity factor of the subgraph is equal to one. The CBMC of each parent graph is calculated as the product of the connectivity factor of the parent graph and the average connectivity factors of the child graphs. Note that a graph can have several sets of glue methods. In this case, the set of glue methods that results in the highest CBMC is selected to involve in cohesion measurement. The CBMC of the class of interest is the CBMC of the class reference graph.

For example, G_c, which is depicted in Figure 2, is a reference graph for a hypothetical class C that has three attributes, three normal methods, and six interactions. There is only one set of glue methods, and it includes m_1, which means that F_c = one-third. Graphs G_{c1} and G_{c2} result from removing the node that represents the glue method and the two interactions that are connected to it. These graphs are not further decomposed because G_{c1} includes a single node and G_{c2} has two methods, wherein each of them is linked with each attribute. As a result, the average F for the child graphs is one, and, therefore, CBMC of the class is (1/3) × 1 = 1/3.

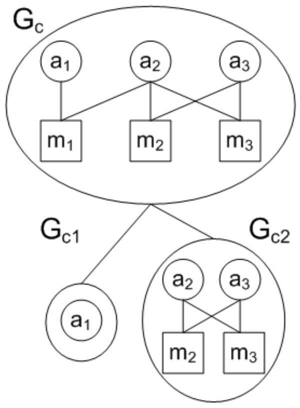

Figure 2. Application of CBMC to a hypothetical class.

CBMC does not distinguish between the cohesion of classes that have disjointed graphs. In this case, the value of CBMC is zero, regardless of the number of disjointed graphs or the connectivity of the individual disjointed graphs. Xu and Zhou (2001) have provided examples that show that in some cases, CBMC does not satisfy the *monotonicity* cohesion property, and that in some other cases, CBMC does not distinguish between graphs that represent classes with the same number of methods and attributes but with different connectivity patterns. However, as far as we know, the discriminative and the fault-detection powers of this metric have not yet been empirically studied.

C. Improved Cohesion Based on Member Connectivity (ICBMC)

In order to overcome the monotonicity problem of CBMC, Xu and Zhou (2001, 2003) improved the metric by proposing the notion of cut sets instead of glue sets. A cut set is the minimum set of edges such that their removal causes the graph to become disjointed. As is the case in CBMC, the reference graph does not represent special methods. This metric defines the connectivity factor as the ratio of the number of edges in a cut set to the product of the number of attributes and the number of nonspecial methods. When removing edges in a cut set, the parent graph is again split into two child graphs. Recursively, in the same manner, each child graph is decomposed into two graphs until the graph has either a single node that represents a method or a single node that represents an attribute; either has one or more edges. Such a graph is called an elementary graph. The connectivity factor of an elementary graph equals one. ICBMC of each parent graph is similarly calculated in the manner in which CBMC is calculated. Note that a graph can have several cut sets. In this case, the cut set that results in the highest ICBMC is selected to involve in cohesion calculation. The ICBMC of the class of interest is the ICBMC of the class reference graph.

For example, G_c, which is depicted in Figure 3, is a reference graph for the same hypothetical class that is represented by G_c in Figure 2. There is only one possible cut set for this graph, and it includes the edge that connects m_1 to a_2, which means that F_c = (*No. of edges in the cut set*)/(*No. of methods* × *No. of attributes*) = 1/(3×3) = 1/9. Graphs G_{c1} and G_{c2} result from removing the edge that is included in the cut set. G_{c1} is an elementary graph, whereas G_{c2} is not. G_{c2} is decomposed into two elementary graphs by removing two edges that are included in a cut set. The connectivity factor of G_{c2} equals 2/(2×2) = 0.5. The connectivity factor for both $G_{c2.1}$ and $G_{c2.2}$ is 1, and, therefore, their average F is one. As a result, the ICBMC for G_{c2} is equal to F_{c2}×1 = 0.5. The average F for G_{c1} and G_{c2} equals (1+0.5)/2 = 0.75. Finally, the ICBMC for G_c equals F_c × 0.75 = (1/9) × 0.75 = 0.083. For G_d, F_d = 1/9, F_{d1} = 1, and F_{d2} = 1; hence, the ICBMC for G_d equals 0.11.

Similarly to CBMC, ICBMC does not distinguish between the cohesion of classes with disjointed graphs. Xu and Zhou (2001, 2003) and Zhou et al. (2004) have claimed that ICBMC satisfies the *monotonicity* cohesion property. Figure 3 depicts a counterexample that demonstrates the opposite. This example demonstrates that the addition of an interaction to G_d so as to form G_c causes the value of the ICBMC to counterintuitively decrease. There are many other similar examples that show that ICBMC violates the *monotonicity* cohesion property in some cases. Without a supporting study, Zhou et al. (2004) have claimed that ICBMC has a high discriminative power. As far as we know, the discriminative and fault-detection powers of ICBMC have not yet been empirically studied.

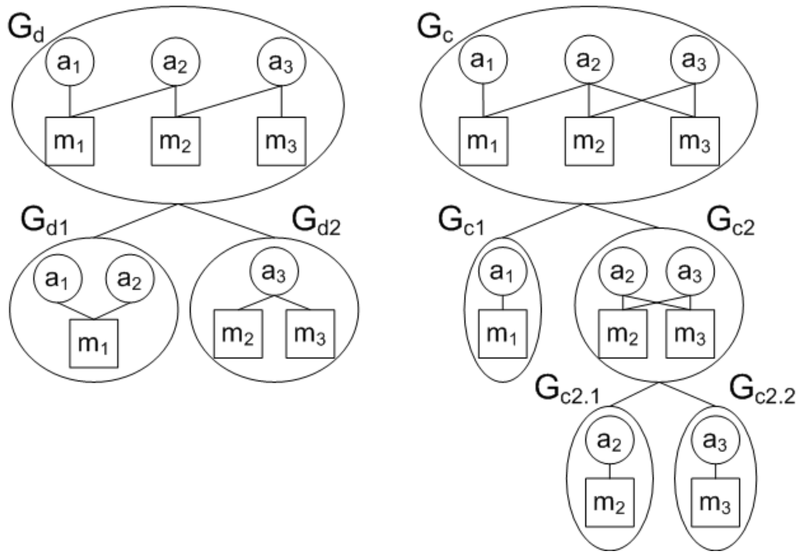

Figure 3. Application of ICBMC to hypothetical classes.

D. OL$_n$

Yang (Yang 2002) has proposed an iteratively calculated class cohesion metric, OL_n, wherein n refers to the number of iterations that are used to calculate OL. In each iteration, the strength S_a of each attribute a and the strength S_m of each method m are calculated. Initially, the strength of each method is set to a value of one. S_a is calculated as the ratio of the summation of the strengths of the methods that are directly connected to attribute a to the total number of normal methods in the class. Returning to the example that is provided in Figure 1, $S_{a1}=(S_{m1}+S_{m2})/3=(1+1)/3=0.67$ and $S_{a2}= S_{a3}=0.33$. Similarly, S_m is calculated as the ratio of the summation of the strengths of the attributes that are directly connected to method m and the total number of attributes in the class; thus, for the same example, $S_{m1}=S_{a1}/3=0.67/3$, $S_{m2}=(0.67+0.33)/3$, and $S_{m3}=0.33/3$. The class cohesion that is calculated in iteration n, OL_n, is defined as the average strengths of the attributes in the class. For the same example, $OL_1=(S_{a1}+S_{a2}+S_{a3})/3=(0.67+0.33+0.33)/3=0.44$. In each iteration, the strengths of the attributes and methods that were obtained in the previous iteration are used to calculate the updated values. This process is applied only for connected reference graphs. If the reference graph is disjointed, the value of OL_n is set to zero. As a result, similarly to CBMC and ICBMC, OL_n does not distinguish between the cohesion of classes with disjointed graphs.

Xu and Zhou (2002) have reported that OL_n satisfies all class cohesion properties; however, they have provided examples that demonstrate that the metric draws conclusions that are inconsistent with intuition in some cases. In addition, they have indicated that the metric is not practical in use because no guidelines have been provided to select a value for n. Finally, Zhou et al. (2004) have assessed the discriminative power of this metric to be moderate. However, as far as we know, the discriminative and fault-detection powers of OL_n have not yet been empirically studied.

2.3. Other related metrics

Connectivity-based metrics are LLD metrics because they rely on information that is provided during the LLD phase. In this study, we compared connectivity-based metrics to 11 other LLD metrics, including LCOM1, LCOM2, LCOM5, Coh, TCC, LCC, DC_D, DC_I, CC, SCOM, and LSCC, as listed in Table 1. The fault-detection powers of these metrics have been thoroughly empirically studied (e.g., Briand et al. 1999, Briand et al. 2000, Briand et al. 2001b, Marcus et al. 2008, Al Dallal and Briand 2010, Al Dallal and Briand 2012, Al Dallal 2011a), and their discriminative powers have been examined (Al Dallal 2011b).

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. $LCOM2 = \begin{cases} P-Q & \text{if } P-Q \geq 0 \\ 0 & \text{otherwise} \end{cases}$
LCOM5 (Henderson-Sellers 1996)	LCOM5= (*a-kl*)/(*l-kl*), 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.
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*.
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*, which 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.
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: $Similarity(i,j) = \frac{\lvert I_i \cap I_j \rvert}{\lvert I_i \cup I_j \rvert}$, where I_i and I_j are the sets of attributes that are referenced by methods *i* and *j*, respectively.
Class Cohesion Metric (SCOM) (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: $Similarity(i,j) = \frac{\lvert I_i \cap I_j \rvert}{\min(\lvert I_i \rvert, \lvert I_j \rvert)} \cdot \frac{\lvert I_i \cup I_j \rvert}{l}$, where *l* is the number of attributes.
Low-level design Similarity-based Class Cohesion (LSCC) (Al Dallal and Briand 2010)	$LSCC(C) = \begin{cases} 0 & \text{if } l = 0 \text{ and } k > 1, \\ 1 & \text{if } (l > 0 \text{ and } k = 0) \text{ or } k = 1, \\ \frac{\sum_{i=1}^{l} x_i (x_i - 1)}{lk(k-1)} & \text{otherwise.} \end{cases}$, where *l* is the number of attributes, *k* is the number of methods, and x_i is the number of methods that reference attribute *i*.

Table 1. The definitions of the considered class cohesion metrics (Al Dallal 2011b).

HLD metrics were not considered in this work because they are not based on the same cohesion information that is used by connectivity metrics. Cohesion among methods in a class (CAMC), normalized hamming distance (NHD), scaled NHD (SNHD), distance design-based direct class cohesion (D_3C_2), and similarity-based class cohesion (SCC) are examples of HLD metrics. CAMC (Bansiya et al. 1999), NHD, and SNHD (Counsell et al. 2006) use method parameter types to predict the interactions between methods and attributes. D_3C_2 (Al Dallal 2007) uses the relationships between parameter types and attribute types to predict the interactions between the methods and attributes. SCC (Al Dallal and Briand 2010) extends D_3C_2 by considering more types of interactions, including interactions that are caused by method invocations that are modeled in UML diagrams.

3. Path-Connectivity Class cohesion (PCCC) Metric

The path connectivity class cohesion metric (PCCC) proposed herein accounts for the connectivity pattern of the class reference graph. The metric was motivated, proposed, and theoretically validated as follows.

3.1. Motivation

As previously indicated, class cohesion refers to the extent to which methods and attributes are related. By definition, there are three types of possible relations, including method-method, method-attribute, and attribute-attribute. In this paper, we rely on the same reference graph that was proposed earlier to represent the interactions among the class members. A reference graph for another hypothetical class is given in Figure 4 and used here so as to clarify the proposed concepts. As before, in this graph, attributes and methods are represented by nodes, and the references of attributes by methods are represented by undirected edges. Between any two nodes, wherein one represents a method and the other an attribute, there exists at most a single edge, which indicates that the method references the attribute at least once. A path between two nodes is represented by a sequence of alternating nodes and edges. The number of edges in the path defines its length. For simplicity, we identify the path by its ordered nodes. For example, in Figure 4, (m_1, a_1, m_2, a_2) is a path of length three and (a_2, m_1, a_1) is a path of length two. Note that the path can start or end at a node that represents either a method or an attribute. A simple path is a path in which all of the nodes are distinct. For example, the two previously indicated paths are simple, whereas path (m_1, a_1, m_2, a_2, m_1) is not.

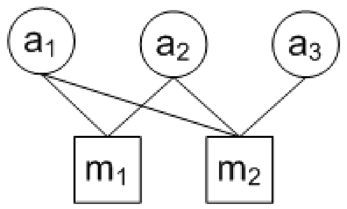

Figure 4. Sample reference graph for a hypothetical class.

The reference graph directly or indirectly represents the three possible types of relations. An edge between two nodes represents a direct method-attribute relation, where such a relation indicates that the method references the attribute. For example, the graph that is depicted in Figure 4 represents five direct method-attribute relations. A transitive method-attribute potential relation is represented in the graph by a simple path that (1) starts and ends at the nodes that represent the method and the attribute, respectively, and (2) has a length that is greater than one. For example, path (m_1, a_1, m_2, a_2) represents a potential transitive relation between method m_1 and attribute a_2. Similarly, paths (m_1, a_1, m_2) and (a_1, m_2, a_2) represent potential transitive method-method and attribute-attribute relations, respectively. Transitive relations are considered potential relations because they do not guarantee the existence of the relation. For example, assume the existence of a reference graph similar to the one given in Figure 4, but without the direct edge between a_1 and m_2. In this case, the path (a_1, m_1, a_2, m_2, a_3) indicates a possible relation between attributes a_1 and a_3. However, attributes a_1 and a_2 are not accessed by a common method, and their values might not be affected by the value of a_2; therefore, attributes a_1 and a_3 might be unrelated. It is important to note that the reference graph does not include any direct edge between a pair of nodes that represent methods or a pair of nodes that represent attributes; hence, the graph does not represent direct method-method and attribute-attribute relations.

Clearly, as a reference graph includes more simple paths, the more complex that graph becomes. The presence of more simple paths in the reference graph implies the existence of more potential relations among the class members. From a node to another node, there may exist several simple paths that hop over different nodes. For example, there are two simple paths between the two attributes a_1 and a_3 in Figure 4: (a_1, m_1, a_2, m_2, a_3) and (a_1, m_2, a_3). The existence of nodes in a simple path indicates the existence of a potential relation between the attributes and methods that are represented by these nodes. In addition, the existence of more simple paths from a node to another node indicates the existence of a potentially stronger relation between the attributes or methods that are represented by the two nodes. This is because the existence of more simple paths between two nodes implies the existence of more relations between the attributes or methods that are represented by the two nodes, wherein each relation includes a different set of attributes and methods. As a result, the number of simple paths between each pair of nodes in the reference graph must be considered in a cohesion measurement. This number of simple paths not only accounts for the strengths of the relations between each pair of nodes but also indirectly accounts for the strengths of relations between the two nodes and the other nodes in the reference graph. This is because each simple path includes a different set of nodes, and, therefore, the existence of more simple paths between two nodes implies that the two nodes are related to other nodes. In some cases, two or more simple paths can have the same nodes but in different orders. In these cases, the paths represent different relations between the nodes. It is important to note that changing an endpoint of an edge in the reference graph changes some of the simple paths that are included in the reference graph, which is highly expected to change the total number of simple paths, as will be shown in the discriminative-power study in Section 4. This means that it is almost certain that the number of simple paths changes as the connectivity pattern of the reference graph changes.

3.2. The metric

In order to count the number of simple paths between a pair of nodes, we propose an *all-simple-paths* tree. This tree represents all of the simple paths that originate at node n to every other node to which node *n* is directly or transitively connected. The procedure given in Figure 5 formally describes how this tree is constructed. The procedure starts by adding to the tree a node *n* for which the simple paths are to be determined. In the process, whenever a node *m* is added to the tree, the procedure traverses all of its connected edges in the reference graph. The traversed edge and another node *k*, to which the edge is connected, is added to the tree only when node *k* is not already encountered on the path from node *n* to node *m*.

Figure 6 depicts the five all-simple-paths trees that correspond to the five nodes that are included in the reference graph depicted in Figure 4. For example, in the construction process of an all-simple-paths tree for node m_2 [Figure 4(e)], a corresponding node is added as a root and is marked *not final*. The three edges that are connected to node m_2 in the graph are traversed. They all lead to nodes that are not yet represented in the tree and, therefore, corresponding nodes are added as children to the tree root node; the added nodes are marked *not final,* and the root node is marked *final*. After that, each of the graph nodes that correspond to the added nodes is examined. First, graph node a_1 is examined, wherein it is found that it is connected to two nodes, m_1 and m_2. However, a node that corresponds to m_2 is already encountered in the tree path from m_2 to a_1; therefore, the node is ignored. Conversely, a node that corresponds to m_1 is not encountered in the path from m_2 to a_1; therefore, a corresponding node is added as a child for the tree node that corresponds to a_1, and the added node is marked *not final*. Now, all graph edges that are connected to node a_1 are traversed, and, therefore, the corresponding tree node is marked *final*. The process similarly continues to construct the tree.

```
Input: A reference graph and a node n for which the simple paths are to be determined.
Output: The all-simple-paths tree for node n.
Procedure:
1.  Add a corresponding node n to the tree as a root node and mark the node as not final.
2.  for each tree leaf node m marked not final
        a.  for each graph node k connected to the node corresponding to node m in the reference
            graph
                i.  if node k is not yet represented in the tree path from the tree root node to tree
                    node m
                        i.1. Add a node corresponding to node k as a child to tree node m.
                        i.2. Mark the added node as not final.
        b.  Mark tree node m as final.
3.  Repeat Step 2 until all tree leaf nodes are marked final.
```

Figure 5. The production of an all-simple-paths tree from a reference graph.

Note that a graph node *m* can be represented more than once in the all-simple-paths tree of a node *n*. This means that there is more than one simple path in the graph between nodes *n* and *m*. The number of occurrences of a node *m* in the all-simple-paths tree of a node *n* represents the number of simple paths from node *n* to node *m*. As a result, the total number of simple paths from node *n* to all other nodes in the graph is equal to the number of nodes, excluding the root, in the corresponding tree. For example, node m_1 is represented twice in the all-simple-paths tree of node a_1 that is depicted in Figure 6(a), which indicates that there are two simple paths from node a_1 to node m_1 in the reference graph. There are eight nodes in this tree, excluding the root, which means that there are exactly eight possible simple paths that start at node a_1. Similarly, there are eight, seven, eight, and seven simple paths that start from nodes *a2*, *a3*, *m1*, and *m2*, respectively. As a result, the total number of simple paths in the reference graph depicted in Figure 4 is 38. In order to facilitate the counting of the number of simple paths in the graph, we propose the algorithm depicted in Figure 7. This algorithm recursively counts the number of simple paths without constructing the required trees. Instead, it uses a *visited* array that simulates the *final* marking schema that is used in the construction process of the all-simple-paths tree.

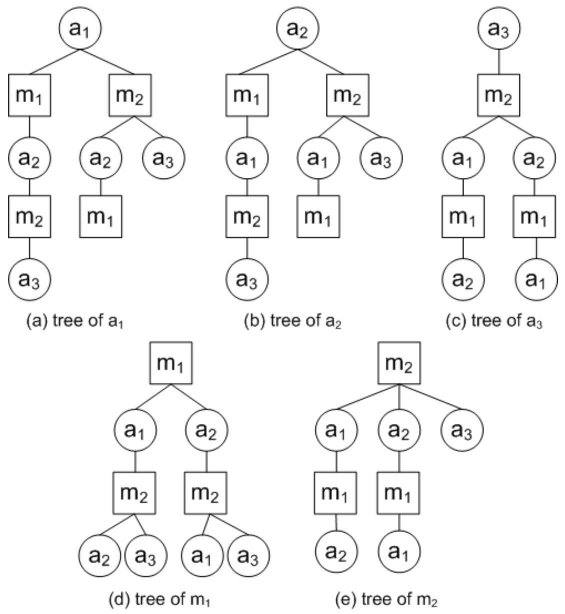

Figure 6. All-simple-paths trees that correspond to the nodes included in Figure 4.

```
Input: A reference graph G
Output: Number of simple paths counter in graph G
Algorithm:
    1. counter = 0
    2. for each node n in G
        a. for each node m in G
            i. create an array visited such that each cell in the array corresponds to a
               node in G
            ii. initialize all cells of visited to false
        b. counter = counter + traverse(G,n,visited)

function traverse(reference graph G, node n, array visited)
    counter=0
    create an array cvisited such that each cell in the array corresponds to a node in G
    copy array visited to cvisited
    set the value of cvisited cell corresponding to node n to value true
    for each node k connected to n in G
        if the cvisited cell corresponding to node k has value false
            counter = counter + 1
            counter = counter + traverse(G,k,cvisited)
    return counter
```

Figure 7. Algorithm for counting the number of simple paths in a reference graph.

The maximum possible number of simple paths in a reference graph, including the same number of nodes as in *G*, is encountered when there is an edge between each node that corresponds to a method and each node that corresponds to an attribute. We call such a reference graph *fully connected* and refer to it as an *FG*. As a result, given a set of attributes *a* and a set of methods *m* of a class *c*, the corresponding fully connected reference graph denoted as $FG_{c,a,m}$ is formally defined as a reference graph in which there is an edge between each node that represents an attribute in *a* and each node that represents a method in *m*. In an *FG*, the number of simple paths starting from any node that represents any attribute is the same for each attribute. This is because, in an *FG*, each node that represents an attribute is connected to the same set of nodes that represents methods; the same applies for each node that represents a method. Therefore, to count the number of simple paths in an *FG*, only two all-simple-paths trees need to be constructed: one starting from a node that represents an attribute and the other starting from a node that represents a method. Given that the number of nodes, excluding the root, in the former and latter trees are S_a and S_m, respectively, and that *a* and *m*, respectively, are the sets of considered attributes and methods in a class *c*, $NSP(FG_{c,a,m})$ is defined as the number of simple paths in $FG_{c,a,m}$ and is computed using the formula: $NSP(FG_{c,a,m})=|a|\times S_a+|m|\times S_m$. For example, S_a and S_m for an *FG* that corresponds to the graph *G* depicted in Figure 4 are 14 and 12, respectively. In this case, $NSP(FG_{c,\{a1,a2,a3\},\{m1,m2\}})=3\times14+2\times12=66$.

In order to satisfy *non-negativity and normalization* cohesion property, we define PCCC as the ratio of the number of simple paths in reference graph *G* to the number of simple paths in the corresponding graph *FG*. The PCCC of a class *C* with a set of attributes *a* and a set of considered methods *m*, which is represented by reference graph *G* with $NSP(G_{c,a,m})$ simple paths, is, thus, formally defined as follows:

$$PCCC(C,a,m)=\begin{cases} 0 & \text{if } |a|=0 \text{ and } |m|>1, \\ 1 & \text{if } |a|>0 \text{ and } |m|=0, \\ \dfrac{NSP(G_{c,a,m})}{NSP(FG_{c,a,m})} & \text{otherwise.} \end{cases}$$

For example, the PCCC for the class that is represented by the reference graph depicted in Figure 4 equals 38/66=0.576. Note that the reference graph is undirected, and, therefore, the simple path between two nodes *n* and *m* is counted twice, one starting from node *n* and ending at node *m* and vice versa. The impact of such duplication is eliminated in the normalized metric because the simple paths are also counted twice when calculating *NSP*(*FG*).

In order to define the PCCC value for special cases, we followed the metric value-assignment criteria that has been proposed by Al Dallal (2011a) as follows. PCCC is undefined for the meaningless case of a class with no attributes and methods. If a class with several methods does not have any attributes, the methods will be considered unrelated, and the cohesion will be the minimum value, which is zero. If a class with attributes does not have methods, the attributes declare a class structure that does not have behavior. In this case, the attributes describe the features of the same object, and the class is expected to be fully cohesive.

Typically, object-oriented classes include constructors, delegation methods, and access methods (i.e., setter and getter methods). In some object-oriented programming languages, such as C++, classes can also have destructors. Usually, the constructor method provides initial values for most or all of the class attributes. Therefore, typically, the number of simple paths starting from a node that represents a constructor method is equal to or higher than that of any other method. As a result, including constructors can cause the PCCC value to artificially increase. The same argument applies to destructors, although destructors are less problematic because they typically do not reference most if not all attributes. An access or delegation method references, by definition, one attribute; therefore, the number of simple paths that start from the node that represents an access or a delegation method is relatively low. Therefore, the presence of access and delegation methods causes the PCCC value to artificially decrease. As a result, when PCCC is applied, constructors, destructors, delegation methods, and access methods should be excluded based on theoretical grounds.

In order to account for class inheritance, all directly and transitively accessible inherited methods and attributes must be represented in the reference graph. Inherited methods can be extracted with source-code analysis, although dynamic binding introduces complications because source-code analysis is statically performed. The inclusion of inherited attributes and methods permits the measurement of the cohesion of the class as a whole, whereas excluding them results in a local measure of the class cohesion. This means that including or excluding inherited attributes and methods depends on the measurement purpose. Inherited methods do not reference the attributes of the inheriting class. Therefore, including only the inherited methods decreases the PCCC value. Conversely, inherited attributes are meant to be referenced by some methods in the inheriting class. Therefore, the change in the PCCC value upon including only the inherited attributes depends on the relative increase in the number of simple paths. Finally, changes in the PCCC value when including both inherited attributes and methods depend on (1) the number of inherited methods and attributes, (2) the number of simple paths in the original reference graph starting from the nodes to which the added nodes are connected, and (3) the connectivity pattern of the added nodes.

3.3. Theoretical validation

We validated PCCC using properties of a class cohesion metric that was proposed by Briand et al. (1998) and discussed in Section 2.1. For simplicity, the reference graph that corresponds to a class *c* with a set of attributes *a* and a set of methods *m* is denoted here as G_c, and the corresponding PCCC is denoted as PCCC(c).

Property PCCC.1: The PCCC metric satisfies the non-negativity and normalization property.

Proof: The minimum value for the PCCC of a class is zero when the reference graph of the class has no edges. In this case, none of the methods reference an attribute. The maximum value for the PCCC of a class is one when there is an edge between each node that represents an attribute and each node that represents a method in the reference graph of the class. In this case, each method references each attribute in the class. As a result, the PCCC metric ranges over the interval [0, 1] and therefore satisfies the non-negativity and normalization property.■

Property PCCC.2: The PCCC metric satisfies the null-and-maximum-values property.
Proof: Given a class with a set of methods and attributes, if none of the methods reference an attribute (that is, the class has no interactions), the PCCC value will be zero. Alternatively, if each attribute is referenced by each method (that is, the class features all possible interactions), the PCCC value will be one (that is, the maximum possible value). Hence, the PCCC metric satisfies the null-and-maximum-values property.■

Property PCCC.3: The PCCC metric satisfies the monotonicity property.
Proof: The addition of an edge to the reference graph does not break any of the simple paths, and it adds at least one more simple path of length one, which is the direct path between the two endpoint nodes of the added edge. As a result, the addition of a cohesive interaction to the class always increases the PCCC value, which means that the PCCC metric satisfies the monotonicity property.

Property PCCC.4: The PCCC metric satisfies the cohesive-module property.
Proof: The merging of two unrelated classes *c1* and *c2* into class *c3* implies that none of the nodes in the reference graph of *c1* is connected by an edge to any node in the reference graph of *c2*. This means that $NSP(G_{c3})=NSP(G_{c1})+NSP(G_{c2})$. However, the fully connected graph FG_{c3} has the same set of edges in FG_{c1} and FG_{c2} in addition to the set of edges between each node in FG_{c1} and each node in FG_{c2}. This implies that $NSP(FG_{c3})=NSP(FG_{c1})+NSP(FG_{c2})+f$, where f is the number of simple paths that are created by the additional edges. The value of f is zero when both classes only have attributes or methods; otherwise, the value of f is greater than zero.
Suppose that PCCC(*c1*)≥PCCC(*c2*); then:

$$\frac{NSP(G_{c1})}{NSP(FG_{c1})} \geq \frac{NSP(G_{c2})}{NSP(FG_{c2})} \Rightarrow NSP(G_{c1}) \times NSP(FG_{c2}) \geq NSP(G_{c2}) \times NSP(FG_{c1})$$

$$\Rightarrow NSP(G_{c1}) \times [NSP(FG_{c1}) + NSP(FG_{c2})] \geq [NSP(G_{c1}) + NSP(G_{c2})] \times NSP(FG_{c1})$$

$$\Rightarrow \frac{NSP(G_{c1})}{NSP(FG_{c1})} \geq \frac{NSP(G_{c1}) + NSP(G_{c2})}{NSP(FG_{c1}) + NSP(FG_{c2})} \geq \frac{NSP(G_{c1}) + NSP(G_{c2})}{NSP(FG_{c1}) + NSP(FG_{c2}) + f}$$

$$\Rightarrow PCCC(c1) > PCCC(c3)$$

Therefore, *Max*{PCCC(*c1*),PCCC(*c2*)}>PCCC(*c3*).
This means that the PCCC metric satisfies the cohesive-module property. ■

Proving that a cohesion metric satisfies expected mathematical properties increases the chances that the metric is a meaningful class cohesion indicator; however, this is not necessarily guaranteed, and it must be empirically explored. In order to show that PCCC is a better-defined metric, the following two sections report the results of empirical studies that have explored whether PCCC can differentiate between two classes of different connectivity patterns and whether it better explains the observed statistical variation in fault occurrences in comparison to other connectivity-based cohesion metrics.

4. Discriminative-Power Simulation Study

Al Dallal (2011b) has defined the discriminative power of a class cohesion metric as "*the ability of a class cohesion metric to obtain different cohesion values for classes of the same number of methods and attributes but different connectivity patterns of cohesive interactions (CPCIs).*" Obtaining the same cohesion values for classes of different connectivity patterns misleads developers by incorrectly considering the classes to be the same in terms of cohesion, even though their connectivity patterns clearly indicate that the degrees of cohesion are different. Given a reference graph with nodes that represent a certain number of attributes and methods, the discriminative power of a metric (DPC) is measured as the ratio of the number of distinct cohesion values that are obtained by applying the metric to the number of possible distinct connectivity patterns that can be formed using the model, wherein the model is defined by its number of attributes and methods. Therefore, to compute DPC, we (1) form all possible distinct CPCIs for the model; (2) apply cohesion metric *m*, for which we want to compute DPC, to each connectivity pattern that is formed in Step 1; (3) count the number of distinct cohesion values that are obtained in Step 2; and (4) divide this number by the number of distinct CPCIs that are formed in Step 1. This process is applied to each considered model using each metric. A supporting tool was used to automate the formation of the distinct CPCIs and the computation of DPC for several metrics, including all of the metrics that are considered in this paper except for PCCC, CBMC, ICBMC, and OL_n. We extended this tool so as to obtain the required results for these metrics. For OL_n, we considered OL_2 and OL_3.

In order to compare the results that were obtained for connectivity-based metrics to those of other metrics, we applied the analysis to the same models that were considered by Al Dallal (2011b), and the results are reported in Table 2. The first three columns of Table 2 depict the number of methods, attributes, and distinct CPCIs. The other columns of Table 2 list the DPC values for the 18 considered metrics, and the best result for each model is highlighted in boldface. The results for the non-connectivity-based metrics are exactly the same as those that are included in (Al Dallal 2011b); they are repeated here to facilitate comparison to the new results. The DPC value of 0.00 reported in Table 2 for some metrics indicates that the resulting values were very small (i.e., values are greater than zero but less than 0.005). For brevity's sake, we show only two graphs here, specifically in Figures 8 and 9, wherein they depict the changes in DPC values for the considered connectivity metrics as the number of attributes was changed across three methods and as the number of methods was changed across three attributes, respectively.

No. of methods	No. of attributes	No. of distinct CPCIs	DPC values												
			PCCC	CBMC	ICBMC	OL_2	OL_3	LCOM3, LCOM4	LCOM1, TCC, DC_D	LCOM2	LCC, DC_I	Coh, LCOM5	LSCC	CC	SCOM
2	1	3	**1.00**	0.67	0.67	0.67	0.67	0.33	0.67	0.67	0.67	**1.00**	0.67	0.67	0.67
2	2	7	**0.86**	0.43	0.43	0.43	0.43	0.29	0.29	0.29	0.29	0.71	0.43	0.43	0.43
2	3	13	**0.77**	0.23	0.31	0.38	0.38	0.15	0.15	0.15	0.15	0.54	0.31	0.38	0.38
2	4	22	**0.73**	0.14	0.23	0.32	0.32	0.09	0.09	0.09	0.09	0.41	0.23	0.32	0.32
2	5	34	**0.74**	0.09	0.18	0.29	0.29	0.06	0.06	0.06	0.06	0.32	0.18	0.32	0.35
2	6	50	**0.70**	0.06	0.14	0.26	0.26	0.04	0.04	0.04	0.04	0.26	0.14	0.26	0.30
2	7	70	**0.70**	0.04	0.11	0.24	0.24	0.03	0.03	0.03	0.03	0.21	0.11	0.27	0.34
2	8	95	**0.73**	0.03	0.09	0.22	0.22	0.02	0.02	0.02	0.02	0.18	0.09	0.24	0.32
2	9	125	**0.74**	0.02	0.08	0.21	0.21	0.02	0.02	0.02	0.02	0.15	0.08	0.23	0.33
2	10	161	**0.74**	0.02	0.07	0.18	0.19	0.01	0.01	0.01	0.01	0.13	0.07	0.20	0.30
3	1	4	**1.00**	0.50	0.50	0.50	0.50	0.25	0.75	0.75	0.75	**1.00**	0.75	0.75	0.75
3	2	13	**0.77**	0.31	0.31	0.38	0.38	0.15	0.31	0.23	0.23	0.54	0.46	0.38	0.54
3	3	36	**0.56**	0.14	0.17	0.31	0.31	0.08	0.11	0.08	0.08	0.28	0.25	0.33	0.42
3	4	87	**0.53**	0.07	0.13	0.37	0.40	0.03	0.05	0.03	0.03	0.15	0.14	0.26	0.31
3	5	190	**0.52**	0.04	0.08	0.33	0.40	0.02	0.02	0.02	0.02	0.08	0.08	0.30	0.29
3	6	386	**0.49**	0.02	0.08	0.28	0.39	0.01	0.01	0.01	0.01	0.05	0.05	0.21	0.23
3	7	734	**0.49**	0.01	0.05	0.23	0.37	0.00	0.01	0.00	0.00	0.03	0.03	0.27	0.23
3	8	1324	**0.47**	0.01	0.05	0.18	0.35	0.00	0.00	0.00	0.00	0.02	0.02	0.25	0.21
3	9	2284	**0.45**	0.00	0.04	0.14	0.34	0.00	0.00	0.00	0.00	0.01	0.01	0.25	0.21
4	1	5	**1.00**	0.40	0.40	0.40	0.40	0.20	0.80	0.60	0.80	**1.00**	0.80	0.80	0.80
4	2	22	**0.73**	0.23	0.23	0.32	0.32	0.09	0.32	0.18	0.23	0.41	0.41	0.41	0.55
4	3	87	**0.53**	0.10	0.13	0.37	0.40	0.03	0.08	0.05	0.06	0.15	0.17	0.29	0.37
4	4	317	**0.38**	0.04	0.06	0.22	0.28	0.01	0.02	0.01	0.02	0.05	0.07	0.17	0.23
4	5	1053	0.38	0.02	0.04	0.17	**0.46**	0.00	0.01	0.00	0.00	0.02	0.03	0.18	0.17
4	6	3250	0.35	0.01	0.03	0.09	**0.40**	0.00	0.00	0.00	0.00	0.01	0.01	0.07	0.11
4	7	9343	0.30	0.00	0.02	0.05	**0.32**	0.00	0.00	0.00	0.00	0.00	0.00	0.11	0.09
5	1	6	**1.00**	0.33	0.33	0.33	0.33	0.17	0.83	0.67	0.83	**1.00**	0.83	0.83	0.83
5	2	34	**0.74**	0.18	0.18	0.29	0.29	0.06	0.32	0.18	0.21	0.32	0.41	0.38	0.53
5	3	190	**0.52**	0.08	0.08	0.33	0.40	0.02	0.06	0.03	0.04	0.08	0.14	0.23	0.29
5	4	1053	0.38	0.03	0.04	0.17	**0.46**	0.00	0.01	0.01	0.01	0.02	0.03	0.09	0.14
5	5	5624	0.22	0.01	0.02	0.05	**0.25**	0.00	0.00	0.00	0.00	0.00	0.01	0.07	0.07
5	6	28576	**0.22**	0.00	0.01	0.02	0.19	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.03
6	1	7	**1.00**	0.29	0.29	0.29	0.29	0.14	0.86	0.71	0.86	**1.00**	0.86	0.86	0.86
6	2	50	**0.70**	0.14	0.14	0.26	0.26	0.04	0.32	0.18	0.18	0.26	0.38	0.38	0.52
6	3	386	**0.49**	0.06	0.08	0.28	0.39	0.01	0.04	0.02	0.02	0.05	0.09	0.18	0.22
6	4	3250	0.35	0.02	0.03	0.09	**0.40**	0.00	0.00	0.00	0.00	0.01	0.02	0.04	0.08
6	5	28576	**0.22**	0.00	0.01	0.02	0.19	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.02
7	1	8	**1.00**	0.25	0.25	0.25	0.25	0.13	0.88	0.75	0.88	**1.00**	0.88	0.88	0.88
7	2	70	**0.70**	0.11	0.11	0.24	0.24	0.03	0.31	0.17	0.17	0.21	0.36	0.36	0.51
7	3	734	**0.49**	0.05	0.05	0.23	0.37	0.00	0.03	0.02	0.02	0.03	0.07	0.13	0.16
7	4	9343	0.30	0.01	0.02	0.05	**0.32**	0.00	0.00	0.00	0.00	0.00	0.01	0.02	0.04
8	1	9	**1.00**	0.22	0.22	0.22	0.22	0.11	0.89	0.67	0.89	**1.00**	0.89	0.89	0.89
8	2	95	**0.73**	0.09	0.09	0.22	0.22	0.02	0.31	0.16	0.17	0.18	0.34	0.37	0.49
8	3	1324	**0.47**	0.04	0.05	0.18	0.35	0.00	0.02	0.01	0.01	0.02	0.05	0.10	0.12
9	1	10	**1.00**	0.20	0.20	0.20	0.20	0.10	0.90	0.70	0.90	**1.00**	0.90	0.90	0.90
9	2	125	**0.74**	0.08	0.08	0.21	0.21	0.02	0.30	0.15	0.15	0.15	0.31	0.34	0.50
9	3	2284	**0.45**	0.03	0.04	0.14	0.34	0.00	0.02	0.01	0.01	0.01	0.04	0.08	0.09
10	1	11	**1.00**	0.18	0.18	0.18	0.18	0.09	0.91	0.73	0.91	**1.00**	0.91	0.91	0.91
10	2	161	**0.74**	0.07	0.07	0.18	0.19	0.01	0.29	0.15	0.14	0.13	0.29	0.34	0.47
Average			**0.63**	0.12	0.15	0.24	0.32	0.06	0.23	0.18	0.20	0.31	0.27	0.34	0.38

Table 2. DPC values for the considered metrics.

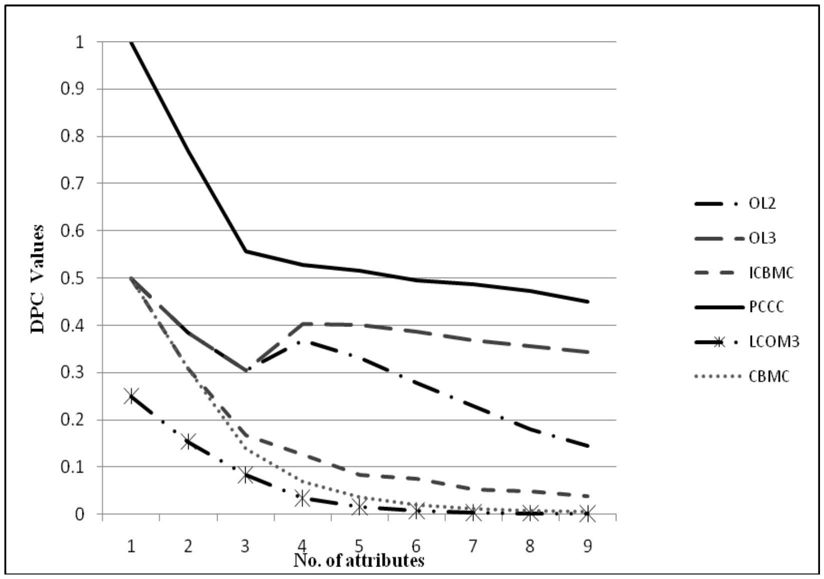

Figure 8. DPC values versus the number of attributes (number of methods = 3).

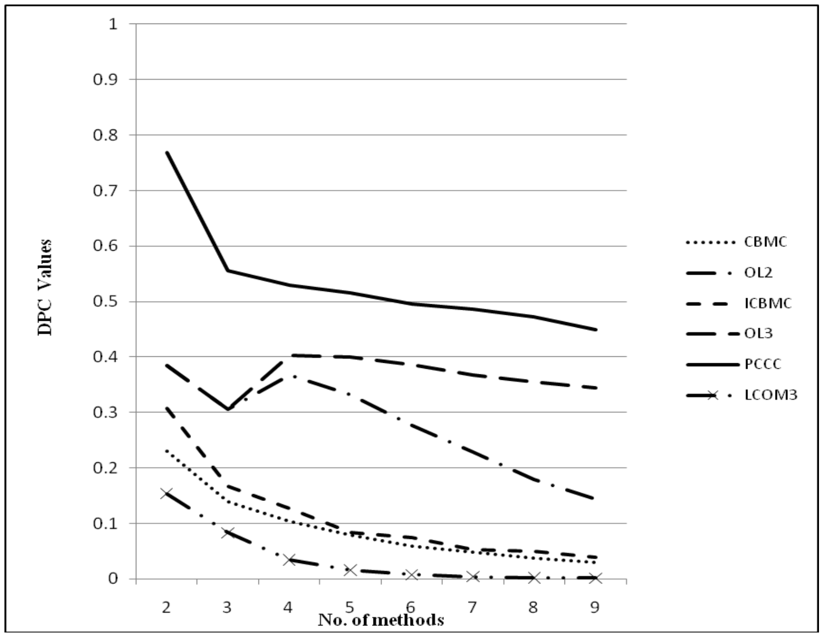

Figure 9. DPC values versus the number of methods (number of attributes = 3).

The results that are listed in Table 2 lead to the following conclusions:

1. The PCCC metric yielded the largest DPC value for most of the considered models, which indicates that PCCC was usually the most discriminative cohesion metric among those considered. In addition, PCCC was almost twice as good as the second-best metric in terms of the average DPC.
2. Among all of the considered metrics, LCOM3 and LCOM4 exhibited the smallest DPC values for all of the models, suggesting that LCOM3 and LCOM4 were the

least discriminative cohesion metrics among those considered. This result is consistent with the empirical results, which will be shown in the next section, and demonstrates that LCOM3 and LCOM4 are not reliable cohesion indicators.

3. Although CBMC and ICBMC have been proposed to distinguish between reference graphs of different connectivity patterns, contradictorily, the results show that they were worse than any of the other considered nonconnectivity metrics. The results indicate that these two metrics did not achieve the purpose for which they were proposed.
4. The results for OL_n indicate that its discriminative power increased as the value of *n* was increased, although other values of *n* must be considered to prove or disapprove this observation. In addition, the results for OL_3 were slightly worse than those for the best nonconnectivity metrics. For some models, OL_3 results were even the best among all of the considered metrics.

Appendix A shows Java code examples for classes that have, contradictory to other considered metrics, different PCCC cohesion values as confirmed by intuition.

In conclusion, the results indicate that the PCCC metric is much better than any of the other considered metrics in terms of discriminative power. This demonstrates that PCCC achieved one of its primary design goals, namely, sensitivity in differentiating the cohesion values of classes with different connectivity patterns. From this perspective, PCCC is the best cohesion indicator. Next, we evaluated the performance of PCCC in comparison to other metrics from a different point of view, i.e., the ability to detect faulty classes.

5. Fault-Detection Empirical Study

We performed two analyses to explore the correlations among the considered metrics and their relative strengths in detecting faulty classes. The first analysis investigated the correlations among 18 cohesion metrics, including PCCC and well-known connectivity and nonconnectivity metrics, and explored the orthogonal dimensions within this set of cohesion metrics using principal-component analysis (Dunteman 1989). The goal was to confirm that connectivity metrics, and specifically PCCC, do indeed contribute new information. The second analysis investigated the ability of the 18 metrics to predict faulty classes. Metrics were first considered individually and then as combinations of predictors. The analysis aimed to (1) investigate whether connectivity metrics are better or worse than nonconnectivity metrics in predicting the presence of faults in classes and (2) specifically explore whether PCCC is better or worse than other connectivity metrics, considered individually or in combination, in predicting faulty classes. Based on a commonly accepted assumption that has widely been used in many studies (e.g., Briand et al. 1998, Briand et al. 2001b, Gyimothy et al. 2005, Aggarwal et al. 2007, and Marcus et al. 2008), a metric that better detects fault occurrences more effectively captures cohesion. Building fault-prediction models requires accounting for many other factors besides cohesion, which are outside of the scope of this study. Instead, our analysis intends to determine if there is empirical evidence, whether direct or indirect, that PCCC is indeed a well-defined measure that is complementary to or better than existing comparable cohesion metrics, or both.

5.1. Software systems and metrics

Four open-source Java software systems from different domains were selected, including Art of Illusion v.2.5 (Illusion 2009), GanttProject v.2.0.5 (GanttProject 2009), JabRef v.2.3 beta 2 (JabRef 2009), and Openbravo v.0.0.24 (Openbravo 2009). The first system, Art of Illusion, is a 3D modeling, rendering, and animation studio system. It includes 481 classes and approximately 88 K lines of code (LOC). The second system, GanttProject, is a project-scheduling application that features resource management, calendaring, and importing or exporting (MS Project, HTML, PDF, and spreadsheets). It consists of 468 classes and about 39 KLOC. The third system, JabRef, is a graphical application for managing bibliographical databases. It consists of 569 classes and about 48 KLOC. Finally, the fourth system is Openbravo, which is a point-of-sale application that is designed for touch screens. It consists of 447 classes and about 36 KLOC. These systems were randomly selected from http://sourceforge.net. The restrictions that were placed on the selection of these systems were that they (1) are implemented using Java, (2) are relatively large in terms of the number of classes, (3) are from different domains, and (4) have available source code and fault repositories.

This study applied seven connectivity and 11 nonconnectivity metrics to the classes of the selected systems. The seven connectivity metrics included PCCC, CBMC, ICBMC, OL_2, OL_3, LCOM3, and LCOM4. The first five metrics have not been previously empirically studied in terms of fault-prediction power. The eleven nonconnectivity metrics consisted of LCOM1, LCOM2, LCOM5, Coh, TCC, LCC, DC_D, DC_I, LSCC, CC, and SCOM. These metrics were selected because they measure cohesion at the same LLD level as the connectivity metrics, and, therefore, all of the selected metrics were based on the same information, namely, information concerning the references of attributes by methods. In addition, these metrics have been extensively studied and compared to one another (Briand et al. 1999, Briand et al. 2000, Briand et al. 2001b, Marcus et al. 2008, Al Dallal and Briand 2010, Al Dallal and Briand 2012, Al Dallal 2011b, Al Dallal 2011a); hence, our results can be compared to those that have been obtained in previous empirical studies. These metrics were selected to compare the fault-prediction power of connectivity and nonconnectivity metrics.

We applied the considered metrics to 1,036 selected classes among the 1,965 classes from the four open-source systems. We excluded all classes for which at least one of the metrics was undefined. First, we excluded all classes that did not contain any attributes (523 classes) or any methods (14 classes) because their CBMC, ICBMC, OL_n, LCOM5, Coh, CC, and SCOM values are undefined. Out of the remaining classes (1,428), we excluded all classes that exclusively consisted of special methods (250 classes) because their CBMC, ICBMC, and OL_n values are undefined. Next, we excluded all classes that consisted of single methods (130 classes) because their LCOM1, LCOM2, TCC, LCC, DC_D, DC_I, CC, and SCOM values are undefined. Finally, classes that consisted of single nonconstructor methods (12 classes) were excluded because their TCC and LCC values are undefined. As a result, all considered connectivity metrics have defined values for 1,178 classes. Among these classes, 1,036 classes have defined values for all of the considered connectivity and nonconnectivity metrics. Excluding the classes that had undefined cohesion values using some of the considered metrics allowed us to perform the same analysis for all metrics on the same set of classes and therefore compare their results in an unbiased manner. Interfaces were also excluded because all of the considered metrics are undefined in this case.

We extended our own Java tool, which was applied to obtain our previously reported results (Al Dallal and Briand 2010, Al Dallal and Briand 2012, Al Dallal 2011a, Al Dallal 2011b), to automate the cohesion measurement process for Java classes using the eighteen considered metrics. The extension implements the cohesion measurement using the connectivity metrics including PCCC. Our tool analyzed the Java source code, extracted the required information to build the models, and calculated the cohesion values using the considered metrics. Table 3 reports the descriptive statistics for each cohesion measure, including the minimum, 25% quartile, mean, median, 75% quartile, maximum value, and standard deviation. As expected, the LCOM-based metrics are not normalized and they feature extreme outliers because of the existence of the accessor methods that typically reference single attributes (Briand et al. 2001b).

The mean and standard deviation results for CBMC, ICBMC, OL_2, and OL_3 were almost identical. We investigated this issue in more detail and found that, after removing the special methods, 77.3% of the classes had disjointed reference graphs and 11.5% of the classes had fully connected reference graphs. Using these four metrics, the cohesion values of these classes were set, by definition, to zero and one, respectively. This empirically demonstrates the weakness of these four metrics in differentiating between classes of different connectivity patterns. Note that only 4.8% of the considered classes have PCCC values of zero, and these are the classes in which the nonspecial methods do not reference any attributes.

Metric	Min	Max	25%	Med	75%	Mean	Std Dev
PCCC	0	1	0.00	0.03	0.58	0.29	0.41
CBMC	0	1	0.00	0.00	0.00	0.16	0.33
ICBMC	0	1	0.00	0.00	0.00	0.13	0.32
OL_2	0	1	0.00	0.00	0.00	0.16	0.33
OL_3	0	1	0.00	0.00	0.00	0.14	0.32
LCOM3	0	13	1.00	1.00	1.00	1.21	0.82
LCOM4	0	13	1.00	1.00	1.00	1.20	0.81
LCOM1	0	3320	3.00	13.00	41.00	59.86	189.07
LCOM2	0	2812	0.00	4.00	25.00	39.02	145.23
LCOM5	0	2	0.57	0.76	0.88	0.71	0.27
Coh	0	1	0.22	0.36	0.56	0.40	0.24
TCC	0	1	0.17	0.42	0.75	0.47	0.35
LCC	0	1	0.19	0.55	1.00	0.56	0.38
DC_D	0	1	0.28	0.50	0.71	0.50	0.30
DC_I	0	1	0.33	0.65	1.00	0.62	0.34
LSCC	0	1	0.04	0.11	0.26	0.20	0.23
CC	0	1	0.08	0.18	0.33	0.25	0.24
SCOM	0	1	0.09	0.22	0.44	0.31	0.28

Table 3. Descriptive statistics for the cohesion metrics.

5.2. Correlation and principal-component analyses

Principal component analysis (PCA) (Dunteman 1989) is a statistical technique that is applied here to demonstrate that connectivity metrics, specifically PCCC, capture new measurement dimensions. In PCA, the underlying orthogonal dimensions that explain the relations between the cohesion metrics are identified and understood (Marcus et al. 2008). For each pair of considered cohesion metrics, we used the Mahalanobis distance (Barnett and Lewis 1994) to detect outliers and found that removing the outliers did not significantly impact the final PCA results. Table 4 depicts the nonparametric Spearman correlation coefficients (Siegel and Castellan 1988) among the considered cohesion metrics, accounting for all four systems. All of the coefficients are statistically significant (p-value < 0.05 for all pairs). The connectivity metrics CBMC, ICBMC, OL_2, and OL_3 have perfect correlations of one among themselves, which means that they are monotonically related, and they are weakly or moderately correlated with other metrics. LCOM3 and LCOM4 are highly correlated with one another and are weakly correlated with other connectivity metrics. Finally, PCCC is weakly or moderately correlated with other connectivity and nonconnectivity metrics.

Metric	CBMC	ICBMC	OL2	OL3	LCOM3	LCOM4	LCOM1	LCOM2	LCOM5	Coh	TCC	LCC	DCD	DCI	LSCC	CC	SCOM
PCCC	0.47	0.47	0.47	0.47	-0.16	-0.14	-0.66	-0.56	-0.62	0.73	0.26	0.19	0.32	0.23	0.65	0.54	0.59
CBMC	1.00	1.00	1.00	1.00	-0.18	-0.17	-0.43	-0.43	-0.57	0.56	0.39	0.40	0.38	0.32	0.56	0.51	0.55
ICBMC	1.00	1.00	1.00	1.00	-0.18	-0.17	-0.43	-0.43	-0.57	0.56	0.39	0.40	0.38	0.31	0.56	0.52	0.55
OL_2	1.00	1.00	1.00	1.00	-0.18	-0.17	-0.43	-0.43	-0.57	0.56	0.39	0.40	0.38	0.32	0.56	0.52	0.55
OL_3	1.00	1.00	1.00	1.00	-0.18	-0.17	-0.43	-0.43	-0.57	0.56	0.39	0.40	0.38	0.32	0.56	0.52	0.55
LCOM3	-0.18	-0.18	-0.18	-0.18	1.00	0.97	0.29	0.31	0.22	-0.23	-0.09	-0.08	-0.15	-0.11	-0.23	-0.13	-0.27
LCOM4	-0.17	-0.17	-0.17	-0.17	0.97	1.00	0.28	0.30	0.21	-0.21	-0.09	-0.08	-0.15	-0.12	-0.21	-0.12	-0.25
LCOM1	-0.43	-0.43	-0.43	-0.43	0.29	0.28	1.00	0.78	0.51	-0.71	-0.30	-0.20	-0.43	-0.32	-0.62	-0.53	-0.62
LCOM2	-0.43	-0.43	-0.43	-0.43	0.31	0.30	0.78	1.00	0.65	-0.75	-0.52	-0.42	-0.67	-0.56	-0.75	-0.74	-0.80
LCOM5	-0.57	-0.57	-0.57	-0.57	0.22	0.21	0.51	0.65	1.00	-0.94	-0.52	-0.44	-0.63	-0.51	-0.97	-0.88	-0.90
Coh	0.56	0.56	0.56	0.56	-0.23	-0.21	-0.71	-0.75	-0.94	1.00	0.48	0.37	0.60	0.46	0.96	0.84	0.89
TCC	0.39	0.39	0.39	0.39	-0.09	-0.09	-0.30	-0.52	-0.52	0.48	1.00	0.93	0.86	0.75	0.57	0.64	0.64
LCC	0.40	0.40	0.40	0.40	-0.08	-0.08	-0.20	-0.42	-0.44	0.37	0.93	1.00	0.80	0.81	0.47	0.56	0.58
DC_D	0.38	0.38	0.38	0.38	-0.15	-0.15	-0.43	-0.67	-0.63	0.60	0.86	0.80	1.00	0.90	0.70	0.76	0.78
DC_I	0.32	0.31	0.32	0.32	-0.11	-0.12	-0.32	-0.56	-0.51	0.46	0.75	0.81	0.90	1.00	0.56	0.64	0.71
LSCC	0.56	0.56	0.56	0.56	-0.23	-0.21	-0.62	-0.75	-0.97	0.96	0.57	0.47	0.70	0.56	1.00	0.94	0.94
CC	0.51	0.52	0.52	0.52	-0.13	-0.12	-0.53	-0.74	-0.88	0.84	0.64	0.56	0.76	0.64	0.94	1.00	0.89

Table 4. Spearman rank correlations among the cohesion metrics.

We applied the *varimax* rotation technique (Jolliffe 1986, Snedecor and Cochran 1989) to obtain the principal components (PCs). In this technique, the eigenvectors and eigenvalues (loadings) are calculated and used to form the PC loading matrix. Table 5 depicts the PCA results: the loading matrix depicts five PCs that captured 90.24% of the data set variance. In addition, Table 5 depicts the eigenvalues (i.e., measures of the variances of the PCs), their percentages, and the cumulative percentage. High coefficients (loadings) for each PC indicate the influential metrics that contribute to the captured dimension. Coefficients in excess of 0.5 are highlighted in boldface in Table 5. Based on an analysis of these coefficients, the resulting PCs can then be interpreted as follows:

PC1: PCCC, CBMC, ICBMC, OL_2, OL_3, LCOM5, Coh, TCC, LCC, DC_D, DC_I, LSCC, CC, and SCOM. These metrics are normalized, and, therefore, they do not feature extreme outliers.

PC2: CBMC, ICBMC, OL_2, OL_3, DC_D, and DC_I. These metrics account for indirect relations between methods either represented in the class reference graph or due to method invocations.

PC3: LCOM1, LCOM2, LCOM3, and LCOM4. These metrics measure the lack of cohesion and are not normalized.

PC4: LCOM3 and LCOM4. These metrics are not normalized and they are based on counting the number of connected components in the class reference graph.

PC5: PCCC. This metric is based on counting the number of simple paths in the class reference graph.

	PC1	PC2	PC3	PC4	PC5
Eigenvalue	9.17	2.66	2.62	1.25	0.54
Percent	50.95	14.78	14.58	6.93	3.00
Cum. Per.	50.95	65.73	80.31	87.24	90.24
PCCC	**0.62**	0.32	-0.20	0.02	**0.68**
CBMC	**0.81**	**0.50**	-0.10	-0.16	-0.10
ICBMC	**0.79**	**0.53**	-0.12	-0.16	-0.07
OL_2	**0.82**	**0.50**	-0.11	-0.15	-0.10
OL_3	**0.80**	**0.52**	-0.12	-0.16	-0.07
LCOM3	-0.26	0.41	**0.67**	**0.52**	0.01
LCOM4	-0.25	0.42	**0.67**	**0.52**	0.01
LCOM1	-0.26	0.29	**0.72**	-0.48	0.02
LCOM2	-0.28	0.36	**0.72**	-0.41	0.04
LCOM5	**-0.88**	0.00	-0.08	-0.19	0.11
Coh	**0.90**	0.00	-0.06	0.22	0.02
TCC	**0.71**	-0.46	0.34	-0.08	0.01
LCC	**0.61**	-0.47	0.43	-0.21	-0.01
DC_D	**0.76**	**-0.50**	0.32	0.00	0.04
DC_I	**0.61**	**-0.53**	0.41	-0.09	0.11
LSCC	**0.92**	0.05	0.03	0.15	-0.08
CC	**0.90**	-0.05	0.11	0.16	-0.08
SCOM	**0.91**	-0.15	0.08	0.12	0.01

Table 5. The loading matrix.

The PCA results demonstrate that the PCCC metric captures a measurement dimension of its own because it is the only significant factor in PC5, although it also contributes to PC1. This result supports the fact that PCCC captures cohesion aspects that are not addressed by any of the other cohesion metrics that have been considered in this analysis, thus confirming the results of the correlation analysis. In addition, except for LCOM3 and LCOM4, the results demonstrate that the other connectivity metrics share measurement dimensions with other nonconnectivity metrics (i.e., PC1, PC2, and PC3), and they do not define their own measurement dimensions.

5.3. Detecting faulty classes

We applied logistic regression (Hosmer and Lemeshow 2000), which is a standard statistical technique, to investigate the relationship between the values of collected metrics and the extent to which a class is fault-prone. This technique is based on maximum-likelihood estimation, and it has been ubiquitously applied to identifying fault-prone classes (i.e., Briand et al. 1998, Briand et al. 2001, Gyimothy et al. 2005, and Marcus et al. 2008). The application of other analysis techniques (e.g., Briand and Wuest 2002, Arisholm et al. 2010) is outside of the scope of this work. In logistic regression, two types of variables are used: dependent and independent. A dependent variable can only take discrete values and is binary in the context of predicting fault-prone classes. Independent, or explanatory, variables are used to explain and predict the dependent variables. The logistic regression model is univariate when it includes only one explanatory variable and multivariate when it includes several explanatory variables. In our analysis, the dependent variable indicates the presence of one or more faults in a class, whereas the explanatory variables are the cohesion metrics. Univariate regression was used to individually study the fault prediction of each metric. This allowed for the comparison of the fault-prediction power of the 18 metrics under consideration. Alternately, multivariate regression was used to study the combined fault prediction of several cohesion metrics, which allowed testing for whether PCCC improved the fit of these combinations.

We collected fault data for the classes in the considered software systems from publicly available fault repositories. The developers of the considered systems used two different on-line Version Control Systems (VCS) that keep track of source code changes. The changes, called revisions, are due to either detected faults or required feature improvements. The developers of the considered systems used two different ways to organize the revisions: according to either the revision time stamp (used for the Art of Illusion, JabRef, and Openbravo systems) or the system package hierarchy (used for the GanttProject system). For the former organization method, each revision is associated with the revision date and a report including the revision description and a list of classes involved in this change. For each class, the system shows the revised code for that revision. For the later organization method, the tracking system provides the package tree hierarchy in which the classes represent the leaf nodes, and when any class is selected, its modification history is provided. The history reports all the revisions of the class, and for each revision, the history reports the revision identification, date, description, and revised pieces of code. The number of revisions reported in the VCS for each of the four considered systems is reported in Table 6.

It is required to inspect the contents of the VCS to count the number of faults detected for each class in each of the considered systems. When inspecting the contents of the VCS, we associated with each class a counter for its number of detected faults, which is initially set to zero. The counting of the number of faults for each class is performed accumulatively while browsing the corresponding VCS. Inspecting the contents of the time-stamp-based VCS requires inspecting each individual revision and reading the revision description to identify whether the revision is due to a detected fault. The developers of the considered systems used phrases such as *fix bug*, *fix mistake*, *fix problem*, or *corrected* to show that this revision is due to a detected fault. If the revision is due to a detected fault, the counter for each class involved in the revision will be incremented. As a result, the time required to perform the fault data collection for systems that use a time-stamp-based VCS depends on the number of revisions and the number of classes involved in the revisions that are due to detected faults. On the other hand, inspecting the contents of the package-hierarchical-based VCS requires tracing the tree-organized package hierarchy starting from the root package down to each leaf node representing a class in the considered system. Once the class link is reached, it is required to inspect the list of revisions of the class and read the description of each revision to identify whether the revision is due to a detected fault. The number of revisions due to detected faults is counted for each class. As a result, the time required to perform the fault data collection for systems that use package-hierarchical-based VCS depends on the complexity of the package hierarchy (i.e., the lengths of the paths from the root to the leaves of the hierarchical tree and the number of nodes in the tree) and the total number of revisions performed on each class.

Two research assistants, one with a B.Sc. in computer science and six years of experience in software development activities and the other with a B.Sc. and Master both in computer science and two years of experience as a research assistant; each of them, manually and independently inspected the contents of the VCS for the four considered systems, read the description of each revision, identified faults and their involved classes, and reported the results in Excel spreadsheets. The author of this paper used Excel formulas to compare the results reported in the spreadsheets and rechecked the results in which the two assistants differed to choose the correct one. For each of the considered systems, Table 6 reports the number of considered classes, number of faulty classes, the sum of the number of faults involved in the classes, and the number of classes that involved a single detected fault. The classes were classified as being fault-free or as having at least one fault because a relatively small percentage of classes contained two or more faults.

	Art of Illusion	Gantt-Project	JabRef	Open-bravo	All systems
No. of reported revisions	518	1,910	7,837	1,270	11,535
No. of considered classes	356	241	216	223	1,036
No. of faulty classes	185	96	186	180	620
No. of faults	209	271	408	275	1,163
No. of classes having one fault	172	61	148	129	510

Table 6: Fault data extracted from the VCS of the considered systems

To collect the metric and fault data from the same version of source code, we traced the contents of the VCS for each faulty class and obtained the source code of the class before correcting the first detected fault. It is important to note that the entire data reported and analyzed in this section, including the data previously reported in Tables 3, 4, and 5, is based on the version of the source code before being corrected.

Table 7 depicts the univariate regression results for each metric and each of the considered systems. The first seven listed metrics are the considered connectivity metrics and the rest are the nonconnectivity metrics. The estimated regression coefficients are reported along with their 95% confidence intervals. 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 Table 3, the considered metrics had significantly different standard deviations. Therefore, in order 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. In this case, the resulting coefficients reported in Table 7 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 variable. The standard error of a metric is defined as the standard deviation of the sampling distribution that is associated with the metric.

Metric	System	Std. Coeff.	Odd ratio	Std. Error	95% Confidence Interval Coeff.	95% Confidence Interval odd ratio	p-value	Precision	Recall	ROC area
PCCC	Art of Illus.	-0.66	0.51	0.13	[-0.92,-0.41]	[0.40,0.66]	< 0.0001	68.9	63.5	63.8
	Gantt	-0.51	0.60	0.14	[-0.75,-0.23]	[0.46,0.71]	0.0003	59.0	60.6	60.8
	JebRef	-0.63	0.53	0.18	[-0.99,-0.28]	[0.37,0.76]	0.0005	74.2	86.1	60.3
	Openbravo	-0.71	0.49	0.16	[-1.02,-0.40]	[0.36,0.67]	< 0.0001	65.2	80.7	68.3
CBMC	Art of Illus.	-0.65	0.52	0.17	[-0.98,-0.31]	[0.38,0.73]	0.0001	66.1	59.0	55.1
	Gantt	-0.44	0.64	0.15	[-0.73,-0.15]	[0.48,0.86]	0.0030	36.2	60.2	56.7
	JebRef	-0.64	0.53	0.16	[-0.96,-0.32]	[0.38,0.72]	< 0.0001	74.2	86.1	58.7
	Openbravo	-0.54	0.58	0.15	[-0.83,-0.25]	[0.43,0.78]	0.0002	70.9	79.4	58.3
ICBMC	Art of Illus.	-0.63	0.53	0.19	[-1.00,-0.26]	[0.37,0.77]	0.0007	68.9	59.3	55.1
	Gantt	-0.36	0.70	0.15	[-0.65,-0.07]	[0.52,0.93]	0.0137	36.2	60.2	57.1
	JebRef	-0.64	0.53	0.15	[-0.94,-0.34]	[0.39,0.71]	< 0.0001	73.9	84.3	57.2
	Openbravo	-0.52	0.59	0.14	[-0.80,-0.24]	[0.45,0.79]	0.0003	70.9	79.4	58.8
OL_2	Art of Illus.	-0.61	0.54	0.16	[-0.93,-0.30]	[0.35,0.74]	0.0001	65.0	58.1	55.1
	Gantt	-0.44	0.64	0.15	[-0.73,-0.15]	[0.48,0.86]	0.0033	36.2	60.2	56.8
	JebRef	-0.63	0.53	0.16	[-0.95,-0.32]	[0.39,0.73]	< 0.0001	74.2	86.1	58.7
	Openbravo	-0.53	0.59	0.15	[-0.81,-0.24]	[0.44,0.79]	0.0003	65.2	80.7	58.1
OL_3	Art of Illus.	-0.63	0.53	0.18	[-0.99,-0.28]	[0.37,0.75]	0.0003	66.5	57.9	55.1
	Gantt	-0.39	0.68	0.15	[-0.68,-0.10]	[0.51,0.90]	0.0082	36.2	60.2	57.0
	JebRef	-0.64	0.53	0.15	[-0.95,-0.37]	[0.39,0.71]	< 0.0001	73.9	84.3	57.8
	Openbravo	-0.52	0.59	0.14	[-0.80,-0.24]	[0.45,0.79]	0.0003	70.9	79.4	58.6
LCOM3	Art of Illus.	0.42	1.52	0.18	[0.07,0.76]	[1.07,2.15]	0.0180	53.1	51.7	52.5
	Gantt	0.23	1.26	0.14	[-0.04,0.50]	[0.96,1.66]	0.0975	52.1	59.3	53.4
	JebRef	0.24	1.27	0.27	[-0.21,0.77]	[0.75,2.15]	0.3779	74.2	86.1	50.4
	Openbravo	-0.01	0.99	0.17	[-0.33,0.32]	[0.72,1.37]	0.9576	65.0	79.8	39.1

Table 7. Univariate logistic regression results.

Metric	System	Std. Coeff.	Odd ratio	Std. Error	95% Confidence Interval Coeff.	95% Confidence Interval odd ratio	p-value	Precision	Recall	ROC area
LCOM4	Art of Illus.	0.38	1.46	0.17	[0.04,0.72]	[1.04,2.05]	0.0277	51.2	50.6	51.6
	Gantt	0.23	1.26	0.14	[-0.04,0.51]	[0.96,1.66]	0.0946	52.9	59.3	53.5
	JebRef	0.18	1.19	0.26	[-0.33,0.68]	[0.72,1.98]	0.4954	74.2	86.1	48.7
	Openbravo	-0.02	0.98	0.16	[-0.33,0.30]	[0.72,1.34]	0.9006	65.0	79.8	38.9
LCOM1	Art of Illus.	1.65	5.21	0.37	[0.92,2.4]	[2.50,10.83]	< 0.0001	64.5	62.1	68.0
	Gantt	0.52	1.68	0.24	[-0.06,0.99]	[1.06,2.68]	0.0279	62.1	61.8	65.2
	JebRef	0.26	1.30	0.41	[-0.54,1.06]	[0.59,2.88]	0.5208	74.2	86.1	56.7
	Openbravo	0.68	1.97	0.43	[-0.16,1.51]	[0.85,4.55]	0.1127	65.2	80.7	61.7
LCOM2	Art of Illus.	1.67	5.32	0.43	[0.82,2.5]	[2.27,12.49]	0.0001	67.3	63.8	67.5
	Gantt	0.51	1.67	0.26	[-0.01,1.03]	[1.00,2.79]	0.0523	61.5	61.4	61.5
	JebRef	0.16	1.18	0.33	[-0.49,0.81]	[0.61,2.26]	0.6230	74.2	86.1	51.2
	Openbravo	0.67	1.96	0.43	[-0.18,1.52]	[0.84,4.58]	0.1221	65.2	80.7	65.4
LCOM5	Art of Illus.	0.61	1.84	0.12	[0.37,0.86]	[1.44,2.35]	< 0.0001	63.9	63.5	65.8
	Gantt	0.33	1.34	0.14	[-0.06,0.61]	[1.06,1.84]	0.0169	50.1	58.1	60.0
	JebRef	0.56	1.76	0.21	[0.16,0.97]	[1.17,2.64]	0.0063	74.2	86.1	62.2
	Openbravo	0.76	2.14	0.17	[0.42,1.10]	[1.53,3.00]	< 0.0001	77.4	81.2	72.0
Coh	Art of Illus.	-0.71	0.49	0.12	[-0.96,-0.47]	[0.38,0.62]	< 0.0001	65.0	64.9	68.7
	Gantt	-.045	0.64	0.14	[-0.73,-0.17]	[0.48,0.84]	0.0016	57.3	60.2	62.3
	JebRef	0.52	0.59	0.20	[-0.91,-0.14]	[0.40,0.87]	0.0078	74.2	86.1	61.8
	Openbravo	-0.73	0.48	0.17	[-1.06,-0.39]	[0.35,0.67]	< 0.0001	77.4	81.2	70.9
TCC	Art of Illus.	-0.74	0.48	0.12	[-0.97,-0.50]	[0.38,0.60]	< 0.0001	66.9	66.9	68.4
	Gantt	-0.33	0.72	0.14	[-0.60,-0.06]	[0.55,0.94]	0.0167	36.2	60.2	56.9
	JebRef	-0.46	0.63	0.20	[-0.86,-0.07]	[0.42,0.93]	0.0209	74.2	86.1	58.3
	Openbravo	-0.48	0.62	0.17	[-0.81,-0.15]	[0.44,0.86]	0.0046	65.2	80.7	61.8
LCC	Art of Illus.	-0.52	0.59	0.11	[-0.75,-0.30]	[0.47,0.74]	< 0.0001	63.0	62.9	64.0
	Gantt	-0.28	0.75	0.13	[-0.55,-0.02]	[0.58,0.98]	0.0353	36.2	60.2	54.9
	JebRef	-0.52	0.60	0.21	[-0.94,-0.07]	[0.39,0.91]	0.0161	74.2	86.1	58.7
	Openbravo	-0.39	0.68	0.18	[-0.73,-0.04]	[0.48,0.96]	0.0269	65.2	80.7	58.8
DC_D	Art of Illus.	-0.50	0.61	0.11	[-0.72,-0.27]	[0.49,0.76]	< 0.0001	62.1	62.1	63.2
	Gantt	-0.25	0.78	0.13	[-0.52,0.01]	[0.60,1.01]	0.0598	47.4	58.9	55.0
	JebRef	-0.22	0.80	0.20	[-0.61,0.17]	[0.55,1.18]	0.2664	74.2	86.1	49.3
	Openbravo	-0.53	0.59	0.17	[-0.87,-0.19]	[0.42,0.83]	0.0022	65.2	80.7	64.1
DC_I	Art of Illus.	-0.35	0.70	0.11	[-0.57,-0.14]	[0.57,0.87]	0.0014	57.9	57.9	59.3
	Gantt	-0.12	0.88	0.13	[-0.39,0.13]	[0.68,1.14]	0.3454	36.2	60.2	50.9
	JebRef	-0.20	0.82	0.20	[-0.60,0.20]	[0.55,1.22]	0.3241	74.2	86.1	48.6
	Openbravo	-0.53	0.59	0.19	[-0.90,-0.16]	[0.41,0.85]	0.0045	65.2	80.7	61.9
LSCC	Art of Illus.	-0.77	0.46	0.15	[-1.07,-0.48]	[0.34,0.62]	< 0.0001	62.3	61.2	69.6
	Gantt	-0.46	0.63	0.16	[-0.77,-0.14]	[0.46,0.86]	0.0041	36.2	60.2	61.9
	JebRef	-0.37	0.69	0.17	[-0.71,-0.04]	[0.49,0.96]	0.0296	74.2	86.1	57.9
	Openbravo	-0.64	0.52	0.15	[-0.94,-0.35]	[0.39,0.71]	< 0.0001	77.4	81.2	72.3
CC	Art of Illus.	-0.75	0.47	0.14	[-1.03,-0.14]	[0.36,0.61]	< 0.0001	59.7	59.3	67.5
	Gantt	-0.42	0.66	0.15	[-0.72,-0.12]	[0.49,0.89]	0.0058	56.3	60.2	60.4
	JebRef	-0.28	0.76	0.18	[-0.62,0.07]	[0.57,1.07]	0.1194	74.2	86.1	54.3
	Openbravo	-0.75	0.48	0.16	[-1.06,-0.43]	[0.35,0.65]	< 0.0001	77.4	81.2	73.0
SCOM	Art of Illus.	-0.64	0.52	0.12	[-0.88,-0.41]	[0.41,0.67]	< 0.0001	61.8	61.5	67.8
	Gantt	-0.60	0.55	0.16	[-0.90,-0.29]	[0.40,0.75]	0.0001	55.9	58.1	61.7
	JebRef	-0.31	0.73	0.18	[-0.67,0.05]	[0.51,1.05]	0.0878	74.2	86.1	54.1
	Openbravo	-0.71	0.49	0.16	[-1.03,-0.40]	[0.36,0.67]	< 0.0001	77.0	80.7	70.4

Table 7 (continuation). Univariate logistic regression results

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. In order 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, and, therefore, in our case, a metric was determined to not be a statistically significant fault predictor if its p-value was greater than $0.05/18 = 0.003$.

A common practice is to use *odd ratios* (Hosmer and Lemeshow 2000) to help interpret coefficients as those are not linearly related to the probability of fault occurrences. In our context, an odd ratio captures how less (more) likely it is for a fault to occur when the corresponding (lack of) cohesion metric changes by one standard deviation. We report odd ratios and their 95% confidence interval in Table 7. As an example, for PCCC (applied on Art of Illusion classes), the probability of fault occurrence when there is an increase of one standard deviation in PCCC is estimated to decrease by 49%. Those can be easily compared across cohesion metrics.

Table 7 reports the traditional precision and recall evaluation criteria (Olson and Delen 2008) used to evaluate the prediction accuracy of logistic regression models. Precision measures the percentage of the faulty classes correctly classified as faulty. It is defined as the number of classes correctly classified as faulty, divided by the total number of classes classified as faulty. Recall measures the percentage of the faulty classes correctly or incorrectly classified as faulty. It is defined as the number of classes correctly classified as faulty, divided by the actual number of faulty classes. A probability threshold has to be selected to predict classes as faulty or not. As in (Briand et al. 2000), we classified a class as faulty if its predicted probability of containing a fault is higher than a threshold selected such that the percentage of classes that are classified as faulty is roughly the same as the percentage of classes that actually are faulty.

Since 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 a better evaluation criterion than standard precision and recall. In the context of fault-prediction problems, the ROC curve graphically represents the ratio of classes correctly classified as faulty versus the ratio of classes incorrectly classified as faulty at different thresholds. The ability of the model to correctly rank classes as faulty or non-faulty is represented by the area under the ROC curve. A perfect model that classifies all classes correctly has a ROC area of 100%. The larger the ROC area, the better the model is in classifying classes. To obtain a more realistic assessment of the predictive ability of the metrics, we used cross-validation, 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 the regression model.

Because of the space limitation, the results reported in Table 7 are only for the metrics defined as-is in terms of inclusion or exclusion of special methods (i.e., constructors, destructors, and access and delegation methods). As a result, for connectivity metrics, except for LCOM3 and LCOM4, special methods were excluded. In Appendix B, only the connectivity metrics are considered, which allows for including classes that consisted of single methods (130 classes) in the analysis. Note that these classes were excluded because their cohesion values were undefined using some of the nonconnectivity metrics. Thus, the number of considered classes in the analysis for which its results are reported in Appendix B is 1,178. The results are classified into four groups. The first group includes the results for the connectivity metrics that were applied when the special methods were included. The second group includes the results for the same metrics constructor methods. The third group depicts the results for the same metrics excluding access methods. The fourth group depicts the results for the same metrics that were applied when all special methods were excluded.

The results in Table 7 and Appendix B lead to the following conclusions:

1. As expected, the estimated regression coefficients for the inverse cohesion measures LCOM1, LCOM2, LCOM3, LCOM4, and LCOM5 were positive, whereas those for the straight cohesion measures (i.e., the other considered metrics) were negative. In each case, this indicates an increase in the predicted probability of fault detection as the cohesion of the class decreases.
2. Except for LCOM3 and LCOM4, the considered cohesion metrics were statistically significant at $\alpha = 0.003$ (i.e., their coefficients were significantly different from zero) for most or all of the considered systems. As a result, only LCOM3 and LCOM4 were determined to not be statistically significant fault predictors. Note that PCCC is one of the metrics with the smallest p-value (which indicates the statistical significance of the regression coefficient) among all of the considered cohesion metrics.
3. PCCC had, by a considerable margin, the largest standardized coefficient among all of the considered connectivity cohesion metrics for most of the considered systems. This is confirmed by the smaller odds ratios, thus suggesting that an increase in PCCC has a stronger impact in reducing the probability of fault occurrence.
4. Some other connectivity metrics were slightly better than PCCC in terms of precision; however, PCCC was clearly the best metric among the seven considered connectivity metrics in terms of the ROC area. As mentioned earlier, the ROC area is a better evaluation criterion than precision or recall. As a result, from this perspective, PCCC is the best, among the considered connectivity metrics, in terms of fault-prediction power. The values for ROC area for PCCC are classified as fair (Hosmer and Lemeshow 2000), whereas the corresponding values for the other connectivity metrics range from poor to no good. Note that the fair classification is not practically acceptable. However, the resulting ROC area values do not mean that the cohesion metrics are useless in practice. This is due to the fact that in practice, cohesion is not applied alone to predict faulty classes. Instead, a prediction model including metrics that measure cohesion

and other quality attributes such as coupling and complexity is applied. Our goal here is to determine if there is empirical evidence, whether direct or indirect, that PCCC is better than existing, comparable cohesion metrics in predicting faulty classes. The resulting ROC area values indicate that PCCC potentially performs better than other connectivity metrics when it is included in a fault prediction model with other metrics that measure other quality attributes.

5. CBMC, ICBMC, OL_2, and OL_3 had almost the same values for the standardized coefficient, precision, recall, and ROC-area values. This indicates that these four metrics are almost the same in terms of their fault-prediction power.
6. Among the connectivity metrics, LCOM3 and LCOM4 had the worst ROC-area values. This indicates that these two metrics are the worst connectivity metrics in terms of fault-prediction power.
7. As depicted in Appendix B, including or excluding special methods in the computation of the connectivity metrics did not considerably change their standardized coefficient, precision, recall, and ROC-area values. This indicates that the fault-prediction power of these metrics was not considerably affected by including or excluding special methods. As a result, in practice, it is recommended to exclude special methods because such exclusion (1) simplifies the reference graph, (2) makes the computation of cohesion easier and faster, (3) better reflects the intuitive cohesion, and (4) does not significantly affect the prediction power.
8. For some of the considered systems, some nonconnectivity metrics, such as LSCC and Coh, are considerably better than any connectivity metric in terms standardized coefficient, precision, recall, and ROC-area values. Simultaneously, PCCC is better, from this perspective, than some nonconnectivity metrics, such as LCC, DC_D, and DC_I. Mostly, LCOM3 and LCOM4 are the worst among all of the considered metrics. As a result, no general conclusion can be drawn as to whether connectivity or nonconnectivity metrics are better in predicting faulty classes. In this case, the decision must be made for individual metrics.
9. Including the classes with single methods slightly improved the univariate regression results, as depicted in Table B4, compared to the results shown in Table 7. However, the general conclusions listed above apply also for the results given in Appendix B.

We used a backward-selection process to perform a multivariate regression analysis, wherein all of the considered metrics were first included into the model and then removed one by one as long as one metric had a p-value in excess of 0.05, starting with measures that exhibited higher p-values. The analysis included all considered connectivity and nonconnectivity metrics. This analysis aimed at exploring whether an optimal yet minimal multivariate model based on connectivity and nonconnectivity metrics would contain PCCC and whether the ROC area of the fault predication model would fall within the acceptable range (i.e., ROC area is greater than 70%), and therefore, it can be used in practice. In addition, this analysis investigated whether connectivity and nonconnectivity metrics are complementary in indicating class cohesion. The analysis results for the four considered systems are depicted in Table 8. For each system, Table 8 depicts the metrics remained in the prediction model as significant covariates and the multivariate regression analysis results of the model. The results indicate that PCCC is the only connectivity metric that contributes to multivariate fault prediction models of two of the considered systems. For both systems, the ROC area results fall within the acceptable range, which indicates that the model is a useful fault prediction model in practice.

System	Metrics	Std. Coeff.	Std. Error	p-value	Precision	Recall	ROC area
Art of Illusion	LCOM1	0.62	0.20	0.002	67.6	67.1	73.6
	TCC	-0.68	0.16	< 0.0001			
	DC_D	0.60	0.16	0.000			
	CC	-0.35	0.11	0.002			
Gantt	PCCC	-0.26	0.10	0.009	64.2	65.1	71.9
	LCC	-0.51	0.17	0.004			
	DC_I	0.82	0.20	< 0.0001			
	LSCC	1.14	0.32	0.000			
	SCOM	-1.59	0.38	< 0.0001			
JebRef	LCOM5	0.43	0.16	0.007	88.7	87.0	65.2
	LCC	-0.59	0.20	0.004			
	DC_D	0.61	0.23	0.009			
Openbravo	PCCC	-0.30	0.12	0.010	78.0	81.2	73.5
	LSCC	0.74	0.34	0.028			
	CC	-0.98	0.33	0.003			

Table 8. The multivariate model based on all of the considered metrics.

As a result, regarding the comparison between connectivity and nonconnectivit metrics, the empirical results above show that (1) the connectivity metrics measure cohesion aspects that are not covered by nonconnectivity metrics, and, therefore, they are complementary in indicating cohesion; (2) some connectivity metrics are better than some nonconnectivity metrics in detecting faulty classes and vice versa; and (3) connectivity and nonconnectivity metrics are complementary in detecting faulty classes. Regarding PCCC, the empirical results above show that PCCC (1) defines its own cohesion dimension, and, therefore, PCCC is complementary in indicating cohesion, (2) is the best among connectivity metrics in detecting faulty classes, and (3) is usually complementary to other nonconnectivity metrics in detecting faulty classes.

5.4. Threats to validity

A. Internal validity

Cohesion is one of the factors that has been shown to relate to the occurrence of faults. In order to detect faulty classes, other factors must also be considered (Arisholm et al. 2010); however, the present study was not aimed at detecting faulty classes. Instead, it intended to explore whether there is empirical evidence that metrics that are considered individually or in combination are strongly related to observable aspects of quality. If this is confirmed for a metric, it will be considered to be a potential well-defined cohesion measure.

B. External validity

The empirical study presented herein has several threats to the external validity regarding the selected systems on which the study was applied. These threats may restrict the generality and limit the interpretation of our results. The first threat is that all of the considered systems were implemented in Java. In addition, although it is common practice in the research community, the selected systems are open-source systems that may not be representative of all industrial domains. Note that 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). Finally, the classes that were included in the selected systems may not be representative in terms of their number and sizes. In order to generalize the results, similar large-scale evaluations should involve large-scale systems that are selected from different domains and written in different programming languages.

6. Conclusions and Future Work

This paper empirically compared connectivity- and non-connectivity-based class cohesion metrics in terms of their discriminative and fault-detection powers. Unexpectedly, the results demonstrate that most existing connectivity metrics are worse than most nonconnectivity metrics from these two viewpoints. In addition, most of the existing connectivity metrics violate one or more of the key cohesion properties, which raises questions about their usefulness as cohesion indicators. Therefore, we proposed a new connectivity metric, PCCC, which satisfies the necessary cohesion properties and is much better than the existing connectivity metrics in terms of both discriminative and fault-detection powers.

In practice, when coding editors indicate the cohesion value of the code while it is being written, developers are expected to take immediate actions to improve the quality of their code. When several implementations of the same code are provided, project managers are expected to choose the one with the highest quality. In addition, project managers can use quality indicators as factors to be considered in evaluating and rewarding developers. Finally, quality indicators can be used as pointers for classes that require more refactoring attention. For all of these four practical scenarios, developers and project managers need a cohesion metric that has a high degree of discriminative power to be able to distinguish between the quality of different classes or the quality of different implementations of the same class. Otherwise, when the difference is not noticed, developers can make incorrect refactoring decisions or perform coding actions that weaken the code quality. In addition, project managers may not be able to choose the best implementations or reward the best developers. From these perspectives, we believe that the proposed metric provides much more useful information to developers and project managers than any of the existing metrics because PCCC is nearly twice as effective as the best connectivity or nonconnectivity metric in terms of discriminative power.

The proposed metric is based on counting the number of simple paths in the reference graph that shows the relations between the attributes and methods in a class. Empirically, the metric was found to measure cohesion aspects that are not covered by any other connectivity or nonconnectivity metrics. In addition, the metric usually contributed to a fault-detection model based on cohesion metrics.

The proposed metric does not differentiate between the methods and attributes of different accessibility levels (i.e., public, private, package, and protected). In addition, the impact of including or excluding inherited attributes and methods on the computation of the proposed metric was not empirically studied. In the future, we plan to theoretically and empirically study the effects of inheritance on several cohesion metrics, including PCCC, for four different scenarios in which (1) both inherited methods and attributes are excluded, (2) only inherited attributes are excluded, (3) only inherited methods are excluded, and (4) both inherited methods and attributes are included. A tool will be developed to analyze the class hierarchy of an application so as to identify the inherited attributes and methods and build the other three versions from the typical version of the inheriting class that does not include inherited attributes and methods. As mentioned above, more empirical studies involving large industrial systems are needed to validate or refute the results presented herein.

Acknowledgments
The author would like to acknowledge the support of this work by Kuwait University Research Grant WI04/08. In addition, the author would like to thank Anas Abdin and Saqiba Sulman for assisting in collecting the cohesion and fault 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, A design-based cohesion metric for object-oriented classes, *International Journal of Computer Science and Engineering*, 2007, 1(3), pp. 195-200.

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, 53(9), pp. 914-928.

J. Al Dallal, Measuring the discriminative power of object-oriented class cohesion metrics, *IEEE Transactions on Software Engineering*, 2011b, 37(7), pp. 788-804.

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

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, 2012.

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*, 2010, 83(1), pp. 2-17.

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.

V. Barnett and T. Lewis, *Outliers in Statistical Data*, John Wiley and Sons, 3rd e, 1994, pp. 584.

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.

J. Bieman and L. Ott, Measuring functional cohesion, *IEEE Transactions on Software Engineering*, 20(8), 1994, pp. 644-657.

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, C. Bunse, and J. Daly, A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs, *IEEE Transactions on Software Engineering*, 27(6), 2001a, pp. 513-530.

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 , S. Morasca , and V. R. Basili, Defining and validating measures for object-based high-level design, *IEEE Transactions on Software Engineering*, 25(5), 1999, pp. 722-743.

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), 2001b, 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.

H. S. Chae, Y. R. Kwon, and D. Bae, Improving cohesion metrics for classes by considering dependent instance variables, *IEEE Transactions on Software Engineering*, 30(11), 2004, pp. 826-832.

Z. Chen, Y. Zhou, and B. Xu, A novel approach to measuring class cohesion based on dependence analysis, *Proceedings of the International Conference on Software Maintenance*, 2002, pp. 377-384.

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.

G. Dunteman, *Principal Components Analysis*, Saga University Paper No. 7-69, Saga Publications, US, pp. 96.

L. Fernández, and R. Peña, A sensitive metric of class cohesion, *International Journal of Information Theories and Applications*, 13(1), 2006, pp. 82-91.

GanttProject, http://sourceforge.net/projects/ganttproject/, August 2009

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.

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/, August 2009.

JabRef, http://sourceforge.net/projects/jabref/, August 2009

I. T. Jolliffe, *Pincipal Component Analysis*, Springer, 1986.

B. Kitchenham, S. L. Pfleeger, and N. Fenton, Towards a framework for software measurement validation, *IEEE Transactions on Software Engineering*, 21(12), 1995, pp. 929-944.

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. 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.

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

Openbravo, http://sourceforge.net/projects/openbravopos, August 2009.

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.

S. Siegel and J. Castellan, *Nonparametric Statistics for the Behavioral Sciences*, McGraw-Hill, 2nd edition, 1988.

G. Snedecor and W. Cochran, *Statistical Methods*, Blackwell Publishing Limited, 1989, 8th edition.

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.

J. Wang, Y. Zhou, L. Wen, Y. Chen, H. Lu, and B. Xu, DMC: a more precise cohesion measure for classes. *Information and Software Technology*, 47(3), 2005, pp. 167-180.

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.

Y. Zhou, B. Xu, J. Zhao, and H. Yang, ICBMC: an improved cohesion measure for classes, *Proc. of International Conference on Software Maintenance*, 2002, pp. 44-53.

Appendix A:

Figures A1 and A2 include two classes Person1 and Collection. Each method in class Person1 accesses one or more attribute and each attribute is accessed by one or more methods, whereas, in class Collection, one of the methods does not access any attribute and two of the attributes are not accessed by any method. Therefore, intuitively, class Person1 is more cohesive than class Collection. Among all of the considered metrics, only PCCC, Coh, and LCOM5 differentiate between the cohesion of the two classes. In some other examples such as classes Person2 and Person3 given in Figures A3 and A4, Coh and LCOM5 do not differentiate between the cohesion of the classes, whereas PCCC does. Intuitively, class Person3 is more cohesive than class Person2 because, all methods and attributes in class Person3 are directly or indirectly related, whereas date attribute and printDate method are not related with other attributes and methods in class Person 2. The cohesion values for the four classes are given in Table A1.

```
public class Person1{
        String firstName, lastName;
        Date date;
        void printName() {
            /*code that accesses firtName
              and lastName attributes*/
        }
        void printDate() {
            //code that accesses date attribute
        }
}
```

Figure A1: Person1 Java class

```
public class Collection{
        String color, name;
        int value;
        void printColor() {
            //code that accesses color attribute
        }
        void printCurrentDate() {
          //code that does not access any attribute
        }
}
```

Figure A2: Collection Java class

```
public class Person2{
        String firstName, lastName;
        Date date;
        void printName1() {
          /*code that prints first the firstName
            and then the lastName*/
        }
        void printName2() {
          /*code that prints first the lastName
            and then the firstName*/
        }
        void printDate() {
          //code that accesses date attribute
        }
}
```

Figure A3: Person2 Java class

```
public class Person3{
        String firstName, lastName;
        Date birth_date;
        void printFirstName() {
            //code that prints the firstName
        }
        void printName() {
            /*code that accesses firtName
              and lastName attributes*/
        }
        void printPersonInfo() {
            /*code that prints lastName
              and birth_date
        }
}
```

Figure A4: Person3 Java class

Metric	Person1	Collection	Person2	Person3
PCCC	0.12	0.03	0.1	0.11
CBMC	0	0	0	0.33
ICBMC	0	0	0	0.11
OL_2	0	0	0	0.2
OL_3	0	0	0	0.07
LCOM3	2	2	2	1
LCOM4	2	2	2	1
LCOM1	1	1	2	1
LCOM2	1	1	1	0
LCOM5	1	1.67	0.67	0.67
Coh	0.5	0.17	0.55	0.55
TCC	0	0	0.33	0.67
LCC	0	0	0.33	1
DC_D	0	0	0.33	0.67
DC_I	0	0	0.33	1
LSCC	0	0	0.22	0.22
CC	0	0	0.33	0.28
SCOM	0	0	0.22	0.39

Table A1: The cohesion values using the considered metrics for the four Java sample examples given in Figures A1, A2, A3, and A4

Appendix B: Univariate logistic regression results for connectivity metrics

Metric	System	Std. Coeff.	Odd ratio	Std. Error	95% Confidence Interval Coeff.	95% Confidence Interval odd ratio	p-value	Precision	Recall	ROC area
PCCC	Art of Illus.	-0.93	0.39	0.25	[-1.42,-0.44]	[0.24,0.64]	0.0002	69.8	63.1	65.2
	Gantt	-0.36	0.69	0.14	[-0.64,-0.09]	[0.53,0.91]	0.0093	39.4	62.8	58.7
	JebRef	-0.77	0.46	0.14	[-1.05,-0.50]	[0.35,0.61]	< 0.0001	67.3	74.8	69.3
	Openbravo	-0.70	0.50	0.14	[-0.97,-0.43]	[0.38,0.65]	< 0.0001	77.2	79.4	69.4
CBMC	Art of Illus.	-0.55	0.58	0.14	[-0.83,-0.27]	[0.44,0,76]	0.0001	60.1	57.7	59.9
	Gantt	-0.49	0.61	0.15	[-0.79,-0.19]	[0.45,0.83]	0.0014	39.4	62.8	58.4
	JebRef	-0.74	0.48	0.14	[-1.01,-0.47]	[0.36,0.62]	< 0.0001	68.8	75.2	63.5
	Openbravo	-0.72	0.49	0.14	[-0.99,-0.45]	[0.37,0.68]	< 0.0001	77.2	79.4	66.5
ICBMC	Art of Illus.	-0.49	0.61	0.16	[-0.81,-0.17]	[0.44,0.84]	0.0029	60.9	57.2	59.4
	Gantt	-0.36	0.69	0.14	[-0.66,-0.07]	[0.52,0.93]	0.0140	39.4	62.8	58.4
	JebRef	-0.70	0.49	0.13	[-0.96,-0.44]	[0.38,0.64]	< 0.0001	68.8	75.2	62.9
	Openbravo	-0.67	0.51	0.13	[-0.94,-0.41]	[0.39,0.66]	< 0.0001	77.2	79.4	66.4
OL_2	Art of Illus.	-0.48	0.62	0.14	[-0.75,-0.21]	[0.47,0.81]	0.0004	62.2	59.1	59.4
	Gantt	-0.49	0.61	0.15	[-0.79,-0.18]	[0.54,0.83]	0.0016	39.4	62.8	58.5
	JebRef	-0.73	0.48	0.14	[-0.99,-0.46]	[0.37,0.63]	< 0.0001	68.8	75.2	63.3
	Openbravo	-0.74	0.47	0.14	[-1.01,-0.48]	[0.36,0.62]	< 0.0001	77.2	79.4	66.9
OL_3	Art of Illus.	-0.46	0.63	0.15	[-0.77,-0.16]	[0.46,0.85]	0.0029	60.2	56.1	59.3
	Gantt	-0.41	0.66	0.15	[-0.71,-0.11]	[0.49,0.89]	0.0067	39.4	62.8	58.4
	JebRef	-0.71	0.49	0.13	[-0.97,-0.45]	[0.38,0.64]	< 0.0001	68.8	75.2	63.1
	Openbravo	-0.69	0.50	0.13	[-0.96,-0.43]	[0.38,0.65]	< 0.0001	77.2	79.4	67.0
LCOM3	Art of Illus.	0.44	1.55	0.18	[0.09,0.79]	[1.10,2.20]	0.0124	61.8	55.0	51.6
	Gantt	0.26	1.29	0.13	[-0.01,0.52]	[0.99,1.68]	0.0560	56.5	62.4	51.6
	JebRef	0.29	1.34	0.21	[-0.12,0.70]	[0.88,2.02]	0.1669	62.1	78.8	50.5
	Openbravo	-0.03	0.97	0.14	[-0.31,0.25]	[0.74,1.28]	0.8315	61.1	77.8	47.1
LCOM4	Art of Illus.	0.40	1.49	0.17	[0.06,0.74]	[1.06,2.09]	0.0198	61.0	54.5	51.1
	Gantt	0.26	1.29	0.13	[-0.01,0.52]	[0.99,1.68]	0.0541	55.0	62.0	51.6
	JebRef	0.23	1.26	0.20	[-0.17,0.63]	[0.84,1.88]	0.2576	62.1	78.8	49.5
	Openbravo	-0.04	0.96	0.14	[-0.31,0.23]	[0.73,1.26]	0.7738	61.1	77.8	49.0

Table B1: Univariate logistic regression results for connectivity metrics when special methods are included

Metric	System	Std. Coeff.	Odd ratio	Std. Error	95% Confidence Interval Coeff.	95% Confidence Interval odd ratio	p-value	Precision	Recall	ROC area
PCCC	Art of Illus.	-1.23	0.30	0.30	[-1.82,-0.64]	[0.16,0.53]	< 0.0001	70.3	63.7	66.8
	Gantt	-0.51	0.60	0.15	[-0.80,-0.22]	[0.45,0.80]	0.0005	39.4	62.8	60.1
	JebRef	-0.90	0.41	0.15	[-1.19,-0.60]	[0.30,0.55]	< 0.0001	62.1	78.8	70.6
	Openbravo	-0.59	0.56	0.14	[-0.86,-0.32]	[0.42,0.73]	< 0.0001	61.2	78.2	65.1
CBMC	Art of Illus.	-0.89	0.41	0.20	[-1.30,-0.49]	[0.27,0.61]	< 0.0001	70.2	62.6	61.8
	Gantt	-0.50	0.60	0.15	[-0.81,-0.20]	[0.45,0.82]	0.0012	39.4	62.8	57.8
	JebRef	-0.75	0.47	0.14	[-1.02,-0.48]	[0.36,0.62]	< 0.0001	67.6	75.2	62.8
	Openbravo	-0.73	0.48	0.14	[-1.00,-0.47]	[0.37,0.63]	< 0.0001	77.9	79.8	64.0
ICBMC	Art of Illus.	-0.82	0.44	0.29	[-1.39,-0.25]	[0.25,0.78]	0.0047	70.5	61.8	62.4
	Gantt	-0.41	0.66	0.15	[-0.71,-0.12]	[0.49,0.89]	0.0058	39.4	62.8	58.1
	JebRef	-0.73	0.48	0.13	[-1.00,-0.47]	[0.37,0.63]	< 0.0001	67.9	74.8	62.0
	Openbravo	-0.71	0.49	0.14	[-0.97,-0.44]	[0.38,0.64]	< 0.0001	77.9	79.8	64.1
OL_2	Art of Illus.	-0.88	0.41	0.21	[-1.28,-0.47]	[0.28,0.62]	< 0.0001	69.5	61.8	61.6
	Gantt	-0.50	0.61	0.16	[-0.81,-0.19]	[0.45,0.82]	0.0013	39.4	62.8	58.0
	JebRef	-0.74	0.48	0.14	[-1.01,-0.47]	[0.36,0.62]	< 0.0001	66.3	76.6	62.7
	Openbravo	-0.73	0.48	0.14	[-1.00,-0.46]	[0.37,0.63]	< 0.0001	77.9	79.8	64.2
OL_3	Art of Illus.	-0.82	0.43	0.25	[-1.31,-0.33]	[0.27,0.72]	0.0010	71.2	61.2	60.8
	Gantt	-0.44	0.64	0.15	[-0.74,-0.14]	[0.47,0.87]	0.0037	39.4	62.8	58.2
	JebRef	-0.73	0.48	0.13	[-1.00,-0.47]	[0.37,0.63]	< 0.0001	67.6	75.2	62.3
	Openbravo	-0.71	0.49	0.14	[-0.98,-0.45]	[0.37,0.64]	< 0.0001	77.9	79.8	64.3
LCOM3	Art of Illus.	0.33	1.39	0.14	[0.06,0.60]	[1.06,1.82]	0.0178	58.9	56.6	53.3
	Gantt	0.34	1.41	0.13	[0.09,0.59]	[1.10,1.81]	0.0070	59.7	63.1	57.4
	JebRef	0.41	1.51	0.23	[-0.03,0.86]	[0.97,2.35]	0.0708	62.1	78.8	52.5
	Openbravo	0.09	0.22	0.15	[-0.35,0.53]	[0.70,1.69]	0.7000	61.2	78.2	47.1
LCOM4	Art of Illus.	0.33	1.39	0.14	[0.06,0.60]	[1.06,1.84]	0.0185	58.8	56.4	53.1
	Gantt	0.37	1.44	0.13	[0.11,0.62]	[1.11,1.87]	0.0056	60.5	63.5	57.5
	JebRef	0.34	1.41	0.22	[-0.09,0.77]	[0.92,2.16]	0.1177	62.1	78.8	50.8
	Openbravo	0.12	0.26	0.20	[-0.40,0.63]	[0.67,1.88]	0.6557	61.2	78.2	46.6

Table B2: Univariate logistic regression results for connectivity metrics when constructor methods are excluded

Metric	System	Std. Coeff.	Odd ratio	Std. Error	95% Confidence Interval Coeff.	95% Confidence Interval odd ratio	p-value	Precision	Recall	ROC area
PCCC	Art of Illus.	-0.56	0.57	0.14	[-0.83,-0.29]	[0.43,0.75]	< 0.0001	66.2	59.6	65.6
	Gantt	-0.43	0.65	0.14	[-0.70,-0.16]	[0.50,0.85]	0.0016	39.4	62.8	61.2
	JebRef	0.76	0.47	0.14	[-1.04,-0.48]	[0.35,0.62]	< 0.0001	62.1	78.8	68.9
	Openbravo	-0.86	0.42	0.15	[-1.15,-0.57]	[0.32,0.56]	< 0.0001	77.7	79.0	74.5
CBMC	Art of Illus.	-0.37	0.69	0.12	[-0.60,-0.14]	[0.55,0.87]	0.0018	56.8	54.5	52.4
	Gantt	-0.43	0.65	0.14	[-0.70,-0.16]	[0.50,0.85]	0.0020	39.4	62.8	57.4
	JebRef	-0.70	0.49	0.14	[-0.97,-0.43]	[0.38,0.65]	< 0.0001	62.1	78.8	61.7
	Openbravo	-0.86	0.42	0.14	[-1.14,-0.57]	[0.32,0.56]	< 0.0001	77.4	78.6	70.7
ICBMC	Art of Illus.	-0.31	0.73	0.12	[-0.55,-0.08]	[0.58,0.92]	0.0090	57.8	54.7	52.1
	Gantt	-0.33	0.72	0.14	[-0.60,-0.06]	[0.55,0.94]	0.0162	39.4	62.8	58.1
	JebRef	-0.69	0.50	0.13	[-0.95,-0.43]	[0.39,0.65]	< 0.0001	67.9	75.5	61.6
	Openbravo	-0.78	0.46	0.14	[-1.05,-0.51]	[0.35,0.60]	< 0.0001	77.7	79.0	70.7
OL_2	Art of Illus.	-0.29	0.74	0.11	[-0.51,-0.07]	[0.60,0.93]	0.0086	54.3	52.8	52.2
	Gantt	-0.44	0.64	0.14	[-0.71,-0.17]	[0.49,0.84]	0.0016	39.4	62.8	57.5
	JebRef	-0.69	0.50	0.14	[-0.96,-0.42]	[0.38,0.66]	< 0.0001	62.1	78.8	61.5
	Openbravo	-0.89	0.41	0.15	[-1.17,-0.60]	[0.31,0.55]	< 0.0001	77.7	79.0	71.1
OL_3	Art of Illus.	-0.33	0.72	0.12	[-0.56,-0.10]	[0.57,0.91]	0.0054	56.5	53.7	52.6
	Gantt	-0.38	0.68	0.14	[-0.65,-0.11]	[0.52,0.90]	0.0061	39.4	62.8	57.6
	JebRef	-0.69	0.50	0.13	[-0.96,-0.43]	[0.38,0.65]	< 0.0001	61.9	77.7	61.6
	Openbravo	-0.82	0.44	0.14	[-1.10,-0.55]	[0.33,0.58]	< 0.0001	77.7	79.0	71.1
LCOM3	Art of Illus.	-0.05	0.95	0.10	[-0.26,0.15]	[0.77,1.16]	0.6056	50.2	50.4	47.8
	Gantt	0.30	1.35	0.15	[0.01,0.59]	[1.01,1.81]	0.0400	63.8	63.9	51.9
	JebRef	0.24	1.27	0.18	[0.60,1.27]	[0.89,1.83]	0.1882	62.1	78.8	50.2
	Openbravo	-0.17	0.84	0.14	[-0.45,0.10]	[0.64,1.11]	0.2232	61.1	77.8	48.6
LCOM4	Art of Illus.	-0.12	0.89	0.11	[-0.33,0.09]	[0.72,1.09]	0.2578	55.3	53.7	49.4
	Gantt	0.30	1.35	0.15	[0.01,0.59]	[1.01,1.81]	0.0400	63.8	63.9	51.9
	JebRef	0.18	1.19	0.17	[0.51,1.19]	[0.85,1.97]	0.3019	62.1	78.8	49.9
	Openbravo	-0.20	0.82	0.14	[-0.48,0.07]	[0.62,1.07]	0.1478	61.1	77.8	48.8

Table B3: Univariate logistic regression results for connectivity metrics when access methods are excluded

Metric	System	Std. Coeff.	Odd ratio	Std. Error	95% Confidence Interval Coeff.	95% Confidence Interval odd ratio	p-value	Precision	Recall	ROC area
PCCC	Art of Illus.	-0.76	0.47	0.13	[-1.02,-0.51]	[0.36,0.60]	< 0.0001	70.2	64.5	66.1
	Gantt	-0.57	0.56	0.13	[-0.83,-0.31]	[0.44,0.37]	< 0.0001	65.3	66.1	63.0
	JebRef	-0.94	0.34	0.17	[-1.28,-0.61]	[0.28,0.54]	< 0.0001	62.1	78.8	69.5
	Openbravo	-0.76	0.47	0.15	[-1.06,-0.46]	[0.35,0.63]	< 0.0001	61.2	78.2	67.8
CBMC	Art of Illus.	-0.68	0.50	0.16	[-1.00,-0.37]	[0.37,0.69]	< 0.0001	66.7	58.8	54.4
	Gantt	-0.45	0.64	0.14	[-0.72,-0.17]	[0.49,0.84]	0.0014	39.4	62.8	56.1
	JebRef	-0.76	0.46	0.14	[-1.04,-0.48]	[0.35,0.61]	< 0.0001	62.1	78.8	62.3
	Openbravo	-0.75	0.47	0.14	[-1.02,-0.47]	[0.36,0.63]	< 0.0001	69.6	76.3	64.4
ICBMC	Art of Illus.	-0.67	0.51	0.17	[-1.01,-0.33]	[0.36,0.72]	0.0001	69.3	59.1	54.4
	Gantt	-0.38	0.68	0.14	[-0.65,-0.11]	[0.52,0.89]	0.0055	39.4	62.8	56.4
	JebRef	-0.77	0.46	0.14	[-1.05,-0.50]	[0.35,0.60]	< 0.0001	62.1	78.8	62.4
	Openbravo	-0.73	0.48	0.14	[-1.00,-0.45]	[0.40,0.63]	< 0.0001	71.8	75.9	64.7
OL_2	Art of Illus.	-0.66	0.52	0.16	[-0.96,-0.35]	[0.38,0.71]	< 0.0001	65.3	58.0	54.3
	Gantt	-0.45	0.64	0.14	[-0.72,-0.17]	[0.49,0.84]	0.0015	39.4	62.8	56.1
	JebRef	-0.76	0.47	0.14	[-1.04,-0.48]	[0.35,0.62]	< 0.0001	62.1	78.8	62.4
	Openbravo	-0.73	0.48	0.14	[-1.01,-0.46]	[0.36,0.63]	< 0.0001	69.6	76.3	64.3
OL_3	Art of Illus.	-0.67	0.51	0.17	[-1.00,-0.34]	[0.38,0.71]	< 0.0001	67.2	57.7	54.6
	Gantt	-0.41	0.66	0.14	[-0.68,-0.13]	[0.51,0.87]	0.0034	39.4	62.8	56.3
	JebRef	-0.77	0.46	0.14	[-1.05,-0.50]	[0.35,0.61]	< 0.0001	62.1	78.8	62.4
	Openbravo	-0.73	0.48	0.14	[-1.00,-0.45]	[0.37,0.63]	< 0.0001	70.6	75.9	64.7
LCOM3	Art of Illus.	0.44	1.55	0.18	[0.10,0.79]	[1.10,2.20]	0.0124	61.8	55.0	51.6
	Gantt	0.26	1.29	0.13	[-0.01,0.52]	[0.99,1.68]	0.0560	56.5	62.4	51.6
	JebRef	0.31	1.37	0.21	[-0.10,0.73]	[0.90,2.07]	0.1427	62.1	78.8	51.1
	Openbravo	-0.03	0.97	0.14	[-0.31,0.25]	[0.74,1.28]	0.8315	61.1	77.8	47.1
LCOM4	Art of Illus.	0.40	1.49	0.17	[0.06,0.74]	[1.07,2.10]	0.0198	61.0	54.5	51.1
	Gantt	0.26	1.29	0.13	[-0.01,0.52]	[0.99,1.68]	0.0540	55.0	62.0	51.6
	JebRef	0.25	1.29	0.21	[-0.15,0.66]	[0.86,1.93]	0.2215	62.1	78.8	50.1
	Openbravo	-0.04	0.96	0.14	[-0.31,0.23]	[0.73,1.26]	0.7738	61.1	77.8	49.0

Table B4: Univariate logistic regression results for connectivity metrics when special methods are excluded

About the author

Jehad Al Dallal received his PhD degree in Computer Science from University of Alberta in Canada and was granted the best PhD research award. He is currently working at Kuwait University, department of Information Science as an Associate Professor. Dr. Al Dallal completed several research projects in the areas of software testing, software metrics, and communication protocols. In addition, he published more than 50 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 an associate editor for several referred journals.