Computer Science Journal of Moldova, vol.**25**, no.1(73), 2017

Class Cohesion Metrics for Software Engineering: A Critical Review [*]

Habib Izadkhah, Maryam Hooshyar

Abstract

Class cohesion or degree of the relations of class members is considered as one of the crucial quality criteria. A class with a high cohesion improves understandability, maintainability and reusability. The class cohesion metrics can be measured quantitatively and therefore can be used as a base for assessing the quality of design. The main objective of this paper is to identify important research directions in the area of class cohesion metrics that require further attention in order to develop more effective and efficient class cohesion metrics for software engineering. In this paper, we discuss the class cohesion assessing metrics (thirty-two metrics) that have received the most attention in the research community and compare them from different aspects. We also present desirable properties of cohesion metrics to validate class cohesion metrics.

Keywords: Cohesion, Class cohesion metrics, Software engineering.

1 Introduction

Producing high quality software systems has always been one of the main goals of software designers and developers. Many software features have been identified that influence the quality of software, for example complexity, coupling and cohesion. Cohesion is an important software quality attribute and high cohesion is one of characteristics of well-structured software design. In overall, module cohesion indicates

[*] Corresponding author email addresses: izadkhah@tabrizu.ac.ir

relatedness in the functionality of a software module [1]. It shows to which degree the elements within that module belong together or are related to each other. Modules with high cohesion are usually robust, reliable, reusable, and understandable while modules with low cohesion are allied with undesirable characteristics such as being difficult to understand, test, maintain, and reuse. Cohesion is usually expressed as high cohesion or low cohesion. There are different types of module cohesion; we can, therefore, create a nominal scale for cohesion measurement. Table 1 shows an ordinal scale for cohesion measurement [1].

Table 1. An ordinal scale for cohesion measurement

Rank	Cohesion type	Quality
6	Functional cohesion	Strongest (good)
5	Sequential cohesion	
4	Communication cohesion	
3	Procedural cohesion	
2	Temporal cohesion	
1	Logical cohesion	
0	Coincidental cohesion	Weakest (bad)

In Table 1, functional cohesion is judged to provide a strongest relationship because all of the elements contribute to a single task. Functional cohesion corresponds to single responsibility principle. Coincidental cohesion takes place when a module performs different and unrelated tasks. The serious problem with these methods is that it depends on subjective human assessment. It means that these metrics specify that the module is cohesive or not cohesive, and do not capture varying strengths of cohesion.

An object oriented program is made of many classes and each class consists of members called methods and attributes. In an object-oriented paradigm, a module can be a class, the data can be attributes, and the methods can be elements. We are mainly interested in the cohesion of object-oriented programs such as classes. Class cohesion can be defined as a measure of the degree to which members (i.e., methods and attributes) of a class belong together. Cohesion is usually measured on structural information extracted entirely from the source code (e.g., attribute references in methods and method calls).

In this paper, most class cohesion assessing metrics that have been presented in the literature are reviewed and categorized in both syntactic and semantic relationships. We also outline directions for further research in class cohesion assessing metrics, such as to define of better class cohesion assessing metrics or the improvement and evaluation of existing ones.

The structure of the paper is as follows. Section 2 presents three different applications of existing class cohesion metrics in order to motivate the need for assessing class cohesion. An overview of the state of the art for assessing class cohesion metrics is presented in Section 3. In section 4, the class cohesion metrics are evaluated from theoretical validation aspects. Section 5 introduces lack of discrimination anomaly problem, so that this problem is addressed on some class cohesion metrics. Section 6 presents some desirable properties of cohesion assessing metrics. Finally, Section 7 concludes the paper and open research challenges related to class cohesion are discussed in this section.

2 Class Cohesion Assessing Applications

Before we present the technical aspects of class cohesion metrics in Section 3, we describe three cases where class cohesion metrics can be used to solve important software engineering problems in the context of program understandability, software maintainability, and Reuse. It was argued that each module (class level) of well structured software should be focused on a single purpose. This means that it should have very few responsibilities. First, before addressing class cohesion metrics applications, we define a program's class design principle related with cohesion [2].

Principle: The Single Responsibility Principle (SRP) - A class should have one, and only one, reason to change.

The SRP indicates high cohesion for a class, in other words highly related, single responsibility, and strongly focused functionality for a class; where a responsibility [2] is defined as "a reason for change". For addressing class cohesion assessing applications, the following classes are considered: Class A performs three distinct operations (i.e., three responsibilities) while Class B performs only one operation (i.e., single responsibility); the cohesion of class B is higher than the class A.

```
Class A {
    1. Init disk() {...}
    2. Apply to college() {...}
    3. Circle Area Calculation () {...}
}
Class B {
    1. Method A part 1() {...}
    2. Method A part 2 () {...}
    3. Method A part 3 () {...}
}
```

2.1 Software Maintainability

In software engineering, maintainability is defined as the effort and the ease required to change in one module (e.g., class), to repair or to meet new requirements, maximize a product's useful life, maximize efficiency, reliability, and safety, cope with a changed environment, and etc. Since class B has only single responsibility, therefore, changing it is easier than changing class A; because changing a specific method in class A may affect other methods, due to the use of shared attributes. So, considering this side effect, it is a hard work to maintain a class. Maintainable class is a class that exhibits high cohesion. Higher cohesion and consequently higher maintainability means less time to make a change. Class cohesion assessing metrics evaluate the cohesion of a class for determining the level of maintainability of a class.

2.2 Program Understandability

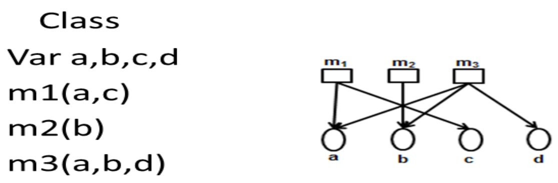
From Eighty-Twenty Rule, 20% of the time is spent creating, and 80% of the time is spent maintaining. Of the maintenance time, 20% is spent changing, while 80% of the time is spent just trying to understand the code. Therefore, understandability or program comprehension is one of the important characteristics of software quality, because it concerns the ways software engineers maintain the existing source code. In order to maintain the software, the programmers need to comprehend the source code. The understandability of the source code depends upon the cohesion, coupling and complexity of the program. A class with single responsibility is easier to understand compared to a class with multiple responsibilities. Class cohesion assessing metrics evaluate the cohesion of a class for determining the level of understandability of a class; a class with higher cohesion is more understandable than a class with low cohesion. For example, in our designed classes, since the class B only does “one thing,” its interface usually has a small number of methods that are fairly self explanatory. It should also have a small number of member variables, thus it is more understandable than class A.

2.3 Software Reusability

One of the most powerful characteristics of object-oriented programming is code reuse. You can write a class and then reuse it many times. High cohesion with loose coupling, information hiding makes code more easily reusable, and decrease the cost and time of development. If a class has multiple responsibilities, and only one of those is needed to assemble new requirements in other programs, then the other redundant responsibilities hinder reusability. Single responsibility implies the reuse of a class anywhere without modification. In our designed classes, class B is more reusable than class A.

3 Class Cohesion Assessing Metrics

Using ordinal scale for cohesion measurement, comparing two cohesive or two non-cohesive classes, or to know whether a code modification increased or reduced the degree of cohesiveness is hard. In this paper, we review the metrics for determining whether a module (class) is cohesive or not cohesive and also the degree of its cohesiveness. Also, these metrics would allow us to judge which module is better designed and more cohesive. These metrics compute class cohesion using manipulations of class elements. The key elements of a class C are its attributes $A_1, ..., A_a$, methods $M_1, ..., M_m$, and the list of p parameter (or, argument) types of the methods $P_1, ..., P_m$. Using class cohesion assessing metrics for computing class cohesion, we need to show a class with corresponding graph. Let a, b, c, d denote attributes and m_1, m_2, m_3 show methods. Figure 1 shows a sample class with its corresponding graph.



```
  Class
Var a,b,c,d
m1(a,c)
m2(b)
m3(a,b,d)
```

Figure 1. Class with corresponding graph

Class cohesion metrics can be broadly classified into two groups:
1. **Interface-based metrics** compute class cohesion from information in method signatures. Table 2 shows most interface-based metrics. Advantage of these metrics is that they can be calculated early in the design stage. Disadvantage of them can be concluded as from only design model, it is impossible to determine what exactly a method is doing (e.g., it may be using class attributes, or calling other methods on its class);
2. **Code-based metrics** also can be classified into two groups of conceptual metrics and structural metrics. Conceptual metrics use information retrieval methods for class cohesion measurement. Information

retrieval methods are based on the extraction the concepts of source code. In conceptual cohesion if methods of one class are conceptually related, then class is considered cohesive. Conceptual metrics of cohesion are introduced in the Tables 3 and 4. Many structural metrics have been proposed for cohesion measurement from research community. These metrics use the structural data that are extracted from the source code. The differences among structural metrics are based on the definition of the relationships between methods. Structural metrics compute class cohesion (method-method invocation) in terms of attribute accesses by methods (operations sharing attributes) and operations invocating other operations. The latter have the strongest cohesion compared to the former ones. We can further classify structural base cohesion metrics into four sub-types based on the methods of quantification of cohesion:

- Disjoint component-based metrics count the number of disjoint sets of methods or attributes in a given class. These metrics are addressed in Table 5.
- Pairwise connection-based metrics compute cohesion as a function of number of connected and disjoint method pairs. These metrics are addressed in Tables 6 and 7.
- Connection magnitude-based metrics count the accessing methods per attribute and indirectly find an attribute-sharing index in terms of the count (instead of computing direct attribute-sharing between methods). These metrics are addressed in Tables 8-11.
- Decomposition-based metrics compute cohesion in terms of recursive decompositions of a given class. The decompositions are generated by removal of pivotal elements that keep the class connected. These metrics are addressed in Table 12.

Table 2. Interface-based metrics

Class Cohesion Metrics	Definition/Formula
CAMC [3]	$CAMC = \frac{a}{kl}$, where, l is the number of distinct parameter types, k is the number of methods and a is the summation of the number of distinct parameter types of each method in the class.
NHD [4]	Let k and l have the same definitions as above for CAMC, and x_j be the number of methods that have a parameter of type j. Then, $NHD = 1 - \frac{2}{lk(k-1)} \sum_{j=1}^{l} x_j(k - x_j)$.
MMAC [5]	Let x_i be the number of methods that have a parameter or a return of type i. $MMAC(C) = \begin{cases} 0 & k = 0 \text{ or } l = 0 \\ 1 & k = 1 \\ \frac{\sum_{i=1}^{i=l} x_i(x_i - 1)}{lk(k-1)} & \text{otherwise} \end{cases}$

Table 3. Conceptual metrics

Class Cohesion Metrics	Definition/Formula
LORM [6]	LORM=Total number of relations in the class / Total number of possible relations to the class, where, Total number of relations in the class = number of pairs of methods in the class for which one method contains conceptual relations forming external links out of the set of concepts that belong to the method to or from the set of concepts belonging to another method in the class. N= total number of member functions (methods) in the class. Total number of possible relations = $\frac{N(N-1)}{2}$.
C3 [7]	Let $M(C) = \{m_1, ..., m_n\}$ be the set of methods of class, V_{m_i} and V_{m_j} be the vectors corresponding to the $m_i, m_j \in M(C)$. Then $C3 = \begin{cases} ACSM, & \text{if } ACSM > 0 \\ 0 & \text{otherwise} \end{cases},$ where $CSM(m_i, m_j) = \frac{V_{m_i}^t \cdot V_{m_j}}{\lvert V_{m_i}\rvert \times \lvert V_{m_j}\rvert}$ $ACSM = \frac{1}{N}\sum_{i=1}^{N} CSM(m_i, m_j)$ $N = C_n^2$
LCSM [7]	Let $M_i = \{m_j \lvert CSM(m_i, m_j) > ACSM(C), m_i \neq m_j\}$ and $P = (M_i, M_j) \lvert M_i \cap M_j = \emptyset$ and $Q = (M_i, M_j) \lvert M_i \cap M_j \neq \emptyset$ Then, LCSM(C)= $\begin{cases} \lvert P\rvert - \lvert Q\rvert, & \text{if } \lvert P\rvert > \lvert Q\rvert \\ 0 & \text{otherwise} \end{cases}$

Table 4. Conceptual metrics (continuation of Table 3)

Class Cohesion Metrics	Definition/Formula
MWE [8]	$MWE = max_{1<i<\|t\|}(O(t_i) \times D(t_i))$ where, $O(t_i) = \frac{\sum_{d=1}^{n} p_{t_i}^d}{n}$ $D(t_i) = \frac{\sum_{d=1}^{n} -q_{t_i}^d \times \log(q_{t_i}^d)}{\log n}$ $q_{t_i}^d = \frac{p_{t_i}^d}{\sum_{d=1}^{n} p_{t_i}^d}$ $p_{t_i}^d$=the probability of topic t_i in a method d $\|t\|$=number of topics n=number of methods

Table 5. Disjoint component-based metrics

Class Cohesion Metrics	Definition/Formula
LCOM1 [9]	Number of pairs of methods that do not share attributes.
TLCOM [10]	Number of method pairs that do not directly or transitively share a common attribute.
LC0M3 [11]	Number of connected components in the graph that represents each method as a node and the sharing of at least one attribute as an edge.
LCOM4 [12]	Similar to LCOM3, where graph G additionally has an edge between vertices representing methods m_i and m_j, if m_i invokes m_j or vice-versa.

Table 6. Pairwise connection-based metrics

Class Cohesion Metrics	Definition/Formula
LCOM2 [9]	Let P be the number of pairs of methods that do not share attributes and Q be the number of pairs of methods that share attributes. Then, $LCOM2(C) = \begin{cases} \|P\| - \|Q\|, & \text{if } \|P\| > \|Q\| \\ 0 & \text{otherwise} \end{cases}$
TCC [13]	Let NDC be the number of pairs of directly connected methods and NP be the maximum possible number of direct or indirect connection. Then, TCC is difined as: $TCC = \frac{NDC}{NP}$.
LCC [13]	Let NIC be the number of pairs of indirect connection in the class. Then, LCC=$\frac{NDC+NIC}{NP}$.
DC_D [14]	Fraction of directly connected pairs of methods, where 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.
DC_I [14]	Fraction of directly or transitively connected pairs of methods, where two methods are transitively connected if they satisfy the condition mentioned above for LCC or if the two methods directly or transitively invoke the same method.

Table 7. Pairwise connection-based metrics (continuation of Table 6)

Class Cohesion Metrics	Definition/Formula
CCM [15]	$CCM(C) = \frac{NC(C)}{NMP(C).NCC(C)}$, where NC(C) is the number of actual connections among the methods of class, NMP(C) is the number of the maximum possible connections among the methods of the class C, NCC(C) is the number of connected components of the connection graph G_c that represents each method as a node and two methods A and B are connected in the connection graph if A and B access one or more attributes in common or method A invokes method B or vice versa or methods A and B invoke one or more methods in common.

Table 8. Connection magnitude-based metrics

Class Cohesion Metrics	Definition/Formula
LCOM5 [16]	Let l be the number of attributes, k be the number of methods and a be the summing of the number of distance attributes that are accessed by each method in a class, then $LCOM5 = \dfrac{a - kl}{l - kl}$.
Coh [17]	Let a, k and l have the same definitions as above, then, $Coh = \dfrac{a}{kl}$.

Table 9. Connection magnitude-based metrics

Class Cohesion Metrics	Definition/Formula
OCC [18]	Consider a set of methods M=$\{m_1, ..., m_n\}$ and a set of attributes A within the class. Let $S = \{(m_i, m_j) \in M \times M \mid m_i\ invokes\ m_j\}$, $S^* = \{(m_i, m_j) \in M \times M \mid (m_i = m_j) \vee (\vee_{n \geq 1} m_i S^n m_j)\}$, $ac(m_i, a) \iff \exists m_j \in M\ s.t\ [(m_i S^* m_j) \wedge (m_j\, accesses\, a)]$ $R_w(m_i)$=set of methods which are reachable by $G_w(V, E)$ $G_w(V, E)$=weak-connection graph, where V=M(number of methods) and E is given by: $E = \{(m_i, m_j) \in M \times M \mid \exists a \in A\ s.t\ (ac(m_i, a) \wedge ac(m_j, a))\}$ Then, OCC(C)= $\begin{cases} max_{i=1,...,n}[\frac{\lvert R_w(m_i) \rvert}{n-1}] & n > 1 \\ 0 & \text{n=1} \end{cases}$
PCC [19]	$PCC(C) = \begin{cases} max_{i=1,...,n}[\frac{\lvert R_s(m_i) \rvert}{n-1}] & n > 1 \\ 0 & \text{n=1} \end{cases}$ where, $R_s(m_i)$=set of methods which are reachable by $G_s(V, E)$ $G_s(V, E)$=strong-connection graph, where V=M(number of methods) and E is given by: $E = \{(m_i, m_j) \in M \times M \mid \exists a \in A\ s.t\ (wr(m_i, a) \wedge re(m_j, a))\}$ $wr(m_i, a) \iff \exists m_j \in M\ s.t\ [(m_i S^* m_j) \wedge (m_j\, writes\ data\ on\ to\, a)]$, $re(m_i, a) \iff \exists m_j \in M\ s.t\ [(m_i S^* m_j) \wedge (m_j\, reads\ data\ from\ a)]$ and S, S^*, M, A, have the same definitions above.

Table 10. Connection magnitude-based metrics

Class Cohesion Metrics	Definition/Formula
CC [19]	Let I_i and I_j be the sets of attributes that are referenced by methods m_i and m_j, respectively. CC is 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 m_i and m_j is defined as: $Similarity(i,j) = \frac{\lvert I_i \cap I_j \rvert}{\lvert I_i \cup I_j \rvert}$.
SCOM [20]	Ratio of the summation of the similarities between all pairs of methods to the total number of pairs of methods. The similarity between methods m_i and m_j is defined as: $Similarity(m_i,m_j) = \frac{\lvert I_i \cap I_j \rvert}{min(\lvert I_i \rvert, \lvert I_j \rvert)} . \frac{\lvert I_i \cup I_j \rvert}{l}$.
PCCC [21]	PCCC= $\begin{cases} 0 & l=0 \text{ and} k>1 \\ 1 & l>0 \text{ and} k=0 \\ \frac{NSP(G_c)}{NSP(FG_c)} & \text{otherwise} \end{cases}$ where, l=number of attributes k=number of methods NSP=number of simple paths in graph G_c FG_c= corresponding fully connected graph.
LSCC [22]	LSCC(C)= $\begin{cases} 0 & l=0 \text{ and } k>1 \\ 1 & l>0 \, and \, k=0 \\ 1 & k=1 \\ \frac{\sum_{i=1}^{i=l} x_i(x_i-1)}{lk(k-1)} & \text{otherwise} \end{cases}$ 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 11. Connection magnitude-based metrics

Class Cohesion Metrics	Definition/Formula
CC [23]	Consider a set of methods $\{m_i\}$(i=1,...,m) accessing a set of attributes $\{V_j\}$(j=1,...,n). Let $\mu(V_j)$ be the number of methods that share the attribute V_j of a class. Then, $CC = \frac{\sum_1^n CV_j}{n}$, where $CV_j = \frac{\mu(V_j)}{m}$.
CO [24]	$CO = 2.\frac{\|E\| - (\|V\| - 1)}{(\|V\| - 1).(\|V\| - 2)}$ where, E and V are the edges and vertices of G from LCOM4.
CBAMU [25]	Let a be the number of attributes in the class, m be the number of methods in the class, $\mu(A_i)$ be the number of methods that access A_i and $\mu(M_j)$ be the number of methods that invoke method M_j, then, $CBAMU = \frac{1}{2}(AU(C) + MU(C))$, where, AU(C)= $\begin{cases} 0 & a = 0 \text{ or } m = 0 \\ \frac{1}{am}\sum_{i=1}^{a} \mu(A_i) & \text{otherwise} \end{cases}$ and MU(C)= $\begin{cases} 0 & m = 0 \text{ or } 1 \\ \frac{1}{m(m-1)}\sum_{j=1}^{m} \mu(M_j) & \text{otherwise} \end{cases}$
RCI [26]	Let CI(C) be the set of all data-data interactions and data-method interactions and Max(C) to be the set of all possible data-data interactions and data-method interactions. Then, $RCI(C) = \frac{\|CI(C)\|}{\|Max(C)\|}$.

Table 12. Decomposition-based metrics

Class Cohesion Metrics	Definition/Formula
CBMC [27]	Let $F_c(G) = \dfrac{M(G)}{N(G)}$ and $F_s(G) = \dfrac{1}{n}\sum_{i=1}^{n} CBMC(G^i)$ where, M(G)=the number of glue methods in graph G, N(G)=the number of non-special methods in graph G G^i = one of children of the structure tree.Then, $CBMC(G) = F_c(G) \times F_s(G)$.
ICBMC [28]	$ICBMC(G) = F_c(G) \times F_s(G)$ where, $F_c(G) = \dfrac{\lvert Q\rvert}{\lvert N_m(G)\rvert \times \lvert N_v(G)\rvert}$ and $F_s(G) = \dfrac{1}{2}\sum_{i=1}^{n} ICBMC(G^i)$ Q=number of edges in the cut set of G $N_m(G)$= number of non-special methods in graph G $N_v(G)$= number of attributes in graph G.
OL_n [29]	The average strength of the attributes, wherein the strength of an attribute is the average strength of the methods that reference that attribute. n is the number of iterations that are used to compute OL.

Now, we evaluate these metrics and compare them through three given examples. Let a_i and m_i denote attributes and methods, respectively, the first example (Fig. 2) shows four sample classes including linking methods of a class with their attributes. The cohesion of these classes increase from C1 to C4, indeed class C1 has the lowest cohesion and class C2 has the highest cohesion. Table 13 shows the cohesion value for some these metrics.

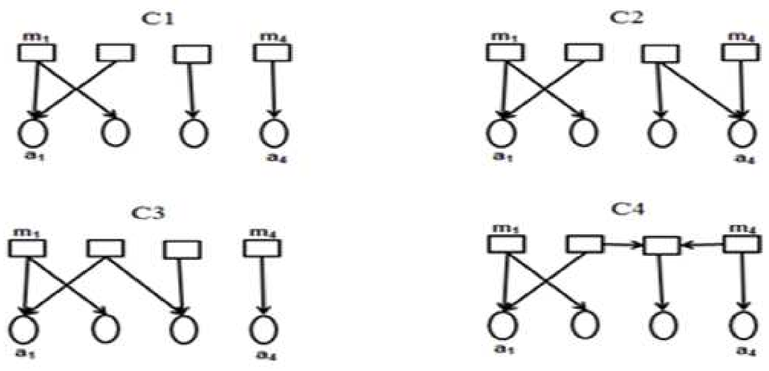


Figure 2. Example cases for capturing degree of cohesion

In Table 13, in the last column, sign (-) means that its corresponding metrics does not give the correct result about cohesion and sign (+) means that the result is acceptable. As it can be seen in Table 13, only LCOM4, TCC, DCD, DCI and CBAMU metrics can find the class cohesion correctly.

The second example (Fig. 3) shows three other classes (C1-C3) with different degrees of cohesion. In this figure, class C3 has the highest cohesion and class C1 has the lowest cohesion. In Table 14, the cohesion values for some of these metrics are presented. As it can be seen in Table 14, only LCOM1, LCOM2, LCOM5, Coh, TCC, CBMC, CCM, CC, SCOM, LSCC, CC and CBAMU metrics can find the class cohesion correctly.

Table 13. Cohesion values for Fig. 2

Metric	$C1$	$C2$	$C3$	$C4$	
$LCOM1$	5	4	4	5	-
LCOM2	4	2	2	4	-
LCOM3	3	2	2	3	-
LCOM4	3	2	2	1	+
LCOM5	0.91	0.83	0.83	0.91	-
Coh	0.31	0.37	0.37	0.31	-
TCC	0.16	0.33	0.33	0.5	+
LCC	0.16	0.33	0.5	0.5	-
DC_D	0.16	0.33	0.33	0.66	+
DC_I	0.16	0.33	0.5	0.66	+
CC	0.5	1	1	0.5	-
SCOM	0.5	1	1	0.5	-
LSCC	0.04	0.08	0.08	0.04	-
CC	0.31	0.37	0.37	0.31	-
CBAMU	0.15	0.18	0.18	0.23	+

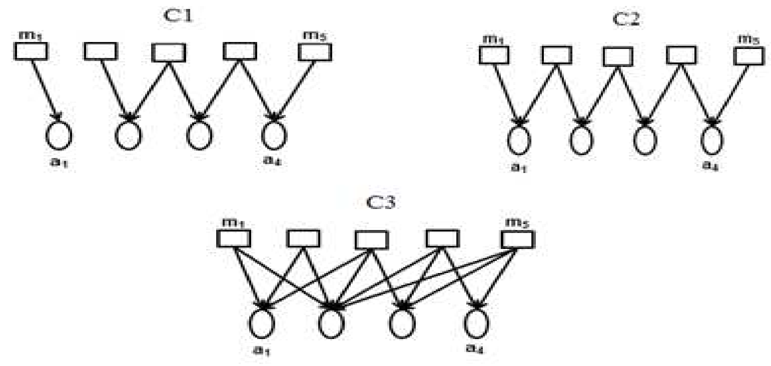


Figure 3. Example cases for capturing degree of cohesion

Table 14. Cohesion values for Fig. 3

Metric	$C1$	$C2$	$C3$	
LCOM1	7	6	0	+
LCOM2	3	2	0	+
LCOM3	2	1	1	-
LCOM4	2	1	1	-
LCOM5	0.81	0 .75	0.43	+
CCM	1	0.4	1	+
OCC	0.75	1	1	-
DC_D	0.3	0.4	0.4	-
DC_I	0.6	1	1	-
Co		0	1	-
CC	1.33	1.66	5.33	+
SCOM	1.37	1.75	6.08	+
Coh	0.35	0.4	0.65	+
LSCC	0.75	0.1	0.42	+
TCC	0.3	0.4	1	+
CC	0.35	0.4	0.65	+
LCC	0.6	1	1	-
CBAMU	0.17	0.2	0.32	+
CBMC	0	0.13	0.6	+

For the third example, consider classes in Fig. 4. In this figure, class C1 has the highest cohesion and class C4 has the lowest cohesion. The cohesion values for some of the metrics are shown in Table 15. As it can be seen in Table 15, only SCOM metrics can find the class cohesion correctly. Note that the SCOM metrics cannot find right cohesion for classes in Fig. 2.

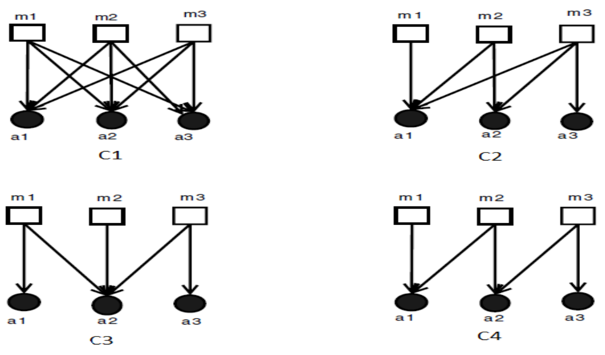


Figure 4. Examples for capturing degree of cohesion

Table 15. Cohesion values for Fig. 4

Metric	$C1$	$C2$	$C3$	$C4$	
TLCOM	0	0	0	1	-
LCOM2	0	0	0	0	-
TCC	1	1	1	0.67	-
SCOM	1	0.89	0.61	0.39	+
CBAMU	0.5	033.	0.27	0.27	-
CBMC	1	0.67	0.67	0.33	-
ICBMC	0.33	0.139	0.069	0.069	-
CCM	1	1	1	0.66	-
PCCC	1	0.2	0.11	0.11	-
RCI	1	0.5	0.41	0.41	-

From our designed different eleven classes with different cohesion, our experiment (Tables 13-15) shows that the described cohesion assessing metrics for all our designed classes in these examples do not show the right cohesion. Indeed, these metrics cannot calculate class cohesion on some classes correctly.

4 Theoretical Validations of Cohesion Metrics

For theoretical validation of the class cohesion assessing metrics a framework proposed in [17]. At this framework, four properties are presented for validation of metrics that measure cohesion values. If cohesion metrics is a true metrics, it should satisfy these properties. These four properties are:

Property 1: non-negativity and normalization: non-negative indicates that the measured class cohesion fits to a specific interval [0, Max], and using normalization the designer can compare the cohesion of different classes.

Property 2: null value and maximum value: the cohesion of a class is equal to 0 if the class has no cohesive interactions; the cohesion is equal to Max if all possible interactions within the class are present.

Property 3: monotonicity: adding cohesive interactions to the module cannot decrease its cohesion.

Property 4: cohesive modules: merging two unrelated modules into one module does not increase the individual modules cohesion. Therefore, given two classes, C1 and C2, the cohesion of the merged class C must satisfy the following condition:

$cohesion(C) \leq max\{cohesion(C1), cohesion(C2)\}$

These four properties (indicated by P1-P4) for the presented metrics (Tables 2-12) are addressed in Table 16. The sign (-) means that its corresponding metrics does not satisfy the property and sign (+) means that it satisfies the property.

Table 16. Theoretical validation results of cohesion metrics

Metric	P_1	P_2	P_3	P_4	Metric	P_1	P_2	P_3	P_4
$LCOM1$	-	+	+	+	PCC	+	+	+	+
$TLCOM$	-	+	+	+	DC_I	+	+	+	+
$LCOM2$	-	+	+	+	CC	+	+	-	+
$LCOM3$	-	-	+	+	MMAC	+	+	+	+
$LCOM4$	-	-	+	+	SCOM	+	+	-	+
CO	-	-	+	+	LSCC	+	+	+	+
$LCOM5$	-	+	+	-	CC	+	+	+	+
Coh	+	+	+	+	CAMC	-		+	+
TCC	+	+	+	+	NHD	-		-	-
LCC	+	+	+	+	CBAMU	+	+	+	+
DC_D	+	+	+	+	ICBMC	+	+	+	+
CBMC	+	+	-	+	PCCC	+	+	+	+
CCM	+	+	+	+	RCI	+	+	+	+
OCC	+	+	+	+	OL_n	+	+	+	+

5 Lack of Discrimination Anomaly

Some class cohesion metrics use normalization operation to allow for easy comparison of the cohesion of different classes, so it is possible that they get the same cohesion values for different classes that have the same number of methods and attributes but different interactions [30]. In these cases, two classes incorrectly are considered same in terms of cohesion. This leads to lack of discrimination anomaly (LDA) problem [30]. A good class cohesion metrics should avoid the LDA problem. If there is the LDA problem in a metrics, the probability that the metrics will distinguish between different classes cohesion is low; thus, the metrics is imprecise. Tables 17 and 18 show some class cohesion metrics and their LDA problems [30].

Table 17. Some class cohesion metrics and their LDA problems

Metrics	When LDA problem occur?
LCOM1	1. when the number of method pairs that share common attributes is the same in both classes, regardless of the number of shared attributes. 2. when the number of method pairs that share common attributes is the same in both classes, regardless of which attributes are shared.
LCOM2	1. same two LDA cases indicated for LCOM1 2. the two classes will have the same LCOM2 value, if $P \leq Q$. In this case, the value of LCOM2 is zero, regardless of the magnitude of the difference between P and Q in each of the two classes.
LCOM3 and LCOM4	1. same first LDA indicated for LCOM1 2. if the number of disjoint components is the same in both classes regardless of the interactions of each of the disjoint components.
LCOM5 and Coh	- two classes have the same number of attributes referenced by methods, regardless of the distribution of these references.
CAMC	- if the two classes have the same number of distinct types of parameters in their methods, regardless of the distribution of these cohesive interactions.
TCC and DC_D	- same two LDA cases indicated for LCOM1
LCC and DC_I	1. same two LDA cases indicated for LCOM1 2. the two classes have the same number of connected methods, regardless whether the methods are connected directly or transitively.

Table 18. Some class cohesion metrics and their LDA problems (continuation of Table 17)

Metrics	When LDA problem occur?
CC	- if the ratio of shared attributes between each pair of methods to the distinct number of attributes referenced by each of the corresponding pair of methods is the same, regardless of the number of shared attributes.
LSCC	1. when each attribute in Class C1 is referenced by the same number of methods that reference the corresponding attributes in Class C2, regardless of the distribution of these methods and their connectivity with other methods. 2. if the result of $\sum_{i=1}^{n} x_i(x_i - 1)$ is the same for both classes, where n is the number of attributes and x_i is the number of methods that reference attribute i, regardless of the distributions or the number of references.
NHD	1. same first LDA indicated for LSCC 2. if the result of $\sum_{i=1}^{n} x_i(n - x_i)$ is the same for both classes, where n is the number of distinct parameter types and x_i is the number of methods that reference parameter type i, regardless of the distributions or the number of references.

6 Desirable Properties of Cohesion Metrics

The addressed class cohesion metrics can be evaluated both empirically and theoretically. In the empirical validation it is determined whether the predicted value is consistent with the measured value. In [21], [26], [31]–[38] several methods are proposed to empirically validate class cohesion metrics.

In the theoretical validation it is determined whether the metrics holds the required properties. In addition to the four main properties proposed in [17] (see section 4), in literature, several properties are introduced to validate software metrics theoretically, as summarized below.

I. **Monotonicity:** adding interactions to the class must not decrease its cohesion. It means that after adding an interaction to the class, the modified class cohesion value will be the same as or higher than the cohesion value of the original class.

II. **Representation condition of measurement theory:** The metrics obtains a value that yields the same order as intuition. It means there should not be inconsistencies between empirical relations and numerical relations such that the empirical relations preserve and are preserved by the numerical relations.

III. **Lack of discrimination anomaly:** as described in section 5.

IV. **Sensitivity:** adding or removing interactions to a class should change the cohesion of the class. Sensitivity is expected to increase as:
1) the number of distinct cohesion values increases and
2) the number of classes with repeated cohesion values decreases

V. **Normalization:** allows for easy comparison of the cohesion of different classes

VI. **Prescription:** The metrics indicates how to enhance the measured element.

VII. **No equivalence of interaction:** Given three classes x, y, and z, if x and y display the same cohesion value, the classes resulting from merging z with each of x and y can show different cohesion values.

The class cohesion metrics that does not have LDA problem and satisfies the other properties is expected to be a well-defined metrics. We suggest the newly introduced metrics that, first, is evaluated from LDA problem point of view. If the metrics is found to have LDA problem, the researcher may decide to revise the metrics formula to solve the LDA problem. Software developers are advised to apply cohesion metrics that does not have LDA problem because such metrics is expected to be more reliable in indicating cohesion.

7 Conclusion and open research problem

This paper presented the state of the art in the development and evaluation of class cohesion metrics. Most existing structural class cohesion metrics is addressed and evaluated from different aspects such as on our eleven designed classes, theoretical validations, and LDA problem. Our experimental results showed that whereas important progresses have already taken place, but there is no comprehensive and reliable class cohesion metrics. Therefore, there are still many possibilities for further research that will benefit software engineers everywhere. Following cases can be considered as a research problem:
1. There are no standard benchmarks for evaluating the class cohesion metrics. Researcher needs comprehensive benchmarks for evaluating the new designed metrics and comparing it with other metrics.
2. We believe the four properties described in section 4 should be revised. For example, due to LDA problem, using the normalization and monotonicity properties, a designer necessarily cannot distinguish the cohesion level of different classes.
3. Information theory based formulas are not addressed in this area. We think that class cohesion can be assessed in terms of information loss. It seems a class with low information loss to have high cohesion. However, this is a research question.
4. The combination of structural and semantic information from a class for cohesion assessing can be considered as a research problem.

References

[1] R. S. Pressman, *Software engineering: a practitioner's approach.* Palgrave Macmillan, 2005.

[2] R. C. Martin, *Agile software development: principles, patterns, and practices.* Prentice Hall PTR, 2003.

[3] J. Bansiya, L. Etzkorn, C. Davis, and W. Li, "A class cohesion metric for object-oriented designs," *Journal of Object-Oriented Programming*, vol. 11, no. 8, pp. 47–52, 1999.

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

[5] J. Al Dallal and L. C. Briand, "An object-oriented high-level design-based class cohesion metric," *Information and software technology*, vol. 52, no. 12, pp. 1346–1361, 2010.

[6] L. Etzkorn and H. Delugach, "Towards a semantic metrics suite for object-oriented design," in *Technology of Object-Oriented Languages and Systems, 2000. TOOLS 34. Proceedings. 34th International Conference on.* IEEE, 2000, pp. 71–80.

[7] A. Marcus and D. Poshyvanyk, "The conceptual cohesion of classes," in *21st IEEE International Conference on Software Maintenance (ICSM'05).* IEEE, 2005, pp. 133–142.

[8] Y. Liu, D. Poshyvanyk, R. Ferenc, T. Gyimóthy, and N. Chrisochoides, "Modeling class cohesion as mixtures of latent topics," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on.* IEEE, 2009, pp. 233–242.

[9] S. R. Chidamber and C. F. Kemerer, *Towards a metrics suite for object oriented design.* ACM, 1991, vol. 26, no. 11.

[10] J. Al Dallal, "Transitive-based object-oriented lack-of-cohesion metric," *Procedia computer science*, vol. 3, pp. 1581–1587, 2011.

[11] E. Wigner, "On a modification of the rayleigh-schrödinger perturbation theory," in *Part I: Physical Chemistry. Part II: Solid State Physics.* Springer, 1997, pp. 131–136.

[12] M. Hitz and B. Montazeri, *Measuring coupling and cohesion in object-oriented systems.* Citeseer, 1995.

[13] J. M. Bieman and B.-K. Kang, "Cohesion and reuse in an object-oriented system," in *ACM SIGSOFT Software Engineering Notes*, vol. 20, no. SI. ACM, 1995, pp. 259–262.

[14] L. Badri and M. Badri, "A proposal of a new class cohesion criterion: an empirical study," *Journal of Object Technology*, vol. 3, no. 4, pp. 145–159, 2004.

[15] A. Jarallah, M. Wasiq, and M. Ahmed, "Principle and metrics for cohesion-based object-oriented component assessment," 2001, confidential Draft Copy.

[16] B. Henderson-Sellers, L. L. Constantine, and I. M. Graham, "Coupling and cohesion (towards a valid metrics suite for object-oriented analysis and design)," *Object Oriented Systems*, vol. 3, no. 3, pp. 143–158, 1996.

[17] L. C. Briand, J. W. Daly, and J. Wüst, "A unified framework for cohesion measurement in object-oriented systems," *Empirical Software Engineering*, vol. 3, no. 1, pp. 65–117, 1998.

[18] H. Aman, K. Yamasaki, H. Yamada, and M.-T. Noda, "A proposal of class cohesion metrics using sizes of cohesive parts," in *Proc. of Fifth Joint Conference on Knowledge-based Software Engineering*, 2002, pp. 102–107.

[19] C. Bonja and E. Kidanmariam, "Metrics for class cohesion and similarity between methods," in *Proceedings of the 44th annual Southeast regional conference.* ACM, 2006, pp. 91–95.

[20] L. Fernández and R. Peña, “A sensitive metric of class cohesion,” vol. 13, pp. 82–91, 2006.

[21] J. Al Dallal, “Fault prediction and the discriminative powers of connectivity-based object-oriented class cohesion metrics,” *Information and Software Technology*, vol. 54, no. 4, pp. 396–416, 2012.

[22] J. Al Dallal and L. C. Briand, “A precise method-method interaction-based cohesion metric for object-oriented classes,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 21, no. 2, p. 8, 2012.

[23] S. Mal and K. Rajnish, “New class cohesion metric: An empirical view,” *International Journal of Multimedia and Ubiquitous Engineering*, vol. 9, no. 6, pp. 367–376, 2014.

[24] M. Hitz and B. Montazeri, “Chidamber and kemerer’s metrics suite: a measurement theory perspective,” *IEEE Transactions on Software Engineering*, vol. 22, no. 4, pp. 267–271, 1996.

[25] A. Adam, “Implementation and validation of object-oriented design-level cohesion metrics,” Ph.D. dissertation, King Fahd University of Petroleum and Minerals, 2005.

[26] L. Briand, S. Morasca, and V. R. Basili, “Defining and validating high-level design metrics,” University of Maryland at College Park, Tech. Rep.

[27] H. S. Chae, Y. R. Kwon, and D.-H. Bae, “A cohesion measure for object-oriented classes,” *Software-Practice and Experience*, vol. 30, no. 12, pp. 1405–1432, 2000.

[28] Y. Zhou, B. Xu, J. Zhao, and H. Yang, “Icbmc: an improved cohesion measure for classes,” in *Software Maintenance, 2002. Proceedings. International Conference on.* IEEE, 2002, pp. 44–53.

[29] X. Yang, “Research on class cohesion measures,” Ph.D. dissertation, MS Thesis, Department of Computer Science and Engineering, Southeast University, 2002.

[30] J. Al Dallal, “Measuring the discriminative power of object-oriented class cohesion metrics,” *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 788–804, 2011.

[31] I. Hermadi, K. El-Badawi, and J. Al-Ghamdi, “Theoretical validation of cohesion metrics in object oriented systems,” in *International Arab Conference on Information Technology*, 2002, pp. 16–19.

[32] L. C. Briand, J. Wüst, J. W. Daly, and D. V. Porter, “Exploring the relationships between design measures and software quality in object-oriented systems,” *Journal of systems and software*, vol. 51, no. 3, pp. 245–273, 2000.

[33] L. C. Briand, J. Wüst, and H. Lounis, “Replicated case studies for investigating quality factors in object-oriented designs,” *Empirical software engineering*, vol. 6, no. 1, pp. 11–58, 2001.

[34] L. C. Briand and J. Wüst, “Empirical studies of quality models in object-oriented systems,” *Advances in computers*, vol. 56, pp. 97–166, 2002.

[35] M. Alshayeb and W. Li, “An empirical validation of object-oriented metrics in two different iterative software processes,” *IEEE Transactions on software engineering*, vol. 29, no. 11, pp. 1043–1049, 2003.

[36] 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*, vol. 31, no. 10, pp. 897–910, 2005.

[37] 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*, vol. 6, no. 10, pp. 127–141, 2007.

[38] 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*, vol. 34, no. 2, pp. 287–300, 2008.

Habib Izadkhah, Maryam Hooshyar,

Received August 25, 2016

Habib Izadkhah
Department of Computer Science,
Faculty of Mathematical Sciences,
University of Tabriz, Tabriz, Iran
E–mail: `izadkhah@tabrizu.ac.ir`

Maryam Hooshyar
Department of Computer Science,
Faculty of Mathematical Sciences,
University of Tabriz, Tabriz, Iran
E–mail: `m.hoshyar93@ms.tabrizu.ac.ir`