DMC: a more precise cohesion measure for classes

Jianmin Wang\textsuperscript{a,*}, Yuming Zhou\textsuperscript{a}, Lijie Wen\textsuperscript{a}, Yujian Chen\textsuperscript{a}, Hongmin Lu\textsuperscript{b}, Baowen Xu\textsuperscript{b}

\textsuperscript{a}School of Software, Tsinghua University, Room 819, Main Building, Beijing 100084, China
\textsuperscript{b}Department of Computer Science and Engineering, Southeast University, Nanjing 210096, China

Received 17 August 2003; accepted 15 July 2004
Available online 12 September 2004

Abstract

In object-oriented systems, a single class consists of attributes and methods and its cohesion denotes the degree of relatedness among these elements. To quantify the cohesiveness of a class, a large number of measures that only depict method–attribute reference relationships have been proposed in last decade. However, the flow-dependence relationships among attributes, the direction of method–attribute references, and the potential dependence relationships among the elements in the class are ignored. To address this problem, this paper first depicts four types of explicit dependence relationships and uses a class element dependence graph to represent all dependencies among the elements in a class. Then, a dependence matrix that reflects the degree of direct dependence and indirect dependence among the elements in a class is computed. Finally, a more precise cohesion measure for classes is proposed.

© 2004 Elsevier B.V. All rights reserved.

Keywords: Object-oriented system; Class; Cohesion; Measure; Class element dependence graph

1. Introduction

In object-oriented systems, classes group data and related operations within a specific domain concept, and support object-oriented properties such as data abstraction, encapsulation and inheritance. Class cohesion is an internal software attribute representing the degree to which the elements (attributes and methods) are bound together within a single class. A large number of measures have been proposed to quantify this concept \cite{1–13}. Most class cohesion measures are based on a graph-based cohesion model that represents the method–attribute reference relationships and/or methods similarity relationships and they calculate the class cohesion based on the number of edges in the graph. Neither the characteristics of methods nor the interaction patterns among the class elements are considered. For example, some methods such as constructors and destructors inherently interact with only part of the attributes of classes. Although these methods do not contribute to the cohesiveness of classes, most cohesion measures do not exclude them. Furthermore, the existing cohesion measures concentrate on the interaction number rather than the interaction pattern among the elements in a class. Thus, they fail to properly reflect the actual cohesiveness of a class.

Actually, the development of a well-defined cohesion measure for classes consists of two steps: first, a well-defined cohesion model that objectively characterizes the relationships among the elements in a class should be constructed; second, a class cohesion measure with good properties should be developed. However, all previous class cohesion measures are questionable in these two aspects. Virtually, no attention has been paid to the cohesion model that objectively depicts the relationships among the elements in a class. Specially, the flow dependence relationships among attributes, the direction of method–attribute references, and the potential dependence relationships among class elements are ignored. Previous measures only depicted so called similarity relationships between methods, the reference relationships between methods and attributes, and/or call relationships between methods. In addition, they do not have some of the goodness properties suggested by Briand et al. \cite{14}. So, the degree of interdependence among the elements in a class can not be well characterized.
To address these problems, this paper depicts four types of relationship among the elements in a class: the call dependence between methods, the read dependence of methods on attributes, the write dependence of attributes on methods, and the (data or control) flow dependence between attributes. Then, a class element dependence graph is used to represent different relationships among the elements in a class, and a weight matrix in which different types of relationships have different weights is used to quantify the degree of direct dependence among the class elements. Finally, a dependence matrix that characterizes the degree of direct dependence and indirect dependence among the class elements is computed and a novel cohesion measure with good properties is developed.

The remainder of this paper is organized as follows. Section 2 discusses how to construct a well-defined cohesion model for a class by a class element dependence graph. Section 3 proposes a dependence matrix based class cohesion measure DMC and shows that it has good properties theoretically. Section 4 demonstrates our measure by a case study and Section 5 compares DMC with related work. Finally, Section 6 gives the conclusion and states the future work.

2. Class element dependence graph

We believe that a well-defined cohesion model that objectively depicts the relationships among the elements in a class is the precondition of a well-defined class cohesion measure. If a cohesion measure can not represent the relationships among the elements in a class properly, then it must be questionable. In previous class cohesion measures, both the (data and control) flow dependence relationships among attributes and the distinction of access types between methods and attributes are ignored. In this section, we will propose a novel cohesion model, class element dependence graph, for representing the dependence relationships among the elements in a class. This cohesion model is based on the analysis of program dependence information, which is used to describe the read dependence of methods on attributes, the write dependence of attributes on methods, the control and data flow dependence among attributes, and the call dependence among methods.

2.1. Types of nodes

Attributes and methods are the basic elements in a class. However, not all methods in a class contribute to its cohesion. There exist some methods, such as access methods, delegation methods, constructor and destructor, inherently accessing only some of the attributes in the class [10,11]. An access method is a method that only reads or writes an attribute. As a result, it typically references only one attribute. A delegation method is a method that only delegates a message to another object, especially to an attribute in the class, thus, generally has one interaction with the attribute. A constructor is a method that initializes essential attributes and a destructor is a method that deinitializes essential attributes. They may not inherently access all of the attributes.

It has been recognized that the nature of these methods determines that they have no influence on the cohesion of a class [10,11,15,16,17]. Therefore, they should be excluded in the cohesion model of a class. To characterize this idea better, special methods and normal methods are defined as follows.

Definition 1. For any class \( c \) in an object-oriented system, let

1. \( M(c) = \{ m \mid m \text{ is a method declared in } c \} \)
2. \( A(c) = \{ a \mid a \text{ is an attribute declared in } c \} \)

Definition 2. For any class \( c \) in an object-oriented system, if a method in \( c \) is a constructor, a destructor, an access method or a delegation method, it is called a special method; otherwise it is called a normal method. The set of special methods in \( c \) is denoted as \( SM(c) \) and the set of normal methods in \( c \) is denoted as \( NM(c) \).

It is clear that

\[
M(c) = SM(c) \cup NM(c)
\]

Most of the existing cohesion measures do not distinguish special methods from normal methods, which lead to the incorrect evaluation of the cohesiveness in some cases. For example, Chidamber et al. did not exclude special methods [1], Briand et al. excluded only access methods and constructors [7], and Bieman et al. excluded only constructors and destructors [3].

We believe that all special methods should be excluded from the cohesion model of a class. However, some normal methods may call access methods or delegation methods. Therefore, if a normal method calls an access method or a delegation method, then the calling statement should be replaced by its corresponding body. In other words, the cohesion model that reflects the relationships among the elements in a class should consist of only two types of nodes: normal methods and attributes. In what follows, the word ‘method’, if not specified, denotes ‘normal method’.

2.2. Types of relationships

Now that the first problem has been solved, it is time to answer the second problem, i.e. what are the relationships among the normal methods and attributes in a class that should be considered? A method body can contain statements that read attributes, write attributes, and call other methods. Traditional cohesion measures only account for the (direct or indirect) access relationships between methods and attributes, or the call relationships between
methods. Both the direction of access relationships and the (control or data) flow dependencies among attributes are not considered, which results in the coarse evaluation of the cohesiveness of a class.

Actually, as Aman et al. stated [18], if a method m reads an attribute a that has an unexpected value, then it is quite possible that the behavior of m is not correct. However, if m has a logical error, the value of a will not be influenced. This fact states that the behavior of m is dependent on the value of a. Likewise, if an attribute a is written in a method m that has a logical error, we cannot expect the value of a is right. However, even if the attribute a has an unexpected value, it will be renewed by m. So, the value of a is dependent on the behavior of m. If one method m1 has been called by another method m2, it is clear that m1 is behavior dependent on m2. Certainly, m2 may also behavior dependent on m1 through the parameters it is called with.

Although the access dependence between methods and attributes has been considered in previous cohesion measures, the directions of dependence are not distinguished. The call relationship between methods is often used to extract the indirect access relationship between methods and attributes, which is often not explicitly represented on the cohesion model of a class in most measures. For example, the access relationship between methods and attributes is used to compute the so-called similarity relationships between methods in LCOM.

Besides, if many attributes are read or written by a method, the (control or data) flow dependence among them might exist. If an attribute a is defined in a statement s, and another attribute b is used in s or if whether s is executed is determined by the execution result of another statement s’ in which an attributed b is used, the value of attribute a is influenced by the value of attribute b, i.e. attribute a is (data or control) flow dependent on the value of attribute b. Note that what an attribute is defined in a statement denotes that a new value is assigned to the attribute. And the same is true in the following sections. To our knowledge, all previous cohesion measures, except our measure proposed in Ref. [13], pay no attention to the flow dependencies among attributes.

If the read dependence of methods on attributes, the write dependence of attributes on methods, the call dependence between methods, and the flow dependence between attributes are all considered, the relatedness among the elements in a class can be precisely characterized and hence a well-defined cohesion measure can possibly be developed.

**Definition 3.** For a method \( m \in M(c) \), let

1. \( \text{CALL}(m) \) be the set of methods directly called by \( m \)
2. \( \text{CALL}^*(m) \) be the set of methods directly or indirectly called by \( m \)
3. \( \text{RA}(m) \) be the attributes that are directly read by \( m \)
4. \( \text{RA}^*(m) \) be the attributes that are directly or indirectly read by \( m \)
5. \( \text{WA}(m) \) be the attributes that are directly written by \( m \)
6. \( \text{WA}^*(m) \) be the attributes that are directly or indirectly written by \( m \).

Actually, the following equations can be followed:

\[
\text{CALL}^*(m) = \{ m' \in M(c) | m' \in \text{CALL}(m) \lor \exists m'' \in \text{CALL}(m), m' \in \text{CALL}^*(m'') \} \\
\text{RA}^*(m) = \{ a \in A(c) | \exists m' \in \text{CALL}^*(m) a \in \text{RA}(m') \} \\
\text{WA}^*(m) = \{ a \in A(c) | \exists m' \in \text{CALL}^*(m), a \in \text{WA}(m') \}
\]

**Definition 4.** Let c be a class, then

1. If \( m \in M(c) \) and \( a \in \text{RA}(m) \), then \( m \) is read dependent on \( a \), denoted by \( m \longrightarrow a \).
2. If \( m \in M(c) \) and \( a \in \text{WA}(m) \), then \( a \) is write dependent on \( m \), denoted by \( a \longrightarrow m \).
3. If \( m_1, m_2 \in M(c) \) and \( m_2 \in \text{CALL}(m_1) \), then \( m_1 \) is call dependent on \( m_2 \), denoted by \( m_1 \longrightarrow \ldots \longrightarrow m_2 \).

Based on the above-mentioned definitions, the direct read dependence of methods on attributes, the direct write dependence of attributes on methods, and the direct call dependence between methods can easily be computed. However, the analysis of flow dependence between attributes is not so straightforward. To analyze the flow dependence between attributes in a method, the MCFG for the method is defined as follows.

**Definition 5.** A method control flow graph (MCFG) for method \( m \) in a class \( c \) is a directed graph \( G = (N,E,s_{\text{entry}}, s_{\text{exit}}) \), where \( N \) contains one node for each statement in \( m \), and \( E \) contains all edges such as \( (s_i, s_j) \) that represents statement \( s_j \) might be executed immediately after the execution of statement \( s_i \) (i.e. such edges represent possible flow of control between statements in \( m \)). Each call site in \( m \) is treated as a regular statement. \( s_{\text{entry}} \) and \( s_{\text{exit}} \) are two special nodes contained in \( N \) that represent entry to and exit from \( m \), respectively. If \( m \) contains multiple exit points, then \( E \) contains an edge from each node that represents an exit point to \( s_{\text{exit}} \).

In the traditional CFG [19], each call site is represented by a call node and a return node, and there is an edge from each call node to its associated return node. At the same time, there is an edge from the call node to the corresponding entry node of the CFG for the called method, and there is an edge from the exit node of the CFG for the called method to the return node. However, each calling statement is treated as one regular statement and is represented as one node in a MCFG.
For the sake of convenience, we assume that at most one variable is defined in one statement. However, some languages such as C++ allow that a few variables are defined in a statement. To construct the MCFG for a method, every statement that defines a few variables should be replaced by several statements, each of which only defines one variable.

**Definition 6.** For any two nodes \( s \) and \( s' \) in a MCFG \( G=(N,E,s_{entry},s_{exit}) \) for a method \( m \in M(c) \), \( s \) is control dependent on \( s' \) iff whether or not \( s \) can be executed depends on the execution result of \( s' \). And let \( CD(m) = \{ (s,s') \mid s \text{ is control dependent on } s' \} \)

**Definition 7.** For any node \( s \) in a MCFG \( G=(N,E,s_{entry},s_{exit}) \) for a method \( m \in M(c) \), the following sets are defined.

1. \( Def(s) = \{ v \mid v \text{ is a variable defined in } s \} \)
2. \( Ref(s) = \{ v \mid v \text{ is a variable referenced in } s \} \)
3. \( In(s) = \{ (s',v) \mid v \in Def(s') \textit{ and } \exists \text{ a path from } s' \text{ to } s \text{ in which } v \text{ is not redefined} \} \)
4. \( Out(s) = \{ (s',v) \mid (s',v) \in In(s) \text{ and } v \notin Def(s) \} \cup \{ (s,v) \mid v \in \text{Def}(s) \} \)

**Definition 8.** For any two statements \( s_1 \) and \( s_2 \) in a MCFG \( G=(N,E,s_{entry},s_{exit}) \), the variable \( v_1 \) defined in \( s_1 \) is direct flow dependent on the variable \( v_2 \) defined in \( s_2 \), denoted by \( (s_1,v_1) \rightarrow (s_2,v_2) \), iff either of the following conditions holds.

- \( v_1 \subseteq Def(s_1) \land v_2 \subseteq Ref(s_2) \land (s_2,v_2) \in In(s_1) \)
- \( (s_1,s_2) \subseteq CD(m) \land v_1 \subseteq Def(s_1) \land v_2 \subseteq Def(s_2) \)
- \( s_1 \in N \setminus (s_1,s_2) \subseteq CD(m) \land v_1 \subseteq Def(s_1) \land v_2 \subseteq Ref(s_2) \land (s_2,v_2) \in In(s_3) \)

In a sequence program, the flow dependencies between variables are transitive, i.e. if \( (s_1,v_1) \rightarrow (s_2,v_2) \), and \( (s_2,v_2) \rightarrow (s_3,v_3) \), then \( (s_1,v_1) \rightarrow (s_3,v_3) \). Consequently, the transitive flow dependencies between variables can be defined as follows.

**Definition 9.** For any three statements \( s_1 \), \( s_2 \), and \( s_3 \) in a MCFG \( G=(N,E,s_{entry},s_{exit}) \), the variable \( v_1 \) defined in \( s_1 \) is transitively dependent on the variable \( v_3 \) defined in \( s_3 \), denoted by \( (s_1,v_1) \rightarrow (s_3,v_3) \), if and only if either of the following conditions holds.

- \( (s_1,v_1) \rightarrow (s_3,v_3) \)
- \( (s_1,v_1) \rightarrow (s_2,v_2) \land (s_2,v_2) \rightarrow (s_3,v_3) \)

It should be noted that all attributes directly and indirectly read or written in the body of method \( m \) are also regarded as its formal parameters. For any node \( s \) in a MCFG \( G=(N,E,s_{entry},s_{exit}) \), \( In(s) \cup \bigcup_{i \in Pred(s)} Out(s_i) \) where \( Pred(s) = \{ (s',s) \mid s' \in E \} \). If \( s \) is a regular statement, then the computations of \( Def(s) \) and \( Ref(s) \) are trivial. However, if \( s \) is the entry node, then \( Def(s) \) consists of all formal parameters and \( Ref(s) \) is an empty set. If \( s \) is a calling statement that calls another method \( m' \), then \( m' \) must first be analyzed to compute the dependencies among its formal parameters. Then, each formal parameter is replaced with the corresponding actual parameter. Therefore, \( Def(s) \) consists of these actual parameters which depend on other actual parameters, and \( Ref(s) \) consists of these actual parameters on which other actual parameters depend. For dependence analysis among formal parameters of a recursive method, we can directly apply the algorithm proposed in the Ref. [20].

**Definition 10.** For any statements \( s \) in a MCFG \( G=(N,E,s_{entry},s_{exit}) \), \( v \in Def(s) \), let

\[ Def_{\_vf}(s,v) = \{ (s',v') \mid (s,v) \rightarrow (s',v') \} \]

**Definition 11.** Given a MCFG \( G=(N,E,s_{entry},s_{exit}) \) for a method \( m \in M(c) \), \( a, b \in RA^*(m) \cup WA^*(m) \). If there is a node \( s \) in \( G \), \( (s_{entry}, b) \in Def_{\_vf}(s, a) \), and \( (s, a) \in In(s_{exit}) \), then \( a \) at the exit of \( m \) is flow dependent on \( b \) outside \( m \), denoted by \( a \rightarrow b \). And let

\[ FD(m) = \{ (a, b) \mid s \in N \setminus (s_{entry}, b) \in Def_{\_vf}(s, a) \wedge (s, a) \in In(s_{exit}) \} \]

Based on these definitions, the class element dependence graph of a class is defined as follows.

**Definition 12.** A class element dependence graph (CEDG) for a class \( c \) is a cohesion model that depicts the dependencies among the elements in \( c \), and is defined to be a directed graph \( G_e = (N_e, E_e) \) with

- \( N_e = NM(c) \cup A(c) \)
- \( E_e = \{ <m, a> \mid m \rightarrow a, m \in NM(c), a \in A(c) \} \)

- \( \cup \{ <a, m> \mid a \rightarrow m, m \in NM(c), a \in A(c) \} \)
- \( \cup \{ <m', m> \mid m \rightarrow m', m, m' \in NM(c) \} \)
- \( \cup \{ <a, b> \mid a \rightarrow b, a, b \in A(c) \} \)

It is clear that the class element dependence graph of a class consists of two types of nodes (normal methods and attributes) and four types of edges that represent read dependencies, write dependencies, call dependencies, and flow dependencies among these nodes, respectively. In what follows, these four types of dependencies are also denoted as explicit dependencies.

### 3. Dependence matrix based class cohesion measure

In this section, we first discuss how to develop a cohesion measure based on the CEDG of a class. Then, we use Briand’s criterion to verify our measure and show that our measure has the properties that a well-defined cohesion should have.
3.1. Measure definition

As Section 2 states, given a class c, all of the read dependencies of methods on attributes, the write dependencies of attributes on methods, the call dependencies between methods, and the flow dependencies between attributes are depicted in its CEDG. All these dependencies contribute to the cohesiveness of c.

However, except these explicit dependencies above mentioned, there are implicit (indirect or potential) dependencies among attributes and methods. For example, if a method m1 is called by m2 and m2 is called by m3, then the behavior of m3 is indirectly dependent on m1. If a method m1 reads an attribute a and m2 calls m1, then a is indirectly read by m2. Also, the sequence of invoked normal methods in a class can not be assumed. If a method m1 is immediately invoked after the invocation of a method m2, m1 uses an attribute a, a is defined in m2, and the definition is still live out of the exit of m2, then the behavior of m1 is dependent on the behavior of m2. It is clear that such potential dependencies occur only if some special conditions are satisfied (such as a special sequence of invoked methods). Actually, all implicit dependencies among attributes and methods are determined by the definition of a class. If the methods and attributes in a class have been defined, then the implied indirect or potential dependencies among them are fixed. That is, the implicit dependencies among the elements in a class also contribute to its cohesiveness.

It should be noted that not only explicit dependencies but also implicit dependencies among the elements in a class are represented in its CEDG. Actually, a path whose length is two or more reflects an implicit dependence. Only if all these dependencies between the elements in a class are considered, its cohesiveness can possibly be evaluated objectively. However, most of the existing cohesion measures only consider the direct read dependencies of methods on attributes, and/or the call dependencies between methods. The direction of the dependencies between methods and attributes, and the potential dependencies are not considered. In our previous work [13], the potential dependence between methods is considered, but no attention has been paid on these potential dependencies between methods and attributes. For example, if a method m1 is immediately invoked after the invocation of a method m2, m1 uses an attribute a, and a is flow dependent on b in m2, then m1 is potential dependent on b. Therefore, we believe that the cohesiveness of a class cannot be precisely evaluated by the existing cohesion measures.

For any two nodes n1 and n2 in a CEDG Gc = (Nc, Ec), if there is a path from n1 to n2, then n1 is dependent on n2 along the path. In other words, if n2 has logical error or some trouble, then the influence of such an error will be reversely transmitted to n1 through these edges on the path. To quantify the degree of such dependence, the dependence degrees of explicit dependence types should be specified firstly (since a path consists of a lot of explicit dependencies). The dependence degrees of different explicit dependence types can be characterized by weights shown in Table 1. It should be noted that \( W_i \in [0, 1] \) (for \( 1 \leq i \leq 4 \)) is required. Generally speaking, the weight of read dependencies is less than that of the other dependencies.

<table>
<thead>
<tr>
<th>Dependence types</th>
<th>Dependence degrees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read dependence</td>
<td>( w_1 )</td>
</tr>
<tr>
<td>Write dependence</td>
<td>( w_2 )</td>
</tr>
<tr>
<td>Call dependence</td>
<td>( w_3 )</td>
</tr>
<tr>
<td>Flow dependence</td>
<td>( w_4 )</td>
</tr>
</tbody>
</table>

Based on above analysis, the dependence degree of a path is defined as follows.

Definition 13. Let n1, n2 be two nodes in a CEDG \( G_c = (N_c, E_c) \). If there is a path from n1 to n2, then the dependence degree that n1 is dependent on n2 on the path is the product of the dependence degrees of all edges on the path.

Thus, the dependence degree between a pair of nodes is defined as follows.

Definition 14. Let n1, n2 be two nodes in a CEDG \( G_c = (N_c, E_c) \). The degree that n1 is dependent on n2 is the sum of the dependence degrees on all paths from n1 to n2.

Actually, for any CEDG \( G_c = (N_c, E_c) \), an adjacency matrix which reflects the dependence degree of explicit dependencies among nodes can be constructed as follows.

Definition 15. Given a CEDG \( G_c = (N_c, E_c) \), its adjacency matrix \( W_c \) is defined as

\[
W_c(n_1, n_2) = \begin{cases} 
  w_1 & \text{if } \{n_1, n_2\} \in E_c \land n_1 \in NM(c) \land n_2 \in A(c) \\
  w_2 & \text{if } \{n_1, n_2\} \in E_c \land n_1 \in A(c) \land n_2 \in NM(c) \\
  w_3 & \text{if } \{n_1, n_2\} \in E_c \land n_1, n_2 \in NM(c) \\
  w_4 & \text{if } \{n_1, n_2\} \in E_c \land n_1, n_2 \in A(c) \\
  0 & \text{otherwise}
\end{cases}
\]

The adjacency matrix \( W_c \) describes the dependence degrees among each pair of nodes caused by a path whose length is one. Based on the adjacency matrix \( W_c \), a dependence matrix, \( D_c \), which reflects the dependence degrees among each pair of nodes on a CEDG, is defined as follows.

Definition 16. Given a CEDG \( G_c = (N_c, E_c) \) for a class c (|\( N_c | \geq 2 \)), then its dependence matrix is defined as:

\[
D_c = \frac{1}{|N_c| - 1} \sum_{k=1}^{\left|N_c\right| - 1} W_c^k
\]

Based on these definitions above mentioned, the cohesiveness of a class can be defined as follows.
Definition 17. Given a CEDG $G_c=(N_c, E_c)$ for a class $c$, then its cohesion is defined as:

$$D(c) = \begin{cases} 1 & |N_c| = 1 \\ \frac{|X| |Y| \sum_{i=1}^{|X|} \sum_{j=1}^{|Y|} D(i, j)}{|N_c^2|} & |N_c| \geq 2 \end{cases}$$

If $|N_c| = 1$, then there is only one normal method or attribute in $c$. The dependence of a node on itself is maximum, thus the cohesion of the class is defined as 1. In other cases, the cohesiveness of $c$ is related to its dependence matrix. It is clear that not only explicit dependencies but also indirect and potential dependencies between the elements in a class are considered in $D(c)$.

3.2. Theoric validation

When defining the cohesion of a class $c$, most measures first use a directed or undirected graph $G_c$ to represent the interactions among the class elements, and then define class cohesion as a function of the number of interactions. If $R_c$ represents the set of relationships between $G_c$’s nodes, a well-defined class cohesion measure should have the following four properties: cohesion 1–cohesion 4 [14].

Cohesion 1. Non-negativity and normalization. The cohesion of a class $c$ is within a specified range, where Max is a positive constant number:

$$\text{Cohesion}(c) \in [0, \text{Max}]$$

Cohesion 2. Null value. The cohesion of a class $c$ is 0 if $R_c$ is empty:

$$R_c = \emptyset \Rightarrow \text{Cohesion}(c) = 0$$

Cohesion 3. Monotonicity. Suppose $G_{c'}$ to be a new graph created by adding some edges to $G_c$, and $c'$ is the new class corresponding to $G_{c'}$, then the Lack_of_Cohesion of $c'$ is no more than that of $c$:

$$R_c \leq R_{c'} \Rightarrow \text{Cohesion}(c) \leq \text{Cohesion}(c')$$

Cohesion 4. Merging of classes. If two unrelated classes $c_1$ and $c_2$ are merged to form a new class $c_3$, then the Lack_of_Cohesion of $c_3$ is no less than the minimum cohesion of $c_1$ and $c_2$:

$$\text{Max}\{\text{Cohesion}(c_1), \text{Cohesion}(c_2)\} \geq \text{Cohesion}(c_3)$$

If a cohesion measure is an inverse measure, then a low value indicates high cohesion and vice versa. As a result, Cohesion 2–Cohesion 4 are not appropriate for such measures. In order to apply these properties to an inverse measure, Briand adapted their definitions as follows, where Lack_of_Cohesion denotes an inverse cohesion measure [15].

Cohesion 2. Null value. The Lack_of_Cohesion of a class $c$ is maximal if $R_c$ is empty:

$$R_c = \emptyset \Rightarrow \text{Lack_of_Cohesion}(c) = \text{Max}$$

Cohesion 3. Monotonicity. Suppose $G_{c'}$ to be a new graph created by adding some edges to $G_c$, and $c'$ is the new class corresponding to $G_{c'}$, then the Lack_of_Cohesion of $c'$ is no more than that of $c$:

$$R_c \leq R_{c'} \Rightarrow \text{Lack_of_Cohesion}(c) \geq \text{Lack_of_Cohesion}(c')$$

Cohesion 4. Merging of classes. If two independent classes $c_1$ and $c_2$ are merged to form a new class $c_3$, then the Lack_of_Cohesion of $c_3$ is no less than the minimum cohesion of $c_1$ and $c_2$:

$$\text{Min}\{\text{Lack_of_Cohesion}(c_1), \text{Lack_of_Cohesion}(c_2)\} \leq \text{Lack_of_Cohesion}(c_3)$$

If a cohesion measure does not satisfy all of the four properties, then it is questionable to evaluate the cohesiveness of a class. Our cohesion measure, $D(c)$, not only precisely represents the relationships among the elements in a class, but also satisfies Cohesion 1–Cohesion 4. In other words, $D(c)$ has these properties that a well-defined cohesion measure should have. Prior to their proofs, it is necessary to prove the following lemma.

Lemma 1. Assume $X$ is a square matrix with $0 \leq X(i,j) \leq 1$ ($1 \leq i, j \leq n$), and $S(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} X(i,j)$. For any real $r$, any square matrix $Y$ that has the same number of elements as $X$, any matrix $Z$, the following conclusions can be drawn.

1. $S(rX) = rS(X)$
2. $S(X + Y) = S(X) + S(Y)$
3. $S\left(\begin{bmatrix} X & O \\ O & Z \end{bmatrix}\right) = S(X) + S(Z)$, where $O$ denotes the area filled with 0.
4. $S\left(\frac{X^k}{n^{k-1}}\right) \geq S\left(\frac{X^{k+1}}{n^k}\right)$, where $k$ is an integer and is larger than 0.

Proof. The conclusions (1)–(3) are trivial. To prove the conclusion (4), let

$$B = \frac{X^k}{n^{k-1}}.$$ 

Then,

$$S\left(\frac{X^k}{n^{k-1}}\right) = \sum_{i=1}^{n} \sum_{j=1}^{n} B(i,j)$$

Therefore,

$$S\left(\frac{X^{k+1}}{n^k}\right) = S\left(\frac{X^k}{n^{k-1}}\right) + \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} B(i,k) \frac{X(k,j)}{n}.$$
However, \(0 \leq X(i,j) \leq 1\) (1 \( \leq i, j \leq n\)). So

\[
S \left( \frac{X^{k+1}}{n^k} \right) \leq \sum_{i=1}^{n} \sum_{j=1}^{n} \left[ \sum_{k=1}^{n} B(i,k) \frac{1}{n} \right] \\
= \sum_{i=1}^{n} \left[ n \sum_{k=1}^{n} B(i,k) \frac{1}{n} \right] = \sum_{i=1}^{n} \sum_{k=1}^{n} B(i,k)S \left( \frac{X^k}{n^{k-1}} \right) \tag*{\Box}
\]

Theorem 1. For any class \(c\), \(0 \leq DMC(c) \leq 1\).

Proof. If \(|N_c| = 1\), then \(DMC(c) = 1\); otherwise, \(0 \leq W_c(i,j) \leq 1\), \(1 \leq i, j \leq |N_c|\). Hence, \(0 \leq W_c^k(i,j) \leq |N_c|^{k-1}\), \(k \geq 1\). Let

\[
B = \sum_{k=1}^{n} \frac{W^k_c}{|N_c|^{k-1}},
\]

then it is easy to conclude that \(0 \leq B(i,j) \leq |N_c| - 1\), \(1 \leq i, j \leq |N_c|\). Since

\[
D_c = \frac{B}{|N_c| - 1},
\]

\(0 \leq D_c(i,j) \leq 1\). Therefore, \(0 \leq DMC(c) = \frac{S(D_c)}{|X|} \leq 1\). \(\Box\)

Theorem 2. If \(G_c = (N_c, E_c)\) is the CEDG of a class \(c\) and \(|N_c| > 1\), then \(E_c = \emptyset \Rightarrow DMC(c) = 0\).

Proof. If \(E_c = \emptyset\), then \(W_c(i,j) = 0\) (1 \( \leq i, j \leq |N_c|\)). Hence, \(D_c(i,j) = 0\) (1 \( \leq i, j \leq |N_c|\)). Therefore, \(DMC(c) = 0\). \(\Box\)

Theorem 3. Suppose the CEDGs of \(c_1\) and \(c_2\) to be \(G_{c_1} = (N_{c_1}, E_{c_1})\), \(G_{c_2} = (N_{c_2}, E_{c_2})\), respectively. If \(N_{c_1} = N_{c_2}\) and \(E_{c_1} \subseteq E_{c_2}\), then \(DMC(c_1) \leq DMC(c_2)\).

Proof. If \(|N_{c_1}| = 1\), then \(DMC(c_1) = DMC(c_2) = 1\). Otherwise, since \(N_{c_1} = N_{c_2}\) and \(E_{c_1} \subseteq E_{c_2}\), \(W_{c_1}(i,j) \leq W_{c_2}(i,j)\) (1 \( \leq i, j \leq |N_{c_1}|\)). Therefore, \(S(W_{c_1}^k) < S(W_{c_2}^k)\) (1 \( \leq k\)). So

\[
DMC(c_2) = \frac{S \left( \frac{1}{|N_{c_2}| - 1} \sum_{k=1}^{|N_{c_2}|} W_{c_2}^k \right)}{|N_{c_2}|^2} \\
= \frac{S \left( \frac{1}{|N_{c_1}| - 1} \sum_{k=1}^{|N_{c_1}|} W_{c_1}^k \right)}{|N_{c_1}|^2} \\
> \frac{S \left( \frac{1}{|N_{c_1}| - 1} \sum_{k=1}^{|N_{c_1}|} W_{c_1}^k \right)}{|N_{c_1}|^2} \\
= DMC(c_1)
\]

Therefore, \(DMC(c_1) \leq DMC(c_2)\). \(\Box\)

Theorem 4. If two unrelated classes \(c_1\) and \(c_2\) are merged to form a new class \(c\), then

\[
\text{Max}\{DMC(c_1), DMC(c_2)\} \geq DMC(c)
\]

Proof. Suppose the CEDGs of \(c_1\), \(c_2\), and \(c\) to be \(G_{c_1} = (N_{c_1}, E_{c_1})\), \(G_{c_2} = (N_{c_2}, E_{c_2})\), \(G_c = (N_c, E_c)\), respectively.

Because \(c\) is constructed by merging \(c_1\) and \(c_2\), \(|N_c| = |N_{c_1}| + |N_{c_2}|\).

Case 1. \(|N_{c_1}| = 1\) or \(|N_{c_2}| = 1\).

If \(|N_{c_1}| = 1\), then \(DMC(c_1) = 1\). Because \(c\) is the merger of two unrelated classes \(c_1\) and \(c_2\),

\[
W_c = \begin{bmatrix} W_{c_1} & O \\ O & W_{c_2} \end{bmatrix},
\]

where \(O\) denotes the area that is filled with 0. According to definition 16 and definition 17, it is easy to conclude that \(DMC(c) < 1\). Hence, \(DMC(c_1) > DMC(c)\). Similarly, if \(|N_{c_2}| = 1\), then we can also conclude that \(DMC(c_1) > DMC(c)\). Therefore, \(\text{Max}\{DMC(c_1), DMC(c_2)\} \geq DMC(c)\).

Case 2. \(|N_{c_1}| \geq 2\) and \(|N_{c_2}| \geq 2\)

\[
S(Dec) = \frac{1}{|N_c| - 1} \left( \sum_{k=1}^{|N_{c_1}|} \frac{W_{c_1}^k}{|N_{c_1}|^{k-1}} \right) + \frac{1}{|N_{c_2}| - 1} \left( \sum_{k=1}^{|N_{c_2}|} \frac{W_{c_2}^k}{|N_{c_2}|^{k-1}} \right)
\]

However,

\[
\frac{1}{|N_{c_1}| - 1} \left( \sum_{k=1}^{|N_{c_1}|} \frac{W_{c_1}^k}{|N_{c_1}|^{k-1}} \right) + \frac{1}{|N_{c_2}| - 1} \left( \sum_{k=1}^{|N_{c_2}|} \frac{W_{c_2}^k}{|N_{c_2}|^{k-1}} \right)
\]

\[
= \frac{1}{|N_{c_1}| - 1} \left( \sum_{k=1}^{|N_{c_1}|} \frac{W_{c_1}^k}{|N_{c_1}|^{k-1}} \right) + \frac{1}{|N_{c_2}| - 1} \left( \sum_{k=1}^{|N_{c_2}|} \frac{W_{c_2}^k}{|N_{c_2}|^{k-1}} \right)
\]

\[
+ \left( |N_{c_1}| - 1 \right) S(D_{c_1}) + \left( |N_{c_2}| - 1 \right) S(D_{c_2})
\]

\[
\leq \frac{1}{|N_{c_1}| - 1} \left( \sum_{k=1}^{|N_{c_1}|} \frac{W_{c_1}^k}{|N_{c_1}|^{k-1}} \right) + \left( |N_{c_2}| - 1 \right) S(D_{c_2})
\]

\[
+ \left( |N_{c_2}| - 1 \right) S(D_{c_2})
\]

\[
= \text{Max}\{DMC(c_1), DMC(c_2)\} \geq DMC(c)
\]
Suppose Max\{\text{DMC}(c_1), \text{DMC}(c_2)\} = \text{DMC}(c_1), then \\
|N_c|^2 \text{DMC}(c) \leq |N_c|^2 \text{DMC}(c_1) + |N_c|^2 \text{DMC}(c_2) \\
\leq |N_c|^2 \text{DMC}(c_1) + |N_c|^2 \text{DMC}(c_1) \\
\leq (|N_c|^1 + |N_c|^2) \text{DMC}(c_1) \\
= |N_c|^2 \text{DMC}(c_1) \\
\text{So, } \text{Max}\{\text{DMC}(c_1), \text{DMC}(c_2)\} \geq \text{DMC}(c) \square \\

These four theorems state that DMC satisfies Cohesion 1–Cohesion 4 proposed by Briand. Therefore, DMC satisfies the necessary conditions of a cohesion measure. From these definitions, lemma, and definitions above mentioned, the following corollaries can be easily concluded.

**Corollary 1.** For any class c 

\[
\frac{1}{|N_c||N_c|}S(W_c^{|N_c|-1}) \leq \text{DMC}(c) \leq \frac{S(W_c)}{|N_c|^2}
\]

**Proof.** According to lemma 1,

\[
s\left(\frac{W_c^k}{|N_c|^{k-1}}\right) \geq s\left(\frac{W_c^{k+1}}{|N_c|^{k}}\right), \; k \geq 1.
\]

Therefore,

\[
s\left(\frac{W_c^{|N_c|-1}}{|N_c|^{|N_c|-2}}\right) \leq \text{DMC}(c) \leq s\left(\frac{W_c^{|N_c|-1}}{|N_c|^{|N_c|-2}}\right)
\]

According to definition 17,

\[
\text{DMC}(c) = \frac{S(D_c)}{|N_c|^2} = \frac{1}{|N_c|^2} \sum_{k=1}^{|N_c|-1} s\left(\frac{W_c^k}{|N_c|^k}\right).
\]

So,

\[
\frac{1}{|N_c|^2}S(W_c^{|N_c|-1}) \leq \text{DMC}(c) \leq \frac{S(W_c)}{|N_c|^2}.
\]

**Corollary 2.** If the CEDG of a class c1 is G_{c1} = (N_{c1}, E_{c1}), where |N_{c1}| \geq 2, and the CEDG of another class c2 is the same as the graph that is constructed by adding an edge to G(c1), then DMC(c1) < DMC(c2).

**Corollary 3.** If two unrelated classes c1 and c2 are merged to form a new class c, and Max\{DMC(c1), DMC(c2)\} > 0, then Max\{DMC(c1), DMC(c2)\} > DMC(c).

Given a class c, the upper bound and low bound of its cohesion can be computed by corollary 1. Corollary 2 means that adding an edge to a CEDG in which the number of nodes is more than one will strictly increase its cohesion. Corollary 3 implies that if either one of two CEDGs has edges, then the cohesion of the new class by their merger is strictly less then the max of their cohesions.

4. Case study

Consider the class Stack with three attributes shown in Fig. 1(a). The class Stack consists of seven methods: a constructor Stack, a destructor ~Stack, an access method Size, and four normal methods Push, Pop, Vtop and Isempty. According to definition 1, we have

\[
M(\text{Stack}) = \{\text{Stack}, \; \sim\text{Stack}, \; \text{Size}, \; \text{Push}, \; \text{Pop}, \; \text{Vtop}, \; \text{Isempty}\}
\]

\[
A(\text{Stack}) = \{\text{array, top, size}\}
\]

According to definition 2, we have

\[
SM(\text{Stack}) = \{\text{Stack}, \; \sim\text{Stack}, \; \text{Size}\}
\]

\[
NM(\text{Stack}) = \{\text{Push, Pop, Vtop, Isempty}\}
\]

According to definition 3, it is easy to conclude that

\[
\text{CALL}(\text{Push}) = \emptyset
\]

\[
\text{RA}(\text{Push}) = \{\text{size, top}\}
\]

\[
\text{WA}(\text{Push}) = \{\text{top, array}\}
\]

\[
\text{CALL}(\text{Pop}) = \{\text{Isempty}\}
\]

\[
\text{RA}(\text{Pop}) = \text{WA}(\text{Pop}) = \{\text{top}\}
\]
The analysis of flow dependencies among attributes in the class Stack is not so straightforward. Fig. 1(b) is the MCFG for Push. Since $s_2$ and $s_3$ are two branch statements of the condition statement $s_1$, both $s_2$ and $s_3$ are control dependent on $s_1$, i.e.,

$$\text{CD}(\text{Push}) = \{(s_2, s_1), (s_3, s_1), (s_4, s_1), (s_4, s_1)\}.$$ 

According to definition 7, we have

$$\text{Def}(s_{\text{entry}}) = \{(\text{top}, \text{size}, \text{array}, \text{item})\},$$

$$\text{Ref}(s_{\text{entry}}) = \{(\text{top}, \text{size})\}.$$

According to definitions 8–10, we have

$$\text{Def}_v(s_3, \text{array}) = \{(s_{\text{entry}}, \text{top}), (s_{\text{entry}}, \text{size}), (s_3, \text{array})\},$$

$$\text{Ref}_v(s_4, \text{top}) = \{(s_{\text{entry}}, \text{top}), (s_{\text{entry}}, \text{size})\}.$$ 

According to definition 11, we conclude that

$$\text{FD}(\text{Push}) = \{(\text{top, top}), (\text{top, size}), (\text{array, top}), (\text{array, size})\}.$$ 

Therefore, the CEDG for Stack is constructed as Fig. 1(c). The adjacency matrix $W_{\text{Stack}}$ for Stack is shown in Fig. 2(a), where $w_{ij} = 1$, $1 \leq i \leq 4$. According to definition 16, the corresponding dependence matrix is easy to be concluded (shown in Fig. 2(b)). Since top is flow dependent on itself in Push or Pop, $W_{\text{Stack}}(\text{top, top}) = 1$, while other the elements of the diagonal elements have the value 0. Note that $D_{\text{Stack}}$ depicts the dependence degrees among each pair of nodes caused by all paths between them. Therefore, for a node $n$, if the value of $W_{\text{Stack}}(n, n)$ is zero, we cannot
conclude that the value of $D_{\text{Stack}}(n, n)$ is also zero. For example, $W_{\text{Stack}}(\text{array}, \text{Pop}) = 0$ but $D_{\text{Stack}}(\text{array}, \text{Pop}) = 0.0334$. According to definition 17, it follows that $DMC(\text{Stack}) = 0.0712$.

For checking whether the intuitive understanding of cohesion corresponds to the formal notation, we consider another class, Stack2, shown in Fig. 3(a). Stack2 is created by deleting the statement that checks if the stack is full from the method Push in the original class Stack. Although other attributes and methods in Stack2 are the same as those in Stack, the modification of Push results in the dependencies among the class elements being simpler. In particular, the attribute size is virtually not be used by any normal method in Stack2. This states that the connectivity among the elements in Stack2 is weaker than that in Stack, i.e. Stack2 is intuitively less cohesive than Stack.

We next analyze the dependencies among the elements in Stack2. Clearly, all four sets, CALL, RA, WA, and FD, related to Pop, Vtop, and Isempty remain no change. For Push in Stack2, it is easy to conclude that

\[
\begin{align*}
\text{CALL}(\text{Push}) &= \emptyset \\
\text{RA}(\text{Push}) &= \{\text{top}\} \\
\text{WA}(\text{Push}) &= \{\text{top, array}\} \\
\text{FD}(\text{Push}) &= \{(\text{top, top})\}
\end{align*}
\]

Therefore, the CEDG for Stack2 is constructed as Fig. 3(b). Compared with the CEDG for Stack, four dependencies disappear: the read dependence of Push on size, the flow dependence of top on size, the flow dependence of array on size, and the flow dependence of array on top. Fig. 4(a) depicts the adjacency matrix $W_{\text{Stack2}}$ (where $w_{ij} = 1$, $1 \leq i \leq 4$) and Fig. 4(b) depicts the corresponding dependence matrix. According to definition 17, $DMC(\text{Stack2}) = 0.0486$. Clearly, $DMC(\text{Stack2}) < DMC(\text{Stack})$. The measurement results indicate that the measure $DMC$ matches our intuitive understanding of cohesion.

5. Comparison with related work

Object-oriented software measurement has become an increasingly popular research area. Specially, many researchers focus on object-oriented cohesion measures. As a result, a large number of cohesion measures for classes in object-oriented systems have been proposed in the last decade. This section gives a brief review of typical cohesion measures and compares them with $DMC$.
LCOM measure can be described as follows: Then, Chidamber and Kemerer’s cohesion attributes. Therefore, LCOM1 is a inverse measure. The higher the value of LCOM1 is, the lower the class cohesion is. It is clear that
\[
\text{LCOM1}(c) = \begin{cases} 
|P| - |Q| & \text{if } |P| > |Q| \\
0 & \text{otherwise}
\end{cases}
\]
Since the upper bounds of LCOM1 is related the number of methods in a class, it does not satisfy normalization property. In addition, if the methods in c do not access any of its attributes, then LCOM1(c)=0. If c is modified so that one method accesses exactly one attribute, then LCOM1(c)>0. Therefore, LCOM1 also violates monotonicity.

5.2. LCOM2 by Hitz and Montazeri [2]

Hitz and Montazeri used an undirected graph \(G_c=(N_c,E_c)\) to represent the relationships among the methods in a class c, where \(N_c=M(c)\) and \(E_c=\{(m_i,m_j)|m_i\) and \(m_j\) share at least one attribute \(\forall m_i\) calls \(m_j\) and \(\forall m_j\) calls \(m_i\). Then, Hitz defined the cohesion of c as the number of connected components of \(G_c\).

\[
\text{LCOM2}(c) = |\text{connected components of } G_c|
\]

However, if two graphs have the same number of connected components, different classes will have the same LCOM2 value. Specially, if \(G_c\) is a connected graph, then \(\text{LCOM2}(c)=1\). To compare the cohesion of all classes whose LCOM2 values are 1, Hitz defined a new \textit{Conn} measure as follows (To avoid name confictions, we use ‘Conn’ instead of the original name C).

\[
\text{Conn}(c) = 2 - \frac{|E_c| - (|N_c| - 1)}{(|N_c| - 1)(|N_c| - 2)}
\]

It is clear that \(\text{Conn}(c)\in[0,1]\). When \(G_c\) is a tree, c achieves the minimum cohesion value 0. When \(G_c\) is a complete graph, c achieves the maximum value 1. Since the upper bound of LCOM2 is related to the number of the methods in a class, LCOM2 is not normalized.

5.3. LCOM3 by Henderson [6]

Assuming that each attribute in a class c is accessed by at least one method, Henderson defined LCOM3 as follows:

\[
\text{LCOM3}(c) = \frac{1}{|M(c)|-1} \sum_{a\in A(c)} |\mu(a,c)| - |M(c)|
\]

where \(\mu(a,c) = \{m|a\in A(c)\land m\in M(c)\land a\) is directly defined or referenced in m\}.

In real programs, it is possible that an attribute is not accessed by any method. In this case, LCOM3 is not normalized. The reason is that its upper bound

\[
\frac{|M(c)|}{|M(c)|-1}
\]

is related to the number of the methods in c.

5.4. RCI by Briand et al. [7]

Briand et al. thought that a class consists of data declarations and methods. For two data declarations a and b, a DD-interaction b only if a change in a’s declaration or use may cause the need for a change in b’s declaration or use. For data declaration a and method m, a DM-interaction m only if a DD-interaction with at least one data declaration of m. Data declarations of methods include their parameters, return types, and local variables. All DD-interactions between data declarations and DM-interactions can be determined from the class interface and can be represented on a cohesive interaction graph. Based on these notions, Briand et al. defined cohesion measure at design level as follows:

\[
\text{RCI}(c) = \frac{|\text{Cl}(c)|}{|\text{Max}(c)|}
\]

where \(\text{Cl}(c)\) is the set of all DD- and DM-interactions and \(\text{Max}(c)\) is the set of all possible DD- and DM-interactions of c. Actually, for a given class c, RCI(c) is the ratio of the number of actual interaction to the number of possible interactions.
For a given class \( c \), Chae et al. uses a reference graph \( G_c = (N_c, E_c) \) to represent the access relationships among its methods and attributes, where \( N_c = NM(c) \cup A(c) \) and \( E_c = \{(m, a) \mid a \in A(c) \land m \in NM(c) \land a \text{ is an attribute accessed by } m \text{ directly or indirectly}\} \). For example, Fig. 5 depicts the reference graphs for two classes \( c_1 \) and \( c_2 \).

For a reference graph \( G_c \), the set of glue methods \( M_g(G_c) \) is a set of normal methods of \( c \) that satisfies the following two characters:

(a) Without the elements of \( M_g(G_c) \) and related edges, \( G_c \) becomes disjointed.
(b) \( M_g(G_c) \) is the minimum set of normal methods that holds character (a).

After all methods of \( M_g(G_c) \) and related edges are removed, each connected sub graph is called a cohesion component (CC). If each method has interactions with all attributes in a CC, the CC is a most cohesive component (MCC). The cohesion of a MCC is defined as 1 in Chae et al.’s method. If each method has interactions with all attributes in a class, the class is a most cohesive class (MC). If the decomposition of a reference graph is not unique, then the set of glue methods, \( G_c \), for a given class \( c \) selected. Chae et al. defined the cohesion of a MCC is defined as 1 in Chae et al.’s method.

\( G_c \) is the \( i \)th sub graph of \( G_c \) and \( N_m(G_c) \) is the number of methods in \( G_c \) and \( n \) is the number of child nodes of \( G_c \).

\( \text{CBMC}(G_c) = F_c(G_c)F_s(G_c) \)

where

\[
F_c(G_c) = \frac{|M_g(G_c)|}{|N_m(G_c)|}
\]

and

\[
F_s(G_c) = \frac{1}{n} \sum_{i=1}^{n} \text{CBMC}(G_{c,i})
\]

(\( G_{c,i} \) is the \( i \)th sub graph of \( G_c \) and \( N_m(G_c) \) is the number of methods in \( G_c \) and \( n \) is the number of child nodes of \( G_c \).)

\( \text{CBMC} \) does not satisfy monotonicity. For example, for two classes \( c_1 \) and \( c_2 \) shown in Fig. 5, \( c_2 \) is more cohesive than \( c_1 \) due to the fact that the interactions on \( G_{c_2} \) are more complex than those on \( G_{c_1} \). However, \( \text{CBMC}(G_{c_1}) = 4/8 \), \( \text{CBMC}(G_{c_2}) = 3/8 \).

Among these typical cohesion measures above mentioned, \( \text{LCOM1} \) and \( \text{LCOM3} \) do not consider all special methods. \( \text{LCOM2} \) and \( \text{Conn} \) considered method invocation. However, neither access methods, nor other special methods are excluded from the method similarity graph of a class. \( \text{RCI} \) considered access methods and constructors but ignored destructors and delegation methods. In the definitions of \( \text{CBMC} \) and \( \text{DMC} \), all special methods are excluded.

Even if special methods have been processed properly, a cohesion measure that simply represents the relationships among the class elements can not accurately evaluate the cohesiveness. Previous cohesion measures did not distinguish read dependence and write dependence and regarded them as attribute reference or access (from methods to attributes). In the definitions of \( \text{LCOM1} \) and \( \text{LCOM2} \), so-called method similarity relationship, which is defined on the common attribute reference, was considered. Also, \( \text{LCOM2} \) considered call dependence. Although \( \text{RCI} \) is a cohesion measure at design level, it can be directly applied to source code level. In the definition of \( \text{RCI} \), so-called DD-interaction and DM-interaction are considered. Actually, they correspond to flow dependence and read dependence, respectively. In the definitions of \( \text{LCOM3} \) and \( \text{CBMC} \), only reference relationships are considered. Compared with previous cohesion measures, \( \text{DMC} \) considers not only all explicit dependencies, but also implicit dependencies.

On the other hand, although Briand et al.’s validation criterion is a necessary condition instead of a sufficient condition for a well-defined cohesion measure, it can validate an ill-defined measure. In these cohesion measures, only \( \text{RCI} \) and \( \text{DMC} \) satisfies Briand et al.’s validation criterion. Table 2 summarizes the differences among these cohesion measures.

### 6. Conclusions

This paper proposes a well-defined cohesion measure, \( \text{DMC} \), based on a dependence matrix that represents
the dependence degree among the elements in a class. In the definition of \( DMC \), not only special methods that do not contribute to the cohesiveness of classes are excluded, but also the relationships among the elements in a class are precisely characterized. Compared with the existing cohesion measures, \( DMC \) has the following advantages.

- Special methods in a class have no influence on the class cohesion [10–11,15]. Most of previous cohesion measures (except \( CBMC \)) do not process special methods properly. \( DMC \) excludes all special methods, which can give a more proper measurement of the cohesiveness.
- Previous cohesion measures consider attribute reference and/or method invocation in a simple manner. Neither the direction of dependencies between methods and attributes, nor flow dependencies and potential dependencies are characterized. \( DMC \) precisely considers the relationships among the elements in a class, which characterizes not only four types of explicit dependencies (read dependence, write dependence, call dependence, and flow dependence), but also the implied indirect and potential dependencies.
- The four cohesion properties defined by Briand characterize cohesion in a reasonably intuitive and rigorous manner [14]. It is likely that a measure which does not fulfill them all is ill-defined. \( DMC \) have all of the four cohesion properties, while most of previous cohesion measures (except \( RCI \)) do not.

In a word, \( DMC \) is a well-defined cohesion measure for classes, which can give a precise evaluation of the cohesiveness. To some extent, when we manually construct a CEDG and compute the cohesiveness for a class, the complexity is relatively high. However, tools to automate the measurement of class cohesion are feasible—they can make use of data flow analysis and control flow analysis libraries provided by the third party.

In the future work, we expect to develop a Visual C++ Add-in that can interacts with the current project to collect the information about classes, calculate the cohesiveness of classes easily, and perform an empirical study of the relationships between \( DMC \) and other cohesion measures.

**Acknowledgements**

We thank the anonymous referees for their careful reviews. Their comments greatly improved both the content and the presentation. We acknowledge the contribution of the referee who suggested that an example should be taken to check whether the intuitive understanding of cohesion corresponds to the formal notation. Supported in part by the National Natural Science Foundation of China (60373011), the National Key Fundamental Research Project (No. 2002CB312000), and the China Postdoctoral Science Foundation (No. 023290002).

**References**


