PAPER Special Section on Knowledge-Based Software Engineering

A Class Cohesion Metric Focusing on Cohesive-Part Size

Hirohisa AMAN[†a)], ***Member***, **Kenji YAMASAKI**[†], ***Nonmember***,
Hiroyuki YAMADA[†], ***Member***, ***and*** **Matu-Tarow NODA**[†], ***Nonmember***

SUMMARY Cohesion is an important software attribute, and it is one of significant criteria for assessing object-oriented software quality. Although several metrics for measuring cohesion have been proposed, there is an aspect which has not been supported by those existing metrics, that is "cohesive-part size." This paper proposes a new metric focusing on "cohesive-part size," and evaluates it in both of qualitative and quantitative ways, with a mathematical framework and an experiment measuring some Java classes, respectively. Through those evaluations, the proposed metric is showed to be a reasonable metric, and not redundant one. It can collaborate with other existing metrics in measuring class cohesion, and will contribute to more accurate measurement.
key words: *object-oriented software, metrics, cohesion, mathematical framework, correlation analysis*

1. Introduction

Cohesion is one of important attributes for software module, representing the degree to which its components are functionally connected within the module [1]–[3]. This attribute could be applied to object-oriented software, where a class corresponds to a module, and its attributes and methods correspond to the module components. In general, high level cohesion may lead to high maintainability, reusability and reliability [4], [5], so that class cohesion would be one of important criteria for assessing object-oriented software quality.

For measuring class cohesion, several metrics have been proposed, such as "Lack of Cohesion in Methods (*LCOM*)" [6]–[10], "Information flow-based Cohesion (*ICH*)" [11], "Tight Class Cohesion (*TCC*)" and "Loose Class Cohesion (*LCC*)" [12]. These metrics are mainly based on "the number of sets of connected methods," or "the density of method-connections/attribute-accesses" in a class. However there is an aspect of cohesion which has not been supported by the above metrics. That is "size of cohesive-part" in a class, in other words, "extent of associations among methods" through attribute-accesses and/or method-invocations in a class. In order to support such lacking aspect, and to contribute more accurate measurement of cohesion, this paper proposes a new cohesion metric focusing on cohesive-part size, and evaluates it in both of qualitative and quantitative ways.

Manuscript received June 20, 2003.
Manuscript revised September 26, 2003.
[†]The authors are with the Department of Computer Science, Faculty of Engineering, Ehime University, Matsuyama-shi, 790–8577 Japan.
a) E-mail: aman@cs.ehime-u.ac.jp

The rest of this paper is organized as follows. Section 2 briefly describes the existing eight metrics for measuring class cohesion. Section 3 presents a new cohesion metric focusing on cohesive-part size, and Sect. 4 evaluates it in both of qualitative and quantitative ways. Section 5 gives our conclusions and future works.

2. Existing Class Cohesion Metrics

Several metrics are proposed in order to measure cohesion of object-classes [6]–[12]. This section presents some brief descriptions of those existing metrics. See the literature [5] for their detailed discussion.

2.1 Lack of Cohesion in Methods (*LCOM*)

"Lack of COhesion in Methods" (*LCOM*) is a well-known class cohesion metric which quantifies poorness of cohesion. The higher value represents the lower cohesion, and vice versa. *LCOM* has thus far been discussed by several researchers [6]–[10]. Through their works, there are the following five different types of *LCOM*. These metrics have studied on usefulness as predictors or indicators of maintenance effort [13], fault-proneness [14] and so on.

2.1.1 *LCOM* by Chidamber and Kemerer

The original *LCOM* is defined by Chidamber and Kemerer [6]. Their approach is based on whether the each pair of methods are sharing an attribute or not in a class. Those attribute-sharing relationships mean that different methods are accessing to the same attribute. Chidamber and Kemerer define *LCOM* as the number of method-pairs which are not sharing any attributes. *LCOM* value represents a weakness of functional connections among methods in a class. For the sake of convenience, we will call the metric "*LCOM1*."

Here is a formal definition of *LCOM1*.

Definition 1 (LCOM1):
Given a class C. Let M be the set of all methods in C, and A be the set of all attributes in C. *LCOM1* value of C is defined as $LCOM1(C) = |P|$, where

$$P = \Big\{ \{m_1, m_2\} \mid m_1, m_2 \in M, \nexists a \in A \text{ s.t. both of } m_1 \text{ and } m_2 \text{ access to } a \Big\}.$$

□

In the literature [7], Chidamber and Kemerer propose another definition of *LCOM*. We will call it "*LCOM2*." *LCOM2* takes into account the following two different factors : (1) the number of method-pairs which are "not sharing" any attributes, and (2) the number of method-pairs which are "sharing" an attribute. The above two factors have complementary meanings each other. The former is a weakening factor for cohesion, while the latter is a strengthening factor for one. *LCOM2* combines those factors.

The following is a formal definition of *LCOM2*.

Definition 2 (LCOM2):
Given a class C. Let M be the set of all methods in C, and A be the set of all attributes in C. *LCOM2* value of C is defined as follows :

$$LCOM2(C) = \begin{cases} |P| - |Q|, & \text{if } |P| > |Q| \text{ ;} \\ 0, & \text{otherwise.} \end{cases}$$

where P is the set described in Def. 1, and

$$Q = \Big\{ \{m_1, m_2\} \mid m_1, m_2 \in M, \\ \exists a \in A \text{ } s.t. \text{ both of } m_1 \text{ and } m_2 \text{ access to } a \Big\}.$$

□

2.1.2 *LCOM* by Hitz and Montazeri

Hitz and Montazeri propose a different *LCOM* [8] using the graph theory. Their approach is also based on the above attribute-sharing relationships, and those relationships are represented by an undirected graph. Now a class corresponds to an undirected graph, where each vertex represents each method, and each edge means each attribute-sharing relationship. Hitz and Montazeri define *LCOM* as the number of "connected components† [15]" in the undirected graph. We will call it "*LCOM3*." *LCOM3* value indicates the number of disjoint sets of methods in terms of the attribute-sharing relationship.

Here is a formal definition of *LCOM3*.

Definition 3 (LCOM3):
Given a class C. Let M be the set of all methods in C, and A be the set of all attributes in C. Consider an undirected graph $G_C = (V, E)$ with $V = M$ and

$$E = \Big\{ \{m_1, m_2\} \mid m_1, m_2 \in M, \\ \exists a \in A \text{ } s.t. \text{ both of } m_1 \text{ and } m_2 \text{ access to } a \Big\}.$$

LCOM3 value of C is defined as $LCOM3(C) = k(G_C)$, where $k(G_C)$ is the number of connected components in G_C. □

Hitz and Montazeri also discuss some effects of "access-methods" upon *LCOM* values [9]. Access-methods are methods to provide reading/writing-accesses to attributes. These methods are sometimes called "accessors" or "getters and setters." When two or more methods access to an attribute via "access-methods", we have no attribute-sharing relationships, because the above attribute-sharing relationships are based on "direct" accesses to attributes by methods. Hitz and Montazeri propose another *LCOM* taking into account such accesses via "access-methods." We will call it "*LCOM4*."

The following is a formal definition of *LCOM4*.

Definition 4 (LCOM4):
Given a class C. Let M be the set of all methods in C, and A be the set of all attributes in C. Consider an undirected graph $G_C = (V, E)$ with $V = M$ and

$$E = \Big\{ \{m_1, m_2\} \mid m_1, m_2 \in M, \\ (\exists a \in A \text{ } s.t. \text{ both of } m_1 \text{ and } m_2 \text{ access to } a) \\ \text{or } (m_1 \text{ invokes } m_2) \Big\}.$$

LCOM4 value of C is defined as $LCOM4(C) = k(G_C)$, where $k(G_C)$ is the number of connected components in G_C. □

2.1.3 *LCOM* by Henderson-Sellers

Henderson-Sellers proposes *LCOM* in a different way [10]. We will call it "*LCOM5*." *LCOM5* value is characterized by the following two cases: (i) if each method accesses to "all" attributes, the *LCOM5* value is 0; (ii) if each method accesses to "only one" attribute, the *LCOM5* value is 1. The case (i) means the most cohesive class, while the case (ii) means a less cohesive one. *LCOM5* represents cohesion-levels using these two cases as benchmarks.

Here is a formal definition of *LCOM5*.

Definition 5 (LCOM5):
Given a class C. Let M be the set of all methods in C, and A be the set of all attributes in C. For each $a \in A$, let $\mu(a)$ be the number of methods accessing to the attribute a.

LCOM5 value of C is defined as follows :

$$LCOM5(C) = \frac{1}{|M| - 1} \left(|M| - \frac{1}{|A|} \sum_{a \in A} \mu(a) \right).$$

□

2.2 Information Flow Based Cohesion (*ICH*)

Lee et al. propose a cohesion metric based on information flows between methods [11]. The approach focuses on the number of method-invocations weighted by the number of their parameters. Using the number of method parameters, the metric takes into account a strongness of link between methods. The metric is called "Information flow based Co-Hesion (*ICH*)."

The following is a formal definition of *ICH*.

Definition 6 (ICH):
Given a class C. Let M be the set of all methods in C. For $m_1, m_2 \in M$, let $\nu(m_1, m_2)$ be the number of invocations of m_2 by m_1, and $\pi(m_2)$ be the number of m_2's parameters.

†A connected component is a maximal sub graph in which all vertexes are reachable each other.

Table 1 Summary of existing class cohesion metrics.

metric	summary
LCOM1	number of method-pairs which do not share any attributes.
LCOM2	*LCOM1* − (the number of method-pairs which share an attribute).
LCOM3	number of method-sets in which the methods are connected in terms of attribute-sharing.
LCOM4	revised version of *LCOM3* considering indirect attribute-sharing via access-method.
LCOM5	density of accesses to attributes by methods.
ICH	number of method-invocations weighted by the number of method parameters.
TCC	density of attribute-sharing relationships among public methods.
LCC	density of relationships among public methods, where the relationships are in the transitive closure of attribute-sharing relationships.

ICH value of C is defined as follows :

$$ICH(C) = \sum_{m_1 \in M} \sum_{m_2 \in M} (1 + \pi(m_2)) \cdot \nu(m_1, m_2).$$

□

2.3 Tight Class Cohesion (*TCC*) and Loose Class Cohesion (*LCC*)

Bieman and Kang propose a couple of class cohesion metrics which are called "Tight Class Cohesion (*TCC*)" and "Loose Class Cohesion (*LCC*)" [12]. *TCC* represents a density of attribute-sharing relationships between public methods in a class. *LCC* represents a density of extended attribute-sharing relationships between public methods, where those extended relationships are constructed by the transitive closure of the above attribute-sharing relationships. These metrics have studied on usefulness as indicators of reusability for object classes [12].

We present definitions of *TCC* and *LCC*.

Definition 7 (TCC, LCC):
Given a class C. Let M_p be the set of all public methods in C, and A be the set of all attributes in C.

When a method m uses an attribute a in the method body, we say "m directly accesses to a." If a method m invokes another method m_1, m_1 invokes m_2, ..., m_{n-1} invokes m_n ($n \geq 1$) and m_n directly accesses to an attribute a, then we say "m indirectly accesses to a." Now consider the following two sets :

$$T = \Big\{ \{m, m'\} \mid m, m' \in M_p, \\ \exists a \in A \ s.t. \ m \text{ and } m' \\ \text{directly or indirectly access to } a \Big\},$$

and

$$L = \Big\{ \{m, m'\} \mid \exists \{m_i \in M_p\}_{i=1}^{k} \ \ s.t. \\ m = m_1, \ m' = m_k, \ k > 1, \\ \{m_j, m_{j+1}\} \in T \ (j = 1, \ldots, k-1) \Big\}.$$

TCC and *LCC* values of C are defined as follows :

$$TCC(C) = \frac{|T|}{\begin{pmatrix} |M_p| \\ 2 \end{pmatrix}}, \quad LCC(C) = \frac{|T| + |L|}{\begin{pmatrix} |M_p| \\ 2 \end{pmatrix}},$$

where $\begin{pmatrix} |M_p| \\ 2 \end{pmatrix} = |M_p|(|M_p| - 1)/2.$ □

3. A New Class Cohesion Metric

3.1 Motivation

We have introduced eight metrics : *LCOM1*~*5*, *ICH*, *TCC* and *LCC*. Table 1 shows a summary of them. *LCOM1*~*4* are based on the numbers of pairs/sets of methods connected by attribute-sharing relationships. *LCOM5* presents a density of accesses to attributes by methods. *ICH* shows the number of method-invocations weighted by the number of method parameters. *TCC* and *LCC* are based on densities of attribute-sharing relationships among methods. However, they do not consider "extents" of attribute-sharing relationships among methods, in other words, "sizes of cohesive-parts" in a class. Since the higher cohesive class would have the lager cohesive-parts, such aspect is also one of the cohesion aspects to be measured. This section proposes a cohesion metric focusing on cohesive-part size.

3.2 Preliminaries

Preliminary to the development of our metric, we define several underlying notions.

At first, we define a mathematical relation between methods through method-invocations.

Definition 8 (binary relation on methods):
Given a class. Let M be the set of all methods in the class. We define a binary relation S as follows :

$$S = \Big\{ (m_1, m_2) \mid m_1, m_2 \in M, \quad m_1 \text{ invokes } m_2 \Big\}.$$

Now we can obtain the reflective transitive closure S^* as follows :

$$S^* = \Bigg\{ (m_1, m_2) \,\Big|\, m_1, m_2 \in M, \\ (m_1 = m_2) \vee \Bigg(\bigvee_{n \geq 1} m_1 \ S^n m_2 \Bigg) \Bigg\},$$

where $S^n = S^{n-1} \circ S \ (n \geq 2)$, and $S^1 = S$; "$\circ$" indicates the composition of relations [16]. □

S^* represents directly or indirectly method-invocations.

Now we define a relationship between a method and an attribute.

Definition 9 (accesses to attributes by methods):
Given a class. Let M be the set of all methods, and A be the set of all attributes, in the class. For any $m \in M$, $a \in A$, we define a predicate ac as follows :

$$ac(m,a) \overset{def}{\Longleftrightarrow} \exists m' \in M \ s.t. \Big[\ (m \ S^* m') \wedge (m' \text{ accesses } a) \ \Big],$$

where an "access" means a direct access to an attribute by a method. □

The predicate ac considers not only direct accesses to attributes but also indirect accesses to ones via access-methods. Using the predicate, we introduce a graph model.

Definition 10 (association-graph):
Given a class. Let M be the set of all methods, and A be the set of all attributes, in the class. We define the *association-graph* as an undirected graph $G_a = (V, E)$, where $V = M$ and

$$E = \Big\{ \ \{m_1, m_2\} \mid m_1, m_2 \in M, \quad m_1 \neq m_2, \ \exists a \in A \ s.t. \ ac(m_1, a) \ \wedge \ ac(m_2, a) \ \Big\}. \tag{1}$$

□

When two or more methods access to one attribute, those methods seem to share the attribute. The association-graph G_a represents those attribute-sharing relationships between methods in a class. In G_a, if there is a path from a method to another method, those methods share an attribute directly or indirectly. Those relationships correspond to the reachabilities in the graph.

Definition 11 (set of reachable methods):
Given a class. Let M be the set of all methods, and A be the set of all attributes, in the class. Consider the association-graph $G_a = (V, E)$, where $V = M$ and E is described in Eq. (1).

For each method $m_i \in M$ ($i = 1, \ldots, |M|$), define the set of reachable methods, $R_a(m_i)$, as follows :

$$R_a(m_i) = \Big\{ \ m_j \mid \exists m_{k_1}, \ldots, m_{k_p} \in M \quad s.t. \ (m_i = m_{k_1}) \wedge (m_j = m_{k_p}) \wedge \{m_{k_t}, m_{k_{t+1}}\} \in E \ (t = 1, \ldots, p-1) \ \Big\}. \tag{2}$$

□

$R_a(m_i)$ is the set of all methods which are reachable by m_i in G_a. The methods belonging to $R_a(m_i)$ seem to be associated with m_i through their attribute-sharing relationships. In other words, those methods are in the "chain of links" where methods are directly or indirectly linked through attribute-sharing relationships. So the methods are functionally connected by the chain of links, and we should not decompose the set into two or more smaller sets. $R_a(m_i)$ seems to form a cohesive-part of the class including m_i, and $|R_a(m_i)|$ is involved in the extent of attribute-sharing relationships.

3.3 Definition

Using the association-graph and the sets of reachable methods, we propose a metric for measuring class cohesion, which is called "Association-Extent based Class Cohesion ($AECC$)."

Definition 12 (AECC):
Given a class C. Let M be the set of all methods, and A be the set of all attributes, in C. Consider the association-graph $G_a = (V, E)$ for C, where $V = M$ and E is in Eq. (1). For each method $m \in M$, let $R_a(m)$ be the set of all methods which are reachable by m in G_a (see Eq. (2)).

Define *Association-Extent based Class Cohesion* ($AECC$) value of C as follows :

$$AECC(C) = \begin{cases} \max\limits_{m \in M} \left[\dfrac{|R_a(m)|}{|M| - 1} \right], & (|M| > 1), \\ 0, & (|M| = 1). \end{cases} \tag{3}$$

□

3.4 Meanings of the Proposed Metric

For each method m, $|R_a(m)|/(|M|-1)$ denotes the percentage of methods to be reachable by m in the association-graph; "-1" in the numerator represents excluding m itself from the percentage calculation. $AECC$ is the maximum percentage for all methods in the class. Since $R_a(m)$ forms a cohesive-part in which all methods are directly or indirectly linked through attribute-sharing relationships, $|R_a(m)|/(|M|-1)$ represents the relative size of the cohesive-part including m, in other words, $|R_a(m)|/(|M|-1)$ denotes the extent of attribute-sharing relationships. That is, $AECC$ quantifies the maximum extent of attribute-sharing relationships among methods in the class. Since the higher cohesive class would have the lager cohesive-parts, $AECC$ represents an aspect of class cohesion.

Although $AECC$ is similar to TCC and LCC, $AECC$ is essentially different from them. TCC and LCC calculate "densities" of attribute-sharing relationships among methods. However, $AECC$ represents the maximum "extent" of attribute-sharing relationships. While the "density" is based on the number of attribute-sharing relationships, the "extent" corresponds to the connectivity among methods via attribute-sharing relationships.

3.5 Examples

We now present a simple example of calculating $AECC$ value. Consider a class shown in Fig. 1, who has seven methods $M = \{m_1, \ldots, m_7\}$ and four attributes $A = \{a_1, \ldots, a_4\}$. Fig. 2 shows its structure[†].

Here the following ten predicates are true :
$ac(m_1, a_1)$, $ac(m_1, a_2)$, $ac(m_2, a_1)$, $ac(m_3, a_2)$,

[†]For the sake of simplicity, we omit the existence of the super class "java.lang.Object."

```
public class Example1
{
    public int m1(){ return a1 − a2; }
    public void m2(int arg){ a1 = arg; }
    public void m3(int arg){ a2 = arg; }
    public void m4(int arg){ a3 = arg; m5(arg); }
    public void m5(int arg){ a4 = arg; }
    public int m6(){ return a3 + m7(); }
    public int m7(){ return a4; }
    private int a1, a2, a3, a4;
}
```

Fig. 1 An example of class written in Java.

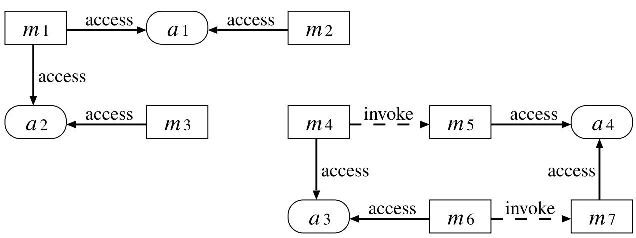

Fig. 2 An example of class model (1).

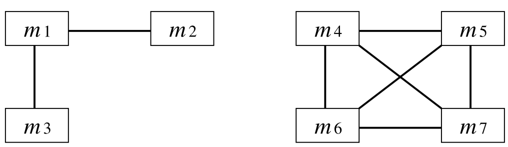

Fig. 3 association-graph for Fig. 2.

$ac(m_4, a_3)$, $ac(m_4, a_4)$, $ac(m_5, a_4)$, $ac(m_6, a_3)$, $ac(m_6, a_4)$ and $ac(m_7, a_4)$.
Then we can consider the association-graph G_a shown in Fig. 3. From Fig. 3, we obtain $|R_a(m_1)| = |R_a(m_2)| = |R_a(m_3)| = 2$ and $|R_a(m_4)| = |R_a(m_5)| = |R_a(m_6)| = |R_a(m_7)| = 3$. Thus *AECC* value of this class is calculated as follows :

$$
\begin{aligned}
AECC &= \max_{m \in M}\left[\frac{|R_a(m)|}{|M| - 1}\right] \\
&= \max\left[\frac{2}{7-1}, \frac{3}{7-1}\right] \\
&= 0.5.
\end{aligned}
$$

This result means that the maximum cohesive-part of the class makes up 50% of the whole.

We next consider another example, and compare with the existing eight metrics described in Sect. 2. Fig. 4 shows another example of class which is made by modifying the class of Fig. 2 where m_4 invokes m_2 instead of m_5. In Fig. 4, all methods are connected through attribute-accesses or method-invocations, while Fig. 2 has two separated groups $\{m_1, m_2, m_3\}$ and $\{m_4, m_5, m_6, m_7\}$. So the class shown in Fig. 4 would intuitively be more cohesive than one in Fig. 2. Table 2 shows metric values of the existing eight metrics (see Table 1) and *AECC*[†] for the classes shown in Fig. 2 and 4, and describes their changes in cohesion level[††] from Fig. 2 to Fig. 4. (For our calculation of *ICH*, we assume m_4 invokes m_2 once in the body.)

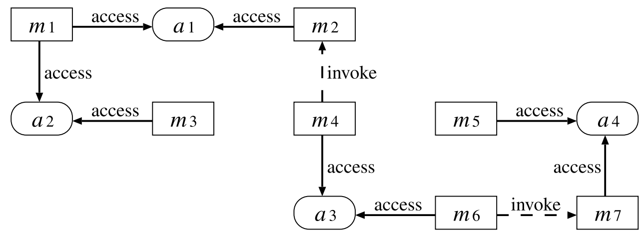

Fig. 4 An example of class model (2).

Table 2 Metric values of the classes in Fig. 2 and Fig. 4.

metric	Fig. 2	Fig. 4	Fig. 2 → Fig. 4
LCOM1	17	17	no change
LCOM2	13	13	no change
LCOM3	3	3	no change
LCOM4	2	1	up
LCOM5	0.833	0.833	no change
ICH	3	3	no change
TCC	0.429	0.333	down
LCC	0.429	1.0	up
AECC	0.5	1.0	up

In this case, *LCOM4*, *TCC*, *LCC* and *AECC* could be sensible to the difference between Fig. 2 and Fig. 4.

LCOM4 represents the number of cohesive-parts in the class, but it could not represent their sizes, so that *LCOM4* essentially differs from *AECC*.

TCC shows an evaluation which is counter to our intuition "Fig. 4 should have higher cohesion than Fig. 2." The reason *TCC* gives such unusual evaluation is that the total number of attribute-sharing relationships decreases from 9 (in Fig. 2) to 7 (in Fig. 4), i.e., the density of relationships decreases while the extent of relationships becomes wider through the change. It is an essentially different point between *TCC* and *AECC*. Although *LCC* is also based on the density of relationships, *LCC* uses the transitive closure of attribute-sharing relationships, so that it could evaluate the difference successfully.

LCC has a similar tendency to *AECC* in the above examples. In order to show a difference in viewpoint between *LCC* and *AECC*, we consider additional two examples shown in Fig. 5 and Fig. 6.

Table 3 shows *LCC* and *AECC* values in the cases of Fig. 5 and Fig. 6.

AECC represents a difference between Fig. 5 and Fig. 6, since the maximum extent of attribute-sharing relationships has changed through those examples : Fig. 5

[†]For the lack of space, we omit their calculation processes. Although we ignore the existence of the super class "java.lang.Object," we have no inconsistency in comparisons of those metrics.

[††]Notice that *LCOM1* ∼ *5* are "reverse" metrics in which the higher values mean the lower cohesion.

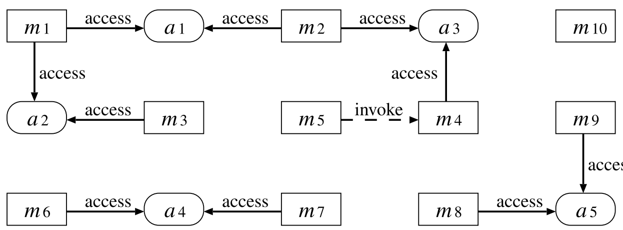

Fig. 5 An example of class model (3).

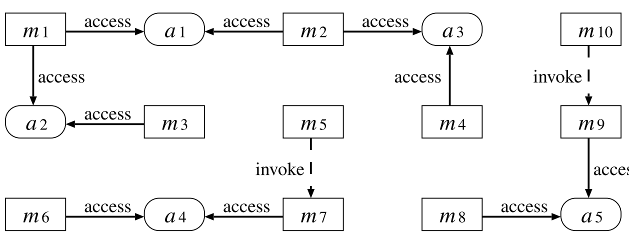

Fig. 6 An example of class model (4).

Table 3 *LCC* value and *AECC* value of the classes in Fig. 5 and Fig. 6.

metric	Fig. 5	Fig. 6	Fig. 5 → Fig. 6
LCC	0.267	0.267	no change
AECC	0.444	0.333	down

has $\{m_1, m_2, m_3, m_4, m_5\}$ as the largest method-set in which the methods are connected by some attribute-sharing relationships, but Fig. 6 has smaller set $\{m_1, m_2, m_3, m_4\}$ as the largest one. However *LCC* has no change through those two examples, because the total number of relationships (on the transitive closure of attribute-sharing relationships) stays constant between Fig. 5 and Fig. 6. (While m_5 has the relationships with m_1, m_2, m_3 and m_4 in Fig. 5, m_5 loses the relationships with them in Fig. 6. However m_5 gets the new relationships with m_6 and m_7; besides m_{10} obtains the new relationships with m_8 and m_9 in Fig. 6.) Thus the density of relationships has no change. The above comparison would show a major difference in viewpoint between *LCC* and *AECC*. Moreover *LCC* does not take into account "non-public" methods while *AECC* uses all methods in the evaluation; it is also a difference between *LCC* and *AECC*.

Although the above examples are just simple examples, they might show that *AECC* supports an aspect of cohesion which has not been considered by the existing eight metrics. We will evaluate it in both of qualitative and quantitative ways in the next section.

4. Evaluations

This section presents evaluations of our *AECC* in the both of qualitative and quantitative ways. The qualitative evaluation will describe that *AECC* satisfies mathematical necessary conditions of cohesion metrics, based on a mathematical framework. The quantitative evaluation will show that *AECC* gives the metric values without depending on the other existing metrics described in Sect. 2.

4.1 Qualitative Evaluation

4.1.1 Mathematical Framework

Briand, Morasca and Basili propose a mathematical framework (BMB framework) including some properties to be satisfied by several types of software metrics [17]. The supported types of metrics are "size," "length," "complexity," "coupling" and "cohesion." Note that BMB framework provides necessary conditions of software metrics, because the framework does not include the all properties to be satisfied by those metrics. We will use it for checking some mathematical necessary conditions of metrics.

In BMB framework, a software is represented by a graph model in which vertexes are corresponding to the components, and edges are corresponding to coupling relationships between the components. BMB framework suggests the following four properties to be satisfied by cohesion metrics. For the sake of convenience, we will write the cohesion of a class C as "$cohe(C)$."

Property 1 :
For any class C, $cohe(C) \in [0, max]$, where max is a positive constant number.

Property 2 :
Let $G = (V, E)$ be the graph model of a class C, where V and E are the vertex set and the edge set, respectively. Then $E = \phi \implies cohe(C) = 0$.

Property 3 :
Consider two classes C and C' whose models are $G = (V, E)$ and $G' = (V, E')$, respectively. Then $E \subseteq E' \implies cohe(C) \leq cohe(C')$.

Property 4 :
Consider two classes C_1 and C_2 whose models are $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, respectively. Let C_{12} be a class whose model is $G_{12} = (V_1 \cup V_2, E_1 \cup E_2)$, i.e., C_{12} is composed of C_1 and C_2. Then

$$\Big(\forall\{u, v\} \in E_1 \cup E_2\ [u \in V_1 \cap V_2 \Longleftrightarrow v \in V_1 \cap V_2]\Big) \implies \max[\ cohe(C_1),\ cohe(C_2)\ ] \geq cohe(C_{12})\ .$$

□

Property 1 represents a nonnegativity and a normalization. The nonnegativity is a basic property to be satisfied by any of mathematical measures [18]. The normalization brings us meaningful comparisons between different metrics.

Property 2 is also a basic property of mathematical measures : the measure of empty set is null. In our context, if there is no relationships among components in a class, then the class cohesion is null.

Property 3 corresponds to a monotonicity. When some relationships between components are added in a class,

Table 4 Metric values of Java classes.

class	*LCOM1*	*LCOM2*	*LCOM3*	*LCOM4*	*LCOM5*	*TCC*	*LCC*	*ICH*	*AECC*
Math	526	524	31	29	1.01	0.0383	0.0383	5	0.625
Throwable	28	28	8	3	1.10	0.143	0.0952	10	0.286
String	1107	888	31	17	0.971	0.194	0.122	54	0.725
SecurityManager	1081	1081	47	10	1.01	0	0	82	0
ClassLoader	1307	1288	44	14	1.01	0.0216	0.0173	87	0.157
Package	204	198	16	10	0.995	0.0221	0	14	0.1
StringBuffer	640	539	22	7	0.908	0.579	0.0254	66	0.789
System	463	430	24	16	0.974	0.0260	0	21	0.129
Thread	887	871	33	11	1.01	0.0189	0.0170	50	0.310
ThreadGroup	415	334	13	5	0.958	0.108	0.171	36	0.871
BasicPermission	21	21	7	5	1.08	0	0	10	0.167
Arrays	4641	4626	92	19	0.979	0.00328	0	674	0.0313
Integer	269	238	16	9	0.984	0.0217	0	22	0.125
Date	507	486	24	5	0.994	0.296	0.257	50	0.656
Component	29493	28851	155	72	0.998	0.0928	0.0303	339	0.837
File	707	424	18	6	0.959	0.206	0.165	24	0.773
FileDescriptor	3	3	3	3	1.38	0	0	0	0
InputStream	28	28	8	7	1.07	0	0	10	0
Container	3834	3752	60	30	0.990	0.0648	0.0244	179	0.455
Writer	5	4	3	3	1.11	0	0	16	0
StringWriter	21	6	4	2	0.688	0	0	5	0
Panel	6	6	4	4	1.22	0	0	6	0
Applet	304	283	20	12	0.933	0.277	0.166	27	0.56
BigInteger	3791	3577	53	13	0.994	0.084	0.0559	267	0.697
Socket	63	0	4	4	0.568	0	0	0	0
ServerSocket	15	0	3	2	0.625	0.0714	0.0714	2	0.25
DatagramPacket	46	37	4	4	0.9	0.0444	0.0444	2	0.2
FileOutputStream	45	45	10	7	1	0	0	10	0
EventObject	0	0	1	1	0	0	0	2	0
Collections	22376	20442	115	97	0.993	0.000209	0.000209	58	0.0227

cohesion-level dose not go down.

Property 4 considers a situation combining a class with another class, where they have no common component. Such situation means that two classes cohabit, but they have no relationship each other. Property 4 says that such combined class does not have a greater cohesion than the maximum of the original classes' cohesion.

4.1.2 Results

AECC satisfies the above four properties (see Appendix for their proofs), and be a metric holding necessary conditions of cohesion metric. We will consider *AECC* to be one of reasonable class cohesion metrics, and compare it with the other existing metrics in the following section.

4.2 Quantitative Evaluation

We have seen *AECC* holds the above necessary conditions of cohesion metric. We next present some measurement data of practical object classes, and evaluate *AECC* using those data, especially we will show that *AECC* measures classes without depending on the other existing metrics, in other words, it is not redundant metric.

Table 4 presents metric values of Java classes which are selected from "Sun J2SE SDK version 1.3.1" randomly.

Unfortunately any sufficient conditions of cohesion metric have never been found nor proposed. Since we have no sufficient condition, it is difficult to present an objective and reliable discussion on the validity of *AECC* as a cohesion metric, even if we use many experimental data. (While highly reusable classes such as *String*, *StringBuffer* and so on, seem to have relative high value of *AECC*, they can not be any grounds for the validity as cohesion metric; Rather they would be materials for investigating a usefulness of *AECC*.) So we have checked *AECC* by mathematical properties of cohesion metric in the above. As a quantitative evaluation of *AECC*, we analyze dependencies between metrics using experimental data shown in Table 4, and we show *AECC* is not redundant metric. For example, we now see three classes shown in the head of Table 4. Although the model of *AECC* is similar to the models of *TCC* and *LCC*, those experimental data show a different tendency between *AECC* and *TCC*/*LCC* (e.g., see Fig. 7). In order to show such actual differences in tendency between metrics, we use the following correlation analysis.

4.2.1 Correlation Coefficient

When some different software metrics measure a software attribute, they have to capture different aspects of the software attribute independently. In other words, metrics should not depend on each other for providing their measurements. If there is a strong correlation [19] between two metrics, one of them is a redundant metric. We can explore such relationships between metrics by calculating their correlation coef-

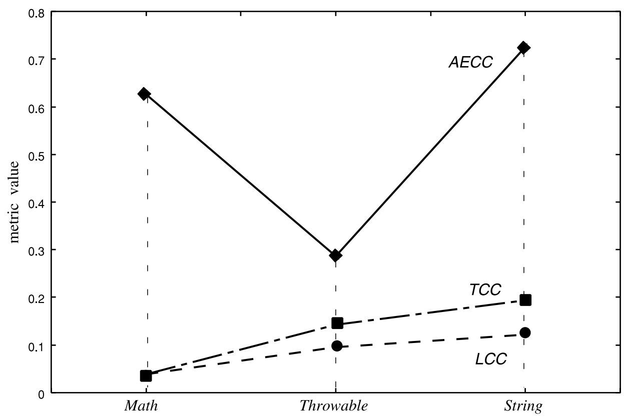

Fig. 7 A comparison of *AECC* and *TCC*/*LCC*.

ficients.

The following is a brief description of calculating correlation coefficients for verifying metrics.

Given two metrics and N sample data(software). Measure each of sample data using those metrics, then obtain 2-dimensional vector $\boldsymbol{x}_i = (x_{i1}, x_{i2})$ for each data ($i = 1, \ldots, N$), where x_{i1} and x_{i2} are the metric values for the i-th sample data, respectively. Now the correlation coefficient r is calculated as follows :

$$r = \frac{\sum_{i=1}^{N}(x_{i1} - \overline{x}_1)(x_{i2} - \overline{x}_2)}{\sqrt{\sum_{i=1}^{N}(x_{i1} - \overline{x}_1)^2 \sum_{i=1}^{N}(x_{i2} - \overline{x}_2)^2}} \quad ,$$

where $\overline{x}_1 = (\sum_{i=1}^{N} x_{i1})/N$ and $\overline{x}_2 = (\sum_{i=1}^{N} x_{i2})/N$.

The higher value of $|r|$ means the stronger correlation between two metrics. Now prepare a threshold value τ for $|r|$. If $|r| \geq \tau$, we will consider those two metrics are dependent on each other, and one of them is a redundant metric.

4.2.2 Experiments

We perform the following experiments in order to show that *AECC* measures cohesion without depending on the other existing metrics (see Sect. 2), in other words, *AECC* is not redundant metric.

Prepare a set of sample data (Java classes). Measure cohesion using the above eight metrics and *AECC*. For each pair of *AECC* and the others, (i.e., for the following eight pairs: {*AECC*, *LCOM1*}, {*AECC*, *LCOM2*}, {*AECC*, *LCOM3*}, {*AECC*, *LCOM4*}, {*AECC*, *LCOM5*}, {*AECC*, *ICH*}, {*AECC*, *TCC*}, and {*AECC*, *LCC*},) calculate their correlation coefficients.

We now use $\tau = 0.8$ [19]. If there is a correlation coefficient is grater than or equal to τ, we will regard *AECC* as a redundant metric. Otherwise, we will consider that *AECC* can provide its measurements without depending on any of the other existing metrics.

Now our set of sample data (Java classes) is shown in Table 4.

4.2.3 Results

Tables 5 shows the correlation coefficients calculated in the above experiment[†]. From this table, we have no metric whose correlation coefficient with *AECC* is grater than or equal to $\tau(= 0.8)$, i.e., no metric who has a strong correlation with *AECC* :

Table 5 Correlation coefficients with *AECC*.

metrics	correlation coefficients : r
LCOM1	0.202744
LCOM2	0.207895
LCOM3	0.259502
LCOM4	0.152871
LCOM5	0.085906
ICH	0.147297
TCC	0.695496
LCC	0.697812

- *LCOM1*∼*LCOM5* and *ICH* have some low correlation coefficients ($|r| = 0.085906 \sim 0.259502$).
- *TCC* and *LCC* have some middle-level correlations ($|r| = 0.695496, 0.697812$), but not strong correlations. The reason they have such correlations is because their underlying model is similar to *AECC*'s model : The models of *TCC* and *LCC* focus on "densities" of attribute-sharing relationships while *AECC*'s model focuses on "extents" of those relationships.

Thus *AECC* is not redundant metric, and will be an important one. It measures an aspect of cohesion which is not supported by the other existing eight metrics. *AECC* can collaborate with those existing metrics in measuring class cohesion, and will contribute to more accurate measurement.

5. Conclusion and Future Work

A class cohesion metric, "Association-Extent based Class Cohesion (*AECC*)," has been proposed in this paper. *AECC* measures class cohesion using the maximum size of cohesive-parts, which is an aspect has not been supported by the other existing metrics. *AECC* has been evaluated in both of qualitative and quantitative ways : it has been showed that

- *AECC* satisfies mathematical conditions of cohesion metrics described in *BMB* framework,
- *AECC* presents cohesion values without depending on any of the other existing metrics : *LCOM1*∼*5*, *ICH*, *TCC* and *LCC*.

Therefore, *AECC* is a reasonable class cohesion metric, and not redundant metric. It can collaborate with the other existing metrics in measuring class cohesion, and will contribute to more accurate measurement.

Our future works include investigations into
(1) practical usefulness of *AECC* and other metrics, such as predictors of software maintenance cost using class cohesion metrics,
(2) relationships between reusability of software components and their cohesion, and so on.

[†]For the lack of space, we omit the metric values for each of sample data.

Acknowledgement

The authors would like to thank the anonymous reviewers for their thoughtful comments and helpful suggestions. This research has been supported by the Kayamori Foundation of Information Science Advancement.

References

[1] G.J. Myers, Software Reliability-Principles and Practices, John Wiley & Sons, New Jersey, 1976.
[2] M. Page-Jones, The Practical Guide to Structured Systems, 2nd ed., Prentice Hall, New Jersey, 1988.
[3] E. Yourdon and L. Constantine, Structured Design, Englewood Cliffs, Prentice Hall, New Jersey, 1979.
[4] N.E. Fenton and S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, PWS Publishing, Boston, 1997.
[5] L.C. Briand, J.W. Daly, and J. Wuest, "A unified framework for cohesion measurement in object-oriented systems," Empirical Software Eng., vol.3, no.1, pp.65–117, 1998.
[6] S.R. Chidamber and C.F. Kemerer, "Towards a metrics suite for object oriented design," Proc. 6th ACM Conf. Object Oriented Programming, Syst., Lang. and Appicat. (OOPSLA), pp.1970–221, 1991.
[7] S.R. Chidamber and C.F. Kemerer, "A metrics suite for object oriented design," IEEE Trans. Softw. Eng., vol.20, no.6, pp.476–493, June 1994.
[8] M. Hitz and B. Montazeri, "Chidamber and Kemerer's metrics suite: A measurement theory perspective," IEEE Trans. Softw. Eng., vol.22, no.4, pp.267–271, April 1996.
[9] M. Hitz and B. Montazeri, "Measuring coupling and cohesion in object-oriented systems," Proc. Int. Symp. on Applied Corporate Computing (ISACC'95), pp.25–27, Oct. 1995.
[10] B. Henderson-Sellers, Softare Metrics, Prentice Hall, Hemel Hempstead, U.K., 1996.
[11] Y.-S. Lee, B.-S. Liang, S.-F. Wu, and F.-J. Wang, "Measuring the coupling and cohesion of an object-oriented program based on information flow," Proc. Int. Conf. on Software Quality, pp.81–90, 1995.
[12] J.M. Bieman and B.-K. Kang, "Cohesion and reuse in an object-oriented system," Proc. ACM Symp. on Software Reusability, pp.259–262, 1995.
[13] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," J. Systems and Software, vol.23, no.2, pp.111–122, 1993.
[14] V.R. Basili, L.C. Briand, and W.L. Melo, "A validation of object-oriented design metrics as quality indicators," IEEE Trans. Softw. Eng., vol.22, no.10, pp.751–761, Oct. 1996.
[15] R.J. Wilson, Introduction to Graph Theory 4th ed., Addison-Wesley, Boston, 1996.
[16] D.F. Stanat and D.F. McAllister, Discrete mathematics in Computer Science, Prentice-Hall, New Jersey, 1977.
[17] L.C. Briand, S. Morasca, and V.R. Basili, "Property-based software engineering measurement," IEEE Trans. Softw. Eng., vol.22, no.1, pp.68–86, Jan. 1996.
[18] M. Adams and V. Guillemin, Measure Theory and Probability, Springer Verlag, Heidelberg, 1996.
[19] P.G. Hoel, Elementary Statistics, John Wiley & Sons, New Jersey, 1976.
[20] H. Aman, K. Yamasaki, H. Yamada, and M.T. Noda, "A proposal of class cohesion metrics using sizes of cohesive parts," in Knowledge-based Software Engineering, ed. T. Welzer, S. Yamamoto, and I. Rozman, pp.102–107, IOS Press, Amsterdam, 2002.

Appendix: Proofs

The followings are the proofs of that *AECC* satisfies the four properties of cohesion metrics described in Sect. 4.1.1 [20]. In these proofs, we will consider the association-graph to be the graph model of a class.

Proof for Property 1 :

Given a class C. Let M be the set of all methods in C.

(i) Case $|M| = 1$:

From Eq. (3), we obtain $AECC(C) = 0$.

(ii) Case $|M| > 1$:

For each $m \in M$, let $R_a(m)$ be the set of methods reachable by m (see Eq. (2)). From the definition, we have $0 \leq |R_a(m)| \leq |M| - 1$, for each $m \in M$. Thus, from Eq. (3), we obtain $0 \leq AECC(C) \leq 1$.

Therefore, we can get $AECC(C) \in [0, 1]$, for any class C. □

Proof for Property 2 :

Given a class C. Let M be the set of all methods in C, and $G_a = (V, E)$ be the association-graph of C.

(i) Case $|M| = 1$:

From Eq. (3), we obtain $AECC(C) = 0$.

(ii) Case $|M| > 1$:

We assume $E = \phi$. For each $m \in M$, let $R_a(m)$ be the set of methods reachable by m (see Eq. (2)). Since $E = \phi$, we have $|R_a(m)| = 0$ for each $m \in M$. Thus, we obtain $AECC(C) = 0$.

Therefore, we can get $AECC(C) = 0$ for any class C such that $E = \phi$. □

Proof for Property 3 :

Given a class C. Let M be the set of all methods in C.

(i) Case $|M| = 1$:

Since we have only one vertex (method) in the graph, we could not add any edges to the association-graph of C, and each vertex is not allowed having a self-loop (an edge from a vertex to the same vertex). Thus, the class C', described in the property 3, must be the same as C. Therefore, we can obtain $AECC(C) = AECC(C')$.

(ii) Case $|M| > 1$:

Let $G_a(C) = (V, E)$ be the association-graph of C. Now we can consider a class C' whose association-graph is $G'_a(C') = (V, E')$ where $E \subseteq E'$. For each $m \in M$, let $R_a(m)$ and $R'_a(m)$ be the sets of reachable methods by m in G_a and G'_a, respectively (see Eq. (2)).

Since $E \subseteq E'$, we obtain $R_a(m) \subseteq R'_a(m)$, for each $m \in M$. Thus, we have $|R_a(m)| \leq |R'_a(m)|$, and we get

$$\max_{m \in M}\Big[\ |R_a(m)|\ \Big] \leq \max_{m \in M}\Big[\ |R'_a(m)|\ \Big],$$

i.e., $AECC(C) \leq AECC(C')$, for each $m \in M$.

Therefore, we can get $AECC(C) \leq AECC(C')$, for all pairs of classes such that $E \subseteq E'$. □

Proof for Property 4 :

Given a pair of classes C_1 and C_2, and consider a class C_{12} composed of C_1 and C_2. Let M_1, M_2 and M_{12} ($= M_1 \cup M_2$) be the sets of all methods in C_1, C_2 and C_{12}, respectively. Let $G_{a_1} = (V_1, E_1)$, $G_{a_2} = (V_2, E_2)$ and $G_{a_{12}} = (V_1 \cup V_2, E_1 \cup E_2)$ be the association-graphs of C_1, C_2 and C_{12}, respectively. For each $m_1 \in M_1$, let $R_{a_1}(m_1)$ be the set of reachable methods by m_1 in G_{a_1} (see Eq. (2)). Similarly, let $R_{a_2}(m_2)$ and $R_{a_{12}}(m_{12})$ be the sets of reachable methods by $m_2 \in M_2$ and $m_{12} \in M_{12}$, respectively.

(i) Case $|M_1| = |M_2| = 1$:

Since $M_{12} = M_1 \cup M_2$, we have $|M_{12}| = 1$. From Eq. (3), we obtain $AECC(C_1) = AECC(C_2) = AECC(C_{12}) = 0$. Thus, we get

$$\max[AECC(C_1), AECC(C_2)] = AECC(C_{12}).$$

(ii) Case $|M_1| > 1$ or $|M_2| > 1$:

This case could be $|M_1| + |M_2| \geq |M_{12}|$, and we put $|M_1| + |M_2| - |M_{12}| = p \ (\geq 0)$.

Let $m_{1,i}$ ($i = 1, \ldots, |M_1|$), $m_{2,j}$ ($j = 1, \ldots, |M_2|$) and $m_{12,k}$ ($k = 1, \ldots, |M_{12}|$) be the methods in C_1, C_2 and C_{12}, respectively. Now we put the relations among $\{m_{1,i}\}$, $\{m_{2,j}\}$ and $\{m_{12,k}\}$ as follows, without loss of generality :

$$m_{12,k} = \begin{cases} m_{1,k}\,, & (k = 1, \ldots, |M_1| - p), \\ m_{2,k-|M_1|+p}\,, & \text{(otherwise)}, \end{cases}$$

where $m_{2,k-|M_1|+p} = m_{1,k}$ ($k = |M_1| - p, \ldots, |M_1|$).

From the assumption of the property 4, we have $\forall\{u, v\} \in E_1 \cup E_2\ [u \in V_1 \cap V_2 \Longleftrightarrow v \in V_1 \cap V_2]$. That is, $G_{a_{12}}$ has no path from a method in C_1 to one in C_2, vice versa. Thus we obtain the following equation :

$$\begin{aligned} &R_{a_{12}}(m_{12,k}) \\ &\quad = \begin{cases} R_{a_1}(m_{1,k}), & (k = 1, \ldots, |M_1| - p), \\ R_{a_2}(m_{2,k-|M_1|+p}), & \text{(otherwise)}. \end{cases} \end{aligned}$$

Since $|M_1|, |M_2| \leq |M_{12}|$, we obtain

$$\frac{|R_{a_{12}}(m_{12,k})|}{|M_{12}| - 1} \leq \frac{|R_{a_1}(m_{1,k})|}{|M_1| - 1}, \quad (k = 1, \ldots, |M_1|),$$

and

$$\frac{|R_{a_{12}}(m_{12,k})|}{|M_{12}| - 1} \leq \frac{|R_{a_2}(m_{2,j})|}{|M_2| - 1}, \quad (k = |M_1| - p + 1, \ldots, |M_{12}|;\ j = k - |M_1| + p).$$

Thereby we get

$$\max_{k=1,\ldots,|M_1|}\left[\frac{|R_{a_{12}}(m_{12,k})|}{|M_{12}| - 1}\right] \leq \max_{k=1,\ldots,|M_1|}\left[\frac{|R_{a_1}(m_{1,k})|}{|M_1| - 1}\right] = AECC(C_1),$$

and

$$\max_{k=|M_1|-p+1,\ldots,|M_{12}|}\left[\frac{|R_{a_{12}}(m_{12,k})|}{|M_{12}| - 1}\right] \leq \max_{\substack{k=|M_1|-p+1,\ldots,|M_{12}| \\ (j=1,\ldots,|M_2|)}}\left[\frac{|R_{a_2}(m_{2,j})|}{|M_2| - 1}\right] = AECC(C_2).$$

Therefore, we get

$$\begin{aligned} &AECC(C_{12}) \\ &\quad = \max_{k=1,\ldots,|M_{12}|}\left[\frac{|R_{a_{12}}(m_{12,k})|}{|M_{12}| - 1}\right] \\ &\quad = \max\left\{\max_{k=1,\ldots,|M_1|}\left[\frac{|R_{a_{12}}(m_{12,k})|}{|M_{12}| - 1}\right],\right. \\ &\qquad\qquad \left.\max_{k=|M_1|-p+1,\ldots,|M_{12}|}\left[\frac{|R_{a_{12}}(m_{12,k})|}{|M_{12}| - 1}\right]\right\} \\ &\quad \leq \max\{\ AECC(C_1),\ AECC(C_2)\ \}, \end{aligned}$$

for all classes such that $\forall\{u, v\} \in E_1 \cup E_2\ [u \in V_1 \cap V_2 \Longleftrightarrow v \in V_1 \cap V_2]$. □

Hirohisa Aman received the Dr. degree in Engineering from Kyushu Institute of Technology, Kitakyushu, Japan, in 2001. Since 2001, he has been with Ehime University. He is currently a research assistant of the Faculty of Engineering, Ehime University. His primary research interest is in software maintenance cost estimation using metrics. He is a member of Information Processing Society of Japan (IPSJ), Japan Society for Software Science and Technology (JSSST), the Institute of Electrical and Electronics Engineering (IEEE), and Japan Society for Fuzzy Theory and Intelligent Informatics (SOFT).

Kenji Yamasaki received the Masters degree in Computer Science from Ehime University, Matsuyama, Japan, in 2004. He is currently with Fujitsu System Solutions Ltd. He is a member of IPSJ.

Hiroyuki Yamada received the Dr. degree in Engineering from Osaka University, Suita, Japan, in 1988. Since 1988, he has been with Ehime University. He is currently an associate professor of the Faculty of Engineering, Ehime University. His primary research interests are in the field of knowledge-based software engineering, especially, software requirements acquisition and software evolution. He is a member of IPSJ, JSSST, IEEE, Association for Computing Machinery (ACM), and Japanese Society for Artificial Intelligence (JSAI).

Matu-Tarow Noda received the Dr. of Science from Osaka City University, Osaka, Japan, in 1969. Since 1971, he has been with Ehime University. He is currently a professor of the Faculty of Engineering, Ehime University. He also is a director of Center of Information Technology, Ehime University(CITE). His primary research interests are in algorithm developments of symbolic-numeric combined computation, and Internet security in mobile environments. He received the Best Paper Award from IPSJ in 1992. He is the president of Japan Society for Symbolic and Algebraic Computation (JSSAC). He is a member of IPSJ, ACM, Japan Society for Industrial and Applied Mathematics (JSIAM), and International Association for Mathematics and Computers in Simulation (IMACS).