A Cohesion Measure for Classes in Object-Oriented Systems

Heung Seok Chae and Yong Rae Kwon
Department of Computer Science
Korea Advanced Institute of Science and Technology
373-1, Kusong-dong, Yusong-gu, Taejon 305-701, Korea
{hschae,yrkwon}@salmosa.kaist.ac.kr

ABSTRACT
Classes are the fundamental concepts in the object-oriented paradigm. They are the basic units of object-oriented programs, and serve as the units of encapsulation, which promotes the modifiability and the reusability of them. In order to take a full advantage of the desirable features provided by classes, such as data abstraction and encapsulation, classes should be designed to have a good quality. Because object-oriented systems are developed by heavily reusing the existing classes, the classes of poor quality can be a serious obstacle to the development of systems.

This paper defines a new cohesion measure for assessing the quality of classes. Our approach is based on the observations on the salient natures of classes which have not been considered in the previous approaches. A *Most Cohesive Component(MCC)* is introduced as the most cohesive form of a class. We believe that the cohesion of a class depends on the connectivity of itself and its constituent components. We propose the connectivity factor to indicate the degree of the connectivity among the members of a class, and the structure factor to take into account the cohesiveness of its constituent components. Consequently, the cohesion of a class is defined as the product of the connectivity factor and the structure factor. This cohesion measure indicates how closely a class approaches MCC; the closely a class approaches MCC, the greater cohesion the class has.

1 Introduction

Object-orientation aims to model the real world as closely to a user's perspective as possible. Classes play an essential role in the object-oriented development. Entities in an application domain are captured as classes, and applications are built with the objects which are instantiated from them. Classes serve as a unit of encapsulation; that is, instances of a class can be manipulated only through the interface defined in the class. Therefore, the internal representation of classes can be changed without affecting any clients as long as the new representation conforms to the interface. In other words, compatible changes can be made safely on classes, which facilitates program evolution and maintenance[9].

In order to take a full advantage of the desirable features provided by classes, such as data abstraction and encapsulation, classes should be designed to have a good quality. Otherwise, classes of poor quality can be a serious obstacle to the development of systems because object-oriented systems are often developed by reusing the existing classes.

Cohesion which originated from structured design refers to the degree of connectivity among the elements of a single module(and for object-oriented design, a single class or object)[2]. Cohesion metrics is being used as a metric which characterizes the quality of a module and developers are using it as a design guideline; they attempt to maximize the cohesion of the modules.

The notion of cohesion can also be applied to a class. Many researches have defined the cohesion for classes or abstract data types(ADTs)[3, 4, 5, 6, 7, 8]. However, the previous approaches did not consider a few important features of classes; they took no notice of the nature of the special methods which inherently interact with only a part of the instance variables, and they regarded the number of the interactions among the members of a class as the only dominant factor in determining a cohesion.

We propose a refined cohesion measure based on our observations on a few salient features of the class which have not been considered in the previous approaches. We note that the special methods which inherently interact with only a part of the instance variables would not reduce the cohesion of a class. In this point of view, the *most cohesive component(MCC)* is introduced as the most cohesive form of a class where the special methods are excluded. We also note that the cohesion of a class depends on the connectivity of not only itself and but also the constituent components. We propose the connectivity factor to indicate the degree of the connectivity among the members of a class, and the structure factor to take into account the cohesiveness of its constituent components. Consequently, the cohesion of a class is defined as the product of the connectivity factor and the structure factor. The cohesion measure is designed to indicate how closely a class approaches MCC; the greater cohesion a class has, the closely the class approaches MCC.

The remainder of this paper is organized as follows. Section 2 provides an overview of current researches on a cohesion measure for classes and describes their weakness. Section 3 proposes a new cohesion measure, and finally, conclusion and future works are given in Section 4.

2 Related Works

Most of researches which define a cohesion measure regarded the interactions between the methods and the instance variables as the dominant factors in determining the cohesiveness of a class.

Briand[3] proposed Ratio of Cohesive Interactions(RCI) as a cohesion metric. RCI is defined as the ratio of the number of the actual interactions to the number of all possible interactions.

Chidamber[5, 6] proposed Lack of Cohesion in Methods(LCOM) as a measure of the relative disparateness of methods in a class. LCOM is defined as the difference between the number of method pairs without shared instance variables and the number of method pairs with shared instance variables. LCOM is an inverse measure for cohesion; that is, the lower value of LCOM means higher cohesiveness.

Li[8] defined LCOM as the number of disjoint sets of methods which share at least one instance variable, and Hitz[7] restated the Li's definition of LCOM in graph-theoretic terms and proposed a linear mapping to discriminate among the cases with LCOM $= 1$.

Previously, we[4] proposed abstraction as the most cohesive form of a class, and defined cohesion of a class as the extent to which a class approaches abstraction. However, without a quantitative measure for cohesion, the cohesiveness of an individual class could not be determined.

The previous approaches have the following problems which lead to the inconsistent cases with our intuition.

- Previous works[3, 5, 6] do not take into account some characteristics of classes. Special methods such as accessor methods, delegation methods, constructor, and destructor are designed to show a specific behavior, and they inherently interact with only a part of the instance variables in a class.

  An accessor method is designed to retrieve or update an instance variable. Consequently, it interacts with only one instance variable. In most of the previous approaches, these accessor methods considerably reduce the cohesion. The reference to one instance variable causes LCOM to increase because it increases the number of methods pairs which do not share any instance variables. While the number of possible interactions will increase by the number of the instance variables, the number of actual interactions increases only by one; thus, RCI decreases sizably.

  However, we believe that these accessor methods should not reduce the cohesion, because they inherently reference only a part of the instance variables in a class; that is, the reference of one instance variable is sufficient for them to complete their behavior(i.e., retrieving or updating the instance variable). In addition, encapsulation does not allow instance variables to be accessed directly. Therefore, it is inevitable to use such accessor methods in order to provide an access path for the instance variables. Even some object-oriented languages such as Trellis/Owl generate them automatically.

  A delegation method achieves its behavior by delegating a message to another class(i.e., invoking a method in another class) and generally has one interaction with the class. For example, when a method *slowDown* in class *Car* is invoked to decrease the speed of a car, the behavior of the method *slowDown* can be achieved by simply invoking a method *activate* to class *Brake*, a component class of class *Car*, without performing any other extra operation.

  The constructor and the destructor in C++ may not inherently access all of the instance variables. The constructor and the destructor in a class may access the only essential instance variables which are required to initialize and deinitialize its instances, respectively.

- The previous work[3] considers only the number of interactions, not the pattern of the interactions among the members of a class. This results in the cases which are not consistent with our intuition about cohesion.

  For example, Figures 1 (a) and (b) show the interactions among the members of classes A and B, respectively. Both of them have the same number of possible interactions(i.e., 12) and actual interactions(i.e., 6). According to the definition of RCI in [3], they are said to have the same cohesion value(i.e., $\frac{6}{12}$). However, these two classes show the distinct patterns of the interactions; the interaction graph of class A is connected, but that of class B is disjoint. From the definition of cohesion, relatedness among the members of a class, class A should be considered more cohesive than class B. This discrepancy originates from the fact that RCI depends only the number of the interactions and does not consider their pattern.

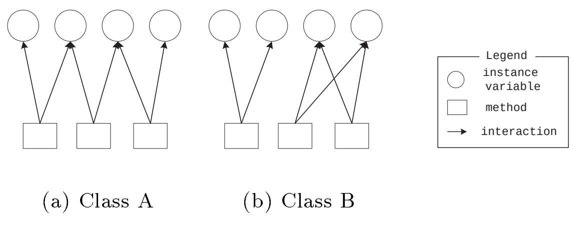


Figure 1: Example 1

- They[4, 5, 6, 8] do not provide a normalized measure for comparing different classes.

  The original version of LCOM[5, 6] depends on the number of methods(actually method pairs) and its maximum value varies with the number of method pairs. LCOM cannot be used to determine which class is more cohesive because it does not offer an absolute measure. Actually, this problem was manifest from the experiment performed by Basili[1], where LCOMs of many classes are set to be 0, although very different cohesions are expected. For example, consider Figures 2[1] (a) and (b) which show the interaction patterns of two classes A and B. The LCOM of class A(i.e., 1-0=1) is equal to that of class B(i.e., 3-2=1). Intuitively, class B should be more cohesive than class A because the members of class B are connected, but those of class A are not.

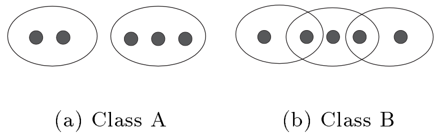


Figure 2: Two classes with the same LCOMs

  Although the abnormal behavior has been corrected in the new version of LCOM[8], it still does not discriminate the cohesiveness of classes which have different interaction patterns. According to the new definition of LCOM, as long as members of a class are connected by the interactions, LCOM = 1. Figures 3[2] shows two extreme cases with LCOM = 1. Figure 3 (a) shows the least cohesive class and Figure 3 (b) represents the most cohesive class. Although LCOM = 1 for both cases, intuitively class B should be more cohesive than class A because class B has tighter interactions than class A.

  Hitz[7] dealt with this problem by proposing a linear mapping function which defines a measure of the deviation of a class from the least cohesive case. However, the linear function also depends only on the number of connectivity among methods, not its pattern, which can produce an abnormal case mentioned above.

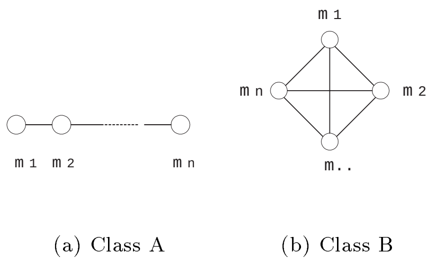


Figure 3: Two extreme cases with LCOM = 1

3 Cohesion Metrics for Classes

In this section, we propose a cohesion metric for classes. Our metric is defined based on the analysis of the patterns of the interactions among the members of a class. As a preparation for defining the metric, we present a few basic definitions.

3.1 Basic Definitions

Definition 3.1 *A method in class C is* **special***, if it is an accessor method, a delegation method, a constructor, or a destructor in class C. A method is called* **normal** *if it is not special.*

For example, class *Stack*(see Figure 4) has three special methods: *Stack*, ∼*Stack* and *is Empty*. The methods *Stack* and ∼*Stack* are the constructor and the destructor, respectively and *isEmpty* is a delegation method which checks the emptiness of the stack by sending a message *isZero* to *top*, an object of class *Natural*.

Special methods inherently interact with a limited number of instance variables of the class to achieve their behavior completely. For example, the destructor ∼*Stack* references only *items* to free the allocated space, and the delegation method *isEmpty* interacts only with *top*. We believe that such limited interaction of special methods should not reduce the cohesiveness of class very much. Therefore, we exclude the special methods in evaluating the cohesion metric.

A reference graph is introduced to represent the interactions with the normal methods in a class. Special

[1]The diagramatic notation is quoted from [7], where each Venn diagram represents a method by the set of instance variables it references.

[2]This graph notation is quoted from [7], where two methods which share at least one instance variable are connected.

```
class Stack {
  private:
    int *items ;
    Natural top ;
  public:
    Stack() {
      top = 0 ; items = new int[MAX] ;
    }
    ~Stack() { delete [] items ; }
    int pop() {
      if ( top > 0 )
        return items[top--];
    }
    void push(int n) {
      if ( top < MAX-1 )
        items[top++] = n ;
    }
    int isEmpty() {
      return top.isZero() ;
    }
} ;
```

Figure 4: Class Stack

methods are excluded because they have no influence on the cohesion.

Definition 3.2 *A* **reference graph** *for class C, $G_r(C)$, is a directed graph $G = (N, A)$ with*

- *$N = N_v \cup N_m$, where N_v and N_m is the set of instance variables and the set of normal methods in class C, respectively.*
- *$A = \{(m, v) \mid m$ reads or updates v, $m \in N_m$, $v \in N_v$ $\}$*

For example, the reference graph for class *Stack* is depicted in Figure 5 which includes two normal methods *push* and *pop*, and the interactions with them.

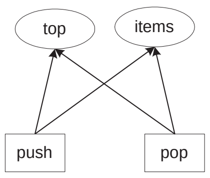


Figure 5: The reference graph for class Stack

A class is the most cohesive when the members of a class have the maximum connectivity among them; that is, every normal method in a class interacts with all of the instance variables in the class. Special methods are excluded because they inherently reference only a part of the instance variables, and have no influence on the cohesiveness.

Definition 3.3 *A reference graph, G_r = (N, A), is a* **Most Cohesive Component(MCC)** *if each method in $M(G_r)$ has interactions with all of instance variables in $V(G_r)$; that is, $A = \{(m, v) \mid m \in M(G_r), v \in V(G_r)\}$, where $M(G_r)$ and $V(G_r)$ denote the set of methods and the set of instance variables in G_r, respectively.*

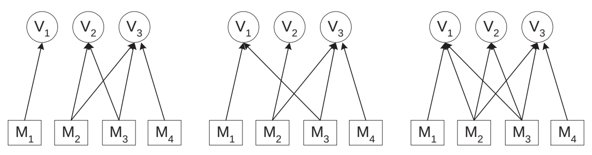
According to this definition, the reference graph for class *Stack* in Figure 5 is a MCC because *push* and *pop* reference both of the instance variables, *items* and *top*. A reference graph which consists solely of instance variables or only methods, the reference graph with one instance variable or one method is a MCC.

3.2 Defining Cohesion

Our cohesion measure for classes is designed to incorporate our observations into its definition. We note that not only the number of interactions among the members of a class, but also its pattern, that is connectivity among the members, affects the cohesiveness of a class.

Connectivity of Class members
We believe that the cohesiveness of a class varies depending not only on the number of their interactions, but also on the pattern of the interactions among members of the class. More specifically, the connectivity of the members of a class determines the cohesiveness; that is, the more tightly connected the members of a class are, the more cohesive it is.

For example, Figure 6 shows the reference graphs of classes A, B, and C which demonstrate different connectivity. $G_r(A)$ is already disjoint. $G_r(B)$ can be decomposed into two sub-reference graphs if either method M_2 or M_3 is removed, and $G_r(C)$ becomes disjoint when both of the methods M_2 and M_3 are removed. In other words, the members of classes A, B, and C are connected by zero, one, and two method(s), respectively. Therefore, it can be claimed that class C is more cohesive than class B, and class B is more cohesive than class A.



(a) Class A (b) Class B (c) Class C

Figure 6: Classes with different connectivity

As shown in Figures 6, a class which is not MCC has a subset of methods without which the members of the class become disjoint. In fact, those methods actually hold the members of a class together. Let's call the minimum set of methods that can separate a reference graph, G_r, the glue methods of G_r, $M_g(G_r)$.

We introduce the notion of the *connectivity factor* in order to represent how strongly the members of a class are connected by the glue methods.

Definition 3.4 *The* **connectivity factor** *of a reference graph G_r, $F_c(G_r)$, represents the strength of the connectivity among the members, and is defined to be the ratio of the number of the glue methods, $|M_g(G_r)|$ to the number of the methods, $|M(G_r)|$.*

$$F_c(G_r) = \frac{|M_g(G_r)|}{|M(G_r)|} \tag{1}$$

For example, all classes A, B, and C in Figure 6 have four methods and their connectivity factors are $\frac{0}{4}$, $\frac{1}{4}$, and $\frac{2}{4}$; because they have zero, one(M_2 or M_3), and two(M_2 and M_3) glue methods, respectively. The connectivity factor of class *Stack* in Figure 5, which is a MCC, is $1(=\frac{2}{2})$.

The value of the connectivity factor ranges from 0 to 1. The connectivity factor is 0 for a class which has no connectivity, and 1 for a class which has the maximum connectivity. As defined in Equation 1, the connectivity factor of a disjoint class is 0 because it has no glue method, and the connectivity factor of a MCC is obviously 1 because all of the methods are also glue methods. In case of a reference graph which consists of either instance variables or methods, the connectivity factor of the reference graph with either single instance variable or single method is defined to be the maximum connectivity(=1) while the reference graph with two or more instance variables(methods) has the minimum connectivity(=0). In addition, the connectivity factor of a class reflects how closely it approaches a MCC; the greater connectivity factor the class has, the more closely it approaches a MCC. Therefore, the cohesion of a class can be defined in terms of its connectivity factor.

Hierarchical Structure of Interaction Patterns
Each component of a class contributes to the cohesiveness of a class differently. Components which are not MCC are apt to reduce the cohesiveness of a class to a certain degree. Therefore, it is desired to incorporate the interaction patterns of the components of a class in defining the cohesiveness of a class.

We construct a *structure tree* for a class to analyze the hierarchical structure of interaction patterns of a class. The structure tree of a class describes how the class is decomposed into its constituent components as the glue methods are removed.

Definition 3.5 *The* **structure tree** *for a reference graph G_r, $T_s(G_r)$, is a tree $T = (N, A)$ with the root node $(G_r, M_g(G_r))$, where*

- $N = \{(G_r{}^i, M_g(G_r{}^i)) \mid G_r{}^i \subseteq G_r,\ M_g(G_r{}^i)$ *is the glue methods of* $G_r{}^i\ \}$
- $A = \{(\ (G_r{}^p, M_g(G_r{}^p)), (G_r{}^c, M_g(G_r{}^c))\ ) \mid G_r{}^c$ *is one of the connected sub-reference graphs obtained from* $G_r{}^p$ *by removing* $M_g(G_r{}^p)\}$

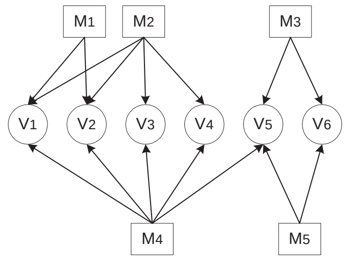


Figure 7: The reference graph of class A

The structure tree for a reference graph can be constructed with the sub-reference graphs which are obtained by removing its glue methods and the associated interactions from the reference graph. This decomposition procedure is applied recursively to the sub-reference graphs until each of them becomes a MCC. After all, the structure tree of a class shows how a class can be decomposed into a collection of MCCs.

Decomposition of a reference graph is not always unique because a reference graph can be separated by different sets of glue methods. In that case, the set of glue methods which results in a higher cohesion for the children is selected.

For example, Figure 8 depicts the structure tree for class A shown in Figure 7. Class A has six instance variables V_1, V_2, V_3, V_4, V_5, and V_6, and five methods M_1, M_2, M_3, M_4, and M_5. The reference graph of Class A can be divided into two sub-reference graphs, $G_r{}^1$ and $G_r{}^2$, by removing the glue method M_4 along with its interactions. $G_r{}^2$ does not need to be decomposed further because it is a MCC. However, $G_r{}^1$ can be partitioned into three sub-reference graphs, $G_r{}^{11}$, $G_r{}^{12}$, and $G_r{}^{13}$ by eliminating the method M_2. After decomposition, $G_r{}^{11}$, $G_r{}^{12}$ and $G_r{}^{13}$ become MCC.

We propose the cohesion of a class depend on the connectivity factors of both itself and its components in the structure tree. Existence of components whose connectivity factor is less than 1 implies that cohesion of a

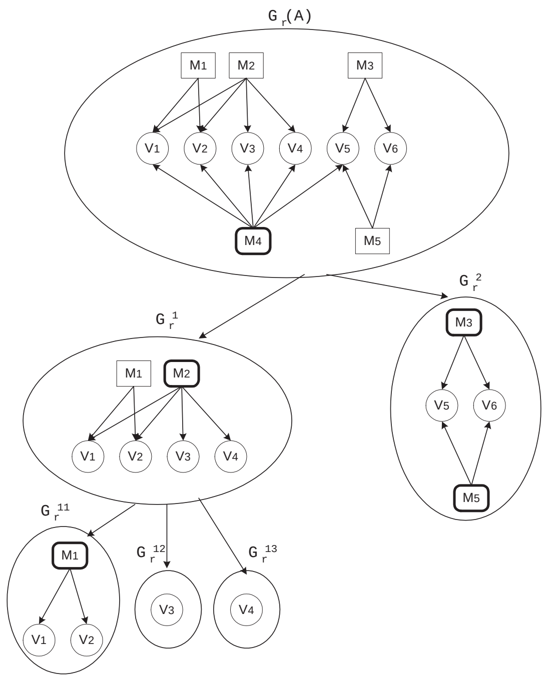


Figure 8: The structure tree of class A

class may have to be rescaled because the class consists of less cohesive components. For example, the cohesion of class A will be reduced from the connectivity factor $\frac{1}{5}$, because class A consists of two components one of which is not a MCC.

Therefore, the connectivity factor of a reference graph needs to be calibrated by its structure factor in order to take into account the cohesiveness of the components. The structure factor for a reference graph indicates the degree of the contribution of its components to the cohesion; the lower value of the structure factor means that the reference graph has less cohesive components, and thus its cohesion should be reduced. As a structure factor we take the average cohesion of the children of the reference graph.

Definition 3.6 *The* **structure factor** *for a reference graph G_r, $F_s(G_r)$ denotes the degree of the contribution of the components to the cohesion and is defined to be the average cohesion of its children in the structure tree.*

$$F_s(G_r) = \frac{1}{n}\sum_{i=1}^{n} CO({G_r}^i), \tag{2}$$

where ${G_r}^i$ is one of the n children of G_r in the structure tree.

The structure factor of a reference graph is recursively defined in terms of its children; the structure factor of a reference graph depends on the cohesion of its children. The value of the structure factor ranges from 0[3] to 1. The structure factor has the maximum value of 1 when every child has the maximum cohesion, that is MCC. The maximum value of the structure factor means the class consists of the most cohesive components, and therefore its cohesion should be equal to the connectivity factor.

Therefore, the cohesion of a reference graph is defined in terms of the connectivity factor and the structure factor.

Definition 3.7 *A cohesion for a reference graph, G_r, $CO(G_r)$ is defined to be its connectivity factor, $F_c(G_r)$ calibrated by its structure factor, $F_s(G_r)$.*

$$\begin{aligned} CO(G_r) &= F_c(G_r) \times F_s(G_r) \\ &= F_c(G_r) \times \frac{1}{n}\sum_{i=1}^{n} CO({G_r}^i) \end{aligned} \tag{3}$$

For example, the cohesion of class A can be computed as in Figure 9; the cohesion of class A is $\frac{1}{5} \times \frac{3}{4}$, where $\frac{1}{5}$ is the connectivity factor of $G_r(A)$, and $\frac{3}{4}$ is the structure factor of $G_r(A)$. The cohesion is reduced by the structure factor $\frac{3}{4}$ from its connectivity factor $\frac{1}{5}$.

In addition, the cohesions of classes A, B, and C in Figure 6 are calculated as 0, $\frac{3}{16}$, and $\frac{1}{2}$, respectively, and they are consistent with our intuition; the tighter the members of a class are bound, the more cohesive it is. The structure trees and the detailed calculations are described in Appendix.

The definition of the cohesion of a class is consistent with our intuition in the following respects.

- A class with disjoint interaction patterns has the lowest cohesion because it has 0 connectivity factor.
- A class with the maximum interaction pattern has the highest cohesion because it has 1 connectivity factor.
- The cohesion of a class is proportional to the connectivity of the class members, that is, the connectivity factor. $CO(C) \approx F_c(G_r(C))$ is evident from the Equation 3.
- When each component of a class has maximum cohesion, the cohesion of the class is equal to the connectivity factor of its reference graph. On the other hand, children with a connectivity factor less than

[3]Actually, the structure factor can not be 0 because each of the children has cohesion greater than 0.

G_r	$F_c(G_r)$	$F_s(G_r)$	$CO(G_r)$
$G_r(A)$	$\frac{1}{5}$	$\frac{1}{2}(CO({G_r}^1) + CO({G_r}^2)) = \frac{1}{2}(\frac{1}{2}+1) = \frac{3}{4}$	$\frac{1}{5} \times \frac{3}{4} = \frac{3}{20}$
${G_r}^1$	$\frac{1}{2}$	$\frac{1}{3}(CO({G_r}^{11}) + CO({G_r}^{12}) + CO({G_r}^{13})) = 1$	$\frac{1}{2} \times 1 = \frac{1}{2}$
${G_r}^2$	$\frac{2}{2}$	-	1
${G_r}^{11}$	$\frac{1}{1}$	-	1
${G_r}^{12}$	1	-	1
${G_r}^{13}$	1	-	1

Figure 9: Cohesion of Class A

1 will reduce the cohesion of the class, and the degree of reduction depends on how far components deviate from MCC.

For example, consider classes A, B, and C in Figure 10. all of $G_r(A)$, $G_r(B)$, and $G_r(C)$ have the connectivity factor c_1(obviously, $c_1 < 1$). The reference graph of class A has two components with maximum cohesion. Therefore, the cohesion of class A is equal to the connectivity factor of $G_r(A)$ because the structure factor $F_s(G_r(A))$ is 1.

Suppose c_2 is greater than c_3. Then, class B should be more cohesive than class C, because $G_r(B)$ has a more cohesive component than $G_r(C)$. This turns out from the calculation; the structure factor of class B, $F_s(G_r(B)) = \frac{1}{2}(c_2 + 1)$, is greater than that of class C, $F_s(G_r(C)) = \frac{1}{2}(c_3 + 1)$, and consequently, the cohesion of class B, $CO(B) = c_1 \times \frac{1}{2}(c_2 + 1)$, is greater than that of class C, $CO(C) = c_1 \times \frac{1}{2}(c_3 + 1)$.

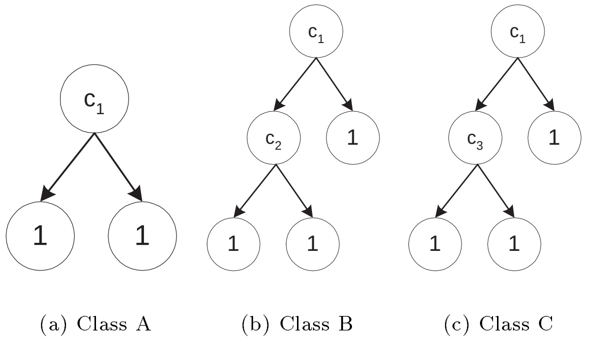


Figure 10: Examples for cohesion

4 Conclusion and Future Works

In this paper, we proposed a new cohesion measure for classes which is based on the observations on the salient features of classes which have not been considered in the previous approaches.

First, the special methods have no effect on the cohesiveness of a class. Therefore, the cohesiveness of a class can be defined on the reference graph which includes the only interactions with the normal methods. A class is defined to be most cohesive, called most cohesive component, if it has all the possible interactions with the normal methods. This maximal cohesiveness is obviously reasonable and corresponds to the previous approaches.

Second, the interaction pattern of a class, not the interaction number is an important factor on the cohesiveness of the class; the more strongly the members of a class are connected, the more cohesive. Connectivity factor was proposed to indicate the degree of the connectivity among the members.

Third, the connectivity factor needs to be calibrated by the structure factor in order to take into account the cohesiveness of components. The structure factor indicates the degree of the contribution of the components to the cohesion of a class. Therefore, the cohesion of a class is defined to be scaled from its connectivity factor by the extent to the components are deviated from the most cohesive form, MCC.

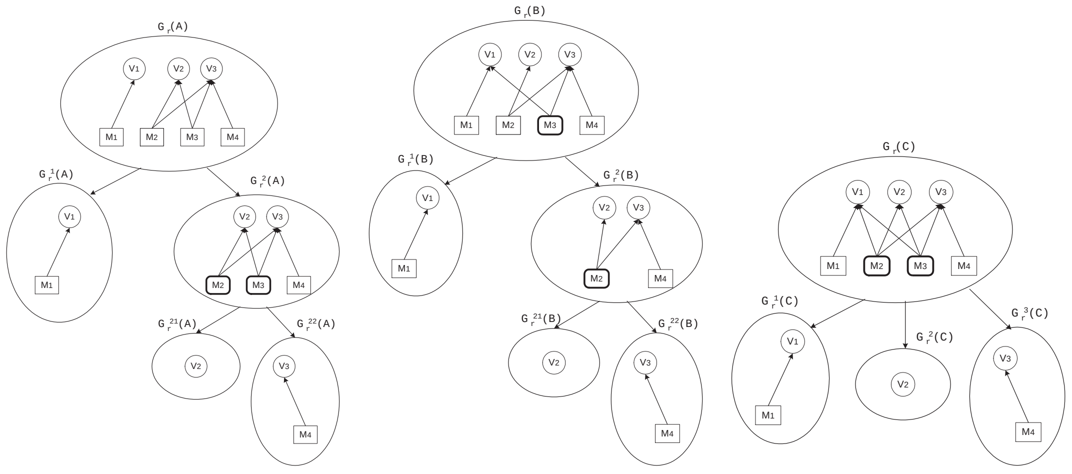
The proposed cohesion measure can be used to evaluate the quality of classes in object-oriented systems such as error-proneness[1] and maintainability[8]. In addition, our cohesion measure can help developers to design a set of classes quality, because our cohesion can be calculated by the information which can be determined at design stage. Empirical evaluation of our cohesion measure remains future works.

Appendix

The structure trees of classes A, B[4]and C in Figure 6 are shown in Figure 11.

Their cohesions are calculated as follows.

[4]Among the methods M_2 and M_3, the method M_3 is selected because it will result in higher structure factor.



(a) Class A (b) Class B (c) Class C

Figure 11: Structure trees for classes A, B, and C

$$
\begin{aligned}
CO(A) &= F_c(G_r(A)) \times F_s(G_r(A)) \\
&= F_c(G_r(A)) \times \frac{1}{2}\Big[CO({G_r}^1(A)) + CO({G_r}^2(A))\Big] \\
&= F_c(G_r(A)) \times \frac{1}{2}\Big[F_c({G_r}^1(A)) + \\
&\Big\{F_c({G_r}^2(A)) \times \frac{1}{2}\Big(CO({G_r}^{21}(A)) + CO({G_r}^{22}(A))\Big)\Big\}\Big] \\
&= \frac{0}{4} \times \frac{1}{2}\Big[1+\Big\{\frac{2}{3} \times \frac{1}{2}\Big(1+1\Big)\Big\}\Big] \\
&= \frac{0}{4} \times \frac{5}{6} \\
&= 0
\end{aligned}
$$

$$
\begin{aligned}
CO(B) &= F_c(G_r(B)) \times F_s(G_r(B)) \\
&= F_c(G_r(B)) \times \frac{1}{2}\Big[CO({G_r}^1(B)) + CO({G_r}^2(B))\Big] \\
&= F_c(G_r(B)) \times \frac{1}{2}\Big[F_c({G_r}^1(B)) + \\
&\Big\{F_c({G_r}^2(B)) \times \frac{1}{2}\Big(CO({G_r}^{21}(B)) + CO({G_r}^{22}(B))\Big)\Big\}\Big] \\
&= \frac{1}{4} \times \frac{1}{2}\Big[1+\Big\{\frac{1}{2} \times \frac{1}{2}\Big(1+1\Big)\Big\}\Big] \\
&= \frac{1}{4} \times \frac{3}{4} \\
&= \frac{3}{16}
\end{aligned}
$$

$$
\begin{aligned}
CO(C) &= F_c(G_r(C)) \times F_s(G_r(C)) \\
&= F_c(G_r(C)) \times \\
&\frac{1}{3}\Big[CO({G_r}^1(C)) + CO({G_r}^2(C)) + CO({G_r}^3(C)\Big] \\
&= F_c(G_r(C)) \times \\
&\frac{1}{3}\Big[F_c({G_r}^1(C)) + F_c({G_r}^3(C)) + F_c({G_r}^3(C))\Big] \\
&= \frac{1}{2} \times 1 \\
&= \frac{1}{2}
\end{aligned}
$$

REFERENCES

[1] Victor R. Basili, Lionel C. Briand and Walcélio L. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Trans. on Software Engineering*, vol. 22, no. 10, pp. 751-761, Oct. 1996.

[2] G. Booch, *Object-Oriented Analysis and Design with Applications*, Benjamin/Cummings Publishing Company, Inc. 1991.

[3] Lionel Briand, Sandro Morasca and Victor R. Basili, "Defining and Validating High-Level Design Metrics," *Technical Report CS-TR-3301-1*, University of Maryland, Dept. of Computer Science, College Park, Md., 1994.

[4] H. S. Chae and Y. R. Kwon, "Assessing and Restructuring of Classes Based on Cohesion," *Proc. of Asia-Pacific Software Engineering Conf.*, pp. 76-82, Seoul, Korea, 1996.

[5] Shyam. R. Chidamber and Chris. F. Kemerer, "Towards a Metrics Suite for Object Oriented Design," *Proc. of 6th ACM Conf. on Object-Oriented Systems, Languages, and Applications*, pp. 197-211, 1991.

[6] Shyam. R. Chidamber and Chris F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. on Software Engineering*, vol. 20, no. 6, pp. 476-493, Jun. 1994.

[7] Martin Hitz and Behzad Montazeri, "Measuring Coupling and Cohesion in Object-Oriented Systems," *Proc. Int. Symposium on Applied Corporate Computing*, Monterrey, Mexico, Oct. 1995.

[8] Wei Li and Sallie Henry, "Object-Oriented Metrics that Predict Maintainability," *Journal of Systems and Software*, vol. 23, no. 2, pp. 111-122, Feb. 1993.

[9] Alan Snyder, "Encapsulation and Inheritance in Object-Oriented Programming Languages," *Proc. of 1th ACM Conf. on Object-Oriented Systems, Languages, and Applications*, pp. 84-91, Sep. 1986.