

# A New Class Cohesion Criterion: An empirical study on several systems

Linda Badri and Mourad Badri  
Département de mathématiques et d'informatique  
Université du Québec à Trois-Rivières  
C.P. 500, Trois-Rivières, Québec, Canada, G9A 5H7  
Phone : (819) 376 5011 {3827 | 3834}  
E-mail : {Linda | Mourad}\_Badri@uqtr.ca

**Abstract** - Class cohesion is considered as one of important object-oriented software attributes. Cohesion refers to the degree of the relatedness of the members in a class. High cohesion is a desirable property of classes. Several metrics have been proposed in the literature in order to measure class cohesion in object-oriented systems. They capture class cohesion in terms of connections among members within a class. The major existing class cohesion metrics are essentially based on instance variables usage criteria. It is a special and a restricted way of capturing class cohesion. Most of these metrics have been experimented and widely discussed in the literature. They do not take into account some characteristics of classes as stated in many papers. We believe that class cohesion metrics should not exclusively be based on common instance variables usage. We present, in this position paper, a new criterion, which focuses on interactions between class methods and propose a new class cohesion metric. We developed a cohesion measurement tool for Java programs and performed a case study on several systems. The selected test systems vary in size and domain. The obtained results demonstrate that our new class cohesion metric captures several pairs of related methods, which are not captured by the existing cohesion metrics.

**Index Terms** - OO Software Quality, Software Metrics, Cohesion, Criterion, Connected methods.

## 1 INTRODUCTION

Software metrics have become essential in some disciplines of software engineering [21]. In the field of software quality, metrics are used for assessing several software attributes (complexity, coupling, cohesion, etc.). They provide, therefore, an important assistance to developers and managers in order to assess and improve software quality during the development process. Cohesion refers to the degree of the relatedness of the members in a component. High cohesion is a desirable property of components. It is widely recognized that highly cohesive components tend to have high maintainability and reusability [4, 5, 9, 20]. The cohesion of a component allows the measurement of its structure quality. The cohesion degree of a component is high, if it implements a single logical function. All the parts of a component must contribute to this implementation. Yourdon and Constantine introduced cohesion in the traditional applications as a measure of the extent of the functional relationships of the elements in a module [22].

In the object paradigm, a class is cohesive when its parts are highly correlated. It should be difficult to split a cohesive class. A class with low cohesion has disparate and non-related members. Cohesion can be used to identify the poorly designed classes. Grady Booch describes high functional cohesion as existing when the elements of a component (such as a class) all work together to provide some well-bounded behaviour [7]. Class cohesion is considered as one of most important characteristics in object-oriented design. Cohesion is an underlying goal to continually consider during the design process [19].

Several metrics have been proposed in the literature in order to measure class cohesion in object-oriented systems. The major existing cohesion metrics have been presented in detail and are

categorized in [5]. They are based on either instance variables usage or sharing of instance variables. These metrics capture class cohesion in terms of connections among members within a class. They count the number of instance variables used by methods or the number of methods pairs that share instance variables. We believe that it is only a particular way of capturing class cohesion, which is based on instance variables usage criteria. Most of these metrics have been experimented and widely discussed in the literature [3, 6, 8, 12, 13, 14]. Several studies have noted that they fail in many situations to properly reflect the cohesiveness of classes [17, 9]. According to many authors, they do not take into account some characteristics of classes, for example, sizes of cohesive parts as stated in [1] and connectivity among members as stated in [9].

Beyond these aspects, we believe that the existing metrics fail to reflect properly the properties of class cohesion, particularly in terms of related methods. They are based on restricted criteria and could lead, as stated in many works, to unexpected values of cohesion in many situations. We believe that class cohesion should not exclusively be based on common instance variables usage as stated in [17] and have to go beyond this aspect by considering the interaction patterns among class methods. We note that, in many situations, several methods are functionally related together without sharing any instances variables. We extend the existing criteria by considering different ways of capturing class cohesion. We introduce, in this paper, a new criterion, which focuses on interactions between class methods. We developed a cohesion measurement tool for Java programs and performed a case study on several systems. The obtained results demonstrate that our new class cohesion metric, based on the proposed cohesion criteria, captures several pairs of related methods, which are not captured by the existing cohesion metrics.

This paper is structured as follows: Section 2 provides an overview of the major class cohesion metrics in object-oriented systems and highlights their weakness. Section 3 presents the proposed class cohesion criteria. Section 4 proposes the new definition of class cohesion that we propose. In section 5, we present the results of our empirical study. Finally, conclusions and future works are presented in section 6.

## 2 CLASS COHESION METRICS

Classes are considered as the basic units of object-oriented software. Classes should then be designed to have a good quality. However, improper modeling in the design phase can produce classes with low cohesion. In order to assess class cohesion in object-oriented systems, several metrics have been

proposed in the literature. Many authors have defined class cohesion by proposing their cohesion metrics. Most of the proposed cohesion metrics are inspired from the LCOM (Lack of COhesion in Methods) metric defined by Chidamber and Kemerer [10]. Many authors have redefined the LCOM metric as referenced in the following paragraphs.

The existing cohesion metrics are based on either instance variables usage or sharing of instance variables. These metrics are presented in detail in [5]. A class is more cohesive, as stated in [9], when a larger number of its instance variables are referenced by a method (LCOM5 [14], Coh [5]), or a larger number of methods pairs share instance variables (LCOM1 [10], LCOM2 [11], LCOM3 [20], LCOM4 [16], Co [16], TCC and LCC [4], DC [2]). Table 1 gives a summary of their definition.

Metric	Definition
LCOM1	Lack of cohesion in methods. The number of pairs of methods in the class using no instance variables in common.
LCOM2	Let P be the pairs of methods without shared instance variables, and Q be the pairs of methods with shared instance variables. Then $LCOM2 =  P  -  Q $ , if $ P  >  Q $ . If this difference is negative, LCOM2 is set to zero.
LCOM3	Consider an undirected graph G, where the vertices are the methods of a class, and there is an edge between two vertices if the corresponding methods share at least one instance variable. Then $LCOM3 =   \text{connected components of G}  $
LCOM4	Like LCOM3, where graph G additionally has an edge between vertices representing methods $M_i$ and $M_j$ , if $M_i$ invokes $M_j$ or vice versa.
Co	Connectivity. Let V be the vertices of graph G from LCOM4, and E its edges. Then $Co = 2 \cdot \frac{ E  - ( V  - 1)}{( V  - 1) \cdot ( V  - 2)}$
LCOM5	Consider a set of methods $\{M_i\}$ ( $i = 1, \dots, m$ ) accessing a set of instance variables $\{A_j\}$ ( $j = 1, \dots, a$ ). Let $\mu(A_j)$ be the number of methods that reference $A_j$ . Then $LCOM5 = \frac{(1/a) \sum_{1 \leq j \leq a} \mu(A_j) - m}{1 - m}$
Coh	A variation on LCOM5. $Coh = \frac{\sum_{1 \leq j \leq a} \mu(A_j)}{m \cdot a}$
TCC	Tight Class Cohesion. Consider a class with N public methods. Let NP be the maximum number of public method pairs: $NP = [N * (N - 1)] / 2$ . Let NDC be the number of direct connections between public methods. Then TCC is defined as the relative number of directly connected public methods. Then, $TCC = NDC / NP$ .
LCC	Loose Class Cohesion. Let NIC be the number of direct or indirect connections between public methods. Then LCC is defined as the relative number of directly or indirectly connected public methods. $LCC = NIC / NP$ .

**Table 1:** The major existing cohesion metrics.

Chidamber and Kemerer propose LCOM (LCOM1 and LCOM2) metric to assess class cohesion. LCOM2 equals the number of methods pairs that have no attributes in common minus the number of methods pairs that share at least one attribute. LCOM2 equals 0 if this value is negative. LCOM2, as a redefinition of LCOM1, has been widely discussed in the literature [2, 5, 8, 14, 16]. LCOM2 of many classes is set to be zero although different cohesions are expected [3]. Li and Henry redefine LCOM in [20]. They define LCOM3 as the number of disjoint sets of methods. Each set contains only methods that share at least one attribute. Hitz and Montazeri redefine LCOM3 in [16]. Their metric is based on graph theory and defined as the number of connected components of a graph. The vertices of the graph represent the methods of the class. There is an edge between two vertices if the corresponding methods access the same instance variable. Hendersen-Sellers propose LCOM5 in [14] which is based on the number of referenced instance variables. A class is more cohesive when a large number of its instance variables are referenced by a method. Briand et al. propose a redefinition of this metric in [5].

Bieman and Kang propose TCC (Tight Class Cohesion) and LCC (Loose Class Cohesion) as cohesion metrics [4]. They also consider methods pairs using instance variables in common. An instance variable may be directly or indirectly used by a method. An instance variable is used directly by a method  $M_i$ , if the instance variable appears in the body of the method  $M_i$ . An instance variable is used indirectly by a method  $M_i$ , if the instance variable is directly used by a method  $M_j$  that is either directly or indirectly invoked by  $M_i$ . Two methods are directly related if they both use either directly or indirectly a common instance variable. TCC is defined as the percentage of methods pairs, which are directly related. LCC is defined as the number of methods pairs that are either directly or indirectly related.

All the existing cohesion metrics attempt to quantify the cohesion of a class by taking into account only the interactions among its methods and its instances variables. This criterion constitutes, in our opinion, a restrictive way for capturing the cohesion of classes. We note, in many situations, that methods of a class may be related together without shared instance variables. We believe that cohesion metrics must also take into account the interaction patterns between methods of a class. Several studies have noted that the existing cohesion metrics do not take into account all characteristics of classes and fail in many situations to properly reflect the cohesion of classes [1, 9, 18].

### 3 RELATED METHODS: COHESION CRITERIA

Two methods can be related (connected) in many ways. We define two fundamental cohesion criteria:

- Attributes Usage Criterion,
- Methods Invocation Criterion.

#### 3.1 Attributes Usage Criterion ( $C_A$ )

Consider a class  $C$ . Let  $A = \{A_1, A_2, \dots, A_n\}$  be the set of its attributes and  $PUM = \{M_1, M_2, \dots, M_n\}$  be the set of its public methods. Let  $UA_{M_i}$  be the set of all the attributes used directly

or indirectly by the public method  $M_i$ . An attribute is used directly by a method  $M_i$ , if the attribute appears in the body of the method  $M_i$ . The attribute is indirectly used by the method  $M_i$ , if it is used directly by another method of the class that is invoked directly or indirectly by  $M_i$ . There are  $n$  sets  $UA_{M_1}, UA_{M_2}, \dots, UA_{M_n}$ . Two public methods  $M_i$  and  $M_j$  are directly related by the *UA relation* if  $UA_{M_i} \cap UA_{M_j} \neq \emptyset$ . It means that there is at least one attribute shared (directly or indirectly) by the two methods.

#### 3.2 Methods Invocation Criterion ( $C_M$ )

Consider a class  $C$ . Let  $PUM = \{M_1, M_2, \dots, M_n\}$  be the set of its public methods and  $PRM = \{I_1, I_2, \dots, I_k\}$  be the set of its others (private and protected) methods.

Let  $PUM_{M_i}$  be the set of all the public methods of the class  $C$ , which are invoked directly or indirectly by the public method  $M_i$ . A public method  $M_j$  is called directly by a public method  $M_i$ , if  $M_j$  appears in the body of  $M_i$ . A public method  $M_j$  is indirectly called by a public method  $M_i$ , if it is called directly by another method of the class  $C$  that is invoked directly or indirectly by  $M_i$ . There are  $n$  sets  $PUM_{M_1}, PUM_{M_2}, \dots, PUM_{M_n}$ .

Let  $PRM_{M_i}$  be the set of all the others methods (private and protected) of the class  $C$ , which are invoked directly or indirectly by the public method  $M_i$ . There are  $n$  sets  $PRM_{M_1}, PRM_{M_2}, \dots, PRM_{M_n}$ .

Let  $IM_{M_i} = PRM_{M_i} \cup PUM_{M_i}$  be the set of all the methods of the class  $C$ , which are invoked by the public method  $M_i$ . Two public methods  $M_i$  and  $M_j$  are directly related by the *IM relation* if  $IM_{M_i} \cap IM_{M_j} \neq \emptyset$ . We also consider that  $M_i$  and  $M_j$  are directly related if  $M_j \in IM_{M_i}$  or  $M_i \in IM_{M_j}$ .

## 4 CLASS COHESION: A NEW MEASURE

The adopted approach for the estimation of class cohesion is based on the different relationships that may exist between its public methods. It takes into account different ways of capturing the functional cohesion of the class, by focusing on the defined cohesion criteria. Class cohesion, at this stage of our work, refers essentially the relatedness of public methods of a class, which represent the functionalities used by its clients. The others methods of the class are included indirectly through the public methods.

Class cohesion in our approach, as stated initially in [2], is defined in terms of the relative number of related methods in the class. It is comparable to the approach adopted by Bieman in [4]. We have revised our initial definition of class cohesion [2] by extending the methods invocation criterion in the one hand and introducing the concept of indirect usage of attributes defined by Bieman and Kang in [4] in the other hand. We have also extended this concept to the methods invocation criterion. Two public methods  $M_i$  and  $M_j$  may be directly connected in many ways : they share at least one instance variable in common (UA relation), or interact at least with another method of the same class (IM relation), or both. It means that:  $UA_{M_i} \cap UA_{M_j} \neq \emptyset$  or  $IM_{M_i} \cap IM_{M_j} \neq \emptyset$ .

Consider a class  $C$  with  $PUM = \{M_1, M_2, \dots, M_n\}$  the set of its public methods. The maximum number of public methods pairs, as stated in [2, 4], is  $N * (N - 1) / 2$ . Consider an undirected graph  $G$ , where the vertices are the public methods of the class  $C$ , and there is an edge between two vertices if the corresponding methods are directly related. Let  $E_D$  be the number of edges in the graph. The degree of cohesion in the class  $C$  based on the direct relation between its public methods is defined as:  $DC_D = |E_D| / [N * (N - 1) / 2] \in [0,1]$ .

$DC_D$  gives the percentage of methods pairs, which are directly (as defined below) related. However, two public methods  $M_i$  and  $M_j$  can be indirectly related if they are directly or indirectly related to a method  $M_k$ . The indirect relation, introduced by Bieman and Kang [4], is the transitive closure of the direct relation. We use this concept in our approach for identifying the indirect related methods. Thus, a method  $M_i$  is indirectly connected with a method  $M_k$  if there is a sequence of methods  $M_1, M_2, M_3, \dots, M_k$  such that  $M_i$  is directly connected to  $M_{i+1}$  ( $i=1, k-1$ ).

Consider now the same graph  $G$ , where the vertices are the public methods of the class  $C$ , and there is an edge between two vertices if the corresponding methods are directly or indirectly related (transitive closure). Let  $E_I$  be the number of edges in the graph. The degree of cohesion in the class  $C$  in this case (direct and indirect relations) is defined as:  $DC_I = |E_I| / [N * (N - 1) / 2] \in [0,1]$ .

$DC_I$  gives the percentage of methods pairs, which are directly or indirectly related. The new definition that we propose for class cohesion seems to be more appropriate than the others, particularly the one proposed by Bieman and Kang in [4]. It allows capturing more properties of classes, particularly, interactions between methods of a class. Two public methods can be related by calling directly or indirectly, for instance, internal methods, which do not use any attribute of the class

(sharing of code). Such characteristics are not captured by the other definitions of class cohesion presented in section 2.

## 5 EMPIRICAL STUDY

We developed a cohesion measurement tool (in Java) for Java programs to automate the computation of the major existing class cohesion metrics presented in table 1 including  $DC_D$  and  $DC_I$  metrics. In summary, height metrics have been implemented: LCOM1, LCOM2, Co, Coh, TCC, LCC,  $DC_D$  and  $DC_I$ . In order to demonstrate the effectiveness of the new criterion and the proposed metrics for class cohesion, we performed a case study on several systems.

### 5.1 Selected systems

As a first experimentation of our approach and to achieve significant and general results, we have chosen several systems, which can be freely downloaded from the web. Our main goal was to analyze a maximum number of Java classes from different systems. The considered systems vary in size and domain. Table 2 provides some of their characteristics. System-1 (Alma) is designed for migrating code written in old languages to newer ones. System-2 (Opencms: <http://www.opencms.org>) allows a company to maintain a website. System-3 (JSP-GNU Java Server Pages: <http://klomp.org/gnujsp/>) is an implementation of Sun's Java Server Pages. System-4 (XXL: <http://dbs.mathematik.uni-marburg.de/research/projects/xxl/>) provides a collection of index structures, query operators and algorithms allowing the performance evaluation of new query processing developments. System-5 (JIU - Java Imaging Utilities: <http://jiu.sourceforge.net/>) offers functionality to load, analyze, process and save pixel images. Finally, System-6 (Apache Ant: <http://ant.apache.org/>) is a Java-based build tool, with functionalities similar to the unix "make" utility.

	System-1	System-2	System-3	System-4	System-5	System-6	Total
# of classes	738	342	75	405	168	504	2232
# of special classes	186	58	12	100	36	58	450
# of considered classes	552	284	63	305	132	446	1782
# of attributes	2346	1587	185	940	763	2795	8616
# of methods	6457	3806	388	2359	1224	4541	18775
# of public methods	5749	3094	351	2208	995	3536	15933
# of protected methods	358	258	2	141	2	539	1300
# of private methods	350	454	35	10	227	466	1542
Size in repository	4.09 MB	5.94 MB	0.29 MB	3.55 MB	1.09 MB	5.12 MB	20.08 MB

**Table 2:** Some characteristics of the selected test systems.

Several classes in the considered systems have only one method or do not have any methods. These classes were considered as special classes and have been excluded from our measurements. We also excluded all abstract classes. Overloaded methods within the same class were treated as one method. Moreover, all special methods (constructors, destructors) were removed. The environment that we have developed for the computation of the selected metrics is composed of several tools. A Java parsing tool ([www.antlr.org](http://wwwantlr.org)) that we have extended, parses the test systems source code. The extracted information contains various data about all classes (attributes, methods, used attributes, invoked methods, etc.). This information is treated, in a second phase, by a metrics tool that we have developed. The obtained results are transferred into Excel for statistical processing. We collected the values for all the selected metrics from the test systems. For each metric, we calculated some descriptive statistics (minimum, maximum, mean, median, and standard deviation).

## 5.2 Results

We measured class cohesion values for the 6 selected test systems. Table 3 provides the descriptive statistics for all the implemented metrics. LCOM1 and LCOM2 count the number of methods pairs with shared instance variables. These measures are not normalized. Coh is based on the instance variable usage and count the number of the interactions between instance variables and methods. Co, TCC and LCC are based on the ratio of methods pairs with shared instance variables. TCC and LCC consider indirect sharing of instance variables by method invocation. TCC and LCC metrics are supposed to take into account the interactions between methods. The problem with these metrics is that when the interactions between methods do not use at all any instance variables, these interactions are not captured despite the fact that the corresponding methods are related. This often occurs that a large number of related methods pairs are not reflected in the cohesion values.

Systems		LCOM1	LCOM2	Co	Coh	TCC	LCC	DC <sub>D</sub>	DC <sub>I</sub>
System-1	Minimum	0	0	0	0	0	0	0	0
	Maximum	5486	5407	1	1	1	1	1	1
	Mean	139.705	119.174	0.146	0.300	0.354	0.416	0.407	0.520
	Median	15.000	9.000	0.000	0.238	0.214	0.300	0.322	0.493
	Std. Dev.	431.163	406.053	0.264	0.303	0.374	0.409	0.374	0.410
System-2	Minimum	0	0	0	0	0	0	0	0
	Maximum	13997	13996	1	1	1	1	1	1
	Mean	244.408	129.063	0.130	0.218	0.285	0.362	0.323	0.425
	Median	10.000	6.000	0.000	0.143	0.029	0.030	0.167	0.278
	Std. Dev.	1307.11	861.646	0.251	0.232	0.380	0.435	0.378	0.432
System-3	Minimum	0	0	0	0	0	0	0	0
	Maximum	287	249	1	1	1	1	1	1
	Mean	26.968	20.968	0.120	0.666	0.728	0.764	0.747	0.805
	Median	0.000	0.000	0.000	1.000	1.000	1.000	1.000	1.000
	Std. Dev.	62.523	55.806	0.269	0.410	0.404	0.401	0.388	0.351
System-4	Minimum	0	0	0	0	0	0	0	0
	Maximum	3003	3003	1	1	1	1	1	1
	Mean	42.915	38.184	0.211	0.296	0.335	0.359	0.489	0.607
	Median	7.000	3.000	0.000	0.278	0.109	0.143	0.500	0.750
	Std. Dev.	247.670	247.729	0.322	0.280	0.387	0.405	0.385	0.409
System-5	Minimum	0	0	0	0	0	0	0	0
	Maximum	2482	2479	1	1	1	1	1	1
	Mean	85.409	68.000	0.176	0.255	0.324	0.438	0.362	0.498
	Median	6.000	6.000	0.061	0.189	0.207	0.333	0.277	0.500
	Std. Dev.	299.329	258.185	0.287	0.276	0.362	0.433	0.366	0.440
System-6	Minimum	0	0	0	0	0	0	0	0
	Maximum	2890	2620	1	1	1	1	1	1
	Mean	84.765	73.217	0.149	0.324	0.381	0.489	0.395	0.533
	Median	14.000	8.000	0.000	0.233	0.300	0.467	0.333	0.600
	Std. Dev.	257.531	238.511	0.267	0.292	0.348	0.436	0.347	0.438

**Table 3:** Cohesion metrics results for the test systems.

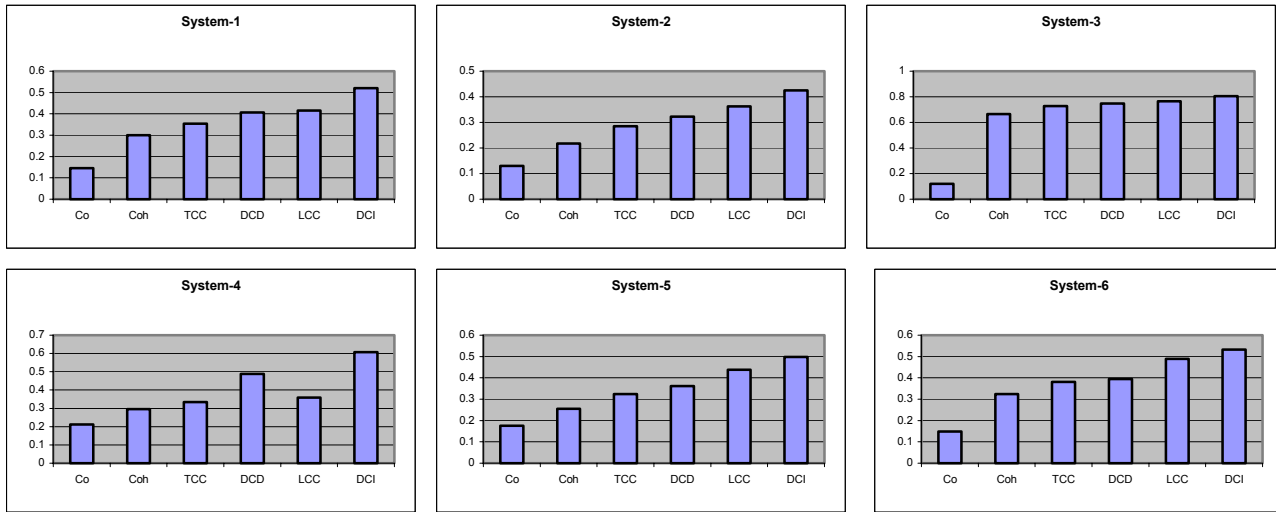


Figure 1: Mean values of the normalized metrics.

The obtained results for  $DC_D$  and  $DC_I$  show clearly that the two metrics capture more pairs of related methods than the others, particularly Co, TCC and LCC metrics. These results indicate that our metrics capture an additional aspect of properties of classes, which is not captured by the others. This is due, in our opinion, to the combination of the proposed criteria. This aspect will be discussed and validated in the next section. Figure 1 shows the mean values of only the normalized metrics (Co, Coh, TCC,  $DC_D$ , LCC, and  $DC_I$ ). Figure 2 shows the distribution of the cohesion values of all the classes of the system-4 according to the TCC,  $DC_D$ , LCC, and  $DC_I$  metrics. These

results indicate, in particular, that the  $DC_D$  and  $DC_I$  metrics capture more pairs of related methods than the TCC and LCC metrics.

The main objective of this work was to demonstrate the effectiveness of the new cohesion criterion that we introduced in section 3. It is for this reason that we will not discuss in detail the cohesion values of the test systems. The results in table 3 show clearly that most of the test systems are not cohesive. However, System 3 is strongly cohesive ( $DC_I = 0.805$ ) compared to the others systems.

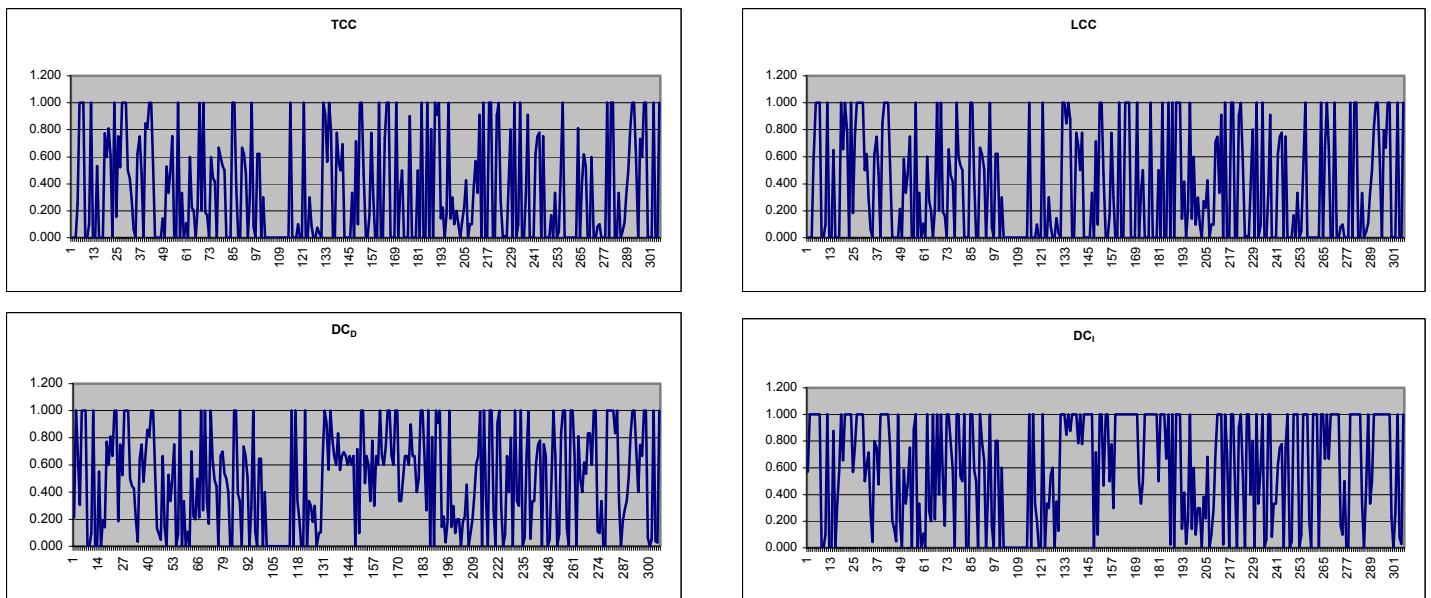


Figure 2: Cohesion values of System-4 classes.

### 5.3 Empirical validation of the $C_M$ criterion

In this section we are interested in comparing the results of  $DC_D$  by considering only the first criterion ( $C_A$ ) and  $DC_D$  by combining the two proposed criteria ( $C_A$  and  $C_M$ ). Our goal is to demonstrate that the second metric is more significant than the first one and allows capturing more pairs of related methods. If we consider the  $DC_D$  metric by taking into account only the first criterion, the metric is then equivalent to the TCC metric [4].

Let  $DC_D(C_A)$  be the degree of cohesion by considering only the first criterion (attributes usage criterion). Let  $DC_D(C_A \text{ and } C_M)$  be the degree of cohesion by considering the two criteria (attributes usage criterion and methods invocation criterion). Let  $Diff$  be the difference between  $DC_D(C_A \text{ and } C_M)$  and  $DC_D(C_A)$ . If there is no difference between the values of  $DC_D(C_A)$  and  $DC_D(C_A \text{ and } C_M)$ , then the population mean of the differences should be significantly zero. We are interested in comparing the two metrics  $DC_D(C_A)$  and  $DC_D(C_A \text{ and } C_M)$ . We collected the data for the two metrics from the 6 test systems. The results with some descriptive statistics are presented in table 4. We believe that the second metric is more significant than the first one. In order to validate this hypothesis, and knowing that the two criteria are dependent, we use an appropriate statistical test (the *paired t-test* [15]). Let  $\mu_1$  be the

mean value of  $DC_D(C_A \text{ and } C_M)$ . Let  $\mu_2$  be the mean value of  $DC_D(C_A)$ . We have then two hypotheses:

- $H_0 : \mu_1 = \mu_2$       The two metrics are equivalent.  
 $H_1 : \mu_1 > \mu_2$        $DC_D(C_A \text{ and } C_M)$  is more significant than  $DC_D(C_A)$ .

Let  $Diff$  be  $(\mu_1 - \mu_2)$ . The precedent test will be equivalent to:  $H_0 : Diff = 0$   $H_1 : Diff > 0$ .

The test statistic is :  $Z = \bar{d} / [S_d / \sqrt{n}]$ , with  $\bar{d}$  : the sample mean value of  $Diff$ ,  $S_d$ : the sample standard deviation of  $Diff$  and  $n$  : the number of classes of the test system.

Knowing that the number  $n$  used in the experiment (for the 6 test systems) is large, the procedure consists in comparing, for each test system,  $Z$  to the normal quantile  $Z_\alpha$  (the chosen values for  $\alpha$  are 0.05 and 0.01). If the value of  $Z$  is greater than the value of  $Z_\alpha$  then we reject the hypothesis  $H_0 : Diff = 0$  and consequently we will accept the hypothesis  $H_1 : Diff > 0$ . In this case, the test statistic will be significant and we will conclude that the  $DC_D(C_A \text{ and } C_M)$  metric is more significant than the  $DC_D(C_A)$  metric. It means that the second criterion (methods invocation criterion) that we introduced in this paper is significant and allows capturing an additional aspect of properties of classes.

Systems	Des. Stat.	$DC_D(C_A)$	$DC_D(C_A \text{ and } C_M)$	Diff	Z	$Z_\alpha$
System-1	Mean	0.407	0.447	0.053	7.368	2.326
	Std. Dev.	0.374	0.382	0.169		
System-2	Mean	0.323	0.343	0.038	4.964	2.326
	Std. Dev.	0.378	0.386	0.129		
System-3	Mean	0.747	0.771	0.019	2.556	2.326
	Std. Dev.	0.388	0.362	0.059		
System-4	Mean	0.489	0.524	0.154	9.639	2.326
	Std. Dev.	0.385	0.380	0.279		
System-5	Mean	0.362	0.386	0.038	2.678	2.326
	Std. Dev.	0.366	0.377	0.163		
System-6	Mean	0.395	0.446	0.014	5.011	2.326
	Std. Dev.	0.347	0.360	0.059		

Table 4: Cohesion criteria comparison.

We collected the data for the two metrics from the 6 test systems and calculated  $Diff$  and  $Z$  for all test systems. These results are presented in table 4. They show clearly, for all the 6 test systems, that  $Z$  is greater than  $Z_\alpha$ . These results show that the  $DC_D(C_A \text{ and } C_M)$  metric is more significant than the  $DC_D(C_A)$  metric. Moreover, they demonstrate that the second criterion (methods invocation criterion) that we introduced in this paper is significant and allows capturing an additional aspect of properties of classes. These results confirm that the proposed metrics for class cohesion are more significant than the others.

## 6 CONCLUSION AND FUTURE WORKS

The major existing class cohesion metrics are essentially based on instance variables usage criteria. We agree that these criteria are important, but we believe that they are not sufficient to capture all the connections among members within a class. This explains in part, in our opinion, why they fail in several situations to reflect the relatedness of the members of a class, and particularly methods of a class. We focused on the interaction patterns among class methods. We suspected that this aspect was not properly reflected in the existing cohesion metrics.

In order to capture additional characteristics of classes and to better measure the cohesion property of classes, we introduced

in this paper a new class cohesion criterion, which is based on methods invocation. We proposed in this work a new approach for class cohesion assessment based on two fundamental criteria: attributes usage criterion and methods invocation criterion. We have revised our initial definition of class cohesion and proposed two new metrics for assessing it. Our main goal in this work was to validate the introduced criterion and our approach for class cohesion definition and assessment. We have developed a cohesion measurement tool for Java programs to automate the computation of the major existing class cohesion metrics including ours. In order to demonstrate the effectiveness of the new criterion and the proposed metrics for class cohesion, we performed a case study on several systems. More than 2000 Java classes have been analyzed.

The obtained results confirm our hypothesis. They show clearly that the proposed metrics, based on a combination of the proposed criteria, capture more pairs of related methods than the existing cohesion metrics, particularly the ones supposed implicitly taking into account the interactions between methods (such as Co, TCC and LCC metrics). These results indicate that the new definition of class cohesion that we propose capture an additional aspect of properties of classes. We believe that the present work constitutes an improvement of class cohesion definition. During our experiment, we collected several data on the analyzed classes. An important part of the collected data has been treated during this work. Actually, we are analyzing the rest of the collected data. As future works we plan to: (1) study in detail the weakly cohesive classes, (2) study the proposed metrics by including other aspects of object-oriented design such as the inheritance between classes, (3) and work on a metric-based approach for assessing classes responsibilities assignment.

## ACKNOWLEDGEMENTS

This work was supported by NSERC (Natural Sciences and Engineering Research Council of Canada) grant.

## REFERENCES

- [1] H. Aman, K. Yamasaki, H. Yamada and MT. Noda, A proposal of class cohesion metrics using sizes of cohesive parts, *Knowledge-Based Software Engineering*, T. Welzer et al. (Eds), pp. 102-107, IOS Press, September 2002.
- [2] L. Badri, M. Badri and S. Ferdinache, Towards Quality Control Metrics for Object-Oriented Systems Analysis, *Proceedings of TOOLS (Technology of Object-Oriented Languages and Systems) Europe'95*, Versailles, France, Prentice-Hall, March 1995.
- [3] V.R. Basili, L.C. Briand and W. Melo, A validation of object-oriented design metrics as quality indicators, *IEEE Transactions on Software Engineering*, 22 (10), pp. 751-761, October 1996.
- [4] J.M. Bieman and B.K. Kang, Cohesion and reuse in an object-oriented system, *Proceedings of the Symposium on Software Reusability (SSR'95)*, Seattle, WA, pp. 259-262, April 1995.
- [5] L.C. Briand, J. Daly and J. Wusr, A unified framework for cohesion measurement in object-oriented systems, *Empirical Software Engineering*, 3 (1), pp. 67-117, 1998.
- [6] L. Briand, J. Wuest, J. Daly and V. Porter, Exploring the relationships between Design Measures and software quality in object-oriented Systems, *Journal of Systems and Software*, No. 51, pp. 245-273, 2000.
- [7] G. Booch, *Object-Oriented Analysis and Design With Applications*, Second edition, Benjamin/Cummings, 1994.
- [8] H. S. Chae and Y.R. Kwon, A cohesion measure for classes in object-oriented systems, *Proceedings of the fifth International Software Metrics Symposium*, Bethesda, MD, pp. 158-166, November 1998.
- [9] H. S. Chae, Y. R. Kwon and D H. Bae, A cohesion measure for object-oriented classes, *Software Practice and Experience*, No. 30, pp. 1405-1431, 2000.
- [10] S.R. Chidamber and C.F. Kemerer, Towards a Metrics Suite for Object-Oriented Design, *Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, Special Issue of SIGPLAN Notices, Vol. 26, No. 10, pp. 197-211, October 1991.
- [11] S.R. Chidamber and C.F. Kemerer, A Metrics suite for object Oriented Design, *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, pp. 476-493, June 1994.
- [12] S.R. Chidamber, David P. Darcy, and C.F. Kemerer, Managerial use of metrics for object-oriented software : An exploratory analysis, *IEEE Transactions on Software Engineering*, Vol. 24, No. 8, pp. 629-639, August 1998.
- [13] K. El Emam and W. Melo, The prediction of faulty class using object-oriented design metrics, *National Research Council of Canada NRC/ERB 1064*, 1999.
- [14] B. Henderson-sellers, *Object-Oriented Metrics Measures of Complexity*, Prentice-Hall, 1996.
- [15] W. W. Hines, D. C. Montgomery, D. M. Goldsman and C. M. Borror, *Probability and statistics in engineering*, Fourth edition, John Wiley & Sons, Inc., 2003.
- [16] M. Hitz and B. Montazeri, Measuring coupling and cohesion in object oriented systems, *Proceedings of the Int. Symposium on Applied Corporate Computing*, pp. 25-27, October 1995.
- [17] H. Kabaili, R.K. Keller, F. Lustman and G. Saint-Denis, Class Cohesion Revisited : An Empirical Study on Industrial Systems, *Proceeding of the Workshop on Quantitative Approaches Object-Oriented Software Engineering, QAOOSE'2000*, France, June 2000.
- [18] H. Kabaili, R.K. Keller and F. Lustman, Cohesion as Changeability Indicator in Object-Oriented Systems, *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering (CSMR 2001)*, Estoril Coast (Lisbon), Portugal, March 2001.
- [19] G. Larman, *Applying UML and Design Patterns, An introduction to object-oriented analysis and design and the unified process*, Second edition, Prentice Hall, 2002.

- [20] W. Li and S. Henry, Object oriented metrics that predict maintainability, *Journal of Systems and Software*, Vol. 23, pp. 111-122, February 1993.
- [21] R. S. Pressman, *Software Engineering, A practitioner's approach*, Fifth edition, Mc Graw Hill, 2001.
- [22] E. Yourdon and L. Constantine, *Structured Design*, Prentice Hall, Englewood Cliffs, N.J., 1979.