JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2004**

Vol. 3, No. 4 (April 2004)
Special issue: TOOLS USA 2003

A Proposal of a New Class Cohesion Criterion: An Empirical Study

Linda Badri and **Mourad Badri**, Department of Mathematics and Computer ScienceUniversity of Quebec at Trois-Rivières, Canada

Abstract

Class cohesion refers to the degree of the relatedness of the members in a class. It is considered as one of most important object-oriented software attributes. 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 only a special and a restricted way of capturing class cohesion. We believe, as stated in many papers, that class cohesion should not exclusively be based on common instance variables usage criteria. 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.

1 INTRODUCTION

Software metrics have become essential in some disciplines of software engineering [Pressman01]. 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. Object technology has been widely used in several areas during the last decade. Class cohesion is considered as one of most important object-oriented software attributes. Cohesion refers to the degree of the relatedness of the members in a component. High cohesion is a desirable property of software components. It is widely recognized that highly cohesive components tend to have high maintainability and reusability [Bieman95, Briand98, Chae00, Li93]. 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 the component must contribute to this implementation.

Cite this article as follows: Linda Badri and Mourad Badri: "A Proposal of a New Class Cohesion Criterion: An Empirical Study", in *Journal of Object Technology*, vol. 3, no. 4, April 2004, Special issue: TOOLS USA 2003, pp. 145-159. http://www.jot.fm/issues/issue_2004_04/article8

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 [Yourdon79]. They have described cohesion as a criterion for the estimation of design quality. 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 behavior [Booch94]. 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. Cohesion is an underlying goal to continually consider during the design process [Larman02].

Several metrics have been proposed in the literature in order to measure class cohesion in object-oriented systems. The major existing class cohesion metrics have been presented in detail and are categorized in [Briand98]. 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 special way of capturing class cohesion, which is based on instance variables usage criteria. These metrics have been experimented and widely discussed in the literature [Basili96, Briand00, Chae98, Chidamber98, ElEmam99, Henderson-Sellers96]. Several studies have noted that the existing cohesion metrics fail in many situations to properly reflect the cohesiveness of classes [Kabaili00, Chae00]. According to many authors, they do not take into account some characteristics of classes, for example, sizes of cohesive parts as stated in [Aman02] and connectivity among members as stated in [Chae00].

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 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 [Kabaili00] and will 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 instance variables. We extended 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 connected methods, which are not captured by the existing cohesion metrics.

The rest of the paper is structured as follows: Section 2 provides an overview of the main class cohesion metrics and highlights their weakness. Section 3 presents the proposed class cohesion criteria. Section 4 presents the new approach that we propose for class cohesion assessment. In section 5, we present the results of our empirical study. Finally, conclusions and future work are presented in section 6.

2 CLASS COHESION: MAJOR EXISTING 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, particularly improper responsibilities assignment decisions, can produce classes with low cohesion. In order to assess class cohesion in object-oriented systems several metrics have been proposed in the literature. Most of the proposed class cohesion metrics are inspired from the LCOM (Lack of COhesion in Methods) metric defined by Chidamber and Kemerer [Chidamber91]. Many authors have redefined the LCOM metric as referenced in the following paragraphs.

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 = | connected components of G |
LCOM4	Like LCOM3, where graph G additionally has an edge between vertices representing methods Mi and Mj, if Mi invokes Mj or vice versa.
Co	Connectivity. Let V be the vertices of graph G from LCOM4, and E its edges. Then $Co = 2 . \frac{\lvert E\rvert - (\lvert V\rvert - 1)}{(\lvert V\rvert - 1) . (\lvert V\rvert - 2)}$
LCOM5	Consider a set of methods {Mi} (I = 1, … , m) accessing a set of instance variables {Aj} (j = 1, …, a). Let μ (Aj) be the number of methods that reference Aj. Then $LCOM5 = \frac{(1/a)\, \Sigma_{1 \leq j \leq a}\, \mu (Aj) - m}{1 - m}$
Coh	A variation on LCOM5. $Coh = \frac{\Sigma_{1 \leq j \leq a}\, \mu (Aj)}{m . a}$
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.

The existing cohesion metrics are based on either instance variables usage or sharing of instance variables. A class is more cohesive, as stated in [Chae00], when a larger number of its instance variables are referenced by a method (LCOM5 [Henderson-Sellers96], Coh [Briand98]), or a larger number of methods pairs share instance variables (LCOM1 [Chidamber91], LCOM2 [Chidamber94], LCOM3 [Li93], LCOM4 [Hitz95], Co [Hitz95], TCC and LCC [Bieman95]). Table 1 gives a summary of their definition. Chidamber and Kemerer propose the 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 [Badri95, Briand98, Chae98, Henderson-Sellers96, Hitz95]. LCOM2 of many classes are set to be zero although different cohesions are expected [Basili96].

Li and Henry redefine LCOM in [Li93]. LCOM3 is defined 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 [Hitz95]. 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. There is also an edge between vertices representing methods M_i and M_j, if M_i invokes M_j or vice versa. Hendersen-Sellers propose LCOM5 in [Henderson-Sellers96] 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 [Briand98].

Bieman and Kang propose TCC (Tight Class Cohesion) and LCC (Loose Class Cohesion) as cohesion metrics [Bieman95]. They also consider the methods pairs using instance variables in common. In their approach, 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 percentage of methods pairs, which are either directly or indirectly related.

The major existing class cohesion metrics attempt to quantify the cohesion of a class by taking into account only the interactions among methods and instance variables of a class. This type of criterion constitutes, in our opinion, a restrictive way of capturing the cohesion of classes. We note, in many situations, that methods of a class may be related together without sharing any 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 their cohesion [Aman02, Chae00, Kabaili0]. If we consider the example given in figure 1, according to the existing cohesion metrics presented in Table 1, the methods M_1 and M_2 are related by sharing the attribute A_2. The method M_3 is not related to the two other methods despite the fact that M_3 shares directly with M_2 the private (or protected) method M_4. The direct relation between the methods M_2 and M_3 in the one hand and the indirect relation between the methods M_1 and M_3 in the other hand are not captured. It should be difficult to split this class in this case.

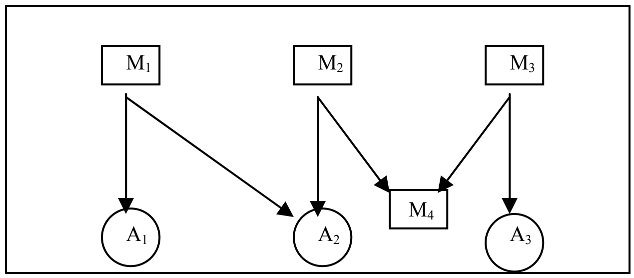

Figure 1 – Connected members.

3 CONNECTIVITY BETWEEN METHODS

Two methods can be connected in many ways. The adopted approach for the estimation of class cohesion is based on different relationships that may exist between its methods. It takes into account several ways of capturing the functional cohesion of the class, by focusing on the proposed cohesion criteria: *Attributes Usage Criterion* and *Methods Invocation Criterion*.

Attributes Usage Criterion (C_A)

Consider a class C. Let $A = \{A_1, A_2, \ldots, A_a\}$ be the set of its attributes and $M = \{M_1, M_2, \ldots, M_n\}$ be the set of its methods. Let UA_{Mi} be the set of all the attributes used directly or indirectly by the 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 M_j of the class that is invoked directly or indirectly by M_i. There are n sets $UA_{M1}, UA_{M2}, \ldots, UA_{Mn}$. Two methods M_i and M_j are directly related by the *UA relation* if $UA_{Mi} \cap UA_{Mj} \neq \emptyset$. It means that there is at least one attribute shared (directly or indirectly) by the two methods.

Methods Invocation Criterion (C_M)

Consider a class C. Let $M = \{M_1, M_2, \ldots, M_n\}$ be the set of its methods. Let IM_{Mi} be the set of all the methods of the class C, which are invoked directly or indirectly by the method M_i. A method M_j is called directly by a method M_i, if M_j appears in the body of M_i. A method M_j is indirectly called by a 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 IM_{M1}, $IM_{M2}, \ldots, IM_{Mn}$. Two methods M_i and M_j are directly related by the *IM relation* if IM_{Mi}

$\cap IM_{Mj} \neq \emptyset$. We also consider that M_i and M_j are directly related if $M_j \in IM_{Mi}$ or $M_i \in IM_{Mj}$.

4 CLASS COHESION: A NEW MEASURE

Class cohesion in our approach, as stated initially in [Badri95], refers essentially the relatedness of public methods of a class, which represent the functionalities used by its clients. It is defined in terms of the relative number of related public methods in the class. The others methods of the class are included indirectly through the public methods. Our approach is comparable to the one adopted by Bieman and Kang in [Bieman95].

We have revised our initial definition of class cohesion proposed in [Badri95] 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 [Bieman95] in the other hand. We have also extended this concept to the methods invocation criterion.

Direct relation between methods

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 class (IM relation), or both. It means that: $UA_{Mi} \cap UA_{Mj} \neq \emptyset$ or $IM_{Mi} \cap IM_{Mj} \neq \emptyset$. Consider a class C with PUM = $\{M_1, M_2, \ldots, M_n\}$ the set of its public methods. The maximum number of public methods pairs, as stated in [Badri95, Bieman95], is $n * (n - 1) / 2$.

Consider an undirected graph G_D, 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 public methods pairs, which are directly (as defined below) related. The LCC_D (Lack of Cohesion in the Class) metric of the class C is then given by: $LCC_D = 1 - DC_D \in [0, 1]$.

Indirect relation between methods

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 in [Bieman95], 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_1 is indirectly connected with a method M_k if there is a sequence of methods $M_1, M_2, M_3, \ldots, M_k$ such that M_i is directly connected to M_{i+1} (i= 1, k-1).

Consider now an undirected graph G_I, 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 of the graph G_D). Let E_I be the number of edges in the graph. The degree of cohesion in the class C based on the direct and indirect relations between its public methods 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 lack of cohesion in the class C is then given by: $LCC_I = 1 - DC_I \in [0, 1]$.

The new definition that we propose for class cohesion assessment seems to be more appropriate than the others, particularly the ones supposed taking into account the interactions between methods. It allows capturing more properties of classes, particularly, in terms of connections between methods. Two public methods can be related by calling directly or indirectly, for instance, private (or protected) methods, which do not use any attribute of the class. 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 assessment, we performed a case study on several systems. In the following sections, we first present the selected test systems for the experiment. Then, we present the obtained results and a discussion of these results.

Selected systems

As a first experimentation of our approach and to achieve significant and general results, we have chosen several systems. Our 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 is designed for migrating code written in old languages to newer ones. System-2 allows a company to maintain a website. System-3 is an implementation of Sun's Java Server Pages. System-4 provides a collection of index structures, query operators and algorithms allowing performance evaluation of new query processing developments. System-5 offers functionality to load, analyze, process and save pixel images. Finally, System-6 is a Java-based build tool, with functionalities similar to the Unix Make utility.

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 (constructor, destructor) are removed in our approach.

	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.

Environment

The developed environment for the computation of the selected metrics is composed of several tools. In its actual version, a Java parsing tool (www.antlr.org) that we have extended, parses the test system source code. The extracted information contains 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 developed. The obtained results are transferred into Excel for statistical processing (Figure 1). 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).

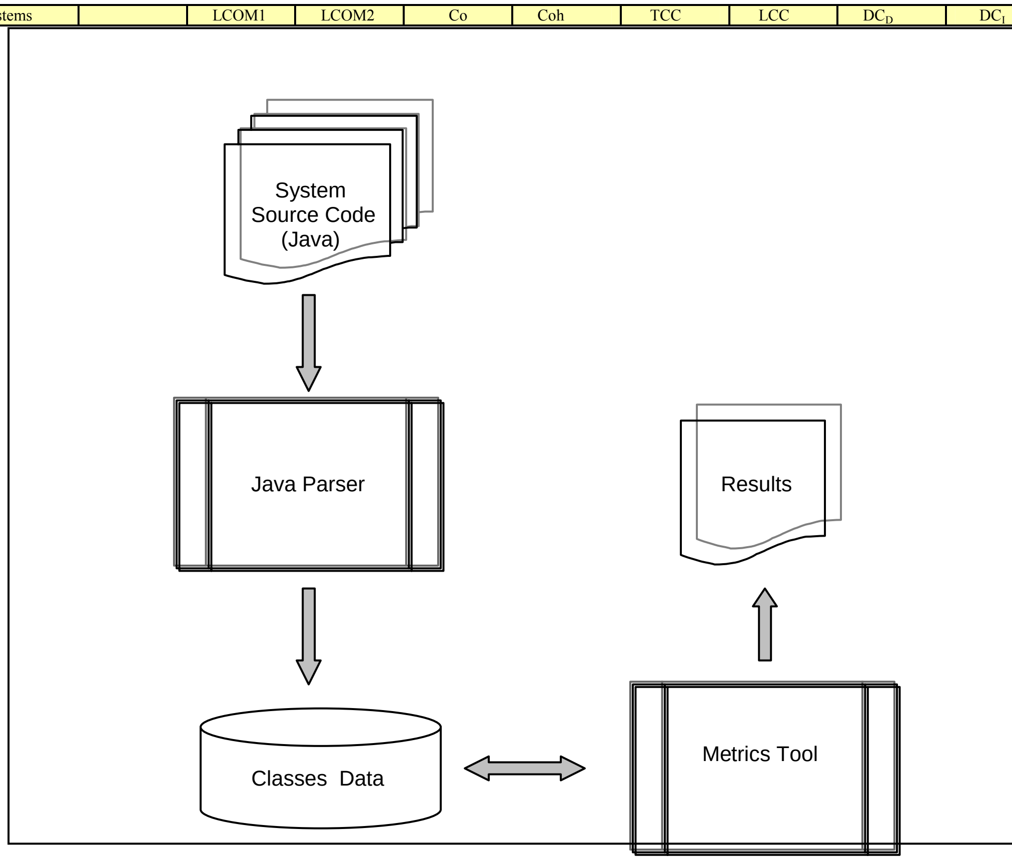

Figure 1 - Metrics calculation Process.

Results

We measured class cohesion values for the 6 selected systems. Table 3 provides the descriptive statistics for all test systems. 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 methods invocation. Co considers only the direct interactions between methods. TCC and LCC metrics are supposed to take into account implicitly the interactions between methods. The problem with these metrics is that many interactions between methods, which do not share any instance variables, 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_D	DC_I
System1	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

System2	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
System3	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
System4	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
System5	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
System6	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.

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. Figure 2 shows the mean values of the normalized metrics for 3 systems. Figure 3 shows the distribution of the cohesion values of all the classes of the system-4 according to the TCC and DC_D metrics. These results indicate that our metrics capture an additional aspect of properties of classes. This is due, in our opinion, to the combination of the proposed criteria. This aspect will be discussed and validated in the next section. 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 raison 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 selected test systems are not cohesive. However, System 3 is strongly cohesive (DC_I = 0.805) compared to the others systems.

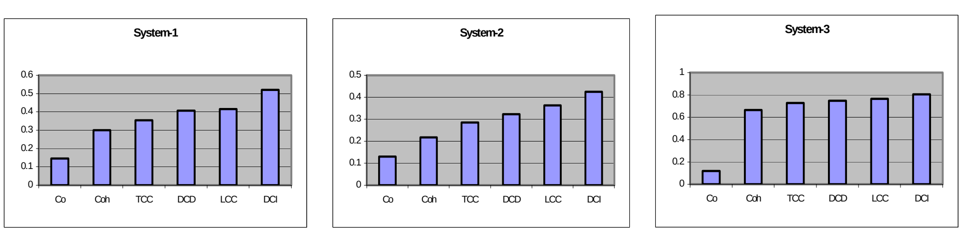

Figure 2 - Mean values of the normalized metrics.

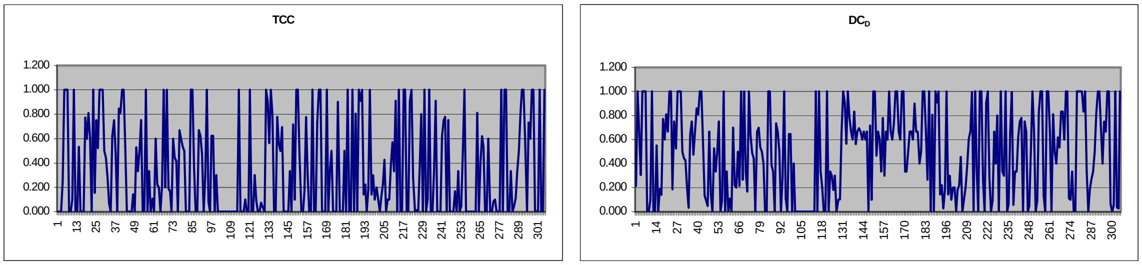

Figure 3 - Cohesion values of Systems-4 classes.

Validation

In this section we are interested in comparing the results of DC_D by considering only the first criterion and DC_D by combining the two proposed criteria. Our goal is to demonstrate that the second metric (including the two criteria) 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 [Bieman95]. The same principle may be applied to the DC_I metric.

Let DC_D (C_A) be the degree of cohesion by considering only the first criterion (*attributes usage criterion*). Let DC_D (C_A 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 $DC_D(C_A$ and $C_M)$ and $DC_D(C_A)$. If there is no difference between the values of $DC_D(C_A)$ and $DC_D(C_A$ and $C_M)$, then the population mean of the differences should be significantly zero. We collected the data for the two metrics from the six 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* [Hines03]).

Let μ_1 be the mean value of $DC_D(C_A$ and $C_M)$. Let μ_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.
$H_1 : \mu_1 > \mu_2$ $DC_D(C_A$ and $C_M)$ is more significant then $DC_D(C_A)$.

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

$$Z = đ / [S_d / \text{sqrt}(n)]$$

With đ : the sample mean value of *Diff,*
S_d : the sample standard deviation of *Diff* and
n : the number of classes of the test system.

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

Table 4 - Cohesion criteria comparison.

Knowing that the number n used in the experiment (for the six test systems) is large, the procedure consists in comparing, for each test system, Z to the normal quantile Z_α (the chosen values for α are 0.05 and 0.01). If the value of Z is greater than the value of Z_α than we will reject the hypothesis H_0: *Diff* $= 0$ and consequently accept the hypothesis H_1: *Diff* > 0. In this case, the statistical test will be significant and we will conclude that the $DC_D(C_A$ 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. We collected the data for the two metrics from the six selected test systems and calculated *Diff* and Z for all test systems. These results are presented in table 4. They show clearly, for all the six test systems, that Z is greater than Z_α. These results show that the $DC_D(C_A$ 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.

6 CONCLUSION AND FUTURE WORK

Class cohesion is considered as one of most important object-oriented software attributes. Cohesion refers to the degree of the relatedness of the members in a class. Members in a class are attributes and methods. Several metrics have been proposed in the literature in order to measure class cohesion in object-oriented systems. These metrics have been defined to capture class cohesion in terms of connections among members within a class. However, several studies have noted that they fail in many situations to properly reflect the cohesion of classes. According to many authors, they do not take into account many characteristics of classes.

We noted that the 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 considered, as stated in many works, that a class cohesion metric have to go beyond this aspect. 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 their cohesion property, 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 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, 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 connected 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). We believe that the present work constitutes an improvement of class cohesion assessment. 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 work we plan to: (1) study in detail the weakly cohesive classes, (2) refine if necessary the proposed criteria for class cohesion assessment, (3) study the proposed metrics by including others aspects of object-oriented design such as inheritance between classes, (4) and work on a metric-based approach for assessing classes responsibilities assignment.

7 ACKNOWLEDGEMENTS

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

REFERENCES

[Aman02] 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.

[Badri95] L. Badri, M. Badri and S. Ferdenache, Towards Quality Control Metrics for Object-Oriented Systems Analysis, *TOOLS* (*Technology of Object-Oriented Languages and Systems*) *Europe'95*, Versailles, France, Prentice-Hall, March 1995.

[Basili96] 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.

[Bieman95] J.M. Bieman and B.K. Kang, Cohesion and reuse in an object-oriented system, *Proceedings of the Symposium on Softwarw Reusability (SSR'95)*, Seattle, WA, pp. 259-262, April 1995.

[Briand98] 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.

[Briand00] 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.

[Booch94] G. Booch, Object-Oriented Analysis and Design With Applications, Second edition, Benjamin/Cummings, 1994.

[Chae98] 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.

[Chae00] 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.

[Chidamber91] S.R. Chidamber and C.F. Kemerer, Towards a Metrics Suite for Object-Oriented Design, *Object-Oriented Programming Systems, Languages and Applications*, Special Issue of SIGPLAN Notices, vol. 26, No. 10, pp. 197-211, October 1991.

[Chidamber94] 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.

[Chidamber98] S.R. Chidamber, David P. Darcy, and C.F. Kemerer, Mangerial use of metrics for object-oriented sofytware : An exploratory analysis, *IEEE Transactions on Software Engineering*, vol. 24, No. 8, pp. 629-639, August 1998.

[ElEmam99] K. El Emam and M. Wacelio, The prediction of faulty class using object-oriented design metrics, *National Research Council of Canada NRC/ERB 1064*, 1999.

[Henderson] B. Henderson-sellers, *Object-Oriented Metrics Measures of Complexity*, Prentice-Hall, 1996.

[Hines03] 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.

[Hitz95] M. Hitz and B. Montazeri, Measuring coupling and cohesion in object oriented systems, *Proceeding of International Symposium on applied Corporate Computing*, pp. 25-27, October 1995.

[Kabaili00] 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*, France, June 2000.

[Kabaili01] 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.

[Larman02] G. Larman, Applying UML and Design Patterns, An introduction to object - oriented analysis and design and the unified process, second edition, Prentice Hall, 2002.

[Li93] W. Li and S. Henry, Object oriented metrics that predict maintainability, *Journal of Systems and Software*, Vol. 23, pp. 111-122, February 1993.

[Pressman01]R. S. Pressman, Software Engineering, A practitioner's approach, Fifth edition, Mc Graw Hill, 2001.

[Yourdon79]E. Yourdon and L. Constantine, *Structured Design*, Prentice Hall, Englewood Cliffs, N.J., 1979.

About the authors

Linda Badri (Linda_Badri@uqtr.ca) is professor of computer science at the Department of Mathematics and Computer Science of the University of Quebec at Trois-Rivières. She holds a PhD in computer science (software engineering) from the National Institute of Applied Sciences in Lyon, France. Her main areas of interest include object-oriented software engineering and Web engineering.

Mourad Badri (Mourad_Badri@uqtr.ca) is professor of computer science at the Department of Mathematics and Computer Science of the University of Quebec at Trois-Rivières. He holds a PhD in computer science (software engineering) from the National Institute of Applied Sciences in Lyon, France. His main areas of interest include object-oriented software engineering, aspect orientation and formal methods.