

# A Design-Based Cohesion Metric for Object-Oriented Classes

Jehad Al Dallal

**Abstract**—Class cohesion is an important object-oriented software quality attribute. It indicates how much the members in a class are related. Assessing the class cohesion and improving the class quality accordingly during the object-oriented design phase allows for cheaper management of the later phases. In this paper, the notion of distance between pairs of methods and pairs of attribute types in a class is introduced and used as a basis for introducing a novel class cohesion metric. The metric considers the method-method, attribute-attribute, and attribute-method direct interactions. It is shown that the metric gives more sensitive values than other well-known design-based class cohesion metrics.

**Keywords**—Object-oriented software quality, object-oriented design, class cohesion.

## I. INTRODUCTION

A popular goal of software engineering is to develop techniques and tools to develop applications that have high quality. Applications that have high quality are more stable and maintainable. In order to assess and improve the quality of an application during the development process, developers and managers use several metrics. These metrics estimate the quality of different software attributes, such as cohesion, coupling, and complexity.

The cohesion of a module refers to the relatedness of the module components. The module that has high cohesion performs one basic function and cannot be split into separate modules easily. Highly cohesive modules are more understandable, modifiable, and maintainable [1].

In object-oriented paradigm, classes are the basic modules. The members of a class are the attributes and methods. Therefore, class cohesion refers to the relatedness of the class members. The class which its members are highly correlated has high cohesion and cannot be split into separate classes easily. The degree of class cohesion gives an indication for the quality of class design; where a highly cohesive class is well designed and a lowly cohesive class is poorly designed.

Several class cohesion metrics have been proposed in the literature. These metrics can be classified into design-based and code-based. Design-based class cohesion metrics require inspecting high-level design artifacts, such as method interfaces [2, 3, 4] to estimate the class cohesion. Code-based

class cohesion metrics e.g., [1, 5, 6, 7, 8, 9, 10, 11], require inspecting the class internal code to measure the class cohesion. Code-based cohesion metrics are more accurate than design-based ones, because they use a finer-grained artifact, which is the code itself. At the code level, all method-method, method-attribute, and attributes-attributes interactions are precisely defined. On the other hand, design-based class cohesion metrics predict cohesion weaknesses early at design level. Detecting class cohesion weaknesses and correcting the class artifacts accordingly late during the implementation phase are much more costly than performing the same tasks early during the design phase. Improving the class cohesion during the design phase saves development time, reduces development costs, and increases the overall software quality.

The proposed design-based class cohesion metrics have several drawbacks. The first drawback is that some design-based metrics are based on poorly studied hypotheses. For instance, in order to increase the accuracy of the measured class cohesion, the design-based metrics attempt to predict the actual interactions that would be implemented during the implementation phase. The method-attribute interactions are not defined at the high-level design phase. Therefore, as in [3] and [14], to predict the method-attribute interactions, the types of method parameters are used instead of the attributes. The hypothesis is that the attributes rely on the method parameters as a basis for the work done by the class. However, this hypothesis is not supported with a strong empirical study. In [3], only 21 C++ classes are studied to support the hypothesis. The second drawback is that some metrics are environmentally dependent. For instance, in [2] a metric is introduced for Ada-object systems. The metric may not be suitable for other environments, such as object-oriented systems. The third drawback is that some key features of object-oriented programming languages, such as inheritance, are not considered in the proposed metrics. The fourth drawback is that, as far as we know, the Unified Modeling Language (UML), a standard language for modeling the object-oriented high-level design, is not used as an artifact to support the measurement of class cohesion in any of the proposed design-based class-cohesion metrics. The fifth drawback is that some proposed metrics, such as CAMC [3] and NHD [4], are not validated against class cohesion metric necessary-properties. The sixth drawback is that some metrics ignore indirect interactions and method invocation interactions. Finally, the research in the area of design-based class cohesion measuring lacks some important empirical

The author would like to acknowledge the support of this work by Kuwait University Research Grant WI03/07.

Jehad Al Dallal is with Department of Information Sciences, Kuwait University, P.O. Box 5969, Safat 13060, Kuwait (e-mail: jehad@cfw.kuniv.edu).

studies to examine the affect of considering the inheritance relationships, method invocation interactions, indirect interactions, and inclusion of the interactions of the special types of methods, such as constructors, destructors, accessors, and modifiers in the metrics.

In this paper, some predefined design-based class cohesion metrics are overviewed. In addition, a new model is introduced to represent the relationship between the parameter types and the attribute types. A novel design-based class cohesion metric that uses the model is introduced and called the Distance Design-based Direct Class Cohesion ( $D_3C_2$ ) metric. The metric considers the method-method, method-attribute, and attribute-attribute direct interactions identified using the Unified Modeling Language (UML) class diagram. The notion of distance between a pair of methods and a pair of attribute types is defined and used as a basis to measure the class cohesion. Several sample cases are used to assess the sensitivity of the metric to model changes comparing to the sensitivity of other well-known design-based class cohesion metrics.

This paper is organized as follows. Section II overviews related work. Section III defines the model used by the introduced metric. Section IV defines the metric, and Section V compares the metric with other metrics in terms of sensitivity. Finally, Section VI provides conclusions and a discussion of future work.

## II. RELATED WORK

In this section, several predefined design-based class-cohesion metrics are overviewed and discussed. In addition, other related work in the area of measuring software cohesion is overviewed. Finally, the specifications of UML class diagram and its relation to this work is briefly illustrated.

### A. Design-based Class-Cohesion Metrics

Several metrics are proposed in the literature to measure the class cohesion during the software high-level design phase. These metrics use different models and different formulas as follows:

#### 1. Ratio of Cohesive Interactions

Briand et al. [2] define a design-based cohesion metric called the Ratio of Cohesive Interactions (RCI) for Ada object-systems. The metric considers only the data to data (DD) and data to subroutine (DS) interactions. In Ada, a type and variables of that type can be defined inside a software part. Briand et al. consider each definition of a variable of a type defined within the software part as a cohesive interaction between the variable and the type. Interactions among variables within subroutines are not considered, because their details are not available during the design phase. The DS interaction occurs if a type defined within the software part matches the type of one of the subroutine parameters, or a variable within the software part is listed in the method parameter list. The RCI metric is defined as the ratio of the number of cohesive interactions of a module to the total

number of possible cohesive interactions. The RCI metric does not take the indirect interactions into account. In addition, Briand et al. consider the inclusion of method invocation interactions and inheritance relations as subjects for future work.

Briand et al. [2] define three properties for cohesion metrics. The first property, called normalization, is that the cohesion measure belongs to a specific interval  $[0, \text{Max}]$ . The second property, called monotonicity, is that adding cohesive interactions to the module cannot decrease its cohesion. The third property, called cohesive modules, is that merging two unrelated modules into one module does not increase the module's cohesion. The total cohesion of split classes is the weighted summation of the cohesion of the individual classes.

#### 2. Cohesion among Methods in a Class

Bansiya et al. [3] propose a design-based class cohesion metric called Cohesion Among Methods in a Class (CAMC). In this metric, only the method-method interactions are considered. The CAMC metric uses a parameter occurrence matrix that has a row for each method and a column for each data type that appears at least once as the type of a parameter in at least one method in the class. The value in row  $i$  and column  $j$  in the matrix equals 1 when  $i$ th method has a parameter of  $j$ th data type, and equals 0 otherwise. In the matrix, the type of the class is always included in the parameter type list, and every method has an interaction with this data type, because every method implicitly has a self parameter. This means that one of the columns is filled entirely with 1s. The CAMC metric is defined as the ratio of the total number of 1s in the matrix to the total size of the matrix. Counsell et al. [4] suggest omitting the type of the class from the parameter occurrence matrix and calculating CAMC using the modified matrix. We refer to this metric as CAMCc.

#### 3. Normalized Hamming Distance (NHD) Metric

Counsell et al. [4] propose design-based class cohesion metric called the Normalized Hamming Distance (NHD). In this metric, only the method-method interactions are considered. The metric uses the same parameter occurrence matrix used by CAMC metric (the type of the class is not considered). The metric calculates the average of the parameter agreement between each pair of methods. The parameter agreement between a pair of methods is defined as the number of places in which the parameter occurrence vectors of the two methods are equal. Formally, the metric is defined as follows:

$$NHD = \frac{2}{lk(k-1)} \sum_{j=1}^{k-1} \sum_{i=j+1}^k c_{ij} = 1 - \frac{2}{lk(k-1)} \sum_{j=1}^l x_j(k-x_j) \quad (1)$$

where  $l$  is the number of columns,  $k$  is the number of rows,  $a_{ij}$  is the number of entries in rows  $i$  and  $j$  in which both are equal to 1, and  $x_j$  is the number of 1s in the  $j$ th column of the parameter occurrence matrix.

### B. Other Cohesion Measuring Related Work

In 1979, Yourdon et al. [12] proposed seven levels of cohesion, which included coincidental, logical, temporal, procedural, communicational, sequential, and functional. The cohesion levels are listed in ascending order according to their desirability. Since then, several cohesion metrics have been proposed for procedural and object-oriented programming languages. Different models are used to measure the cohesion of procedural programs, such as control flow graph [13], variable dependence graph [14], slicing metrics [15], and program data slices [16] [17]. In [18] and [19], cohesion is measured indirectly by examining the quality of the structured designs.

Several code-based class cohesion metrics are proposed in the literature. These metrics are based on the use or share of the class instance variables. In [5], the LCOM metric counts the number of pairs of methods that do not share instance variables. Chidamber [6] proposes another version for the LCOM metric that calculates the difference between the number of method pairs that do and do not share instance variables. Li and Henry [20] use an undirected graph that represents each method as a node and the sharing of at least an instance variable as an edge. The class cohesion is measured as the number of connected components in the graph. This class cohesion is extended in [7] by adding an edge between a pair of methods, if one of them invokes the other. Bieman and Kang [8] propose two class cohesion metrics, TCC and LCC, to measure the relative number of directly connected pairs of methods and relative number of directly or indirectly connected pairs of methods, respectively. These two metrics consider two methods to be connected, if they share at least one instance variable. Cohesion metrics DCD and DCI [9] are similar to TCC and LCC, respectively, but by considering two methods connected when one of them invokes the other. Wang et al. [10] introduce a DMC class cohesion metric based on a dependence matrix that represents the dependence degree among the instance variables and methods in a class. In [11] a class cohesion metric that considers the cardinality of intersection between each pair of methods is proposed. In [21] and [22] class cohesion metrics similar to CAMC but for the method/instance variable matrix are proposed.

### C. UML Class Diagram

UML is a standard language used for modeling object-oriented design. UML 2.0 [23] consists of 13 types of diagrams. In this paper, we are interested in class diagram. The class diagram describes the system's classes and the static relationship between them. The description of a class includes the names and types of the attributes and the names, return types, and parameter types of the methods. Fig 1 shows a sample class diagram for the *AccountDialog* class.

## III. MODEL DEFINITION

In [3] and [4], the parameter occurrence matrix uses the parameter types as bases for their metrics. Their argument is that it is expected that the method uses the attributes that their

types are matching the types of the parameters. The main criticism for this argument is that some methods can have parameters of types not matching the types of the attributes. In this case, methods that share these types are considered cohesive despite the fact that they do not share any attributes. In addition, the parameter occurrence matrix does not inform whether all attributes are used within the methods. Therefore, in some cases, the class is considered fully cohesive despite the fact that some of its attributes are never used by the methods. Since the aim is to predict the share of attributes between the methods, we introduce the Direct Attribute-Type (DAT) matrix that uses the types of the attributes themselves instead of using the types of the method parameters. The matrix is a binary  $k \times l$  matrix, where  $k$  is the number of methods and  $l$  is the number of distinct attribute types in the class of interest. To construct the matrix, the names and return types of the methods and the types of the parameters and the attributes are extracted from the UML class diagram overviewed in Section 2.3. The DAT matrix has rows indexed by the methods and columns indexed by the distinct attribute types, and for  $1 \leq i \leq k, 1 \leq j \leq l$ ,

$$m_{ij} = \begin{cases} 1 & \text{if } j\text{th data type is a type for at least one of} \\ & \text{the parameters or return of } i\text{th method,} \\ 0 & \text{otherwise} \end{cases}$$

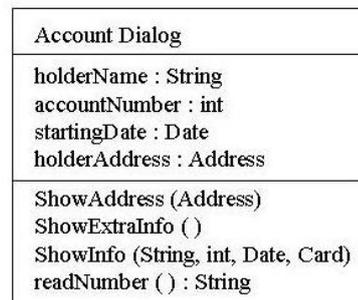


Fig. 1 UML class diagram for *AccountDialog*

The matrix explicitly models the direct attribute-method interactions. A method has a cohesive interaction with an attribute, if the attribute type matches the type of at least one parameter or return of the method. In addition, the matrix implicitly models the method-method and attribute-attribute interactions. A method has a cohesive interaction with another method, if their parameters or returns share the same attribute type. An attribute has a cohesive interaction with another attribute, if their types are shared in a method. This indicates that the method defines an interaction between the two attributes. A binary value 1 in the DAT matrix indicates a cohesive attribute-method interaction. A cohesive method-method interaction is represented in the DAT by two rows sharing binary values 1 in a column. Similarly, a cohesive attribute-attribute interaction is represented in the DAT by two columns sharing binary values 1 in a row. In this matrix, the return type of the method is considered. The reason is that it is

occasionally noticeable that some methods access some attributes not passed as parameters and return results that match the types of the accessed attributes. Consequently, the return type gives an indication for the accessed attributes within methods, and therefore, it should be considered in the class cohesion measurement.

Fig. 2 shows the DAT matrix of the *AccountDialog* class. The matrix is constructed using the information provided by the UML class diagram given in Fig 1. The matrix shows that three of the attribute types are used by *showInfo* method, one of the attribute types is used by *showAddress* method, and one of the attributes is used by *readName* method (as a return type).

	String	int	Date	Address
showInfo	1	1	1	0
showAddress	0	0	0	1
showExtraInfo	0	0	0	0
readName	1	0	0	0

Fig. 2 The DAT matrix for the *AccountDialog* class

#### IV. THE DISTANCE DESIGN-BASED DIRECT CLASS COHESION ( $D_3C_2$ ) METRIC DEFINITION

The  $D_3C_2$  metric uses the DAT matrix to measure the method-method interactions caused by sharing attribute types, the attribute-attribute interactions caused by the expected use of attribute within the methods, and the attribute-method interactions. The different types of cohesion caused by the three types of interactions are referred to as Method-Method through Attributes Cohesion (MMAC), Attribute-Attribute Cohesion (AAC), and Attribute-Method Cohesion (AMC), respectively.

##### A. MMAC and AAC Metrics

The similarity between two items is the collection of their shared properties, and the distance is the opposite [24]. In the context of the DAT matrix introduced in Section 3, the distance between two rows and two columns quantifies the lack of cohesion between a pair of methods and a pair of attributes, respectively. The distance between a pair of rows or columns is defined as the number of entries in a row or column that have different binary values than the corresponding ones in the other row or column. The normalized distance, denoted as  $ndist(i,j)$ , between a pair of rows or columns  $i$  and  $j$  is defined as the ratio of the distance between the two rows or columns to the number of entities  $Y$  in the row or column of the matrix, and it is defined formally as follows:

$$MMAC(C) = 1 - \frac{2}{k(k-1)} \sum_{i=1}^{k-1} \sum_{j=i+1}^k ndist(i, j) \quad (2)$$

where  $\oplus$  is the logical exclusive-or relation (i.e., equals 1 if the two operands have different values). A distance measure has to satisfy three properties: (1) the distance is always

greater than or equal to zero, (2) the distance relation is symmetric, and (3) the distance between an element and itself is equal to zero [24]. For a pair of rows or columns, the minimum number of corresponding entries that have different binary values is zero; that is it when both rows or columns are identical. On the other hand, the maximum number of corresponding entities that have different binary values is equal to the total number of cells in the row or column. This occurs when each corresponding entries have different binary values. As a result, the normalized distance ranges in the interval [0, 1]. Since relation  $\oplus$  is symmetric, the normalized distance between any pair of rows or columns is symmetric. Finally, the exclusive-or of an entry and itself is equal to zero. Therefore, the distance between a row or column and itself is equal to zero. As a result, the metric given in Formula 2 is a distance measure.

Generally, cohesion refers to the degree of similarity between module components. Similarity and distance are complementary measures. As a result, cohesion and distance are complementary measures [24]. Formally, we define cohesion of a pair of methods or attributes as the degree of similarity between them, and it is calculated as follows:

$$C(i,j) = 1 - ndist(i,j) \quad (3)$$

The MMAC is the average cohesion of all pairs of methods, and the AAC is the average cohesion of all pairs of attributes. Formally, using the DAT matrix, the MMAC of a class  $C$  consists of  $k$  methods and  $l$  distinct attribute types is defined formally as follows:

$$MMA(C) = \begin{cases} 0 & \text{if } k=0 \text{ or } l=0, \\ 1 & \text{if } k=1, \\ \frac{2}{k(k-1)} \sum_{i=1}^{k-1} \sum_{j=i+1}^k C(i, j) & \text{otherwise.} \end{cases} \quad (4)$$

Given the fact that the normalized distance between any pair of methods is symmetric and by substituting Formula 3 into Formula 4, the MMAC of class  $C$  is calculated in the case of having more than one method in the class as follows:

$$MMAC(C) = 1 - \frac{2}{k(k-1)} \sum_{i=1}^{k-1} \sum_{j=i+1}^k ndist(i, j) \quad (5)$$

By substituting Formula 2 into Formula 5, the MMAC of class  $C$  is calculated in the case of having more than one method in the class as follows:

$$MMA(C) = 1 - \frac{2}{lk(k-1)} \sum_{i=1}^{k-1} \sum_{j=i+1}^k \sum_{w=1}^l (m_{iw} \oplus m_{jw}) \quad (6)$$

The following metric is an alternative form of the MMAC metric, which facilitates the analysis of the metric and speeds up its computation:

$$MMA(C) = \begin{cases} 0 & \text{if } k=0 \text{ or } l=0, \\ 1 & \text{if } k=1, \\ \frac{\sum_{i=1}^l x_i(x_i-1)}{lk(k-1)} & \text{otherwise.} \end{cases} \quad (7)$$

where  $x_i$  is the number of 1s in the  $i$ th column of the DAT matrix.

Proof: By definition, when  $k=1$  or  $k=0$  and  $l=0$ , Formula 4 and 6 are equal. Otherwise, for the  $i$ th column, there are  $x_i(x_i-1)/2$  similarities between the methods, and therefore, there are  $k(k-1)/2 - x_i(x_i-1)/2$  differences between the methods. By definition,

$$\begin{aligned} MMA(C) &= 1 - \frac{2}{lk(k-1)} \sum_{i=1}^l \left( \frac{k(k-1)}{2} - \frac{x_i(x_i-1)}{2} \right) \\ &= 1 - \frac{2}{lk(k-1)} \left[ \frac{lk(k-1)}{2} - \sum_{i=1}^l \frac{x_i(x_i-1)}{2} \right] \end{aligned}$$

The result follows. ■

Similarly, the AAC of a class C is defined formally as follows:

$$AA(C) = \begin{cases} 0 & \text{if } k=0 \text{ or } l=0, \\ 1 & \text{if } l=1, \\ \frac{\sum_{i=1}^k y_i(y_i-1)}{kl(l-1)} & \text{otherwise.} \end{cases} \quad (8)$$

where  $y_i$  is the number of 1s in the  $i$ th row of the DAT matrix.

For example, using Formula 7, the MMAT for *AccountDialog* class is calculated as follows:

$$MMAT(\text{AccountDialog}) = \frac{2(1) + 1(0) + 1(0) + 1(0)}{4(4)(3)} = 0.042$$

Using Formula 8, the AAC for *AccountDialog* class is calculated as follows:

$$AAC(\text{AccountDialog}) = \frac{3(2) + 1(0) + 0(-1) + 1(0)}{4(4)(3)} = 0.125$$

### B. AMC Metric

The notion of similarity and distance is applicable only when the considered pair is of the same entity. Therefore, the notion of similarity and distance is applicable for pairs of method-method and attribute-attribute, but it is not applicable for pairs of attribute-method, because attributes and methods are of two different entities. In this case, the cohesion is the average number of attribute-method interactions represented in the DAT matrix. In other words, the AMC is the ratio of the number of 1s in the DAT matrix to the total size of the matrix. The AMC of a class C is defined formally as follows:

$$AMC(C) = \begin{cases} 0 & \text{if } k=0 \text{ or } l=0, \\ \frac{\sum_{i=1}^k \sum_{j=1}^l m_{ij}}{kl} & \text{otherwise.} \end{cases} \quad (9)$$

Using Formula 9,  $AMC(\text{AccountDialog}) = 5/16 = 0.313$

### C. $D_3C_2$ Metric

The  $D_3C_2$  metric is not defined if the class has no methods and no attributes. The  $D_3C_2$  metric is defined as the weighted summation of the MMAC, AAC, and AMC metrics. The  $D_3C_2$  of a class C is defined for  $k>1$  and  $l>1$  as follows:

$$D_3C_2(C) = \frac{MP * MMA(C) + AP * AAC(C) + lk * AMC(C)}{MP + AP + lk} \quad (10)$$

where MP is the number of method pairs, and AP is the number of distinct attribute-types pairs. By substituting MP and AP with their formulas in Formula 10 and considering all cases of k and l except when both are equal to 0, the  $D_3C_2$  is more formally defined as follows:

$$D_3C_2(C) = \begin{cases} 0 & \text{if } k=0 \text{ and } l=1, \\ 1 & \text{if } k=1 \text{ and } l=0, \\ \frac{k(k-1)MMA(C) + l(l-1)AAC(C) + 2lkAMC(C)}{k(k-1) + l(l-1) + 2lk} & \text{otherwise.} \end{cases} \quad (11)$$

Using Formula 11, the  $D_3C_2$  for *AccountDialog* class is calculated as follows:

$$D_3C_2(\text{AccountDialog}) = \frac{4(3)(0.042) + 4(3)(0.125) + 2(4)(4)(0.313)}{3(3) + 4(3) + 2(4)(4)} = 0.227$$

## V. SENSITIVITY

Table I shows several patterns for the matrices used by CAMC, NHD, and  $D_3C_2$  metrics. The table shows that the value of the CAMC metric result is the same for classes A and B despite the fact that the intuition informs that class A is more cohesive than class B. The same scenario applies for classes D and C. The NHD metric violates the intuition by giving the same result for classes A, B, and D and by considering class C to be more cohesive than class D. The  $D_3C_2$  metric follows the intuition for all the listed cases. The metric results show that class A is more cohesive than class B, which is expected, because none of the pairs of rows or columns in the matrix representing class B share any commonality, whereas the pairs of columns in the matrix representing class A share some commonalities. Class D is more cohesive than class A, because the pairs of columns and the pairs of rows in the matrix representing class D share some commonalities. Class C is more cohesive than class D, because the matrix representing class C has more pairs of rows sharing some commonalities. Finally, class E is the most cohesive, because its matrix shows that it has the largest number of cohesive interactions among the other matrices. This shows that the  $D_3C_2$  metric is more sensitive than the other two metrics, and it gives more meaningful and

representative results.

## VI. CONCLUSION AND FUTURE WORK

This paper introduces a design-based class cohesion metric that considers three types of interactions: method-method interactions caused by sharing attribute types, attribute-attribute interactions caused by the expected use of attributes within the methods, and attribute-method interactions. The metric uses a matrix constructed using a UML class diagram available at the high-level design phase. The metric uses the distance between pairs of methods and pairs of attributes as bases to compute their degree of similarity.

The introduced metric can be improved in several directions, such as considering indirect interactions and method invocation interactions. In the future, we plan to compare our metric with others empirically. In addition, we plan to introduce a similar code-based class metric and study it empirically.

TABLE I  
 VALUES OF DIFFERENT COHESION METRICS ON 5 SAMPLE CLASSES

Class	Matrix pattern	CAMC <sub>c</sub>	NHD	D <sub>3</sub> C <sub>2</sub>
A		0.2	0.6	0.16
B		0.2	0.6	0.11
C		0.36	0.68	0.29
D		0.36	0.44	0.26
E		0.8	0.6	0.76

## REFERENCES

- [1] Z. Chen, Y. Zhou, and B. Xu, *A novel approach to measuring class cohesion based on dependence analysis*, Proceedings of the International Conference on Software Maintenance, 2002, pp. 377-384.
- [2] L. C. Briand, S. Morasca, and V. R. Basili, *Defining and validating measures for object-based high-level design*, IEEE Transactions on Software Engineering, Vol. 25, No. 5, 1999, pp. 722-743.
- [3] J. Bansiya, L. Etzkorn, C. Davis, and W. Li, *A class cohesion metric for object-oriented designs*, Journal of Object-Oriented Program, Vol. 11, No. 8, pp. 47-52. 1999.
- [4] S. Counsell, S. Swift, and J. Crampton, *The interpretation and utility of three cohesion metrics for object-oriented design*, ACM Transactions on

- Software Engineering and Methodology (TOSEM), Vol. 15, No. 2, 2006, pp.123-149.
- [5] 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, 1991, pp. 197-211.
- [6] S.R. Chidamber and C.F. Kemerer, *A Metrics suite for object Oriented Design*, IEEE Transactions on Software Engineering, Vol. 20, No. 6, 1994, pp. 476-493.
- [7] M. Hitz and B. Montazeri, *Measuring coupling and cohesion in object oriented systems*, Proceedings of the International Symposium on Applied Corporate Computing, 1995, pp. 25-27.
- [8] J. M. Bieman and B. Kang, *Cohesion and reuse in an object-oriented system*, Proceedings of the 1995 Symposium on Software reusability, Seattle, Washington, United States, pp. 259-262, 1995.
- [9] L. Badri and M. Badri, *A Proposal of a new class cohesion criterion: an empirical study*, Journal of Object Technology, Vol. 3, No. 4, 2004.
- [10] J. Wang, Y. Zhou, L. Wen, Y. Chen, H. Lu, and B. Xu, *DMC: a more precise cohesion measure for classes*. Information and Software Technology, Vol. 47, No. 3, 2005, pp. 167-180.
- [11] L. Fernández, and R. Peña, *A sensitive metric of class cohesion*, International Journal of Information Theories and Applications, Vol. 13, No. 1, 2006, pp. 82-91.
- [12] E. Yourdon and L. Constantine, *Structured Design*, Prentice-Hall, Englewood Cliffs, 1979.
- [13] T. Emerson, *A discriminant metrics for module cohesion*, In Proceedings of the 7th International Conference on Software Engineering, 1984, pp. 294-303.
- [14] A. Lakhotia, *Rule-based approach to computing module cohesion*, Proceedings of the 15th international conference on Software Engineering, Baltimore, US, 1993, pp. 35-44.
- [15] L. Ott and J. Thuss, *Slice based metrics for estimating cohesion*, Proceedings of the First International Software Metrics Symposium, Baltimore, 1993, pp. 71-81.
- [16] J. Bieman and L. Ott, *Measuring functional cohesion*, IEEE Transactions on Software Engineering, Vol. 20, No. 8, 1994, pp. 644-657.
- [17] J. Al Dallal, *Using distance measurement for software functional cohesion*, IASTED International Conference on Software Engineering (SE 2005), Innsbruck, Austria, 2005, pp. 132-137.
- [18] D. Troy and S. Zweben, *Measuring the quality of structured designs*, Journal of Systems and Software, 2, 1981, pp. 113-120.
- [19] J. Bieman and B. Kang, *Measuring design-level cohesion*, IEEE Transactions on Software Engineering, Vol. 24, No. 2, 1998, pp. 111-124.
- [20] W. Li and S.M. Henry, *Maintenance metrics for the object oriented paradigm*. In Proceedings of 1st International Software Metrics Symposium, Baltimore, MD, 1993, pp. 52-60.
- [21] B. Henderson-sellers, *Object-Oriented Metrics Measures of Complexity*, Prentice-Hall, 1996.
- [22] L. C. Briand, J. Daly, and J. Wuest, *A unified framework for cohesion measurement in object-oriented systems*, Empirical Software Engineering - An International Journal, Vol. 3, No. 1, 1998, pp. 65-117.
- [23] D. Pitone and N. Pitman, *UML 2.0 in a Nutshell*, O'Reilly Media, Inc., 2nd edition, 2005, pp. 234.
- [24] F. Simon, S. Loffler, C. Lewerentz, *Distance based cohesion measuring*, Proceedings of the FESMA'99, Amsterdam, Netherlands, 1999, pp. 69-83.