Article accepted for Publication and to Appear in the

***Journal of Object-Oriented Programming* 1998.**

A Class Cohesion Metric For Object-Oriented Designs

Jagdish Bansiya, Letha Etzkorn, Carl Davis and Wei Li

The University of Alabama in Huntsville

jbansiya@cs.uah.edu

Abstract

Cohesion is a measure of relatedness or consistency in functionality of a software component. It is a highly desirably design characteristic because it measures separation of responsibilities, independence of components, and control of complexity. Cohesion has a significant effect on a design's understandability, effectiveness and adaptability. An early evaluation of design components for cohesion can significantly improve the quality of a design, by helping identify and redesign components that have scattered functionality or inconsistencies and that are complex. In this article we present a new metric to evaluate cohesion among methods of a class early in the analysis and design phase. The metric evaluates the consistency (focus) of methods in a class' interface using the parameter lists of the methods. This metric can be applied on class declarations that only contain method prototypes (method name and parameter types). The effectiveness of the metric is validated and demonstrated by correlating its results with existing measures of cohesion such as the Lack of Cohesion Among Methods Metric (LCOM) proposed by Chidamber and Kemerer and later interpreted by Li and Henry, which can be applied only after the implementation of methods of a class. Also, the metric is correlated with the cohesiveness evaluation of classes done by human experts.

Cohesion has long been recognized as a highly desirable property in software components such as methods, classes, and modules because it measures separation of responsibilities, independence of components, and control of complexity. Typically, cohesion refers to the collaboration among the constituents of a software component. All parts of a component should directly or indirectly contribute to the services provided by the component. When a component includes parts that do not relate to the logical function of the component, the parts represent an unrelated grouping of operations and the component is termed to have low cohesion. For example, in a procedural language, a function can be a component, the statements

in the function can be the parts. In object-oriented paradigm, a class can be a component, the data attributes and the methods are parts.

This definition of relatedness or consistency with respect to the functionality of a component is effective and useful in controlling complexity of components and therefore aids in the development of simple and easy to understand systems. Because cohesion reflects how well complexity has been controlled in software design and the integrity of functionality in a module, it has a significant effect on a component's reusability, understandability, effectiveness, and adaptability[1].

Functions and modules have been the most frequently used components for the study of cohesion in the procedural paradigm, where a module is typically a collection of related functions. In an object-oriented paradigm, classes are the most basic static components of a system. A class describes a template behavior using a set of data attributes and a set of related methods.

Designing a cohesive class is an important consideration for every object-oriented developer. Recognizing the cohesiveness or lack of it is important for anyone who maintains object-oriented systems. While an expert software analyst may not have difficulty recognizing cohesiveness or the lack of cohesiveness in software components, it is difficult for the general population of developers and analysts to do so. Software product metrics that evaluate components for cohesiveness can be of great value to most designers, developers, and managers in identifying the cohesiveness of the classes in an object-oriented design.

The existing cohesion metrics such as "data slicing"[2], which measures cohesiveness of functions and Lack of Cohesion in Methods (LCOM)[3,4,5,6] which measures lack of cohesiveness in classes, can only be applied after the implementation of the component. While these metrics can be helpful in identifying low-cohesion components, they do not help in preventing the creation of a component that is not cohesive during design. An early evaluation of cohesion in component

design can help to identify and redesign components that have low cohesion, thereby improving the quality of the component.

In this article we present a new metric for evaluating the cohesion of a class. The metric, Cohesion Among Methods of Classes (CAMC), evaluates the relatedness of methods in the interface of a class using the parameter lists defined for the methods. The metric can be applied earlier in the development than can traditional cohesiveness metrics because it relies only on method prototypes declared in a class. The effectiveness of the metric is validated and demonstrated by correlating with the existing LCOM metrics proposed by Chidamber and Kemerer and interpreted by Li and Henry, and with a team of highly trained software experts using a set of 17 classes from a variety of commercial projects.

CLASSES IN OBJECT-ORIENTED DESIGNS

Object-oriented systems are developed around class hierarchies. A class hierarchy is a generalization-specialization structure in which there is a systematic refinement of ideas, concepts, attributes, and operations using multiple levels, with each level capturing common attributes and operations of the idea or concept being refined gradually. These generalization-specialization structures, and therefore class hierarchies, represent the process of abstraction (refinement in stages) in a design, which is implemented using the inheritance mechanism. The use of generalization-specialization structures promotes abstraction, it also spreads information across multiple classes and thus can make a design difficult to understand. While inheritance promotes internal reuse and abstraction, it breaks encapsulation of classes, increases class coupling and decreases cohesion of individual classes in the inheritance branch[10]. This breach of encapsulation of classes and the decrease in the cohesiveness of classes can make the classes difficult to understand, reuse, and adapt.

Significant time and effort in the analysis and design of object-oriented systems is spent in the identification of the classes, assignment of responsibilities to the classes, and achieving

appropriate granularity of the classes[11]. The object-oriented analysis and design process is iterative and may require several refinements. A poor design may be created if a class does not represent one cohesive entity, if it has disparate and non-related functionality, or if too many responsibilities are assigned to a class. These classes will exhibit low cohesion and should be identified early in the design process so that they can be corrected.

Poor design can result from the creation of ad-hoc classes, using classes to bundle groups of methods arbitrarily, or the use of a class as a communal blackboard for a variety of unrelated information to be passed in and out of the methods of the class. These practices can lead to the development of highly non-cohesive classes that can severely degrade the quality of components and the whole system. The identification of these low cohesion classes early in the development process is important so that they can be corrected. Therefore, a metric which can objectively evaluate the cohesive property of classes before they are committed to an implementation can be of significant assistance in the development of a good design.

CLASS COHESION

The study of cohesion among methods involves understanding the behavior of the various methods of a class. In a cohesive class, all the methods of the class collaborate to provide the services of the class. The collaboration patterns of methods are reflected by the types of information on which the methods operate. The different types of information that methods process and the different data declarations with which the methods interact define the information access patterns of methods. Cohesion can be studied by looking at the closeness (or relatedness) of the information access patterns in method implementations. The information that is accessed in a method can be in the form of (1) data attributes of the class, (2) local variables declared in the method, and (3) parameters that are passed to the method during the method invocation. The (LCOM) metric proposed by Chidamber and Kemerer is based on the access patterns of methods to data attributes in a class. This paper proposes a cohesion metric that is based on the parameter

access pattern in methods as an alternative to the LCOM metric. A significant advantage of the new metric over LCOM is that the new metric is not dependent on the implementation of the methods of a class and can thus be used during the design phase to measure the cohesiveness of methods in a class.

LACK OF COHESION IN METHODS (LCOM) METRIC

The LCOM metric was one of the six metrics proposed in a draft suite of measurement theory based software metrics in 1991 by Chidamber and Kemerer[3]. The metric was interpreted by Li and Henry[4,7], which was later redefined by Hitz and Montazeri[5]. Chidamber and Kemerer[6] also later proposed a revised definition for LCOM. The multiple interpretations of the LCOM metric can result in differing values for a particular class[5,9]. However, in spite of its many definitions, the LCOM metric is still currently the most widely used metric for measuring cohesiveness of a class.

The LCOM metric uses class data attribute access patterns to compute the lack of cohesion. The metric determines the sets of methods that have one or more attribute accesses in common in the implementation of the methods of a class. The number of sets of non-overlapping attributes gives the LCOM measure for a class. Using the Chidamber and Kemerer revised definition, the LCOM value for completely cohesive classes is zero and can be as high as $\binom{n}{2}$ for a class with '*n*' methods. With the Li and Henry interpretation[3], the LCOM measure is close to one for cohesive classes and can be as high as the number of methods in the class for completely non-cohesive classes. The Li and Henry interpretation of LCOM provides a number that indicates the number of classes with which a non-cohesive class should be replaced with. In an article[9] which compares the various implementations of LCOM, Etzkorn et. al. showed that the revised definition of LCOM by Chidamber and Kemerer and the extensions made by Li and

Henry provide the best interpretation of lack of cohesion in methods of a class. It was also shown that the better implementations of LCOM did not include inherited variables[9].

COHESION AMONG METHODS IN CLASS METRIC (CAMC)

The CAMC metric uses different properties of a class than the LCOM metric. In the CAMC metric, the cohesion in the methods of a class is determined by the types of objects (parameter access pattern of methods) that method's take as input parameters. The metric determines the overlap in the object types of the methods parameter lists. The amount of overlap in object types used by the methods of a class can be used to predict the cohesion of the class. This information is available when all method's prototypes have been defined, well before a class' methods are completely implemented.

The CAMC metric is based on the premise that the parameters of a method reasonably define the types of interaction that methods may implement. While parameters define the external information that is available to the methods of a class, the state of the object maintained by its attributes represents the internal information that is also available to the methods of the class. Since all methods of a class have access to the state of the object, parameters represent the information that can make methods significantly different within a class. If all the methods of a class have access to similar parameter types, then it can be reasonably assumed that the methods process closely related information and thus must be cohesive in terms of the information processed.

The CAMC metric measures the extent of intersections of individual method parameter type lists with the parameter type list of all methods in the class. To compute the CAMC metric value, an overall union (T) of all object types in the parameters of the methods of a class is determined. A set M_i of parameter object types for each method is also determined. An intersection (set P_i) of M_i with the union set T is computed for all methods in the class. A ratio

of the size of the intersection (P_i) set to the size of the union set (T) is computed for all methods. The summation of all intersection sets P_i is divided by product of the number of methods and the size of the union set T, to give a value for the CAMC metric. For a class with 'n' methods :

If M_i is the set of parameters of method i, then

T = Union of M_i, $\forall\ i = 1\ to\ n$

If P_i is the intersection of set M_i with T i.e. $P_i = M_i \bigcap T$ then

$$CAMC = \frac{\sum_{i=1}^{n} |P_i|}{|T| \times n}$$

The metric value ranges between 0 and 1.0. A value of 1.0 represents maximum cohesion and 0 represents a completely un-cohesive class.

In C++, all methods implicitly receive "this" (pointer to the object-class) as the first parameter in a method invocation. Therefore the set T of types will be a non-empty set, that will at least contain the class pointer "this" as a member. Figure 1 gives the design of three classes *Employee, EmployeeNode* and *EmployeeList* as part of an employee information program. Figure 2 shows the CAMC metric calculations for the three classes. Class *EmployeeNode* has a CAMC metric value of 1.0 and is the most cohesive of the three classes because it has only one method. Class *EmployeeNode* with four methods and class *Employee* with five methods have cohesion values of 0.75 and 0.5 respectively.

```
class Employee {

  public:
        Employee () { name = address = 0; }
        double Pay(int no_hours);
        double Tax(double tax_rate, int no_deductions);
        double SetPayRate (double new_pay_rate);
        void   ChangeAddress(char *new_address);
  private:
        char *name;
        char *address;
        double pay_rate;
        int hours_worked;
};

class EmployeeNode {
    public:
         EmployeeNode (Employee * obj,
                              EmployeeNode * _next = 0);
    private:
         Employee *pEmp;
         EmployeeNode *next;
};

class EmployeeList {
    public:
         EmployeeList (void);
         void Add(Employee obj);
         void Delete(Employee obj);
         int Length(void);

    private:
         EmployeeNode *start, *end;
         int length;
};
```

Figure 1. Three C++ Classes Used to Illustrate the Computation of CAMC

The metric gives a value of 1.0 when set T and all sets M_i are identical, i.e. all methods have the same parameter types. The metric gives lower values of CAMC when the methods of a class have different parameter types. The closer the metric value is to zero the greater is the diversity in types of parameters used by methods and thus more likely the class is to be un-cohesive. It has been our observation that CAMC values of 0.35 and greater indicate classes that

are reasonably cohesive. Classes with a CAMC measure of 0.35 and below are the most likely to be un-cohesive.

CAMC Calculations For Class Employee :

$T = \{ \text{this}, \text{char}, \text{int}, \text{double} \}$ and $|T| = 4$

$P_{Constructor} = |\{ \text{this} \} \cap T| = 1$
$P_{Pay} = |\{ \text{this}, \text{int} \} \cap T| = 2$
$P_{Tax} = |\{ \text{this}, \text{double}, \text{int} \} \cap T| = 3$
$P_{ChangePayRate} = |\{ \text{this}, \text{double} \} \cap T| = 2$
$P_{ChangeAddress} = |\{ \text{this}, \text{char} \} \cap T| = 2$

$CAMC_{Employee} = \sum \{1, 2, 3, 2, 2\} / 4 \times 5 = 10 / 20 = 0.5$

CAMC Calculations For Class EmployeeNode :

$T = \{ \text{this}, \text{Employee} \}$ and $|T| = 2$

$P_{Constructor} = |\{ \text{this}, \text{Employee} \} \cap T| = 2$

$CAMC_{EmpNode} = \sum \{ 2 \} / 1 \times 2 = 2 / 2 = 1.0$

CAMC Calculations For Class EmployeeList :

$T = \{ \text{this}, \text{Employee} \}$ and $|T| = 2$

$P_{Constructor} = |\{ \text{this} \} \cap T| = 1$
$P_{Add} = |\{ \text{this}, \text{Employee} \} \cap T| = 2$
$P_{Delete} = |\{ \text{this}, \text{Employee} \} \cap T| = 2$
$P_{Length} = |\{ \text{this} \} \cap T| = 1$

$CAMC_{LinkedList} = \sum \{1, 2, 2, 1\} / 2 \times 4 = 6 / 8 = 0.75$

Figure 2. Calculations of CAMC for the Three Classes in Figure 1

Some alternative implementations of the CAMC metric can exclude the presence of the "this" parameter and/or the inclusion of parameterless methods in the metric computation. The number of methods '*n*' used in the computation in this implementation of the metric will be the number of methods that have one or more parameters declared in the class. While the inclusion of the constructor(s) in the computation of CAMC may have some influence on the metric's cohesion values for classes with small numbers of methods, for classes with a large number of methods the inclusion or exclusion of the constructor is observed to have no significant impact on the class' cohesion measure.

Validation Tests

As part of our validation study, CAMC was statistically correlated with LCOM. The LCOM metric has been shown to effectively predict cohesiveness of classes in several studies[4,6,8,9]. In this study, the CAMC metric was correlated with the revised definition of LCOM by Chidamber and Kemerer (LCOM1) and with the Li and Henry (LCOM2) interpretation of the LCOM metric. Since the inclusion or exclusion of constructors of a class can give different results in LCOM implementations, the correlation was validated with both implementations. Also, the CAMC metric's assessment of class cohesion was compared with a cohesiveness assessment of the classes provided by a team of highly trained domain experts. The following hypothesis and test were used in the validation.

Hypothesis 1

$H_0 : \rho = 0$ There is no significant correlation between CAMC and LCOM.

$H_1 : \rho \neq 0$ The CAMC metric can predict cohesion among methods as measured by LCOM.

$\alpha = 0.5$

Hypothesis 2

$H_0 : \rho = 0$ There is no significant correlation between CAMC and expert evaluation of cohesion.

$H_1 : \rho \neq 0$ The CAMC metric can predict cohesion among methods as assessed by experts.

$\alpha = 0.5$

Hypothesis 1 Test

The hypothesis, that CAMC should correlate well with LCOM, is based upon our observations and empirical understanding that within method implementations, parameter instances of a method interact with attributes of the class in method statements such as assignments and expressions. Since LCOM bases its measure of the attributes used in the same statements in which parameters are also used, the CAMC measure based on the use of parameters in a method's implementation should closely relate with LCOM.

Metric measures for the four LCOM interpretations and CAMC are shown in Table 1 for 17 classes drawn from three well known graphical user interface packages. These classes were chosen to closely reflect comparable capabilities in the different packages. The experts column shows the average ranking of the cohesion of each class by an external evaluation team on a scale from 0 to 1.

Spearman's rank correlation (r_s) was used to determine the correlation for the non-parametric data in Table 1. The correlation coefficient, r_s, is a measure of the ability of one rank-variable to predict the value of another rank-variable. If E_1 and E_2 are two independent evaluations of '*n*' items that are to be correlated, then the values of E_1 and E_2 are ranked (either increasingly or decreasingly) from 1 to '*n*' according to their relative size within the evaluations. For each E_1, E_2 pair in the relative rankings, the difference in the ranks '*d*' is computed. The sum of all the d^2's, denoted $\sum d^2$ is used to compute r_s using the formula :

$$r_s = 1 - \frac{6\sum d^2}{n(n^2-1)} \qquad -1.00 \le r_s \le +1.00$$

Class	DESIGN CAMC	Experts	LCOM1 with Constructor	LCOM1 without Constructor	LCOM2 with Constructor	LCOM2 without constructor
1	1	0.86	1	0	2	1
2	0.93	0.89	4	3	3	3
3	0.87	0.92	13	10	5	5
4	0.80	0.75	28	21	8	7
5	0.58	0.79	34	34	8	8
6	0.31	0.70	66	49	10	9
7	0.40	0.71	91	78	14	13
8	0.39	0.61	76	66	10	10
9	0.36	0.75	287	276	18	18
10	0.38	0.82	28	15	8	6
11	0.28	0.65	136	120	17	16
12	0.23	0.54	435	406	30	29
13	0.24	0.57	323	298	25	24
14	0.17	0.68	343	321	24	24
15	0.30	0.54	1142	1118	37	44
16	0.67	0.69	378	351	27	26
17	0.32	0.73	24	16	6	5

Table 1. CAMC, Expert, LOCM1 and LCOM2 Cohesion Measures of 17 Classes

Table 2 shows the relative rankings of the 17 metric values within each column of Table 1. While the CAMC values and expert's scores are ranked based on a decreasing value of the metrics, the LCOM metric values are ranked based on an increasing value of the metrics because LCOM measures lack of cohesion (lower values indicate better cohesion) rather than presence of cohesion among methods.

The Spearman's rank correlation coefficient, r_s is computed between CAMC and the four LCOM metrics, and CAMC and the evaluation team's ranking. Table 3 shows the computed values of the correlation coefficient r_s for the CAMC metric with experts and the four LCOM implementations for all 17 classes, and the first 15 classes in Table 2. While an $r_s \approx 0.6$ is

computed between CAMC and the four LCOM metrics when all the 17 classes are used, a stronger correlation of $r_s \approx 0.82$ is observed when the two fringe classes 16 and 17, for which the CAMC and LCOM disagree, are excluded from the correlation.

Class	DESIGN CAMC	Experts	LCOM1 with Constructor	LCOM1 without Constructor	LCOM2 with Constructor	LCOM2 without constructor
1	1	3	1	1	1	1
2	2	2	2	2	2	2
3	3	1	3	3	3	3
4	4	6	5	6	5	6
5	6	5	7	7	6	7
6	12	10	8	8	8	8
7	7	9	10	10	10	10
8	8	14	9	9	9	9
9	10	7	12	12	12	12
10	9	4	6	4	7	5
11	14	13	11	11	11	11
12	16	17	16	16	16	16
13	15	15	13	13	14	13
14	17	12	14	14	13	14
15	13	16	17	17	17	17
16	5	11	15	15	15	15
17	11	8	4	5	4	4

Table 2. Relative Ranking of the 17 Classes Based on the Metric Values

For a sample size of 17 and $\alpha = 5\% (0.05)$, the Spearman's cutoff for accepting H_0 is 0.48. Since the computed r_s in each of the correlations is well above the cutoff, the null hypothesis H_0, of no correlation between CAMC and LCOM, is rejected, and the alternate hypothesis that the CAMC metric is a predictor of cohesion as would be measured by LCOM is accepted. Figure 1 shows the plot of CAMC rankings of the first 15 classes with the LCOM2 rankings of the classes. The figure indicates a close relation between all CAMC and LCOM ranks of the classes and thus points to a strong prediction that the CAMC metric can assess lack of cohesion.

CAMC Correlation	Experts	LCOM1 with Constructor	LCOM1 without Constructor	LCOM2 with Constructor	LCOM2 without constructor
Using 17 Classes	0.70	0.59	0.58	0.60	0.58
Using first 15 Classes	0.73	0.84	0.82	0.85	0.82

Table 3. Correlation Values of CAMC Rankings with Experts, LCOM1 and LCOM2

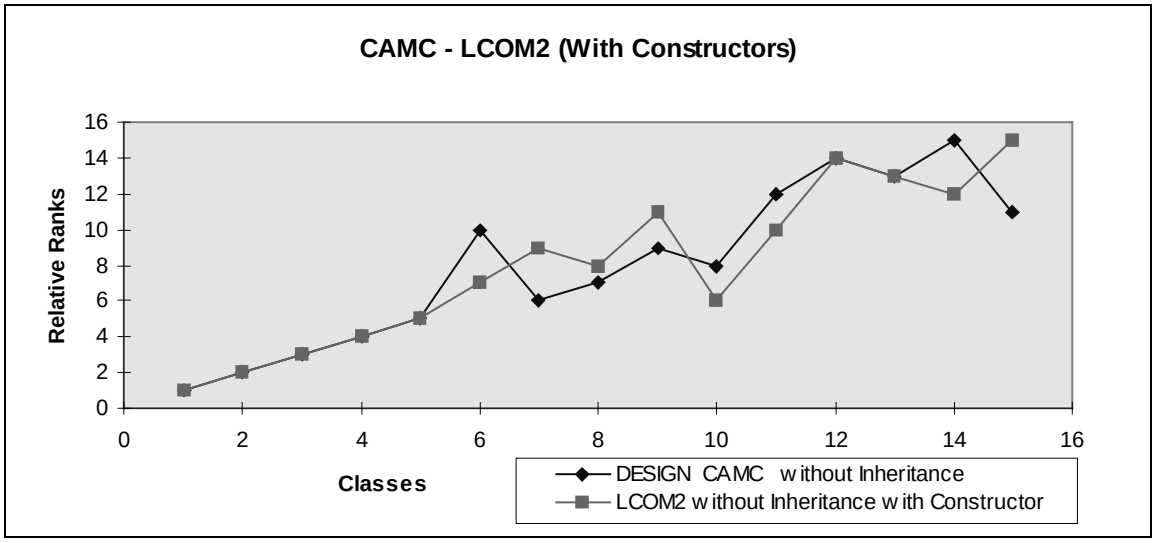

Figure 2. Plot of the CAMC and LCOM2 Rankings of the 15 Classes in Table 3

Hypothesis 2 Test

In Table 3, an r_s of 0.70 and 0.73 is computed between CAMC and expert evaluation of the first 17 and 15 classes of Table 1 respectively. Since the computed value of r_s between CAMC and the experts exceeds the cutoff of 0.48, the null hypothesis is rejected and the alternate hypothesis that the CAMC metric's assessment of cohesion significantly agrees with expert evaluation of cohesion among methods of a class is accepted.

Expert - LCOM Correlation	LCOM1 with Constructor	LCOM1 without Constructor	LCOM2 with Constructor	LCOM2 without Constructor
Using all 17 Classes	0.79	0.81	0.80	0.80
Using first 15 Classes	0.82	0.84	0.84	0.84

Table 4. Correlation Values of Experts Ranking with LCOM1 and LCOM2 Rankings

Table 4 shows the correlation that was computed between expert evaluation of cohesion and the four LCOM implementations. An $r_s \approx .80$ is computed when all 17 classes are correlated, and $r_s \approx .84$ is computed when the first 15 classes are correlated. This indicates that all the four interpretations of LCOM are equally good and any one of these could be used to compute the LCOM measure of a class. Table 5 summarizes the differences between the CAMC and LCOM metrics.

CAMC	LCOM
Measures cohesion among methods of a class.	Measures lack of cohesion in methods of a class.
Relies on class declarations which contain method prototypes for its computation.	Relies on method implementations for its computation.
Can be applied in design phase.	Can be applied in implementation phase.
Measures cohesion based on parameter types defined in methods.	Measures cohesion based on attribute instances of the class used in method implementations.
The metric value is the summation of all individual method parameter types intersected with the union of parameter types from all methods in the class, divided by the number of methods in the class.	The metric value is the number of pairs of member functions without shared instance variables, minus the number of pairs of member functions with shared instance variables[6].
The metric works well in recognizing wild un-cohesion caused by parameters.	The metric works well in recognizing wild un-cohesion caused by attributes.
The metric values are bounded with the range 0 to 1.0. A value 1.0 indicates maximum cohesion.	The metric values are positive integers numbers. A value of 1 indicates the least lack of cohesion[4].
The metric measure is not linear between the range of 0 and 1.0. A class with a CAMC value of 0.5 doesn't indicate that the class is half cohesive or half un-cohesive.	The metric values are not linear on the integer range, i.e., two classes with LCOM measures of 2 and 4 does not mean that the second class is twice as un-cohesive as the first.

Table 5. Comparison between CAMC and LCOM

CONCLUSION

We have proposed a new metric, Cohesion Among Methods of Classes (CAMC), for assessment of design cohesion in a class and validated it by comparing the CAMC metric with the LCOM metric. In addition, independent evaluation of design cohesiveness by a team of software developers also showed positive correlation to the CAMC metric. The main advantage of the new CAMC metric over LCOM is that the metric only requires the definition of the method prototypes in a class for its assessment.

This metric shows considerable promise as an easy and early way to assess the cohesiveness of classes in a design. It needs to be further validated on a wide set of projects from various domains, but initial results show it can be a significant help in improving object-oriented design quality.

References

1. Dormey, G.R. Cornering the Chimera, IEEE SOFTWARE, 13(1) , pp. 33-43, 1996.
2. Bieman, J.M. and L.M. Ott. Measuring functional cohesion, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 20(8), August 1994, pp. 644-657.
3. Chidamber, S. R. and C.F. Kemerer. Towards a metrics suite for object-oriented design, PROCEEDINGS: OOPSLA '91, July 1991, pp. 197-211.
4. Li, W. and S. Henry. Maintenance metrics for the object-oriented paradigm, PROCEEDINGS OF THE FIRST INTERNATIONAL SOFTWARE METRICS SYMPOSIUM, May 21-22, 1993, pp. 52-60.
5. Hitz, M. and B. Montazeri. Chidamber and Kemerer's Metrics Suite: a measurement theory perspective, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 22(4), April 1996, pp. 267-271.
6. Chidamber, S. R. and C.F. Kemerer. A metrics suite for object-oriented design, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 20(6), June 1994, pp. 476-493.

7. Li, W., S. Henry, D. Kafury, and R. Schulman. Measuring object-oriented design, JOURNAL OF OBJECT-ORIENTED PROGRAMMING, July/August 1995, pp. 48-55.

8. Basili, B., L. Briand, and W.L. Melo. A validation of object-oriented metrics as quality indicators, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 22(10), October 1996, pp. 751-761.

9. Etzkorn, L., C. Davis, and W. Li. A Practical Look at the Lack of Cohesion in Methods Metrics, JOURNAL OF OBJECT-ORIENTED PROGRAMMING (to appear in 1998).

10. Gamma, E., R. Helm, R. Johnson and J. Vlissides, DESIGN PATTERNS, Eddison-Wesley, 1994.

11. Booch, G. OBJECT-ORIENTED ANALYSIS AND DESIGN WITH APPLICATIONS, 2nd Edition, Benjamin/Cummings Publishing Company, Inc., 1994.