The Interpretation and Utility of Three Cohesion Metrics for Object-Oriented Design

STEVE COUNSELL and STEPHEN SWIFT
Brunel University
JASON CRAMPTON
University of London

The concept of cohesion in a class has been the subject of various recent empirical studies and has been measured using many different metrics. In the structured programming paradigm, the software engineering community has adopted an informal yet meaningful and understandable definition of cohesion based on the work of Yourdon and Constantine. The object-oriented (OO) paradigm has formalised various cohesion measures, but the argument over the most meaningful of those metrics continues to be debated. Yet achieving highly cohesive software is fundamental to its comprehension and thus its maintainability. In this article we subject two object-oriented cohesion metrics, CAMC and NHD, to a rigorous mathematical analysis in order to better understand and interpret them. This analysis enables us to offer substantial arguments for preferring the NHD metric to CAMC as a measure of cohesion. Furthermore, we provide a complete understanding of the behaviour of these metrics, enabling us to attach a meaning to the values calculated by the CAMC and NHD metrics. In addition, we introduce a variant of the NHD metric and demonstrate that it has several advantages over CAMC and NHD. While it may be true that a generally accepted formal and informal definition of cohesion continues to elude the OO software engineering community, there seems considerable value in being able to compare, contrast, and interpret metrics which attempt to measure the same features of software.

Categories and Subject Descriptors: D.2.8 [**Software Engineering**]: Metrics; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Object-oriented design methods*

General Terms: Measurement, Theory

Additional Key Words and Phrases: Cohesion

1. INTRODUCTION

It is approximately twenty-five years since Yourdon and Constantine [1979] first proposed their seven point ordinal scale for component *cohesion* relating

Authors' addresses: S. Counsell, S. Swift, Department of Information Systems and Computing, Brunel University, Uxbridge, UB8 3PH, England; email: {steve.counsell,stephen.swift}@brunel.ac.uk; J. Crampton, Information Security Group, Royal Halloway, University of London, TW20 0EX, England; email: jason.crampton@rhul.ac.uk.

to the procedural programming paradigm. At one end of their scale *functional* cohesion indicated that a module performed a single well-defined function. At the other end of the scale *coincidental* cohesion indicated that the module performed more than one function, and that those functions were unrelated. The scale they proposed gave an intuitive feel for software cohesion, yet gave very little in the way of a quantifiable and justifiable metric for this feature of software. For the structured paradigm, the informal definition of cohesion was built on the basis of sound programmer practice and experience and underpinned work on measuring module cohesion [Lakhotia 1993]. High cohesion thus reflected use of development techniques known to produce robust and maintainable code.

More recently, and in an object-oriented (OO) sense, the most well known metric for measuring cohesion has been the *lack of cohesion of methods* metric (LCOM) [Chidamber and Kemerer 1994]. It is based on the assumption that a class is cohesive if the same instance variables appear in most or all of the methods in a class. Indeed, the LCOM metric has become the standard by which all other attempts to measure OO cohesion have been compared. This is despite the fact that the LCOM metric is difficult to interpret, gives little insight into the nature of the class itself, and has been refined multiple times from its original form, primarily because of its inadequacies [Briand et al. 1998; Hitz and Montazeri 1996; Li and Henry 1993]. In contrast to the structured paradigm, the OO community has yet to establish a generally accepted informal definition of cohesion.

An alternative approach to that of LCOM is to regard a class as cohesive if the methods of the class use the same set of parameter types [Bansiya et al. 1999]. Any metric based on this alternative definition of cohesion (which we will use throughout the remainder of this article) has a number of key advantages over the approach that the LCOM metric takes.

Firstly, any such metric can be viewed as a design metric, applicable at an earlier stage of development, since the prototypes of a class' methods are available earlier on than the usage of attributes in method bodies. It is also far easier and cheaper to modify a class at the design stage than at later stages of development.

Secondly, the attributes passed as parameters to the method are just as indicative (if not more) of what is actually going on inside the body of the method, since it is the parameters which largely dictate the behaviour of the method. The instance variables rely on parameters as a basis for the work done by the class; we would further contest they give a better feel for the behaviour of the class than consideration of the instance variables themselves (as earlier OO cohesion metrics would have us believe). Moreover, we justify the use of method parameters empirically through a prior visual inspection of the header files of the 21 C++ classes used in this article. The aim of this static code inspection was to establish if any of the methods in those 21 classes failed to access an instance variable of the same type as the parameter passed to it. No occurrence of such a parameter was found; in other words, for every class, every parameter in a method's signature was used in the body of that method. Indeed, assignment to the instance variables of a class is usually achieved in this fashion. A method,

passed a parameter of type T, will usually assign to an instance variable of the same type T; this gives us as much insight into the use of instance variables as LCOM. We do accept, however, that a parameter can be simply used for output purposes (i.e., it is not always used in an assignment operation as such). We also accept that using a sample of significantly more than 21 classes would give us a higher degree of confidence in our claim about parameters and assignment of those parameters to instance variables.

Clearly, it is unlikely that every method will use the same number of parameters each of the same type, so it is necessary to define metrics which can measure the degree of correspondence between the parameter types across each of the methods in a class. Two metrics for measuring cohesion have been proposed recently: the *cohesion among methods in a class* metric (CAMC) Bansiya et al. [1999] and the *normalised Hamming distance* metric (NHD) Counsell et al. [2002]. However, the values of these metrics, which are assumed to represent cohesive classes, have no formal justification. Furthermore, it is not immediately obvious that these metrics even measure cohesion in an appropriate way. In short, the criticisms aimed at the LCOM metric appear to apply equally well to the CAMC and NHD metrics.

The purpose of this article is to subject these metrics to a rigorous mathematical analysis. The analysis will determine whether these metrics have any qualitative meaning given the definition of cohesion above, and what values of these metrics should be considered to represent a cohesive class. We emphasise that the objective of the article is not to introduce the *definitive* cohesion metric or to suggest that there is a *best* way of capturing cohesion. Rather, that with an underlying relational system for cohesion as a basis and in the context of past attempts, there are interesting properties about the approaches taken so far at capturing cohesion *per se*. Mathematically, one approach may exhibit more desirable properties than another. We stress the point that until the OO community can decide on a generally accepted informal definition of cohesion, we will still be unable to claim that one class is more cohesive than another, other than from *our own* interpretation of what constitutes a cohesive class.

In particular, we establish the maximum and minimum values that CAMC and NHD can take. In the case of the NHD metric, these values are sensitive to different arrangements of the same parameters when occurrences of those parameters are represented as matrices. Hence, we find that the NHD has a much richer interpretation and provides a finer level of discrimination between classes than the CAMC metric. We also answer several questions raised by empirical investigations and provide a meaningful interpretation of CAMC and NHD. The analysis also suggests that there may be more appropriate ways of defining a cohesion metric. In particular, we introduce a new cohesion metric SNHD, based on NHD, which attempts to address the shortcomings of the CAMC and NHD metrics. Furthermore, our investigation raises some questions about the suitability of the definition of cohesion.

The following section describes the motivation for our work and is followed in Section 3 by some preliminary definitions and concepts. A detailed, formal comparison of the CAMC and NHD metrics is presented in Section 4. In Section 5 we compare the values of the metrics for several classes in three

different C++ systems. In Section 6 we discuss some of the issues raised in the study and point to its limitations. Finally, in Section 7 we discuss some conclusions and pointers to further work.

2. MOTIVATION AND RELATED WORK

Motivation for the study in this article stems from a number of sources. Firstly, a mathematical comparison of the properties of cohesion metrics is an under-researched area. We see our work as going some way to redressing this deficiency. The OO community has yet to arrive at a consensus about the appropriate measurement of cohesion and so any research of this type is useful. Secondly, identifying common failings or properties of cohesion metrics informs our understanding of OO systems, OO languages and their different traits. Finally, cohesion is just one aspect of software which the software engineering community has tried to capture through metrics. Examination and scrutiny of current cohesion metrics may reveal further relevant research issues of concern to us as researchers and practitioners.

The roots of cohesion go as far back as the early seventies, when Stevens et al. [1974] first began looking at inter-module metrics. Later, Yourdon and Constantine [1979] proposed their seven point ordinal scale for component cohesion. Both of these studies related to a view of cohesion from a procedural programming viewpoint where modules were the key elements by which cohesion was measured. More recently, the focus of attempts to capture cohesion has switched from the procedural to the object-oriented paradigm, and the notion of class cohesion has superceded that of module cohesion. The best known attempt at evaluating cohesion from an OO perspective is the LCOM metric of Chidamber and Kemerer [1991].

The original definition of the metric calculates cohesion according to the use of class attributes in the methods of a class. The metric is based on the principle that an instance variable occurring in many methods of a class causes that class to be more cohesive than one where the same variable is used in very few methods of the class. A high value of the LCOM metric indicates that the methods in the class are unrelated and a low value of the metric indicates that they are related. The LCOM metric suffers from several disadvantages. Firstly, the definition of the metric itself is difficult to understand. Secondly, values produced by the metric are difficult to interpret and give little insight into the nature of the class other than the distribution of attributes therein. Finally, the LCOM metric is an implementation metric. It is widely recognised that a measure of cohesion is required earlier in the development process (that is, at design time).

Several researchers have proposed extensions to the LCOM metric [Bieman and Ott 1994; Henderson-Sellers et al. 1996; Briand et al. 1998] have provided a uniform framework in which LCOM and its derivatives can be evaluated, and which can be extended to evaluate new cohesion metrics. A more general treatment of the properties and validation issues, including cohesion and coupling metrics, in object-based and object-oriented systems can be found in [Basili et al. 1996; Briand et al. 1996, 1999].

The CAMC metric Bansiya et al. [1999] is a cohesion metric that can be evaluated at design time. Bansiya et al. computed CAMC for 17 C++ classes drawn from three well-known graphical user interface packages. It was shown that there exists a strong correlation between the CAMC and LCOM metrics; the implication of this is that assuming the LCOM metric is a good indicator of cohesion, it is preferable to use CAMC because it is easier to collect and is also a design time metric. The authors do not consider the validity of either metric in terms of exactly what software attribute they are trying to quantify. The CAMC metric was also found to correlate with external experts' evaluations of cohesiveness of the same 17 classes, suggesting further that the metric reflects the views on cohesion of system developers. The CAMC metric forms part of the QMOOD suite of metrics which has been used to evaluate the quality of object-oriented systems [Bansiya and Davis 2002]. Interestingly, results herein show a negative correlation between CAMC and LCOM (see Table IV), thus contradicting the earlier results of Bansiya et al. [1999].

Data slicing has also been used as a measure of functional cohesion [Bieman and Ott 1994]. Program slices can be used to assess the frequency of attribute use in programs and the dependency between parts of code and attributes used. Empirical studies have also been undertaken in this area Binkley et al. [2000]. One final approach to evaluating cohesion and coupling has been to employ information theoretic techniques [Allen and Khoshgoftaar 1999]. Investigating the dynamic features of cohesion metrics (i.e., at run-time as opposed to statically) is also a topic of some current research [Mitchell and Power 2004]. Cohesion (and coupling) metrics have also been applied to knowledge-based systems; frames were used a basis for those metrics [Kramer and Kaindl 2004].

The lack of rigour, appeal to measurement theory, and empirical evaluation of cohesion metrics is highlighted well by Briand et al. [1998], which clarifies the terminology associated with the measurement of cohesion and presents a unified framework for measuring cohesion and comparing measures of cohesion.

A variation of the metric proposed in this article was first used by Counsell et al. [2001] to determine the disagreement between four groups of subjects; the subjects were taking part in an experiment in which they had to identify four faults seeded in a requirements document. The metric gave a valuable insight into the characteristics within the individual groups and allowed comparisons between the four groups to be made. Its usefulness for establishing the distance between two randomly selected entities was a key motivation for using it rather than the CAMC metric to measure cohesion in this article.

3. PRELIMINARIES

In the examples and tables throughout this article we use classes randomly chosen from three industrial-sized C++ systems. The three systems were chosen because they represent a variety of different application domains, have evolved to varying degrees, and were originally developed by industrial programmers. The same three systems have also been the subject of a number of previous empirical studies, the most notable of these being Counsell et al. [2004]. It could thus be argued that our understanding of cohesion issues in these three systems

is informed by that prior knowledge obtained about these systems. The systems used were Edge, a graph editor, consisting of approximately 30.8 thousand noncomment source lines (KNCSL) and containing 80 classes; Rocket, a compiler, consisting of 32.4 KNCSL and containing 322 classes; and Et++, a user interface framework, consisting of approximately 56.3 KNCSL and containing 508 classes.

Underlying the capture of any software feature through metrics is the notion of an entity relational system (ERS) [Fenton and Pfleeger 1996], which provides a mapping from the real world attribute of the entity being measured to values representing those attributes in the empirical world. We accept that there are potentially many ways of measuring cohesion and that no single measure can be considered definitive; each has its merits. The ERS we introduce is based on the strong belief that the set of attributes is less susceptible to change than the processes that manipulate those attributes. The purpose of the methods provided by the class is merely to facilitate computation.

We consider a cohesion metric to be a function from the set of classes to the real numbers that assigns a measure of the similarity between the parameter types of the methods for each class. A class X *is more cohesive* than class Y if this function returns a higher value for X when there is greater sharing of parameters between the methods of a class; a lower value is produced by this function when the opposite is true. It is entirely feasible for the function to return the same value for two different classes. It then makes sense to state that one class is twice as cohesive as another class. Such an ERS is also able to distinguish between the cohesiveness of n classes using the same metric. Observations about cohesiveness in the real world rest on the belief that developers always aim to minimise the number of methods in a class and maximise the use of the instance variables across those methods; this is made possible by the parameters of those methods. In the subsequent analysis, we include every type of method, whether public, private, or protected; we also include the class constructors and destructors as if they are normal methods. An interesting extension to our work would be to independently analyse each method type.

3.1 Notation

Throughout, we denote the (i, j)th entry of a matrix M by m_{ij}. We denote the binomial coefficient $\frac{n!}{r!(n-r)!}$ by $\binom{n}{r}$. We use the fact that $\binom{n}{2} = \frac{1}{2}n(n-1)$ extensively. We define $\binom{n}{r} = 0$ if $n < r$. We denote the smallest integer larger than x by $\lceil x \rceil$, and the largest integer smaller than x by $\lfloor x \rfloor$. For example, $\lceil 7/2 \rceil = 4$ and $\lfloor 7/2 \rfloor = 3$.

Given a class C with k methods, the *parameter type list* L is the set of the data types that appear at least once as the type of a parameter in at least one method in the class. We denote the length of L by l.

The *parameter occurrence matrix* O has columns indexed by the members of L and rows indexed by the methods and, for $1 \leqslant i \leqslant k$, $1 \leqslant j \leqslant l$,

$$o_{ij} = \begin{cases} 1 & \text{if the } j\text{th data type occurs as a parameter in the } i\text{th method,} \\ 0 & \text{otherwise.} \end{cases}$$

```
Alert(AlertType, byte, *text = 0, Bitmap *bm = 0);   /* constructor */
~Alert();                                            /* destructor */
VObject *DoCreateDialog();
int Show(char *fmt);
int ShowV(char *fmt, va_list ap);
class Menu *GetMenu();
void InspectorId(char *buf, int sz);
```

(a) Methods.

O	AlertType	byte	Bitmap	char	va_list	int
Alert	1	1	1	0	0	0
~Alert	0	0	0	0	0	0
DoCreateDialog	0	0	0	0	0	0
Show	0	0	0	1	0	0
ShowV	0	0	0	1	1	0
GetMenu	0	0	0	0	0	0
InspectorId	0	0	0	1	0	1

(b) The parameter occurrence matrix.

A	Alert	~Alert	DoCreateDialog	Show	ShowV	GetMenu
~Alert	3					
DoCreateDialog	3	6				
Show	2	5	5			
ShowV	1	4	4	5		
GetMenu	3	6	6	5	4	
InspectorId	1	4	4	5	4	4
	13	25	19	15	8	4

(c) The parameter agreement matrix.

Fig. 1. The `Alert` class and associated matrices.

In other words, the parameter occurrence matrix is a binary $k \times l$ matrix. The ith row in O is called a *parameter occurrence vector* (for method i). A parameter occurrence vector is a bit pattern which indicates the presence of data types in the ith method. An example of a parameter occurrence matrix is shown in Figure 1b. It represents the parameter occurrence matrix for the C++ class `Alert` shown in Figure 1a.

We denote the number of 1s in the ith row by r_i, the number of 1s in the jth column by c_j, and the number of 1s in the parameter occurrence matrix by σ. That is,

$$r_i = \sum_{j=1}^{l} o_{ij}, \quad c_j = \sum_{i=1}^{k} o_{ij}, \quad \sigma = \sum_{i=1}^{k} \sum_{j=1}^{l} o_{ij}.$$

4. COHESION METRICS

In the following section we describe the cohesion metrics which will be used throughout the remainder of the article.

4.1 CAMC

The CAMC metric for a class C is computed as follows [Bansiya et al. 1999]:

$$\text{CAMC}(C) = \frac{1}{kl} \sum_{i=1}^{k} \sum_{j=1}^{l} o_{ij} = \frac{\sigma}{kl}. \tag{1}$$

In other words, CAMC is the average of the entries in the parameter occurrence matrix. For example,

$$\text{CAMC}(\texttt{Alert}) = \frac{1}{7.6}(3+0+0+1+2+0+2) = \frac{8}{42} \approx 0.19.$$

Bansiya et al. [1999] defined the CAMC metric and evaluated it for 17 different C++ classes. In these calculations the type of the class was always included in the parameter type list (since every method implicitly has a "self" parameter). In other words, in this formulation one column of the parameter occurrence matrix would consist entirely of 1s. It could be suggested that the self parameter was included to provide a value for the CAMC metric when no methods of the class had any parameters. We view as undefined any metric value for a class with parameterless methods.

We will denote this calculation of the metric by CAMC_s. For example, the parameter occurrence matrix in Figure 1 would be used to calculate CAMC, rather than CAMC_s.

PROPOSITION 4.1. *For any class C,*

$$\frac{1}{k} \leqslant \text{CAMC}(C) \leqslant 1, \tag{2}$$

$$\frac{l+k-1}{kl} \leqslant \text{CAMC}_s(C) \leqslant 1, \tag{3}$$

$$\text{CAMC}(C) \leqslant \text{CAMC}_s(C), \text{ with equality if } \text{CAMC}(C) = 1. \tag{4}$$

PROOF. Proof of (2): Clearly, the maximum is attained if and only if every entry in the parameter occurrence matrix is 1. The minimum is attained if and only if every column in the matrix contains a single 1 (note that every column must contain at least one nonzero entry, otherwise, the parameter type does not appear in the class and would not be included in the matrix). Hence, the sum of the entries in the matrix is at least l. Therefore, $\text{CAMC}(C) \geqslant l/kl = 1/k$.

Proof of (3): The parameter matrix must include one column in which every entry is 1. The remaining $l-1$ columns must include at least one nonzero entry. Hence, the sum of the entries in the matrix is at least $k+l-1$. The result follows.

Proof of (4): By definition

$$\text{CAMC}(C) = \frac{\sigma}{kl}.$$

To compute CAMC_s we add a column of 1s to the parameter occurrence matrix. Hence,

$$\text{CAMC}_s(C) = \frac{\sigma + k}{k(l+1)}$$

and

$$\text{CAMC}_s(C) - \text{CAMC}(C) = \frac{\sigma + k}{k(l+1)} - \frac{\sigma}{kl} = \frac{kl - \sigma}{l(l+1)} \geqslant 0,$$

since $\sigma \leqslant kl$ by definition. □

PROPOSITION 4.2. *Let $l \leqslant \sigma \leqslant kl$. Then* CAMC *is invariant for all classes whose parameter occurrence matrix has k rows, l columns and contains σ 1s.*

PROOF. By definition, $\text{CAMC} = \sigma/kl$. The result follows immediately. □

Proposition 4.2 means that CAMC is incapable of discriminating between two classes which have the same number of 1s in their respective parameter occurrence matrices. Indeed, CAMC merely calculates the relative frequency of 1s in the parameter occurrence matrix. In this sense, the ratio gives an impression of matrix sparseness. However, we believe that different distributions of parameter types across the methods of a class will lead to different levels of cohesion.

4.1.1 *Interpreting* CAMC . It has been noted that the CAMC metric tends to find smaller classes more cohesive [Counsell et al. 2002]. It is clear from the above analysis that this is a natural consequence of the definition of CAMC, particularly if the "self" parameter is included. Specifically, if the parameter occurrence has few rows and columns, and it is constrained to have at least one 1 in each column (by definition of CAMC) and a column containing 1s (by definition of CAMC_s), then a large number of 1s must be present in the matrix. Hence, the value of both CAMC and CAMC_s will be large. This is not true in general for classes with large numbers of methods and a large parameter type list.

Bansiya et al. [1999] consider classes that have a CAMC value of 0.35 and above to be cohesive classes. However, (3) shows that for $k = 18$ and $l = 2$, for example, CAMC_s is always at least $19/36 > 0.5$. It is difficult, therefore, to have much faith in this interpretation of the CAMC metric.

Table I shows the *minimum* values of CAMC and CAMC_s for different values of k and l (correct to three decimal places). It can be seen that very few minimum values of CAMC_s are below the threshold value of 0.35. One has to wonder, therefore, why values of 0.35 should be considered cohesive.

We also note that the value of both CAMC and CAMC_s increases linearly with the number of 1s in the parameter occurrence matrix and is unaffected by the shape of the matrix. The only impact that the shape of the matrix has is on the threshold value of σ under which CAMC cannot be calculated. For example, suppose $\sigma \leqslant 36$: if $k = 9$ and $l = 4$ then CAMC can be calculated if $\sigma \geqslant 4$ and CAMC_s can be calculated if $\sigma \geqslant 12$; if $k = l = 6$ then CAMC can be calculated if $\sigma \geqslant 6$ and CAMC_s can be calculated if $\sigma \geqslant 11$.

Table I. Minimum Values of CAMC and $CAMC_s$

kl	k	l	σ	CAMC	σ_s	$CAMC_s$
6	2	3	3	0.500	4	0.667
	3	2	2	0.333	4	0.667
12	2	6	6	0.500	7	0.583
	3	4	4	0.333	6	0.500
	4	3	3	0.250	6	0.500
	6	2	2	0.167	7	0.583
24	2	12	12	0.500	13	0.542
	3	8	8	0.330	10	0.417
	4	6	6	0.250	9	0.375
	6	4	4	0.167	9	0.375
	8	3	3	0.125	10	0.417
	12	2	2	0.083	13	0.542
36	2	18	18	0.500	19	0.528
	3	12	12	0.333	14	0.389
	4	9	9	0.250	12	0.333
	6	6	6	0.167	11	0.306
	9	4	4	0.111	12	0.333
	12	3	3	0.083	14	0.389
	18	2	2	0.056	19	0.528

In short, the CAMC metric suffers from the following problems: It cannot distinguish between the cohesion of different matrices with the same value of σ; the use of 0.35 as a threshold for an indicator of cohesion is fundamentally flawed (particularly if the "self" parameter type is included); and the metric is likely to find smaller classes more cohesive, irrespective of their actual properties.

4.2 NHD

The *hamming distance* (HD) metric was introduced by Counsell et al. [2001]. Informally, it provides a measure of disagreement between rows in a binary matrix. The definition of HD leads naturally to the NHD metric Counsell et al. [2002], which measures agreement between rows in a binary matrix. Clearly, this means that the NHD metric could be used as an alternative measure of the cohesion in the sense computed by the CAMC metric.

The *parameter agreement* between methods m_i and m_j is the number of places in which the parameter occurrence vectors of the two methods are equal. The *parameter agreement matrix* A is a lower triangular square matrix of dimension $k-1$, where a_{ij} is defined to be the parameter agreement between methods i and j for $1 \leqslant j < i \leqslant k$, and 0 otherwise. The parameter agreement matrix for the `Alert` class is shown in Figure 1c (the column totals have been shown in the parameter agreement matrix in order to facilitate the calculation of the NHD metric; 0s have been omitted for convenience).

For a class C, the NHD is defined as follows [Counsell et al. 2002]:

$$\mathrm{NHD}(C) = \frac{1}{l\binom{k}{2}} \sum_{j=1}^{k-1} \sum_{i=j+1}^{k} a_{ij} = \frac{2}{lk(k-1)} \sum_{j=1}^{k-1} \sum_{i=j+1}^{k} a_{ij} \tag{5}$$

Hence,

$$\text{NHD } (\texttt{Alert}) = \frac{2}{6.7.6}\left(13+25+19+15+8+4\right) = \frac{168}{252} \approx 0.67.$$

We now state an alternative form of the NHD metric. This leads to a more efficient way of computing NHD and facilitates the analysis of its properties.

PROPOSITION 4.3. *For any class C,*

$$\text{NHD} = 1 - \frac{2}{lk(k-1)} \sum_{j=1}^{l} c_j(k-c_j).$$

where c_j is the number of 1s *in the jth column of the parameter occurrence matrix.*

PROOF. For the jth parameter, there are $c_j(k-c_j)$ disagreements between the methods. Hence, there are $\binom{k}{2} - c_j(k-c_j)$ agreements between the methods. By definition,

$$\text{NHD} = \frac{1}{l\binom{k}{2}} \sum_{j=1}^{l} \left(\binom{k}{2} - c_j(k-c_j) \right) = \frac{1}{l\binom{k}{2}} \left(l\binom{k}{2} - \sum_{j=1}^{l} c_j(k-c_j) \right).$$

The result follows. □

In other words, to calculate NHD we compute the sum of the disagreements between methods over all parameters and subtract from 1. For example, the parameter occurrence matrix has a single 1 in the first column. Therefore, there will be 6 disagreements between methods on this parameter. In the fourth column there are three 1s. Therefore, there will be 4.3 disagreements between methods on this parameter. Recalculating NHD for `Alert`, we have

$$\text{NHD} = 1 - \frac{2}{6.7.6}\left(6+6+6+4.3+6+6\right) = 1 - \frac{84}{252} \approx 0.67.$$

Of course, Proposition 4.3 provides a method for computing NHD that does not require the construction of the parameter agreement matrix. Hence, the NHD metric can be computed in $\mathcal{O}(l)$ rather than $\mathcal{O}(k^2)$ time.

Remark 4.1. Unlike the metric CAMC, the metric NHD can distinguish between different parameter occurrence matrices with the same number of 1s. If we admit that certain parameter occurrence matrices with σ 1s have a higher cohesion than other matrices with σ 1s, we conclude that the NHD metric is a more meaningful measure of cohesion than the CAMC metric.

THEOREM 4.1. *Given a parameter occurrence matrix that contains σ 1s, $l \leqslant \sigma \leqslant kl$,*

$$\text{NHD}_{\min} \leqslant \text{NHD} \leqslant \text{NHD}_{\max},$$

where

$$\text{NHD}_{\min} = 1 - \frac{2}{lk(k-1)}\Big(q(d+1)(k-d-1) + (l-q)d(k-d)\Big), \tag{6}$$

$$\text{NHD}_{\max} = 1 - \frac{2}{lk(k-1)}\Big((r+1)(k-r-1) + (l-c-1)(k-1)\Big), \tag{7}$$

with

$$d = \left\lfloor \frac{\sigma}{l} \right\rfloor, \quad \sigma \equiv q \mod l, \quad c = \left\lfloor \frac{\sigma - l}{k-1} \right\rfloor, \quad \sigma - l \equiv r \mod k - 1.$$

PROOF. The graph of $c(k-c)$, $0 \leqslant c \leqslant k$ is a parabola which takes its maximum value when $c = k/2$ and is 0 when $c = 0$ or $c = k$. By definition of the parameter occurrence matrix, $c_j > 0$, $1 \leqslant j \leqslant l$. In other words, the sum of the disagreements is maximised (and NHD minimised) when the 1s are distributed as evenly as possible between each of the l columns. In this case, there are $d+1$ 1s in q columns and d 1s in $l-q$ columns. The value of $\text{NHD}_{\min}$ follows.

Similarly, the sum of the disagreements is minimised when the first row of the parameter occurrence matrix consists of 1s with the remaining $\sigma - l$ 1s distributed among the fewest number of columns. This can be done by filling c columns with 1s and putting the remaining r 1s in another column. The remaining $l-c-1$ columns contain a single 1. The value of $\text{NHD}_{\max}$ follows. □

Let $\sigma = 8, k = 7$ and $l = 6$ as in the `Alert` class. The matrices $O_{\min}$ and $O_{\max}$ show parameter occurrence matrices that minimise and maximise the value of NHD.

$O_{\min} =$

O	t_1	t_2	t_3	t_4	t_5	t_6
m_1	1	1	1	1	1	1
m_2	1	1	0	0	0	0
m_3	0	0	0	0	0	0
m_4	0	0	0	0	0	0
m_5	0	0	0	0	0	0
m_6	0	0	0	0	0	0
m_7	0	0	0	0	0	0

$O_{\max} =$

O	t_1	t_2	t_3	t_4	t_5	t_6
m_1	1	1	1	1	1	1
m_2	1	0	0	0	0	0
m_3	1	0	0	0	0	0
m_4	0	0	0	0	0	0
m_5	0	0	0	0	0	0
m_6	0	0	0	0	0	0
m_7	0	0	0	0	0	0

Of course, any permutation of the values in a given column and any permutation of the columns results in the same value of NHD. In other words, there are several matrices that give rise to the same value of NHD, $O'_{\max}$ being one example for $\sigma = 8, k = 7$ and $l = 6$. In fact, it can be seen that NHD for `Alert` is equal to the maximum possible value, since the parameter occurrence matrix for `Alert` has a column that contains three 1s, the remaining columns each containing a single 1.

$O'_{\max} =$

O	t_1	t_2	t_3	t_4	t_5	t_6
m_1	0	0	0	0	0	0
m_2	0	1	1	1	1	0
m_3	1	1	0	0	0	0
m_4	0	0	0	0	0	0
m_5	0	0	0	0	0	1
m_6	0	1	0	0	0	0
m_7	0	0	0	0	0	0

Remark 4.2. It can be shown that

$$\mathrm{NHD}_{\min} = q(2d - k + 1) + \frac{1}{2}l(2d^2 - 2dk + k(k-1)), \tag{8}$$

$$\mathrm{NHD}_{\max} = \frac{1}{2}(k-1)(k(l-1)+2c) + \frac{1}{2}(2r^2 - 2r(k-1) + (k-1)(k-2)). \tag{9}$$

Given that d takes discrete integer values that change every l values and q takes integers between 0 and $l-1$, the graph of $\mathrm{NHD}_{\min}$ is a series of line segments that is approximately parabolic. Similarly, $\mathrm{NHD}_{\max}$ is a series of parabolae, each of whose starting points lie on a straight line.

Figure 2 shows two graphs comparing $\mathrm{NHD}_{\max}$ and $\mathrm{NHD}_{\min}$ with CAMC as σ varies for values of $k = l = 6$ and $k = 9, l = 4$. Note that the CAMC metric takes the same values in both figures. That is, the CAMC metric is not sensitive to parameter occurrence matrices of different shapes.

The CAMC_s metric includes the "self" parameter type in the parameter occurrence matrix. We can define the NHD_s metric in the same way.

PROPOSITION 4.4. *For any class C,*

$$0 \leqslant \mathrm{NHD}(C) \leqslant 1, \tag{10}$$

$$\mathrm{NHD}(C) \leqslant \mathrm{NHD}_s(C). \tag{11}$$

PROOF. Proof of (10): Clearly, there are no disagreements if the parameter occurrence matrix contains only 1s. In this case, $\mathrm{NHD}(C) = 1$.

We maximise the number of disagreements by setting $c_j = k/2$, $1 \leqslant j \leqslant l$. In this case, $c_j(k - c_j) = k^2/4$ and for $k \geqslant 2$,

$$\mathrm{NHD}(C) = 1 - \frac{2}{lk(k-1)}\frac{lk^2}{4} = \frac{1}{2k(k-1)}(2(k-1)-k) = \frac{k-2}{2(k-1)} \geqslant 0,$$

with equality when $k = 2$. Clearly, NHD has no meaning when $k = 0$ or $k = 1$; therefore, $\mathrm{NHD} = 0$ if there are two methods ($k = 2$) and half the entries in the matrix are 1. This obviously corresponds to the intuitive case where we have two bit patterns which disagree on each bit.

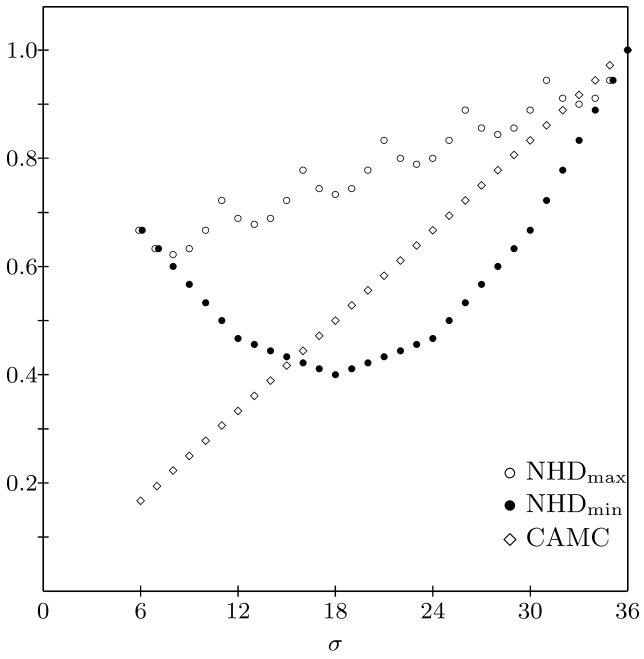
Proof of (11): By definition,

$$\mathrm{NHD}(C) = \frac{2}{lk(k-1)}\sum_{j=1}^{l} c_j(k - c_j).$$

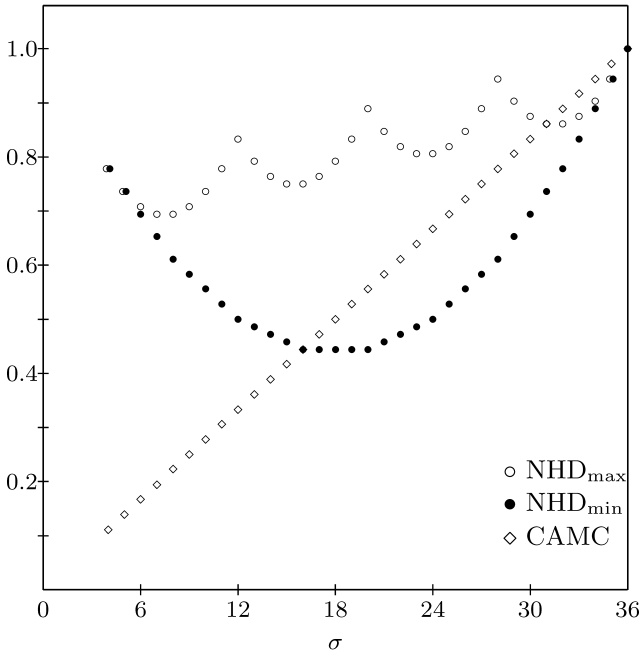
To compute NHD_s, we append a column of 1s to the parameter occurrence matrix forming the $(l+1)$th column. Clearly, there are no disagreements in this new column. Hence,

$$\mathrm{NHD}_s(C) = 1 - \frac{2}{(l+1)k(k-1)}\sum_{j=1}^{l+1} c_j(k - c_j) = 1 - \frac{2}{(l+1)k(k-1)}\sum_{j=1}^{l} c_j(k - c_j)$$

$$\geqslant 1 - \frac{2}{lk(k-1)}\sum_{j=1}^{l} c_j(k - c_j) = \mathrm{NHD}(C) \quad \square$$



(a) $k = l = 6$



(b) $k = 9,\ l = 4$

Fig. 2. A comparison of NHD_{min}, NHD_{max} and CAMC.

4.2.1 *Interpretation of the* NHD *Metric.* Counsell et al. [2002] suggest that a class for which the NHD metric is more than 0.5 should be considered cohesive. The basis for this decision is not rigorously justified.

It is likely that the parameter occurrence matrix for a class with a large number of methods and parameter types will be sparse. The analysis in the previous section and Figure 2 implies that the value of NHD for such a class will be high because there will be a high number of agreements arising from the 0s in the matrix. It is questionable whether this is satisfactory behaviour for a cohesion metric, as small classes are generally regarded as being more cohesive than large ones Counsell et al. [2002].

We must also consider the interpretation of the matrices that give rise to $\text{NHD}_{\max}$ and $\text{NHD}_{\min}$. In particular, is it intuitively reasonable to regard $O_{\min}$ as less cohesive than $O_{\max}$? $O_{\min}$ contains d rows that agree in every position and $k-d-1$ rows that agree in every position, while $O_{\max}$ exhibits agreement within a subset of the parameter types. It is not obvious which of these extremes represents a cohesive class. In short, we must reconsider carefully what we mean by cohesion. Only then will we be able to interpret NHD (and CAMC) properly. Clearly, there is a case for saying that the classes represented by $O_{\min}$ and $O_{\max}$ are both cohesive, albeit in different ways. This is obviously an attractive suggestion for proponents of the NHD metric, as it extends the range of classes about which the NHD metric has something meaningful to say.

4.2.2 *The Scaled* NHD *Metric.* Finally, we introduce the scaled NHD metric (SNHD). The SNHD metric is intended to make use of the fact that both ends of the range of values for NHD could be considered to represent cohesion in a class.

Informally, SNHD represents how close the NHD metric is to the maximum value of NHD compared to the minimum value. Formally, we have

$$\text{SNHD} = \begin{cases} 0 & \text{if } \text{NHD}_{\min} = \text{NHD}_{\max} \text{ and } \sigma < kl, \\ 1 & \text{if } \sigma = kl, \\ 2\left(\dfrac{\text{NHD} - \text{NHD}_{\min}}{\text{NHD}_{\max} - \text{NHD}_{\min}}\right) - 1 & \text{otherwise.} \end{cases}$$

Hence, SNHD has the following properties:

(1) $|\text{SNHD}| \leqslant 1$. In particular, $\text{SNHD} = -1$ implies that $\text{NHD} = \text{NHD}_{\min}$ and $\text{SNHD} = 1$ implies that $\text{NHD} = \text{NHD}_{\max}$.
(2) The closer SNHD is to ± 1, the closer NHD is to either the maximum or minimum value of NHD.
(3) The closer SNHD is to 0, the less cohesive the class.
(4) $\text{SNHD} = 0$ whenever $\text{NHD}_{\min} = \text{NHD}_{\max}$. We chose to define SNHD in this way because if $\sigma < kl$ and $\text{NHD}_{\max} = \text{NHD}_{\min}$ implies that $\sigma = l$ or $\sigma = l + 1$. Hence, there is little that can be said about the cohesion of a class with such a sparse parameter occurrence matrix (note that if $\sigma = l$ no pair of methods shares a parameter of the same type, and if

Table II. Evaluation of Cohesion Metrics

System	Class	k	l	$CAMC_s$	CAMC	NHD_s	NHD	$SNHD_s$	SNHD
Et++	Alert	7	6	0.306	0.190	0.714	0.667	1.000	1.000
	ApplDialog	4	3	0.438	0.250	0.625	0.500	1.000	0.000
	BagItem	11	4	0.309	0.136	0.804	0.755	1.000	1.000
	Dialog	15	6	0.248	0.122	0.810	0.778	0.830	−0.586
	CycleItem	14	11	0.202	0.130	0.797	0.778	0.512	−0.451
	BitMap	22	10	0.169	0.086	0.856	0.842	0.757	−0.555
	Assoc	11	3	0.409	0.212	0.773	0.697	1.000	1.000
Rocket	Arc	5	2	0.467	0.200	0.733	0.600	1.000	0.000
	ArcList	9	3	0.389	0.185	0.764	0.685	1.000	1.000
	CallGraph	11	4	0.273	0.091	0.855	0.818	1.000	0.000
	DDGArcTypeList	9	3	0.389	0.185	0.764	0.685	1.000	1.000
	DDGNNodePtrList	10	3	0.475	0.300	0.661	0.548	0.227	−0.835
	DataType	20	5	0.225	0.070	0.887	0.864	0.989	−1.000
	DeclaratorPtrList	11	3	0.477	0.303	0.664	0.552	0.177	−0.875
Edge	null_dummy	7	4	0.343	0.179	0.733	0.667	1.000	0.000
	constr_descriptor	8	9	0.225	0.139	0.757	0.730	1.000	0.000
	constr_queue	8	8	0.292	0.203	0.687	0.647	0.344	−0.807
	constr_manager	17	8	0.229	0.132	0.827	0.805	0.773	0.206
	gne_default	14	3	0.393	0.190	0.775	0.700	0.868	0.013
	elist	10	2	0.533	0.300	0.704	0.556	0.521	−0.490
	intersect	21	4	0.314	0.143	0.800	0.750	0.758	−0.750

$\sigma = l + 1$ then one pair of methods shares a parameter of the same type).

5. EMPIRICAL RESULTS

We computed the values of CAMC, $CAMC_s$, NHD, NHD_s, SNHD, and $SNHD_s$ for seven classes from each of three C++ systems: Et++ is a user interface framework, Rocket is a compiler system, and Edge is a graph editor. The seven classes were chosen at random from each of the three systems. We concede that the small size of each sample is not enough to draw meaningful relationships about all classes in the system. However, they are large enough to allow a detailed mathematical appraisal of the metrics under study. Table II is a summary of the results. The parameter occurrence matrices for each of the classes used are shown in Appendix (these matrices were compiled by inspection of the source code for each of the classes.)

Table IIIa shows Pearson's correlation coefficient (PCC) [Snedecor and Cochran 1989] between each distinct pair of metrics. PCC measures the likelihood of the existence of a linear relationship between two sets of data. The absolute value of PCC is less than or equal to 1, where a positive value indicates a linear relationship with positive gradient, and vice versa. The significance of a given value of PCC varies with the number of observations. In our case, there are 21 observations and 1% significance holds if the absolute value of PCC is

Table III. Pearson's Correlation Coefficient

	CAMC	NHD $_s$	NHD	SNHD $_s$	SNHD
CAMC $_s$	**0.891**	**−0.716**	**−0.869**	−0.234	0.092
CAMC	–	**−0.896**	**−0.944**	−0.522	−0.044
NHD $_s$	–	–	**0.962**	0.415	0.032
NHD	–	–	–	0.360	0.010
SNHD $_s$	–	–	–	–	**0.648**

(a) Pairwise by metric.

	CAMC $_s$	CAMC	NHD $_s$	NHD	SNHD $_s$	SNHD
kl	**−0.801**	**−0.638**	**0.615**	**0.708**	−0.131	−0.338
k	**−0.552**	**−0.562**	**0.730**	**0.718**	−0.115	−0.406
l	**−0.843**	**−0.551**	0.363	**0.556**	−0.136	−0.219

(b) By metric with respect to k, l and kl.

greater than 0.549 (entries marked in bold in Table III exceed this threshold value).

From Table IIIa it can be seen that there is a high correlation between CAMC and CAMC$_s$, NHD and NHD$_s$, and SNHD and SNHD$_s$; this would be expected since the metrics are computed from parameter occurrence matrices which differ only by an additional column of 1s. In many cases this changes the matrix only slightly. The most striking features of this table are that the SNHD variants have poor correlation with the CAMC metrics, whilst the NHD variants have a very strong negative correlation with the CAMC metrics. This means that if a class has a high value for the NHD metric, then it is likely that the value of CAMC will be low, and vice versa. Figure 3 is a scatter plot of the values of CAMC and NHD from Table II and shows the (negative) linear relationship between the two metrics.

Table IIIb shows PCC for each class when compared to each of k, l and kl. It is clear from the table that CAMC, CAMC$_s$, NHD, and NHD$_s$ each have a strong linear relationship of some type with kl (this confirms our earlier claim that CAMC would be higher for small groups and that NHD would be higher for large groups). On the other hand, the table indicates that SNHD would seem to be independent of class size features, a desirable characteristic of any cohesion metric. For completeness, Appendix B contains the corresponding Spearman's and Kendall's correlation coefficients corresponding to Tables IIIa and IIIb. Further empirical investigations will need to be undertaken, in conjunction with practitioners, to establish whether SNHD corresponds to an application developer's sense of cohesion.

5.1 Cross Comparison of the Three Metrics

In order to illustrate the extent to which the three metrics (LCOM, SNHD, and CAMC) correlate, coefficients for the latter two metrics versus LCOM were computed. Table IV shows the values produced, together with the mean values of the three coefficients. None of the correlation values were significant at the

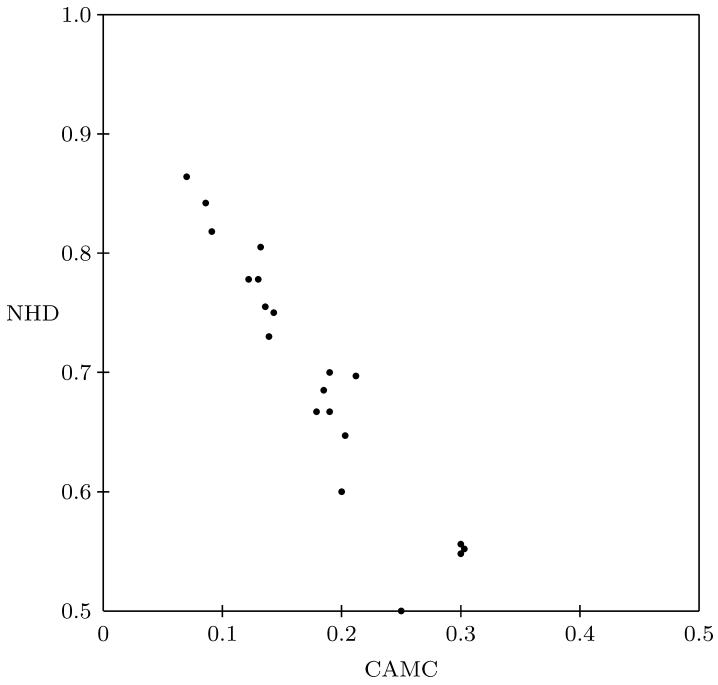


Fig. 3. Scatter plot of NHD against CAMC.

Table IV. Correlations Between the Three Cohesion Metrics

Comparison	Pearson's	Spearman's	Kendall's	Mean
LCOM versus SNHD	−0.458	−0.425	−0.337	−0.407
LCOM versus CAMC	−0.540	−0.239	−0.156	−0.312

1% or 5% level (although for LCOM versus SNHD, the values were just outside this threshold).

Interestingly, for the nonparametric correlation coefficients (Spearman's and Kendall's), LCOM versus SNHD showed the highest negative correlation values. Negative values are also evident between LCOM and CAMC. Since high values of LCOM indicate low cohesion and vice versa, and the table shows that there is evidence of correlation between design-time metrics (such as CAMC and SNHD) and code-based metrics (such as LCOM), we may infer that CAMC and SNHD provide a useful alternative to LCOM, since they can be computed far earlier in the development cycle. This is despite the fact that in many methods there is direct relationship (i.e., instance variables are assigned to the value of parameters in many cases).

Table Va confirms our earlier analysis. In particular, CAMC takes values that are considerably less than NHD. The median values of CAMC and NHD for the 20 classes in Table Va are approximately 0.43 and 0.81, respectively. The value of NHD is slightly higher than one might expect by inspection of Figure 2, and the value of CAMC is slightly lower than might be expected.

Table V. Cumulative Frequencies of NHD and CAMC Values for Two Sets of Classes

Value	NHD	CAMC
< 0.1	0	0
< 0.2	0	0
< 0.3	0	6
< 0.4	1	12
< 0.5	2	17
< 0.6	6	19
< 0.7	9	20
< 0.8	19	20
< 0.9	20	20
< 1.0	20	20

(a) AWT toolkit

Value	NHD	CAMC
< 0.1	2	0
< 0.2	2	0
< 0.3	2	0
< 0.4	5	1
< 0.5	8	4
< 0.6	9	10
< 0.7	12	12
< 0.8	13	13
< 0.9	13	13
< 1.0	20	20

(b) Event package

We would expect from our analysis that the classes in AWT give rise to large parameter occurrence matrices (which yield relatively high values of NHD and relatively low values of CAMC). Indeed, the average number of methods in the 20 AWT classes studied was approximately 13.80.

In contrast, the values of NHD and CAMC are more similar in Table Vb. This is partly because of the peculiarity of two classes having an NHD value of 0. This can only happen if the class has two methods and the parameter occurrence vectors disagree on each coordinate (in this case, CAMC = 0.5, which seems a particularly inappropriate result for this metric since the methods disagree on every parameter type). It is also due to the fact that the average number of methods of the 20 Event classes studied was only 4.15. That is, the classes in the Event classes were smaller on average than those in the AWT classes. We have already seen that CAMC tends to be larger for small classes and NHD tends to be smaller.

In short, the results of empirical studies confirm the results of our analysis: The NHD metric produces higher results than CAMC; the NHD metric finds higher cohesion in larger classes; and the CAMC metric finds higher cohesion in smaller classes.

6. DISCUSSION

Our interpretation of cohesion has been made on the assumption that a class which shares a high proportion of its parameter types is more cohesive than one which shares a low proportion of its parameter types. We accept that there may be classes that would have low cohesion by our measures, yet be generally considered cohesive: For example, a class representing a Person entity with attributes such as Age and Social Security Number; both attributes are related in the context of class Person but are unrelated in terms of potential computation. We would argue that while it is entirely possible for such a class to exist, three arguments could be put forward mitigating this criticism. Firstly, based on our experience of the way classes are written, it is unlikely that the setting of the attributes of such a class would be coded via one method per attribute. A

developer is more likely to set a number of attributes (such as those described) in a single method and for those values to be passed in the form of parameters. Secondly, classes in systems do not generally tend to fall into this category of class; the nature of most classes is for attributes to be shared amongst methods. This is particularly apparent if the class has evolved over time and new attributes and methods have been added. Finally, if developers seek to minimise the number of methods in a class, then it makes more sense to localise the assignment of instance variables to as few methods as possible.

There are a number of limitations of this research that are worth stating. Firstly, we could be criticised for introducing metrics which merely add to the body of current work on cohesion, some of which claims to have produced the *best* definition of cohesion. In our defence, we have introduced variations of a metric that attempt to capture *our* interpretation of cohesion; those metrics have been introduced to further our mathematical analysis and in comparison with other existing metrics. Secondly, the small number of classes used in the study may pose a threat to the scalability of the results. In our defence, the chief purpose of the article was to assess the relative merits of two metrics; this could be achieved without using a large sample. We thus leave for future work the scaling-up of the empirical study, the results of which we believe would support our conclusions. Thirdly, classes in different OO application *types* may exhibit significantly different trends in metric values to those examined in this article; for example, a GUI application compared with a set of library classes. Identifying traits across applications is beyond the scope of this study. Further empirical research needs to be undertaken before any concrete conclusions can be drawn in this sense. Finally, the interplay between cohesion and OO coupling has not been explicitly addressed in this article. Extending the study to investigate this feature of software may shed further light on the issues raised, especially in view of the fact that calculation of our cohesion metrics and others deal with class coupling in the form of parameter types.

7. CONCLUSIONS AND FURTHER WORK

In this article we have analysed three cohesion metrics, namely, CAMC, NHD, and SNHD. We have examined the formal properties of these metrics and then applied the metrics to 21 classes from three C++ applications.

The CAMC metric has the following properties:

(1) The metric is an indication of the sparsity of the parameter occurrence matrix.
(2) Given a fixed number σ, any parameter occurrence matrix containing σ 1s has the same value of CAMC.
(3) The CAMC metric tends to produce high values for small matrices and vice versa.

We believe that the first two of these properties have little relevance to determining the cohesion of a class. For example, it would be simple to construct two nontrivial parameter occurrence matrices with the same value for σ that would

seem to have very different cohesion. The third property means that CAMC is not independent of the size of the parameter occurrence matrix. If it is the case that cohesion is some property of the distribution of the 0s and 1s in the matrix (and not simply the number of rows and columns in the matrix), then CAMC is a poor measure of cohesion.

The NHD metric has the following properties:

(1) Given a parameter occurrence matrix, NHD measures the number of agreements between methods on usage of parameter types.
(2) Given a fixed number σ, there exist parameter occurrence matrices that attain minimum and maximum values for the NHD metric.
(3) The NHD metric tends to produce high values for large matrices, and vice versa.

The first property for the NHD metric suggests that NHD is measuring a quantity closer to our interpretation of cohesion. The second property suggests that given a fixed value for σ, there are arrangements which can reflect a very cohesive class and a poorly cohesive class. Unfortunately, the third property suggests that NHD, like CAMC, is biased by the size of the underlying class.

For a given value of σ, there exists a maximum and a minimum possible value of NHD. The parameter occurrence matrix that gives rise to either the maximum or minimum possible value can be regarded as representing a cohesive class. Hence, we introduce the SNHD metric, which reflects how close the value of the NHD metric for a class is to the maximum or minimum possible value of NHD.

If we adopt the interpretation that the NHD metric is useful at both ends of its range for a given value of σ, then it is clear that SNHD offers a significant improvement on the CAMC metric. Even if we adopt the interpretation that the NHD metric is useful at the upper end of its range for a given value of σ, then we claim it is still a more useful measure of cohesion than the CAMC metric because it can distinguish between several parameter occurrence matrices with the same value of σ.

In conclusion, we believe that the NHD and SNHD metrics are far more useful measures of cohesion than the CAMC metric. However, we also believe that it may be necessary to refine our definition and assumptions about what cohesion means in object-oriented software in light of the matrices $O_{\max}$ and $O_{\min}$.

There are numerous opportunities for further work in this area. Firstly, we will undertake a more formal and extensive analysis of the SNHD metric, similar to that presented in this article for CAMC and NHD. In particular, it would be valuable to prove that SNHD is independent of the size of the parameter occurrence matrix (as suggested by the empirical evidence in Table IIIa). Secondly, we intend to conduct more extensive tests on whole systems. In order to do this, an automated method of generating the parameter occurrence matrix of a class is needed. Finally, it is vital that we establish that significant values of the SNHD metric actually correspond to a consensus view of cohesion amongst application and system developers.

APPENDIX

A. PARAMETER OCCURRENCE MATRICES

A.1 Et++ System

ApplDialog

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

Alert

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Assoc

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

BagItem

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

CycleItem

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Dialog

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Bitmap

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

A.2 Rocket System

Arc

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

ArcList

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

DDGArcTypeList

$$\begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

DDGNNodePtrList

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

DeclaratorPtrList

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

CallGraph

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

DataType

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

A.3 Edge System

`null_dummy`

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

`constr_queue`

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

`constr_descriptor`

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

`elist`

$$\begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

`gne_default`

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

`constr_manager`

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

`intersect`

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

B. CORRELATION COEFFICIENTS

B.1 Spearman's

The following table gives the value of Spearman's correlation coefficient pairwise by cohesion metric.

	CAMC	NHD_s	NHD	$SNHD_s$	SNHD
$CAMC_s$	**0.881**	**−0.698**	**−0.827**	−0.012	0.064
CAMC	–	**−0.894**	**−0.935**	−0.167	0.006
NHD_s	–	–	**0.973**	0.104	0.062
NHD	–	–	–	0.067	0.022
$SNHD_s$	–	–	–	–	**0.742**

The following table gives the value of Spearman's correlation coefficient with respect to kl, k and l.

	$CAMC_s$	CAMC	NHD_s	NHD	$SNHD_s$	SNHD
kl	**−0.869**	**−0.742**	**0.705**	**0.813**	−0.365	−0.299
k	−0.448	**−0.564**	**0.775**	**0.747**	−0.462	−0.319
l	**−0.932**	**−0.687**	0.467	**0.636**	−0.177	−0.157

B.2 Kendall's

The following table gives the value of Kendall's correlation coefficient pairwise by cohesion metric.

	CAMC	NHD_s	NHD	$SNHD_s$	SNHD
$CAMC_s$	**0.733**	**−0.505**	**−0.607**	0.022	0.035
CAMC	–	**−0.742**	**−0.807**	−0.114	0.000
NHD_s	–	–	**0.906**	0.059	0.050
NHD	–	–	–	0.022	0.010
$SNHD_s$	–	–	–	–	**0.582**

The following table gives the value of Kendall's correlation coefficient with respect to kl, k and l.

	$CAMC_s$	CAMC	NHD_s	NHD	$SNHD_s$	SNHD
kl	−0.324)	−0.394)	(**0.609**)	(**0.581**)	(−0.359)	(−0.231)
k	(**−0.822**)	(**−0.528**)	(0.357)	(**0.466**)	(−0.174)	−0.108)
l	(**−0.710**)	(**−0.547**)	(**0.536**)	(**0.639**)	(−0.287)	(−0.207)

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their insightful and helpful comments from which the article has benefitted hugely. The authors

would also like to thank Jim Bieman at the Department of Computer Science, Colorado State University, for providing access to the three systems used for the empirical work in this article.

REFERENCES

ALLEN, E. AND KHOSHGOFTAAR, T. 1999. Measuring coupling and cohesion: An information-theory approach. In *Proceedings of the IEEE International Symposium on Software Metrics* (Boca Raton, Fla), 119–127.

BANSIYA, J. AND DAVIS, C. 2002. A hierarchical model for object-oriented design quality assessment. *IEEE Trans. Soft. Eng. 28*, 1, 4–17.

BANSIYA, J., ETZKORN, L., DAVIS, C., AND LI, W. 1999. A class cohesion metric for object-oriented designs. *J. Object-Oriented Program. 11*, 8, 47–52.

BASILI, V., BRIAND, L., AND MELO, W. 1996. A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Soft. Eng. 22*, 10, 751–761.

BIEMAN, J. AND KANG, B.-K. 1995. Cohesion and reuse in an object-oriented system. In *Proceedings of the ACM Symposium on Software Reusability SSR'95* (Seattle, Wash.), 259–262.

BIEMAN, J. AND OTT, L. 1994. Measuring functional cohesion. *IEEE Trans. Softw. Eng. 20*, 8, 644–657.

BINKLEY, D., HARMAN, M., RASZEWSKI, I., AND SMITH, C. 2000. An empirical study of amorphous slicing as a program comprehension tool. In *Proceedings of the 8th International Workshop on Program Comprehension IWPC 2000* (Limerick, Ireland), 161–170.

BRIAND, L., DALY, J., AND WUST, J. 1998. A unified framework for cohesion measurement in object-oriented systems. *Empirical Softw. Eng. J. 3*, 1, 65–117.

BRIAND, L., MORASCA, S., AND BASILI, V. 1996. Property-based software engineering measurement. *IEEE Trans. Soft. Eng. 22*, 1, 68–85.

BRIAND, L., MORASCA, S., AND BASILI, V. 1999. Defining and validating measures for object-based high-level design. *IEEE Trans. Soft. Eng. 25*, 5, 722–743.

CHIDAMBER, S. AND KEMERER, C. 1991. Towards a metrics suite for object-oriented design. In *Proceedings of the OOPSLA'91* (Phoenix, Ariz.), 197–211.

CHIDAMBER, S. AND KEMERER, C. 1994. A metrics suite for object oriented design. *IEEE Trans. Soft. Eng. 20*, 6, 467–493.

COUNSELL, S., MENDES, E., SWIFT, S., AND TUCKER, A. 2001. An empirical investigation of fault seeding in requirements document. In *Proceedings of the Empirical Assessment in Software Engineering EASE'01* (Keele, UK).

COUNSELL, S., MENDES, E., SWIFT, S., AND TUCKER, A. 2002. Evaluation of an object-oriented cohesion metric through Hamming distances. Tech. Rep. BBKCS-02-10, Birkbeck College, University of London, UK.

COUNSELL, S., NEWSON, P., AND MENDES, E. 2004. Design level hypothesis testing through reverse engineering of object-oriented software. *Int. J. Soft. Eng. Knowl. Eng. 14*, 2, 207–220.

FENTON, N. AND PFLEEGER, S. 1996. *Software Metrics, A Rigorous and Practical Approach*. Thomson International.

HENDERSON-SELLERS, B., CONSTANTINE, L., AND GRAHAM, I. 1996. Coupling and cohesion (towards a valid metrics suite for object-oriented analysis and design). *Object Oriented Syst. 3*, 3, 143–158.

HITZ, M. AND MONTAZERI, B. 1995. Measuring coupling and cohesion in object-oriented systems. In *Proceedings of the 3rd International Symposium on Applied Corporate Computing ISACC'95* (Monterrey, Mexico).

KRAMER, S. AND KAINDL, H. 2004. Coupling and cohesion metrics for knowledge-based systems using frames and rules. *ACM Trans. Soft. Eng. Methodol. 13*, 3, 332–358.

LAKHOTIA, A. 1993. Rule-based approach to computing module cohesion. In *Proceedings of the 15th International Conference on Software Engineering* (Baltimore, Md.), 35–44.

LI, W. AND HENRY, S. 1993. Maintenance metrics for the object-oriented paradigm. In *Proceedings of the 1st International Software Metrics Symposium* (Baltimore, Md.), 52–60.

MITCHELL, A. AND POWER, J. 2004. Run-time cohesion metrics: An empirical investigation. In *Proceedings of the International Conference on Software Engineering Research and Practice* (Las Vegas, Nev.), 9–14.

SNEDECOR, G. AND COCHRAN, W. 1989. *Statistical Methods*, 8th ed., Iowa State University Press, Ames, Iowa.

STEVENS, W., MYERS, G., AND CONSTANTINE, L. 1974. Structured design. *IBM Syst. J. 13*, 2, 115–139.

YOURDON, E. AND CONSTANTINE, L. 1979. *Structured Design*. Prentice Hall, Englewood Cliffs, NJ.

Received July 2002; revised August 2004; accepted August 2004