# Comprehension of object-oriented software cohesion: the empirical quagmire

Steve Counsell*, Emilia Mendes** and Stephen Swift***
*Department of Computer Science, Birkbeck College, University of London,
London WC1E 7HX. Email: steve@dcs.bbk.ac.uk
**Department of Computer Science, University of Auckland, New Zealand.
***Department of Information Systems and Computing, Brunel University, Uxbridge.

## Abstract

*It is a little over ten years since Chidamber and Kemerer's object-oriented (OO) metric suite which included the Lack of Cohesion Of Methods (LCOM) metric was first proposed [9]. Despite considerable effort both theoretically and empirically since then, the software engineering community is still no nearer finding a generally accepted definition or measure of OO cohesion. Yet, achieving highly cohesive software is a cornerstone of software comprehension and hence, maintainability. In this paper, we suggest a number of suppositions as to why a definition has eluded (and we feel will continue to elude) us. We support these suppositions with empirical evidence from three large C++ systems and a cohesion metric based on the parameters of the class methods; we also draw from other related work. Two major conclusions emerge from the study. Firstly, any sensible cohesion metric does at least provide insight into the features of the systems being analysed. Secondly however, and less reassuringly, the deeper the investigative search for a definitive measure of cohesion, the more problematic its understanding becomes; this casts serious doubt on the use of cohesion as a meaningful feature of object-orientation and its viability as a tool for software comprehension.*

## 1 Introduction

It is over twenty years since Yourdon and Constantine [19] first proposed their seven-point ordinal scale for component cohesion. At one end of their scale *functional* cohesion indicated that a module performed a single well-defined function, for example operations on a single data structure. At the other end of the scale *coincidental* cohesion indicated that the module performed more than one function, and that those functions were unrelated. More recently, the Lack of Cohesion Of Methods in a class (LCOM) metric as part of the Chidamber and Kemerer (C&K) metrics suite was seen as the seminal OO cohesion metric. A number of attempts have been made to improve upon that metric and capture cohesion more fully. It is only recently, however, that cohesion has become the subject of any real scrutiny. For example, Briand et al. [6] propose a framework for cohesion measurement in which various cohesion metrics are evaluated theoretically. An empirical evaluation of OO features including that of cohesion, with respect to the probability of fault detection, was also undertaken in Briand et al. [7].

In this paper, we examine cohesion from a practical viewpoint. We play devil's advocate through three suppositions which support the belief that there is no universally acceptable measure of cohesion. We believe further, that any cohesion metric has more of a role to play during application development than post-implementation. We thus introduce an OO cohesion metric based on the parameters of the methods of an OO class. This acts firstly, as a means of illustrating some of the problems faced with any cohesion metric of this type. Secondly, it acts as a means of evaluating the metric at the earliest development stage possible, i.e., when the skeleton layout of a class becomes available (and the declaration of method parameters is known). We accept that cohesion is a subjective concept and the proposed metric is just our view of what cohesion is. Nonetheless, the metric provides a useful starting point for analysing features of cohesion. A key feature of our metric is the well documented principle of Hamming Distance [14]. Sixty classes across three large C++ systems were used as the basis of our empirical study and collection of this metric. The three systems



Proceedings of the 10 th International Workshop on Program Comprehension (IWPC'02)
1092-8138/02 $17.00 © 2002 IEEE

comprised a GUI application, a compiler and a user interface framework.

In Section 2, related work is described. A description of the three systems studied and the three suppositions examined are described in Section 3 (empirical evaluation). Data analysis to support the three suppositions is then given (Section 4), and a discussion of the issues raised is given in Section 5. Finally, some conclusions are drawn (Section 6).

## 2 Related Work

The roots of cohesion go as far back as the early seventies, when Stevens et al. [18] first began looking at inter-module metrics. Later, Yourdon and Constantine [19] proposed their seven-point ordinal scale for component cohesion. More recently, the best known attempt at capturing cohesion through a metric is the Lack of Cohesion in Methods (LCOM) metric proposed by Chidamber and Kemerer [9]. A number of attempts have been made to refine the originally defined metric [16, 15], both for practical and theoretical reasons. The original metric calculates cohesion according to the use of class attributes in the methods of a class. The metric is based on the principle that a 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, a low value of the metric indicating that they are related. However, definition of the metric is difficult to follow. The metric values produced are difficult to interpret and give little insight into the nature of the class other than the distribution of attributes therein. The LCOM is also an implementation metric which ignores the need for a measure of cohesion earlier in the development process (i.e. at design time).

The metric used for the empirical evaluation described in this paper is based on the Cohesion Among Methods in a Class (CAMC) metric proposed and evaluated in Bansiya et al. [2]. To compute the CAMC metric, for a class with $n$ methods, the union of parameter types in the method headers of a class $T$ is composed. A set $M_i$ of all parameter object types for each method is constructed. An intersection set ($P_i$) of $M_i$ with the union set $T$ is then calculated. The cardinality of each of these intersection sets is calculated and all those values summed. That sum is then divided by: $|T|$ multiplied by $n$. More formally expressed:

$T = \cup\ M_i, \forall i = 1 \text{ to } n.$

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

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

We note that the set $T$ is always non-empty, since it will always contain at least one parameter, namely, the `this` pointer received by all methods.

In the study by Bansiya et al. seventeen C++ classes were used for the case study drawn from three well-known graphical user interface packages. The CAMC metric was shown to correlate strongly with the LCOM metric of Chidamber and Kemerer, indicating that, being a design metric, it is thus preferable to the LCOM metric. It is also stated in the same study that the CAMC metric is easier to collect and provides the developer with an earlier indication of cohesion. The CAMC metric was also found to correlate with external experts' evaluation of cohesiveness of the same seventeen classes, suggesting further that the metric reflects the views on cohesion of system developers.

Data slicing has also been used as a measure of functional cohesion [3]. Analysis of program slices allows analysis of the frequency of attribute use in programs and empirical studies have been undertaken in this area also [4]; information theoretic approaches have also been used to tackle the measurement of cohesion and coupling [1].

The lack of rigour, appeal to measurement theory and empirical evaluation of cohesion metrics is highlighted well in Briand et al. [6] where clarification of the terminology associated with the measurement of cohesion, a framework for measuring cohesion and comparing measures of cohesion together with a review of current work are given.

A variation of the metric proposed in this paper was first used in Counsell et al. [11] 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 comparison 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 paper. Our metric also eliminates some of the theoretical anomalies associated with the CAMC metric.

## 3 Empirical Evaluation

In the following section, we state and describe three suppositions about cohesion, all of which we attempt to support through empirical investigation.

### 3.1 Suppositions about cohesion

- **Supposition one**. No sensible measure of cohesion can avoid incorporating coupling in its calculation. We define coupling in terms of an association metric (NAS) in Section 3.2. Cohesion has often been associated with coupling and in most software engineering texts the two are juxtaposed, see Pressman [17], for example. In terms of OO metrics, the relationship is even more defined – cohesion is almost a surrogate measure for coupling.

- **Supposition two**. Size is a confounding factor in assessing and calculating cohesion. In this paper, we refute the argument that small classes are more cohesive than larger classes (where size is defined in terms of the number of methods in a class). At present, only anecdotal evidence exists to suggest that larger classes are less cohesive than smaller ones.

- **Supposition three**. The existence of class features such as constructors confuses the measurement of cohesion and assessment of cohesion by expert developers.

Before examining each of these suppositions, we define the cohesion metric which will be used to support those suppositions and the data collected.

### 3.2 The HD metric

To compute the HD metric for a class with $n$ methods, the set of parameter types of the class are listed as column headings of a matrix. Each method is then considered individually. A value of one is placed under a parameter type if the method being considered uses that parameter type in its parameter list; otherwise a value of zero is placed under the parameter type. There are hence $n$ rows containing combinations of zeros and ones in the body of the matrix. A Hamming Distance is then taken between each pair of rows.

The value obtained from the above indicates the level of disagreement between the rows in the matrix constructed. Subtracting this value from one gives the level of agreement between the methods of the class and hence the HD measure. The value of the HD metric is a positive value $x$ where $0 \leq x \leq 1$. The closer to zero the value of the HD metric, the less cohesive the class; the closer to one the HD metric, the greater the cohesion of the class. In this paper, calculation of the HD metric includes all methods whether constructors or non-constructors.

#### 3.2.1 Example

Consider the following class definition for `Alert` (a class found in the Et++ system, see Section 3.3). It has six methods in total, one of which is a constructor and one a destructor. The constructor and destructor are defined:

```
Alert(AlertType, byte *text= 0,
      Bitmap *bm= 0);
~Alert();
```

It has four other methods, defined as:

```
VObject *DoCreateDialog();
int Show(char *fmt);
int ShowV(char *fmt, va_list ap);
void InspectororId(char *buf, int sz);
```

The enumerated parameter type set for this class is therefore:

```
{AlertType,byte,Bitmap,char,va_list,int}
```

We note that the `char` parameter is only counted once even though it occurs three times in the methods. This was a necessary simplification for calculation of the HD metric. A matrix is then constructed by considering the occurrence of each parameter type in each method and allocating a zero or one accordingly. For the `Alert` class, this gives:

| | AlertType | byte | Bitmap | char | int | va_list |
|---|---|---|---|---|---|---|
| Meth. 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| Meth. 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| Meth. 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| Meth. 4 | 0 | 0 | 0 | 1 | 0 | 0 |
| Meth. 5 | 0 | 0 | 0 | 1 | 0 | 1 |
| Meth. 6 | 0 | 0 | 0 | 1 | 1 | 0 |

From this matrix, each pair of rows can then be compared for disagreement. For example, row 1 disagrees with row 2 in three columns. Row 2 disagrees with row 5 in two columns. Comparing every combination of pairs of rows for disagreement, gives the following matrix:

| | Row 1 | Row 2 | Row 3 | Row 4 | Row 5 |
|---|---|---|---|---|---|
| Row 2 | 3 | | | | |
| Row 3 | 3 | 0 | | | |
| Row 4 | 4 | 1 | 1 | | |
| Row 5 | 5 | 2 | 2 | 1 | |
| Row 6 | 5 | 2 | 2 | 1 | 2 |

We note that the pairwise comparison is uni-directional to prevent double counting. Summing the values of the individual disagreements gives a numerator for the HD

metric of thirty-four. The total number of comparisons is fifteen and the number of parameters six, giving a denominator of ninety. The HD metric is therefore 1 - 0.38 i.e., 0.62.

The HD metric has a value of one when there are no disagreements between all rows of the matrix formed as above – the class has the highest possible cohesion. A value of zero (lowest possible cohesion) is obtained for the HD metric when there is maximum disagreement between the rows of the matrix. This can only occur in the case of a class with two methods due to the binary nature of entries in the matrix composed of ones and zeros. We also note that the HD metric cannot be calculated in the case where the parameter type set for all methods of a class is empty. However, it is felt that this case does not occur frequently enough to cause any real threat to the validity of the metric. Based upon the definition above, we feel that the HD metric is a suitable measure of what we define as cohesion. It gives an average measure of all pairwise method comparisons on the similarity of method parameter types.

### 3.3 Data Collection

The HD metric was collected manually from the definitions of sixty classes in three systems. We note that the choice of which classes to analyse was not predetermined on any particular criteria. The first twenty classes in alphabetic order were taken from the directory in which each of the systems was stored.

1. System One, Edge, a Graph Editor, consisting of approximately 30.8 thousand non-comment source lines (KNCSL) and containing 80 classes.

2. System Two, Rocket, a compiler, consisting of 32.4 KNCSL and containing 322 classes.

3. System Three, Et++, a user interface framework, consisting of approximately 56.3 KNCSL and containing 508 classes.

For the study described in this paper, as well as the HD metric values, the following metrics were also collected:

1. The Number of Methods defined by each Class (NMC). This included private, protected and public methods.

2. The Number of class Associations (NAS). This was collected from the object model by counting the number of lines emanating from a class on a UML class diagram. It represents the number of other classes to which the class is coupled. The NAS does not include the self-coupling `this` feature of C++ or coupling due to inheritance. In the Example from Section 3.3, the class has two associations, namely `AlertType` and `Bitmap`.

3. The cardinality of the parameter type set, henceforward known as P, was also collected for each class, together with the number of values in P which were system defined parameters (as opposed to other possible types of parameter). In the Example, `byte`, `char` and `int` are the system defined parameter types. Also considered system defined types, but not explicitly mentioned here would be `double`, `boolean`, `float`, `long` and `short`. The developer defined parameter types are:

   ```
   {AlertType, Bitmap, va_list}
   ```

Table 1 provides summary data for each of the metrics collected (for the twenty classes analysed in each system). A number of features can be seen from this table. The Et++ system contains the highest median value for P and Edge has the lowest median value for P. Inspection of the classes reveals methods in Et++ to have relatively larger signatures and a higher proportion of system defined, as opposed to user defined parameters. In contrast, classes in the Edge system tend to have relatively small signatures and small parameter sets. Inspection of the classes reveals a relatively low number of system defined parameters in Edge, and even fewer system defined parameters in the Rocket system.

## 4 Data Analysis

In the following sections, each of the three suppositions described in Section 3.1 are re-stated (for clarity) and then examined.

### 4.1 Suppostion one

Supposition one is re-stated as:

> No sensible measure of cohesion can avoid incorporating coupling in its calculation.

The reason why cohesion is associated with coupling in the OO metric sense is that, in every proposed measure of cohesion, there is a surrogate relationship with coupling.

According to the definition of the HD metric, the value of P is the set of parameters of the class. This is likely to contain a mixture of primitive and non-primitive attributes. For supposition one, we thus examined the relationship between P and NAS for the three systems. Table 2 shows the correlation between

| System | Metric | Min | Max | Median | Mean |
|---|---|---|---|---|---|
| System One (Et++) | NAS | 2 | 13 | 3.00 | 4.10 |
| | NMC | 2 | 21 | 6.50 | 7.70 |
| | P | 1 | 8 | 5.00 | 4.75 |
| System Two (Rocket) | NAS | 1 | 7 | 2.00 | 2.50 |
| | NMC | 3 | 39 | 9.00 | 10.15 |
| | P | 1 | 4 | 3.00 | 2.65 |
| System Three (Edge) | NAS | 1 | 7 | 2.50 | 2.45 |
| | NMC | 2 | 24 | 11.50 | 12.45 |
| | P | 1 | 6 | 2.00 | 2.90 |

**Table 1. Summary data for the three systems studied**

the P values and the corresponding values for the NAS metric. The relationships between the variables were measured using three correlation coefficients namely, Pearson's, Kendall's and Spearman's correlation coefficients. Pearson's correlation coefficient is appropriate for parametric data, where certain assumptions about the distribution of data are assumed. Kendall's and Spearman's correlation coefficients relate to non-parametric data, and do not make any assumptions about the distribution of the data.

Every value for Systems One and Three is statistically significant at the 1% level. Interestingly, System Two shows a different pattern. It would appear that methods in the classes of Rocket contain relatively smaller levels of coupling when compared with the two other classes. Inspection of the classes reveals two factors contributing to the lack of significant correlation for this system. Firstly, the majority of the methods in Rocket classes have no parameters; nearly all classes had a `void dump();` method which prints the data structure that the class manipulates. Most of the Rocket classes studied also had at least one operator overloading method (in each case declaring no parameters). Methods in the classes of the Rocket system therefore tended to be very basic in their parametric requirements. Secondly, unlike the other two systems, a substantial amount of the coupling in Rocket eminates from the return types of the methods; again, this is due to the nature of the system, where lists are frequently passed as parameters and returned as such. A compiler would inevitably need to manipulate such data structures as an essential part of its functionality. Hence the value of P is not so high for this system.

A similar result to that shown in Table 2 was found for a set of 114 Java classes in an earlier study of three packages Counsell et al.[10]. In that study, the CAMC metric (on which the HD metric is based) was evaluated. Taking all classes together, significant correlation values, at the 1% level, between the NAS and a $T$ value were found to be 96% (Pearson's), 64% (Kendall's) and 51% (Spearman's). The only difference between the $T$ value and the P value herein is that the former counts the `this` self-coupling value in the parameter set, while the latter does not. We thus find strong support for supposition one regarding the relationship between cohesion and coupling.

We accept that our HD metric ignores the distribution of attributes in the class (i.e., declared as either public, private or protected). This could be seen as a drawback of the metric, since the original LCOM and many subsequent alternative versions of LCOM have incorporated attributes in their calculation. To counter this criticism, five random classes were selected from each set of twenty classes and the number of non-primitive attributes counted. It was found that for the Et++ system, 63% of attributes for the selected five classes were non-primitive. For the Rocket system, 57% of all attributes were non-primitive; classes in this system tended to have very few (and in most cases zero explicitly stated attributes). Finally, for the edge system, 64% of attributes in the chosen five classes were non-primitive. We note that in the case of the Rocket system, more often than not, a class contained zero attributes. This is a feature of systems which would severely impair a cohesion measure based solely on those attributes.

We conclude that any measure of cohesion which uses parameters of class methods, the attributes declared by a class, or a combination of both, cannot avoid including a high degree of coupling to other classes in its calculation. Comprehension of class cohesion is largely an exercise in comprehension of class coupling.

### 4.2 Supposition two

Supposition two is re-stated as:

> Size is a confounding factor in assessing and calculating cohesion.

| System | Pearson's | Kendall's | Spearman's |
|---|---|---|---|
| System One (Et++) | 0.59* | 0.61* | 0.69* |
| System Two (Rocket) | 0.41 | 0.28 | 0.30 |
| System Three (Edge) | 0.83* | 0.62* | 0.74* |
| * significant at the 1% level | | | |

**Table 2. NAS vs. P correlation coefficients for the three systems investigated.**

A commonly-held belief regarding cohesion is that small classes (with few methods) are generally more cohesive than larger classes. A small number of methods would seem to imply that those methods are more inter-related than those in larger classes. Table 3 shows the correlation coefficients for NMC versus HD for each of the three systems. Counter-intuitively, the values in the table suggest that, for each system, the larger the number of methods, the more cohesive the class. A simple explanation may account for this result. For systems such as Rocket, with a small set of parameters and a high proportion of methods with zero parameters, the disagreement is relatively low amongst the methods of its classes. Agreement is therefore quite high and hence so too is cohesion. The smaller the NMC, the greater the disagreement if one or two parameters are distributed unevenly within the signatures of the methods. Classes in Et++ on the other hand tended to have relatively few methods with zero parameters, suggesting that this promoted disagreement between the parameter sets and detracted from the cohesiveness of the classes studied in this system.

In terms of the HD metrics' susceptability to size, we could criticise it for being sensitive to classes with a large number of methods having zero parameters. In terms of the HD metric and its ability to capture cohesion, it would seem that the best way to develop classes and hence systems is to firstly, minimise the set of parameters to the classes and secondly, utilise as many of that set of parameters in as many methods of the class as possible (we note that this says nothing with respect to how large or small the classes should be – in theory a large class can be equally cohesive as a small class). Interestingly, the first of these two guidelines would seem to have been the goal of every cohesion metric advocated in the past; the LCOM metric itself is based on this principle.

A similar result to that stated was found for the 114 Java classes studied in [10] where the CAMC metric was used. Since the CAMC was shown to correlate strongly with the LCOM metric [2], it would be fair to say that, to date, there is no empirical evidence to suggest that smaller classes are any more cohesive (according to the measures of cohesion currently available) than larger classes. Since, in this paper, we view coupling as a surrogate measure for cohesion, further evidence for the confounding effect of size would be to show the lack of positive correlation between coupling and size (given by the NMC and NAS metrics, respectively). In other words, class cohesion (i.e., complexity of coupling) is unrelated to class size (its number of methods). Table 4 shows the correlation values between these two metrics.

From Table 4 it can be seen that only two of the nine values are significant. The two significant values are both Pearson's, which (being parametric in nature) could be considered less reflective of data typically extracted from features of software. The trend indicates no link between coupling and size. This relationship becomes even more tenuous when you consider that the NAS metric only counts *unique* occurrences of an association between two classes. One obvious example which further supports this supposition is the case of a key class [12], a class which typically contains a large number of methods to provide key functionality to other classes. We would expect such a class to be relatively cohesive due to its importance in the system being considered and the large amount of care given to it during its lifetime by developers (owing to its importance).

We further support our supposition with a recent paper by El Emam et al. [13] which claims that size is a confounding factor for various metrics, including the set of metrics initially proposed by Chidamber and Kemerer when related to the fault-proneness of a class; it casts doubt on validity of previous studies suggesting that the conclusions from such studies should be re-examined. The danger in not controlling confounding factors is also highlighted in Briand et al. [5].

From the evidence presented, it would seem that size (expressed here in terms of the number of methods) is indeed a confounding factor in the measurement of cohesion. Comprehension of class cohesion should account for size anomalies in the metric itself.

### 4.3 Supposition three

Supposition three is re-stated as:

> The existence of class features such as constructors confuses issues associated with mea-

| System | Pearson's | Kendall's | Spearman's |
|---|---|---|---|
| System One (Et++) | 0.75* | 0.70* | 0.84* |
| System Two (Rocket) | 0.57* | 0.27 | 0.34 |
| System Three (Edge) | 0.50* | 0.42* | 0.55* |
| * significant at the 1% level | | | |

**Table 3. HD vs. NMC correlation coefficients for the three systems investigated.**

| System | Pearson's | Kendall's | Spearman's |
|---|---|---|---|
| System One (Et++) | 0.82* | 0.31 | 0.38 |
| System Two (Rocket) | 0.50* | 0.26 | 0.35 |
| System Three (Edge) | 0.21 | 0.14 | 0.17 |
| * significant at the 1% level | | | |

**Table 4. NMC vs. NAS correlation coefficients for the three systems investigated.**

suring cohesion and assessment of cohesion by expert developers.

In the paper by Bansiya et al., the CAMC metric was calculated with *and* without the constructors of each class. It was suggested in the same paper that calculation of the CAMC metric for classes with a small number of methods may be unfairly influenced by the inclusion of constructors in its calculation, since, in theory, the constructor is more likely to use parameters than other methods; the same influence is less likely with larger classes. In this paper therefore, a sample of six classes were taken from each of the three systems studied and the HD metric re-calculated excluding the constructors. The three largest classes and the three smallest classes of each set of twenty classes made up these six classes (we omit the data for the Rocket system for space reasons). Table 5 shows HD values for the three largest and three smallest classes with and without constructors, together with the difference in HD values after removing the constructor(s). The number of constructors in each class is shown in brackets after the NMC values. Removal of the constructor(s) does seem to affect the smaller classes more than the larger, however. The exception to this is Class 5 which shows no change in the metric. Inspection of this class (`BagIter`) reveals it to have one parameter and that parameter is used in the constructor and one other of its three methods. Removal of the constructor does not therefore affect the value of the HD metric since the removal of an agreement is offset by the introduction of a disagreement.

Table 6 shows the corresponding values for the Edge system. Removal of the constructor in the large classes causes very little change in the values of the HD. For smaller classes, removal of the constructor rendered the HD metric not computable (nc) for two of the smaller classes. In these two cases, removing the constructor caused the class to have an empty parameter set. Interestingly, Edge had the lowest mean NAS and the smallest mean and median P values. Similar results was found for the 114 Java classes in which the CAMC was evaluated.

A further reason why constructors confuse the measurement of cohesion is that, by their nature, they are methods which are more likely to use the class attributes declared in initialising values than the non-constructor methods would. In addition, constructors tend to comprise few (if at all) parameters in their signatures. In other words, a metric based on the parameters of the methods should exclude constructors. It could be argued that destructors fall into the same category, although they tend to be declared less often in classes.

One might think that a purer measure of cohesion than that given would be to modify the HD metric to ignore any parameters which appear in the form of other classes (i.e., class coupling). However, the problem with omitting such parameters is that a high percentage of the classes analysed used this form of coupling extensively. The modified metric would hold no real value based on a relatively small subset of all class parameters. Table 7 emphasises the difference between the three systems in this respect. It shows the frequency of system defined parameters in the methods of each class. It can be seen that in Et++, only two classes had zero system defined parameters with seven classes having two system defined parameters. Contrast this with nine of the twenty classes in Edge having zero system defined parameters and nine with a single system defined parameter. Fifteen of the twenty Rocket classes had zero system defined parameters. The remaining five had just one system defined parameter.

| Class | NMC | With Constructors | Without Constructors | Difference |
|---|---|---|---|---|
| *Class 1* | 21 (2) | 0.77 | 0.87 | 0.10 |
| *Class 2* | 15 (1) | 0.78 | 0.79 | 0.01 |
| *Class 3* | 14 (3) | 0.80 | 0.70 | 0.10 |
| *Class 4* | 2 (1) | 0.20 | 0.50 | 0.30 |
| *Class 5* | 3 (1) | 0.33 | 0.33 | no change |
| *Class 6* | 3 (1) | 0.33 | 0.00 | 0.33 |

**Table 5. HD values with and without inclusion of constructors (three largest and three smallest classes – Et++).**

| Class | NMC | With Constructors | Without Constructors | Difference |
|---|---|---|---|---|
| *Class 1* | 24 (1) | 0.66 | 0.71 | 0.05 |
| *Class 2* | 23 (2) | 0.61 | 0.60 | 0.01 |
| *Class 3* | 22 (2) | 0.58 | 0.58 | no change |
| *Class 4* | 5 (1) | 0.77 | 0.66 | 0.11 |
| *Class 5* | 4 (1) | 0.50 | nc | nc |
| *Class 6* | 2 (1) | 0.20 | nc | nc |

**Table 6. HD values with and without inclusion of constructors (three largest and three smallest classes – Edge).**

One explanation for the lack of system defined attributes in Rocket is that parameters to the methods in the classes of this this system tend to be only link and list data structure oriented, very often receiving themselves as a parameter type. For example, the Rocket `LoopList` class uses `LoopList`, `Dlink` and `Boolean` as parameters to its methods; this is a typical feature of classes in this system.

There is another issue which arises when trying to establish developer confidence in a measure of cohesion and ties supposition two and three together. It can be seen from Tables 5 and 6 that large classes tend to have higher HD values than small classes. This, we explained, was the fault of the HD metric definition. In the paper by Bansiya et al. [2], a strong correlation was found between expert opinion of class cohesion and the CAMC values. Although we cannot say with any certainty, we hypothesise (based on our earlier observations) that in the Bansiya paper, the experts subconsciously used the size of the class as a surrogate for its cohesion, in which case the underlying supposed, significant relationship is flawed. The experiment with expert developers needs to be replicated before any firm conclusions can be made. We conclude that any cohesion metric should be normalised to remove the size effect before any attempt to show correlation with developer opinions is made.

## 5 Discussion

A feature of the HD metric (which could be considered a weakness) is the single counting of parameter types, irrespective of how many times that parameter type occurs in the methods of the class. In defence of the metric adopted, inclusion of a count of each parameter would render the metric cumbersome and very difficult to interpret.

Generally speaking, a criticism that could be levelled against a study of this sort is that the classes examined were all of the same type and the results are thus not generalisable to other domains. However, three types of domain were chosen for the case study, reflected in the three systems investigated. Each system represented a different C++ application and, as such, the threat to the validity of the study is reduced from this viewpoint.

Another criticism of this study is that the HD metric has not been theoretically validated to conform to certain guidelines Briand et al. [8]. In defence of this, we stress that the main purpose of the paper was to show the practical limitations associated with cohesion and the HD in particular; addressing theoretical issues of the metric is left for future work.

Finally, the choice of classes in alphabetic order from the three systems could be criticised because it may cause selection of clusters of classes related to each other through their name. However, we feel that any selection criteria chosen would have been subject to one criticism or another.

| System | Zero | One | Two | Three | Four |
|---|---|---|---|---|---|
| System One (Et++) | 2 | 8 | 7 | 2 | 1 |
| System Two (Rocket) | 15 | 5 | 0 | 0 | 0 |
| System Three (Edge) | 9 | 9 | 2 | 0 | 0 |

**Table 7. Frequency of system defined parameter types in the three systems.**

## 6 Conclusions

The software engineering concept of cohesion has always been considered a subjective concept and therefore difficult to capture in a metric. In this paper, a metric based on Hamming Distance was used as a vehicle for demonstrating problems in measuring cohesion. Three suppositions related to cohesion were investigated and data collected from sixty classes across three industrial C++ systems. Results indicated that the metric did reveal some interesting issues in terms of classes and systems studied. However, any attempt at defining cohesion firstly, cannot be divorced from coupling; secondly, size is a confounding factor in the calculation of cohesion and, finally, class features such as constructors confuse its definition. A sensible cohesion metric will certainly aid comprehension of the features of the system under scrutiny. It will not, however, provide a definitive view of cohesion. The empirical evaluation described in this paper ideally needs to be replicated to either support or refute its arguments and to further our understanding of cohesion.

## Acknowledgements

The authors acknowledge the help of Jim Bieman at the Dept. of Computer Science, Colorado State University, U.S., for access to the three systems used in this paper.

## References

[1] E. Allen and T. Khoshgoftaar. Measuring coupling and cohesion: an information-theory approach. In *IEEE International Symposium on Software Metrics, Boca Raton, Florida, US*, pages 119–127, Nov 1999.

[2] J. Bansiya, L. Etzkorn, C. Davis, and W. Li. A class cohesion metric for object-oriented designs. *Journal of Object-Oriented Programming (January)*, pages 47–52, 1999.

[3] J. M. Bieman and L. Ott. Measuring functional cohesion. *IEEE Transactions on Software Engineering* 20(8):644–657, 1994.

[4] D. Binkley, M. Harman, I. Raszewski, and C. Smith. 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*, pages 161–170, 2000.

[5] L. Briand, E. Arisholm, S. Counsell, F. Houdek, and P. Thevenod-Fosse. Empirical studies of object-oriented artifacts, methods and processes: State of the art and future direction. *Empirical Software Engineering, An International Journal*, 4(4):387–404, 1999.

[6] L. Briand, J. Daly, and J. Wust. A unified framework for cohesion measurement in object-oriented systems. *Empirical Software Engineering Journal*, 3(1):65–117, 1998.

[7] L. Briand, J. Wust, J. Daly, and V. Porter. Exploring the relationships between design measures and software quality in object-oriented systems. *The Journal of Systems and Software*, 51:245–273, 2000.

[8] L. C. Briand, S. Morasca, and V. R. Basili. Property-based software engineering measurement. *IEEE Transactions on Software Engineering*, 22(1):68–85, 1996.

[9] S. R. Chidamber and C. F. Kemerer. Towards a metrics suite for object-oriented design. In *OOPSLA '91, Phoenix, Arizona*, pages 197–211, 1991.

[10] S. Counsell, E. Mendes, P. Newson, S. Swift, and A. Tucker. Evaluation of a class cohesion metric for OO design. *Birkbeck Technical Report 02-01*, 2002.

[11] S. Counsell, E. Mendes, S. Swift, and A. Tucker. An empirical investigation of fault seeding in requirements documents. In *Proceedings of Empirical Assessment in Software Engineering (EASE) '01, Keele, UK*, 2001.

[12] S. Counsell, P. Newson, and E. Mendes. Architectural level hypothesis testing through reverse engineering of object-oriented software. In *Proceedings of the 8th International Workshop on Program*

*Comprehension (IWPC'2000), Limerick, Ireland*, pages 60–66, 2000.

[13] K. El Emam, S. Benlarbi, N. Goel, and S. Rai. The confounding effect of class size on the validit y of object-oriented metrics. *IEEE Transactions on Software Engineering*, 27(7):630–650, 2001.

[14] R. Hamming. Error detecting and error correcting codes. *Bell System Technic al Journal*29:147–160, 1950.

[15] M. Hitz and B. Montazeri. Measuring coupling and cohesion in object-oriented systems. In *Proceedings International Symposium on Applied Computing*, Sep 1996.

[16] W. Li and S. Henry. Maintenance metrics for the object-oriented paradigm. In *Pr oceedingsof the First International Software Metrics Symposium, Baltimore Maryland*, pages 52–60, May 1993.

[17] R. Pressman. *Softwre Engineering: A Practioner's Approach (Fifth Edition.* McGraw Hill, 2000.

[18] W. P .Stevens, G. J. Myers, and L. L. Constantine. Structured design. *IBM Systems Journal*, 13(2):115–139, 1974.

[19] E. Y ourdonand L. Constantine. *Structured Design.* Prentice Hall, 1979.