Object Oriented Systems **3,** (1996) 143–158

Coupling and cohesion (towards a valid metrics suite for object-oriented analysis and design)

B. HENDERSON-SELLERS and L.L. CONSTANTINE

School of Computing Sciences, University of Technology, Sydney, Broadway, NSW, Australia

and I.M. GRAHAM

Swiss Bank Corporation, London, UK

The best known OO metrics are the suite of six proposed by MIT researchers. Of these six, three have problems in their definitions, one seriously (LCOM or lack of cohesion of methods). Here, this suite of metrics is evaluated from a mathematical point of view and errors in the definition of CBO (coupling between object classes) and LCOM removed. In addition, a totally new formulation for the concept of 'lack of cohesion' at the method level is proposed. Illustrative examples are used throughout.

Keywords: Metrics, object-oriented, coupling, cohesion

1. Introduction

Proposals to date for commercially-collectable OO metrics have been minimal, the best known being those from MIT ([1–3], labelled CK hereafter). Other more extensive, reviews of OO metrics are to be found in [4] and [5, Chapter 10]. Here, we focus on the suite of six CK metrics and re-evaluate their validity from a mathematical viewpoint.

It should, however, be noted that no formalized attempt has been made (either by the originators or here) to relate any of these measures to macroscale properties such as quality, reuse and costs nor to evaluate these measures in any systematic, empirical way. Neither do the authors of these CK measures discuss explicitly either lifecycle phase or abstraction level in any detail. For example, Henderson-Sellers [6] differentiates between internal class measures (e.g. implementation details), external class measures (e.g. services offered) and system measures (e.g. fan-out, inheritance).

It needs also to be stressed that, by deriving a metric first, and then trying to find out to what it applies: and then measuring it against some (disputed) criterion, adoption by industry becomes hard since it is never clear to what purpose and utility the particular metric has been targeted. However, such metrics do contribute to the pool of potential measures [6] for adoption within a specific application of Basili and Rombach's GQM framework [7], while remaining to be evaluated and their practicality assessed within specific contexts and questions.

Finally, Chidamber and Kemerer [1] observe that their six proposed metrics ''are unlikely to be comprehensive, and further work could result in additions, changes and possible deletions from this suite''. Here we offer some corrections, refinements and extensions to this set of six metrics (1991, 1993 and 1994 versions), particularly to the measures for coupling and cohesion.

2. The three versions of the Chidamber and Kemerer metrics suite

The CK suite of metrics for OO design has had three incarnations. The original ideas were put forward in an *OOPSLA* paper [1] in which the underlying ontological and measurement theory ideas were presented. An evaluation against the Weyuker [8] axioms (despite the agreed inconsistency of these axioms, e.g. [9, 10]) was also undertaken. The paper then evolved and in doing so some of the definitions of the six proposed metrics were changed radically. In addition, the later versions [2, 3] included some small amount of empirical data. The work finally saw journal publication in 1994 in much the same format as that of the 1993 paper but with fuller discussion. The mathematical definitions of LCOM and CBO are radically different from the 1991 definitions, despite the textual explanations being essentially identical. It will be assumed that it is the MIT researchers' intent that the 1994 and not the much-cited 1991 paper should be viewed as a definitive state of their metrics suite and it is these definitions that will be considered most deeply here.

However, the 1991 paper, with its errors and later-changed definitions has received much citation; several papers and unpublished studies stating that they have implemented these 1991 metrics. This is problematic – at least one of the metrics (LCOM) is impossible to implement from the 1991 definition since it always gives the same answer (zero). It can only be guessed that implementors have interpreted the *words* used in the 1991 paper and implemented the intention and not the stated definition. If this is the case, then it is extremely likely that multiple definitions have been used so that studies stating that the researchers have used LCOM (1991) are unlikely to be compatible with each other.

The six metrics proposed by Chidamber and Kemerer [1] are mostly focused on size, coupling and cohesion. Chidamber and Kemerer [1] are careful to ground each of their six proposed metrics in measurement theory and to incorporate the ontological ideas of Bunge [11], although reliance on Bunge's ontology is questioned by Graham [12, p. 410]. He notes that this philosophy is atomistic and assumes objects to be defined by their properties; whereas in object technology *all* an object's properties can change without altering the object's inherent identity. Despite this theoretical grounding, in actual fact Chidamber and Kemerer's validation assessment places more reliance on the (suspect and self-contradictory) Weyuker's set of axioms [8] than on measurement theory as propounded by, for example, Zuse [9].

The first metric of this suite of six, Weighted Methods per Class (WMC), is a 'size' measure which weights a method count with its complexity. It is given by

$$WMC = \sum_{i=1}^{n} c_i \tag{1}$$

where c_i is the static complexity of each of the n methods – a complexity often represented by a directed acyclic graph (DAG); although in the original 1991 paper the weights all appear to be taken as unity

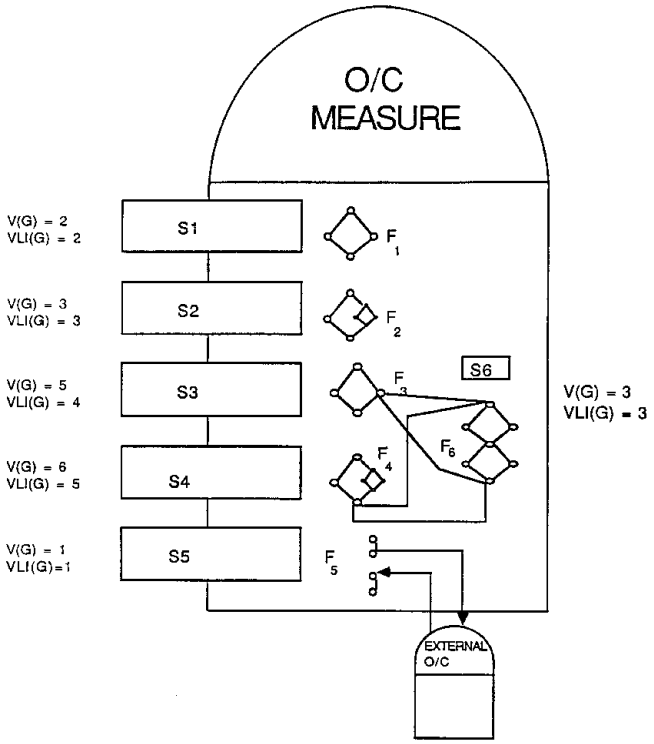

Fig. 1. Object/class MEASURE has five (publicly-available) services ($S_1 - S_5$) and one private service/method (S_6). Values for McCabe's [14] cyclomatic complexity, $V(G)$, and for Henderson-Sellers and Tegarden's [15] $VLI(G)$ are given.

(see similar comments by Sharble and Cohen [13]). This ambiguity is clarified in [3] when it is noted that ''complexity is deliberately not defined more specifically here in order to allow for the most general application of this metric''. Thus, if c_i in Equation 1 is given by $V_i(G)$, the McCabe [14] cyclomatic complexity $V(G)$ for the ith method, then $WMC = \Sigma_{i=1}^{n} V_i(G)$; although it should be noted that summing the complexities does not realistically allow for DAGs that overlap (see for example methods S_3 and S_4 in Fig. 1). This is easily dealt with using the summation rules of Henderson-Sellers and Tegarden [15] for callable 'subroutines' in which the node which sends the call is split into two and one is used to represent the outgoing control flow and one the incoming/return control flow from the subroutine. The resulting DAG can then be analysed using regular techniques. On the other hand, if $c_i = 1$ we get WMC = number of methods. (Note: as a size measure this ignores the contribution from the number of attributes.) They argue that classes with a large number of methods ''are likely to be more application-specific, limiting the possibility of reuse''.

Another minor problem with the WMC metric is that it fails to distinguish attributes (i.e. the get and set access methods) from other methods so that the metric applies well to Smalltalk but not to C++ with its member functions and separate data members. Furthermore 'number of methods' itself is an ambiguous phrase. Does WMC just relate to externally visible methods (i.e. services) or to all methods,

public/visible plus private [16]? There is no clarification in any of the three papers on this point. (For a further discussion and clarification on internal class measures see [6]).

Depth of inheritance tree (DIT) and number of children (NOC) are two obvious metrics for inheritance; yet their validity as useful measures [17] is still open to question. Deeper inheritance trees lead to lower level object classes (O/Cs: [5]) using directly ancestral methods. Thus from a maintenance viewpoint, complexity is likely to be increased. At the same time, Chidamber and Kemerer [1] propose that it is better to have depth than breadth in the inheritance hierarchy. Although defined at the class level (i.e. each class in the hierarchy possesses its own DIT value), others have, quite reasonably, interpreted it as a single value for the hierarchy, essentially DIT_{max} [12, p. 409]; thus changing its nature from an external-class level to a systems level measure. In addition, we should also remember that inheritance is not the only tree structure in an object model; depth of composition structure is also required. The metrics in the SOMA method [12] include both depth metrics and the software tool that collects the SOMA metrics, SOMATiK [18], collects these for each class and maximum, minimum and average values for each *island* of classification structure (i.e. relating each hierarchy to a base class as in C++ or Eiffel rather than a common root class `Object`, as in Smalltalk). Chidamber and Kemerer do not discuss this island phenomenon, assuming implicitly the mono-hierarchy of the Smalltalk model.

The NOC measure, which counts the number of immediate subclasses, has larger values for poorer designs. NOC is a system level metric [12].

The fourth metric proposed is Coupling Between Objects (CBO) [1] or Coupling Between Object Classes (CBO) [3]. It is clearly stated in the 1991 version that CBO counts object/class (both words are used as if interchangeably) connectivity *other than* by inheritance. It would thus appear to attempt to be a measure of the number of 'collaborators'. The 1991 version, however, states that the two collaborating objects need to ''act upon each other''. In other words, only bi-directional collaborations are counted (although it is our guess that the authors did not mean this). It is also stated that coupling prevents reuse, whereas for most authors coupling *is* reuse (both by inheritance and peer–peer collaboration). This emphasizes the restricted view of object technology by the authors at that time; one in which inheritance was seen as the main (perhaps only) structuring mechanism. By 1993, CBO was changed to a coupling definition in which the bi-directionality requirement had been removed [2]. The phrase ''non-inheritance related'' had also been removed. Thus the 1993 version of CBO implicitly required inheritance coupling to be included (or more accurately did not preclude it). That this is correct was stated clearly in Chidamber and Kemerer ([3] p. 479 footnote 5 and p. 487).

Class coupling should be minimized, in the sense of constructing autonomous modules; yet a tension exists between this aim of a weakly coupled system and the very close coupling evident in the class/superclass relationship [19]. Berard [20, p. 102] differentiates between necessary and unnecessary coupling. The rationale is that without any coupling (i.e. minimization of coupling) the system is useless. Consequently, for any given software solution there is a baseline or necessary coupling level – it is elimination of extraneous coupling that is the developer's goal. Such unnecessary coupling does indeed needlessly decrease the reusability of classes.

To our minds, amalgamating inheritance coupling with non-inheritance coupling (often called collaboration) is a serious mistake since the two mechanisms can be regarded as orthogonal. In the following, the term coupling will be taken as a synonym for non-inheritance coupling. There is another view [12] that one should, for reasons of enabling statistical analysis of metrics data, make an even finer distinction. We should collect coupling information in terms of inheritance, aggregation, message and association (and role) links. This boils down to counting fan-in and fan-out

for these four orthogonal structures, where fan-in and fan-out refer, respectively, to the number of other collaborating classes irrespective of the number of references made statically or dynamically, i.e. for a pair of classes the fan-in/fan-out value is either zero or one. Then metrics such as non-inheritance coupling can be reconstructed easily.

While Li and Henry [21] found the 1991 definition of CBO ambiguous, it is clear at least that this 1991 definition excludes inheritance coupling. Li and Henry went on to substitute their own coupling definition, data abstraction coupling (DAC), defined by them as

$$\text{``DAC} = \text{number of ADTs defined in a class.''} \tag{2}$$

but perhaps more clearly expressed as the number of non-simple attributes of distinct type defined in a class.

Both definitions would appear to exclude inheritance (although strictly Equation 2 does not) and, at least in Li and Henry's DAC, count 1 for each separate *type* referenced in a class. This is essentially unidirectional and can be likened to fan-out. Coupling of multiple objects from the same class would still give an overall coupling value of unity. Initially (i.e. early in analysis), having a metric which is binary is more than adequate; yet as the design is fleshed out, a single connection may need to be expanded to show multiple message paths when several services of a class are sued by the same client. A count which includes several non-simple attributes of the same type and/or messages sent to the same non-simple attribute from various parts of the class will clearly exceed the value of DAC (see following discussion).

Li and Henry [21] offer such an emphasis in their MPC (message-passage coupling) metric which is the 'number of send statements defined in a class' (a measure also proposed in [4], p. 33). The difference

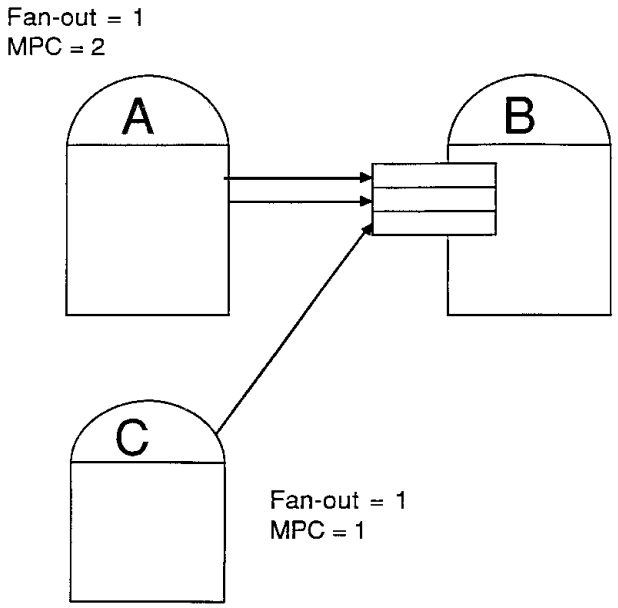

Fig. 2. Three classes, A, B and C, interact such that A and B are coupled (fan-out of A = 1) but where objects of class A use *two* different services of objects of class B, i.e. MPC = 2.

Let RS = $M \cup R_i$
where

M is the set of all methods in the class
$R_i = \{R_{ij_i}\}$ is the set of methods called by M_i

Then RFC = |RS|

Example:

Class A has four methods:

A::f1( ) calls B::f1( ), B::f2( ) and C::f3( )
A::f2( ) calls B::f1( )
A::f3( ) calls A::f4( ), B::f3( ), C::f1( ) and C::f2( )
A::f4( ) does not call any other methods

Then:

$$\begin{aligned}RS = &\{A::f1,\ A::f2,\ A::f3,\ A::f4\}\\ &\cup \{B::f1,\ B::f2,\ C::f3\}\\ &\cup \{B::f1\}\\ &\cup \{A::f4,\ B::f3,\ C::f1,\ C::f2\}\\ &= \{A::f1,\ A::f2,\ A::f3,\ A::f4,\ B::f1,\ B::f2,\ B::f3,\ C::f1,\ C::f2,\ C::f3\}\end{aligned}$$

giving:

RFC = 10

Fig. 3. Example RFC calculation [23].

between fan-out and MPC is that only one connection between any pair of classes is counted in fan-out; whereas one is added to the MPC count for each unique 'send' in the class (Fig. 2). It would thus appear that if two different methods in class A access the same method in class B, then MPC = 2 (this is in contrast to RFC – see below). A similar approach is taken by Rajaraman and Lyu where they define coupling at the method level.

The fifth metric, RFC (response for a class) measures both internal and external communication by counting the number of methods, internal and external, available (i.e. potentially used by) a class. It represents the number of message paths but does not discriminate between two messages sent to the same method but from different parts of the class. It is given by RFC = |RS| where RS, the response set of the class, is given by

$$RS = M \cup_{\text{all } i} R_i = \{M_i\} \cup_{\text{all } i} \{R_{ij_i}\} \tag{3}$$

where $M = \{M_i\} = \{M_1, M_2 \ldots, M_n\}$ = set of all methods in the class (total n) and $R_i = \{R_{ij_i}\}$ = set of methods called by the ith method, M_i (i.e. each set R_i has max (j_i) elements and there are n such sets R_i).

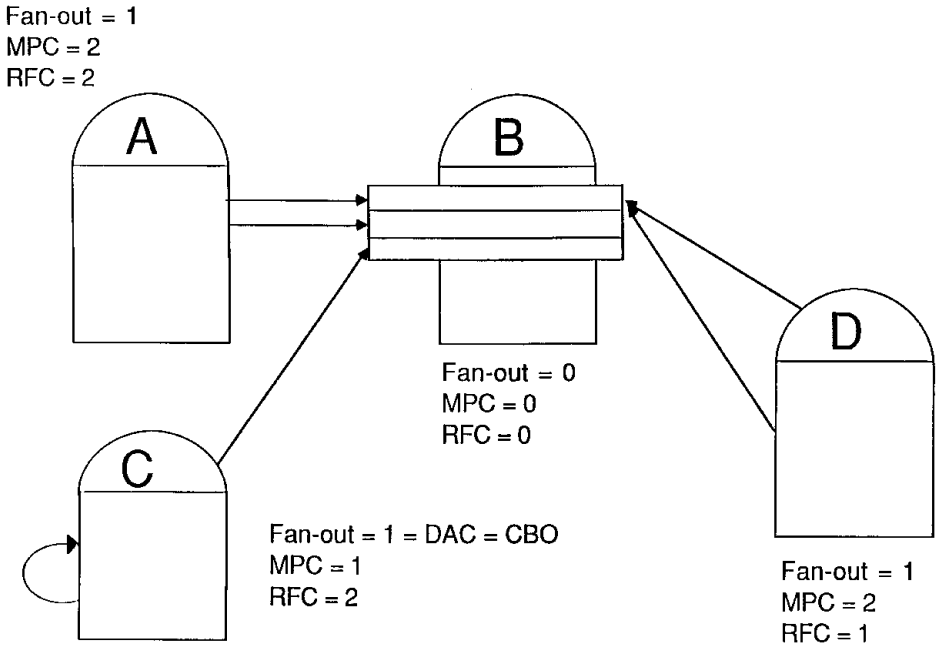

Fig. 4. As for Fig. 2 but in which a fourth class, D, accesses one of B's services from two different places. This gives a different value for MPC and RFC whereas for the interaction between A and B seen in Fig. 2 the values of RFC and MPC are necessarily identical. While for class D MPC $>$ RFC, the reverse is true for class C (RFC $>$ MPC). This reflects the internal message coupling within class C whereby one message makes an internal call, thus adding to the value of RFC but not adding to the coupling values of MPC or fan-out.

An alternative (equivalent) definition is

$$\text{RFC} = \text{NLM} + \text{NRM} \tag{4}$$

where NLM = number of local methods = $|M_i|$ and NRM = number of remote methods = $\Sigma_{i=1}^{n} |R_i|$. An example is shown in Fig. 3.

However, RFC simply addresses the notion of 'how many methods do I have access to from within the class in question'. It does not make any statement regarding the frequency of use, from different parts of the class, of those methods. This can be addressed, for the external methods by MPC (the message-passing coupling of [21]) and for the internal methods, to some degree, by $V(G)$. It should be noted that RFC and CBO are thus not orthogonal. Chidamber and Kemerer [1] suggest that the greater the value of the RFC, the 'greater the level of understanding required on the part of the tester'.

Lorenz and Kidd [4] also note that there are ''multiple ways to measure the coupling''. They go on to discuss coupling in terms of (a) number of collaborating classes (presumably CBO (1991) or fan-out) and (b) the amount of collaboration (presumably related to MPC and/or RFC) but they do not offer any quantification or any results. It should also be noted that while their use of the term ''collaborator'' seems to exclude inheritance coupling, their later discussion implies that they do intend to include inheritance in their future coupling measures.

In summary, there are various ways of counting non-inheritance coupling. First, the declaration of an object of a remote ADT creates a potential collaboration. This is measurable by fan-out ≡ DAC ≡ CBO (1991). If two O/Cs are collaborators, then a maximum of one is added to the fan-out count irrespective of how many messages flow between the two collaborating O/Cs.

Secondly, when two O/Cs collaborate, as do O/Cs A and B in Fig. 4, for each unique service accessed, one is added to the NRM count and hence to the RFC count (since RFC = NLM + NRM: Equation 4).

Thirdly, if one particular service is accessed from different parts of the 'client O/C', it adds nothing to the MPC count (O/C D in Fig. 4). Finally, class C has also one internal method call, i.e. RFC = 2 (since NRM = 1 and NLM = 1).

High fan-outs represent class coupling to other O/Cs and thus an 'excessively complex dependence' on other O/Cs. On the other hand, high fan-ins represent good object designs and a high level of reuse. Since these two are compensatory, it would not appear to be possible to maintain a high fan-in *and* a low fan-out across the whole system, since the cumulative values must be equal. More research is clearly needed in this crucial area.

Systems in which one class has a high non-inheritance coupling count and all other classes a value zero indicate a structured not an object-oriented design, with a main 'driver' class (Kreindler and Mickel, 1993, personal communication). Many classes with a large non-inheritance coupling value may (but not necessarily) indicate that the designer has been over-enthusiastic [23] and the classes are of too fine a granularity.

Chidamber and Kemerer's final metric in their proposed suite is the lack of cohesion of methods (LCOM) [1]. This is intended to be an internal measure of cohesion. The idea behind this metric (not currently realized – see below) is that a high value suggests that the methods in the class are not really related to each other nor, therefore, to a single overall abstraction. Such a high value therefore suggests that the class should be split into two or more classes. As we shall confirm in detail below, its definition is ambiguous [12] and differs between the 1991 definition and that in Chidamber and Kemerer [3].

Chidamber and Kemerer [1] suggest a metric to evaluate the internal cohesion by considering the number of disjoint sets formed by the intersection of the n sets created by taking all the instance variables used by each of n methods. For a class with methods $M_1, \ldots, M_n$, consider the set of instance variables used by method $M_i = I_i = \{I_{ij}\}$. For n methods, there are n such sets, $I_1, \ldots, I_m$ (alternatively written as $\{I_{1j_1}\} \ldots \{I_{mj_m}\}$ where j_i indexes the instance variables used by the ith method). Then the lack of cohesion of methods (LCOM) metric is given formally by Chidamber and Kemerer [1, p. 203] as

$$\text{''LCOM} = \text{The number of disjoint sets formed by the intersection of the } n \text{ sets.''} \quad (5)$$

Table 1. Evaluation of the Chidamber and Kemerer (1991) metrics against seven of the Weyuker (1988) axioms [1]

	Metric						
	1	3	4	5	6	7	9
WMC	Y	Y	Y	Y	Y	N	N
DIT	Y	Y	Y	N	Y	N	N
NOC	Y	Y	Y	Y	Y	N	N
RFC	Y	Y	Y	Y	N	N	N
LCOM	Y	Y	Y	Y	Y	N	N
CBO	Y	Y	Y	Y	Y	N	N

That this is ill-defined is easily seen. The intersection of a number of sets is a single set containing element(s) common to *all* the original sets. Formally we can state that the intersection $\cap$ of sets $S_i(i \in I)$ is given as

$$\cap_{i \in I} S_i = \{x | x \in S_i \, \forall \, i \in I\} \tag{6}$$

[24, p. 68]. In other words, the result is a *single* set. On the other hand, the notion of disjointedness involves two (or more) sets: two sets A and B are disjoint if A $\cap$ B $= \emptyset$. We must presume, therefore, that Chidamber and Kemerer [1] meant something different; the most likely rephrasing might be the number of partitions formed by counting the number of subsets of $I = (I_1, I_2, \ldots, I_n)$ such that the methods which use members of this subset do not use its complement.

The LCOM measure is derived from Bunge's [11] definition of ''similarity'' between two objects as the intersection of the sets of their properties. The degree of similarity $\sigma()$ between two methods is given by

$$\sigma(M_1, M_2) = I_1 \cap I_2 \tag{7}$$

Thus, if there are not common properties, similarity $= 0$.

One problem occurs in the inappropriate extension in Chidamber and Kemerer [1] of this definition to n sets where

$$\sigma(M_1, M_2, M_3, \ldots, M_n) = I_1 \cap I_2 \cap I_3 \ldots \cap I_n \tag{8a}$$

(correcting the set notation of [1]). In general, such an intersection is likely to be small or zero. Secondly, Equation 8a gives a *set* as a value for σ; whereas the number of elements in that set would be more appropriate, i.e.

$$\sigma(M_1, M_2, M_3, \ldots, M_n) = |I_1 \cap I_2 \cap I_3 \ldots \cap I_n| \tag{8b}$$

Perhaps Graham's interpretation of LCOM as a measure of ''the non-overlapping of sets of instance variables used by the methods of a class'' [12] or Li and Henry's interpretation as the number of disjoint sets of local methods [21] are nearer the intended meaning; yet even here, in the latter case, maximum similarity would give LCOM $= 1$ whereas all authors discuss full similarity as occurring when LCOM $= 0$. For an example, see Fig. 5. In the first part of this example, $I_1 = \{i1, i2\}$ and $I_2 = \{i3\}$. Hence $I_1 \cap I_2 = \emptyset$ and LCOM $= 2$. In the second case, $I_1 \cap I_2 = \{i1, i2\} \neq \emptyset$ and LCOM $= 1$.

It is intended that a small value of LCOM should imply high similarity while a high value of LCOM can be used to indicate that a class may be more successful if split into two or more classes.

A more useful measure which quantifies these intuitive relationships might be

$$\text{LCOM}' = |I_i \cap I_i = \emptyset, \, \forall i, j, \, i < j| \tag{9}$$

i.e. the number of empty sets formed from taking the intersection of pairs of variable sets $I_i, I_j (i < j)$. This would give, for the two examples in Fig. 5, values of LCOM$'$ of 1 and 0 respectively.

However, in [2], the definition of LCOM is altered totally to read: if $P = \{(I_i, I_j) | I_i \cap I_j = \emptyset\}$ and $Q = \{(I_i, I_j) | I_i \cap I_j \neq \emptyset\}$ then

$$\begin{aligned} \text{LCOM} &= |P| - |Q| \quad \text{if } |P| > |Q| \\ &= 0 \qquad \text{otherwise} \end{aligned} \tag{10}$$

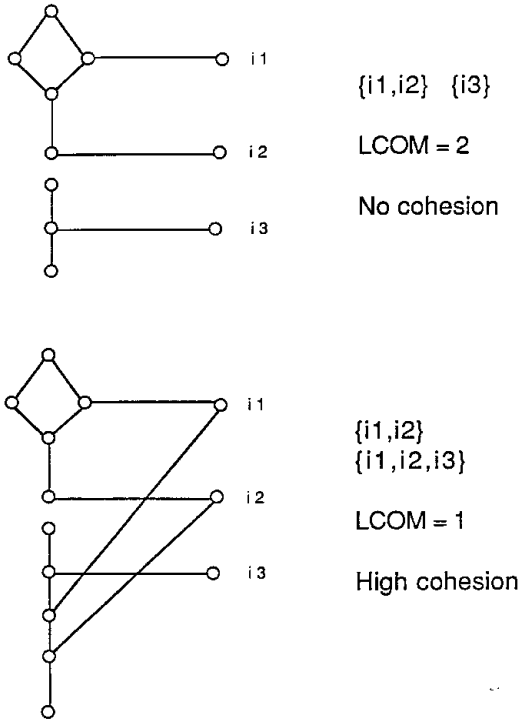

Fig. 5. Two classes drawn as DAGs. In the upper part of the figure, a class with two disparate methods is illustrated. This class should probably be divided and this is reflected in a 'high' value of LCOM of 2. A lower value of LCOM is seen in the lower example which has a much more cohesive nature.

In this case, the revised values of LCOM for the two examples of Fig. 5 are (i) LCOM $= 1$ ($|P| = 1$, $|Q| = 0$) and (ii) LCOM $= 0$ ($|P| = 0$, $|Q| = 1$). In words, 'LCOM is a count of the number of method pairs whose similarity is 0 (i.e. $\sigma()$ is a null set) minus the count of method pairs whose similarity is not zero'. Thus, when there is no cohesion, we expect the cardinality of the set P (the number of pairs which have no similarity) to be high and of set Q (the number of pairs which have some similarity) to be low or zero – and thus LCOM has a large value.

Another test of the validity and usefulness of a measure is whether two classes with equal values of LCOM are intuitively of the same cohesion. Consider Fig. 6(a). The set, $\{I_i\}$, has elements given as $I_1 = \{1\}, I_2 = \{2\}, I_3 = \{3\}, I_4 = \{5\}, I_5 = \{5, 6\}$. Intuitively, we expect a low cohesion. Our intuitive advice might go so far as to suggest this example would be better as four separate classes. Using Equation 10, we find that $|P| = 9$ and $|Q| = 1$ (since $I_4 \cap I_5$ is the only non-empty set). In other words, the LCOM value is 8 – which does seem to be significantly non-zero (but can we say sufficiently large?) to indicate low cohesion.

In contrast, consider the example in Fig. 6(b). It is easily shown that $|P| = 18$ and $|Q| = 10$. Every method accesses at least two data elements and all data are accessed, most of them by more than one method. The general impression is of a cohesive class; or more realistically that it would be difficult to

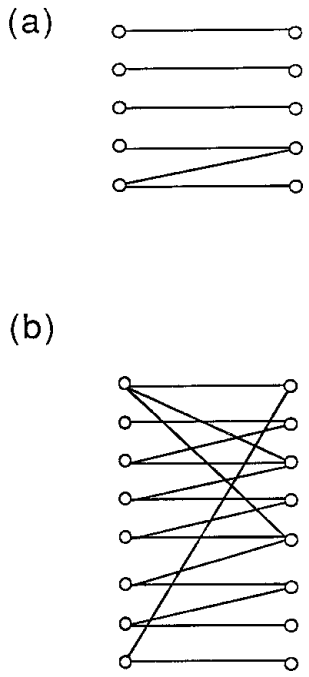

Fig. 6. Two extreme and more elaborate (cf. Fig. 5) examples are illustrated: (a) a highly non-cohesive class for which LCOM (1994) = 8 and (b) a highly cohesive class for which LCOM (1994) = 8.

see any easy way to divide the class into a number of smaller ones. Yet the value of LCOM (8) is identical to that of Fig. 6(a)! In that sense, one of the basic axioms of measurement theory, that a measure should be able to distinguish two dissimilar entities, is violated by the 1994 LCOM definition given in Equation 10.

Furthermore, it is of serious concern that, whilst a high value of LCOM implies low similarity and therefore low cohesion, a value of LCOM = 0 does not imply the reverse. If $|P| \leq |Q|$, LCOM = 0 and this can occur even for cases of obvious dissimilarity. For instance, consider extending Chidamber and Kemerer's example [3] by adding $I_4 = \{x, y, z, d\}$ (Fig. 7). When all four sets are considered, LCOM = 0 implying a good cohesive structure; yet intuitively M_1 and M_2 are a pair of cohesive methods as are M_3 and M_4 and the designer would suspect that two classes should be formed, not one[1]. Yet LCOM suggests high cohesiveness! In other words, this measure is not very discriminating (i.e. it again fails Weyuker's basic first axiom [8]) for low cohesive structures. A second example will reinforce the point. Intuitively a class with four methods accessing variables according to $I_i = \{a, b, c\}$; $I_2 = \{c, d, e\}$; $I_3 = \{e, f, g\}$; $I_4 = \{a, g, h\}$ does not seem to be a candidate for good cohesion. Yet LCOM is readily shown to be equal to zero.

One further observation on LCOM. Chidamber and Kemerer state that if no methods use any instance

[1]But NOT by subclassing as suggested by Chidamber and Kemerer [1, p. 204].

$I_1 = \{a, b, c, d, e\}$
$I_2 = \{a, b, e\}$
$I_3 = \{x, y, z\}$
$I_4 = \{x, y, z, d\}$

Consider a class supporting the first three sets.

$|P| = 2,\ |Q| = 1 \Rightarrow \text{LCOM} = 1$

Consider a class supporting all four sets.

$|P| = 3,\ |Q| = 3 \Rightarrow \text{LCOM} = 0$

Fig. 7. Extension of Chidamber and Kemerer's (1994) example for the calculation of LCOM [3].

variables, they have no similarity (agreed) and therefore LCOM = 0 [3]. But such a situation should be reflected in a large value of $|P|$ and a zero value for Q (and thus *large* LCOM to indicate the disparate nature of the class – although a class without data is a better sign of a non-OO program). It can therefore be concluded from Chidamber and Kemerer's discussion [3], that a zero value of LCOM (1994) could either be (1) a highly cohesive class, (2) a not very cohesive class or (3) a class with no cohesion at all! Clearly, much clarification and research is needed to devise a useful measure of internal cohesion.

It is interesting to note that, despite errors in their formulation, these six metrics were used in an empirical study of two companies, one using Smalltalk, one using C++ [2, 3] and also by Sharble and Cohen [13]. In many commentaries and applications of LCOM [21, 23] it is the 1991 definition of Equation 5 that is supposedly used.

3. A new measure for 'LCOM'

There are two serious problems with the 1994 version of LCOM (and of course the 1991 version violates all known axioms of measurement giving all values equally zero!): that there are a large number of dissimilar examples, all of which will give a value of LCOM = 0 (which, in fairness, is noted in footnote 28 of [3]). Hence, while a large value of LCOM suggests poor cohesion; a zero value does not necessarily indicate good cohesion. Secondly, there is no guideline on the interpretation of any particular value. Is a value of 8 an indicator of medium, low or abysmal cohesion? Indeed, in Fig. 6, we indicated that such a value might belong to a class requiring splitting (abysmal cohesion – Fig. 6(a)) or to a fairly cohesive class (Fig. 6(b)).

This suggests that the requirements for LCOM (and indeed any other measure) *must* include (i) the ability to give values across the full range and not for any specific value (in the above a preferred value of zero) to have a higher probability of attainment than any other, all other things being equal. (ii) Secondly, the measure must give values which can be uniquely interpreted in terms of cohesion. Our suggestion here is to make the measure have values on a percentage range. Thus, we consider the notion of 'perfect cohesion' and then present any particular datum as a fraction/percentage of that perfect value.

In the following, we consider a set of m methods accessing a total of a data/attributes. Perfect cohesion is considered to be when all methods access all attributes. For this we expect our new

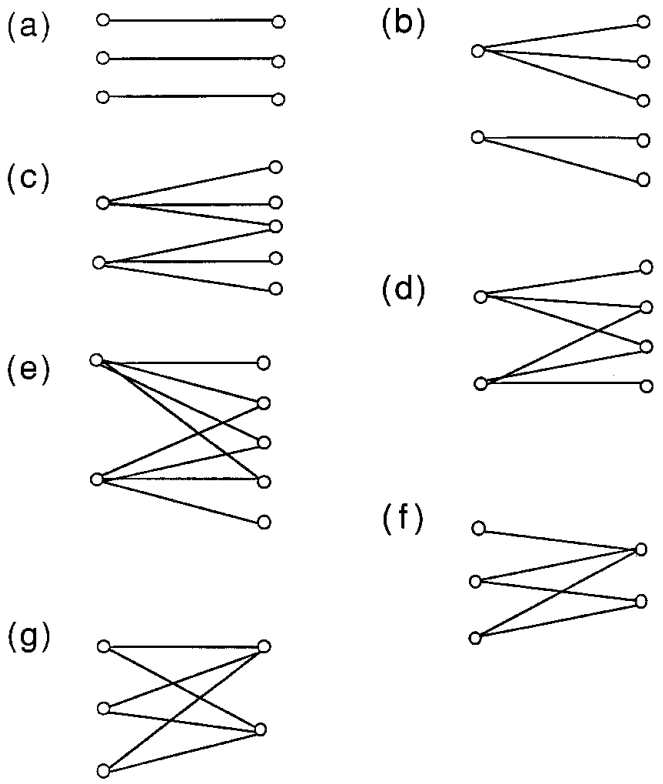

Fig. 8. Seven graded examples (the values of the three versions of LCOM* are given in Table 2). Parts (a) and (b) should clearly be divided into two. At the other extreme, part (g) is maximally cohesive in which all methods access all attributes. The intermediate parts (c–f) show various degrees of cohesion.

LCOM, call it LCOM*, to have a (fractional) value of 0. At the opposite end of the spectrum, we consider that each method only accesses a single variable. In this case, we expect LCOM* = 1 (and of course $m = a$).

Consider a set of methods, $\{M_i\}$ ($i = 1, \ldots, m$) accessing a set of attributes, $\{A_j\}$ ($j = 1, \ldots, a$). Let the number of attributes accessed by each method, M_i, be written as $\alpha(M_i)$ and the number of methods which access each datum be $\mu(A_j)$. The simplest formula which gives the properties discussed above (a value of zero with full cohesion and a value of 1 for no cohesion) is

$$\text{First version of} \quad LCOM^* = \frac{\left(\frac{1}{a}\sum_{j=1}^{a}\mu(A_j)\right)\left(\frac{1}{m}\sum_{i=1}^{m}\alpha(M_i)\right) - am}{1 - am} \tag{11}$$

Seven examples are shown in Fig. 8. The values of this First Version of LCOM* are given in the first entry of Table 2. In both portions (a) and (b), there is a need to split the class; although this version (Equation 11) differentiates the two cases (some might think unnecessarily).

Table 2. Values of three versions of LCOM* from Equations 11–13 for the examples of Fig. 8

	(a)	*(b)*	*(c)*	*(d)*	*(e)*	*(f)*	*(g)*
Equation 11	1	0.83	0.71	0.50	0.40	0.37	0
Equation 12	1	1.00	0.80	0.50	0.40	0.25	0
Equation 13	0.67	0.50	0.40	0.25	0.20	0.17	0

In one sense, it could be argued that so long as data are accessed by all the methods, it is only necessary to consider the values of the $\mu(A_j)$ and not the $\alpha(M_i)$. In addition, we note that, necessarily, $\Sigma_{j=1}^{a}\,\mu(A_j) = \Sigma_{i=1}^{m}\alpha(M_i)$ such that the first term of Equation 11 is essentially quadratic. The second new approach has the added advantage of simplifying the metric from a counting point of view such that

$$\text{Second version of} \quad LCOM^* = \frac{\left(\frac{1}{a}\sum_{j=1}^{a}\mu(A_j)\right) - m}{1-m} \tag{12}$$

Not only are the values easier to calculate but this second version assigns a similar value for cases (a) and (b) – the case for splitting – of unity indicating extreme lack of cohesion.

A third version, intended to be a clarification of the earlier 1991 LCOM and not the 1994 version as are Equations 11 and 12, is given in words by Graham [12, p. 409] as ‘‘the percentage of methods that do not access an attribute, averaged over all attributes’’ i.e.

$$\text{Third version of} \quad LCOM^* = \frac{1}{a}\sum_{j=1}^{a}\frac{m-\mu(A_j)}{m} \tag{13}$$

This was based on the way LCOM was implemented in the McCabe Tools software product [25]. Values derived from Equation 13 are given in the third line of Table 2. It can be seen that this has a correct lower limit but for a fully uncohesive class LCOM* $\ll 1$. This is because in a fully uncohesive class, such as Fig. 8(a), there is always likely to be one method-attribute connection resulting in a non-zero value for ‘‘the percentage of methods that do not access [that] attribute’’.

4. Conclusions

The CK metrics have provided a major contribution to the initiation of research into object-oriented metrics. Nevertheless, errors and confusion in their definition may hamper development of the field. Here, we have demonstrated that the original (1991) versions of WMC, CBO and LCOM were ambiguous and/or in error. We note that the revised (1994) versions do nothing to clarify the ambiguity of WMC and the new definition of CBO (which now includes inheritance) seems to confuse two major types of coupling in object-oriented systems: peer-to-peer coupling (collaboration) and coupling by inheritance. Rather we recommend that these different types of coupling are kept separate and that different metrics are introduced to allow for (i) essential coupling (binary measure between any pair of object classes), (ii) accesses of several methods in the one class and (iii) multiple accesses of the same method from the one class. Measures MPC and RFC augment the basic fan-out measure (= 1991 version of CBO). In other words,

CBO (1994 version) confuses too many characteristics of object-oriented systems and a suite of metrics (at least fan-out, RFC, MPC, DIT and NOC) are required to adequately measure coupling in these systems.

Coupling measures need to be augmented by cohesion measures. The 1991 version of LCOM has been shown to be useless and wrongly defined. A more useful measure has been shown here to be

$$\text{LCOM}' = |I_i \cap I_j = \emptyset,\ \forall i, j, i \neq j|$$

which counts the number of empty sets formed from taking the intersection of pairs of variable sets $\{I_i\}$.

The 1994 version of LCOM is an improvement but still violates the basic axioms of a good measure, being unable to distinguish classes of very different cohesion. A new and simplified version of LCOM is proposed which gives cohesion based on attribute accesses so that

$$\text{LCOM}^* \frac{\left(\frac{1}{a}\sum_{j=1}^{a} m(A_j)\right) - m}{1 - m}$$

It is recommended that these definitions of coupling and cohesion are theoretically superior to those proposed by Chidamber and Kemerer. They exhibit useful properties. The next stage in the research is the collection of data to assess their usefulness in connection as possible measures of external characteristics [17]. This has already commenced. The 'Metrics Club' is an industry-focus group which pledges to collect these metrics as well as metrics for task points [12] in a worldwide co-ordinated effort to advance the subject of industrial-strength object-oriented metrics research.

Acknowledgements

This research was funded in part by grants from the Australian Research Council. This is Contribution no 95/16 of the Centre for Object Technology Applications and Research of the University of Technology, Sydney.

References

1. S. Chidamber and C. Kemerer. Towards a metric suite for object-oriented design, in *Proc. OOPSLA'91, Sigplan Notices*, **26**(11) (1991) 197–211.
2. S.R. Chidamber and C.F. Kemerer. A metrics suite for object oriented design, CISR Working Paper No. 249, MIT Sloan School of Management, Cambridge, MA, 1993.
3. S. Chidamber and C. Kemerer. A metrics suite for object oriented design, *IEEE Trans. Software Eng*., **20** (1994) 476–493.
4. M. Lorenz and J. Kidd. *Object-Oriented Software Metrics. A Practical Guide* (Prentice Hall, Englewood Cliffs, NJ, 1994).
5. B. Henderson-Sellers and J.M. Edwards. *BOOKTWO of Object-Oriented Knowledge: The Working Object* (Prentice Hall, Sydney, 1994).
6. B. Henderson-Sellers. Identifying internal and external characteristics of classes likely to be useful as structural complexity metrics, *OOIS'94 Proceedings*, D. Patel, Y. Sun and S. Patel, eds (Springer-Verlag, London 1994) pp. 227–230.

7. V.R. Basili and H.D. Rombach. The TAME project: towards improvement-orientated software environments, *IEEE Trans. Software Eng*., **14** (1988) 758–773.
8. E. Weyuker. Evaluating software complexity measures, *IEEE Trans. Software Eng*., **14** (1988) 1357–1365.
9. H. Zuse, *Software Complexity: Measures and Methods* (Walter de Gruyter, Berlin, 1990).
10. N.E. Fenton. *Software Metrics: a Rigorous Approach* (Chapman and Hall, London, 1991).
11. M. Bunge. *Treatise on Basic Philosophy: Ontology I: The Furniture of the World* (Reidel, Boston, 1977).
12. I.M. Graham, *Migrating to Object Technology* (Addison-Wesley, Wokingham, 1995).
13. R.C. Sharble and S.S. Cohen. The object-oriented brewery: a comparison of two object-oriented development methods, *ACM SIGSOFT Software Engineering Notes*, **18**(2) (1993) 60–73.
14. T.J. McCabe. A complexity measure, *IEEE Trans. Software Eng*., **2** (1976) 308–320.
15. B. Henderson-Sellers and D. Tegarden. The theoretical extension of two versions of cyclomatic complexity to multiple entry/exit modules, *Software Quality Journal*, **3** (1994) 253–269.
16. N.I. Churcher and M.J. Shepperd. Comments on 'A metrics suite for object oriented design', *IEEE Trans. Software Eng*., **21** (1995) 263–265.
17. N.E. Fenton. Software measurement: a necessary scientific basis, *IEEE Trans on Software Engng*, **20** (1994) 199–206.
18. Bezant. *SOMATiK User Guide* (Bezant Object Technologies, Wallingford, Oxon, 1995).
19. G. Booch. *Object Oriented Design with Applications* (Benjamin/Cummings, Menlo Park, CA, 1991).
20. E.V. Berard. *Essays on Object-Oriented Software Engineering* (Prentice Hall, Englewood Cliffs, NJ, 1992).
21. W. Li and S. Henry. Object-oriented metrics that predict maintainability, *J. Systems Software*, **23** (1993) 111–122.
22. C. Rajaraman and M.R. Lyu. Reliability and maintainability related software coupling metrics in C++ programs, *IEEE*, 303–311.
23. R. Kolewe. Metrics in object-oriented design and programming, *Software Development*, October (1993) 53–62.
24. J.B. Fraleigh. *A First Course in Abstract Algebra* (Addison-Wesley, Reading, MA, 1967).
25. T.J. McCabe. *OO Tool Features New Metrics, The Outlook* (McCabe and Associates, Columbia, Maryland, 1993).