An Evaluation of the MOOD Set of Object-Oriented Software Metrics

Rachel Harrison, *Member*, *IEEE Computer Society*,
Steve J. Counsell, *Member*, *IEEE Computer Society*, and Reuben V. Nithi

Abstract—This paper describes the results of an investigation into a set of metrics for object-oriented design, called the MOOD metrics. The merits of each of the six MOOD metrics is discussed from a measurement theory viewpoint, taking into account the recognized object-oriented features which they were intended to measure: encapsulation, inheritance, coupling, and polymorphism. Empirical data, collected from three different application domains, is then analyzed using the MOOD metrics, to support this theoretical validation. Results show that (with appropriate changes to remove existing problematic discontinuities) the metrics could be used to provide an overall assessment of a software system, which may be helpful to managers of software development projects. However, further empirical studies are needed before these results can be generalized.

Index Terms—Empirical software engineering, validating software metrics, assessing object-oriented software.

1 Introduction

Analyzing object-oriented software in order to evaluate its quality is becoming increasingly important as the paradigm continues to increase in popularity. However, widespread adoption of object-oriented metrics in numerous application domains should only take place if the metrics can be shown to be theoretically valid, in the sense that they accurately measure the attributes of software which they were designed to measure [1], [2], [3], [4], and have also been validated empirically [3], [4], [5]. Without such a basis, we will not be able to draw meaningful conclusions from analyses of metrics data. In this paper, we consider a set of metrics for object-oriented design called the MOOD metrics [6], [7], [8], [9] from a measurement theory viewpoint, and then consider their empirical evaluation using three different projects. The strength of this investigation centers on the consideration of a number of criteria [2] for valid metrics:

1) For an attribute to be measurable, it must allow different entities to be distinguished from one another.
2) A valid metric must obey the *representation condition* [1]; that is, it must preserve all intuitive notions about the attribute and the way in which the metric distinguishes between entities.
3) Each unit of an attribute contributing to a valid metric is equivalent.
4) Different entities can have the same attribute value (within the limits of measurement error).

A distinction should be made between *direct measurement* of an attribute, which is measurement which does not depend on any other attribute [1], and *indirect measurement* which involves the measurement of one or more other attributes. We also need to distinguish between *internal attributes* of a product or process (those attributes which can be measured purely in terms of the product itself), and *external attributes* of a product or process (those attributes which can only be measured with respect to how the product or process relates to entities in its environment [1]). Managers are often particularly interested in measuring external attributes such as reliability and maintainability. However, OO software metrics are often based on internal (low-level) attributes, under the assumption that they are related to external (high-level) attributes. According to the framework of Kitchenham et al. [2], indirect metrics should exhibit the following properties in addition to those listed earlier:

1) The metric should be based on an explicitly defined model of the relationship between certain attributes (usually, relating internal and external attributes).
2) The model must be dimensionally consistent.
3) The metric must not exhibit any unexpected discontinuities.
4) The metric must use units and scale types correctly.

Alternative views of metrics validation abound [10], [11], [12], [3]. In particular, there is some controversy concerning the use of scale types, and admissible statistical operations. Such issues are beyond the scope of this paper; we merely note that there is general consensus on the *need* to validate metrics, and draw attention to the approach adopted here.

The following sections present a theoretical validation of the MOOD metrics under the headings *encapsulation*, *inheritance*, *coupling*, and *polymorphism*. We then consider some empirical results from three different projects with these previous analyzes in mind, and finally draw some conclusions.

2 Theoretical Validation

2.1 Encapsulation

The Method Hiding Factor (MHF) and Attribute Hiding Factor (AHF) metrics were proposed jointly as measures of

• *R. Harrison, S.J. Counsell, and R.V. Nithi are with the Department of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, U.K. E-mail: {rh, sjc, rvn95r}@ecs.soton.ac.uk.*

Manuscript received 13 May 1997; revised 18 Feb. 1998.
Recommended for acceptance by D. Ross Jeffery.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 105049.

encapsulation [9], and so should be considered together. Earlier publications had proposed MHF and AHF as measures of "the use of the information hiding concept" [7], [8]. MHF is defined formally [9] as:

$$\frac{\sum_{i=1}^{TC}\sum_{m=1}^{M_d(C_i)}(1-V(M_{mi}))}{\sum_{i=1}^{TC}M_d(C_i)}$$

where $M_d(C_i)$ is the number of methods declared in a class, and

$$V(M_{mi}) = \frac{\sum_{j=1}^{TC} is\_visible(M_{mi}, C_j)}{TC-1}$$

where TC is the total number of classes, and

$$is\_visible(M_{mi}, C_j) = \begin{cases}1 & iff \quad j \neq i \wedge C_j \; may\ call\ M_{mi}\\ 0 & otherwise\end{cases}$$

Thus, for all classes, C_1, C_2, ..., C_n, a method counts as 0 if it can be used by another class, and 1 if it cannot. The total for the system is divided by the total number of methods defined in the system, to give the percentage of hidden methods in the system. AHF was defined in an analogous fashion, but using attributes rather than methods. Note that the definitions of MHF and AHF cause discontinuities for systems with only one class.

When considering the validity of these metrics as measures of encapsulation, we must first define our terms. *Data encapsulation* (or, more simply *encapsulation*) is often taken to mean the power of a language to hide implementation details through (for example) the separate compilation of modules, the separation of interface from implementation, the use of opaque types [13], etc. *Information hiding* [14], on the other hand, can be defined in terms of the visibility of methods and/or attributes to other code ("Every module... is characterized by its knowledge of a design decision which it hides from all others" [14]). Information can be hidden without being encapsulated, and vice-versa.

Thus, if we consider encapsulation to be related to compilation facilities, it is difficult to agree that MHF and AHF could be used as direct or indirect measures of *encapsulation*. Over the past three decades, *information hiding* has become universally recognized as referring to the extent to which methods and attributes are visible to code in other classes. Thus the attribute which represents information hiding (and which we would like to measure) is 'code visibility.' Both MHF and AHF use code visibility to measure information hiding, and thus should be validated using all eight validation criteria.

For systems written in C++, the calculation of MHF is complicated by the existence of protected methods; this adjustment is problematic. For a protected method in C++, the method is counted as a fraction between 0 and 1, calculated as:[1]

$$\frac{\text{number of classes not inheriting the method}}{\text{total number of classes} - 1}$$

The validation criteria are reiterated in summary form to guide the reader through the theoretical validation. For direct metrics:

1. The denominator has the value 1 subtracted from the total number of classes because the base class under consideration should not be included.

1) For an attribute to be measurable, it must allow different entities to be distiguished from one another.
2) A valid metric must obey the representation condition.
3) Each unit contributing to a valid metric is equivalent.
4) Different entities can have the same attribute value.

Additionally, for indirect metrics:

1) The metric should be based on an explicitly defined model of the relationship between certain attributes.
2) The model must be dimensionally consistent.
3) The metric must not exhibit any unexpected discontinuities.
4) The metric must use units and scale types correctly.

Assuming that this metric's discontinuity is taken into account, we can agree that MHF and AHF meet three of the four criteria for direct metrics as proposed by Kitchenham, et al. listed above. However, the third criteria states that 'each unit of an attribute contributing to a valid metric is equivalent'; this asks us whether all methods (or attributes, for AHF) are equivalent, as far as information hiding is concerned. At first it would seem that this question must be answered in the negative; the need to hide a method or attribute depends to a certain extent on its functionality, the desire to facilitate reusability, etc. However, MHF and AHF are intended to measure the relative *amount* of information hiding, (and not the *quality* of the information hiding design decisions), and so we conclude that these metrics do meet the first four criteria. The criteria for indirect metrics are also met, if the metric's discontinuity in the case of single-class systems is accounted for.

2.2 Inheritance

The Method Inheritance Factor (MIF) and Attribute Inheritance Factor (AIF) metrics can be defined as follows:

$$\frac{\sum_{i=1}^{TC}M_i(C_i)}{\sum_{i=1}^{TC}M_a(C_i)}$$

where

$$M_a(C_i) = M_d(C_i) + M_i(C_i)$$

and

$M_d(C_i)$ = the number of methods declared in a class,
$M_a(C_i)$ = the number of methods that can be invoked in association with C_i,
$M_i(C_i)$ = the number of methods inherited (and not overridden) in C_i.

For MIF, for each class C_1, C_2, ..., C_n, a method counts as 0 if it has not been inherited and 1 if it has been inherited. The total for the system is divided by the total number of methods, including any which have been inherited (i.e., methods which are inherited are counted as belonging to their base class as well as to all inheriting subclasses). AIF is defined in an analogous fashion. Thus, MIF and AIF measure *directly* the number of inherited methods and attributes respectively as a proportion of the total number of methods/attributes. It is intuitively clear that there is a relationship between the relative amount of inheritance in a system and the number of methods/attributes which have been inherited.

We will consider each of the criteria relating to a direct metric in turn, using MIF to represent both MIF and AIF:

1) Two programs with different amounts of inheritance will have different MIF values.
2) A program with more inheritance will have larger MIF values than one with low inheritance.
3) Each method which is inherited contributes in an equivalent way to MIF. (Of course, for a large system, the addition of a new inherited method will result in an increase in the MIF which will be smaller than the increase in the MIF for a smaller system, because of the way that the metric is normalized).
4) Different programs can have the same MIF value.

Thus, within the validation framework of Kitchenham, et al. [2], it would appear that both MIF and AIF are valid as direct measures of inheritance.

2.3 Coupling

The Coupling Factor (CF) metric was proposed as a measure of coupling between classes [7], [8], excluding coupling due to inheritance. CF has been defined formally [9] as:

$$\frac{\sum_{i=1}^{TC}\left[\sum_{j=1}^{TC} is\_client(C_i, C_j)\right]}{TC^2 - TC}$$

where

$$is\_client(C_c, C_s) = \begin{cases} 1 & iff \quad C_c \Rightarrow C_s \wedge C_c \neq C_s \\ 0 & otherwise \end{cases}$$

and $C_c \Rightarrow C_s$ represents the relationship between a client class, C_c, and a supplier class, C_s.

CF is calculated by considering all possible pairwise sets of classes, and asking whether the classes in the pair are related, either by message passing or by semantic association links (reference by one class to an attribute or method of another class). These relationships are considered to be equivalent as far as coupling is concerned [7], [9]. Thus, CF is a direct measure of the size of a relationship between two classes, for all pairwise relationships between classes in a system.

There are two possible approaches to validating CF. Firstly, we could consider CF to be a *direct* measure of interclass coupling. Alternatively, we could consider CF to be an *indirect* measure of the attributes to which it was said to be related [8]:

- complexity
- lack of encapsulation
- lack of reuse potential
- lack of understandability
- lack of maintainability

Considering CF as a *direct* measure first of all, we again turn to our list of four criteria. CF will enable us to distinguish between different programs with different levels of coupling. Intuitively, we would agree that a system with a high level of interclass coupling will have a high CF value. Adding an extra interclass relationship to a system will contribute in an equivalent way to the CF metric. We also note that different programs could have the same CF value, in which case they would exhibit the same amount of coupling. Thus, CF is a valid measure of interclass coupling.

If we now consider CF as an *indirect* measure of the attributes listed above, does a high degree of coupling indicate a high degree of complexity? Possibly not, as it is feasible to construct a simple system which is highly coupled, or a complex system with little or no coupling. If we consider encapsulation to mean the ability to compile modules separately, then the relationship between encapsulation and CF is not clear either. As far as reuse is concerned, we may have a low degree of coupling, but high degree of reuse (through inheritance, for example). Understandability is a very similar attribute to complexity; we may find a class which is not coupled but is still not understandable, and similarly for maintainability. Consequently, it is difficult to pronounce on the validity of CF as a measure for these external attributes. Empirical evidence would help to clarify the significance of CF in this respect.

2.4 Polymorphism

The Polymorphism Factor (PF), metric was proposed as a measure of polymorphism potential. It has been defined formally as follows [9]:

$$\frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC}\left[M_n(C_i) \times DC(C_i)\right]}$$

where

$$M_d(C_i) = M_n(C_i) + M_o(C_i)$$

and

$M_n(C_i)$ = the number of new methods,

$M_o(C_i)$ = the number of overriding methods,

$DC(C_i)$ = the descendants count

(the number of classes descending from C_i).

We can summarize the definition of PF informally as follows: PF is the number of methods that redefine inherited methods, divided by the maximum number of possible distinct polymorphic situations [7] (the latter represents the case in which all new methods in a class are overridden in all its derived classes). Thus, PF is an indirect measure of the relative amount of dynamic binding in a system.

Considering the validity of PF as an indirect metric, we notice that the denominator includes, as a multiplier, the number of descendant classes of a base class, and so the value of PF for a system without any inheritance will always be undefined. Thus, the metric exhibits an unexpected discontinuity, giving an undefined result where a result of 0 would have been expected. Consequently, we conclude that PF is not a valid metric.

Assuming that the metric's definition is revised to remove the discontinuity, we can validate PF using Kitchenham et al.'s eight validation criteria, with an overriding method as the attribute unit. The contribution of all overriding methods to the relative amount of polymorphism can be considered to be equivalent. The model is dimensionally consistent (remembering that we are working here with *relative* quantities), and the metric uses scale types and units appropriately (remembering that fractions must be reduced to canonical form before meaningful comparisons can be made). Hence, if the metrics discontinuity is removed, it would be theoretically valid according to this framework.

3 Data Analysis

The summary size metrics for three releases (R1, R2, and R3) of our laboratory electronic retail system (ERS), developed in collaboration between the Universities of Southampton and Manchester [15], [16], and the second release of a suite of image processing programs (EFOOP2) [17] are shown in Table 1. Note the difference in the total numbers of methods and attributes between R1 and R2 of the ERS. These differences can be explained by additional requirements. The total number of methods and attributes stabilized for R3. Table 2 shows the results of collecting the MOOD metrics for the three releases (R1, R2, and R3) of our ERS.

TABLE 1
Size Metrics, ERS Released
(R1, R2, and R3) and EFOOP2)

	R1	R2	R3	EFOOP2
LOC	1149	2536	2753	8977
Total Classes	4	13	13	12
Total Methods	20	96	96	134
Total attributes	8	35	35	33

TABLE 2
The MOOD Metrics, ERS Releases
(R1, R2, and R3) and EFOOP2

	R1 (%)	R2 (%)	R3 (%)	EFOOP2 (%)
AHF	100	100	100	100
MHF	0	20.4	20.4	6.3
AIF	12.5	0	0	0
MIF	9.1	0	0	0
CF	0	5.8	5.8	3.0
PF	60	*Undefined*	*Undefined*	*Undefined*

The Attribute Hiding Factor (AHF) metric for all of these systems has its maximum value of 100 percent, indicating that all the attributes were declared as private. Method Hiding Factor (MHF), on the other hand, has relatively low values, indicating a lack of information hiding. Inheritance was not utilized at all, with the exception of R1 of ERS, as shown by Method Inheritance Factor (MIF) and Attribute Inheritance Factor (AIF). The undefined Polymorphism Factor (PF) values also reflect this lack of inheritance in the other systems. All of the systems displayed only small amounts of interclass coupling (shown by the Coupling Factor (CF)), possibly pointing to well-designed systems. It seems that developers and maintainers may be able to capture a high-level view of a system's architecture from these metrics. However, interpretation of numerical values will remain open to debate until sufficient empirical validations have been performed. This will be very expensive, because the metrics are defined at the systems level; a large number of systems will be required for a meaningful statistical analysis.

Tables 3 and 4 give the MOOD metrics and code metrics, respectively, for nine samples of a large commercial retail system, with sizes ranging from 16 to 47 KLOC.

From Table 3, we can see that the values for the AHF vary between 44 to 68 percent. This is interesting as a figure of 50 percent would suggest an even balance between the public and private data attributes, and is in contrast with the high percentage values for our ERS and EFOOP2 systems, in which all attributes were declared privately. Ideally, the AHF should be close to 100 percent, to adhere to the concept of information hiding.

The values for the Method Hiding Factor (MHF) vary between 8 to 25 percent. These low values, which are similar to those reported for the ERS and EFOOP2 systems, indicate a low degree of information hiding, possibly suggesting a lack of abstraction at the design stage.

The Attribute Inheritance Factor (AIF) varies between 11 to 47 percent. These are also rather low, suggesting only a moderate use of inheritance. Similar results were obtained for the Method Inheritance Factor (MIF) values (14 to 46 percent). Mean values of 56.2 and 73.5 percent for AIF and MIF, respectively are cited by Abreu [7]. Low values for the inheritance factors for the second and third ERS releases and EFOOP2 support this lack of utilization of inheritance.

The Coupling Factor metric ranges from 3 to 6 percent, suggesting little interclass coupling. Abreu suggests that CF should be neither too low, nor too high [7]. Low coupling reduces potentially harmful side-effects such as unnecessary dependencies and limited reuse. (Similar low values were obtained for the ERS and EFOOP2 systems). However, a very low value of CF (0 percent) indicates that a system has no interclass coupling, which might point to a pathological system in which classes only communicate via inheritance, or in which there is excessive code duplication. On the other hand, a CF of 100 percent may also indicate a problematic communications infrastructure; excessive coupling implies that software will be difficult to maintain, evolve, and reuse.

The values for the Polymorphism Factor metric range from 3 to 9 percent; these low values are fairly typical [7] and unsurprising considering the relatively moderate use of inheritance.

4 Conclusions

Our investigation into the validity of the six MOOD metrics has led us to conclude that, as far as *information hiding*, *inheritance*, *coupling*, and *dynamic binding* are concerned (with appropriate changes to rectify existing problematic discontinuities) the six MOOD metrics can be shown to be valid measures within the context of this theoretical framework [2]. Proposals concerning the validation of software metrics have been published frequently ([18], [10], [19], [4], [20], [21]) and continue to be debated. The main problems which we encoutered during our theoretical validation stemmed from imprecise definitions of the attributes (and contributing units) to be measured; hopefully such problems will be alleviated as the importance of defining terms precisely becomes accepted. The community is also beginning to accept that software metrics must be not only validated theoretically, but also empirically, through observation of their use during software development projects both in the laboratory and in the real world [22].

Our empirical results indicate that the MOOD metrics operate at the systems level. Comparing them with those of Chidamber and Kemerer [11], we see that the two sets are complementary, offering different assessments of a system.

TABLE 3
THE MOOD METRICS, FROM NINE COMMERCIAL SAMPLES

System Label	1 (%)	2 (%)	3 (%)	4 (%)	5 (%)	6 (%)	7 (%)	8 (%)	9 (%)
AHF	45.9	66.7	66.3	44.0	62.5	67.5	52.4	48.5	50.8
MHF	10.1	7.7	16.4	9.5	25.4	15.4	15.8	15.7	15.4
AIF	17.1	11.3	15.3	30.6	46.8	26.1	19.7	36.6	32.0
MIF	15.2	14.3	20.7	27.4	45.5	33.6	22.5	36.5	26.5
CF	3.5	3.5	3.8	6.3	3.1	4.5	5.4	4.9	4.6
PF	4.3	5.4	8.9	2.9	6.7	4.5	6.6	6.2	6.4

TABLE 4
PRODUCT METRICS, FROM NINE COMMERCIAL SAMPLES

System Label	1	2	3	4	5	6	7	8	9
LOC	15837	23750	47106	23154	20747	44930	28582	19254	20085
Total Classes	65	57	91	51	154	92	71	69	74
Total Methods	1446	1535	2141	1420	2814	2224	1978	1815	1876
Total Attributes	537	876	1178	538	1113	1132	839	675	700

For example, the coupling metrics (CBO and CF) are closely related, in that CBO offers a class-level view of coupling, whereas CF offers a systems-level view. However, the definition of CBO explicitly includes coupling via inheritance [11], (a change from the earlier definition [23]) whereas CF does not. The Chidamber and Kemerer metrics appear to be useful to designers and developers of systems, giving them an evaluation of a system at the class level [3], despite the fact that their correctness from a measurement theory perspective [1], [24], continues to be debated [25], [26], [27]. Indeed, the validation of software metrics also continues to invoke controversy [2], [3], [12], [28], [29]. The MOOD metrics, on the other hand, could be of use to project managers, as the metrics operate at a systems level, providing an overall assessment of a system. However, their utility will continue to be questioned until a sufficient number of empirical validations have been performed at a systems level to establish causal relationships between the metrics and external quality attributes of systems, such as reliability, maintainability, testability, etc,. Without further empirical validation, we cannot be sure that it is worth paying attention to these metrics.

ACKNOWLEDGMENTS

The authors wish to acknowledge the support of the EPSRC, which has funded the MOOPS project (GR/K83021) and also the University of Southampton for funding a Research Studentship to participate in this work.

REFERENCES

[1] N.E. Fenton, "Software Measurement: A Necessary Scientific Basis," *IEEE Trans. Software Eng.*, vol. 20, no. 3. pp. 199-206, 1994.

[2] B.A. Kitchenham, N. Fenton, and S. Lawrence Pfleeger, "Towards a Framework for Software Measurement Validation," *IEEE Trans. Software Eng.*, vol. 21, no. 12, pp. 929-944, 1995.

[3] V.R. Basili, L.C. Briand, and W.L. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Trans. Software Eng.*, vol. 22, no. 10, pp. 751-761, 1996.

[4] N.F. Schneidewind, "Methodology for Validating Software Metrics," *IEEE Trans. Software Eng.*, vol. 18, no. 5, pp. 410-422, 1992.

[5] L. Briand, K. El Emam, and S. Morasca, "Theoretical and Empirical Validation of Software Metrics," ISERN Technical Report 95-03, 1995.

[6] F. Brito e Abreu, and R. Carapuca, "OO Software Engineering: Measuring and Controlling the Development Process," *Proc. Fourth Int'l Conf. Software Quality*, Virginia, 1994.

[7] F. Brito e Abreu, M. Goulao, and R. Estevers, "Toward the Design Quality Evaluation of OO Software Systems," *Proc. Fifth Int'l Conf. Software Quality*, 1995.

[8] F. Brito e Abreu, "The MOOD Metrics Set," *Proc. ECOOP'95 Workshop on Metrics*, 1995.

[9] F. Brito e Abreu and W. Melo, "Evaluating the Impact of OO Design on Software Quality," *Proc. Third Int'l Software Metrics Symp.*, Berlin, 1996.

[10] E.J. Weyuker, "Evaluating Software Complexity Measures," *IEEE Trans. Software Eng.*, vol. 14, pp. 1,357-1,365, 1988.

[11] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Software Eng.*, pp. 467-493, 1994.

[12] L. Briand, S. Morasca, and V.R. Basili, "Property-Based Software Engineering Measurement," *IEEE Trans. Software Eng.*, vol. 22, no. 1, pp. 68-85, 1996.

[13] N. Wirth, *Programming Modula-2*. Springer-Verlag, 1988.

[14] D.L. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules," *Comm. ACM*, vol. 15, no. 12, pp. 1,053-1,058, 1972.

[15] R. Harrison, R. Nithi, K.T. Phalp, L.G. Samaraweera, and A.P. Smith, "An Empirical Study of a Software Maintenance Process," *Proc. Software Quality Conf.*, Dundee, July 1996.

[16] R.M. Greenwood, B.C. Warboys, and J. Sa, "Cooperating Evolving Components A Rigorous Approach to Evolving Large Software Systems," *Proc. 18th Int'l Conf. Software Eng.*, Mar. 1996.

[17] R. Harrison, L.G. Samaraweera, M.R. Dobie, and P.H. Lewis, "Comparing Programming Paradigms: An Evaluation of Functional and Object-Oriented Programs," *Software Eng. J.*, vol. 11, pp. 247-254, July 1996.

[18] S. Conte, V. Shen, and H. Dunsmore, *Software Eng. Metrics and Models*. Benjamin-Cummings, 1986.

[19] A.C. Melton, D.A. Gustafson, J.M. Bieman, and A.L. Baker, "A Methematical Perspective for Software Measures Research," *Software Eng. J.*, vol. 5, pp. 246-254, 1990.

[20] L.C. Briand and S. Morasca, "Property-Based Software Engineering Measurement," *IEEE Trans. Software Eng.*, vol. 22, no. 1, pp. 68-85, 1996.

[21] L.C. Briand, K. El Emam, and S. Morasca, "On the Application of Measurement Theory in Software Engineerng," *Empirical Software Engineering J.*, vol. 1, no. 1, pp. 61-88, 1996.

[22] V. Basili, "The role of Experimentation in Software Engineering: Past, Present and Future," *Proc. 18th ICSE*, pp. 442-449, 1996.

[23] S.R. Chidamber and C.F. Kemerer, "Towards a Metrics Suite for Object-Oriented Design, " *Proc. OOPSLA'91*, Phoenix, Ariz., pp. 197-211, 1991.

[24] N.E. Fenton and S. Lawrence Pfleeger, *Software Metrics, A Rigorous and Practical Approach*. International Thomson Computer Press, 1996.

[25] N.I. Churcher and M.J. Shepperd, “Comments on ‘A Metric Suite for Object-Oriented Design’,” *IEEE Trans. Software Eng.*, vol. 21, no. 3, pp. 263-265, 1995.

[26] M. Hitz and B. Montazeri, “C&K Metrics Suite: A Measurement Theory Perspective,” *IEEE Trans. Software Eng.*, vol. 22, no. 4, pp. 267-271, 1996.

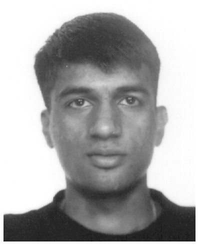
[27] B. Henderson-Sellers, L.L. Constantine, and I.M. Graham, “Coupling and Cohesion (Towards a Valid Metrics Suite for Object-Oriented Analysis and Design),” *Object-Oriented Systems*, vol. 3, no. 3, pp. 143-158, 1996.

[28] H. Zuse, “Reply to: Property-Based Software Engineering Measurement,” *IEEE Trans. Software Eng.*, vol. 23, no. 8, pp. 533, 1997.

[29] S. Morasca, L.C. Briand, E.J. Weyuker, and M.V. Zelkowitz, “Comments on: Towards a Framework for Software Measurement Validation,” *IEEE Trans. Software Eng.*, vol. 23, no. 3, pp. 187-195, 1997.

Rachel Harrison obtained her MA degree in mathematics from Oxford University, an MSc degree in computer science from London Univarsity, and a PhD degree in computer science from the University of Southampton. Dr. Harrison is a member of the engineering faculty at the University of Southampton. Her current research interests center around empirical software engineering, particularly measurement and modeling of the object-oriented paradigm, and the evaluation of formal methods and hypermedia systems. Dr. Harrison is a member of the IEEE Computer Society, the ACM, and the BCS.

Steve J. Counsell obtained his BSc degree in computer studies from the Unversity of Brighton in 1987, and an MSc in systems analysis from the City University, London, in 1988. He has recently completed his PhD degree in computer science from Birkbeck College, University of London, 1998, and is currently a postdoctoral researcher in the Department of Electronics and Computer Science at Southampton. For several years, he worked in industry as a software engineer. His current research interests center around the measurement of the object-oriented paradigm, empirical software engineering, software reliability modeling, and business process modeling.



Reuben V. Nithi received his BSc degree in 1992 in computer science form the University of Southampton, England. He is presently a doctoral candidate in the Department of Electronics and Computer Science at the University of Southampton. His current research interests are in the areas of empirical software engineering, software quality modeling, and object oriented design metrics.