Empirical Software Engineering, 3, 255–273 (1998)

An Investigation into the Applicability and Validity of Object-Oriented Design Metrics

R. HARRISON rh@ecs.soton.ac.uk
Department of Electronics and Computer Science, University of Southampton, Southampton, UK

S. J. COUNSELL
Department of Electronics and Computer Science, University of Southampton, Southampton, UK

R. V. NITHI
Department of Electronics and Computer Science, University of Southampton, Southampton, UK

Abstract. In this paper we consider empirical evidence in support of a set of object-oriented software metrics. In particular, we look at the object oriented design metrics of Chidamber and Kemerer, and their applicability in different application domains. We briefly describe the metrics, and present our empirical findings, arising from our analysis of systems taken from a number of different application domains. Our investigations have led us to conclude that a subset of the metrics can be of great value to software developers, maintainers and project managers.

Keywords: object-oriented metrics, empirical software engineering, software quality

1. Introduction

In this paper, we focus on the applicability and validity of metrics specifically for application within an object-oriented environment. A large number of sets of such object-oriented metrics have been proposed (Chidamber and Kemerer, 1994; Brito e Abreu et al., 1995; Lorenz and Kidd, 1994; Whitmire, 1992; Lake and Cook, 1992). All seem to accept that object-oriented systems require a different approach to that of traditional metrics (Brooks, 1993; Chidamber and Kemerer, 1994; Wilde and Huitt, 1992; Tearden and Sheetz, 1992; Sheetz et al, 1991; Whitmire, 1992; Brito e Abreu and Carapuca, 1994; Moreau and Dominick, 1989; Pfleeger and Palmer, 1990). We concentrate on the metrics of Chidamber and Kemerer, referred to here as the C&K metrics (Chidamber and Kemerer, 1994), looking particularly at empirical evaluation of their use in the field.

We focus on the C&K metrics for a number of reasons. The set of metrics proposed by Chidamber and Kemerer represents a minimal set which addresses many of the main concerns of OO designers: coupling, cohesion, inheritance and class size, and was among the first to address such fundamental OO design issues. Previously, metrics focusing on OO design had remained largely neglected. Evidence of the utility of the C&K metrics has been supported empirically in two OO industrial sites, adding credibility to the claims for the utility of metrics (Chidamber and Kemerer, 1994). Since their inception, a number of criticisms have been made of the C&K metrics in terms of their validity and practicality (Churcher and Shepperd, 1995; Hitz and Montazeri, 1996; Henderson-Sellers, 1996; Graham, 1996). However, the C&K metrics have been instrumental in promoting a debate in

the metrics community about what constitutes a valid metric, and also spawned a large body of work in the areas of empirical software engineering. The C&K metrics have become the *de-facto* standard against which new proposed sets of metrics have been judged, compared and evaluated. Lessons learnt from the C&K metrics have been used by researchers in their quest to improve upon them. Finally, the C&K metrics have permitted researchers to learn more about OO systems, particularly about the features of OO systems which need to be urgently addressed.

In the sections which follow, we present a review of relevant literature in the field, summarise the C&K metrics and present empirical evidence from several different application domains. Finally, we draw some conclusions from our research.

2. Literature Review

A number of reports on the theoretical validation and empirical investigation of object-oriented design metrics have been published recently. These include a description of a theoretical framework, providing definitions of measurement concepts such as size, length, complexity, cohesion and coupling (Briand et al., 1996c). The framework is generic (not being specific to any particular software artifact), and is based on precise mathematical concepts.

Numerous authors have proposed sets of properties which valid software metrics should possess (Weyuker, 1988; Melton et al., 1990; Ejiogu, 1993; Conte et al., 1986; Schneidewind, 1992; Kearney et al., 1986; Brito e Abreu and Carapuca, 1994; Sheetz et al., 1991; Tsai et al., 1986; Lorenz and Kidd, 1989). More recently, Kitchenham et al have identified four theoretical criteria that a valid measure needs to satisfy (Kitchenham et al., 1995).

Providing empirical evidence of the behaviour of a set of metrics plays a key role in demonstrating the validity of those metrics (Briand et al., 1996b; Briand et al., 1997; Fenton, 1994; Kitchenham et al., 1995; Basili et al., 1996; Schneidewind, 1992).

The paper by Chidamber and Kemerer which describe the C&K metrics (Chidamber and Kemerer, 1994) gives details of empirical analyses of systems at two sites, one of which used C++ and the other Smalltalk. The authors conclude that their metrics offer informative insights into whether developers are following object-oriented principles in their design and claim that using several of their metrics together helped managers and designers to make better design decisions. For example, using WMC, DIT and NOC might help to decide whether the application is getting too "top heavy." Of note is the observation made by the authors that the DIT values at both sites were small (10 or less). The explanation given is that designers want to retain comprehensibility and simplicity in favour of reuse. In a later working paper, Chidamber, Darcy and Kemerer (1996) describe how the metrics can be used to give managerial insights into productivity, effort and rework; three commercial object-oriented system are empirically investigated, and, again, none showed significant use of inheritance.

Cartwright and Shepperd (1996) described the collection of a subset of metrics from a large telecommunications system (133,000 lines of C++). Their main finding was a positive

correlation between the DIT metric and number of user-reported problems, casting doubt on the effective use of inheritance. They also reported relatively little use of inheritance and polymorphism in the system they analysed. It was suggested that caution by developers about using inheritance and lack of obvious candidates for inheritance in the problem area contributed to the low levels of inheritance. Lastly, classes with the highest change densities were found to be low down in the inheritance hierarchy.

In Basili, Briand and Melo (1996), the results of an empirical study of the C&K metrics are presented. The metrics are used as predictors of fault-prone classes. Data from eight medium-sized management systems, developed in C++, was collected and the advantages and disadvantages of the metrics are discussed. The C&K metrics are shown to be better predictors than traditional code metrics.

3. Summary of the C&K Metrics

In this section, we describe each of the C&K metrics briefly, and discuss the validity of one metric (LCOM).

3.0.1. Weighted Methods per Class (WMC)

WMC was defined as the sum of the complexities of the member functions in a class. Chidamber and Kemerer did not define complexity, but suggested that complexity might be considered to be unity, in which case WMC will be equal to the number of member functions in a class. WMC was intended to indirectly measure three attributes (Chidamber and Kemerer, 1994):

(a) the complexity of a class

(b) the effort to develop or maintain a class

(c) the application-specific nature of a class

3.0.2. Depth of the Inheritance Tree (DIT)

DIT was defined as the depth of inheritance of a class, i.e. its location in the inheritance hierarchy. The viewpoints presented (Chidamber and Kemerer, 1994) suggest that DIT represents:

(a) the complexity of the behaviour of a class

(b) the complexity of the design of a system

(c) potential reuse

3.0.3. Number of Children (NOC)

NOC was defined to be the number of immediate subclasses subordinate to a class in the class hierarchy. The NOC metric is intended to be an indirect measure of:

(a) reuse

(b) a design problem (misuse of inheritance)

(c) the effort required to test the class

3.0.4. Coupling Between Objects (CBO)

CBO was defined as the number of classes to which a class is coupled via the use of methods or attributes. Coupling between object classes is a count of the number of classes to which a class is coupled. Two classes are coupled when one uses methods or variables defined in the other. This includes coupling via inheritance (Chidamber and Kemerer, 1994).

A high CBO measure is regarded as a disadvantage, as it is intended to measure 3 attributes:

(a) lack of reuse potential

(b) effort needed during maintenance

(c) the effort required to test the class

3.0.5. Response for a Class (RFC)

RFC is defined to be equal to the size of the response set, RS, of a class:

$$\text{RFC} = |RS|$$

where the response set of a class of objects, RS, is defined as:

$$\text{RS} = \{\text{M}\} \bigcup_i \{R_i\}$$

where $\{R_i\}$ = set of all methods called by method i and
$\{\text{M}\}$ = set of all methods in the class.

However, alternative definitions and interpretations of RFC do exist (Chidamber and Kemerer, 1991; Churcher and Shepperd, 1995). Briand et al. view RFC as the number of methods invoked by a class, and suggest a number of alternative metrics (Briand et a., 1996a; Briand et al., 1997). The indirect attributes which RFC measures are given as:

(a) complexity of testing and debugging

(b) complexity of a class

(c) effort required to test a class

3.0.6. Lack of Cohesion in Methods (LCOM)

LCOM is defined as follows (Chidamber and Kemerer, 1994):

$$LCOM = \begin{cases} |P| - |Q|, & \text{if } |P| > |Q| \\ 0, & \text{otherwise} \end{cases}$$

where

P = all empty intersections of sets of variables used by member functions (i.e., all disjoint sets)

Q = all non-empty intersections of sets of variables used by member functions.

Thus, LCOM is calculated by considering the use of instance variables within a class. Clearly, if the number of non-empty intersections of sets of variables is greater than or equal to the number of empty intersections of sets of variables, then LCOM will be zero. Indirectly, LCOM is presented as measuring the following:

(a) the amount of cohesiveness present

(b) how well a system has been designed

(c) how complex a class is

3.0.7. Validity of the LCOM Metric

The representation condition of measurement (Fenton, 1991) states that for a measure to be useful, any measurement mapping must map entities into numbers, and empirical relations into numerical relations, such that those relations are preserved. In other words, our observations in the real world must be reflected in the numerical values we obtain from the mathematical world. We take satisfaction of the representation condition to be a pre-requisite for any metric to be viewed as valid.

Directly, LCOM measures the number of pairs of classes which do not use the same instance variables minus the number of pairs of classes which do. The authors state that "within the set of classes with LCOM = 0, some may be more cohesive than others" (Chidamber and Kemerer, 1994). This is certainly true, as a value of 0 can be arrived at in a number of ways. Thus LCOM violates the representation condition and so is not valid as a measure of cohesion. The validity of the LCOM metric has also been debated by other researchers (Hitz and Montazeri, 1996).

3.0.8. An Alternative Measure of Cohesion

As an alternative to the LCOM metric, we propose the following metric. This metric, Cohesiveness Of Methods (COM) is normalised by the number of attributes, giving it a range of zero to 1.

$$COM = \frac{\sum_{i=1}^{n} \frac{methods\ using\ attribute\ i}{total\ number\ of\ methods}}{number\ of\ class\ attributes}$$

Assuming j to be the number of methods in the class, the two extremes (*minimal* and *maximal* cohesion) of the COM metric occur when:

1. n attributes are used in zero methods giving a value for the metric of 0.

2. n attributes are used in j methods giving a value for the metric of 1.

Thus, the more an attribute is used by the methods in a class, the higher the value of the COM metric.

4. Empirical Evidence

In order to demonstrate the validity of software metrics, it is necessary to consider not only their theoretical basis but also to collect empirical evidence of their utility and applicability. Empirical case studies and/or formal experiments can be used to investigate the association between proposed software metrics and other indicators of software quality, such as reliability, maintainability etc. (Briand et al., 1997; Basili et al., 1986; Basili et al., 1988; Card and Glass, 1990; Card, 1991; Schneidewind, 1992; Li and Henry, 1993; Rajaraman and Lyu, 1992). However, care must be taken with this approach (Kitchenham et al., 1995; Courtney and Gustafson, 1993).

Preferably, these studies should be performed in a variety of application domains, and on large-scale projects.

4.1. The Application Domains

Three projects were chosen for this empirical analysis. The first (which will be referred to as System 1), is a traffic light simulation system designed and implemented in C++ as assessed work for six undergraduate student group projects. The average size of the simulation systems was 2.6 KLOC and the mean number of classes per system was 12. Results and analyses of collecting the C&K metrics from all six systems are provided. System 2, the "EFOOP2" system, consists of about 8.9 KLOC of C++ (12 classes), developed as part of an image analysis system to provide functions for storing and manipulating pixel images. Lastly, System 3, the "LEDA" system, is a Library of Efficient Data Algorithms, written in C++ and consisting of about 123 KLOC (197 classes), designed and developed at the Max Planck Institute in Saarbruecken, Germany. LEDA incorporates algorithms for manipulating data structures such as trees, graphs, lists and vectors and is used extensively in industry.

4.2. Data Collection

We used both automated tools and a manual data capture technique in order to collect the data and calculate the metrics. This hybrid approach was felt to be beneficial as it increased the level of trust which we could place in the data, as has also been reported by other researchers (Basili and Rombach, 1988; Sutton, 1991; Wolf and Rosenblum, 1993). The four C&K metrics collected (WMC, DIT, NOC and CBO) have already been described. The RFC metric data duplicated that for WMC, hence its omission from the tables. The LCOM metric data was not collected because manual collection proved impractical for anything but small systems. However, we were fortunate in that the LCOM data for System 2 (EFOOP2) had been collected during an earlier investigation. In addition, we collected the following metrics:

- NCSL: the number of non-comment, non-blank source lines.
- SU: Software Understanding (Boehm et al., 1995), which ranks software according to structure, application clarity and self-descriptiveness. Software understanding is rated on an ordinal scale of 1 to 5 (where 1 represents the simplest class, and 5 the most complex). A program with strong modularity, well documented with a clear match between program and application world-views would be considered easily understandable. A program with low cohesion with no match between program and application world-views and containing obscure code would be rated at the other end of the scale.

For these metrics, NCSL was measured using an automated software tool, and the subjective complexity rating was provided by the developers in System 2, and by the data collection team for Systems 1 and 3. For System 2, data relating to errors and modification requests were also collected as defined below:

- KE: the number of known errors found during system and integration testing.
- TKE: the time to fix known errors, measured in minutes.
- MR: the number of modifications requested during code reviews, testing and maintenance; this represents the number of changes that were requested excluding changes for fault-clearance.
- TMR: the time to implement modifications, measured in minutes.

(Known error data was also collected for System 1)

Finally, we also collected the following dependent variables for System 2:

- DT: the time taken to develop the programs, measured in minutes, including time to analyse, design, code and unit test.
- TT: the time taken to test the programs, measured in minutes. This represents the time-span from completion of unit testing to completion of acceptance testing.

4.3. Analysis and Results

After calculating the metrics, box-plots of the sets of data were drawn to inspect the distribution of values and identify any anomalies. Scatter diagrams were plotted to spot any obvious trends or relationships. Finally the relationships were measured using three correlation coefficients (Pearson's, Kendall's and Spearman's). The latter two coefficients were used in addition to the more common Pearson's coefficient because of the tendency of software-related data to be skewed and so less amenable to parametric statistical tests (Kitchenham et al., 1990; Fenton, 1991). We only report Spearman's coefficient here, as the remaining coefficients provide support for these results.

4.3.1. Summary Statistics: System 1 (Group Projects)

The summary statistics for System 1, incorporating the six group projects are shown in Table 1. All of the projects' summary statistics were found to be very similar. The WMC values show a wide range of class sizes in the six systems studied. The similarity of the maximum WMC values is due to the reuse of a class handling the graphical layout of the traffic light simulator.

Although 5 of the 6 group projects used inheritance, the maximum DIT in all the projects was only one. This could be explained by the nature of the application: classes tended to inherit only the graphics features needed from superclasses, many of which came from a class library. As one would expect, the six projects tended to use the same library classes. This also explains the low occurrence of non-zero values for NOC (only Project 1 has non-zero NOC values). Metrics with a maximum value of zero are omitted from the table.

The non-zero CBO values for projects are attributable to coupling due to inheritance and to the use of other classes declared as return types or parameters to methods. The nature of the application also explains the need for coupling; a typical traffic light simulator has various sub-systems which must communicate with each other. Chidamber and Kemerer suggest this feature of systems as a source of interface coupling (Chidamber and Kemerer, 1994). We compare these results with the work of others in a later section.

4.3.2. Measuring the Relationships Between Attributes: System 1 (Group Projects)

The aim of this analysis was to determine if there were any significant correlations between the C&K metrics and our complexity, size and error metrics (SU, NCSL and KE respectively). Based on guidelines relating to the structure, clarity and descriptiveness of software, the SU measure can provide a subjective means of assessing the complexity of software, a difficult attribute to define. As an alternative to an in-depth analysis of what constitutes complexity for each application domain, SU can provide an intuitive, consistent and knowledgeable expert's subjective estimate based on a set of guidelines (Boehm et al., 1995). Although use of subjective assessment has been criticised (MacDonell, 1991), it is often the only way in which cognitive complexity ratings can be obtained, and consequently continues to be used by workers in the field (Kafura and Reddy, 1987; Van Verth, 1987).

Table 1. Chidamber and Kemerer's metrics, summary statistics, System 1 (Six Group Projects).

Summary Statistics	Metric	Min	Max	Median	Mean	S.D.
Project 1	WMC	1	24	3.5	5.4	5.67
	DIT	0	1	0.5	0.5	0.51
	NOC	0	3	0	0.17	0.71
	CBO	0	4	1	1	1.19
Project 2	WMC	1	24	7.0	8.67	7.42
	DIT	0	1	0	0.33	0.5
	NOC	0	0	0	0	0
	CBO	0	2	0	0.44	0.73
Project 3	WMC	0	26	5.0	8.17	7.53
	DIT	0	1	0	0.44	0.51
	NOC	0	0	0	0	0
	CBO	0	3	0	0.5	0.86
Project 4	WMC	3	24	5.5	7.2	5.65
	DIT	0	1	1	0.79	0.43
	NOC	0	0	0	0	0
	CBO	0	2	0	0.21	0.58
Project 5	WMC	2	24	8	10	8.49
	DIT	0	1	0	0.22	0.44
	NOC	0	0	0	0	0
	CBO	0	2	0	0.44	0.73
Project 6	WMC	3	24	11	11.8	7.66
	DIT	0	0	0	0	0
	NOC	0	0	0	0	0
	CBO	0	1	0	0.2	0.45

Table 2. Spearman's rank correlation coefficients, System 1 (Group Projects).

	Project	*WMC*	*DIT*	*NOC*	*CBO*
SU	1	0.54*	−0.40†	−0.23	−0.09
	2	0.24	0.29	nc	0.33
	3	0.36	0.59*	nc	0.21
	4	0.53†	−0.79*	nc	0.57†
	5	0.82*	0.06	nc	−0.15
	6	0.67*	nc	nc	0
NCSL	1	0.81*	−0.36	−0.07	−0.35
	2	0.43	−0.46	nc	−0.49
	3	0.79*	0.11	nc	−0.28
	4	0.65*	−0.71*	nc	0.24
	5	0.62†	0	nc	−0.34
	6	0.70*	nc	nc	0

*significant at the 1% level
†significant at the 5% level
nc not computable

Table 2 gives the figures for one-tailed significance tests for Spearman's correlation coefficient, with significance at the 1% and 5% levels highlighted. These results confirm the relationships suggested by corresponding scatter diagrams, and also confirm results from Kendall's and Pearson's correlation tests.

There is a statistically significant positive relationship between WMC and SU (Table 2) for four of the systems, which provides evidence for Chidamber and Kemerer's theory relating WMC with complexity.

For DIT, two systems (Projects 1 and 4) show a significant negative relationship with SU; Project 3 shows a significant positive relationship. However, the low DIT values impede data analysis here and explain the extremes found in the correlation values. No statistically significant relationships were found between SU and NOC; CBO shows only one positive significant relationship.

Thus, these results show a relationship between the number of methods in a class and ease of understanding, but do not provide much evidence of any relationships between inheritance hierarchies and ease of understanding, nor between coupling and understandability.

Table 2 also indicates a strong statistically significant positive relationship between NCSL and WMC, as would be expected.

The correlation results for DIT versus NCSL show a statistically significant negative relationship for Project 4 and negative relationships for Projects 1 and 2, supporting the thesis that reuse due to inheritance may result in smaller classes. No significant relationships were found for NOC versus NCSL, nor for CBO.

The results in Table 3 indicate some positive relationships between KEs and WMC. The DIT values show no significant relationships, nor do those for NOC or CBO. Table 3 also shows remarkably little correlation between error density and any of the C&K metrics. The lack of correlation between WMC and error density indicates that larger classes are

Table 3. Spearman's rank correlation coefficients, System 1 (Group Projects).

	Project	*WMC*	*DIT*	*NOC*	*CBO*
KE	*1*	0.39	0.15	−0.15	0.22
	2	0.44	−0.49	nc	−0.48
	3	0.11	0.02	nc	0.06
	4	0.62*	−0.06	nc	−0.26
	5	−0.09	0.11	nc	0.39
	6	0.78*	nc	nc	0.40
KE/KNCSL	*1*	0.32	0.12	−0.15	0.20
	2	0.41	−0.49	nc	−0.48
	3	0.11	0.02	nc	0.06
	4	0.56†	0.03	nc	−0.25
	5	−0.28	0.16	nc	0.50
	6	−0.20	nc	nc	0.35

*significant at the 1% level
†significant at the 5% level
nc not computable

Table 4. Summary statistics, System 2 (EFOOP2).

	Min	Max	Median	Mean	S.D.
WMC	4	21	10.5	11.3	5.3
DIT	0	0	0	0	0
NOC	0	0	0	0	0
CBO	0	1	0	0.3	0.5
LCOM	6	215	15.5	50.8	62

not necessarily more error-prone than smaller ones in this case. The DIT values show no significant relationships, nor do those for NOC and CBO.

4.3.3. Summary Statistics: System 2 (EFOOP2)

The summary statistics for System 2 are shown in Table 4. The WMC values show a similar range to that of the six group projects. The absence of any inheritance is reflected by the zero DIT and NOC values, again supporting the findings of Chidamber and Kemerer (1994) and Cartwright and Shepperd (1996) in which low values of DIT and NOC were found. The low values for CBO may be explained by the nature of the application: each class is independent and fulfils a specific purpose. A class has no need to be coupled to any other classes. Table 4 shows a high maximum value for LCOM of 215, suggesting that there is a lot of sharing of data attributes between the methods of that class.

Table 5. Spearman's Rank correlation coefficients, System 2 (EFOOP2).

	WMC	*DIT*	*NOC*	*CBO*	*LCOM*
SU	0.45	nc	nc	0.32	0.32
NCSL	0.86*	nc	nc	0.18	0.86*
KE	0.46	nc	nc	−0.18	0.53†
KE/KNCSL	0.30	nc	nc	−0.21	0.19
TKE	0.67*	nc	nc	−0.21	0.75*
MR	0.55†	nc	nc	−0.18	0.64†
TMR	0.67*	nc	nc	−0.26	0.67*
DT	0.55†	nc	nc	−0.16	0.55†
TT	0.42	nc	nc	−0.37	0.44

*significant at the 1% level
†significant at the 5% level
nc not computable

4.3.4. Measuring the Relationships Between Attributes: System 2 (EFOOP2)

Table 5 shows the results of our investigation into the correlations between Chidamber and Kemerer's metrics and various dependent variables for System 2.

WMC and LCOM were found to have statistically significant correlations with many of the dependent variables. However, the CBO results are remarkable in that they show no significant correlation whatsoever.

In terms of the DT and TT metrics, the main point to note from Table 5 is the strong correlation of the WMC metric with DT. These results support Chidamber and Kemerer's thesis that WMC can be used to predict development time. Clearly, more data is needed to support or refute this result. Relationships between the C&K metrics and TT, however, were not so pronounced. Again, there is a complete lack of correlation between CBO and these dependent variables.

4.3.5. Summary Statistics: System 3 (LEDA)

The summary statistics for System 3 are shown in Table 6. These show an extremely wide range in the WMC values. This reflects the variety of data algorithms supported by LEDA. The small DIT values hide the true LEDA inheritance structure; it comprises a large number of small trees, each of which has shallow depth and correspondingly small numbers of children (reflected in the NOC values). The CBO values largely reflect coupling due to inheritance.

Table 6. Summary statistics for System 3 (LEDA).

	Min	*Max*	*Median*	Mean	S.D.
WMC	0	337	16	24.12	38.25
DIT	0	4	0	0.63	0.79
NOC	0	4	0	0.2	0.62
CBO	0	6	1	1.2	1.2

Table 7. Spearman's rank correlations coefficients, System 3 (LEDA).

	WMC	*DIT*	*NOC*	*CBO*
SU	0.08	0.15†	0.04	0.09
NCSL	0.78*	0.10	0.09	0.23*

*significant at the 1% level
†significant at the 5% level

4.3.6. Measuring the Relationships Between Attributes: System 3 (LEDA)

The SU correlations for System 3 are shown in Table 7. The significant correlation between DIT and SU can be explained by the *forest-like* design, which impaired understandability. The optimal shape of inheritance hierarchies requires further research.

The NCSL correlation results for System 3 are also also shown in Table 7. As expected, there is a statistically significant positive relationship between NCSL and WMC. The correlation with CBO may be explained by the nature of the LEDA system: methods in LEDA often use objects from other classes as parameters or return types, because of frequent reuse of data types, hence increasing the level of inter-class coupling.

4.4. Summary of Results

Table 8 presents a summary of results, for all three systems investigated. The table highlights the positive relationships found between WMC and NCSL, and the lack of statistically significant relationships between CBO and the dependent variables. The lack of inheritance prevented us from analysing relationships between the inheritance metrics (DIT and NOC) and the dependent variables in the majority of cases.

4.5. Comparison of Results

From the summary statistics for the three systems we see a range of WMC values, the highest median value being 16. Chidamber and Kemerer also reported a range of values for WMC, with the highest median being 10 (Chidamber and Kemerer, 1994). Basili et al.

Table 8. Results summary for the three systems investigated.

		WMC	*DIT*	*NOC*	*CBO*	*LCOM*
System 1 Group Projects	*SU*	4/6 +ive	+ive and −ive	nc	1/6 −ive	nc
	NCSL	5/6 +ive	nr	nc	nr	nc
	KE	2/6 +ive	nr	nc	nr	nc
	KE/KNCSL	1/6 +ive	nr	nc	nr	nc
System 2 EFOOP2	*SU*	nr	nc	nc	nr	nr
	NCSL	+ive	nc	nc	nr	+ive
	KE	nr	nc	nc	nr	+ive
	KE/KNCSL	nr	nc	nc	nr	nr
	TKE	+ive	nc	nc	nr	+ive
	MR	+ive	nc	nc	nr	+ive
	TMR	+ive	nc	nc	nr	+ive
	DT	+ive	nc	nc	nr	+ive
	TT	nr	nc	nc	nr	nr
System 3 LEDA	*SU*	nr	+ive	nr	nr	nc
	NCSL	+ive	nr	nr	+ive	nc

+ive: positive relationship
−ive: negative relationship
nc: not computable
nr: no relationship
n/6: relationship significant for n of 6 projects

describe a collection of the C&K metrics from eight medium sized C++ systems (Basili et al., 1996). Of 180 classes analysed, the median WMC was found to be 9.5 (maximum 99). For each system studied in this paper, there appear to be classes with particularly high WMC values, implying high functionality. These classes can be thought of as *key* or controlling classes offering services to other classes. Use of these classes is particularly evident in the case of System 1 (Group Projects) and System 3 (LEDA). It could be argued that this contravenes good OO design principles.

Chidamber and Kemerer also report low DIT and NOC values, with a median DIT of 1 and median NOC of 0 (Chidamber and Kemerer, 1994). Similar findings with respect to DIT and NOC have also been reported (Chidamber et al., 1996) and (Cartwright and Shepperd, 1996). In the latter, the systems studied generally exhibited flat inheritance hierarchies with classes having few children. Similarly, all three systems studied in this paper have low levels of inheritance (EFOOP2 contains no inheritance). Again, this reflects the nature of the respective applications. System 1 tended to inherit from key library classes only. System 3 (LEDA) has a tree structure composed of a large number of small sub-trees; this feature is not captured adequately by either the DIT or NOC metric.

In the three systems studied, the CBO values reflect coupling coming from two sources: inheritance and the use of objects from other classes as return types or parameters to methods. The low levels of inheritance in the three systems suggests that such parameterisation accounts for the greater part of the coupling. Chidamber and Kemerer report similar findings, emphasising that excessive coupling is detrimental to modular design.

For the correlations of three systems studied, there appear to be strong positive relation-

ships between WMC and SU, and also between WMC and NCSL. This is to be expected; classes with more methods are often harder to understand and usually contain more lines of code. The DIT comparisons are also interesting, with no clear indication as to whether the use of inheritance aids or hinders understanding of a class nor whether it leads to smaller classes. It may be that there is an *optimum* level of inheritance (Daly et al., 1996), but the lack of inheritance in the systems studied here has prevented us from making much progress in this area. Our current research seeks to remedy this situation.

5. Experimental Validity

External validity is the degree to which results from an experiment can be generalised to the population, i.e., other groups. For example, the use of students as subjects in an experiment, or the generality of the application domain chosen for study are two possible threats to external validity.

The major threat to the external validity of the experiments described in this paper is the size of the systems studied. Many industrial-sized systems are significantly larger than those studied here. However, this research should be seen as exploratory, and while this threat limits generalisation of the results, it does not prevent us from using the research as a basis for future studies, and we hope others will as well.

6. Conclusions

In order to validate software metrics it is necessary to show that the metrics have a sound empirical basis by providing empirical evidence of their application through formal experiments or case studies. This paper presents the results of three such studies of the application of the Chidamber and Kemerer (C&K) metrics. Summary statistics resulting from the collection of the metrics are presented, showing means and standard deviations. These results can be used by managers and developers who are in need of comparative statistics. We feel that the results from our empirical analysis are interesting, as they reinforce the findings of other researchers, such as the under-utilisation of inheritance shown by the depth of the inheritance tree and number of children metrics.

In addition, we also report the results of statistical analyses which investigated the correlations between the C&K metrics and certain dependent variables, including the number of known errors, the number of non-comment source lines (NCSL) and a measure of software understandability. The significant correlation between NCSL and the number of methods in a class (WMC) suggests that the latter (which may be available early in the design process) could be used to estimate the former, and so be useful in cost/effort estimation.

These results show that the C&K metrics can be used by developers and maintainers to gain insights into their systems' architectures. With large, complicated systems this information could prove to be very valuable to maintainers of legacy code as additional design documentation.

Acknowledgments

The authors wish to acknowledge the support of the UK 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. The authors would also like to thank the anonymous referees for their insightful comments.

References

Basili, V. R., Briand, L. C., and Melo, W. L. 1996. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering* 22(10): 751–761.

Basili, V. R., and Rombach, H. D. 1988. The TAME project: Towards improvement-oriented software environments. *IEEE Transactions on Software Engineering* 14(6): 758–773.

Basili, V. R., Selby, R. W., and Hutchens, D. H. 1986. Experimentation in software engineering. *IEEE Software* 12(7): 733–743.

Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., and Selby, R. 1995. Cost models for future software life cycle processes: Cocomo 2.0. *Annals of Software Engineering* 1(1): 57–94.

Briand, L., Daly, J. W., and Wust, J. 1996a. A unified framework for coupling measurement in object-oriented systems. ISERN-96-14, Fraunhofer Institute for Experimental Software Engineering.

Briand, L., Devanbu, P., and Melo, W. 1997. An investigation into coupling measures for C++. In *Proc. 19th ICSE*, Boston, USA.

Briand, L., El Emam, K., and Morasca, S. 1996b. Viewpoint: On the application of measurement theory in software engineering. *Empirical Software Engineering* 1(1): 61–88.

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

Brito e Abreu, F., and Carapuca, R. 1994. Candidate metrics for object-oriented software within a taxonomy framework. *Journal of Systems and Software* 26: 87–96.

Brito e Abreu, F., Goulao, M., and Esteves, R. 1995. Toward the design quality evaluation of 00 software systems. In *5th Int. Conf. of Software Quality*.

Brooks, I. 1993. Object-oriented metrics collection and evaluation with a software process. In *Workshop on Processes and Metrics for Object Oriented Software Development, OOPSLA'93*, Washington.

Card, D. 1991. What makes a software measure successful? *Amer. Programmer* 49(9): 2–8.

Card, D., and Glass, R. 1990. *Measuring Software Design Quality*. Prentice-Hall, Inc.

Cartwright, M., and Shepperd, M. 1996. An empirical analysis of object-oriented software in industry. In *Bournemouth Metrics Workshop, Bournemouth, UK*.

Chidamber, S. R., Darcy, P. D., and Kemerer, C. F. 1996. Managerial use of object-oriented software metrics. Working Paper Series No. 750.

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

Chidamber, S. R., and Kemerer, C. F. 1994. A metric suite for object oriented design. *IEEE Transactions on Software Engineering* 20(6): 467–493.

Churcher, N. I., and Shepperd, M. J. 1995. Comments on 'a metric suite for object-oriented design'. *IEEE Transactions on Software Engineering* 21(3): 263–265.

Conte, S., Shen, V., and Dunsmore, H. 1986. *Software Engineering Metrics and Models*. Benjamin Cummins Publishing, Inc.

Courtney, R. E., and Gustafson, D. A. 1993. Shotgun correlations in software measures. *Software Engineering Journal* 8(1): 5–13.

Daly, J. et al. 1996. Evaluating inheritance depth on the maintainability of object-oriented software. *Empirical Software Engineering* 1(2): 109–132.

Ejiogu, L. O. 1993. Five principles for the formal validation of software metrics. *ACM SIGPLAN Notices* 28(8): 67–76.

Fenton, N. E. 1991. *Software Metrics, A Rigorous Approach*. Chapman & Hall.

Fenton, N. E. 1994. Software measurement: a necessary scientific basis. *IEEE Transactions on Software Engineering*, 20(3): 199–206.

Graham, I. M. 1996. Making progress in metrics. *Object Magazine* 68–73.

Henderson-Sellers, B. 1996. *Object-Oriented Metrics: Measures of Complexity*. Prentice-Hall Object-Oriented Series.

Hitz, M., and Montazeri, B. 1996. C&K metrics suite: a measurement theory perspective. *IEEE Transactions on Software Engineering* 22(4): 267–271.

Kafura, D., and Reddy, G. R. 1987. The use of software complexity metrics in software maintenance. *IEEE Transactions on Software Engineering* 7(5): 335–343.

Kearney, J. K., Sedlmeyer, R. L., Thompson, W. B., Gray, M. A., and Adler, M. A. 1986. Software complexity measurement. *CACM* 29(11): 1044–1050.

Kitchenham, B. A., Fenton, N., and Pfleeger, S. L. 1995. A framework for validation. *IEEE Transactions on Software Engineering* 21(2).

Kitchenham, B. A., Pickard, L. M., and Linkman, S. J. 1990. An evaluation of some design metrics. *Software Engineering Journal*, 5(1): 50–58.

Lake, A., and Cook, C. 1992. A software complexity metric for C++. Technical Report 92-60-03, Oregon State University.

Li, W., and Henry, S. May 1993. Maintenance metrics for the object-oriented paradigm. In *Proceedings of the First International Software Metrics Symposium*, Baltimore, Maryland, 52–60.

Lorenz, M., and Kidd, J. 1989. *Software Metrics: Measures and Methods*. Walter de Gruyter.

Lorenz, M., and Kidd, J. 1994. *Object-Oriented Software Metrics*. Prentice-Hall Object-Oriented Series.

MacDonell, S. G. 1991. Rigor in software complexity measurement experimentation. *Journal of Systems and Software* 16: 141–149.

Melton, A. C., Gustafson, D. A., Bieman, J. M., and Baker, A. L. 1990. A mathematical perspective for software measures research. *Software Engineering Journal* 5: 246–254.

Moreau, D. R., and Dominick, W. D. 1989. Object-oriented graphical information systems: Research plan and evaluation metrics. *The Journal of Systems and Software* 10: 23–28.

Pfleeger, S. L., and Palmer, J. D. 1990. Software estimation for object-oriented systems. In *1990 Int. Function Point Users Group Fall Conf.* 181–196.

Rajaraman, C. and Lyu, M. R. 1992. Some coupling measures for c++ programs. In *TOOLS USA 92*, vol. 6, 225–234.

Schneidewind, N. F. 1992. Methodology for validating software metrics. *IEEE Transactions on Software Engineering* 18(5): 410–422.

Sheetz, S. D., Tegarden, D. P., and Monarchi, D. E. 1991. Measuring object-oriented system complexity. In *Proc. 1st Workshop in IT and Systems*.

Sutton, S. M. 1991. Accommodating manual activites in automated process programs. In *Proc. of the 7th Int. Software Process Workshop*. IEEE CS Press.

Tegarden, D. P., and Sheetz, S. D. 1992. Effectiveness of traditional software metrics for object-oriented systems. In *25th Annual Conference on Systems Science*, Maui, HI, USA, 359–368.

Tsai, W. T., A, L. M., Rodriguez, V., and Volovik, D. 1986. An approach measuring data structure complexity. In *COMPSAC 86* 240–246.

Van Verth, P. B. 1987. A program complexity model that includes procedures. In *Proceedings of COMPSAC'87*, Tokyo, 252–258.

Weyuker, E. J. 1988. Evaluating software complexity measures. *IEEE Transactions on Software Engineering* 14(9): 1357–65.

Whitmire, S. 1992. Measuring complexity in object-oriented software. In *3rd Int. Conference on App. Software Measurement*, CA, USA.

Wilde, N., and Huitt, R. 1992. Maintenance support for object-oriented programs. *IEEE Transactions on Software Engineering* 18(12): 1038–1044.

Wolf, A. L., and Rosenblum, D. S. 1993. A study in software process data capture and analysis. In *Proc. of the 2nd Int. Conf. on the Software Process*, 115–124. IEEE CS Press.

Rachel Harrison is a member of the Engineering Faculty at the University of Southampton. She obtained an M.A. in Mathematics from Oxford University, an M.Sc. in Computer Science from London University, and a Ph.D. in Computer Science from the University of Southampton. Her current research interests centre around empirical software engineering, particularly measurement and modelling of the object-oriented paradigm, the evaluation of formal methods and of hypermedia systems.

Dr. Harrison is a member of the IEEE Computer Society, the ACM, and the BCS.

Steve Counsell obtained the B.Sc. degree in Computer Studies from the University of Brighton in 1987 and an M.Sc. in Systems Analysis from The City University, London in 1988. He has just completed a Ph.D. in Computer Science from Birkbeck College, University of London and is currently a post-doctoral 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 centre around the the measurement of the object-oriented paradigm, empirical software engineering, software reliability modelling and business process modelling.

Reuben Nithi received the B.Sc. degree in 1992 in Computer Science from 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 area of empirical software engineering, software quality modeling, and object oriented design metrics.