Coupling Metrics for Object-Oriented Design

R. Harrison, S. Counsell, R. Nithi
Department of Electronics and Computer Science,
University of Southampton, Southampton, SO17 1BJ, U.K.
Tel. +44 (0) 1703 593249, Fax. +44 (0) 1703 593045.
rh@ecs.soton.ac.uk

Abstract

In this paper, we describe and evaluate some recently innovated coupling metrics for object-oriented (OO) design. The Coupling Between Objects (CBO) metric of Chidamber and Kemerer are evaluated empirically using five OO systems, and compared with an alternative OO design metric called NAS, which measures the Number of Associations between a class and its peers. The NAS metric is directly collectible from design documents such as the Object Model of OMT. Results from all systems studied indicate a strong relationship between CBO and NAS, suggesting that they are not orthogonal.

We hypothesised that coupling would be related to understandability, the number of errors and error density. No relationships were found for any of the systems between class understandability and coupling. Only limited evidence was found to support our hypothesis linking increased coupling to increased error density.

The work described in this paper is part of the 'Metrics for OO Programming Systems' (MOOPS) project, which aims are to evaluate existing OO metrics, and to innovate and evaluate new OO analysis and design metrics, aimed specifically at the early stages of development.

1 Introduction

Various object-oriented metrics suites have been suggested as a means of determining whether systems under investigation hold desired properties of object-oriented software, or whether that software is of sufficient quality [11, 12, 17, 9, 18, 14]. Many of these sets of metrics have been proposed in an *ad hoc* fashion, with no regard to their appropriateness or validity, and with scant emphasis on an empirical evaluation. In addition, they are often only collectible at late stages in the development process,

In this paper, we empirically evaluate the CBO metric of Chidamber and Kemerer [12] using data collected from five OO systems of various application domains and sizes. Data was also collected from the same systems for an alternative OO design coupling metric measuring the Number of Associations (NAS) between a class and its peers. The NAS metric was developed using the top-down Goal-Question Metric (GQM) approach [3], and is directly collectible from OMT Object Model diagrams [20]. A correlation analysis of these metrics shows a strong relationship between CBO and NAS.

In the next section, a description of related work is given. In Section 3, we provide details of the empirical evaluation, giving details of the metrics collected and the application domains. Data analysis, giving the correlation analyses, and error analyses for two of the systems are given in Section 4. Finally, some conclusions and further work are described in Section 5.

2 Related Work

A number of recent studies have been carried out into coupling and associated metrics. A general consensus in the software engineering community is that too much coupling is harmful in terms of system structure and increases system complexity [6, 15, 7, 13].

Chidamber and Kemerer [10] gave an initial definition of coupling as any evidence of a method of one object using methods or instance variables of another object. The initial definition of the Chidamber and Kemerer Coupling Between Objects (CBO) metric excluded coupling due to inheritance, but this was changed in later work [12]. The Chidamber and Kemerer metrics were empirically validated in [2], in which several of the metrics, (including CBO) were found to be useful for predicting fault-prone classes.

Briand et al. [6] describe coupling as the degree of interdependence among the components of a software system. They propose a suite of coupling measures,

taking into account the different features offered by the C++ language. These measures were empirically evaluated, and shown to be useful as OO quality indicators. The use of *friends* as a form of coupling is also considered in this paper. Briand et al. [7] provide a review of current coupling metrics and give a unified framework for measurement of coupling in OO systems.

Hitz and Montazeri [15] propose a framework for dealing with coupling and cohesion in OO systems making the distinction between object level and class level coupling, the former relating to the state of an object at run-time, which may be different to the static class level coupling of the latter.

Li and Henry [17] describe class level coupling in terms of data abstraction coupling (DAC), measured in terms of instance variables having an abstract data type. A distinction is made between coupling with other classes, and coupling as a result of messages sent to *self* (i.e., an instance of a class type).

Henderson-Sellers et al. [13] evaluate the Chidamber and Kemerer metrics from a mathematical viewpoint. Inconsistencies in the CBO metric definition are rectified as a result. A further theoretical study of the same set of metrics is given in [8], in which complexity measures such as coupling and cohesion are evaluated and defined.

3 Empirical Evaluation

Empirical evaluation can be used to investigate the association between proposed software metrics and other indicators of software quality such as maintainability or understandability; thorough qualitative or quantitative analyses can be used to support these investigations [5, 2, 21, 1]. A commonly-held view is that there is a link between coupling and complexity in software, and that too much coupling is indicative of a poorly thought out design. This paper addresses this issue through the investigation of a number of hypotheses.

3.1 Hypotheses

Three hypotheses related to coupling are investigated in this paper:

1. **H1**: As inter-class coupling increases, the understandability of a class decreases.
2. **H2**: As inter-class coupling increases, the total number of errors found in a class increases.
3. **H3**: As inter-class coupling increases, the error density of a class increases.

In order to investigate these hypotheses, we studied the relationships between the independent variables (CBO and NAS), and three dependent variables: a measure of software understandability (SU), the number of known errors (KE) and errors per thousand non-comment source lines (KE/KNCSL) which we call *error density*, as described in the following section. In this paper, we view an error as an external departure of a system from its required behaviour. A fault is a defect in the code which may or may not lead to an error [16].

3.2 Data Collection

The Coupling Between Objects (CBO) metric 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 [12]. A high CBO measure is regarded as a disadvantage, as it is suggested that it indicates the extent of:

a) lack of reuse potential
b) effort needed during maintenance
c) effort required to test the class

The Number of Associations (NAS) metric is defined as the number of associations of each class, counted by the number of association lines emanating from a class on an OMT diagram. This measure therefore also includes inheritance; note that the NAS metric can be collected directly from design documents.

We note that the CBO and NAS metrics are identical, except in the following case: if a class uses a method of another class more than once, then the CBO metric will count each usage as a separate occurrence of coupling. NAS, on the other hand counts repeated invocations as a single occurrence of coupling. Hence, we expect values of CBO for a class to be greater than the NAS for the same class (a quantitative assessment of the differences between CBO and NAS metrics is left for future research).

In addition, for each class in a system, the following dependent variables were also collected:

- NCSL: the number of non-comment, non-blank source lines.
- SU: Software Understanding [4], 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 and easiest to understand class, and 5 the most complex and difficult to understand). Thus, H1 requires us to ask whether there is a positive relationship between SU and CBO, and also between SU and NAS. The SU measure is based on ratings of individual classes, and was provided by the data collectors (two of the authors and a research assistant).

The data was collected as follows: NCSL was measured using an automated software tool, and for all systems, the SU was provided by the the data collectors. Although subjective in nature, Boehm's SU metric represents an easily-collectible, consistent and useful reflection of class complexity.

For Systems 3 (SEG1) and 4 (SEG2), data relating to errors were also collected, as defined below:

- KE: the number of known errors (per class) found during testing.
- KE/KNCSL: The density of errors per KNCSL (per class).

In the next section, we describe the application domains of each of the five systems studied.

3.3 Application Domains

Our empirical analysis consisted of five systems. These were:

1. System One, the GNU C++ Class Library, consisting of 53.5 KLOC (96 classes).
2. System Two, a Library of Efficient Data Algorithms (LEDA), written in C++ and consisting of about 123 KLOC (197 classes), designed and developed at the Max Planck Institute in Saarbruecken, Germany.
3. System Three, SEG1, consisting of ten medium-sized traffic simulation systems written in C++. The average size of a system was 2.5 KNCSL, and a total of 113 classes were analysed for the ten systems. Each of the ten systems was developed in a Windows '95 programming environment.
4. System Four, SEG2, consisting of eleven medium-sized traffic simulation systems written in C++. The average size of a system was 8.5 KNCSL, and a total of 61 classes were analysed. Each of the eleven systems was developed in an MSDOS (Version 6.0) programming environment.
5. System Five, EFOOP2, 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. The EFOOP2 system features no inheritance.

3.4 The Systems' Architectures

Since both CBO and NAS are related to the architecture of a system, in terms of inter-class relationships (including coupling due to inheritance) it is appropriate to describe the features of each of the five systems studied. This will also aid understanding of the results described later in this paper.

All of the five systems except EFOOP2 contained inheritance. The inheritance structures in each of the four remaining systems differed significantly. The Gnu (System 1) inheritance hierarchies tended to be fairly shallow (with a mean depth of inheritance tree of 0.93). The LEDA (System 2) inheritance structure, on the other hand, contained a large number of small inheritance trees (the mean depth of inheritance trees was 0.63). Consequently, we would expect more coupling due to inheritance to be found in Gnu (System 1). In both LEDA and Gnu, extensive use was made of friend functions thus avoiding the use of inheritance via *invisible* coupling (neither CBO nor NAS take coupling due to friend functions into account).

SEG 1 (System 3) contained relatively little inheritance with a large proportion of classes not engaging in any inheritance, whilst SEG 2 (System 4) did contain inheritance, but the inheritance hierarchies tended to be shallow. Neither SEG1 or SEG 2 used friend functions to the extent of LEDA and Gnu.

Interestingly, in all systems, particularly SEG1, there was a high proportion of singleton classes (i.e., classes without any inter-class coupling). Classes in SEG1 tended to perform a single function such as a display of a screen layout, or the change of state of a light bulb; this required no inter-class relationships. Classes with similar functionality were found in the LEDA and Gnu systems, and more particularly in the EFOOP2 system.

However, all the systems investigated showed some evidence of coupling due to:

1. a class X using class Y as a parameter of one of its methods
2. a class Y being a return data type for a method of class X.

	Metric	No. Classes	Min	Max	Median	Mean	S.D.
System One (Gnu)	CBO	96	0	4	1	1.03	0.88
	NAS	96	0	2	1	0.78	0.56
System Two (LEDA)	CBO	197	0	6	1	1.16	0.62
	NAS	197	0	5	0	0.62	0.96
System Three (SEG1)	CBO	113	0	5	0	0.53	0.66
	NAS	113	0	4	0	0.25	0.66
System Four (SEG2)	CBO	61	0	9	1	1.74	2.17
	NAS	61	0	4	1	1.10	1.15
System Five (EFOOP2)	CBO	12	0	1	0	0.33	0.33
	NAS	12	0	1	0	0.33	0.33

Table 1: Summary statistics for the five systems investigated

4 Data Analysis

Table 1 shows the summary data for each of the five systems investigated.

The most noticeable feature of Table 1 is the similarity of the CBO and NAS median values which are identical for four out of the five systems. For the EFOOP2 System, in which there is no inheritance, the values of all the summary data are identical.

Table 2 shows correlation values for CBO vs NAS for each of the five systems investigated. This shows significant positive relationships at the 1% level for four of the five systems. Only System Two (LEDA) deviates from this trend, but the correlation is still significant at the 5% level. This lower correlation value suggests that LEDA had classes which were multiply coupled with other classes.

Table 3 shows a correlation analysis of SU against CBO and NAS. No significant relationships are shown, suggesting that understandability is unrelated to the level of coupling for these systems. This result is interesting, considering that three different assessors took part in the allocation of SU values. We therefore reject hypothesis H1. It is not necessarily the case that as coupling increases, the understandability of a class decreases.

A number of explanations for this lack of correlation are possible. Firstly, when considering the understandability of a class, the assessor may tend to consider the class in isolation. Coupling with other classes is considered outside the boundary of what needs to be understood. This would also include coupling due to inheritance. An alternative measure of understandability, incorporating the relationships between classes would provide interesting results, and is left for future research.

An assessor would consider the number and type of data attributes in a class, the complexity of the method interfaces, the structure of the code in the method bodies, and the cohesiveness of the class rather than inter-class coupling which describes relationships between classes, not the classes themselves.

As a further investigation into the relationship between NAS and CBO, we now describe an analysis of the faults found during testing of Systems 3 and 4.

4.1 Fault Analysis

Table 4 shows correlations for CBO and NAS against KE for System 3 and System 4. A statistically significant negative relationship was found for NAS against KE (System 4). This counter-intuitive result suggests that independent classes had the most errors. Investigating further, it was found that, for SEG1, 57% of faults could be attributed to just two classes. These two classes controlled the layout and configuration of the traffic simulation systems and were not highly-coupled classes. Thus, no evidence was found to support hypothesis H2, which we therefore reject. It is not necessarily the case that as inter-class coupling increases, the total number of errors found in a class increases.

Table 5 shows correlations for CBO and NAS against KE/KNCSL for System 3 and System 4. Here, a statistically significant positive relationship was found for NAS against KE/KNCSL (System 4). This provides some support for hypothesis H3 (as inter-class coupling increases, the error density of a class increases).

We also investigated whether a significant relationship between coupling and class size existed for the systems studied. If so, then we could not view coupling as independent of error density, since error density has a size component. H3 would need to be re-formulated as

System	*Spearman's*
System 1 (Gnu)	0.45*
System 2 (LEDA)	0.13†
System 3 (SEG1)	0.77*
System 4 (SEG2)	0.75*
System 5 (EFOOP2)	1.00*
*significant at the 1% level †significant at the 5% level	

Table 2: Spearman's correlation coefficients for CBO vs. NAS

System	*CBO*	*NAS*
System 1 (Gnu)	0.18	0.13
System 2 (LEDA)	0.09	-0.06
System 3 (SEG1)	0.08	-0.05
System 4 (SEG2)	0.19	-0.25
System 5 (EFOOP2)	0.32	0.32

Table 3: Spearmans rank correlation coefficient for all five Systems vs. SU

a result. However, in at least four of the systems we looked at, conflicting results were found (i.e., both positive and negative significant relationships) for the relationship between coupling and size (given by NCSL). This suggests that the relationship between coupling and error density is a design question.

The lack of statistically significant correlations may be due to a number of reasons. The most plausible explanation is that in all of the systems studied, a large proportion of classes are singleton classes (i.e., are not related to any other classes). This can be attributed to the nature of the systems investigated. Classes tended to be stand-alone because they fulfilled a single purpose, without having to be coupled to other classes. For example, the operations on a single data structure common to classes in the LEDA and Gnu systems, or handling the graphical layout of the traffic light system in the SEG1 and SEG2 systems.

Table 6 summarises the number of classes with CBO values of zero, one and two or more for each of the systems investigated, together with the percentage of the total number of classes that each of these numbers represent.

Table 6 shows SEG 1 (System 3) to have the highest proportion of zero-coupled classes. For System 3, correlations for KE against NAS and against CBO were re-run with all singleton classes removed; still no significant correlations were found. Re-running the correlations for KE/KNCSL against NAS and CBO similarly showed no significant correlations. Table 6 also shows each of the five systems investigated to have a high proportion of classes coupled to only one other class. A high proportion of one-coupled classes in Systems 1 to 4 can be explained by a tendency for classes in those systems to inherit from a single root class, and to contain no other forms of coupling. For System 3 (SEG1) classes tended to inherit from one of two root classes; one handling the system screen layout, the other handling the configuration of traffic lights in the system. Thus, ignoring singleton classes, we see that the majority of classes were found to be coupled to only one other class, and consequently correlation calculations were unlikely to discover any significant relationships.

In the next section, we consider the threats to the validity of our study.

4.2 Threats to validity

We consider here threats to the external and internal validity of this study [19]. *External* validity is the degree to which results from the study can be generalised to the population, i.e., other groups. The systems investigated in this paper represent a mix of projects of different sizes and application domains, reducing this threat to external validity. Note that Systems 3 and 4, for which we performed error analyses, were developed by undergraduate students, not professional programmers. However, the conditions in which they worked on these group projects are intended to mimic those in the real-world as far as possible; the students (who have at least eighteen months programming experience) are expected to re-engineer medium-sized legacy systems,

System	*CBO*	*NAS*
System 3 (SEG1)	-0.10	0.02
System 4 (SEG2)	-0.19	-0.32†
†significant at the 5% level		

Table 4: Spearman's correlation coefficients for System 3 and System 4 vs. KE

System	*CBO*	*NAS*
System 3 (SEG1)	-0.07	0.05
System 4 (SEG2)	0.12	0.31†
†significant at the 5% level		

Table 5: Spearman's correlation coefficients for System 3 and System 4 vs. KE/KNCSL

according to changing requirements.

Internal validity is the degree to which we can conclude that the dependent variable is accounted for by the independent variable. The SU values were provided by three different assessors, and the results of the SU correlations from the five different systems concurred (Table 3). One possible confounding effect was the use of friend functions by several of the systems. However, for Systems 3 and 4 (which were analysed in detail), little evidence of the use of friends was found. Another possible confounding effect was the high proportion of zero-coupled classes. However, for System 3, no change was noted in the significance levels after re-running the KE and KE/KNCSL correlations without these classes. Finally, the nature of the application domain seems to have a strong bearing on the architecture of the system. A system with a graphical user interface may have different structural properties to that of a system containing library classes; this could be considered a threat to external validity.

5 Conclusions and Future Research

In this paper, we have analysed two coupling metrics: the CBO metric proposed by Chidamber and Kemerer, and the NAS metric, developed through use of the Goal Question Metric approach as an object-oriented analysis metric. A strong relationship was shown to exist between the CBO and NAS metrics, implying that only one of these is needed to assess the level of coupling in a system at design time. The NAS metric is available early on in systems design, is simple to collect from design documents, is easy to interpret and could be used by managers to obtain early coupling estimates.

We hypothesised that coupling would be related to understandability, the number of errors and error density. No relationships were found for any of the systems between class understandability and coupling. This can be attributed partly to the way that the subjective understandability metric was evaluated by the developer. Only limited evidence was found to support our hypothesis linking increased coupling to increased error density. The lack of discovered statistically significant relationships may be attributable to the high proportion of uncoupled or singly-coupled classes in the system under investigation.

There is a need for research to address many urgent issues arising from the use of coupling and its effects. For example, we hypothesise that there may be an optimum level of inheritance-based coupling in systems and similarly for non-inheritance based coupling. The main focus of future research will therefore be to carry out further empirical evaluations to establish whether such an optimum level exist. This empirical research should be performed with as many industrial-sized systems of varying application domains as possible, supported by well-designed hypotheses. Industrial-strength tools are needed to help in the collection of data.

An investigation into the relationship between pattern use and coupling, (and the effect of pattern use on understandability) is much needed. It would also be interesting to analyse both the number of known errors and their severity. Finally, an in-depth study of the effect of the use of friends in systems would be revealing, and will be a focus of future research.

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

System	*Zero*		*One*		$\geq$ *2*	
	Total	*Percent.*	*Total*	*Percent.*	*Total*	*Percent.*
System 1 (Gnu)	29	30	42	44	25	26
System 2 (LEDA)	28	14	72	37	97	49
System 3 (SEG1)	71	63	28	25	14	12
System 4 (SEG2)	29	48	13	21	19	31
System 5 (EFOOP2)	8	62	4	38	0	0

Table 6: CBO values for each system

this work.

REFERENCES

[1] V. R. Basili. The role of experimentation in software engineering: Past, current, and future. In *Proc 18th ICSE*, pages 442–449, 1996.

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

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

[4] B. W. Boehm, B. Clark, and E. Horowitz et al. COCOMO 2.0. *Annals of Software Engineering 1(1)*, pages 1–24, 1995.

[5] L. Briand, L. Bunse, J. Daly, and C. Differding. An experimental comparison of the maintainability of object-oriented and structured design documents. In *Proceedings of Empirical Assessment in Software Engineering (EASE) '97, Keele, UK*, 1997.

[6] L. Briand, P. Devanbu, and W. Melo. An investigation into coupling measures for c++. In *Proceedings of the 19th International Conference on Software Engineering (ICSE'97), Boston, USA*, pages 412–421, 1997.

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

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

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

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

[11] S. R. Chidamber and C. F. Kemerer. MOOSE: Metrics for object oriented software engineering. In *Workshop on Processes and Metrics for Object Oriented Software Development, OOPSLA '93, Washington*, 1993.

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

[13] 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*, 3(3):143–158, 1996.

[14] S. Henry and D. Kafura. Software structure metrics based on information flow. *IEEE Transactions on Software Engineering*, 7(5):510–518, 1981.

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

[16] IEEE729. Glossary of software engineering terminology. International standard, IEEE Computer Society Press, 1983.

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

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

[19] A. A. Porter, L. G. Votta, and V. R. Basili. Comparing detection methods for software requirements inspections: A replicated experiment. *IEEE Transactions on Software Engineering*, 21(6):563–575, 1995.

[20] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorenson. *Object-oriented modelling and design*. PHI, 1991.

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