Reusability Ranking of Software Components by Coupling Measure

Gui Gui, Paul D.Scott
Department of Computer Science, University of Essex, Colchester, UK
ggui@essex.ac.uk, scotp@essex.ac.uk

Abstract: This paper provides an account of new measures of coupling developed to assess the reusability of Java components retrieved from the internet by a search engine. These measures differ from the majority of established metrics in two respects: they reflect the degree to which entities are coupled or resemble each other, and they take account of indirect couplings or similarities. An empirical comparison of the new measures with eight established metrics is described. The new measure is shown to be consistently superior at ranking components according to their reusability.

Keywords: Component, Coupling metrics, Measurement, Reusability

1. INTRODUCTION

The work reported in this paper arose as part of a project whose goal is to develop a system for retrieving Java components from the internet. The user of such a search engine would be seeking components that can be readily integrated to perform some clearly defined and distinct function within a larger software system. The first step of such a search is clearly to find components offering the required functionality. This aspect of the system is outside the scope of the present paper but has been described elsewhere [1]. However, material retrieved from the internet is notoriously variable in quality. Consequently, it seems highly desirable that the search engine should also provide the user with an indication of both how reliable the component is and how readily it may be adapted for inclusion as part of a larger software system.

These two aspects of a component are closely related. A well designed component, in which the functionality has been appropriately distributed to its various subcomponents, is more likely to be fault free and will be easier to adapt. Appropriate distribution of function underlies two key concepts of object-oriented design: coupling and cohesion. Coupling is the extent to which the various subcomponents interact. If they are highly interdependent then changes to one are likely to have significant effects on the behaviour of others. Hence loose coupling between its subcomponents is a desirable characteristic of a component.

Coupling is widely used as symbols to measure software quality. Many metrics have been proposed to measure the coupling to predict the fault-proneness and maintainability of software. Their reasonability and performance had been evaluated in theory and empirically [3][4][5] [13][14][15][16[17][18]. However, few researches had been done using coupling to measure reusability of components because of their limitations and the difficulties to evaluate the reusability of components.

We therefore decided that the component search engine should provide the user with reusability rankings of retrieved components based on measures of their coupling. (Other factors, notably cohesion and customizability, also have implications for reusability, but these, although part of our system, are outside the scope of the present paper). There is a substantial literature on coupling metrics and this is briefly surveyed in the next section. We then describe in detail the metrics we have developed for use in our system which attempt to address some of the limitations of existing metrics. In particular, we consider both the strength and transitivity of dependencies. The following section describes an empirical comparison of our proposed metrics and several popular alternatives as predictors of reusability. Section 5 presents an analysis of the results which demonstrate that our proposed metrics consistently outperform the others. The paper concludes with a discussion of the implications of the research.

2. COUPLING METRICS

During the last 15 years a considerable amount of research effort has been devoted to developing coupling metrics for object-oriented software. Space does not permit an exhaustive review of this work but the appendices of a recent paper by Kanmani, Uthariraj, Sankaranarayanan and Thambidurai [9] provide a useful summary of many of these metrics. Here we concentrate on their major features with an emphasis on those widely-used metrics used in our comparative study.

Coupling is defined as: two objects are coupled if and only if at least one of them acts upon the other [18]. Since coupling is the degree of interaction between classes, the basic idea underlying all coupling metrics is very simple: count how many interclass interactions there are in the system. Nevertheless there is considerable variation depending on what counts as an interaction, how the counting is done and how the totals are normalised. Kanmani et al. [9] tabulate 29 such metrics, of which 18 were produced by a single research group [3].

Name	Definition
CBO [5][6]	Classes are coupled if methods or instance variables in one class are used by the other. CBO for a class is number of other classes coupled with it.
RFC [5][6]	Count of all methods in the class plus all methods called in other classes.
CF [4][7]	Classes are coupled if methods or instance variables in one class are used by the other. CF for a software system is number of coupled class pairs divided by total number of class pairs.
DAC [10][13]	Data abstraction coupling. DAC for a class is the number of attributes having other classes as their types.

TABLE 1. Coupling metrics

Table 1 summarises the characteristics of the metrics used in our comparative study. Two of them (CBO and RFC) are part of the influential CK metric suite [5][6][12] developed by Chidamber and Kemperer. CBO is perhaps the most obvious measure: a count of the number of classes that are coupled to a given class. RFC is not really a pure coupling measure since it also includes the class's own methods; however, it performed well in our experiments. CF is part of the MOOD metric suite [4][7] while DAC [10] is substantially different as it measures the coupling due to data abstraction.

All four measures have two important features in common. First, most of them treat interaction between a pair of classes as a binary quantity; that is, no account is taken of how many interactions there may be between a given pair of classes. Although other measurements can specify how much message or data communications between classes (RFC [5][6] and DAC[10][13]), they didn't concern the impact of the total number of attributes in the classes. The percentage of the invocations in a class is more important than the pure number of invocations because it presents how much a class depends on other classes. Second, they treat coupling as an intransitive relation; that is no account is taken of the indirect coupling that arises if a class A interacts with another class B and B in turn interacts a third class C but A has no direct interaction with C. Both these characteristics may lead to an inaccurate measure of the mutual dependencies of the subcomponents of a system.

3. PROPOSED NEW METRICS

We carried out a pilot study, essentially a small scale version of that described in the next section, using the four metrics listed in Table 1. They were selected, as representative of the much larger range described in the literature, largely on the basis of how frequently they were referred to in later publications. The results suggested that none was very effective in ranking the reusability of Java components. We therefore decided to develop alternative coupling metrics in the hope of achieving superior performance. Since all the metrics considered treated class interaction and method similarity as binary quantities, one obvious step was to develop measures that reflected the extent to which a pair of classes was coupled or a pair of methods resembled each other. Because none of the measures treated coupling or similarity as transitive relations, we decided that such indirect dependencies should be incorporated into our metrics.

In order to develop a coupling metric that takes account of both the degree of coupling and transitive (i.e indirect) coupling between classes, we begin by regarding any object-oriented software system as a directed graph, in which the vertices are the classes comprising the system. Suppose such a system comprises a set of classes C ≡ { C_1, C_2,…C_m}. Let $M_j \equiv \{M_{j,1}, M_{j,2}, \ldots. M_{j,n}\}$ be the methods of the class C_j, and $R_{j,i}$ the set of methods and instance variables in class C_i invoked by class C_j for $j \neq i$ ($R_{j,j}$ is defined to be null). Then the edge from C_j to C_i exists if and only if $R_{j,i}$ is not null. Thus an edge of the graph reflects the direct coupling of one class to another. The graph is directed since $R_{j,i}$ is not necessarily equal to $R_{i,j}$. R_j, the set of all methods and instance variables in other classes that are invoked by class C_j, can be defined as formula (1):

$$R_j \equiv \bigcup_{1 \le i \le m} R_{j,i} \tag{1}$$

The next step is to associate a number with each edge that reflects the extent of direct coupling from one class to another. Clearly it should be larger if the class invokes more of the other's methods. However, this number should also reflect the fact that a class that invokes many methods has a greater likelihood of invoking methods from any particular class. We therefore define CoupD(i,j), our measure of direct coupling of class C_i to C_j, is defined by formula (2).

$$CoupD\ (i,\ j) = \frac{|R_{i,j}|}{|R_i| + |M_i|} \tag{2}$$

Note that the denominator is the total number of methods invoked by class C_i and that $1 \geq CoupD(i,j) \geq 0$.
The next step is to include the indirect coupling between classes. Suppose that CoupD(i,j) and CoupD(j,k) have finite values but that CoupD(i,k) is zero. Thus although there is no direct coupling between classes C_i and C_k, there is a dependency because C_i invokes methods in C_j which in turn invokes methods in C_k. Since the strength of this dependency depends on the two direct couplings of which it is composed, a reasonable measure is provided by their product, CoupD(i,j) × CoupD(j,k). This notion is readily generalised. A coupling between two classes exists if there is a path from one to the other made up edges whose CoupD values are all non-zero. The strength of the coupling is the product of all those CoupD values. Thus we define CoupT(i,j,π), the transitive coupling between classes C_i and C_j due to a specific path π, as

$$CoupT\ (i,\ j, \pi) = \prod_{e_{s,t} \in \pi} CoupD\ (s, t) = \prod_{e_{s,t} \in \pi} \frac{|R_{s,t}|}{|R_s| + |M_s|} \tag{3}$$

$e_{s,t}$ denotes the edge between vertices s and t. Note first that CoupT includes the direct coupling, which corresponds to path of length one, and second that, because the CoupD values are necessarily less than one, transitive couplings due to longer paths will typically have lower values.

In general there may be more than one path having a non-zero CoupT value between any two classes. This raises the question. of how their values may be combined to produce an overall measure of the coupling between the classes. One approach would be to develop some method of adding their values but this raises a number of complications such as how to deal with the fact that some of the paths will have edges in common. While solutions to these difficulties may exist, a simpler pragmatic approach is to simply select the path with largest CoupT value and hence define Coup(i, j), the strength of coupling between the two classes, C_i and C_j to be:

$$Coup(i,\ j) = CoupT(i,\ j, \pi_{\max}) \tag{4}$$

where

$$\pi_{\max}(i,\ j) = \arg\max_{\pi \in \Pi} CPT(i,\ j, \pi) \tag{5}$$

and Π is the set of all paths from C_i to C_j. Note that, because of the attenuation of coupling over longer paths, where a direct coupling exists, that will typically be value selected for the strength of coupling; Coup. While there are obvious objections to ignoring the other weaker couplings, the ultimate test of this method is whether it is effective. The results reported later in this paper suggest that this simple approach is adequate.
Having established a measure for the strength of coupling between pairs of classes, the final step is to use this as a basis for a measure of the total coupling of a software system. This is readily achieved by summing all the couplings of all class pairs and dividing by the total number of such pairs. The weighted transitive coupling (WTCoup) of a system is thus defined as formula (6), where m is the number of classes in the system.

$$WTCoup = \frac{\sum_{i,j=1}^{m} Coup\ (i,\ j)}{m^2 - m} \tag{6}$$

2. AN EXPERIMENTAL COMPARISON

One of the major methodological difficulties in software metrics research is the difficulty of empirical evaluation. Having proposed a new metrics, how does one set about determining if it is effective or better than the alternatives? Consequently, most existing metrics rest on a very slender experimental support [3][6][8]. In the present study, we had the advantage that our messages were to be used for a very specific purpose: predicting how much effort would be required to reuse a component within a larger system.

We therefore chose to measure reusability as simply the number of lines of code required to modify a component in order to extend its functionality in a predefined way because they can reflect the effort required to reuse the component in the new system. The codes required to extend a component include the new codes added and the codes modified. In the experiments, very few codes are deleted; therefore, the deleted codes can be ignored. The more lines required, the lower the reusability. This appears to us to be a crude but reasonable measure of the effort that would be required for a programmer to adapt a component for use within a larger system. Time spent on modifying the components is not selected as a measure because it is difficult to collect the accurate time people use to program. Further, time spent on programming can not properly reflect the effort required because of different thinking methods and the interruption from outside during the process of programming. Three case studies were carried out:

Case 1: HTML Parser
The original components analysed HTML documents, eliminated tags and comments and output the text. The required extension was to count and output the number of tags found during parsing.

Case 2: Lexical Tokenizer
The original components tokenized a text document using user supplied token rules and output the tokens on a web interface. The required extension was to count and output the number of tokens retrieved.

Case 3: Barcode Generator
The original components accepted a sequence of alphanumeric characters and generated the corresponding barcode. The required extension was to count the number of letters.

All three included web based user interfaces. For each case, 15, 25 and 20 Java components with the specified functionality were retrieved from a repository of about 10,000 Java components retrieved from the internet. The requisite extensions were then implemented by a very experienced Java programmer and the number of additional lines of code required (NLOC) counted. The modifications required were modest, largely because of the effort involved in implementing them; about 5000 lines of code were needed for the 60 components considered in the experiment. Despite the relative simplicity of the extensions, there was considerable variation in the quantity of extra code required, as shown in Table 2.

Cases	Mean	Max	Med	Min	Std Dev
HTML Parser	116.3	158	124	58	34.2
Lexical Token.	62.2	110	61	17	26.4
Barcode Gener.	74.1	120	74	24	30.3

TABLE 2: Number of lines of code (NLOC) required to modify components

Having established a measure of the reusability of the components, we then proceeded to investigate how successful the various measures of coupling and cohesion are in predicting this quantity. Five measures of coupling (our proposed measure WTCoup, CF [3], CBO [4], RFC [4] and DAC [10]) were calculated. The data obtained for the five coupling measures when applied to the HTML parser components are shown in Table 3.

Where metrics were defined for classes rather than complete systems, the average value for all the classes in the system was the measure used. The scope of measures is various. WTCoup and CF are in (0,1) because all of them normalized by the total number of attributes in the objects. On the other hand, CBO, RFC and DAC are in the scope $(1, +\infty)$ because they only count the number of interacted objects. In order to facilitate presentation of the results on the same graph, those measures that do not necessarily produce values in the range (0,1) were divided by the maximum value 100 to produce values of comparable magnitude. From the data, it can be seen that each measure has enough variance to distinguish each component. Therefore, they are effective to be used in analysis procedure. The consistent data is observed in other cases as well.

Measures	Mean	Med	Max	Min	StdDev
WTCoup	0.29	0.314	0.37	0.14	0.065
CF	0.32	0.32	0.425	0.18	0.06
CBO	21.05	16	57	6	14.20
RFC	53.73	50	92	18	17.51
DAC	17	18	35	7	0.075

TABLE 3: Descriptive statistics data for coupling measures

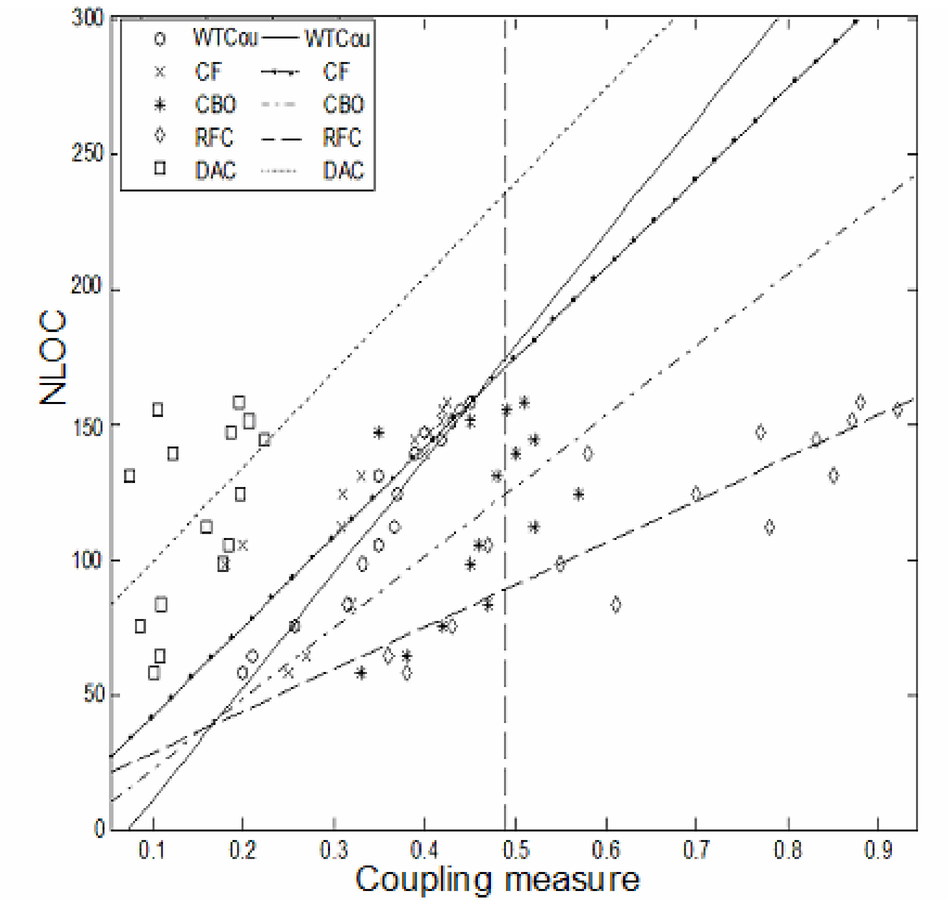
5. AN EXPERIMENTAL COMPARISON

Two approaches were used to evaluate the performance of the various measures in predicting reusability: linear regression and rank correlation [15].

5.1 Linear Regression

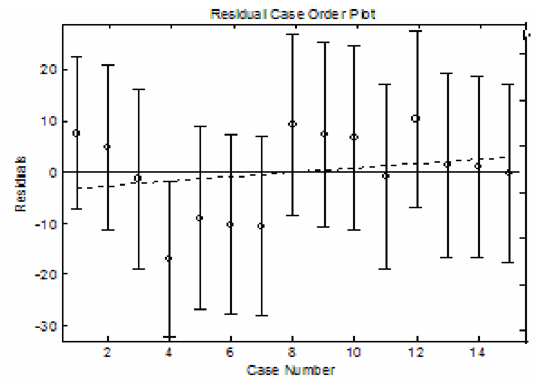
Logistic regression [2][17]is not selected because NLOC is not binary. The regression lines obtained for the five coupling measures when applied to the HTML parser components are shown in Figure 1. The results for the other two sets of components were similar. It is clear that the distribution of WTCoup is closer to a line than other measures. Therefore, WTCoup provides much more consistent predictors than others. There are no obvious systematic departures from linearity so the use of linear regression appears reasonable.

FIGURE 1: Regression of coupling measures against reusability (NLOC) for HTML parsers

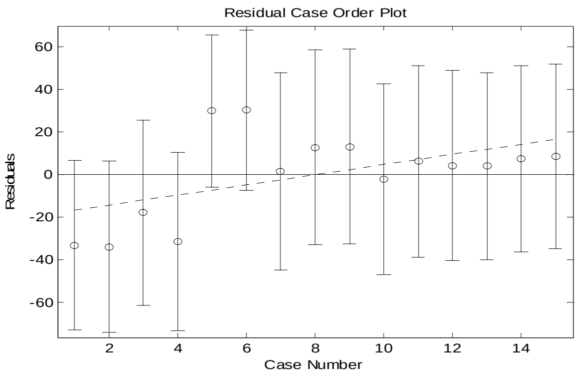


The residuals in 95% confidence interval obtained for the five coupling measures when applied to the HTML parser components are shown in Figure 2. Less residual indicates that the regression lines can reflect the distribution of data more precisely. The results for the other two sets of components were similar. It is clear that WTCoup measure is much more close to the regression line than other measures. Therefore, WTCoup has better linear predictive capability on reusability.

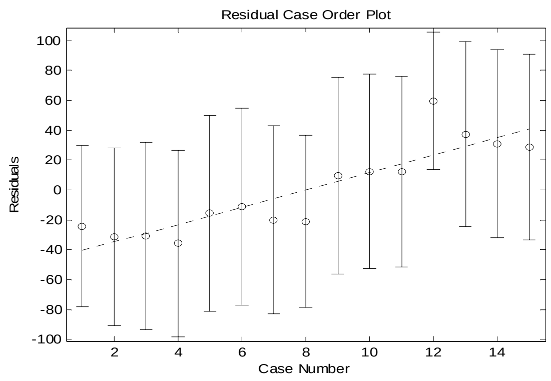
FIGURE 2. Residuals of coupling measures for HTML parsers



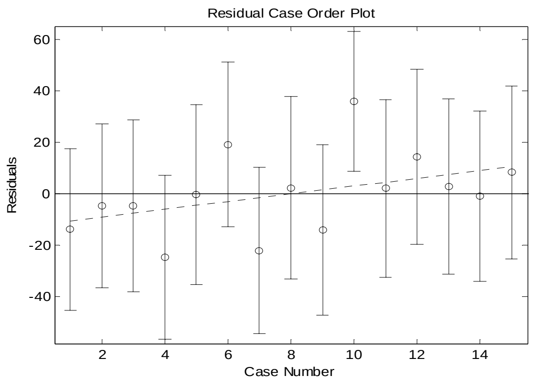
A. WTCoup



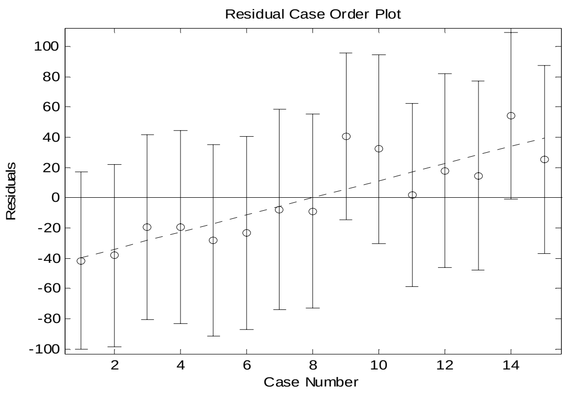
B. CF



C. CBO



D. RFC



E. DAC

	HTML Parser	Lexical Tokenizer	Barcode Generator
WTCoup	0.846	0.836	0.958
CF	0.621	0.098	0.693
CBO	0.259	0.004	0.121
RFC	0.793	0.729	0.534
DAC	0.254	0.738	0.507

TABLE 4: R^2 values for coupling measure regression lines

The coefficient of determination, R^2, provides a measure of how much of the variation in NLOC is accounted for by the coupling measure. Table 4 displays the values of R^2 obtained for each of the coupling measures on all three sets of components. In each case, our proposed new measure, WTCoup, gave the largest value of R2, indicating that it was the best linear predictor of reusability. The best of the alternatives was RFC whose results were consistent but substantially lower than those obtained for WTCoup. The remaining three coupling measures produced at least one R^2 value so low as to indicate that the correlation was not significantly above chance at the 5% level. Further, p value demonstrates consist results as R^2. For most measures, quite tiny p values demonstrate the significance of linear relationship except for CBO which indicate the relationship of CBO and NLOC is not significant.

5.2 Spearman Rank Correlation
Although these results provide a strong indication that the proposed new measures are better predictors of reusability than the alternatives, our primary purpose is simply to rank a set of components retrieved from the repository. We therefore also computed the Spearman rank correlation coefficients [15] between the rankings determined by NLOC and those produced by the various coupling (Tables 5).
The relative performances of the various measures are consistent with the regression studies. In all cases, the proposed measure, WTCoup, produced the highest rank correlations. They are in fact extremely high; no value was lower than 0.95.

Measures	HTML Parser	Lexical Tokenizer	Barcode Generator
WTCoup	0.975	0.952	0.974
CF	0.882	0.291	0.758
CBO	0.465	0.117	0.485
RFC	0.896	0.822	0.656
DAC	0.507	0.817	0.800

TABLE 5: Spearman rank correlations values for coupling measures

6. DISCUSSION

These results clearly demonstrate that our proposed metrics for coupling is a very good predictor of the number of lines of code required to make simple modifications to Java components retrieved from the internet. Furthermore, they demonstrate that, for this predictive task, the proposed metrics is superior to any of the alternative established metrics that we tested in our experiment. In this concluding section we first consider possible improvements to our metrics. Then we assess whether they are adequate for the purpose for which they were developed: producing a reusability ranking for components retrieved from the internet. Finally we consider whether our results may be of relevance to other applications of software metrics.

6.1 Possible Improvements
Both our proposed measures share two significant characteristics in the way they measure relationships between pairs of entities. First, they are weighted; that is, they use a numeric measure of the degree of coupling or similarity between entities rather than a binary quantity. Second they are transitive; that is, they include indirect coupling or similarity mediated by intervening entities.

It is reasonable to enquire whether both these characteristics are necessary to achieve good prediction performance. In fact our investigations suggest that both contribute to the performance. For example, Table 9 shows both the R^2 values of linear regression and the Spearman rank correlations achieved by WICoup, an intransitive variant of WTCoup in which all indirect couplings were set to zero. The performance of the intransitive version is consistently poorer than that of WTCoup, although it still compares favourably with the other coupling metrics (See Table 4 and Tables 5).

The method used, in our metrics, to determine the strength of indirect relationships when a pair of vertices was linked by multiple paths was crude though effective: we simply chose the strength indicated by the strongest path. The consequence of this is that indirect relationships may be underestimated. It would be possible to remedy this by aggregating the weights contributed by all possible paths between two vertices. However, in view of the fact using only the largest works well, we see no reason to make such a change.

	HTML Parser		Lexical Tokenizer		Barcode Generator	
	R2	Rho	R2	Rho	R2	Rho
WICoup	.828	.921	.782	.888	.818	.908
WTCoup	.846	.975	.836	.952	.958	.974

TABLE 6. R^2 values and rank correlations (Rho) for WICoup and WTCoup.

In this paper, only the impact of coupling on the reusability is considered, as we mentioned in the previous section, other factors including customizability cohesion can also be used to measure the component reusability. In the future, research and experiments are required to determine if the degree of cohesion can be presented numerically and if the intransitive relationship on the similarity of subcomponents can reflect the reusability. Assume all of the factors can be used to measure the component reusability; further problems are if the combination of these measures can improve the reliability of measures and how to combine them. Therefore, further experimental investigation is required to determine how to combine these measures.

6.2 Adequacy for Component Ranking

The question of whether the proposed metrics provide trustworthy rankings of Java components depends essentially on the reliability of the quantity we have chosen to measure reusability: the number of lines of code required to implement a straightforward modification. The results clearly show that the metrics are excellent predictors of the ranking of this quantity. Hence the worth of the metrics has been demonstrated provided it is reasonable to assume that NLOC is itself a good indicator of how readily a component may be adapted for use in a larger software system.

It could be objected that making a modest extension to the functionality is not the same as integrating the component into a larger system. This is true, but the types of changes needed are likely to be similar. It could be argued that the modifications used in our study were of a broadly similar nature and therefore may not be indicative of what would be found on a wider range of components and modifications. This is a legitimate objection that could only be addressed by an appreciably larger study. The present investigation involved modifying 60 programs and hence represents a substantial effort. A further criticism might be that all the modifications were carried out by one programmer; other programmers might have produced different results. Again this is a fair objection that could only be remedied by a larger scale study. Ultimately we expect that our system will be tested in practical use as part of a component search system and feedback from users will be provide an indication of the how good the quality ranking metrics are.

6.3 Wider Implications

This work arose from, and is intended primarily as a contribution to, search engine technology. Nevertheless, we believe it may be of interest to a wider body of researchers: in particular, those involved in developing and evaluating software metrics. The majority of coupling metrics treat coupling and similarity as simple binary quantities. Our results suggest that a numeric representation of the degree of the relationship leads is advantageous. Transitive dependencies have received relatively little attention in the literature. Our findings suggest that coupling and coherence metrics may be improved by their inclusion. The reliability of proposed metrics is only tested on Java components, and we expect they also can be applied in other components in future investigation.

Our methodology may also be of interest to those engaged in evaluating software metrics. Empirical evaluations are difficult in this field, largely because it is difficult to obtain suitable material for carrying out systematic studies. Because our study was carried out as part of the development of a component search engine, we were in the happy position of being able to access significant numbers of independently developed programs that had essentially similar functionality. The approach of implementing the same small change to all of them enables a substantial number of comparable data points to be assembled for use in evaluations.

ACKNOWLEDGMENTS

We are grateful to the four UK higher education funding bodies (for England, Scotland, Wales and Northern Ireland) for an Overseas Research Studentship (ORS/2002015010) awarded to G. Gui.

REFERENCES

[1] Gui, G. and Scott, P. D. Vector Space Based on Hierarchical Weighting: A Component Ranking Approach to Component Retrieval. In Proceedings of the 6th International Workshop on Advanced Parallel Processing Technologies (APPT'05) (Hong Kong, China, Oct 2005).
[2] Bieman, J. M. and Kang, B-Y. Cohesion and Reuse in an Object-Oriented System. In Proc. ACM Symposium on Software Reusability (SSR'95). (April 1995) 259-262.
[3] Briand, L., Devanbu, P. and Melo, W. An investigation into coupling measures for C++. Proceedings o+f ICSE 1997, (Boston, USA, 1997).

[4] Brito e Abreu, F. and Melo, W. Evaluating the impact of OO Design on Software Quality. Proc. Third International Software Metrics Symposium. (Berlin 1996).
[5] Chidamber, S. R. and Kemerer, C. K. Towards a Metrics Suite for Object Oriented Design. Proceedings of 6th ACM Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA'91), (Phoenix, Arizona, 1991), 197-211.
[6] Chidamber, S. R. and Kemerer, C. K. A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering, Vol. 20 (June 1994), 476-493.
[7] Harrison, R., S.J.Counsell, and R.V.Nith. Evaluation of the MOOD Set of Object-Oriented Software Metrics. IEEE Transactions on Software Engineering, Vol. 24 (June 1998), 491-496.
[8] Hitz , M. and Montazeri, B. Measuring coupling and cohesion in object-oriented systems. Proceedings of International Symposium on Applied Corporate Computing. (Monterrey, Mexico, 1995).
[9] Kanmani, S., Uthariraj, R., Sankaranarayanan, V. and Thambidurai, P. Investigation into the Exploitation of Object-Oriented Features. ACM Sigsoft, Software Engineering Notes, Vol. 29 (March 2004).
[10] Li, W. and Henry, S. Object-Oriented metrics that predict maintainability. Journal of Systems and Software. 23(2) 1993 111-122.
[11] Li, X., Liu, Z. Pan, B. and Xing, B. A Measurement Tool for Object Oriented Software and Measurement Experiments with It. In Proc. IWSM 2000. (Lecture Notes in Computer Science 2006, Springer-Verlag, Berlin, Heidelberg, 2001), 44-54
[12] Subramanyam, R. and Krishnan, M. S. Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects. IEEE Transactions on Software Engineering, Vol. 29 (April 2003), 297-310
[13] Y.-S.Lee, B.-S.Liang, S.-F.Wu, F.-J.Wang, Measuring the coupling and Cohesion of an Object-Oriented Program Based on Information Flow, In Proc. International Conference on Software Quality, Maribor, Slovenia, 1995.
[14] H. Washizaki, Y.Yamamoto and Y. Fukazawa. Software Component Metrics and It's Experimental Evaluation, In Proc. of International Symposium on Empirical Software Engineering (ISESE2002), Vol.II (2002).
[15] S.D.Conte, H.E.Dunsmore and V.Y.Shen, Software Engineering Metrics and Models Benjamin-Cummings Publishing Co., Inc.
[16] D.Kung, j. Gao, P.Hsia, F.Wen, Y.Toyoshima, C.Chen, Change Impact Identification in Object-Oriented Software Maintenance, In Proc. Conf. Software Maintenance, pp. 202-211, 1994.
[17] Briand, L., Daly. J ,Porter. V, Wust.J. A Comprehensive Empirical Validation of Product Measures for Object-Oriented System, Technical Report ISERN-98-07,1998.
[18] N. Fenton, "Software Measurement: A Necessary Scientific Basis," In IEEE Trans. Software Eng., vol. 20, no. 3, pp. 199-206, March 1994.