An Empirical Investigation into the Dimensions of Run-Time Coupling in Java Programs

Áine Mitchell[1] [2] and James F. Power
Department of Computer Science, National University of Ireland,
Maynooth, Co. Kildare, Ireland.

Abstract Software quality is an important external software attribute that is difficult to measure objectively. Several studies have identified a clear empirical relationship between static coupling metrics and software quality. However due to the nature of object-oriented programs, static metrics fail to quantify all the underlying dimensions of coupling, as program behaviour is a function of its operational environment as well as the complexity of the source code. In this paper a set of run-time object-oriented coupling metrics are described. A method of collecting such metrics which utilises the Java Platform Debug Architecture is described and a collection of Java programs from the SPECjvm98 benchmark suite are evaluated. A number of statistical techniques including descriptive statistics, a correlation study and principal component analysis are used to assess the fundamental properties of the measures and investigate whether they are redundant with respect to the Chidamber and Kemerer static CBO metric. Results to date indicate that run-time coupling metrics can provide an interesting and informative qualitative analysis of a program and complement existing static coupling metrics.

1 Introduction

Coupling is well recognised as one of the fundamental qualitative measures of the *external complexity* of a software design. A large body of research has gone into investigating how this complexity measure in turn characterises such *external quality* attributes of a design as its maintainability, reusability, reliability, and provides a means of estimating the effort needed for testing. Coupling metrics have been designed for use at various stages of the software life cycle, the majority of which are evaluated through static code analysis [4]. However, these static measures only capture certain underlying dimensions of coupling. Other dependencies regarding the dynamic behaviour of a program can only be inferred from run-time information. Features of object-oriented programming such as polymorphism, dynamic binding and inheritance render the static coupling metrics imprecise as they do not reflect perfectly the run-time situation. The quality of a software product will therefore be influenced by its operational environment as well as the source code complexity. Consequently measures that access the runtime quality may aid in the analysis of software quality.

In this analysis the Coupling Between Objects (CBO) metric, defined by Chidamber and Kemerer [6], was used. The reason being the findings from a large number of previously conducted empirical studies have shown this metric to be a good predictor of a number of external quality attributes. Studies have shown the ability of this metric to evaluate the fault proneness, maintainability, testability, change proneness and reusability of software [2, 5, 7, 12]. CBO for a class is defined as, "a count of the number of other classes to which it is coupled". Two classes are said to be coupled when "methods declared in one class use methods or instances variables of the other class".

The first set of run-time metrics we define extends the previous work by taking into account the *direction* of coupling. In the coupling relationship a class may act as a client or a server, that is it may access methods or instance variables from another class or it may have its own methods or instance variable used. To account for this, two run-time metrics were defined: run-time import coupling between objects (R_I) and run-time export coupling between objects (R_E).

The second set of run-time measures we define are designed to quantify the *strength* of the coupling relationship, that is the amount of association between the classes. A count is taken of the amount of times a class accesses methods or instances variables from other classes as a proportion

[1]Please address correspondence to ainem@cs.may.ie

[2]This work is funded by the Embark initiative, operated by the Irish Research Council for Science, Engineering and Technology (IRCSET).

Accepted for the Third International Conference on the Principles and Practice of Programming in Java
Las Vegas, Nevada, USA, June 16-18, 2004, pp. 9-14

Name	Description
RD_I	A count of number of accesses a class makes as a proportion of total number of access.
RD_E	A count of number of accesses made to a class as a proportion of total number of access.
R_I	Number of classes from which a class accesses methods or instance variables at run-time
R_E	Number of classes who access methods or instance variables from a given class
CBO	Number of classes a class accesses methods or instance variables from statically

Table 1: Definition of Metrics Used in this Analysis

of total number of methods or instance variables accessed. This will give the degree to which the class is accessing data from outside its own class and is defined as run-time import degree of coupling (RD_I). The metric is also computed from the perspective of the class as a server, that is the amount of times classes are accessing data from this class as a function of the total number of accesses (RD_E). These metrics represent an improvement over $R_{\mathbf{x}}$ as they are normalised coupling measurements, that is they are measures that have a notion of maximum coupling. This may be more useful in comparing classes. Table 1 presents a summary of all metrics used in this study, further details can be found in [10, 11].

2 Experimental Platform

The dynamic analysis of any program involves a huge amount of data processing. However, the level of performance of the collection mechanism was not considered to be a critical issue at this time. It was only desirable that the analysis could be carried out in reasonable and practical time. It was however necessary to be able to collect a wide variety of dynamic information, therefore the collection mechanism had to be designed with a high degree of flexibility in mind.

We are currently concentrating on programs written in Java, as all Java programs are executed on a Virtual Machine this provides a ideal platform for profiling and analysis.

Our metric data collection system consists of a number of parts. Runtime trace information was obtained by utilising the Java Platform Debug Architecture (JPDA) [9]. This is a multi-tiered debugging architecture contained within Sun Microsystem's Java 2 SDK version 1.4.0_01. It consists of two interfaces, the Java Virtual Machine Debug Interface (JVMDI), and the Java Debug Interface (JDI), and a protocol, the Java Debug Wire Protocol (JDWP). The first layer of the JPDA, the JVMDI, is a programming interface implemented by the virtual machine. The second layer, the JDWP, defines the format of information and requests transferred between the process being debugged and the debugger front-end which implements the JDI. The JDI, which comprises the third layer, defines information and requests at the user code level. It provides introspective access to a running virtual machine's state, the class, array, interface, and primitive types, and instances of those types. This was selected because of the ease with which it is possible to obtain specific information about the run-time behaviour of a program. It also has the advantage of being compatible with a number of commercial virtual machine implementations. However, it is currently not possible to directly obtain information about the state of the execution stack using this approach.

To overcome this problem an `EventTrace` analyser class, which we have implemented in Java, carries out a stack based simulation of the entire execution in order to obtain information about the state of the execution stack. This class also implements a filter which allows the user to specify which events and which of their corresponding fields are to be captured for processing. This allows a high degree of flexibility in the collection of the dynamic trace data.

The final component of our collection system is a `Metrics` class, which is responsible for calculating the desired metrics on the fly. It is also responsible for outputting the results in text format. The metrics to be calculated can be specified from the command line. The addition of the metrics class allows new metrics to be easily defined as the user need only interface with this class. See [10, 11] for additional information.

3 Data Analysis Methodology

In this section the data is analysed to determine if the run-time coupling metrics are redundant with respect to the static CBO measure. The procedure is derived from the method outlined in [5]. This consists of descriptive statistics, a correlation study and principal component analysis.

For each case study the **distribution** (mean) and **variance** (standard deviation) of each measure is calculated. These statistics are used to select metrics that exhibit enough variance to merit further analysis, as a low variance metric would not differentiate classes very well and therefore would not be a useful predictor of external quality. Descriptive statistics will also aid in explaining the results of the subsequent analysis.

The subsequent statistical techniques all require a **normal (bivariate) data distribution**. The Shapiro-Wilk test was used to test whether the data was normally distributed. Any data that did not exhibit a normal distribution was transformed by calculating the logarithm of each data point.

A **correlation study** was undertaken to investigate how strongly the metrics are related. The Pearson or 'product moment' correlation test was used. The correlation coefficient (r) is a number that summarizes the direction and degree (closeness) of linear relations between two variables and is also known as the Pearson Product-Moment Correlation Coefficient. r can take values between -1 through 0 to +1. The sign (+ or -) of the correlation affects its interpretation. When the correlation is positive ($r > 0$), as the value of one variable increases, so does the other. The closer r is to zero the weaker the relationship. If a correlation is negative, when one variable increases, the other variable decreases. The following general categories indicate a quick way of interpreting a calculated r value [3]:

- 0.0 to 0.2 Very weak to negligible correlation
- 0.2 to 0.4 Weak, low correlation (not very significant)
- 0.4 to 0.6 Moderate correlation
- 0.7 to 0.9 Strong, high correlation
- 0.9 to 1.0 Very strong correlation

The value of r is calculated for each metric, the results of which are displayed in a correlation matrix table. In summary, the Pearson correlation is a measure of the *STRENGTH* of a relationship between two variables

Any relationship between two variables should be assessed for its *SIGNIFICANCE* as well as its strength. A standard two tailed t-test was used to determine whether the correlation coefficient was statistically significant. Coefficients were considered significant if the t-test p-value was below 0.05. This tells how unlikely a given correlation coefficient, r, will occur given no relationship in the population. Therefore the smaller the p-level, the more significant the relationship.

Principal Component Analysis (PCA) is used to analyse the covariate structure of the metrics and to determine the underlying structural dimensions they capture. In other words PCA can tell if all the metrics are likely to be measuring the same class property. PCA usually generates a large number of principal components. The number will be decided based on the amount of variance explained by each component. A typical threshold would be retaining principal components with eigenvalues (variances) larger than 1.0. This is the Kaiser criterion. See [8] for further details on PCA.

4 Results

The descriptive statistics results are illustrated by Table 2. The measures were shown to exhibit large variances which makes them suitable candidates for further analysis.

The results for the Pearson correlation coefficient test for the programs under evaluation are shown by Table 3. The values that are deemed to be significant at the level $p < 0.05$ are highlighted

in bold. In all cases the static CBO metric shows some significant degree of correlation with either R_I, RD_E metrics or both. As R_I is essentially CBO compounded at run-time, some degree of correlation between these two would be expected. For most cases the degree of correlation seem to be moderate, that is r < 0.7.

It is interesting to note the negative correlation between the RD_E and CBO that was observed in a number of cases. This could indicate an interesting feature of class behaviour in object-oriented programs. Classes may act predominately as clients or servers in any class-class relationship. If a class has a high static CBO it has the potential to send messages to a large number of classes. As its functioning principally as a client class it may not require many messages to be sent from other classes, hence the decrease of RD_E with increasing CBO. Future work will involve evaluating a static export CBO and seeing if a similar relationship exists. It would also be interesting to investigate client/server specific behaviour of classes in object-oriented programs.

Intuitively a stronger correlation would be expected between RD_I and CBO as the direction of the coupling measurement is the same. However, in all the programs studied static CBO did not show any significant correlation with RD_I or R_E.

Table 4 shows the results of the principal component analysis when all of the metrics are taken into consideration. Using the Kaiser criterion to select the number of factors to retain we find that the metrics mostly capture three orthogonal dimensions in the sample space formed by all measures. In other words for each of the programs analysed three principal components are retained.

A significant amount of variance is captured by the run-time metrics that is not accounted for by the CBO metric alone. The R_I metric belongs to the same principal component as the static CBO in most cases. This is also the case for CBO and RD_E for a select number of examples also. Analyzing the definitions of the measures that exhibit high loadings in PC1, PC2 and PC3 yields the following interpretation of the coupling dimensions:

- PC1: Measures Static CBO, R_I.
- PC2: Measures RD_I, RD_E are all normalised coupling measures, that is they have a notion of maximum coupling.
- PC3: Measures R_E.

Overall the PCA results seem to suggest that the run-time coupling metrics are not redundant with the static CBO metric and that they capture additional dimensions of coupling. Therefore the values show that they are not just surrogate static CBO metrics, suggesting that additional information over and above that which is obtainable form the static CBO metrics can be extracted using run-time metrics.

Figure 1 illustrates the results for the RD_I and RD_E metrics for all programs used in this analysis. Each bar in the graph represents the number of classes that exhibit a run-time import/export coupling that falls within the specified range. Looking at the compress results in the R_I_DC graph, this program has 10 classes that exhibit 0%-25% import coupling, 3 classes in the 25%-50% range, 5 in the 50%-75% range and 5 in the 75%-100% range.

5 Conclusion and Future Work

This paper proposed number of run-time coupling metrics designed to quantify the external quality of an object-oriented application. A method for collecting such measures was proposed which utilised the Java Platform Debug Architecture. An empirical investigation of the metrics was conducted using Java programs from the SPECjvm98 and Benchmark Suite.

The differences in the underlying dimensions of coupling captured by static versus run-time coupling metrics was assessed using a correlation study and principal component analysis. The investigation was conducted using the static CBO metrics as defined by Chidamber and Kemerer. The results indicated that the run-time metrics did capture different properties than the static metrics although it should be noted that some degree of correlation did exist. Results indicate that

_201_compress

	Mean	SD
RD_I	33.75	36.37
RD_E	52.91	33.07
R_I	1.72	2.11
R_E	1.80	1.16
CBO	6.24	6.2

_202_jess

	Mean	SD
RD_I	66.36	31.48
RD_E	73.55	26.55
R_I	2.97	7.21
R_E	2.97	9.01
CBO	6.99	4.78

_205_raytrace

	Mean	SD
RD_I	52.12	34.52
RD_E	66.24	30.14
R_I	2.14	4.25
R_E	2.06	1.89
CBO	7.25	7.51

_209_db

	Mean	SD
RD_I	41.15	37.95
RD_E	51.50	31.58
R_I	1.81	1.98
R_E	1.88	1.54
CBO	9.12	6.6

_213_javac

	Mean	SD
RD_I	60.24	39.54
RD_E	74.24	32.15
R_I	3.21	3.01
R_E	3.01	2.87
CBO	8.54	7.15

_222_mpegaudio

	Mean	SD
RD_I	52.86	41.52
RD_E	54.77	34.99
R_I	2.60	2.36
R_E	2.60	2.70
CBO	5.745	4.9

_227_mtrt

	Mean	SD
RD_I	53.45	35.25
RD_E	67.21	31.21
R_I	2.19	4.35
R_E	2.14	1.94
CBO	7.52	7.61

_228_jack

	Mean	SD
RD_I	50.21	42.07
RD_E	65.99	33.70
R_I	2.68	5.37
R_E	2.68	2.39
CBO	6.05	7.51

Table 2: Descriptive Statistic Test Results for all Programs

_201_compress

	RD_I	RD_E	R_I	R_E	CBO
RD_I	1	−0.34	**0.78**	−0.17	0.36
RD_E	−0.34	1	**−0.58**	0.05	**−0.51**
R_I	**0.78**	**−0.58**	1	−0.25	**0.57**
R_E	−0.17	0.05	−0.25	1	0.16
CBO	0.36	**0.51**	**0.57**	0.16	1

_202_jess

	RD_I	RD_E	R_I	R_E	CBO
RD_I	1	**0.51**	**0.24**	**−0.19**	0.23
RD_E	**0.51**	1	−0.06	0.05	**−0.26**
R_I	**0.24**	−0.06	1	0.12	**0.64**
R_E	**−0.19**	0.05	0.12	1	0.06
CBO	0.23	**−0.26**	**0.64**	0.059	1

_205_raytrace

	RD_I	RD_E	R_I	R_E	CBO
RD_I	1	**0.38**	0.39	−0.28	0.16
RD_E	**0.38**	1	0.26	**0.43**	0.63
R_I	0.39	**0.26**	1	−0.11	**0.65**
R_E	−0.28	**0.43**	−0.11	1	0.11
CBO	0.16	0.63	**0.65**	0.11	1

_209_db

	RD_I	RD_E	R_I	R_E	CBO
RD_I	1	−0.40	**0.66**	−0.07	0.32
RD_E	−0.40	1	**−0.62**	0.18	−0.35
R_I	**0.66**	**0.62**	1	0.19	**0.54**
R_E	−0.07	0.18	−0.19	1	0.11
CBO	0.32	−0.35	**0.54**	0.11	1

_213_javac

	RD_I	RD_E	R_I	R_E	CBO
RD_I	1	−0.43	**0.56**	−0.21	0.34
RD_E	−0.43	1	**0.49**	0.01	0.65
R_I	**0.56**	**0.49**	1	−0.31	**0.35**
R_E	−0.21	0.01	−0.31	1	0.15
CBO	0.34	0.65	**0.35**	0.15	1

_222_mpegaudio

	RD_I	RD_E	R_I	R_E	CBO
RD_I	1	**0.52**	**0.64**	−0.02	0.25
RD_E	**0.52**	1	**0.37**	0.06	0.04
R_I	**0.64**	0.37	1	−0.01	**0.090**
R_E	−0.02	0.06	−0.01	1	0.65
CBO	0.25	0.04	**0.55**	0.09	1

_227_mtrt

	RD_I	RD_E	R_I	R_E	CBO
RD_I	1	**0.33**	0.126	−0.28	0.06
RD_E	**0.33**	1	0.15	**0.43**	0.63
R_I	0.13	**0.15**	1	−0.01	**0.65**
R_E	−0.28	**0.43**	−0.01	1	0.03
CBO	0.06	0.63	**0.65**	0.03	1

_228_jack

	RD_I	RD_E	R_I	R_E	CBO
RD_I	1	**0.32**	0.12	−0.26	0.09
RD_E	**0.32**	1	**0.29**	**−0.39**	**−0.45**
R_I	0.12	**−0.29**	1	−0.01	**0.79**
R_E	−0.26	**−0.39**	−0.01	1	0.05
CBO	0.09	**−0.45**	**0.79**	0.05	1

Table 3: Pearson Correlation Coefficient Test Results for all Programs

it is worthwhile to continue the investigation into run-time coupling metrics and their relationship with the external quality.

There are plans to extend this work in a number of ways. We hope to develop a comprehensive set of run-time object-oriented metrics that can intuitively quantify such aspects of object-oriented applications such as inheritance, dynamic binding, polymorphism and dynamic binding.

We also hope to investigate other applications of run-time coupling metrics such as the quantification of software testing strategies. Clearly a static analysis is relatively independent of program behaviour, whereas any run-time analysis will be fundamentally influenced by the testing strategy and test input. Therefore using this type of approach of it will be possible to measure how different test case inputs affect the internal properties of a program.

It is also a goal to improve on the additional performance overhead that results from the use of the JPDA during the collection of the dynamic trace information.

References

[1] Arisholm, E., "Dynamic Coupling Measures for Object-Oriented Software" *8th IEEE International Software Metrics Symposium (METRICS 2002)*, Ottawa, Canada, 4-7 June 2002.

[2] Basili, V.R., Briand, L.C. and Melo W.L., "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Transactions on Software Engineering*, Vol. 22, no. 10, pp. 751–761, October 1996.

[3] The Correlation Coefficient, Available at the following WWW site: `http://www.bized.ac.uk/timeweb/crunching/crunch_relate_expl.htm`.

Accepted for the Third International Conference on the Principles and Practice of Programming in Java Las Vegas, Nevada, USA, June 16-18, 2004, pp. 9-14

_201_compress			
	$PC1$	$PC2$	$PC3$
RD_I	0.034	**0.816**	0.163
RD_E	0.452	**0.734**	0.081
R_I	**0.874**	0.022	0.015
R_E	**0.625**	0.054	**0.551**
CBO	**0.524**	0.226	0.011

_202_jess			
	$PC1$	$PC2$	$PC3$
RD_I	0.227	**0.618**	0.010
RD_E	0.002	**0.767**	0.122
R_I	**0.775**	0.011	0.007
R_E	0.006	0.065	**0.895**
CBO	**0.771**	0.065	0.009

_205_raytrace			
	$PC1$	$PC2$	$PC3$
RD_I	0.043	**0.722**	0.006
RD_E	0.230	**0.665**	0.321
R_I	**0.616**	0.064	0.017
R_E	0.171	0.151	**0.743**
CBO	**0.743**	0.654	0.004

_209_db			
	$PC1$	$PC2$	$PC3$
RD_I	0.423	**0.865**	0.007
RD_E	0.199	0.432	0.128
R_I	**0.785**	0.011	0.033
R_E	0.030	0.001	**0.557**
CBO	**0.580**	0.159	0.137

_213_javac			
	$PC1$	$PC2$	$PC3$
RD_I	0.023	**0.893**	0.114
RD_E	0.112	**0.704**	0.217
R_I	**0.578**	0.187	0.003
R_E	0.232	0.216	0.455
CBO	**0.775**	0.359	0.036

_222_mpegaudio			
	$PC1$	$PC2$	$PC3$
RD_I	0.112	**0.854**	0.003
RD_E	**0.508**	**0.745**	0.175
R_I	**0.780**	0.310	0.035
R_E	0.003	0.017	**0.675**
CBO	0.340	0.029	0.110

_227_mtrt			
	$PC1$	$PC2$	$PC3$
RD_I	0.143	**0.631**	0.326
RD_E	0.331	**0.754**	0.104
R_I	**0.626**	0.124	0.003
R_E	0.278	0.031	**0.614**
CBO	**0.705**	0.044	0.002

_228_jack			
	$PC1$	$PC2$	$PC3$
RD_I	0.018	0.373	0.344
RD_E	0.432	**0.654**	0.001
R_I	**0.654**	0.189	0.005
R_E	0.108	0.451	**0.575**
CBO	0.482	0.111	0.002

Table 4: Principal Component Analysis Test Results for all Programs

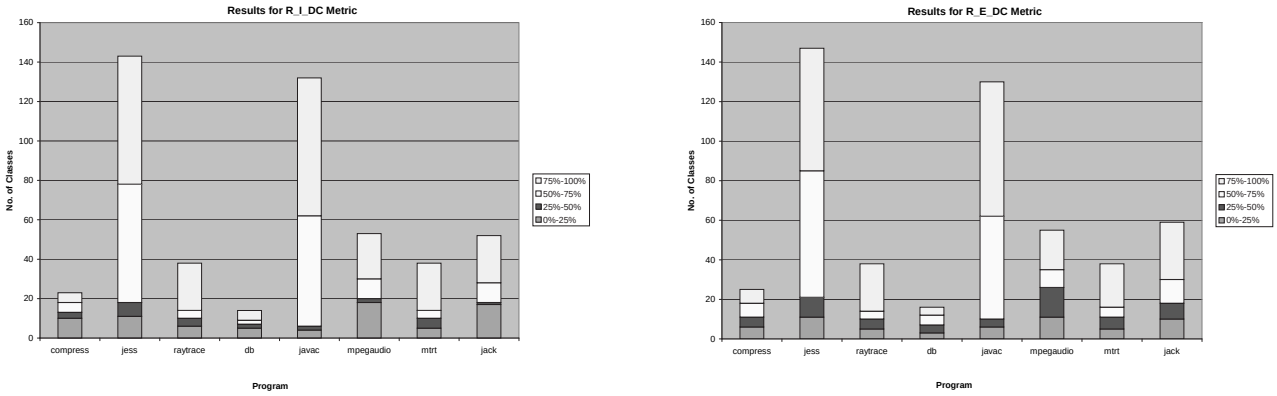

Figure 1: Dynamic Metric Results for Sample Programs

[4] Briand, L.C., Daly, J.W. and Wüst, J.K., "A Unified Framework for Coupling Measurement in Object-Oriented Systems," *IEEE Transactions on Software Eng.*, Vol. 25, no. 1 pp. 91–121, Jan/Feb 1999.

[5] Briand, L.C. and Wüst, J., "Empirical Studies of Quality Models in Object-Oriented Systems," *Advances in Computers*, Academic Press, vol. 59, pp. 97-166, 2002.

[6] Chidamber, S.R. and Kemerer, C.F., "A Metrics Suite for Object-Oriented Design," *IEEE Transactions on Software Engineering*, Vol. 20, no. 6, pp. 467–493, June 1994.

[7] Eder J., Kappel G. and Schrefl M., "Coupling and Cohesion in Object–Oriented Systems" *Technical Report*, University of Klagenfurt, 1994.

[8] Jolliffe, I.T., "Principal Component Analysis", 2nd Edition, Springer Verlag, 2002.

[9] Java Platform Debug Architecture (JPDA), Available at the following WWW site: `http://java.sun.com/products/jpda`.

[10] Mitchell, A. and Power, J.F., "An Approach to Quantifying the Run-time Behaviour of Java GUI Applications," *Winter International Symposium on Information and Communication Technologies* , Cancun, Mexico, Jan. 5th-8th, 2004.

[11] Mitchell, A. and Power, J.F., "Towards a definition of run-time object-oriented metrics" *Proceedings of the 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'2003)*, Darmstadt, Germany, July 22, 2003.

[12] Wilkie, F.G. and Kitchenham, B.A., "Coupling Measures and change ripples in C++ Application Software," *The Journal of Systems and Software*, Vol. 52, pp. 157–164, 2000.