Using Object-Level Run-Time Metrics to Study Coupling Between Objects

Áine Mitchell[*]
Department of Computer Science
National University of Ireland, Maynooth,
Co. Kildare, Ireland
ainem@cs.may.ie

James F. Power
Department of Computer Science
National University of Ireland, Maynooth,
Co. Kildare, Ireland
jpower@cs.may.ie

ABSTRACT

In this paper we present an investigation into the run-time behaviour of objects in Java programs, using specially adapted coupling metrics. We identify objects from the same class that exhibit non-uniform coupling behaviour when measured dynamically.

We define a number of object level run-time metrics, based on the static Chidamber and Kemerer coupling between objects (CBO) measure. These new metrics seek to quantify coupling at different layers of granularity, that is at class-class and object-class level. We outline our method of collecting such metrics and present a study of the programs from the JOlden benchmark suite as an example of their use.

A number of statistical techniques, principally agglomerative hierarchical clustering analysis, are used to facilitate the identification of such objects.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*complexity measures, product metrics*; G.3 [**Probability and Statistics**]

General Terms

Measurement

Keywords

Object-level Coupling Metrics, Object Behaviour, Cluster Analysis

1. INTRODUCTION

Coupling was first introduced in the context of structured development techniques by Stevens et al. [18]. They defined coupling as, "the measure of the strength of association established by a connection from one module to another". They stated that the stronger the coupling between modules, i.e., the more inter-related they are, the more difficult these modules are to understand, change, and correct and thus the more complex the resulting software system.

A good software system should exhibit low coupling between its units. Coad and Yourdon [9] transfered the principal of low coupling to object-oriented software. A large assortment of coupling measurement for object-oriented systems have been defined [4].

The Chidamber and Kemerer CBO metric [8] is the most accepted and widely used object-oriented coupling design metric. A number of empirical studies have shown this metric to be a good predictor of the maintainability, fault proneness, testability, change proneness and reusability of a software design [2, 3, 11, 19]. However, the CBO metric is defined based on a static analysis of class code, and the ability of the CBO metric to accurately predict the actual amount of coupling between *objects* is as yet unproven. As a static metric, CBO cannot capture all the dimensions of object-level coupling, as features of object-oriented programming such as polymorphism, dynamic binding and inheritance render CBO imprecise in evaluating the run-time behaviour of an application. The behaviour of a program is going to be a function of its operational environment as well as the complexity of the source code. Therefore static metrics may fall short when determining the run-time properties of a program.

If the static CBO measure is to be believed it would be expected that objects derived from the same class would exhibit similar coupling behaviour, that is that they would be coupled to the same classes and make the same accesses. In this paper we evaluate whether static CBO provides a true measure of coupling between objects, or whether it is restricted to being a measure of the level of coupling between classes. To this end, we define a number of metrics to evaluate run-time coupling between objects at the class-class and object-class level. We employ a number of statistical techniques to evaluate this data to analyze run-time object behaviour.

The remainder of this paper is organized as follows. Section 2 describes related work in the field of static and run-time coupling metrics. Section 3 describes the goals and hypothesis of the study. Section 4 outlines the experimental design and defines the run-time metrics. Section 5 presents a study of the metrics using the JOlden benchmark suite, and

[*]Please address correspondence to Áine Mitchell at the above address.

these results are discussed in Section 6. Section 7 concludes the paper and describes future research.

2. RELATED WORK

Chidamber and Kemerer originally defined CBO [7] for a class as “a count of the number of noninheritance related couples with other classes.” An object of a class is coupled to another if methods of one class use methods or instance variables defined by the other. They later revised their definition to state [8], “CBO for a class is a count of the number of other classes to which it is coupled.” A footnote stated that this “includes coupling due to inheritance.”

Briand et al. [4] carried out an extensive survey of the available literature on coupling in object-oriented systems and concluded that all the metrics at that time measured coupling statically, at the class level. No measures of run-time object level coupling had been proposed.

Yacoub et al. [17] described a set of dynamic coupling metrics designed to evaluate the change-proneness of a design. The metrics were applied at the early development phase to determine design quality. They used executable object-oriented design models to model the application to be tested. The metrics were evaluated for a number of different execution scenarios, and they extended the scenarios to have an application scope.

Arisholm et al. [1] defined and validated a number of dynamic coupling metrics and studied the relationship of these with the change proneness of a system. They found that the dynamic coupling measurement did capture additional properties to the static coupling metrics and were good predictors of the change proneness of a class.

In previous work we conducted a number of studies [14, 15] on the quantification of a variety of run-time class-level coupling metrics for object-oriented programs. We used a statistical analysis to investigate the differences in the underlying dimensions of coupling captured by static versus the run-time coupling metrics. The results indicated that the run-time metrics did capture different properties than the static metrics alone. We concluded that it is worthwhile to continue the investigation into run-time coupling metrics and their relationship with external quality, as extra information can be provided by the run-time metrics to complement that obtainable from a simple static analysis.

Clustering analysis has been proposed in many areas of research [12]. In general, whenever one needs to classify a large amount of information into manageable meaningful piles, cluster analysis is of use. To the best of our knowledge there is no work describing the application of clustering analysis in identifying object behaviours, however it has been used in such studies as identifying candidate objects in procedural source code [16].

3. GOALS AND HYPOTHESES

The GQM/MEDEA framework proposed by Briand et al. [5] was used to set up the experiments for this study. First, the goal of the experiments is outlined. Next, we state the perspective on the goal. Finally, the environment is described to determine the context in which the study will be carried out.

Goal: To determine if objects from the same class behave differently at runtime from the point of view of coupling.

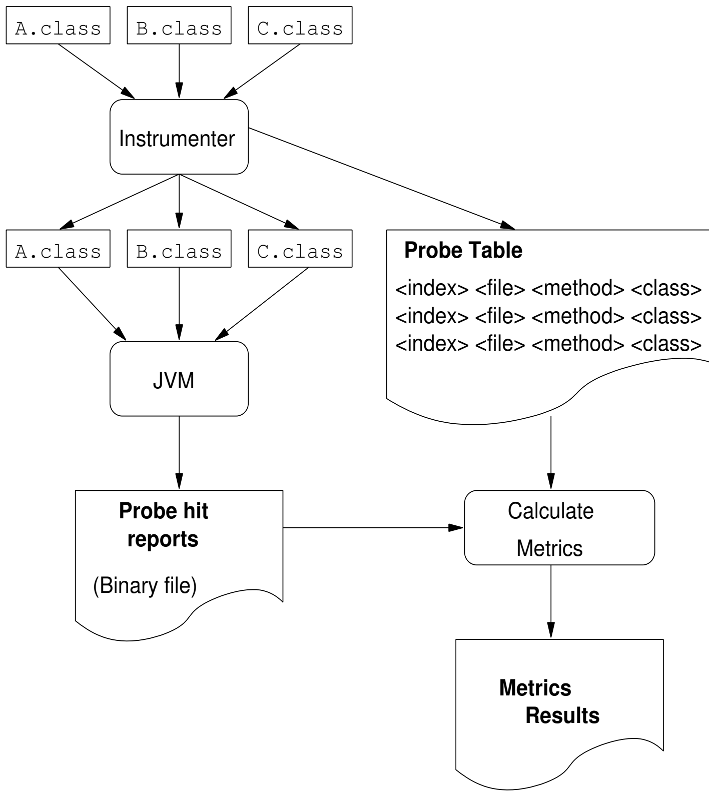

Figure 1: Collection of run-time metrics

Perspective: We investigate the behaviour of objects at run-time with respect to coupling using a number of metrics which measure the level of coupling at different layers of granularity. We use a number of statistical techniques capable of separating objects from a class into groups based on their similarity.

Environment: Since we are studying object behaviour a set of Java programs which create a large number of objects at run-time are used.

The following hypothesis is investigated in this paper:

H_0**:** Objects from a class behave similarly at run-time from the point of view of coupling

H_1**:** Objects from a class behave differently at run-time from the point of view of coupling

4. EXPERIMENTAL DESIGN

4.1 Metrics Data Collection Tool

There are a number of methods available for collecting run-time information from Java programs. Instrumenting a Virtual Machine, such as Kaffe, is one. However, this can result in a huge amount of data being generated for the simplest of programs due to the logging of bytecode instructions and there can be compatibility issues when compared with the Java class libraries released by Sun.

An alternative is to use Sun’s JVM profiler API, the Java Platform Debug Architecture (JPDA). This is faster than instrumenting a VM and is more robust. Previously we have used this method in class-level metrics analysis. However, it is still very time consuming to generate a profile for a large application and it is difficult to conduct an object-level analysis using this approach.

Another technique is bytecode instrumentation. This involves modification of the class file content in order to acquire run-time information. This approach seems to add the least overhead to the execution of the program and also provides object-level accuracy, which is essential for this type of analysis.

Our metrics data collection tool uses the publicly available Apache Byte Code Engineering Library (BCEL) [10], and is based on the Gretel coverage monitoring tool [13].

The BCEL is an API which can be used to analyze, create, and manipulate (binary) Java class files. Classes are represented by BCEL objects which contain all the symbolic information of the given class, such as methods, fields and byte code instructions. Such objects can be read from an existing file, be transformed by a program and dumped to a file again.

The first stage our *Instrumenter* program takes a list of class files and instruments them. During this phase the BCEL inserts probes into these files to flag events like method calls or instance variable accesses. During instrumentation, the class files are changed in-place, and a file containing information on method and field accesses is created. Each method and field are given a unique index in this file. When the application is run, each probe records a “hit” in another file. Our *metrics* program then calculates the run-time measures utilizing the information in these files. This is illustrated in Figure 1.

The use of the BCEL still imposes some overhead on the execution of a program, although it is minimal compared to using other approaches of obtaining run-time information. Since none of the metrics dealt with in this paper are performance related this was not a significant concern at this time.

4.2 Object-level versus Class-level Coupling

Run-time *object level* coupling quantifies the level of dependencies between objects in a system. Run-time *class level* coupling quantifies the level of dependencies between the classes that implement the methods or variables of the caller object and the receiver object [1]. The class of the object sending or receiving a message may be different from the class implementing the corresponding method due to the impact of inheritance.

4.3 Run-time Object-Level Coupling Metrics

The following measures are based on the static Chidamber and Kemerer CBO metric as this is considered to be the seminal coupling metric for methods in a class.

We define the Run-time Coupling Between Objects (RCBO) metric as the number of classes that are accessed by another class at run-time. This measure will be some function of the static CBO measure, as this measure determines the classes that can be theoretically accessed at run-time. This is a coarse-grained measure which will assess class-class coupling at the object level.

For the second measure we fix one class and determine the distribution of **unique** accesses per object. We construct a matrix of such values for each class in the program under consideration. Table 1 gives an example of such a matrix, where we record the run-time coupling values for individual objects of class BlackNode, $BlackNode_1, \ldots, BlackNode_4$, against the classes GreyNode, QuadTreeNode and WhiteNode.

	$GreyNode$	$QuadTreeNode$	$WhiteNode$
$BlackNode_1$	0	2	0
$BlackNode_2$	0	2	0
$BlackNode_3$	0	2	0
$BlackNode_4$	0	2	0

Table 1: Matrix of unique accesses per object, for objects $BlackNode_1, \ldots, BlackNode_4$ to classes GreyNode, QuadTreeNode and WhiteNode

This data is statistically analyzed using cluster analysis to evaluate the behaviour of the objects. This technique groups objects together based on their similarity. The number of clusters are determined and this becomes the Number of Object-Class Clusters (N_{OC}).

4.4 Statistical Analysis

4.4.1 *Coefficient of Variation (C_V):*

In order to test our hypothesis the coefficient of variance, C_V, was calculated for the RCBO results to determine how the RCBO values varied across the objects of a class. C_V measures the relative scatter in data with respect to the mean and is calculated by dividing the mean by the standard deviation. It has no units and can be expressed as a simple decimal value or reported as a percentage value as used here. When the C_V is small the data scatter relative to the mean is small. When the C_V is large compared to the mean the amount of variation is large. Equation 1 defines the coefficient of variation as a percentage, where μ is the mean and σ is the standard deviation.

$$C_V = \sigma/\mu * 100 \qquad (1)$$

If the C_V for all classes under consideration is zero then this would lead us to accept the null hypothesis, H_0, as all objects of this classes would be accessing the same variables. However, if there was variation in the CBO values, $C_V > 0$, this would lead us to reject H_0 and accept H_1, as the objects would be behaving differently at run-time from the point of view of coupling.

4.4.2 *Cluster Analysis:*

The N_{OC} values for each class were determined using cluster analysis. This a data exploratory statistical procedure that helps reveal associations and structures of data in a domain set [16]. A measure of proximity or similarity/dissimilarity is needed in order to determine groups from a complex data set. A wide variety of such measures exist but no consensus prevails over which is superior. For the purpose of this analysis, two widely used dissimilarity measures, Pearson dissimilarity and Euclidean distance, were chosen. The analysis was conducted using these two different measures in order to verify the results.

Equation 2 defines the Pearson Dissimilarity, where μ_x and μ_y are the means of the first and second sets of data, and σ_x and σ_y are the standard deviations of the first and second sets of data.

$$d(x,y) = \frac{\frac{1}{n}\sum_i x_i y_i - \mu_x \mu_y}{\sigma_x \sigma_y} \tag{2}$$

Equation 3 defines the Euclidean Distance between two sets of data.

$$d(x,y) = \sqrt{\sum_i^n (x_i - y_i)^2} \tag{3}$$

The next step is to select the most suitable type of clustering algorithm for the analysis. An agglomerative hierarchical clustering (AHC) algorithm was chosen as it provides the output that is most related to the means of identifying coupling clusters. Also, it does not require the number of clusters the data should be grouped into be specified in advance. AHC algorithms start with singleton clusters, one for each entity. The most similar pair of clusters are merged, one pair at a time, until a single cluster remains.

Throughout the cluster analysis, there is a symmetric matrix of dissimilarities maintained between the clusters. Once two clusters have been merged, it is necessary to generate the dissimilarity between the new cluster and every other cluster. The unweighted pair group average linkage algorithm was employed here as it is theoretically the best method to use and the most likely to mimic correctly input groupings. This algorithm clusters objects based on the average distance between all pairs.

Suppose we have three clusters A, B and C, with i being the distance between A and B, and j being the distance between B and C. If A and B are the most similar pair of entities and are joined together into a new cluster D, the method of calculating the new distance k between C and D is given by Equation 4.

$$k = (i * size(A) + j * size(B))/(size(A) + size(B)) \tag{4}$$

The analysis was also repeated using Ward's method to verify the results. With this method cluster membership is assessed by calculating the total sum of squared deviations from the mean of a cluster. The criterion for fusion is that it should produce the smallest possible increase in the error sum of squares.

The output of AHC is usually represented in a special type of tree structure called a dendrogram, as illustrated by Figure 2. Each branch of the tree represents a cluster and is drawn vertically to height where the cluster merges with neighbouring clusters. The cutting line is a line drawn horizontally across the dendrogram at a given dissimilarity level to determine the number of clusters. The cutting line is determined by constructing a histogram of node levels to find where the increase in dissimilarity is strongest, as then we have reached a level where we are grouping groups that are already homogenous. The cutting line is selected before this level is reached.

In order to accept H_0 we would expect objects from the same class to group together and occupy the same cluster, therefore expecting values of N_{OC} to be 1. The formation of a number of different clusters, where $N_{OC} > 1$, would lead us to reject H_0 and accept H_1.

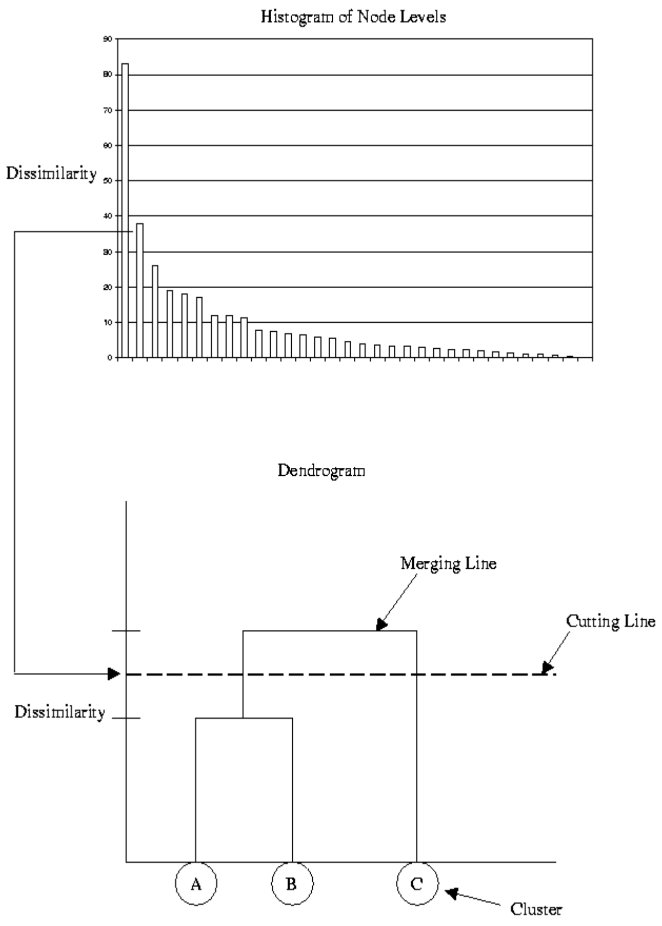

Figure 2: Dendrogram: At the cutting line there are two clusters

5. CASE STUDY

Metric data was collected for the programs from the JOlden benchmark suite [6]. The original Olden benchmarks are a suite of pointer intensive C programs with have been translated into Java. They were deemed suitable for this study due to the fact that each program exhibited a large volume of object creation.

Table 2 illustrates the Mean and C_V (RCBO) results for the objects of the classes from the programs from the JOlden benchmark suite. The static CBO values for each class are also listed for the sake of comparison. If all objects from the same class are behaving in a similar fashion we would expect them to make accesses to the same classes at run-time. Consequently, there should be little or no variability in the RCBO values for objects from the same class, for example, with class Vector in MST which had a C_V of 0. However for the programs studied the C_V values varied from 0 to 46.7%. Classes like GreyNode and QuadTreeNode in Perimeter exhibited significant variances indicating that not all of their objects were accessing variables from the same classes.

Table 3 illustrates the N_{OC} results for the programs from the JOlden benchmark suite. To ease the presentation of data, we omit classes that were instantiated a small number of times, which we define as classes that give rise to less than 10% of the maximum number of objects created by a class from the programs under evaluation.

Since cluster analysis groups objects together based on the similarity of the accesses they make to other classes, we would expect that objects from the same class would occupy the same cluster. This is the case for a number of the classes analyzed, for example class *Vertex* from MST. However, more than one cluster was observed for a number of classes used in this study. Four clusters were found for class *HashTable* from MST.

6. DISCUSSION

This ultimate goal of this study is to investigate the hypothesis that objects of a class behave differently at run-time from the point of view of coupling. To do this we use a number of run-time object-level metrics based on the static CBO measure.

The first metric, RCBO, was used to investigate whether objects of the same class type were coupled to the same classes at run-time. We found that in some cases a number of classes from the programs studied had objects that were coupled to different classes, shown by the variability in the RCBO results, ($C_V > 0$). A class might create one group of objects that access one set of classes and another that accessed a different set. So we have a number of objects from the same class that are behaving differently at run-time at the class-class level. From this, at the class-class level, we can reject H_0 and accept H_1. One cannot observe such behaviour simply by calculating the static CBO value for that class.

The next metric looks at the unique accesses an object is making from the other classes it is coupled to. Cluster analysis attempts to group the objects together based on the similarity of such accesses. We would expect objects from the same class to exhibit a single cluster ($N_{OC} = 1$) if all objects were making the same accesses. However, there were a number of clusters formed for a number of classes used in

Table 2: Static CBO and descriptive statistics for RCBO results, for classes from programs from JOlden benchmark suite.

BH			
	CBO	μ	$C_V(\%)$
$Body\$2Enumerate$	2	2	41.2
$Body$	7	2	20.9
$Cell$	3	3	14.9
$MathVector$	1	1	3.5
$Node\$HG$	2	2	0
$Node$	2	1	0

BiSort			
	CBO	μ	$C_V(\%)$
$Value$	1	1	0

Em3d			
	CBO	μ	$C_V(\%)$
$Node\$Enumerate$	1	1	35.0

Health			
	CBO	μ	$C_V(\%)$
$Hospital$	3	3	0
$List\$Enumerator$	2	2	46.7
$List$	2	1	19
$Village$	4	4	12.1

MST			
	CBO	μ	$C_V(\%)$
$HashTable$	3	3	7.7
$Vertex$	1	1	0

Perimeter			
	CBO	μ	$C_V(\%)$
$BlackNode$	2	1	0
$GreyNode$	4	4	34.2
$QuadTreeNode$	4	3	30
$WhiteNode$	1	1	0

Power			
	CBO	μ	$C_V(\%)$
$Branch$	2	2	0
$Lateral$	2	2	0
$Leaf$	1	1	15.2

TreeAdd			
	CBO	μ	$C_V(\%)$
$TreeNode$	1	1	39

TSP			
	CBO	μ	$C_V(\%)$
$Tree$	1	1	0

Voronoi			
	CBO	μ	$C_V(\%)$
$Edge$	3	1	26.4
$Vertex$	4	3	15.3

Table 3: Results of cluster analysis for classes from programs from JOlden benchmark suite, for the N_{OC} measure.

BH	
	N_{OC}
Body$2Enumerate	3
Body	4
Cell	1
MathVector	2
Node$HG	1
Node	1

BiSort	
	N_{OC}
Value	1

Em3d	
	N_{OC}
Node$Enumerate	6

Health	
	N_{OC}
Hospital	2
List$Enumerator	3
List	1
Village	4

MST	
	N_{OC}
HashTable	4
Vertex	1

Perimeter	
	N_{OC}
BlackNode	1
GreyNode	4
QuadTreeNode	4
WhiteNode	1

Power	
	N_{OC}
Branch	1
Lateral	1
Leaf	4

TreeAdd	
	N_{OC}
TreeNode	2

TSP	
	N_{OC}
Tree	1

Voronoi	
	N_{OC}
Edge	4
Vertex	5

this study. So we have the situation where a single class template was creating objects that were exhibiting different behaviours on an object-class level at run-time. Therefore at the object-class level we can reject H_0 and accept H_1.

There seems to be a relationship between the C_V and the number of clusters, with a high C_V leading to >1 clusters. Intuitively this would make sense as it is easy to see how variation in the number of classes used by an object would lead to variation in the variables they use and consequently leading to a number of groups of objects behaving differently.

From these finding it is suggested that the static coupling between objects metric would be better defined as coupling between classes as it does not give a true measure of run-time coupling between objects.

6.1 Threats to Validity

There are a number of factors which may potentially affect the validity of these run-time coupling metrics. The JOlden benchmark suite may not be representative of all classes of Java programs. Previously we have conducted studies using other benchmark suites such as the SPEC and the Java Grande Forum Benchmark suites. We have also developed a technique for collecting run-time trace information from Java GUI programs. Future work will involve evaluating a number of common "real world" Java applications using these metrics.

A general problem with any type of run-time analysis is that the results are based on dynamic measurement and are thus tied to the inputs or test cases used. The use of different test cases may produce different results. This is not an issue when using benchmark suites as they come with predefined sets of test cases which have been chosen to ensure a 'typical' run of the application. However when using GUI driven programs the test cases must be carefully chosen so as to achieve maximum coverage of the program to be analyzed.

At the current time there exists no benchmark suite specially designed for the purpose of measuring coupling. Future work may also involve developing a set of benchmark programs specifically for the purpose of evaluating properties of object-oriented programming such as coupling, cohesion etc.

7. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a number of metrics designed to evaluate run-time coupling between objects. A method of collecting such measures which utilized the BCEL was proposed. An empirical investigation of the metrics was conducted using Java programs from the JOlden Benchmark Suite.

A study into run-time object coupling behaviour was conducted using descriptive statistics and cluster analysis. These techniques were used to separate objects from the same class into groups based on their coupling properties. We identified classes that created a number of groups of objects that behaved differently at run-time from the point of view of coupling at the class-class and object-class level. This leads us to *reject* our null hypothesis and conclude that objects from the same class can behave differently at run-time from the point of view of coupling. Such behaviour is not identifiable from a simple static analysis of the source code, giving merit to the further investigation into run-time metrics and their applications.

7.1 Future Work

Future work will involve investigating the role run-time metrics may play in software testing. Run-time metrics may have implications for the quantification of the effectiveness 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.

Run-time metrics may also have a role to play in areas of research such as refactoring, as they give a better understanding of the behaviour of code in its operational environment.

We believe that this work poses some interesting challenges for future research. In particular, empirically validating run-time coupling metrics and their co-relation with external quality aspects of a design and investigating the possibility of using hybrid models which use a combination of static and run-time metrics to evaluate a design.

8. ACKNOWLEDGEMENTS

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

9. REFERENCES

[1] E. Arisholm, L.C. Briand, and A. Foyen. Dynamic coupling measures for object-oriented software. *IEEE Transactions on Software Engineering*, 30(8):491–506, 2004.

[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, October 1996.

[3] L.C. Briand. Empirical investigations of quality factors in object-oriented software. In *Empirical Studies of Software Engineering*, Ottawa, Canada, March 4–5 1999.

[4] L.C. Briand, J.W. Daly, and J.K. Wüst. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering*, 25(1):91–121, Jan/Feb 1999.

[5] L.C. Briand, S. Morasca, and V.R. Basili. An operational process for goal-driven definition of measures. *IEEE Transactions on Software Engineering*, 28(12):1106–1125, December 2002.

[6] B. Cahoon and K.S. McKinley. Data flow analysis for software prefetching linked data structures in Java. In *International Conference on Parallel Architectures and Compilation Techniques*, pages 280–291, Barcelona Spain, September 8-12 2001.

[7] S.R. Chidamber and C.F. Kemerer. Towards a metrics suite for object-oriented design. In *Object Oriented Programming Systems Languages and Applications*, pages 197–211, Phoenix, Arizona, USA, November 1991.

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

[9] P. Coad and E. Yourdon. Object-oriented analysis. 2, 1991.

[10] Markus Dahm. Byte code engineering library (BCEL), version 5.1, April 25 2004. http://jakarta.apache.org/bcel/.

[11] J. Eder, G. Kappel, and M. Schrefl. Coupling and cohesion in object–oriented systems. Technical Report 2/93, Department of Information Systems, University of Linz, Linz, Austria, 1993.

[12] J.A. Hartigan. *Clustering Algorithms.* John Wiley and Son, New York, 2nd edition, 1975.

[13] C. Howells. Gretel: An open-source residual test coverage tool, June 2002. http://www.cs.uoregon.edu/research/perpetual/-Software/Gretel/.

[14] Á. Mitchell and J.F. Power. Toward a definition of run-time object-oriented metrics. In *7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*, Darmstadt, Germany, July 22 2003.

[15] Á. Mitchell and J.F. Power. An empirical investigation into the dimensions of run-time coupling in java programs. In *3rd Conference on the Principles and Practice of Programming in Java*, Las Vegas, Nevada, June 16-18 2004.

[16] S. Phattarsukol and P. Muenchaisri. Identifying candidate objects using hierarchical clustering analysis. In *8th Asia-Pacific Software Engineering Conference*, pages 381–389, December 4-7 2001.

[17] H.H. Ammar S.M. Yacoub and T. Robinson. Dynamic metrics for object-oriented designs. In *5th International Software Metrics Symposium*, pages 50–61, Boca Raton, Florida, USA, Nov 4-6 1999.

[18] W. Stevens, G. Myers, and L. Constantine. Ibm systems j. *IEEE Transactions on Software Engineering*, 13(2):115–139, 1974.

[19] F.G. Wilkie and B.A. Kitchenham. Coupling measures and change ripples in C++ application software. *The Journal of Systems and Software*, 52(2–3):157–164, June 2000.