A Study of the Influence of Coverage on the Relationship Between Static and Dynamic Coupling Metrics

Áine Mitchell and James F. Power [*]

Department of Computer Science, National University of Ireland, Maynooth, Co. Kildare, Ireland.

Abstract

This paper examines the relationship between the static coupling between objects (CBO) metric and some of its dynamic counterparts. The dimensions of the relationship for Java programs are investigated, and the influence of instruction coverage on this relationship is measured. An empirical evaluation of 14 Java programs taken from the SPEC JVM98 and the JOlden benchmark suites is conducted using the static CBO metric, six dynamic metrics and instruction coverage data.

The results presented here confirm preliminary studies indicating the independence of static and dynamic coupling metrics, but point to a strong influence of coverage on the relationship. Based on this, this paper suggests that dynamic coupling metrics might be better interpreted in the context of coverage measures, rather than as stand-alone software metrics.

Key words: Software Engineering, Software metrics, Coupling, Coverage, Regression Analysis

1 Introduction

The concept of *coupling* was first introduced in the context of structured development techniques and defined as "the measure of the strength of association established by a connection from one module to another" [1]. The stronger the coupling between modules, i.e., the more inter-related they are, the more

[*] Corresponding author

Email address: {ainem,jpower}@cs.nuim.ie (Áine Mitchell and James F. Power).

difficult these modules are to understand, change, and correct and thus the more complex the resulting software system.

The principal of coupling was initially transfered to object-oriented software by Coad and Yourdon [2]. However, the seminal object-oriented design coupling metric for methods in a class is the *Coupling Between Objects* measure defined by Chidamber and Kemerer [3]. A number of empirical studies show this metric to be a good predictor of the maintainability, fault-proneness, testability, change-proneness and re-usability of a software design [4–6].

Arisholm et al. [7] define and validate a number of *dynamic* coupling metrics, measured at run-time, and study the relationship of these with the change-proneness of classes. They also conduct a preliminary investigation on the relationship between the static and run-time measures. Their preliminary results show that the dynamic metrics complement existing static coupling measures.

While there exists a significant body of work on static metrics, and a growing body of work on dynamic metrics, there is still relatively little research on the factors controlling dynamic metrics, and governing their relationship with their static counterparts. Although the study by Arisholm et al. indicates a link between dynamic coupling metrics and software fault detection, the influence of the test cases used is not explored in detail.

Intuitively, when comparing static and dynamic measures, it is important to have a thorough understanding of the degree to which the analyzed source code corresponds to the code that is actually executed. This paper characterizes this correspondence using instruction coverage measures, and investigates the influence of coverage on the relationship between static and dynamic metrics. It is demonstrated that coverage results have a significant influence on the relationship and thus should always be a measured, recorded factor in any such comparison.

The remainder of this paper is organized as follows. Section 2 gives an overview of the related work in the field of static and dynamic coupling and coverage measures. Section 3 outlines the goals and hypothesis of this study and Section 4 describes the experimental design. The results of the statistical analyses are presented in Sections 5 and 6. Section 5 presents a study using Principal Component Analysis to characterize the relationship between static and dynamic metrics, and Section 6 uses linear regression to examine the influence of instruction coverage on this relationship. Section 7 concludes the paper and describes future research.

Variable	Description
CBO	Static Coupling Between Objects
I_C	Instruction Coverage
IC_CC	Import, Class Level, Number of Distinct Classes
IC_CM	Import, Class Level, Number of Distinct Methods
IC_CD	Import, Class Level, Number of Dynamic Messages
EC_CC	Export, Class Level, Number of Distinct Classes
EC_CM	Export, Class Level, Number of Distinct Methods
EC_CD	Export, Class Level, Number of Dynamic Messages

Table 1
Abbreviations for the main variables used in this study. Here, CBO is the Chidamber and Kemerer static coupling metric, I_c is the instruction coverage measure, and the remaining six variables represent the dynamic coupling metrics of Arisholm et al.

2 Background and Related Work

2.1 Static and Dynamic Coupling Metrics

The most accepted and widely used object-oriented coupling design metric is the Coupling Between Objects (CBO) measure proposed by Chidamber and Kemerer [3]. A number of empirical studies show this metric to be a good predictor of the external quality attributes of a design. Chidamber and Kemerer originally defined CBO for a class as "a count of the number of non-inheritance related couples with other classes" [8]. 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 "CBO for a class is a count of the number of other classes to which it is coupled", noting that this includes coupling due to inheritance [3].

Briand et al. carried out an extensive survey of all the currently available literature on coupling in object-oriented systems and concluded that all the metrics at that time measured coupling *statically*, at the class level [9]. No measures of run-time, or *dynamic*, coupling had at that time been proposed. Subsequently, a set of dynamic coupling metrics was described by Yacoub et al. which were intended to evaluate the change-proneness of a design [10]. 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.

A number of dynamic coupling metrics for classes are defined and validated by

Arisholm et al. [7], and are listed in the last six rows of Table 1. Each dynamic coupling metric name starts with either IC or EC to distinguish between import coupling and export coupling for the class, based on the direction of the method calls. The remaining letters distinguish three types of class-level coupling. The first metric, CC, counts the number of *distinct classes* that a method in a given class uses or is used by. The second metric, CM, counts the number of *distinct methods* invoked by each method in each class while the third metric, CD, counts the total number of *dynamic messages* sent or received from one class to or from other classes.

Arisholm et al. study the relationship of these measures with the change-proneness of classes. They find that the dynamic coupling metrics do capture additional properties compared to the static coupling metrics and are good predictors of the change-proneness of a class. Their study uses a single software system called *Velocity*, and its associated test suite, to evaluate the dynamic coupling metrics. These test cases are found to originally have 70% method coverage, which is increased to 90% for the methods that "might contribute to coupling" through the removal of dead code. However, they do not study the impact of code coverage on their results nor are results given for programs other than versions of *Velocity*.

A number of studies on the quantification of a variety of run-time class-level coupling metrics for object-oriented programs have been conducted [11, 12]. These studies use statistical analysis to investigate the differences in the underlying dimensions of coupling captured by static versus the run-time coupling metrics. The results indicate that the run-time metrics capture different properties than the static metrics alone. The studies conclude 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.

Features of object-oriented programming such as polymorphism, dynamic binding and inheritance may render CBO imprecise in evaluating the run-time behavior of an application. Therefore the static version of the metric may fall short when evaluating the run-time properties of a program. Alexander et al. conduct an extensive analysis on the effect of polymorphism on coupling measures, and develop a set of test criteria to ensure that all instances of such coupling are exercised [13]. A number of *object*-level (as opposed to class-level) run-time metrics are also used to analyze run-time object coupling behavior [14]. However, further consideration of the influence of polymorphism or inter-object differences on coupling metrics are beyond the scope of this paper.

The term *dynamic metrics* is also used for Java programs to describe general dimensions of software, such as program size, polymorphism, memory use and concurrency [15]. Indeed, a significant amount of the research on the analysis and manipulation of Java programs seeks to combine static and dynamic data, or to manipulate the dynamic behavior of Java programs through static code transformations. Recent examples include work on static and dynamic slicing [16], conflict analysis [17], super-instruction selection [18] and program optimization [19].

While such research is not directly related to coupling and cohesion metrics, many of the issues and approaches to measurement are similar. Indeed, any research that performs both static and dynamic analyses of programs benefits from being viewed in the context of some *overall* perspective of the relationship between the static and dynamic data. In particular, this paper uses dynamic coverage data, in the context of a regression analysis, to characterize the relationship between static and dynamic coupling metrics.

2.2 Dynamic Coverage Metrics

Dynamic coverage measures are typically used in the field of software testing as an estimate of the effectiveness of a test suite [20,21]. The basis of software testing is that software functionality is characterized by its execution behavior, with improved test coverage leading to improved fault coverage and improved software reliability [22]. Higher execution coverage by a program means that execution of the program provides a better characterization of that program's behavior.

There are a number of established procedures for determining test coverage in a program. These include statement coverage, branch coverage, and multiple condition coverage, as well as path and data-flow coverage measures [21]. As is the case with some static software metrics, there can often be subtle differences in the definition of these coverage measures. However, each of them seeks to measure, typically as a percentage, the degree to which a certain aspect of the program source code has been exercised at run-time.

This paper uses only one coverage measure, *instruction coverage* at the bytecode level. Working in Java bytecode instructions, rather than program statements, removes dependencies on the program source code, as well as ambiguities in the definition of what constitutes a program statement. Furthermore, this approach eliminates the potential effects of differences among Java compilers by working directly with the compiled bytecode. While more complex forms of coverage could be considered, the additional complexity in analysis was not warranted based on the scope of this study. Further, the inter-dependencies between various kinds of coverage measures would obscure the results of the regression analysis used here.

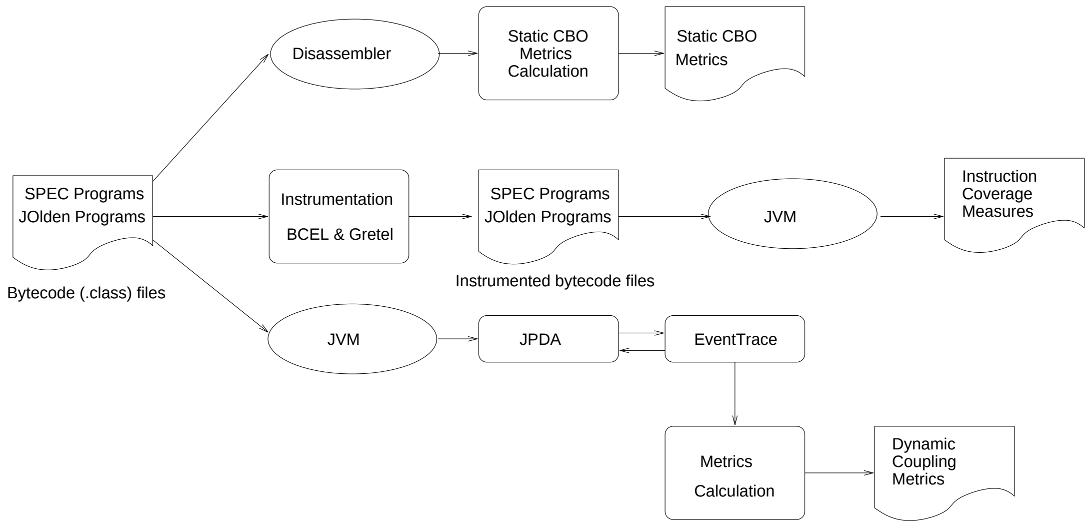


Fig. 1. *An overview of the data collection system.* This figure depicts the processes used to collect static metrics, dynamic metrics and instruction coverage data for the benchmark programs used.

3 Statement of goals and hypothesis

The experiments for this study are set up using the GQM/MEDEA framework proposed by Briand et al. [23]. Working within this framework the goal of the experiment, the perspective on the goal, and the experimental environment are all outlined below.

Goal: To examine the relationship between static CBO and dynamic coverage metrics, particularly in the context of the influence of instruction coverage.

Perspective: Intuitively, one would expect the better the coverage of the test cases used, the better the static and dynamic metrics should correlate. A number of statistical techniques are used, including multiple regression analysis, which are capable of determining if there is a significant correlation.

Environment: A number of Java programs from well-defined, publicly available benchmark suites are used in this study. Each benchmark program comes with its own set of inputs, thus defining both the static and dynamic context of the work. This contrasts with some other approaches which, at worst, can use arbitrary software packages, often proprietary, with an ad hoc set of test inputs.

The following principal hypothesis is investigated in this paper:

H_0**:** The coverage of the test cases used to evaluate a program has no influence on the relationship between static and dynamic coupling metrics.

H_1**:** The coverage of the test cases used to evaluate a program has an influence on the relationship between static and dynamic coupling metrics.

Program	Description
BH	A hierarchical O(Nlog(N)) force-calculation algorithm
Em3d	A models the propagation of electromagnetic waves through objects in 3 dimensions.
Health	A simulation of the Columbian health-care system
MST	Uses Bentley's algorithm to compute the minimum spanning tree of a graph
Perimeter	Computes the total perimeter of a region in a binary image represented by a quadtree.
Power	An optimal power pricing algorithm
Voronoi	Computes a Voronoi diagram for a random set of points
_201_compress	Modified Lempel-Ziv compression method (LZW)
_202_jess	An Expert Shell based on the CLIPS expert shell system
_205_raytrace	A raytracer that works on a scene depicting a dinosaur
_209_db	Performs multiple database functions on memory resident database
_213_javac	The Java compiler from SUN's JDK 1.0.2.
_222_mpegaudio	Decompresses ISO MPEG Layer-3 audio files
_228_jack	A Java parser generator that is based on PCCTS

Table 2
The benchmark programs used in this study. The seven programs in the top half of the table are from the JOlden benchmark suite, while the seven programs in the bottom-half of the table are from the SPEC JVM98 benchmark suite.

4 Experimental Design

In order to conduct the practical experiments underlying this study, it is necessary to select a suite of Java programs and measure:

- the static CBO metric
- the instruction coverage percentages
- the dynamic coupling metrics

The components of the data collection system are illustrated in Figure 1, and are described in the remainder of this section.

The main design objective influencing the development of the data collection system is flexibility, as it is necessary to be able to collect a variety of static and dynamic information. The dynamic analysis of any large program involves a huge amount of data processing; however, the performance level of the collection tool is not considered to be a critical issue at this time. It is only desirable that the analysis could be carried out in a reasonable and practical time.

4.1 The benchmark programs

The programs used in this study consist of the JOlden [24] and SPEC JVM98 [25] benchmark suites. The JOlden benchmarks are Java versions of pointer intensive C programs, designed to exhibit a massive number of object creations. The SPEC JVM98 suite consists of seven Java programs which are intended to represent different classes of "real world" Java applications. The programs included in the two suites are summarized in Table 2. While the SPEC JVM98 benchmark programs are more directly comparable to other studies that use Java software, this study includes the more synthetic JOlden programs to ensure that it considers programs that create significantly large populations of objects.

All the programs in the JOlden suite are distributed as Java source code, and were compiled using the *javac* compiler from Sun's SDK version 1.4.2. The programs in the SPEC suite are distributed in class file format, and were not recompiled or otherwise modified. The SPEC programs were run individually, and thus none of these results are directly comparable with the standard SPEC JVM98 metric.

Both benchmark suites include not just the programs themselves, but a test harness to ensure that results from different executions are comparable. The JOlden benchmarks were run with the supplied standard parameter settings, and the SPEC benchmarks were run at size 100. Each program was run using the client virtual machine from Sun's SDK version 1.4.2.

4.2 Static Metrics Calculation

The calculation of the static CBO metrics was carried out on each bytecode class file in the 14 benchmark programs. Each class file was disassembled, and a simple processor written in Java was then used to calculate the CBO metrics.

4.3 Dynamic Data Collection

The Java runtime system provides an almost ideal environment for profiling and analysis. Typically, such an analysis can be conducted at three main levels of granularity.

- Instrumenting a JVM whose source is publicly available.
  There are a number of open-source implementations of the JVM and, since

the JVM source code can be modified and recompiled, all aspects of a running Java program can be observed. The main drawback of this approach is the need for a detailed low-level understanding of JVM internals. Also, a number of open-source JVMs rely on third-party class libraries that might cause compatibility issues for certain programs.

- Instrumenting the Java bytecode.
  This involves modification of the class file content in order to acquire run-time information, and can be performed using a number of publicly available tools. This approach provides a relatively simple approach to dynamic analysis as it does not require a low-level knowledge of the JVM internals. It also seems to add the least overhead to the execution of the programs. However, this approach is not practical when collecting information from many different points in the bytecode and, of course, any change in the programs being analyzed requires the re-instrumentation of the code.
- Run-time profiling
  Sun Microsystem's Java 2 SDK versions 1.4 and later implement the Java Platform Debug Architecture (JPDA) [26]. The JPDA provides introspective access to a running JVM's state, including providing information about classes, arrays, interfaces, and primitive types, and their instances. This approach is faster than instrumenting a VM and is more robust. It is relatively easy to learn and the same agent works with all Virtual Machines supporting the JPDA. A drawback is that generating a profile for a large application is still quite time-consuming.

4.3.1 Coverage Data Collection

In order to calculate the instruction coverage, it is necessary to record whether or not each instruction in the program is executed. In fact, well-known techniques exist for identifying a sequence of consecutive instructions with a single entry point, known as a basic block. The instrumentation overhead is somewhat smaller if records are kept only for basic blocks instead of every instruction. Because static code analysis is required to determine basic block entry points, it is efficient to instrument the bytecode during the analysis.

The instrumentation framework uses the Apache Byte Code Engineering Library (BCEL) [27] along with the Gretel Residual Test Coverage Tool [28]. The Gretel tool statically works out the basic blocks in a Java class file and inserts a probe consisting of small sequence of bytecode instructions at each basic block. Whenever the basic block is executed, the probe code records a "hit" as a simple boolean value. The number of bytecode instructions in the basic block can then be used to calculate instruction coverage.

4.3.2 Dynamic Metrics Tool

The JPDA framework is used to measure the dynamic metrics. This framework provides an event-based notification system. This system enables user-supplied code to respond to JVM events as they occur at run-time. The principal events that are traced to measure the dynamic metrics are object creation and method call events.

In order to match objects against method calls it is necessary to model the execution stack of the JVM, as this information is not provided directly by the JPDA. An `EventTrace` analyzer class is implemented in Java, and this 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 the dynamic metrics collection system is a `Metrics` class that 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 needs only interact with this class. See [11, 29] for additional information.

5 The relationship between static and dynamic metrics

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 also aid in explaining the results of the subsequent analysis.

The descriptive statistic results are summarized in Appendix A.1. The metric values exhibit large variances which makes them suitable candidates for further analysis.

5.1 Principal Component Analysis

Principal Component Analysis (PCA) is used to analyze 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

BH

	$PC1$	$PC2$	$PC3$
CBO	0.403	0.002	**0.520**
IC_CC	**0.728**	0.224	0.012
IC_CM	**0.536**	0.391	0.001
IC_CD	**0.5553**	0.376	0.000
EC_CC	0.358	**0.522**	0.109
EC_CM	0.203	**0.763**	0.025
EC_CD	0.203	**0.763**	0.025

Em3d

	$PC1$	$PC2$	$PC3$
CBO	0.134	0.034	**0.712**
IC_CC	**0.933**	0.013	0.016
IC_CM	**0.772**	0.168	0.039
IC_CD	**0.772**	0.168	0.039
EC_CC	0.139	**0.702**	0.082
EC_CM	0.223	**0.716**	0.039
EC_CD	0.223	**0.716**	0.039

Health

	$PC1$	$PC2$	$PC3$
CBO	0.238	0.187	**0.521**
IC_CC	**0.956**	0.005	0.017
IC_CM	**0.936**	0.024	0.010
IC_CD	**0.940**	0.028	0.009
EC_CC	0.076	**0.831**	0.086
EC_CM	0.070	**0.919**	0.002
EC_CD	0.065	**0.094**	0.003

MST

	$PC1$	$PC2$	$PC3$
CBO	0.000	0.013	**0.972**
IC_CC	**0.900**	0.063	0.032
IC_CM	**0.956**	0.010	0.026
IC_CD	**0.941**	0.012	0.027
EC_CC	0.356	**0.609**	0.033
EC_CM	0.121	**0.877**	0.001
EC_CD	0.118	**0.881**	0.000

Perimeter

	$PC1$	$PC2$	$PC3$
CBO	0.231	0.123	**612**
IC_CC	**0.541**	0.169	0.281
IC_CM	**0.876**	0.080	0.002
IC_CD	**0.905**	0.056	0.038
EC_CC	0.236	**0.752**	0.000
EC_CM	0.147	**0.830**	0.023
EC_CD	0.142	**0.828**	0.026

Power

	$PC1$	$PC2$	$PC3$
CBO	0.329	0.014	**0.626**
IC_CC	**0.617**	0.073	0.161
IC_CM	**0.624**	0.338	0.036
IC_CD	**0.712**	0.228	0.041
EC_CC	0.022	**0.915**	0.015
EC_CM	0.007	**0.880**	0.112
EC_CD	0.008	**0.824**	0.164

Voronoi

	$PC1$	$PC2$	$PC3$
CBO	0.198	0.213	**0.526**
IC_CC	**0.718**	0.123	0.069
IC_CM	**0.812**	0.088	0.134
IC_CD	**0.773**	0.176	0.141
EC_CC	0.043	**0.911**	0.005
EC_CM	0.067	**0.934**	0.004
EC_CD	0.148	**0.834**	0.054

Table 3
PCA Test Results for the JOlden programs. Those values deemed to be significant at the level $p \leq 0.05$ are highlighted.

be measuring the same class property. This technique is used to investigate whether the dynamic coupling metrics are not simply surrogate measures for static CBO.

A similar study is carried out by Arisholm et al. using only their *Velocity* program [7]. This paper extends their work to cover the fourteen programs from the benchmark suites in order to demonstrate the robustness of these results over a larger range and variety of programs.

PCA can generate a large number of principal components, depending on the amount of variance explained by each component. A typical threshold, known as the Kaiser criterion, is used in this paper, and involves retaining principal components with eigenvalues (variances) larger than 1.0 [30].

5.2 PCA Results and Discussion

Tables 3 and 4 show the results of the principal component analysis used to investigate the covariate structure of the static and dynamic metrics. Using the

_201_compress

	$PC1$	$PC2$	$PC3$
CBO	0.113	0.014	**0.712**
IC_CC	**0.865**	0.065	0.186
IC_CM	**0.766**	0.154	0.097
IC_CD	**0.866**	0.073	0.100
EC_CC	0.023	**0.873**	0.176
EC_CM	0.143	**0.799**	0.035
EC_CD	0.098	**0.834**	0.096

_202_jess

	$PC1$	$PC2$	$PC3$
CBO	0.198	0.187	**0.672**
IC_CC	**0.963**	0.007	0.005
IC_CM	**0.912**	0.003	0.016
IC_CD	**0.874**	0.032	0.004
EC_CC	0.154	**0.812**	0.002
EC_CM	0.298	**0.734**	0.054
EC_CD	0.098	**0.923**	0.002

_205_raytrace

	$PC1$	$PC2$	$PC3$
CBO	0.123	0.087	**0.723**
IC_CC	**0.834**	0.021	0.019
IC_CM	**0.912**	0.017	0.008
IC_CD	**0.896**	0.103	0.001
EC_CC	0.198	**0.763**	0.003
EC_CM	0.125	**0.709**	0.017
EC_CD	0.097	**0.821**	0.002

_209_db

	$PC1$	$PC2$	$PC3$
CBO	0.012	0.163	**0.843**
IC_CC	**0.893**	0.088	0.002
IC_CM	**0.923**	0.004	0.000
IC_CD	**0.976**	0.003	0.013
EC_CC	0.178	**0.763**	0.002
EC_CM	0.110	**0.793**	0.027
EC_CD	0.087	**0.823**	0.017

_213_javac

	$PC1$	$PC2$	$PC3$
CBO	0.187	0.000	**0.973**
IC_CC	**0.633**	0.083	0.184
IC_CM	**0.834**	0.033	0.023
IC_CD	**0.723**	0.143	0.002
EC_CC	0.138	**0.834**	0.004
EC_CM	0.078	**0.734**	0.012
EC_CD	0.067	**0.759**	0.034

_222_mpegaudio

	$PC1$	$PC2$	$PC3$
CBO	0.244	0.137	**0.583**
IC_CC	**0.943**	0.004	0.087
IC_CM	**0.898**	0.034	0.041
IC_CD	**0.943**	0.023	0.001
EC_CC	0.034	**0.943**	0.043
EC_CM	0.134	**0.754**	0.085
EC_CD	0.098	**0.845**	0.005

_228_jack

	$PC1$	$PC2$	$PC3$
CBO	0.004	0.243	**0.634**
IC_CC	**0.605**	0.234	0.154
IC_CM	**0.723**	0.194	0.076
IC_CD	**0.604**	0.195	0.098
EC_CC	0.194	**0.749**	0.098
EC_CM	0.103	**0.694**	0.049
EC_CD	0.094	**0.749**	0.104

Table 4
PCA Test Results for the SPEC JVM98 programs. Those values deemed to be significant at the level $p \leq 0.05$ are highlighted.

Kaiser criterion to select the number of factors to retain shows that the metrics mostly capture three orthogonal dimensions in the sample space formed by all measures. In other words, the coupling is divided along three dimensions for each of the programs analyzed.

Analyzing the definitions of the measures that exhibit high loadings in PC1, PC2 and PC3 yields the following interpretation of the coupling dimensions:

- $PC1 = \{IC\_CC, IC\_CD, IC\_CM\}$, the dynamic import coupling metrics.
- $PC2 = \{EC\_CC, EC\_CD, EC\_CM\}$, the dynamic export coupling metrics.
- $PC3 = \{CBO\}$, the static coupling metric.

Overall the PCA results demonstrate 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 the information obtainable from the static CBO metrics, can be extracted using run-time metrics. This confirms that the findings of Arisholm et al. for the single *Velocity* program they studied are applicable across a variety of programs.

The results also indicate that the direction of coupling is a greater determining factor than the type of coupling, with $PC2$ containing the three export-based metrics, and $PC3$ containing the three import-based metrics.

6 Measuring the influence of instruction coverage

The general purpose of multiple regression analysis is to learn more about the relationship between several independent or predictor variables and a dependent or criterion variable [31]. In this study it is used to determine if instruction coverage is a factor influencing the relationship between static and dynamic metrics. The two independent variables are thus the static CBO metric and the instruction coverage measure I_c; each of the six dynamic coupling metrics in turn is then used as the dependent variable.

6.1 Multiple Regression Analysis

The general computational problem that needs to be solved in multiple regression analysis is to fit a straight line to a number of points. When there is more than one independent variable, the regression procedures will estimate a linear equation of the form shown in Equation (1), where Y is the dependent variable, X_i stands for a set of independent variables, a is a constant and each b_i is the slope of the regression line. The constant a is also known as the *intercept*, and the slope as the *regression coefficient*.

$$Y = a + b_1X_1 + b_2X_2 + \ldots + b_pX_p \tag{1}$$

The regression line expresses the best prediction of the dependent variable Y given the independent variables X_i. However, usually there is substantial variation of the observed points around the fitted regression line. The deviation of a particular point from the line is known as the *residual* value. The smaller the variability of the residual values around the regression line relative to the overall variability, the better the prediction. In most cases the ratio will fall somewhere between 0.0 and 1.0. If there is no relationship between the X and Y variables the ratio will be 1.0, while if X and Y are perfectly related the ratio will be 0.0. The least squares method is employed to perform the regression.

The R^2 or the coefficient of determination is 1.0 minus this ratio. The R^2 value is an indicator of how well the model fits the data. If there is an R^2 close to 1.0 this indicates that almost all of the variability with the variables specified in the model has been accounted for.

The correlation coefficient R expresses the degree to which two or more independent variables are related to the dependent variable, and can assume values between -1 and 1. The sign (plus or minus) of the correlation coefficient interprets the direction of the relationship between the variables. If R is positive, then the relationship of this variable with the dependent variable is positive. If R is negative then the relationship is negative. If it is zero then there is no relationship between the variables.

6.2 Analysis of Variance (ANOVA)

ANOVA is used to test the significance of the variation in the dependent variable that can be attributed to the regression of one or more independent variables. The results enable determination of whether or not the explanatory variables bring significant information to the model. ANOVA gives a statistical test of the null hypothesis H_0, which is, there is no linear relationship between the variables versus the alternative hypothesis H_1, which is, there is a relationship between the variables.

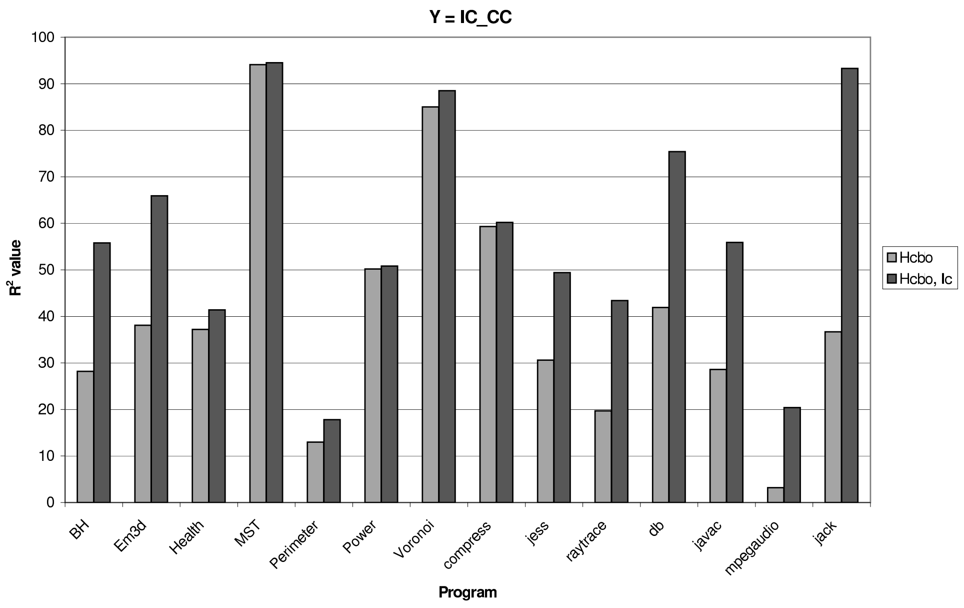
There are four parts to ANOVA results, the sum of squares, degrees of freedom, mean squares and the F test. Fisher's F test, as given by Equation (2), is used to test whether the R^2 values are statistically significant. Values are deemed to be significant at $p \leq 0.05$.

$$F = \frac{R^2 * (N - K - 1)}{(1 - R^2) * K} \tag{2}$$

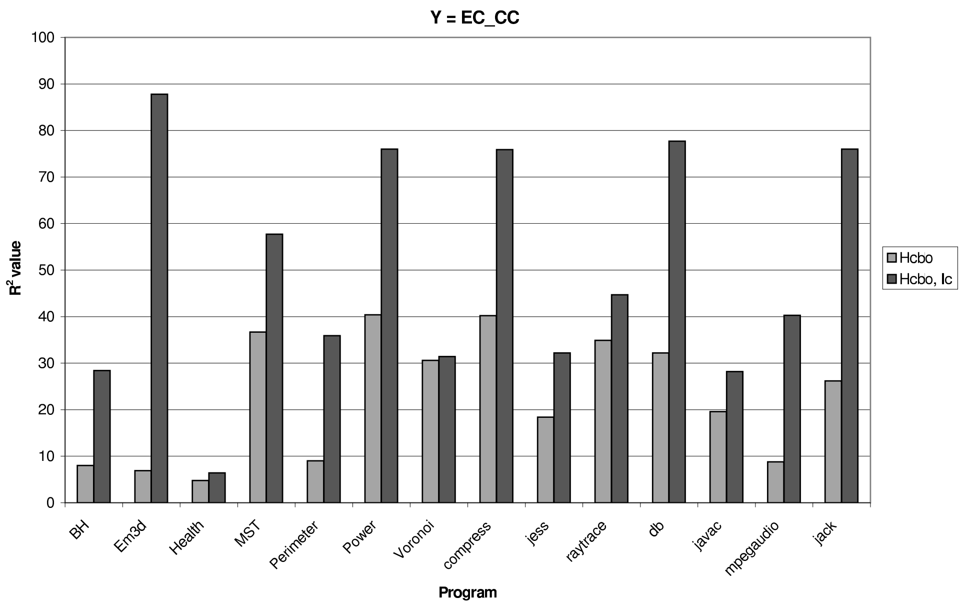
Here, K is the number of independent variables (two in this case) and N is the number of observed values.

6.3 Multiple Regression Analysis

A comprehensive list of results from the multiple regression analysis can be found in Appendices A.2 and A.3. These results are used to test the hypothesis that the instruction coverage of the test cases used to evaluate a program has no influence on the relationship between static and dynamic coupling metrics. These tables display the R^2 value that provides a measure of the proportion of variance explained by the model, while the R value gives the correlation between the dependent and independent variables. The F ratio is the test statistic used to decide whether the model as a whole has a statistically significant predictive capability.

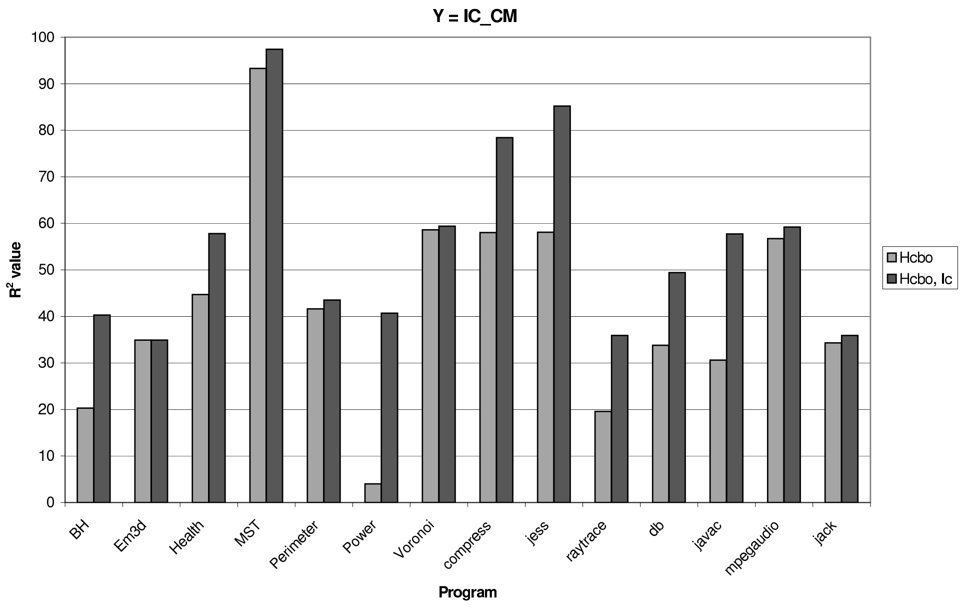


(a) *Results from the multiple linear regression where $Y = IC\_CC$.*

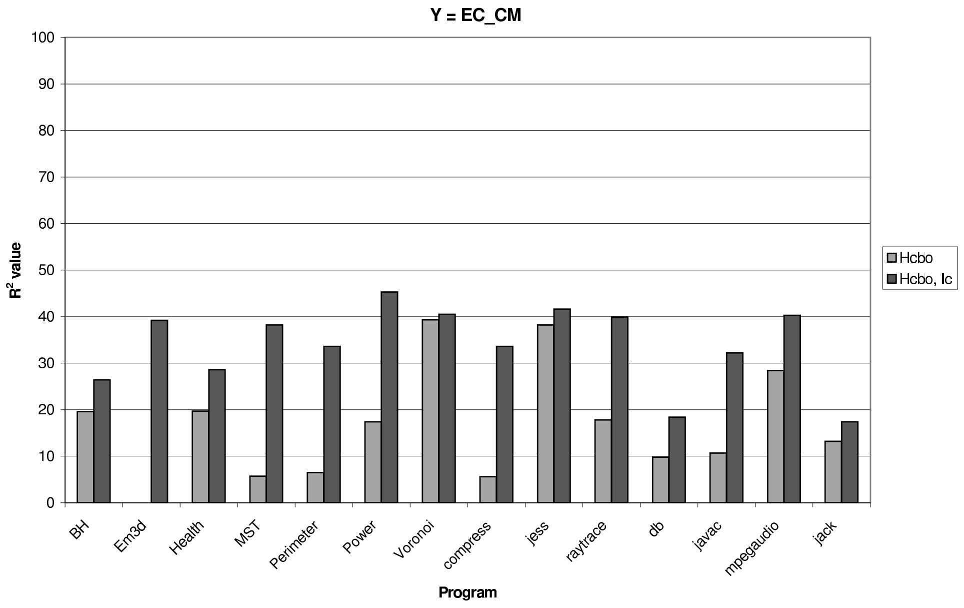


(b) *Results from the multiple linear regression where $Y = EC\_CC$.*

Fig. 2. *Regression results for Class-Level metrics (IC_CC and EC_CC).* In both graphs the lighter bar represents the R^2 value for CBO, and the darker bar represents the R^2 value for CBO and I_c combined.



(a) *Results from the multiple linear regression where $Y = IC\_CM$*



(b) *Results from the multiple linear regression where $Y = EC\_CM$*

Fig. 3. *Regression results for Method-Level metrics (IC_CM and EC_CM).* In both graphs the lighter bar represents the R^2 value for CBO, and the darker bar represents the R^2 value for CBO and I_c combined.

The first thing to note from the data is that there is a positive correlation between the dependent (dynamic metric) and independent variables CBO and I_c for all the programs used in this analysis, as all R values are positive. This means as the values for CBO and I_c increases/decreases so will the observed value for the dynamic metric under consideration.

The results of the regression analysis for each of the fourteen benchmark programs are summarized in Figures 2(a) and 2(b) for class-level dynamic coupling, and Figures 3(a) and 3(b) for method-level dynamic coupling. Of particular interest in all four graphs is the difference between the lighter bars, representing the influence of CBO, and the darker bars, representing the influence of both CBO and I_c, since these indicate the additional amount of the variation of the dynamic metric that can be allocated to instruction coverage.

6.3.1 Distinct Classes: IC_CC and EC_CC

It is immediately apparent from Figures 2(a) and 2(b) that the instruction coverage is a significant influencing factor. For example, from Figure 2(a) it can be seen that in seven of the programs, I_c accounts for an additional 20% variation. Two of the programs in Figure 2(a) that show little increase, MST and Voroni, already exhibit a high correlation with CBO alone that would have been difficult to improve on. While the increase is not uniform throughout the programs in Figure 2(a), the overall data demonstrates that instruction coverage is an important contributory factor.

Figure 2(b), representing the contribution of CBO and I_c to export coupling measured at the class level, presents a sharper contrast. Here, the influence of I_c is clearly a vital contributing factor, accounting for at least an extra 20% of the variation in nine of the fourteen programs. The important factor here is that the overall contribution of CBO to export coupling is much lower than to import coupling, as can be seen from contrasting the lighter-shaded bars in Figure 2(a) with those in Figure 2(b). Thus classes with a high level of static coupling exhibit a higher level of *import* coupling at run-time. This indicates that the coupling being exercised at run-time is from classes behaving as *clients*, making use of other class' methods, rather than those behaving as *servers*, offering their methods for use by others. The greater influence of I_c in export coupling results from there being less of a drop in its influence between IC_CC and EC_CC, suggesting that instruction coverage, as a predictor of coupling, is not as sensitive to the direction of that coupling.

6.3.2 Distinct Methods: IC_CM and EC_CM

The results for the IC_CM and EC_CM, illustrated by Figures 3(a) and 3(b), present a similar picture. Both of these dynamic metrics are scaled by the number of methods involved in the coupling relationship. Given that CBO is defined on a class-level, it does surprisingly well in influencing the IC_CM metric. Instruction coverage is also defined at a class level, but nonetheless accounts for roughly an extra 20% of the variance for four programs, and roughly an extra 10% for four other programs. The drop between import and export coupling is accentuated here but, while Figure 3(b) shows CBO proving a bad predictor for EC_CM, instruction coverage dramatically improves this for over half the programs studied.

Overall, these results show that coverage has a significant impact on the correlation between static CBO and the four run-time coupling metrics defined for distinct classes and distinct methods.

6.3.3 Dynamic Messages: IC_CD and EC_CD

Neither of the dynamic metrics based on distinct method counts, IC_CD and EC_CD exhibit a significant relationship for the programs under consideration, and are not summarized graphically. The lack of a relationship for these metrics is expected, since they are defined in terms of a count of the number of distinct times a method was executed. It is reasonable to speculate that such metrics might be more influenced by the "hotness" of a particular method, and the distribution of execution focus through the program, rather than instruction coverage data.

7 Conclusion and Future Work

This paper investigates whether the coverage of test cases used to evaluate a program have any influence on the correlation between static and dynamic metrics. An empirical investigation is conducted using the set of dynamic metrics proposed by Arisholm et al. on Java programs from the SPEC JVM98 and JOlden benchmark suites. The differences in the underlying dimensions of coupling captured by the static versus the dynamic metrics are assessed using principal component analysis. Three components are identified containing the static CBO, the import-based dynamic metrics, and the export-based dynamic metrics. This establishes that the dynamic metrics were not simply surrogate static measures, making them suitable candidates for further analysis.

A study into the predictive ability of the static CBO and instruction coverage data is then conducted using multiple regression analysis. The purpose of this is to show how well the static CBO metric and instruction coverage measure I_c could predict the six dynamic metrics under consideration. The PCA analysis places import and export based coupling in different components, and this difference is also seen in the regression analysis. Both CBO and instruction coverage have less influence overall on the export-based metrics, EC_CC and EC_CM than on the import-based dynamic metrics, IC_CC and IC_CM.

It is shown from the regression analysis that the combination of static metrics with instruction coverage gives a significantly better prediction of the run-time behavior of programs than the use of static metrics alone for the class-based and method-based metrics. This leads to the *rejection* of the null hypothesis for these four dynamic metrics, and suggests that the correlation between static and dynamic is as much a factor of coverage as an intrinsic property of the metrics themselves.

The results for the two dynamic metrics based on distinct message counts, EC_CD and EC_CD are not within the chosen significance level, and thus no determination is made on the relationship for these metrics.

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 behavior, whereas any run-time analysis will be fundamentally influenced by the testing strategy and test input. Another important aspect would be to further investigate the correlation between run-time metrics and external aspects of a design, including investigating the possibility of using hybrid models that use a combination of static and run-time metrics to evaluate a design.

Much of the work on the dynamic analysis of Java programs has come from the language design and compiler community. The work in this paper forms part of an increasing link between this community and the software engineering community, with an emphasis on collecting, analyzing and comparing quantitative static and dynamic data. Other possible examples of this synthesis include relating studies of polymorphicity with testing inheritance relationships, or relating measures of program "hot-spots" with metrics based on distinct messages such as IC_CD and EC_CD. Run-time metrics may also have a role to play in areas of research such as reverse engineering and program comprehension, as they contribute to a better understanding of the behavior of code in its operational environment.

8 Acknowledgements

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

References

[1] W. Stevens, G. Myers, L. Constantine, Structured design, IBM Systems Journal 13 (2) (1974) 115–139.

[2] P. Coad, E. Yourdon, Object-Oriented Analysis, Vol. 2, Prentice Hall, 1991.

[3] S. Chidamber, C. Kemerer, A metrics suite for object-oriented design, IEEE Transactions on Software Engineering 20 (6) (1994) 467–493.

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

[5] J. Eder, G. Kappel, M. Schrefl, Coupling and cohesion in object–oriented systems, Tech. Rep. 2/93, Department of Information Systems, University of Linz, Linz, Austria (1993).

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

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

[8] S. Chidamber, C. Kemerer, Towards a metrics suite for object-oriented design, in: Object Oriented Programming Systems Languages and Applications, Phoenix, Arizona, USA, 1991, pp. 197–211.

[9] L. Briand, J. Daly, J. Wüst, A unified framework for coupling measurement in object-oriented systems, IEEE Transactions on Software Engineering 25 (1) (1999) 91–121.

[10] S. Yacoub, H. Ammar, T. Robinson, Dynamic metrics for object-oriented designs, in: 5th International Software Metrics Symposium, Boca Raton, Florida, USA, 1999, pp. 50–61.

[11] A. Mitchell, 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, 2003, pp. 1–7.

[12] A. Mitchell, 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, 2004, pp. 9–14.

[13] R. T. Alexander, J. Offutt, Coupling-based testing of O-O programs, Journal of Universal Computer Science 10 (4) (2004) 391–427.

[14] A. Mitchell, J. F. Power, Using object-level run-time metrics to study coupling between objects, in: ACM Symposium on Applied Computing, Santa Fe, New Mexico, 2005, p. (to appear).

To appear in Science of Computer Programming 2005
- DRAFT VERSION ONLY -

[15] B. Dufour, K. Driesen, L. J. Hendren, C. Verbrugge, Dynamic metrics for Java, in: Conference on Object-Oriented Programming Systems, Languages and Applications, Anaheim, CA, USA, 2003, pp. 149–168.

[16] F. Umemori, K. Konda, R. Yokomori, K. Inoue, Design and implementation of a bytecode-based Java slicing system, in: Third International Workshop on Source Code Analysis and Manipulation, Amsterdam, The Netherlands, 2003, pp. 108–117.

[17] C. von Praun, T. Gross, Static conflict analysis for multi-threaded object-oriented programs, in: Conference on Programming Language Design and Implementation, San Diego, California, USA, 2003, pp. 115–128.

[18] M. Ertl, D. Gregg, Optimizing indirect branch prediction accuracy in virtual machine interpreters, in: Conference on Programming Language Design and Implementation, San Diego, California, USA, 2003, pp. 278–288.

[19] K. Ishizaki, M. Takeuchi, K. Kawachiya, T. Suganuma, O. Gohda, T. Inagaki, A. Koseki, K. Ogata, M. Kawahito, T. Yasue, T. Ogasawara, T. Onodera, H. Komatsu, T. Nakatani, Effectiveness of cross-platform optimizations for a Java just-in-time compiler, in: Conference on Object-Oriented Programming Systems, Languages and Applications, Anaheim, California, USA, 2003, pp. 187–204.

[20] G. Myers, The Art of Software Testing, John Wiley & Sons, 1979.

[21] R. Binder, Testing Object Oriented Systems: Models, Patterns and Tools, Addison Wesley, 1999.

[22] Y. K. Malaiya, M. N. Li, J. M. Bieman, R. Karcich, Software reliability growth with test coverage, IEEE Transactions on Reliability 51 (4) (2002) 420–426.

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

[24] B. Cahoon, K. McKinley, Data flow analysis for software prefetching linked data structures in Java, in: International Conference on Parallel Architectures and Compilation Techniques, Barcelona Spain, 2001, pp. 280–291.

[25] SPEC, SPEC releases SPEC JVM98, first industry-standard benchmark for measuring Java virtual machine performance, Press Release, http://www.specbench.org/osg/jvm98/press.html (August 19 1998).

[26] Sun Microsystems, Inc., Java platform debug architecture (JPDA), http://java.sun.com/products/jpda.

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

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

[29] A. Mitchell, J. F. Power, An approach to quantifying the run-time behaviour of Java GUI applications, in: Winter International Symposium on Information and Communication Technologies, Cancun, Mexico, 2004, pp. 1–6.

[30] I. Jolliffe, Principal Component Analysis, 2nd Edition, Springer Verlag, 2002.

[31] R. Freund, W. Wilson, Regression Analysis: Statistical Modeling of a Response Variable, Academic Press, 1998.

A Appendices

Appendix A.1 contains the descriptive statistic results for the programs from the SPEC JVM98 and JOlden benchmark suites.

Appendices A.2 and A.3 contain the results from the multiple linear regression used to test the hypothesis H_0, that coverage has no effect on the relationship between static and dynamic metrics for the programs from the JOlden and SPEC JVM98 benchmark suites. All significant results are highlighted.

A.1 *Descriptive statistic results for the programs from the SPEC JVM98 and JOlden benchmark suites.*

BH

	Mean	*SD*
CBO	5.22	3.40
IC_CC	2.33	2.50
IC_CM	7.44	8.86
IC_CD	8.67	10.84
EC_CC	2.33	1.33
EC_CM	5.77	4.44
EC_CD	6.25	4.74

Em3d

	Mean	*SD*
CBO	4.20	2.86
IC_CC	3.22	0.71
IC_CM	3.87	1.01
IC_CD	4.76	3.96
EC_CC	3.75	1.33
EC_CM	3.35	3.49
EC_CD	4.65	3.46

Health

	Mean	*SD*
CBO	3.43	3.46
IC_CC	2.43	2.46
IC_CM	3.35	4.24
IC_CD	4.25	5.46
EC_CC	3.35	3.46
EC_CM	3.55	2.43
EC_CD	4.46	4.43

MST

	Mean	*SD*
CBO	4.34	3.45
IC_CC	3.54	2.45
IC_CM	4.23	3.45
IC_CD	7.54	4.54
EC_CC	3.45	3.34
EC_CM	3.45	2.45
EC_CD	4.56	4.32

Perimeter

	Mean	*SD*
CBO	5.34	4.34
IC_CC	3.34	3.45
IC_CM	4.34	2.45
IC_CD	8.56	6.45
EC_CC	3.54	3.45
EC_CM	4.54	3.43
EC_CD	6.54	3.54

Power

	Mean	*SD*
CBO	4.50	2.54
IC_CC	1.32	0.45
IC_CM	5.23	2.23
IC_CD	5.64	2.56
EC_CC	1.54	1.45
EC_CM	4.12	4.56
EC_CD	4.67	5.35

Voronoi

	Mean	*SD*
CBO	5.43	3.46
IC_CC	2.43	1.45
IC_CM	4.54	0.45
IC_CD	7.45	3.46
EC_CC	3.45	3.46
EC_CM	4.45	2.45
EC_CD	5.36	2.46

_201_compress

	Mean	*SD*
CBO	6.24	6.2
IC_CC	1.72	2.11
IC_CM	4.34	3.54
IC_CD	7.56	5.46
EC_CC	1.80	1.16
EC_CM	4.35	4.76
EC_CD	6.56	4.56

_202_jess

	Mean	*SD*
CBO	6.99	4.78
IC_CC	2.97	7.21
IC_CM	4.34	3.43
IC_CD	5.45	4.54
EC_CC	2.97	9.01
EC_CM	4.34	4.35
EC_CD	7.56	6.56

_205_raytrace

	Mean	*SD*
CBO	7.25	7.51
IC_CC	2.14	4.25
IC_CM	4.45	3.54
IC_CD	7.56	6.56
EC_CC	2.06	1.89
EC_CM	4.54	4.53
EC_CD	6.56	4.56

_209_db

	Mean	*SD*
CBO	9.12	6.60
IC_CC	1.81	1.98
IC_CM	6.56	4.46
IC_CD	9.67	8.68
EC_CC	1.88	1.54
EC_CM	6.45	5.67
EC_CD	9.57	7.65

_213_javac

	Mean	*SD*
CBO	8.54	7.15
IC_CC	3.21	3.01
IC_CM	5.45	4.56
IC_CD	7.56	7.56
EC_CC	3.01	2.87
EC_CM	3.45	4.56
EC_CD	5.45	5.65

_222_mpegaudio

	Mean	*SD*
CBO	5.75	4.90
IC_CC	2.60	2.36
IC_CM	4.54	3.56
IC_CD	7.56	6.56
EC_CC	2.60	2.70
EC_CM	5.45	4.56
EC_CD	5.87	5.46

_228_jack

	Mean	*SD*
CBO	6.05	7.51
IC_CC	2.68	5.37
IC_CM	3.45	3.43
IC_CD	5.45	4.45
EC_CC	2.68	2.39
EC_CM	5.45	4.56
EC_CD	7.56	6.56

A.2 Multiple linear regression results for the JOlden programs

BH				
Hypothesis	Y	R	R^2	$P > F$
H_{CBO}	IC_CC	0.531	0.282	0.038
H_{CBO,I_c}	IC_CC	0.767	0.588	0.044
H_{CBO}	EC_CC	0.092	0.008	0.0001
H_{CBO,I_c}	EC_CC	0.533	0.284	0.0001
H_{CBO}	IC_CD	0.431	0.185	0.247
H_{CBO,I_c}	IC_CD	0.617	0.381	0.237
H_{CBO}	EC_CD	0.443	0.196	0.232
H_{CBO,I_c}	EC_CD	0.514	0.264	0.398
H_{CBO}	IC_CM	0.45	0.203	0.024
H_{CBO,I_c}	IC_CM	0.635	0.403	0.013
H_{CBO}	EC_CM	0.443	0.196	0.032
H_{CBO,I_c}	EC_CM	0.514	0.264	0.024

MST				
Hypothesis	Y	R	R^2	$P > F$
H_{CBO}	IC_CC	0.97	0.941	0.001
H_{CBO,I_c}	IC_CC	0.972	0.945	0.0001
H_{CBO}	EC_CC	0.606	0.367	0.002
H_{CBO,I_c}	EC_CC	0.76	0.577	0.001
H_{CBO}	IC_CD	0.966	0.933	0.002
H_{CBO,I_c}	IC_CD	0.987	0.974	0.004
H_{CBO}	EC_CD	0.239	0.057	0.649
H_{CBO,I_c}	EC_CD	0.618	0.382	0.486
H_{CBO}	IC_CM	0.966	0.933	0.002
H_{CBO,I_c}	IC_CM	0.987	0.974	0.004
H_{CBO}	EC_CM	0.239	0.057	0.049
H_{CBO,I_c}	EC_CM	0.618	0.382	0.086

Em3d				
Hypothesis	Y	R	R^2	$P > F$
H_{CBO}	IC_CC	0.617	0.381	0.046
H_{CBO,I_c}	IC_CC	0.748	0.659	0.001
H_{CBO}	EC_CC	0.262	0.069	0.03
H_{CBO,I_c}	EC_CC	0.937	0.878	0.024
H_{CBO}	IC_CD	0.59	0.349	0.294
H_{CBO,I_c}	IC_CD	0.591	0.349	0.651
H_{CBO}	EC_CD	0.02	0.00	0.975
H_{CBO,I_c}	EC_CD	0.626	0.392	0.608
H_{CBO}	IC_CM	0.59	0.349	0.194
H_{CBO,I_c}	IC_CM	0.591	0.349	0.151
H_{CBO}	EC_CM	0.02	0.000	0.075
H_{CBO,I_c}	EC_CM	0.626	0.392	0.008

Perimeter				
Hypothesis	Y	R	R^2	$P > F$
H_{CBO}	IC_CC	0.36	0.13	0.306
H_{CBO,I_c}	IC_CC	0.422	0.178	0.503
H_{CBO}	EC_CC	0.095	0.009	0.194
H_{CBO,I_c}	EC_CC	0.599	0.359	0.211
H_{CBO}	IC_CD	0.512	0.262	0.131
H_{CBO,I_c}	IC_CD	0.585	0.343	0.230
H_{CBO}	EC_CD	0.256	0.065	0.476
H_{CBO,I_c}	EC_CD	0.58	0.336	0.238
H_{CBO}	IC_CM	0.645	0.416	0.044
H_{CBO,I_c}	IC_CM	0.66	0.435	0.135
H_{CBO}	EC_CM	0.256	0.065	0.076
H_{CBO,I_c}	EC_CM	0.58	0.336	0.038

Health				
Hypothesis	Y	R	R^2	$P > F$
H_{CBO}	IC_CC	0.601	0.372	0.04
H_{CBO,I_c}	IC_CC	0.643	0.414	0.003
H_{CBO}	EC_CC	0.22	0.048	0.06
H_{CBO,I_c}	EC_CC	0.254	0.064	0.13
H_{CBO}	IC_CD	0.659	0.434	0.075
H_{CBO,I_c}	IC_CD	0.753	0.566	0.124
H_{CBO}	EC_CD	0.444	0.197	0.27
H_{CBO,I_c}	EC_CD	0.535	0.286	0.431
H_{CBO}	IC_CM	0.669	0.447	0.07
H_{CBO,I_c}	IC_CM	0.76	0.578	0.116
H_{CBO}	EC_CM	0.444	0.197	0.207
H_{CBO,I_c}	EC_CM	0.535	0.286	0.431

Power				
Hypothesis	Y	R	R^2	$P > F$
H_{CBO}	IC_CC	0.709	0.502	0.042
H_{CBO,I_c}	IC_CC	0.713	0.508	0.001
H_{CBO}	EC_CC	0.635	0.404	0.011
H_{CBO,I_c}	EC_CC	0.872	0.76	0.001
H_{CBO}	IC_CD	0.104	0.011	0.844
H_{CBO,I_c}	IC_CD	0.723	0.523	0.329
H_{CBO}	EC_CD	0.363	0.132	0.479
H_{CBO,I_c}	EC_CD	0.632	0.399	0.465
H_{CBO}	IC_CM	0.067	0.004	0.9
H_{CBO,I_c}	IC_CM	0.638	0.407	0.456
H_{CBO}	EC_CM	0.417	0.174	0.010
H_{CBO,I_c}	EC_CM	0.673	0.453	0.005

Voronoi				
Hypothesis	Y	R	R^2	$P > F$
H_{CBO}	IC_CC	0.922	0.85	0.009
H_{CBO,I_c}	IC_CC	0.941	0.885	0.0001
H_{CBO}	EC_CC	0.553	0.306	0.255
H_{CBO,I_c}	EC_CC	0.561	0.314	0.568
H_{CBO}	IC_CD	0.762	0.58	0.078
H_{CBO,I_c}	IC_CD	0.768	0.589	0.263
H_{CBO}	EC_CD	0.627	0.393	0.183
H_{CBO,I_c}	EC_CD	0.636	0.405	0.459
H_{CBO}	IC_CM	0.765	0.586	0.076
H_{CBO,I_c}	IC_CM	0.77	0.594	0.059
H_{CBO}	EC_CM	0.627	0.393	0.083
H_{CBO,I_c}	EC_CM	0.636	0.405	0.029

A.3 Multiple linear regression results for the SPEC JVM98 programs

_201_compress				
$Hypothesis$	Y	R	R^2	$P > F$
H_{CBO}	IC_CC	0.775	0.593	0.003
H_{CBO,I_c}	IC_CC	0.798	0.602	0.0001
H_{CBO}	EC_CC	0.634	0.402	0.01
H_{CBO,I_c}	EC_CC	0.870	0.759	0.007
H_{CBO}	IC_CD	0.512	0.262	0.421
H_{CBO,I_c}	IC_CD	0.599	0.359	0.201
H_{CBO}	EC_CD	0.239	0.057	0.054
H_{CBO,I_c}	EC_CD	0.422	0.178	0.134
H_{CBO}	IC_CM	0.762	0.58	0.003
H_{CBO,I_c}	IC_CM	0.885	0.784	0.006
H_{CBO}	EC_CM	0.235	0.056	0.04
H_{CBO,I_c}	EC_CM	0.58	0.336	0.035

_209_db				
$Hypothesis$	Y	R	R^2	$P > F$
H_{CBO}	IC_CC	0.419	0.178	0.0001
H_{CBO,I_c}	IC_CC	0.868	0.754	0.001
H_{CBO}	EC_CC	0.567	0.322	0.002
H_{CBO,I_c}	EC_CC	0.881	0.777	0.001
H_{CBO}	IC_CD	0.691	0.478	0.522
H_{CBO,I_c}	IC_CD	0.768	0.589	0.263
H_{CBO}	EC_CD	0.312	0.097	0.609
H_{CBO,I_c}	EC_CD	0.429	0.184	0.816
H_{CBO}	IC_CM	0.582	0.338	0.003
H_{CBO,I_c}	IC_CM	0.703	0.494	0.006
H_{CBO}	EC_CM	0.313	0.098	0.019
H_{CBO,I_c}	EC_CM	0.428	0.184	0.016

_202_jess				
$Hypothesis$	Y	R	R^2	$P > F$
H_{CBO}	IC_CC	0.553	0.306	0.002
H_{CBO,I_c}	IC_CC	0.703	0.494	0.001
H_{CBO}	EC_CC	0.428	0.184	0.031
H_{CBO,I_c}	EC_CC	0.567	0.322	0.023
H_{CBO}	IC_CD	0.765	0.586	0.145
H_{CBO,I_c}	IC_CD	0.868	0.754	0.321
H_{CBO}	EC_CD	0.691	0.748	0.246
H_{CBO,I_c}	EC_CD	0.723	0.523	0.135
H_{CBO}	IC_CM	0.762	0.581	0.023
H_{CBO,I_c}	IC_CM	0.922	0.852	0.012
H_{CBO}	EC_CM	0.618	0.382	0.001
H_{CBO,I_c}	EC_CM	0.645	0.416	0.002

_213_javac				
$Hypothesis$	Y	R	R^2	$P > F$
H_{CBO}	IC_CC	0.535	0.286	0.005
H_{CBO,I_c}	IC_CC	0.748	0.559	0.002
H_{CBO}	EC_CC	0.443	0.196	0.004
H_{CBO,I_c}	EC_CC	0.531	0.282	0.007
H_{CBO}	IC_CD	0.512	0.262	0.234
H_{CBO,I_c}	IC_CD	0.606	0.367	0.176
H_{CBO}	EC_CD	0.872	0.76	0.765
H_{CBO,I_c}	EC_CD	0.922	0.85	0.567
H_{CBO}	IC_CM	0.553	0.306	0.034
H_{CBO,I_c}	IC_CM	0.76	0.577	0.024
H_{CBO}	EC_CM	0.321	0.107	0.042
H_{CBO,I_c}	EC_CM	0.567	0.322	0.034

_205_raytrace				
$Hypothesis$	Y	R	R^2	$P > F$
H_{CBO}	IC_CC	0.444	0.197	0.021
H_{CBO,I_c}	IC_CC	0.659	0.434	0.002
H_{CBO}	EC_CC	0.59	0.349	0.043
H_{CBO,I_c}	EC_CC	0.669	0.447	0.032
H_{CBO}	IC_CD	0.256	0.065	0.342
H_{CBO,I_c}	IC_CD	0.36	0.13	0.365
H_{CBO}	EC_CD	0.239	0.057	0.123
H_{CBO,I_c}	EC_CD	0.363	0.132	0.432
H_{CBO}	IC_CM	0.443	0.196	0.034
H_{CBO,I_c}	IC_CM	0.599	0.359	0.032
H_{CBO}	EC_CM	0.422	0.178	0.012
H_{CBO,I_c}	EC_CM	0.632	0.399	0.032

_222_mpegaudio				
$Hypothesis$	Y	R	R^2	$P > F$
H_{CBO}	IC_CC	0.174	0.032	0.003
H_{CBO,I_c}	IC_CC	0.452	0.204	0.001
H_{CBO}	EC_CC	0.296	0.088	0.013
H_{CBO,I_c}	EC_CC	0.635	0.403	0.006
H_{CBO}	IC_CD	0.734	0.538	0.165
H_{CBO,I_c}	IC_CD	0.885	0.784	0.214
H_{CBO}	EC_CD	0.948	0.899	0.234
H_{CBO,I_c}	EC_CD	0.978	0.956	0.654
H_{CBO}	IC_CM	0.753	0.567	0.001
H_{CBO,I_c}	IC_CM	0.769	0.592	0.002
H_{CBO}	EC_CM	0.533	0.284	0.021
H_{CBO,I_c}	EC_CM	0.635	0.403	0.03

_228_jack				
$Hypothesis$	Y	R	R^2	$P > F$
H_{CBO}	IC_CC	0.606	0.367	0.003
H_{CBO,I_c}	IC_CC	0.966	0.933	0.012
H_{CBO}	EC_CC	0.512	0.262	0.002
H_{CBO,I_c}	EC_CC	0.872	0.76	0.003
H_{CBO}	IC_CD	0.239	0.057	0.465
H_{CBO,I_c}	IC_CD	0.618	0.382	0.045
H_{CBO}	EC_CD	0.363	0.132	0.123
H_{CBO,I_c}	EC_CD	0.419	0.178	0.576
H_{CBO}	IC_CM	0.585	0.343	0.013
H_{CBO,I_c}	IC_CM	0.599	0.359	0.002
H_{CBO}	EC_CM	0.363	0.132	0.045
H_{CBO,I_c}	EC_CM	0.417	0.174	0.032