SCIENCE CHINA
Information Sciences

An in-depth investigation into the relationships between structural metrics and unit testability in object-oriented systems

ZHOU YuMing, LEUNG Hareton, SONG QinBao, ZHAO JianJun, LU HongMin, CHEN Lin and XU BaoWen

Citation: SCIENCE CHINA Information Sciences **55**, 2800 (2012); doi: 10.1007/s11432-012-4745-x

View online: http://engine.scichina.com/doi/10.1007/s11432-012-4745-x

View Table of Contents: http://engine.scichina.com/publisher/scp/journal/SCIS/55/12

Published by the Science China Press

SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

Progress of Projects Supported by NSFC

December 2012 Vol. 55 No. 12: 2800–2815
doi: 10.1007/s11432-012-4745-x

An in-depth investigation into the relationships between structural metrics and unit testability in object-oriented systems

ZHOU YuMing[1,2], LEUNG Hareton[3], SONG QinBao[4], ZHAO JianJun[5], LU HongMin[2], CHEN Lin[1,2] & XU BaoWen[1,2]*

[1] *State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210046*, China;*
[2] *Department of Computer Science and Technology, Nanjing University, Nanjing* 210046*, China;*
[3] *Department of Computing, Hong Kong Polytechnic University, Hongkong* 999077*, China;*
[4] *Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an* 710049*, China;*
[5] *School of Software, Shanghai Jiaotong University, Shanghai* 200240*, China*

Received June 22, 2012; accepted October 13, 2012

Abstract There is a common belief that structural properties of classes are important factors to determine their unit testability. However, few empirical studies have been conducted to examine the actual impact of structural properties of classes. In this paper, we employ multiple linear regression (MLR) and partial least square regression (PLSR) to investigate the relationships between the metrics measuring structural properties and unit testability of a class. The investigated structural metrics cover five property dimensions, including size, cohesion, coupling, inheritance, and complexity. Our results from open-source software systems show that: (1) most structural metrics are statistically related to unit testability in an expected direction, among which size, complexity, and coupling metrics are the most important predictors; that (2) multivariate regression models based on structural metrics cannot accurately predict unit testability of classes, although they are better able to rank unit testability of classes; that (3) the transition from MLR to PLSR could significantly improve the ability to rank unit testability of classes but cannot improve the ability to predict the unit testing effort of classes.

Keywords testability, metrics, object-oriented, testing effort, unit testing, prediction

Citation Zhou Y M, Leung H, Song Q B, et al. An in-depth investigation into the relationships between structural metrics and unit testability in object-oriented systems. Sci China Inf Sci, 2012, 55: 2800–2815, doi: 10.1007/s11432-012-4745-x

1 Introduction

Recent years have seen an increasing interest in the testability of object-oriented software systems [1–9]. There is a common belief that, in object-oriented systems, structural properties of classes such as size, cohesion, coupling, inheritance, and complexity are important factors to determine their unit testability [2–5, 10, 11]. Previous research has focused on the theoretical relationships between structural metrics used to measure these properties and the testability of object-oriented software, yet has also made the point that these theoretical relationships need to be validated by empirical studies [10]. However, few

*Corresponding author (email: bwxu@nju.edu.cn) empirical studies have so far investigated the actual impact of these structural metrics on object-oriented software testability [1,6–9]. In practice, it is essential for software practitioners to have such quantitative data [1]. First, managers could use them to allocate testing resources and to monitor testing activities. Second, testers could use them to determine what code should be focused. Third, developers could use them to identify the code that needs to be refactored for testability enhancement.

In this paper, we deeply analyze the relationships between structural properties and unit testability, where units consist of the classes of an object-oriented software system. In this work, testability denotes "attributes of software that bear on the effort needed to validate the software product" [11]. To identify these relationships, we investigate most of structural metrics proposed in the literature to date, which capture the size, cohesion, coupling, complexity, and inheritance properties of software products. More specifically, in this study, we investigate 80 structural metrics including 8 size metrics, 24 cohesion metrics, 25 coupling metrics, 19 inheritance metrics, and 4 complexity metrics. For the analysis, we use not only multiple linear regression (MLR) but also partial least square regression (PLSR) techniques. MLR is the most commonly used regression technique that is easy to apply. It requires that the independent variables have a low collinear and the number of independent variables is smaller than the number of data points [12]. However, in software metrics data, the independent variables are often highly collinear and sometimes even the number of independent variables is larger than the number of data points. In this case, MLR is inappropriate or even not applicable. We hence also use PLSR in this study, as it is especially appropriate for dealing with such data. The objective is two-fold. First, we aim to use PLSR to build prediction models with a high accuracy. Second, we aim to introduce PLSR, a relatively new modeling technique, to software metrics community by demonstrating its usefulness in dealing with high dimensional and noisy data. The reason is that PLSR is little used in software metrics literature, although it is extremely popular in various disciplines such as chemistry, economics, medicine, psychology, and bioinformatics [12]. Based on MLR and PLSR, we attempt to answer the following questions:

(1) How are the metrics related to the effort that is required to unit test a class?

(2) Which metrics are strongly related to the effort that is required to unit test a class?

(3) How accurately do the metrics predict the effort involved in unit testing a class?

(4) Can partial least square regression models predict the effort involved in unit testing effort more accurately than multiple linear regression models?

The rest of this paper is organized as follows. Section 2 describes the research method used in this study, including data source, the independent variables, the dependent variable, the hypotheses about the independent and dependent variables, and the data analysis procedure. Section 3 introduces the modeling techniques used to build predictive models. Section 4 reports the experimental results. Section 5 concludes the paper and outlines directions for future work.

2 Research method

In this section, we describe the systems used for this study, the dependent variable, the independent variables, the hypotheses that we will investigate, and the data collection and analysis procedures.

2.1 Systems

The object-oriented software systems under study are Apache Ant 1.7.0 and JFreeChart 1.0.8. Ant 1.7.0 is a Java-based build tool which is being developed as a subproject of the Apache Web server. JFreeChart 1.0.8 is a Java chart library which makes it easy for developers to display professional quality charts. The Ant 1.7.0 and JFreeChart 1.0.8 releases also respectively provide 192 and 306 test classes, each of which tests a corresponding Java class. Each test class is a sub class of the JUnit TestCase class and hence can be activated via the JUnit test driver. A test class can consist of a number of test cases, which may need to use a common object structure. More specifically, a test class contains: (a) a number of test methods with a name starting with test; and (b) the setUp and tearDown methods used to initialize and release a common object structure (called the test fixture). In each test run, setUp and tearDown are respectively called before and after each test method. The purpose of this is to ensure that there is no side effect between test runs. Each test method typically refines the test fixture, invokes the method under test, and compares the actual results to expected results via a series of assertEquals methods. In our context, a test class is called a unit test suite. Programmers design these test classes during development and run them nightly. The one-to-one corresponding relationship between the test class and the tested Java class provides a unique opportunity to investigate the relationships between the structural properties of tested classes and their unit testability.

2.2 Independent variables

The independent variables consist of 8 size metrics, 24 cohesion metrics, 25 coupling metrics, 19 inheritance metrics, and 4 complexity metrics. They capture the most important structural properties of classes in object-oriented software, including size, cohesion, coupling, inheritance, and complexity. In [13], Briand et al. investigated a large number of size, cohesion, coupling, and inheritance metrics. We selected the same metrics for our study, since they are the main OO metrics developed during the period of 1990–1998. In particular, they have been extensively used in previous software engineering empirical studies [14–19]. In addition, we included many OO metrics developed more recently [20–26]. All the metrics are collected at the class-level.

2.3 Dependent variable

In this study, we use the size of the corresponding test class to indicate the effort involved in unit testing. We use dLOCC (lines of code for class) to measure the size of a test class. The “d” prefixed to the name of dLOCC denotes that it is the dependent variable in this study. dLOCC is the number of source lines of code of a test class, where we do not count either blank lines or lines containing only comments. dLOCC of a test class consists of two types of code: code initializing/releasing common object structures and code executing test cases and comparing actual to expected results. As stated in [1], dLOCC reflects the test suite size required to unit test a class. Indeed, this is the most commonly used proxy for unit testability previous literature [1, 6–9]. Note that dLOCC is very different from SLOC. In this study, a number of tested Java classes have associated test classes. SLOC is an independent variable, which counts source lines of code of tested classes. However, dLOCC is a dependent variable, which counts source lines of code of test classes.

2.4 Hypotheses

In this study, we aim to test the following hypotheses on the relationships between the metrics measuring structural properties and unit testability.

(1) H-Size (for size metrics): A large class is likely to require more unit testing effort than a small class. A large class contains more methods, parameters, or attributes than a small class. If a class contains more attributes, this may increase the complexity of both the code for initializing and releasing the test fixture and the code for examining whether a test case execution is successful. If a class contains more methods, unit testing may require more complex test. If methods in a class contain more parameters, this will increase the number of possible ways that methods can be invoked and consequently require more complex test. If a class contains more lines of code, it is likely to be more difficult to understand and test. Therefore, unit testing a large class is likely to require more effort.

(2) H-Cohesion (for cohesion metrics): A class with low cohesion is likely to require more unit testing effort than a class with high cohesion. Weak cohesion indicates that a class may serve several unrelated goals, which suggests inappropriate design. Encapsulating unrelated attributes/methods is likely to lead to more complex test. Therefore, more effort is likely to be required to unit test a class with low cohesion.

(3) H-Coupling (for coupling metrics): A class with high coupling is likely to require more unit testing effort than a class with a low coupling. A class having a high import coupling depends on many externally provided services. When unit testing such a class, many stubs have to be developed to emulate the behavior of any externally provided services that are not yet available. On the other hand, a class with

high export coupling has a large influence on the system: many other classes rely on it. Such a class will likely receive extra attention during testing and hence is also likely to require a larger test suite. Therefore, more effort is likely to be required to unit test a class with high coupling.

(4) H-Inheritance (for inheritance metrics): A class with many ancestors/many descendents/deep inheritance hierarchy/many overridden methods/many overloaded methods is likely to require more unit testing effort than a class with few ancestors/few descendents/shadow inheritance hierarchy/few overridden methods/few overloaded methods. If a class has more ancestors or is situated deep in an inheritance hierarchy, it will likely inherit more attributes/methods from its ancestors. More effort is hence likely to be required to cover the inherited attributes/methods and the code used to examine whether a test case execution is successful may also increase in complexity. If a class has more overridden methods, this may increase the number of interactions between inherited and overridden methods (i.e. polymorphism). More effort will likely be required to exercise such interactions. If a class has a greater number of overloaded methods, it is likely to be more difficult to understand and test. In addition, a class with more descendents has a large influence on the system: many other classes inherit its attributes and methods. Such a class will likely receive extra attention during testing and hence will likely require more complex test. Therefore, it is likely that more effort is required to unit test a class with many ancestors/many descendents/deep inheritance hierarchy/many overridden methods/many overloaded methods.

(5) H-Complexity (for complexity metrics): A highly complex class is likely to require more unit testing effort than a low complexity class. If a class is highly complex (for example, its methods have highly complex control flow structures and/or the interaction patterns between methods and attributes are highly complex), it is likely to require more effort. The code used to examine whether a test case execution is successful may also increase in complexity. Therefore, more effort is likely to be required to unit test a highly complex class.

2.5 Data collection procedure

We used the reverse engineering tool, Understand for Java[1], to collect the independent and dependent variables for this study. Based on Understand's Perl interface, we developed a Perl script to find tested classes that have associated test classes. Of the 1186 Java classes in Ant 1.7.0, 192 such classes were found. Of the 561 Java classes in JFreeChart 1.0.8, 306 such classes were found. We hence obtained 192 $\langle \mathrm{C}, \mathrm{T} \rangle$ pairs from Ant 1.7.0 and 306 $\langle \mathrm{C}, \mathrm{T} \rangle$ pairs from JFreeChart 1.0.8. Here, $\langle \mathrm{C} \rangle$ is a tested Java class and $\langle \mathrm{T} \rangle$ is its associated test class. For each pair $\langle \mathrm{C}, \mathrm{T} \rangle$, we used the Perl script to collect the independent variables from $\langle \mathrm{C} \rangle$ and the dependent variables from $\langle \mathrm{T} \rangle$ by accessing these two Understand databases. We exclude 20 pairs from the 192 $\langle \mathrm{C}, \mathrm{T} \rangle$ pairs for Ant 1.7.0 and 22 pairs from the 306 $\langle \mathrm{C}, \mathrm{T} \rangle$ pairs for JFreeChart 1.0.8. The reason is that their tested classes have undefined cohesion metric values. Consequently, we obtained 172 valid $\langle \mathrm{C}, \mathrm{T} \rangle$ pairs for Ant 1.7.0 and 284 valid $\langle \mathrm{C}, \mathrm{T} \rangle$ pairs for JFreeChart 1.0.8. In other words, we obtained 172 data points from Ant 1.7.0 and 284 data points from JFreeChart 1.0.8.

2.6 Data analysis procedure

At a high level, our analysis will proceed as follows:

(1) We first use univariate regression analysis to study the relationships between each of the investigated metrics and unit testing effort. The purpose of this is to identify metrics that are significantly related to unit testing effort and to identify potentially useful predictors. For all univariate analysis, we will examine the influential data points that could bias the result. We will use Cook's distance as a measure of the influence of a data point. In this study, a data point with a Cook's distance that is greater than 2 is regarded as an influential data point [17]. It is important to remove influential data points from further analysis, as we want our results to be stable.

(2) We then use multivariate analysis to study the relationships between the investigated metrics and unit testing effort when the metrics are used in a multivariate prediction model. We will use not only

[1] Available from http://www.scitools.com.

the most commonly used multiple linear regression but also a relatively new technique called partial least square regression to build multivariate models. We have three reasons for using multivariate analysis. First, we want to determine how well we can predict or estimate unit testing effort. Second, we want to identify metrics that play a more predominant or significant practical role in unit testing effort prediction. Third, we want to know whether PLSR is superior to MLR in building unit testing effort prediction models. All results will be examined for the presence of influential data points.

(3) We apply cross-validation to estimate the predictive performance of the multivariate models when they are applied to new data sets. We will use leave-one-out (LOO) cross-validation method. In an LOO cross-validation, a data set consisting of n observations is divided into n parts, each part being used to test the model built using the remainder of the data set.

3 Modeling techniques

In this section, we first introduce the two modeling techniques used in this study, multiple linear regression and orthogonal signal correction based partial least square regression, at a high level . Then, we describe the criteria for evaluating and comparing the prediction performance of these models.

3.1 Multiple linear regression

Multiple linear regression (MLR) is the most commonly used technique to model the linear relationship between a dependent variable y and k independent variable x_j $(j = 1, \dots, k)$. For a data set with n data points (i.e. sample size), the model can be written as

$$\boldsymbol{y} = \mathbf{1}b_0 + \boldsymbol{X}\boldsymbol{b} + \boldsymbol{e},$$

where $\mathbf{1}$ is an $n \times 1$ uniform column vector (i.e. each element is 1),

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad \boldsymbol{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1k} \\ x_{21} & x_{22} & \cdots & x_{2k} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nk} \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix}, \quad \text{and } \boldsymbol{e} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix}.$$

Here y_i $(i = 1, \dots, n)$ is the ith value of y, x_{ij} is the ith value of x_j, b_0 is the intercept of the regression model, b_j is the regression coefficient for x_j, and e_i is the ith random error term (assumed independent, with zero mean and common variance σ^2). MLR requires that (a) $n > k$; and (b) there are no collinearities between the independent variables.

In this study, we use SPSS 18.0[2] to build MLR models. When building an MLR model, we employ the most commonly used stepwise selection method to ensure that the model only includes important independent variables. At the same time, we use Cook's distance to exclude influential data points (whose presence or absence has a strong impact on the coefficients of the model) from the model fit. In our study, a data point with a Cook's distance that is greater than 2 is regarded as an influential data point [17]. Also, we use VIF to examine whether the model has a high multi-collinearity. In case several independent variables in the model have a VIF greater than 10, the one with the largest VIF will be removed [27]. This process is repeated until all the independent variables in the final model have a VIF below 10.

3.2 Partial least square regression

Partial least square regression (PLSR) is a generalization of MLR. It was first developed by Wold in the 1960s to address econometric path modeling [28] and was subsequently adopted in the 1980s for regression problems in chemometric and spectrometric modeling [29]. The advantage of PLSR is that it can analyze

[2] Available from http://www.spss.com.

high dimensional and noisy data where the number of independent variables is comparable to or larger than the number of data points and/or when the independent variables are highly collinear.

The underlying idea of PLSR is to successively extract a small number of components from the original set of independent variables under two conditions: (a) these successively extracted components have maximum covariance with the unexplained part of the dependent variable; and (b) these components are orthogonal, i.e. they are mutually independent linear combinations of the original set of independent variables. These components (also called factors or latent variables) then replace the original set of independent variables to build ordinary linear regression models for the dependent variable. Consequently, ordinary linear regression can still be performed even if the number of original independent variables is far larger than the number of data points and/or the original independent variables are highly collinear.

One important problem for PLSR is the determination of h, i.e. how to select the proper number of the PLS components. For a PLSR model, using more extracted components helps to improve the model fit to the training data set. However, using too many components may cause "over-fitting", i.e. the model fits the training data set very well but has little or no predictive power for unknown data set. In PLSR, the cross-validation coefficient of determination Q^2 (i.e. cross-validated R^2) is used to determine the number of PLS components that should be extracted from the data.

In particular, PLSR uses a metric called variable importance in the projection (VIP) to evaluate the influence on the dependent variable y of each independent variable x_j ($j = 1, \ldots, k$). Indeed, VIP quantifies the influence on the dependent variable of an independent variable summed over all components relative to the total sum of squares of the model. Since the average of squared VIP scores for all independent variable is equal to 1, the "greater than one rule" is the most commonly used variable selection criterion in practice [30].

Prior to generating the PLSR model, a multivariate calibration method called orthogonal signal correction (OSC) can be used to remove variation in the independent variables that is unrelated to the dependent variable [31]. In OSC, the removed part (i.e. OSC component) is mathematically orthogonal to the dependent variable. OSC cannot only improve the prediction performance [31] but also can simplify the structure and interpretation of the resulting PLSR model [32].

In this study, we use SIMCA-P+ 12.0.1[3] to build PLSR models. For a given data set, SIMCA-P+ can: (a) run the OSC algorithm to pre-process the data, where the number of OSC components can be specified by the users; (b) run the PLSR algorithm to build a PLSR model, where the number of PLS components is automatically determined using cross-validation; (c) compute the VIP values for individual independent variables. We use the following strategy to build the PLSR model on a data set D:

(1) Let $g = \min(k, 10)$, where k is the number of independent variables in D.

(2) Let $i = 1$, $j = 1$, and $MaxQ2 = 0$.

(3) Use the OSC algorithm with i OSC components to pre-process D. The resulting data set is denoted by D_i.

(4) Use the PLSR algorithm to build the PLSR model M_i based on D_i. If the Q^2 of M_i is larger than $MaxQ2$, then assign it to $MaxQ2$ and let $j = i$.

(5) If $i < g$, let $i = i + 1$ and return to step (3); otherwise, continue to step (6).

(6) Rebuild the PLSR model M_j with D_j on the condition that only those independent variables with a VIP value larger than 1 are used. Repeat this step until the Q^2 of M_j cannot be further improved.

3.3 Model evaluation criteria

For each model investigated on Ant 1.7.0, we will have 172 predicted values presenting the effort required to unit test classes. For each model investigated on JFreeChart 1.0.8, we will have 284 predicted values presenting the effort required to unit test classes. These will come from either the model fit or from a LOO cross-validation. We will assess the performance of the models from two points of view: effort prediction accuracy and class ranking capability.

(1) Effort prediction accuracy. We use the following criteria to evaluate the degree to which the effort that the model predicted matches the effort actually exerted:

[3] Available from http://www.umetrics.com.

- Predicted correlation coefficient (PR). PR is the Pearson correlation coefficient between the predicted effort and the actual effort. We use Williams's T2 test to determine whether two models are significantly different in terms of PR [33, 34].
- Absolute residual error (ARE). For a class i, let y_i be its actual effort and $\hat{y}_i$ the predicted effort from the prediction model. Then, $\mathrm{ARE}_i = |y_i - \hat{y}_i|$. We use both the paired t-test and the Wilcoxon signed-rank test to determine whether two models are significantly different in terms of their AREs.
- Magnitude of relative error (MRE). MRE is a normalized ARE. For a class i, $\mathrm{MRE}_i = |y_i - \hat{y}_i| / y_i$. We use both the paired t-test and the Wilcoxon signed-rank test to determine whether two models are significantly different in terms of MREs.
- Prediction at level q (PRED(q)). $\mathrm{PRED}(q) = k/n$, where n is the total number of classes in the system and k is the number of classes whose MRE is less than or equal to $q/100$. We use McNemar's test to determine whether two models are significantly different in terms of PRED(q) [35]. In this study, we let $q = 25$ and $q = 50$ because they are often used in previous literature [36].

(2) Class ranking capability. In practice, prediction models are also used rank classes. More specifically, classes are ranked from the highest to the lowest, according to the predicted effort required to unit test a class. In order to determine how well a prediction model ranks classes, we therefore introduce three reference models:

- Random model—in this model, the order of the classes is completely random. A comparison of the prediction model with the Random model indicates if following the prediction model is better than not employing any model at all.
- Best model—in this model, the order of classes is based on their actual unit testing effort (this is the theoretical best model). A comparison of the prediction model with the Best model gives an indication of how far the prediction model is from being perfect.
- Simple size model—a common belief is that unit testing a large class tends to require more effort. In practice, however, the Simple size model is often used because of its simplicity. A comparison of the prediction model with the Simple size model gives an indication of whether it is necessary to develop complex prediction models.

We use three complementary methods of comparison to evaluate the ranking capability of the unit testing effort prediction models. The first method is to use an Alberg diagram to plot the percentage of the cumulative unit testing effort against the percentage of classes that have to be analyzed when the classes are ranked in decreasing order according to the effort predicted by the models. The second method is to use both the paired t-test and the Wilcoxon signed-rank test to determine whether two prediction models are significantly different in terms of class ranking capability. The third method is to quantify the models' ranking performance. The ranking performance of a model related to a preferred cutoff percentile value c is defined as $\phi(c) = \hat{G}(c)/G(c)$ [37], where $\hat{G}(c)$ and $G(c)$ are respectively the percentages of the cumulative unit testing effort accounted for by classes above c in the rankings given by the prediction model and the Best model.

All the statistical tests used in this study are two-tailed. The level of statistical significance α provides an insight into the accuracy of testing. The selection of α may depend on the analyst and the property of the investigated problem. In this study, we set $\alpha = 0.05$.

4 Experimental results

In this section, we report the experimental results. In Subsection 4.1, we show the results of univariate linear regression analysis. In Subsection 4.2, we present the results of multivariate analysis using two regression methods, namely, multivariate linear regression and partial least square regression.

4.1 Univariate analysis

In Subsections 4.1.1–4.1.5, we report the results of univariate analysis for the size, cohesion, coupling, inheritance, and complexity metrics, respectively.

Table 1 Univariate analysis results for size metrics

Metric	Ant 1.7.0					JFreeChart 1.0.8				
	N	Coeff.	Std. err.	p-value	R^2	N	Coeff.	Std. err.	p-value	R^2
NAIMP	172	6.239	0.832	<0.001	0.249	284	4.409	0.546	<0.001	0.188
NMIMP	172	3.023	0.406	<0.001	0.246	284	2.694	0.199	<0.001	0.395
NMpub	172	3.144	0.531	<0.001	0.171	284	2.955	0.218	<0.001	0.395
NMNpub	172	7.762	1.158	<0.001	0.209	284	12.962	1.728	<0.001	0.166
NumPara	171	2.866	0.343	<0.001	0.295	284	2.014	0.155	<0.001	0.375
SLOC	172	0.281	0.029	<0.001	0.360	284	0.238	0.017	<0.001	0.412
Stmt	172	0.426	0.045	<0.001	0.347	284	0.355	0.025	<0.001	0.414
StmtExe	172	0.661	0.069	<0.001	0.353	284	0.540	0.029	<0.001	0.406

4.1.1 *Univariate analysis for size metrics*

In Tables 1–5, columns "N", "Coeff.", "Std. err.", "p-value", and "R^2" state for each metric the number of observations used for univariate analysis (after excluding influential data points), the estimated regression coefficient, the standard error, the p-value, and R^2.

Table 1 presents the results for the size metrics. We make the following observations:

(1) The coefficients of all size metrics are positive and very significant. The results strongly support the hypothesis H-Size that a large class is more likely to require more unit testing effort.

(2) All size metrics except NAIMP and NMNpub have a larger R^2 on JFreeChart 1.0.8 than on Ant 1.7.0.

(3) Metrics (SLOC, Stmt, and StmtExe) available only after implementation are better predictors than those available before implementation.

(4) SLOC and Stmt respectively have the largest R^2 on Apache Ant and JFreeChart 1.0.8. Our study shows that they are strong predictors for unit testing effort.

4.1.2 *Univariate analysis for cohesion metrics*

Table 2 presents the results for the cohesion metrics. We make the following observations:

(1) Many cohesion metrics have expected and significant associations with unit testing effort on both systems. As expected, the coefficients for the inverse cohesions LCOM1 to LCOM3 are positive and significant, those for the straight cohesion metrics Co′, Coh, CAMC, CAMCs, iCAMCs, SNHDs, and iSNHDs are negative and significant. However, the coefficients of Co and DCD are not significant. Overall, the results support the hypothesis H-Cohesion that a class with low cohesion is more likely to require more unit testing effort.

(2) Although the coefficients of ICH, NHD, NHDs, and iNHDs are very significant, the positive correlations between them and the dependent variable are not expected. As argued in [13], ICH is very likely not a cohesion metric because: (a) unlike most cohesion metrics, it is based on method invocation and pays no attention to class attributes; and (b) it is additive. NHD, NHDs, and iNHDs are based on the number of agreements between methods on usage of parameter types. Previous research has reported that NHD and its variants have a very strong negative correlation with CAMC [21]. This is further confirmed by our study: on Ant 1.7.0, the Spearman correlation coefficients of NHD, NHDs, iNHDs with CAMC are respectively −0.973, −0.942, and −0.883; on JFreeChart 1.0.8, the Spearman correlation coefficients of NHD, NHDs, iNHDs with CAMC are respectively −0.939, −0.919, and −0.888. Our empirical results indicate that NHD and its variants are very likely not cohesion metrics.

(3) LCOM2 has the largest R^2 on Ant 1.7.0, and ICH has the largest R^2 on JFreeChart 1.0.8.

4.1.3 *Univariate analysis for coupling metrics*

Table 3 presents the results for the coupling metrics. We make the following observations:

Table 2 Univariate analysis results for cohesion metrics

Metric	Ant 1.7.0					JFreeChart 1.0.8				
	N	Coeff.	Std. err.	p-value	R^2	N	Coeff.	Std. err.	p-value	R^2
LCOM1	170	0.131	0.017	<0.001	0.262	284	0.033	0.003	<0.001	0.309
LCOM2	169	0.173	0.020	<0.001	0.304	284	0.034	0.003	<0.001	0.300
LCOM3	172	5.362	0.888	<0.001	0.177	284	9.127	0.643	<0.001	0.416
LCOM4	172	6.956	2.411	0.004	0.047	284	3.268	3.150	0.300	0.004
Co	172	−6.882	33.081	0.835	<0.001	284	8.874	14.251	0.534	0.001
Co′	172	−154.095	36.447	<0.001	0.095	284	−88.172	25.206	0.001	0.042
LCOM5	172	157.492	42.578	<0.001	0.074	284	17.643	26.080	0.499	0.002
Coh	172	−190.341	37.674	<0.001	0.131	284	−110.00	31.354	0.001	0.042
TCC	172	−55.927	28.040	0.048	0.023	284	−14.224	18.762	0.499	0.002
LCC	172	2.620	24.654	0.915	<0.001	284	49.382	19.607	0.012	0.022
ICH	172	1.449	0.215	<0.001	0.211	284	1.883	0.129	<0.001	0.431
OCC	172	4.180	34.294	0.903	<0.001	284	56.146	20.747	0.007	0.025
PCC	172	-4.082	32.915	0.901	<0.001	284	71.537	19.524	<0.001	0.045
DC_D	172	−52.873	28.047	0.061	0.020	284	−12.471	18.755	0.507	0.002
DC_I	172	5.992	25.453	0.814	<0.001	284	58.919	19.553	0.003	0.031
CAMC	172	−237.093	49.783	<0.001	0.118	284	−321.244	65.989	<0.001	0.078
CAMCs	172	−214.075	38.911	<0.001	0.151	284	−280.872	45.442	<0.001	0.119
iCAMCs	172	−227.783	41.406	<0.001	0.151	284	−299.444	46.814	<0.001	0.127
NHD	172	236.501	42.137	<0.001	0.156	284	254.398	40.491	<0.001	0.123
NHDs	172	372.461	68.436	<0.001	0.148	284	300.087	51.997	<0.001	0.106
iNHDs	172	372.000	64.728	<0.001	0.163	284	328.168	49.435	<0.001	0.135
SNHD	172	−21.871	14.614	0.136	0.013	284	−38.209	10.813	<0.001	0.042
SNHDs	172	−55.638	25.372	0.030	0.028	284	−99.419	15.728	<0.001	0.124
iSNHDs	172	−49.185	22.814	0.032	0.027	284	−103.056	16.987	<0.001	0.115

(1) The coefficients of all coupling metrics are positive as expected and almost all of them are significant at $\alpha = 0.05$. The results strongly support the hypothesis H-Coupling that a class with high coupling is more likely to require more unit testing effort.

(2) All export coupling metrics predict unit testing effort on Ant 1.7.0 better than JFreeChart 1.0.8.

(3) All import coupling metrics predict unit testing effort on JFreeChart 1.0.8 better than on Ant 1.7.0.

(4) OMMEC and NIH-ICP respectively have the largest R^2 value on Ant 1.7.0 and JFreeChart 1.0.8.

4.1.4 *Univariate analysis for inheritance metrics*

Table 4 presents the results for the inheritance metrics. We make the following observations:

(1) On Ant 1.7.0, the coefficients of NOD, NMA, SPA, SPD, SP, DPD, and OVO are positive as expected and are significant at $\alpha = 0.05$. On JFreeChart 1.0.8, the coefficients of CLD, NMO, NMA, PII, SPD, DPA, DPD, DP, and OVO are positive as expected and are significant at $\alpha = 0.05$. In other cases, the coefficients of the inheritance metrics are not significant. Therefore, the results support the hypothesis H-Inheritance that a class with more ancestors/descendents, deeper inheritance hierarchy, or more overridden/overloaded methods is more likely to require more unit testing effort.

(2) On Ant 1.7.0, all inheritance metrics except NMA have a small R^2. On JFreeChart 1.0.8, all inheritance metrics except NMO, NMA, and OVO have a small R^2.

(3) On Ant 1.7.0, NMA has the largest R^2. On JFreeChart 1.0.8, OVO has the largest R^2.

Table 3 Univariate analysis results for coupling metrics

Metric	Ant 1.7.0					JFreeChart 1.0.8				
	N	Coeff.	Std. err.	p-value	R^2	N	Coeff.	Std. err.	p-value	R^2
CBO	172	3.435	0.581	<0.001	0.171	284	2.787	0.398	<0.001	0.148
RFC	172	0.077	0.039	0.049	0.023	284	0.141	0.046	0.003	0.032
RFC1	172	0.279	0.125	0.027	0.028	284	0.152	0.050	0.003	0.032
MPC	171	1.050	0.208	<0.001	0.131	283	1.329	0.127	<0.001	0.281
MPC′	172	0.193	0.037	<0.001	0.137	283	0.214	0.023	<0.001	0.231
DAC	172	19.409	4.710	<0.001	0.091	283	19.120	2.438	<0.001	0.180
DAC′	172	27.301	5.906	<0.001	0.112	284	33.838	4.486	<0.001	0.168
ICP	172	0.431	0.097	<0.001	0.104	283	0.633	0.060	<0.001	0.284
IH-ICP	172	1.478	0.311	<0.001	0.117	284	1.074	0.193	<0.001	0.099
NIH-ICP	172	0.460	0.124	<0.001	0.075	283	0.928	0.078	<0.001	0.333
ACMIC	172	33.324	28.465	0.243	0.008			N/A		
OCAIC	172	16.888	5.845	0.004	0.047	284	43.141	5.413	<0.001	0.184
OCAEC	172	5.246	0.995	<0.001	0.141	284	0.911	4.303	0.833	0.000
OCMIC	172	4.260	1.348	0.002	0.055	284	5.925	0.615	<0.001	0.248
OCMEC	172	1.283	0.374	0.001	0.065	284	0.687	0.408	0.094	0.010
AMMIC	172	2.398	0.532	<0.001	0.107	284	2.541	0.384	<0.001	0.134
DMMEC	171	2.034	0.690	0.004	0.049	282	0.521	0.229	0.024	0.018
OMMIC	172	1.054	0.282	<0.001	0.076	283	1.881	0.178	<0.001	0.285
OMMEC	170	0.998	0.162	<0.001	0.184	283	0.693	0.140	<0.001	0.080
CC	172	0.628	0.150	<0.001	0.094	284	0.956	0.089	<0.001	0.290
AMC	172	0.972	4.967	0.845	¡0.001	284	6.287	2.464	0.011	0.023
UCL	172	2.686	0.583	<0.001	0.111	284	2.665	0.226	<0.001	0.329

Table 4 Univariate analysis results for inheritance metrics

Metric	Ant 1.7.0					JFreeChart 1.0.8				
	N	Coeff.	Std. err.	p-value	R^2	N	Coeff.	Std. err.	p-value	R^2
DIT	172	−4.620	6.166	0.455	0.003	284	1.747	4.843	0.719	<0.001
CLD	172	19.708	10.979	0.074	0.019	284	26.960	9.976	0.007	0.025
NOC	172	−0.004	1.483	0.998	<0.001	284	4.377	3.301	0.186	0.006
NOD	169	20.514	5.787	0.001	0.070	283	1.193	1.960	0.543	0.001
NMO	172	−0.040	3.002	0.990	<0.001	284	12.048	2.083	<0.001	0.106
NMI	172	−0.149	0.211	0.480	0.003	284	0.015	0.067	0.227	<0.001
NMA	172	3.132	0.411	<0.001	0.255	284	2.687	0.209	<0.001	0.370
SIX	172	−36.804	60.983	0.547	0.002	284	14.999	28.743	0.602	0.001
PII	172	133.409	103.080	0.197	0.010	284	131.480	53.524	0.015	0.017
SPA	172	24.012	11.066	0.031	0.027	284	0.478	14.331	0.973	<0.001
SPD	171	26.109	9.410	0.006	0.044	284	18.236	8.256	0.028	0.017
SP	172	13.791	5.680	0.016	0.034	284	13.806	7.166	0.055	0.013
DPA	172	−0.409	2.704	0.880	¡0.001	284	6.738	1.504	<0.001	0.066
DPD	171	6.750	2.172	0.002	0.054	284	1.718	0.536	0.001	0.035
DP	172	1.701	1.204	0.160	0.012	284	2.300	0.500	<0.001	0.070
OVO	172	7.592	1.879	<0.001	0.088	282	10.352	0.725	<0.001	0.422

Table 5 Univariate analysis results for complexity metrics

Metric	Ant 1.7.0					JFreeChart 1.0.8				
	N	Coeff.	Std. err.	p-value	R^2	N	Coeff.	Std. err.	p-value	R^2
WMC	172	1.402	0.165	<0.001	0.299	284	1.094	0.078	<0.001	0.410
CDE	172	49.750	7.053	<0.001	0.226	284	50.526	5.739	<0.001	0.216
CIE	172	41.490	6.460	<0.001	0.195	284	35.842	4.728	<0.001	0.169
CCE	172	26.709	6.755	<0.001	0.084	284	16.783	3.533	<0.001	0.074

4.1.5 *Univariate analysis for complexity metrics*

Table 5 presents the results for the complexity metrics. We make the following observations:

(1) The coefficients of all complexity metrics are positive as expected and are very significant. The results strongly support the hypothesis H-Complexity that a class with high complexity is more likely to require more unit testing effort.

(2) The metric CDE available before implementation is a better predictor than the metrics CIE and CCE available only after implementation.

(3) WMC has the largest R^2 values (0.299 and 0.410, respectively) for predicting unit testing effort.

4.2 Multivariate analysis

We apply linear regression and partial least square regression techniques to build two multivariate regression models from those metrics that show a p-value below 0.05 in the univariate analysis. These models will allow us to: (a) investigate the feasibility of building prediction models using structural metrics and their relative performances; (b) identify those metrics which are practical significant in terms of unit testing effort prediction when used in combination; and (c) compare PLSR to MLR for modeling unit testing effort. In 4.2.1, we present the multivariate regression model, including MLR and PLSR models. In 4.2.2, we evaluate the prediction performance of these multivariate models.

4.2.1 *Model fit*

Table 6 (a) and (b) respectively present the results of applying MLR combined with stepwise variable selection method on the metrics available on Ant 1.7.0 and JFreeChart 1.0.8. In this table, column "Beta" states for each metric the standardized regression coefficient. The larger the absolute value of the Beta, the stronger the impact of the independent variable on the effort required to unit test a class. As can be seen, the MLR model on Ant 1.7.0 consists of six metrics: two size metric and four coupling metrics. The most important predictor in this model is StmtExe. The R^2 value is 0.540 and all the variables have a VIF smaller than 6. The MLR model on JFreeChart 1.0.8 consists of eleven metrics: one size metric, three cohesion metrics, five coupling metrics, and two inheritance metrics. The most important predictor in this model is OVO. The R^2 value is 0.646 and all the variables have a VIF smaller than 10. We can see that both MLR models include StmtExe and DAC.

Table 7 (a) and (b) respectively present the results of applying PLSR combined with VIP based variable selection method on the metrics available from Ant 1.7.0 and JFreeChart 1.0.8 that have been preprocessed by OSC. We used ten OSC components to remove variation in the structural metrics that is unrelated to the dependent variable on Ant 1.7.0 and nine OSC components on JFreeChart 1.0.8. As a result, we obtained two OSC-preprocessed data sets. We used the OSC-preprocessed data sets to build two PLSR models: one for Ant 1.7.0 and the other for JFreeChart 1.0.8, which are shown in Table 7 (a) and (b), respectively. As can be seen, the PLSR model on Ant 1.7.0 consists of eight OSC-preprocessed metrics: four size metrics, two cohesion metrics, one inheritance metric, and one complexity metric. The most important predictor in this model is the OSC-preprocessed LCOM2. The R^2 and Q^2 values of the model are respectively 0.731 and 0.716. The PLSR model on JFreeChart 1.0.8 includes only two OSC-preprocessed metrics: LCOM1 and LCOM2. The most important predictor is the OSC-preprocessed

Table 6 MLR models. (a) Ant 1.7.0; (b) JFreeChart 1.0.8

(a)

Metric	Coefficient	Std. err.	Beta	p-value
Constant	22.796	8.026		0.005
StmtExe	0.960	0.119	0.862	<0.001
NIHICP	−0.526	0.189	−0.313	0.006
OCAEC	3.882	0.762	0.277	<0.001
NMNpub	2.559	1.221	0.151	0.038
UCL	−2.673	1.002	−0.332	0.008
DAC′	12.5724	6.239	0.154	0.046

(b)

Metric	Coefficient	Std. err.	Beta	p-value
Constant	18.087	10.475		0.085
OVO	8.469	0.848	0.779	<0.001
ICH	1.077	0.222	0.376	<0.001
LCOM2	−0.034	0.006	−0.545	<0.001
AMC	11.687	2.202	0.279	<0.001
PCC	40.052	13.211	0.119	0.003
DAC	−3.958	1.915	−0.105	0.040
RFC	−0.134	0.048	−0.169	0.006
StmtExe	0.445	0.094	0.525	<0.001
OCAIC	−16.979	5.665	−0.169	0.003
DPD	−1.081	0.379	−0.118	0.005
IHICP	−0.481	0.227	−0.141	0.035

Table 7 PLSR models. (a) Ant 1.7.0; (b) JFreeChart 1.0.8

(a)

Metric	Coefficient	VIP
Constant	90.529	
LCOM2	0.112	1.134
LCOM1	0.099	1.130
NMA	0.922	0.971
SLOC	0.034	0.954
NMIMP	0.785	0.953
Stms	0.043	0.949
WMC	0.059	0.945
StmsExe	0.059	0.940

(b)

Metric	Coefficient	VIP
Constant	111.479	
LCOM1	0.040	1
LCOM2	0.043	0.999

LCOM1. The R^2 and Q^2 values of this model on JFreeChart 1.0.8 are respectively 0.752 and 0.747, which is better than those of the model on Ant 1.7.0.

Table 8 presents the goodness of fit for MLR and PLSR models. We provide the PR, mean/median ARE, mean/median MRE, PRED(25), and PRED(50) for MLR and PLSR models (indicated by the columns). We also provide the results of statistical tests for determining whether there is a significant difference between MLR and PLSR models on the same system (indicated by the column “p-value”). These statistical tests include the Williams’s T2 test for comparing PRs, the paired t-test for comparing mean AREs/MREs, the Wilcoxon signed-rank test for comparing median AREs/MREs, and the McNe-

Table 8 Good-of-fit: Comparison of MLR and PLSR models

		Ant 1.7.0			JFreeChart 1.0.8		
		MLR	PLSR	p-value	MLR	PLSR	p-value
PR		0.735	0.855	<0.001	0.812	0.867	<0.001
ARE	Mean	44.935	37.790	0.017	38.811	34.943	0.049
	Median	26.492	26.823	0.217	26.602	24.340	0.259
MRE	Mean	0.893	0.853	0.623	0.453	0.439	0.518
	Median	0.431	0.428	0.684	0.310	0.265	0.368
PRED	25	0.285	0.320	0.511	0.384	0.482	0.003
	50	0.541	0.547	1	0.757	0.732	0.381

Table 9 LOO cross-validation: Comparison of MLR and PLSR models

		Ant 1.7.0			JFreeChart 1.0.8		
		MLR	PLSR	p-value	MLR	PLSR	p-value
PR		0.672	0.845	<0.001	0.762	0.863	<0.001
ARE	Mean	48.494	38.946	0.005	41.757	35.275	0.004
	Median	26.767	27.400	0.176	27.078	24.432	0.044
MRE	Mean	0.931	0.870	0.491	0.480	0.442	0.091
	Median	0.455	0.439	0.604	0.320	0.266	0.111
PRED	25	0.273	0.308	0.551	0.373	0.475	0.002
	50	0.535	0.541	1	0.715	0.732	0.583

mar's test for comparing PREDs. A p-value below 0.05 indicates that there is a significant difference between the MLR and PLSR models and otherwise there is no significant difference between them. As can be seen, on both systems, the PLSR model is significantly better than the MLR model in terms of PR and mean ARE. However, there is no significant different between them in terms of median ARE, mean/median MRE, and PRED(50). The overall results suggest that the transition from MLR to PLSR cannot help to significantly improve model building.

4.2.2 *Model evaluation*

In this section, we investigate the prediction performance of PLSR and MLR multivariate prediction models based on LOO cross-validation. The main goal is to better understand the predictive power of these models when they are applied to data sets other than the one from which the models were derived.

Table 9 presents the results from LOO cross-validations of MLR and PLSR dLOCC models, which follows the same structure as Table 8. We make the following observations:

(1) For any model on Ant 1.7.0, the mean MRE is larger than 0.80, median MRE is larger than 0.40, and PRED(25) is less than 0.35. For any model on JFreeChart 1.0.8, the mean MRE is larger than 0.4, median MRE is larger than 0.25, and PRED(25) is less than 0.50;

(2) For the models on Ant 1.7.0, the transition from MLR to PLSR brings a significant improvement in terms of PR and mean ARE. For the models on JFreeChart 1.0.8, the transition from MLR to PLSR brings a significant improvement in terms of PR, mean/median ARE, and Pred(25). In addition, for all models, the transition from MLR to PLSR does not bring a significant improvement in MRE.

By the usual effort estimation standard, the prediction performance of a model is considered good if it has a Pred(25) equal to or larger than 0.75. From a practical perspective, the cross-validation results indicate that (a) the unit testing effort of a class cannot be accurately predicted based on structural metrics alone; and (b) the transition from MLR to PLSR does not significantly improve the prediction accuracy of the effort for unit testing a class.

We next analyze class ranking capability of MLR/PLSR models. Figure 1 (a) and (b) respectively

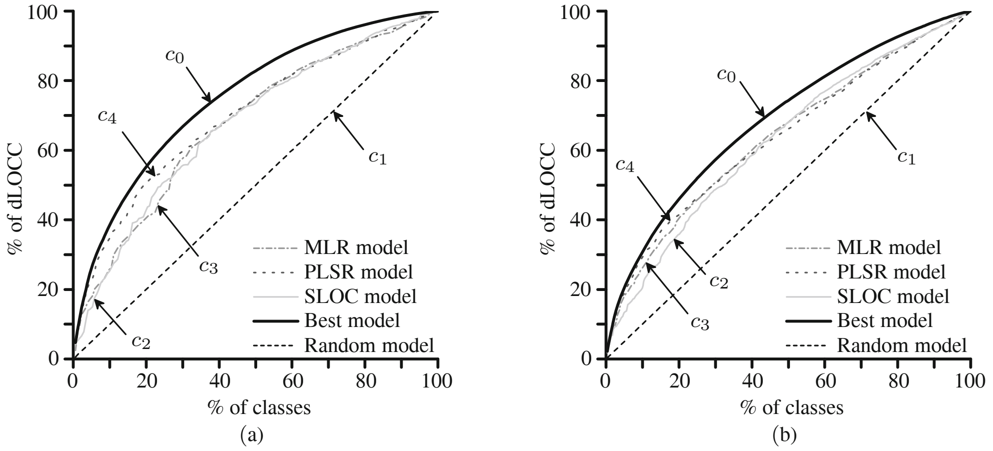

Figure 1 Alberg diagrams associated with MLR/PLSR models. (a) Ant 1.7.0; (b) JFreeChart 1.0.8.

shows the Alberg diagrams that compare the ranking capability of the MLR and PLSR models on Ant 1.7.0 and JFreeChart 1.0.8. On each Alberg diagram, we include three reference models: the Random, Best, and Simple size (SLOC) models. In this context, for the Best model, the order of classes is given by their actual dLOCC. We use SLOC as the simple size metric. The line c_0, c_1, c_2, c_3, and c_4 in Figure 1, respectively, represent the percentage of the cumulative dLOCC with respect to the class rankings produced by the Best model, the Random model, the Simple size model, the MLR model, and the PLSR model. We make the following observations:

(1) On each Alberg diagram, c_2, c_3, and c_4 are between c_0 and c_1. This indicates that all the MLR, PLSR, and Simple size models have a better class ranking capability than the Random model;

(2) In Figure 1(a), c_2 and c_3 interweave, while c_4 is closer to c_0 than c_2 and c_3 to c_0 in most cases. We performed both paired t-test and Wilcoxon signed-rank test between all pairs of data values in c_2 and c_3 that fall above the following representative cutoff percentile values: top 10 percent, top 20 percent, top 30 percent, and top 40 percent. The results show that c_3 is significantly better than c_2 for the cutoff point top 10 and 20 percents and there is no significant difference for the cutoff points top 30 and 40 percents. This implies that the Simple size model is better than the MLR model when ranking classes. When similar analysis was performed on c_4 and c_3 or c_4 and c_2, the p-values for all cutoff points are less than 0.05. This indicates that the PLSR model is significantly better than both the Simple size and MLR models when ranking classes.

(3) In Figure 1(b), c_4 is closer to c_0 than c_3 to c_0, while c_3 is closer to c_0 than c_2 to c_0 in top ranking. The results of both paired t-test and Wilcoxon signed-rank test on c_2, c_3 and c_4 show that for all the representative cutoff percentile values, the PLSR model is significantly better than the MLR model, which is significantly better than the Simple size model, when ranking classes.

Overall, the results indicate that (a) the MLR and PLSR models both offer a significant improvement of ranking capability over the Random model; (b) the MLR and PLSR models both offer a significant improvement of ranking capability over the Simple size model; and (c) PLSR models offer a significant improvement of ranking capability over MLR models.

Figure 2 (a) and (b) respectively visualize the ranking performance $\phi(c)$ of MLR and PLSR models relative to the Best model for the representative cutoff percentile values on Ant 1.7.0 and JFreeChart 1.0.8. From Figure 2 (a) and (b), we make the following observations:

(1) All the MLR/PLSR dLOCC models and the Simple size models show an improvement of 30% in $\phi(c)$ over the Random model when c varies between 5% and 40%;

(2) PLSR model shows an improvement in $\phi(c)$ over the MLR and Simple size (SLOC) models for all c values between 5% and 30%. In particular, the improvement decreases with the increase of c;

(3) The best model, i.e. PLSR dLOCC code model, shows a robust and high $\phi(c)$ ($> 85\%$) when c varies between 5% and 40%.

The overall results indicate that: (a) when used to rank classes, even the Simple size model has a large improvement in ranking performance over the Random model for the range of c of interest; (b)

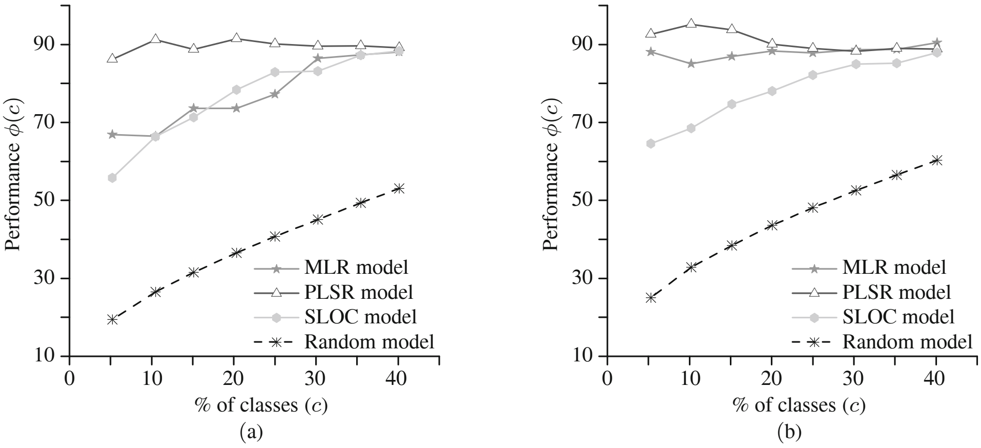

Figure 2 Ranking performance comparison of different prediction models from LOO CV. (a) Ant 1.7.0; (b) JFreeChart 1.0.8.

multivariate regression models, especially PLSR code models, can produce a high and robust class ranking performance, even though they are inaccurate quantitative models.

5 Conclusions and future work

Based on Ant 1.7.0 and JFreeChart 1.0.8, we employed multiple linear regression technique and partial least square regression technique to investigate the relationships between structural metrics and the required unit testing effort of classes. We analyzed 80 structural metrics, including size, cohesion, coupling, inheritance, and complexity metrics. We find that most of structural metrics are statistically related to unit testing effort of classes in an expected direction, among which size, complexity, and coupling metrics are the most important predictors. We also find that they are unable to accurately predict the required effort for unit testing a class. However, when used for unit testability ranking of classes, they have a high ranking capability. Furthermore, our results show that the transition from MLR to PLSR can significantly improve the capability for unit testability ranking of classes but not the prediction of unit testing effort of individual classes. In the future work, we will replicate this study to validate the findings using more projects and across testing frameworks. In addition, the usefulness of these metrics for unit testability prediction may depend on the programming language used. There is, therefore, also a need to validate our findings in other object-oriented programming languages such as C++ and C#.

Acknowledgements

This work was partially supported by National Natural Science Foundation of China (Grant Nos. 90818027, 91018005, 61272082, 61073029, 61170071, 61003020) and Hong Kong Competitive Earmarked Research Grant (Grant No. PolyU5225/08E).

References

1 Bruntink M, Deursen A. An empirical study into class testability. J Syst Software, 2006, 79: 1219–1232
2 Binder R V. Design for testability in object-oriented systems. Commun ACM, 1994, 37: 87–101
3 Baudry B, Traon Y. Measuring design testability of a UML class diagram. Inf Software Technol, 2005, 47: 859–879
4 Jungmayr S. Improving testability of object-oriented systems. PhD dissertation. der FernUniversität in Hagen, 2003
5 Jungmayr S. Testability measurement and software dependencies. In: Proceedings of the 12th International Workshop on Software Measurement, Magdeburg, 2002. 179–202
6 Bruntink M, Deursen A. Predicting class testability using object-oriented metrics. In: Proceedings of the 4th IEEE International Workshop on Source Code Analysis and Manipulation, Chicago, 2004. 136–145
7 Singh Y, Saha A. A metric-based approach to assess class testability. In: Proceedings of the 9th International Conference Agile Processes in Software Engineering and Extreme Programming, Limerick, 2008. 224–225

8 Badri L, Badri M, Toure F. Exploring empirically the relationship between lack of cohesion and testability in object-oriented systems. In: Proceedings of International Conference on Advances in Software Engineering, Jeju Island, 2010. 78–92

9 Badri L, Badri M, Toure F. An empirical analysis of lack of cohesion metrics for predicting testability of classes. Int J Softw Eng Appl, 2011, 2: 69–85

10 Mouchawrab S, Briand L C, Labiche Y. A measurement framework for object-oriented software testability. J Syst Software, 2005, 47: 979–997

11 ISO. International Standard ISO/IEC 9126, Information Technology—Software Product Evaluation: Quality Characteristics and Guidelines for Their Use, 1991

12 Wang H W. Partial Least Square Regression: Method and Applications (in Chinese). Beijing: National Defense Industry Press, 1999

13 Briand L C, Wüst J, Daly J W, et al. Exploring the relationships between design measures and software quality in object-oriented systems. J Syst Software, 2000, 51: 245–273

14 Basili V R, Briand L C, Melo W L. A validation of object-oriented design metrics as quality indicators. IEEE Trans Softw Eng, 1996, 22: 751–761

15 Chidamber S R, Darcy D P, Kemerer C F. Managerial use of metrics for object-oriented software: an exploratory analysis. IEEE Trans Softw Eng, 1998, 24: 629–639

16 Zhou Y, Leung H, Xu B. Examining the potentially confounding effect of class size on the associations between object-oriented metrics and change-proneness. IEEE Trans Softw Eng, 2009, 35: 607–623

17 Subramanyan R, Krisnan M S. Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects. IEEE Trans Softw Eng, 2003, 29: 297–310

18 Gyimóthy T, Ference R, Siket L. Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Trans Softw Eng, 2005, 31: 897–910

19 Koru A G, Tian J. Comparing high-change modules and modules with the highest measurement values in two large-scale open-source products. IEEE Trans Softw Eng, 2005, 31: 625–642

20 Aman H, Yamasaki K, Yamada H, et al. A proposal of class cohesion metrics using sizes of cohesive parts. In: Welzer T, Yamamoto S, Rozman I, eds. Knowledge-based Software Engineering. Amsterdam: IOS Press, 2002. 102–107

21 Counsell S, Swift S, Crampton J. The interpretation and utility of three cohesion metrics for object-oriented design. ACM Trans Softw Eng Methodol, 2006, 15: 123–149

22 Bansiya J, Etzkorn L, Davis C, et al. A class cohesion metric for object-oriented designs. J Object-Oriented Program, 1999, 11: 47–52

23 Miller B K, Hsia P, Kung D C. Object-oriented architecture measures. In: Proceedings of the 32nd Annual Hawaii International Conference on System Sciences, Hawaii, 1999. 1–18

24 Benlarbi S, Melo W L. Polymorphism measures for early risk prediction. In: Proceedings of the 21st International Conference on Software Engineering, Los Angeles, 1999. 334–344

25 Bansiya J, Davis C, Etzkorn L. An entropy-based complexity measure for object-oriented designs. Theory Pract Object Syst, 1999, 5: 111–118

26 Badri L, Badri M. A proposal of a new class cohesion criterion: An empirical study. J Object Technol, 2004, 3: 145–159

27 Belsley D, Kuh E, Welsch R. Regression Diagnostics: Identifying Influential Data and Sources of Collinearity. New York: John Wiley and Sons Inc., 1980

28 Wold H. Soft modeling: the basic design and some extensions. In: Systems Under Indirect Observation, Vols. I and II. Amsterdam: North-Holland, 1982. 1–54

29 Wold S, Sjöström M, Eriksson L. PLSR-regression: a basic tool of chemometrics. Chemometrics Intell Lab Syst, 2001, 58: 109–130

30 Chong I G, Jun C H. Performance of some variable selection methods when multicollinearity is present. Chemometrics Intell Lab Syst, 2005, 78: 103–112

31 Wold S, Antti H, Lindgren F, et al. Orthogonal signal correction of near-infrared spectra. Chemometrics Intell Lab Syst, 1998, 44: 175–185

32 Yu H, MacGregor J F. Post processing methods (PLS-CCA): simple alternatives to preprocessing methods (OSC-PLS). Chemometrics Intell Lab Syst, 2004, 73: 199–205

33 Williams E J. The comparison of regression variables. J R Stat Soc Ser B-Stat Methodol, 1959, 21: 396–399

34 Steiger J H. Tests for comparing elements of a correlation matrix. Psychol Bull, 1980, 87: 245–251

35 Mittas N, Angelis L. Comparing cost prediction models by resampling techniques. J Syst Softw, 2008, 81: 616–632

36 Lucia A D, Pompella E, Stefanucci S. Assessing effort estimation models for corrective maintenance through empirical studies. Inf Softw Technol, 2005, 47: 3–15

37 Khoshgoftaar T M, Liu Y, Seliya N. A multiobjective module-order model for software quality enhancement. IEEE Trans Evol Comput, 2004, 8: 593–608