Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems

Lionel C. Briand, Jürgen Wüst, John W. Daly[1], and D. Victor Porter[1]

Fraunhofer Institute for Experimental Software Engineering

Kaiserslautern, Germany.

Abstract

The first goal of this paper is to empirically explore the relationships between existing object-oriented coupling, cohesion, and inheritance measures and the probability of fault detection in system classes during testing. In other words, we wish to better understand the relationship between existing design measurement in OO systems and the quality of the software developed. The second goal is to propose an investigation and analysis strategy to make these kind of studies more repeatable and comparable, a problem which is pervasive in the literature on quality measurement. Results show that many of the measures capture similar dimensions in the data set, thus reflecting the fact that many of them are based on similar principles and hypotheses. However, it is shown that by using a subset of measures, accurate models can be built to predict which classes contain most of the existing faults. When predicting fault-prone classes, the best model shows a percentage of correct classifications higher than 80% and finds more than 90% of faulty classes. Besides the size of classes, the frequency of method invocations and the depth of inheritance hierarchies seem to be the main driving factors of fault proneness.

Keywords: coupling, cohesion, inheritance, object-oriented, metrics, measurement, empirical validation

1.0 Introduction

Many measures have been proposed in the literature to capture the structural quality of object-oriented (OO) code and design (Chidamber and Kemerer, 1991; Chidamber and Kemerer, 1994; Li and Henry, 1993; Lee et al., 1995; Briand et al., 1997b; Henderson-Sellers, 1996; Hitz and Montazeri, 1995; Bieman and Kang, 1995; Lake and Cook, 1994; Lorenz and Kidd, 1994; Tegarden et al., 1992). Such measures are aimed at providing ways of assessing the quality of software, for example, in the context of large scale software acquisition (Mayrand and Coallier, 1996). Such an assessment of design quality is objective, and the measurement can be automated. Once the necessary measurement instruments are in place, the assessment of even large software systems can be performed quickly, at a low cost, with little human involvement. But how do we know what measures actually capture important quality aspects? Despite numerous theories about what constitutes good OO design, only empirical studies of actual systems' structure and quality can provide tangible answers. Unfortunately, only a few studies have so far investigated the actual impact of these measures on quality attributes such as fault-proneness (Basili et al., 1996; Briand et al., 1997b; Cartwright and Shepperd, 1999), productivity or effort (Chidamber et al., 1998), or the amount of maintenance modifications (Li and Henry, 1993).

In this paper, we empirically investigate most of the measures proposed in the literature to date that capture structural aspects of OO designs. As far as we know, this is the first time such a comprehensive set of mea-

1. John Daly and Victor Porter were with Fraunhofer IESE when this research was performed. John Daly is now at Hewlett Packard Ltd, QA Dept, Queensferry Microwave Division, South Queensferry, Scotland, UK EH30 9TG.
e-mail: john_daly2@hp.com.
Victor Porter is now at IBM Perth, Level2, Pitheavlis, Perth, Scotland, PH2 0XB.
e-mail: victor_porter@uk.ibmmail.com

sures are investigated together. Based on data collected in an experiment in a university setting, we attempt to answer the following questions:

- Are existing OO design measures capturing different dimensions and structural aspects? If not, what are the underlying structural dimensions they actually capture?
- How are the measures related to the fault-proneness of classes? Which ones strongly affect fault-proneness?
- How accurate are the existing measures in predicting faulty classes? To what extent can they be used to drive code and design inspections?

Analyzing the structural dimensions covered by the design measures will shed some light on the amount of redundancy that is present among existing measures, and will help us to better interpret what individual measures are really capturing. By relating the measures to fault-proneness, we can identify the important drivers of fault-proneness, which are candidates to be used as quality benchmarks. Finally, we evaluate the accuracy of prediction models in the context of a realistic usage scenario, to demonstrate the potential of such models, and how they can be applied in practice.

Our answers to these questions are based on one data set only. Our study should be replicated in order to obtain generalizable results. In order to facilitate such replications, we propose here a precise, complete, and repeatable analysis procedure, which, when followed in a replicated study, will enable clear comparisons to be made across studies.

The results of this paper show that the number of dimensions actually measured is much lower than the number of measures themselves, despite their apparent differences. Some measures, in particular coupling and inheritance ones, are shown to be significantly related to the probability of detecting a fault in a class during testing. When combined, a subset of the measures enables the construction of a very accurate model to predict which classes most of the faults will lie in. Based on these results, it seems reasonable to claim that such a model could help focus testing and inspections resources on fault-prone parts of the design and code in a cost effective manner. For such a result to be achieved, it is important to note that organizations should collect enough data on their own software products to obtain representative distributions on which to perform the data analysis.

The paper is organized as follows: Section 2.0 describes the goals of the empirical study we are conducting, the hypotheses associated with the study, and the data collected. Section 3.0 describes the methodology used to analyze the data and build predictive models. The results of this analysis are then presented in Section 4.0. We draw our conclusions in Section 5.0.

2.0 The Empirical Study Design

In this section, we provide some background on the systems that are used in this study, the data collected, the dependent and independent variables, and the hypotheses we wish to investigate.

2.1 Description of the Empirical Study

The systems used for this study were developed by students participating in an upper division undergraduate/graduate level course offered by the Department of Computer Science at the University of Maryland. The objective of this class was to teach OO software analysis and design. The students were not required to have previous experience or training in the application domain or OO methods. All students had some experience with C or C++ programming and relational databases and therefore had the basic skills necessary for such a study. The systems were developed over a course of four months.

The students were grouped into eight teams of three students each. Each team was asked to develop a medium-sized management information system that supports the rental/return process of a hypothetical video rental business, and maintains customer and video databases. Such an application domain had the advantage of being easily comprehensible and, therefore, we could make sure that system requirements could be easily interpreted by students regardless of their educational background.

The development process was performed according to a sequential software engineering life-cycle model derived from the Waterfall model. This model includes the following phases: analysis, design, implementa-

tion, testing, and repair. At the end of each phase, a document was delivered: Analysis document, design document, code, error report, and finally, modified code, respectively. Requirement specifications and design documents were checked to verify that they matched the system requirements. Errors found in these first two phases were reported to the students. This maximized the chances that the implementation began with a correct OO analysis/design. Acceptance testing was performed by an independent group. During the repair phase, the students were asked to correct their system based on the errors found by the independent test group.

OMT, an OO Analysis/Design method, was used during the analysis and design phases (Rumbaugh et al., 1991). The C++ programming language, the GNU software development environment, and OSF/MOTIF were used during the implementation. Sparc Sun stations running on a UNIX operation system were used as the implementation platform. Although this is constantly changing, the development environment and technology we used are representative of what was used in industry and academia at that time. Our results are thus more likely to be generalizable to other development environments (external validity).

The following libraries were provided to the students:

1. MotifApp. This public domain library provides a set of C++ classes on top of OSF/MOTIF for manipulation of windows, dialogues, menus, etc. The MotifApp library provides a way to use the OSF/Motif widgets in an OO programming/design style.
2. GNU library. This public domain library is provided in the GNU C++ programming environment. It contains functions for manipulation of string, files, lists, etc.
3. C++ database library. This library provides a C++ implementation of multi-indexed B-Trees.

We also provided a specific domain application library in order to make our study more representative of industrial conditions. This library implemented the graphical user interface for insertion/removal of customers and was implemented in such a way that the main resources of the OSF/Motif widgets and MotifApp library were used. Therefore, this library contained a small part of the implementation required for the development of the rental system.

No special training was provided for the students to teach them how to use these libraries. However, a tutorial describing how to implement OSF/Motif applications was given to the students. In addition, a C++ programmer, familiar with OSF/Motif applications, was available to answer questions about the use of OSF/Motif widgets and the libraries. A hundred small programs exemplifying how to use OSF/Motif widgets were also provided. In addition, the complete documentation and, where available, the source code of the libraries were provided. Finally, it is important to note the students were not required to use the libraries and, depending on the particular design they adopted, different choices were expected.

The testing phase was accomplished by an independent group composed of experienced software professionals. This group tested all systems according to similar test plans and using functional testing techniques, spending eight hours testing each system.

The following relevant data items were collected:

1. the source code of the C++ programs delivered at the end of the implementation phase,
2. data about faults found by the independent testers during the testing phase.

The “M-System”, a measurement tool based on GEN++ (Devanbu, 1992), was developed at IESE to extract the values for the object-oriented design measures directly from the source code of the programs delivered at the end of the implementation phase. To collect item 2, fault report and component origination forms were used.

2.2 Dependent Variable

The goal of this study is to empirically investigate the relationships between object-oriented design measures and fault-proneness at the class level. We therefore need to select a suitable and practical measure of fault-proneness as the dependent variable for our study.

In this paper, fault-proneness is defined as the probability of detecting a fault in a class. As described in a section below, we chose a classification technique, called logistic regression, which is based on predicting event probabilities. In our instance, an event is a detection of a fault in a class during acceptance testing.

The probability of fault detection is described as a function of the structural properties of the classes. A rigorous definition, and examples to illustrate this concept, are given in Section 3.2, where we introduce logistic regression.

Clearly, other choices for the dependent variable could have been used (e.g., fault density) but, to a large extent, such a definition is driven by the choice of the modeling technique used for data analysis. Furthermore, the alternative choice of fault density as a dependent variable has its own problems. In (Rosenberg, 1998), it is shown that, even when there is no causal relationship between size and number of faults, a negative correlation between size and fault density can be found. This is, however, a pure mathematical artifact which makes any analysis of the impact of size on fault-proneness difficult.

Our dependent variable will be estimated based on fault data collected during testing and using maximum likelihood estimation in the context of logistic regression. Clearly, fault data may not be complete. But considering the systems used here are small, relatively simple, and thoroughly tested, we believe a very large proportion of faults has been detected. If this should not be the case, our analysis results should not show strong, consistent trends.

2.3 Independent Variables

The measures of coupling, cohesion and inheritance identified in a literature survey on object-oriented design measures (Briand et al., 1999; Briand et al., 1998a) are the independent variables used in this study. More specifically, we focus on measures defined at the class level, although there also exist measures defined for attributes, methods, and (sub)systems. Measures at the method and attribute level could not be validated, as the fault data we have only assigns faults to specific classes, but not to individual methods or attributes. Measures at the system level could not be validated because such measures provide only one data point per system. We have too few systems to be able to perform a valid statistical analysis for such measures.

We consider a total of 28 coupling measures, 10 cohesion measures, and 11 inheritance measures. In order to compare the relationship of these measures to size, we also investigate a small selection of size measures. The definitions of all measures used are summarized in Appendix A.

The eight systems under study consist of a total of 180 classes. Of these 180 classes, 67 were reused verbatim or with minor modifications (less than 25% of the code changed) from class libraries, the other 113 classes were developed from scratch or reused with extensive modifications (more than 25% of the code changed). The latter 113 classes will be referred to as ‘application classes’ or ‘non-library classes’ whereas the other 67 classes will collectively be called ‘library classes’. We collected the values for each of the measures presented above for the 113 application classes only.

At this stage, it is pertinent to consider the influence of library classes. Application classes can be coupled to library classes either through inheritance or by using some of their functionality. For coupling measures, a decision regarding whether or not to count coupling to library classes will have an impact on the computed measures’ values. We hypothesized that there would be a different effect on our dependent variable, if, for example, an application class is coupled to another application class rather than a library class, i.e., we hypothesized that a class is more likely to be fault prone if it is coupled to an application class than if it is coupled to a library class (although this may be dependent on the experience of the developer with the class library being used, or how well the library is documented). Consequently, the results for each coupling measure were calculated for each application class twice: counting coupling to other application classes only, and counting coupling to library classes only. As an example, consider Figure 1, where classes are represented as rectangles, the lines between classes denote coupling connections, and the classes are grouped according to library- and application classes. Class c is coupled to two other application classes and one library class. The measure CBO will yield 2 when counting coupling to application classes only, and 1 when counting coupling to library classes only. In this way, all coupling measures were calculated twice for the application classes, and analysis was then performed on both variants of the measures. The coupling measures are not calculated for the library classes.

Similarly, we could have distinguished ‘inheritance between application classes’ and ‘inheritance between application and library classes’ for the inheritance measures. However, as we will see later on, there are only very few inheritance relationships in the latter category. Measures focused on these inheritance relationships between application- and library classes have therefore little variance. Therefore the distinction was not

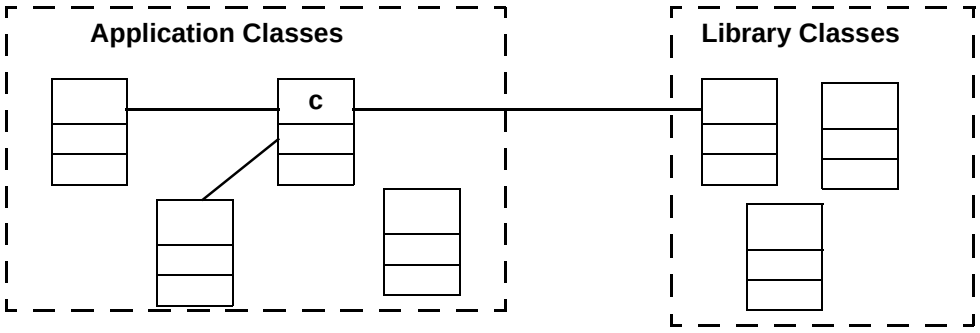

FIGURE 1. Coupling to library and application classes

made for inheritance measurement, the measures count inheritance relationships from application classes to both library classes and other application classes.

2.4 Hypotheses

In this study, we want to test the following hypotheses, which relate the design measures to fault-proneness.

H-IC (for all import coupling measures): A class with high import coupling is more likely to be fault-prone that a class with low import coupling. A class with high import coupling relies on many externally provided services. Understanding such a class requires knowledge of all these services. The more external services a class relies on, the larger the likelihood to misunderstand or misuse some of these services. Therefore, the class is more difficult to understand and develop, and thus likely to be more fault-prone.

H-EC (for export coupling measures): A class with high export coupling is more likely to be fault-prone than a class with low export coupling. A class with high export coupling has a large influence on the system: many other classes rely on it. Failures occurring in a system are therefore more likely to be traced back to a fault in the class, i.e., the class is more fault-prone.

H-COH (for cohesion measures): A class with low cohesion is more likely to be fault-prone than a class with high cohesion. Low cohesion indicates inappropriate design, which is likely to be more fault-prone.

H-Depth (measures DIT, AID): A class situated deeper in the inheritance hierarchy is more likely to be fault-prone than a class situated higher up (i.e., closer to the root) in the inheritance hierarchy. The deeper the class in the inheritance hierarchy, the less likely it is to consistently extend or specialize its ancestor classes and, therefore, the more likely it is to contain a fault.

H-Ancestors (measures NOA, NOP, NMI): A class with many ancestors is more likely to be fault-prone than a class with few ancestors. The more parents or ancestors a class inherits from, and the more methods a class inherits, the larger the context that is needed to know in order to understand what an object of that class represents, and therefore the more fault-prone the class is likely to be.

H-Descendents (measures NOC, NOD, CLD): A class with many descendents is more likely to be fault-prone than a class with few descendents. A class with many descendents has a large influence on the system, as all descendents rely on it. The class has to serve in many different contexts, and is therefore more likely to be fault-prone.

H-OVR (measures NMO, SIX): The more use of method overriding is being made, the more difficult/complex it is to understand or test the class, the more fault-prone it will be. Also, heavy use of method overriding indicates inappropriate design, which is more fault-prone.

H-SIZE (measure NMA and the size measures): The larger the class, the more fault-prone it is likely to be as it contains more information.

3.0 Data Analysis Methodology

In this section we describe the methodology used to analyze the coupling, cohesion, and inheritance measure data collected for the 113 system classes. In Section 3.1 we provide a description of the overall analysis

procedure and the techniques used. In Section 3.2 we provide a detailed description of our primary analysis technique, logistic regression.

3.1 Procedure for Data Analysis

The procedure used to analyze the data collected for each measure is described in four stages: (i) data distribution and outlier analyses, (ii) principal component analysis, (iii) prediction model construction, and (iv) correlation to size. This procedure is aimed at making the results of our study repeatable and comparable for future replications across different environments.

3.1.1 Data distribution and outlier analyses

Analysis of data distributions and outliers are performed through the following two steps:

1. The distribution and variance of each measure is examined. Measure that vary little do not differentiate classes very well and therefore are not likely to be useful predictors in our data set. Only measures with more than five non-zero data points were considered for all subsequent analyses.
   Analyzing and presenting the distribution of measures is important for comparison of the results with replicated studies. It allows researchers to determine if the data collected across studies stem from similar populations. If not, this information will likely be helpful to explain different findings across studies.
2. Identification of outliers. Outliers are data points which are located in an empty part of the sample space. Inclusion or exclusion of outliers can have a large influence on the analysis results and prediction models (influential outliers). It is important that conclusions drawn are not solely dependent on a few outlying observations. Otherwise, the resulting prediction models are unstable and cannot be reliably used. Again, when comparing results across replicated studies, it is particularly crucial to ensure that differences in observed trends are not due to singular, outlying data points. Thus, it is important to identify outliers, test their influence, explain them, and possibly remove them. We distinguish univariate and multivariate outliers:
   - Univariate outliers: A class that has an outlying value in the distribution of any one of the measures used in the study. The influence of the identified data point is tested: an outlier is *influential*, if the significance of the relationship between the measure and fault-proneness depends on the absence or presence of the outlier. Such influential outliers were removed for the results from univariate analysis.
   - Multivariate outliers: Our set of n independent variables spans an n-dimensional sample space. To identify multivariate outliers in this sample-space, we calculate for each data point the Mahalanobis Jackknife Distance from the sample space centroid. The Mahalanobis Distance is a distance measure which takes correlations between measures into account. Multivariate outliers are data points with a large distance from the centroid. Again, a multivariate outlier may be over-influential and therefore be removed, if the significance of any of the variables in the model depends on the absence or presence of the outlier.

   More detailed information on outlier analysis can be found in (Barnett and Price, 1995).

3.1.2 Principal component analysis

If a group of variables in a data set are strongly correlated, these variables are likely to measure the same underlying dimension (i.e., class property) of the object to be measured. Principal component analysis (PCA) is a standard technique to identify the underlying, orthogonal dimensions that explain relations between the variables in a data set.

Principal components (PCs) are linear combinations of the standardized independent variables. The sum of the squares of the coefficients of the standardized variables in one linear combination is equal to one. PCs are calculated as follows. The first PC is the linear combination of all standardized variables which explain a maximum amount of variance in the data set. The second and subsequent PCs are linear combinations of all standardized variables, where each new PC is orthogonal to all previously calculated PCs and captures a maximum variance under these conditions. Usually, only a subset of all variables have large coefficients - also called the loading of the variable - and therefore contribute significantly to the variance of each PC. The variables with high loadings help identify the dimension the PC is capturing but this usually requires some degree of interpretation.

In order to identify these variables, and interpret the PCs, we consider the *rotated* components. This is a technique where the PCs are subjected to an orthogonal rotation. As a result, the rotated components show a clearer pattern of loadings, where the variables either have a very low or high loading, thus showing either a negligible or a significant impact on the PC. There exist several strategies to perform such a rotation. We used the *varimax* rotation, which is the most frequently used strategy in the literature.

For a set of n measures there are, at most, n orthogonal PCs, which are calculated in decreasing order of variance they explain in the data set. Associated with each PC is its eigenvalue, which is a measure of the variance of the PC. Usually, only a subset of the PCs is selected for further analysis (interpretation, rotated components, etc.). A typical stopping rule that we also use in this study is that only PCs whose eigenvalue is larger than 1.0 are selected. See (Dunteman, 1989) for more details on PCA and rotated components.

We do not consider the PCs for use as independent variables in the prediction model. Although this is often done with least-square regression, in the context of logistic regression, this has shown to result in models with a sub-optimal goodness of fit (when compared to models built using the measures directly), and is not current practice. In addition, principal components are always specific to the particular data set on which they have been computed, and may not be representative of other data sets. A model built using principal components is likely not to be applicable across different systems.

Still, it is interesting to interpret the results from regression analyses (see next sections) in the light of the results from PCA, e.g., analyze from which PCs the measures that are found significant stem from. This shows which dimensions are the main drivers of fault-proneness and may help explain why this is the case.

Regarding replications of this study, it would be interesting to see which dimensions would also be observable in other systems, and find possible explanations for differences in the results. We would expect to see consistent trends across systems for the strong PCs which explain a large percentage of the data set variance, and can be readily interpreted.

3.1.3 Prediction model construction

Constructing the prediction model requires three steps: (i) univariate logistic regression, (ii) multivariate logistic regression, and (iii) model evaluation (i.e., goodness of fit). We describe these steps below.

(i) Univariate logistic regression is performed, for each individual measure (independent variable), against the dependent variable to determine if the measure is statistically related, in the expected direction, to fault-proneness. This analysis is conducted to test the hypotheses in Section 2.4.

(ii) Multivariate logistic regression is performed to build a prediction model for the fault-proneness of classes. This analysis is conducted to determine how well we can predict the fault-proneness of classes, when the measures are used in combination. For the selection of measures to be used in the model, a strategy must be employed that

- minimizes the number of independent variables in the model. Using too many independent variables can have the effect of increasing the estimated standard error of the model's prediction, making the model more dependent on the data set, i.e., less generalizable. A rule of thumb is to have at least ten data points per independent variable in the model.
- reduces multicollinearity, i.e., independent variables which are highly correlated. This makes the model more interpretable.

This study is exploratory in nature, that is, we do not have a strong theory that tells us which variables should be included in the prediction model. In this situation, a stepwise selection process can be used, where prediction models are built in a stepwise manner, at each step one variable entering or leaving the model.

The two major stepwise selection processes used in logistic regression are forward selection and backward elimination (Hosmer and Lemeshow, 1989). The general forward selection procedure starts with a model that includes the intercept only. Based on certain statistical criteria, variables are selected one at a time for inclusion in the model, until a stopping criteria is fulfilled. Similarly, the general backward elimination procedure starts with a model that includes all independent variables. Variables are selected one at a time to be deleted from the model, until a stopping criteria is fulfilled.

Because of the large number of independent variables used in this study, the initial model based on a backward selection process could not be fitted. Therefore, we opted for the forward selection procedure to build the prediction models[1]. In each step, all variables not already in the model are tested: the most significant variable is selected for inclusion in the model. If this causes a variable already in the model to become not significant (at α_{Exit}=0.10), it is deleted from the model. The process stops when adding the best variable no longer improves the model significantly (at α_{Enter}=0.05). The significance of a variable is tested by a loglikelihood ratio test, which will be explained in Section 3.2.

The significance levels to enter and exit the model (0.05 and 0.10, respectively) are stricter than those suggested in (Hosmer and Lemeshow, 1989). We made this choice because it is an indirect means to control the number of variables in the final model. A less stringent choice (e.g. 0.25 to enter the model as suggested in (Hosmer and Lemeshow, 1989)) resulted in models which violated the rule of thumb to have at least ten data points per independent variable.

Multivariate models should be tested for multicollinearity. The presence of multicollinearity makes the interpretation of the model difficult, as the impact of individual covariates on the dependent variable can no longer be judged independently from other covariates. In severe cases, multicollinearity results in inflated standard errors for the estimated coefficients, which renders predicted values of the model unreliable.

According to (Hosmer and Lemeshow, 1989), tests for multicollinearity used in least-squares regression are also applicable in the context of logistic regression. They recommend the test suggested by (Belsley et al., 1980), which is based on the *conditional number* of the correlation matrix of the covariates in the model. This conditional number can conveniently be defined in terms of the eigenvalues of principal components as introduced in Section 3.1.2:

Let $x_1,...,x_n$ be the covariates of our model. We perform a principal component analysis on these variables, and set λ_{max} to be the largest eigenvalue, λ_{min} the smallest eigenvalue of the principal components. The conditional number is then defined as $\sqrt{\lambda_{max}/\lambda_{min}}$. A large conditional number (i.e., discrepancy between minimum and maximum eigenvalue) indicates the presence of multicollinearity. A series of experiments showed that the degree of multicollinearity is harmful, and corrective actions should be taken, when the conditional number exceeds 30 (Belsley et al., 1980).

(iii) To evaluate the model's goodness of fit, we apply the prediction model to the classes of our data set from which we derived the model. A class is classified fault-prone, if its predicted probability to contain a fault is higher than a certain threshold p_0. We select the threshold p_0 such that the percentage of classes being classified fault-prone is roughly the same as the percentage of classes that actually are fault-prone. We then compare the predicted fault-proneness of classes to the actual fault-proneness. We use the following measures of the goodness of fit of the prediction model:

- Completeness:
  Assume we use the prediction model to select classes that are classified fault-prone for inspection. Further assume that inspections are 100% effective, i.e., all faults in a class are found during inspection. Completeness is then defined as the number of faults in classes classified fault-prone, divided by the total number of faults in the system. It is a measure of the percentage of faults that would have been found if we used the prediction model in the stated manner. Low completeness indicates that many faults are not detected. These faults would then slip to subsequent development phases, where they are more expensive to correct.
  Counting the percentage of faults found is more precise than counting the percentage of fault-prone classes found. It ensures that detecting a class containing many faults contributes more to completeness than detecting a class with only one fault. Of course, faults can be more or less severe (e.g., measured by the effort required to fix a fault). It would be desirable to also account for the severity of faults when measuring completeness. However, for this study we had no reliable data available concerning the severity of the faults.

1. We, however, also tried a backward selection process using Principal Component Analysis to pre-select variables (i.e., highest loading variables). This is a classical procedure recommended in many textbooks. But the results we obtained were not better than the ones we obtained with a forward selection process.

- Correctness:
  We can always increase the completeness of our prediction model by lowering the threshold p_0 used to classify classes as fault-prone ($\pi > p_0$). This causes more classes to be classified fault-prone, thus completeness increases. However, the number of classes that are incorrectly being classified as fault-prone also increases. It is therefore important to consider the correctness of the prediction model. Correctness is the number of classes correctly classified as fault-prone, divided by the total number of classes classified as fault-prone. Low correctness means that a high percentage of the classes being classified as fault-prone do not actually contain a fault. We want correctness to be high, as inspections of classes that do not contain faults is a waste of resources.
- Kappa:
  Kappa (Cohen, 1960) is a measures of the degree of agreement of two variables. Kappa values range between -1 and 1, the higher the value, the better the agreement. A Kappa of zero indicates that the agreement is no better than what can be expected from chance. Kappa can have a negative value if agreement is weaker than expected by chance, but this is rare. In our case, we use Kappa to measure the agreement between predicted and actual fault-proneness of a class. Applying Kappa in this context requires that the number of classes classified fault-prone, and the number of classes that actually are fault-prone are roughly the same. This is ensured be our choice of threshold p_0.

A fourth measure of the goodness of fit is the R^2 statistic. Unlike completeness, correctness, and Kappa, the definition of R^2 is specific to regression techniques based on maximum likelihood estimation, and will be explained in Section 3.2.

For replications of this study, it will be interesting to compare a number of aspects of the resulting prediction models: which measures (or rather, which structural dimensions discovered in PCA that the measures capture) are consistently included in the prediction models? And how does the goodness of fit of the models compare? Besides PCA, the results from univariate analysis will be helpful here to explain differences found across studies with respect to models' variables and goodness of fit.

3.1.4 Correlation to size

For each design measure, we analyze its relationship to the size of the class. This is to determine empirically whether the measure, even though it is declared as a coupling, cohesion, or inheritance measure, is essentially measuring size. This is important for several reasons. First, if a measure is strongly related to size, then it might shed light on its relationship with fault-proneness: it is clear that larger classes are more likely to contain faults. Recall that we are interested in increasing our understanding of OO design and code quality, independently of their size. Second, a model that systematically identifies bigger classes as more fault-prone is, a priori, less useful: the predicted fault-prone classes are likely to cover a larger part of the system, and inspection and testing efforts, for instance, could not be focused very well.

For the purpose of analyzing correlations with size, we select the size measures showing the strongest relationship to fault-proneness (see Section 4.3.4): measure *Stmts*, the number of it executable and declaration statements in a class. We then calculate the Spearman's rho coefficient between each coupling, cohesion, and inheritance measure and the selected size measure. A non-parametric measure of correlation was preferred, given the skewed distributions of the design measures that we usually observe.

3.2 Logistic Regression

Logistic regression is a standard technique based on maximum likelihood estimation. In the following, we give a short introduction to logistic regression, full details can be found in (Hosmer and Lemeshow, 1989) or (Khoshgoftaar and Allen, 1997).

A multivariate logistic regression model is based on the following equation (the univariate logistic regression model is a special case of this, where only one variable appears):

$$\pi(X_1, X_2, \ldots, X_n) = \frac{e^{(C_0 + C_1X_1 \ldots + C_nX_n)}}{1 + e^{(C_0 + C_1X_1 \ldots + C_nX_n)}} \qquad (*)$$

where π is the probability that a fault was found in a class during the validation phase, and the X_i's are the design measures included as independent variables in the model (called covariates of the logistic regression equation).

Our dependent variable, π, is a conditional probability: the probability that a fault is found in a class, as a function of the class' structural properties. Unlike with other regression techniques (e.g., linear regression, poisson regression), the dependent variable is not measured directly. To further illustrate this concept, we consider a simple example. We could have a prediction model for the fault-proneness of classes as a function of its depth in the inheritance tree (DIT). A result such as π(DIT=3)=0.4 could be interpreted as "there is a 40% probability that we detect a fault in a class with DIT=3", or "40% of all classes with DIT=3 contain a fault".

The curve between π and any single X_i - assuming that all other X_j's are constant - takes a flexible S shape which ranges between two extreme cases:

1. When a variable is not significant, then the curve approximates a horizontal line, i.e., π does not depend on X_i.
2. When a variable entirely differentiates fault-prone software parts, then the curve approximates a step function.

Such an S shape is perfectly suitable as long as the relationship between X_i's and π is monotonic, an assumption consistent with the empirical hypotheses to be tested in this study. Otherwise, higher degree terms have to be introduced in equation (*).

The coefficients C_i are estimated through the maximization of a likelihood function, built in the usual fashion, i.e., as the product of the probabilities of the single observations, which are functions of the covariates (whose values are known in the observations) and the coefficients (which are the unknowns). For mathematical convenience, *l = ln[L]*, the loglikelihood, is usually the function to be maximized. This procedure assumes that all observations are statistically independent. In our context, an observation is the (non) detection of a fault in a C++ class. Each (non) detection of a fault is assumed to be an event independent from other fault (non) detections. This is justified by the way we collect fault data, where each fault instance in a class comes from a distinct fault report form. Each data vector in the data set describes an observation and has the following components: an event occurrence (fault, no fault) and a set of OO design measurements (described in Section 2.3). In particular, if more than one fault is detected in a given class, each fault detection results in a separate observation in our data set. This is a direct consequence of our assumption that fault detections are independent events. Thus, the number of faults detected in a class is taken into account when we fit our models.

We also want to assess the impact of the independent variable on the dependent variable. In logistic regression, the regression coefficients C_i cannot be easily interpreted for this purpose. Instead, we use a measure $\Delta\psi$, which is based on the notion of odd ratio (Hosmer and Lemeshow, 1989). More specifically, the odds ratio $\psi(X)$ represents the ratio between the probability of having a fault and the probability of not having a fault when the value of the measure is X. As an example, if, for a given value X, $\psi(X)$ is 2, then it is twice as likely that the class does contain a fault than that it does not contain a fault. The value of $\Delta\psi$ is computed by means of the following formula:

$$\Delta\psi = \frac{\psi(X+\sigma)}{\psi(X)}$$

σ is the standard deviation of the measure. Therefore, $\Delta\psi$ represents the reduction/increase in the odds ratio when the value X increases by one standard deviation. This is designed to provide an intuitive insight into the impact of independent variables.

To assess the statistical significance of each independent variable in the model, a likelihood ratio chi-square test is used. Let *l = ln[L]* be the loglikelihood of the model given in equation (*), and l_i be the loglikelihood of the model without variable X_i. Assuming the null hypothesis that the true coefficient of X_i is zero, the statistic G=-2(l-l_i) follows a chi-square distribution with one degree of freedom (denoted by $\chi^2(1)$). We test p=P($\chi^2(1)$>G). If p is larger than some level of significance α (typically, α=0.05), the observed change in the

loglikelihood may well be due to chance, and X_i is not considered significant. If p<=α, X_i, the observed change in the loglikelihood is unlikely to be due to chance, and X_i is considered significant.

The global measure of goodness of fit we will use for such a model is assessed via R^2 which should not be confused with the least-square regression $\mathbf{R^2}$ - they are built upon very different formulae, even though they both range between 0 and 1 and are similar from an intuitive perspective. The higher R^2, the higher the effect of the model's explanatory variables, the more accurate the model. However, as opposed to the $\mathbf{R^2}$ of least-square regression, high R^2s are rare for logistic regression. For this reason, the reader should not interpret logistic regression R^2s using the usual heuristics for least-square regression $\mathbf{R^2}$s. (The interested reader may refer to (Menard, 1995) for a detailed discussion of this issue). Logistic regression R^2 is defined by the following ratio:

$$R^2 = \frac{LL_S - LL}{LL_S}$$

where:

- LL is the loglikelihood obtained by Maximum Likelihood Estimation of the model described in formula (*)
- LL_S is the loglikelihood obtained by Maximum Likelihood Estimation of a model without any variables, i.e., with only C_0. By carrying out all the calculations, it can be shown that LL_S is given by

$$LL_S = m_0 ln\left(\frac{m_0}{m_0 + m_1}\right) + m_1 ln\left(\frac{m_1}{m_0 + m_1}\right)$$

where m_0 (resp., m_1) represents the number of observations for which there are no faults (resp., there is a fault). Looking at the above formula, LL_S / (m_0 + m_1) may be interpreted as the uncertainty associated with the distribution of the dependent variable Y, according to Information Theory concepts. It is the uncertainty left when the variable-less model is used. Likewise, LL / (m_0 + m_1) may be interpreted as the uncertainty left when the model with the covariates is used. As a consequence, (LL_S - LL) / (m_0 + m_1) may be interpreted as the part of uncertainty that is explained by the model. Therefore, the ratio (LL_S - LL) / LL_S may be interpreted as the proportion of uncertainty explained by the model.

4.0 Analysis Results

This section presents the analysis results, following the procedure described in Section 3.0: descriptive statistics (Section 4.1), principal component analysis (Section 4.2), univariate regression analysis (Section 4.3), correlation to size (Section 4.4), and multivariate analysis (Section 4.5). In Section 4.6, we consider the threats to the validity of this study.

4.1 Descriptive Statistics

In sections 4.1.1 to 4.1.4, we discuss the descriptive statistics for the coupling, cohesion, inheritance, and size measures, respectively.

4.1.1 Descriptive Statistics for Coupling Measures

Table 1 presents the descriptive statistics for the coupling measures. Columns “Max.”,”75%”, “Med.”, “25%”, “Min.”, “Mean”, and “Std. Dev” state for each measure the maximum value, interquartile ranges, median, minimum, mean value, and standard deviation. Column “N>5” indicates if enough non zero data points are present to allow further analysis with that measure: “no” means the measure has fewer than six non-zero data points.

Since each coupling measure has been measured twice (once counting coupling to library classes only, once counting coupling to non-library classes only), we consider the two versions of each measure to be distinct

Measure	Coupling to non-library classes only								Coupling to library classes only							
	Max.	75%	Med.	25%	Min	Mean	Std. Dev.	N>5	Max.	75%	Med.	25%	Min	Mean	Std. Dev.	N>5
CBO	12	3	2	0	0	2.018	2.155		9	3	2	1	0	2.027	1.503	
CBO'	12	3	2	0	0	1.973	2.140		9	2	1	1	0	1.805	1.315	
RFC_1	86	24	13	8	2	16.938	13.716		130	14	5	2	0	14.708	21.482	
RFC_∞	138	26	13	8.5	2	19.513	20.531		182	26	10	2	0	19.265	25.592	
MPC	58	6	0	0	0	4.434	9.047		63	15	8	3.5	0	11.062	11.254	
ICP	97	9.5	0	0	0	7.478	15.128		166	42	20	7.5	0	29.646	30.851	
IH-ICP	12	0	0	0	0	0.372	1.956		10	0	0	0	0	0.522	1.643	
NIH-ICP	97	9	0	0	0	7.106	14.991		163	42	20	6.5	0	29.124	30.227	
DAC	10	1	0	0	0	0.858	1.569		4	0	0	0	0	0.363	0.768	
DAC'	6	1	0	0	0	0.593	0.960		3	0	0	0	0	0.310	0.642	
IFCAIC	2	0	0	0	0	0.071	0.320		0	0	0	0	0	0.000	0.000	no
ACAIC	3	0	0	0	0	0.027	0.282	no	1	0	0	0	0	0.044	0.207	no
OCAIC	8	1	0	0	0	0.726	1.390		3	0	0	0	0	0.310	0.656	
FCAEC	2	0	0	0	0	0.071	0.320		2	0	0	0	0	0.035	0.229	no
DCAEC	3	0	0	0	0	0.027	0.282	no	0	0	0	0	0	0.000	0.000	no
OCAEC	27	1	0	0	0	0.726	2.736		6	0	0	0	0	0.283	0.977	
IFCMIC	9	0	0	0	0	0.283	1.366	no	0	0	0	0	0	0.000	0.000	no
ACMIC	13	0	0	0	0	0.115	1.223	no	4	0	0	0	0	0.212	0.773	
OCMIC	38	2	0	0	0	2.425	5.540		2	0	0	0	0	0.354	0.706	
FCMEC	9	0	0	0	0	0.283	1.392	no	1	0	0	0	0	0.009	0.094	no
DCMEC	13	0	0	0	0	0.115	1.223	no	0	0	0	0	0	0.000	0.000	no
OCMEC	95	0	0	0	0	2.425	11.600		45	2	0	0	0	1.779	5.390	
IFMMIC	13	0	0	0	0	0.558	2.066		0	0	0	0	0	0.000	0.000	no
AMMIC	7	0	0	0	0	0.212	1.145		4	0	0	0	0	0.363	0.897	
OMMIC	56	3	0	0	0	3.637	8.365		60	15	8	3	0	10.699	10.806	
FMMEC	16	0	0	0	0	0.558	2.142		8	0	0	0	0	0.124	0.888	no
DMMEC	21	0	0	0	0	0.212	1.984		5	0	0	0	0	0.044	0.470	no
OMMEC	36	5	0	0	0	3.637	7.053		49	5	2	0	0	4.407	7.412	

Table 1: Descriptive statistics for coupling measures

measures. To distinguish them in the discussion, we denote the version counting coupling to library classes by appending an "_L" to its name. For example, MPC_L denotes the measure that counts invocations of methods from library classes, whereas MPC denotes the measure that counts invocations of methods from non-library classes.

From Table 1 we can make the following observations:

- The measures counting coupling between classes related through inheritance (all A**IC and D**EC measures, and NIH-ICP) have relatively low mean values and standard deviations. As we will see in Section 4.1.3, inheritance has not been used a lot in the systems under study, which explains why these coupling measures have low values and variance.
- For non-library data, the largest maximum value is for RFC_∞, which also has the largest mean and standard deviation. This may be explained by the fact that RFC_∞ is the only measure to count indirect coupling, whereas all other measures count connections to directly coupled classes only. For library data, ICP_L has the largest mean and maximum values, which should be due to the weighting of method invocations by the number of parameters of the invoked methods. For many library classes, the implementation is not available, therefore the indirectly invoked methods often cannot be determined, and RFC_∞_L is closer to RFC_1_L.

- There is evidence of export coupling from non-library classes to library classes, as OCAEC _L and some of the other export coupling measures have non-zero values. This stems from classes which were reused with minor modifications: in some instances, these classes actually use the non-library classes (which were developed from scratch or reused with extensive modifications).

The measures with little or no non-zero data cannot be used in the regression analyses. This is not to say that these measures are not useful in general. In a data set where the values of the measures have sufficient variance they may still have an impact on the dependent variable. If additional studies show they have no variance, then an investigation of why this is the case may be warranted. One outcome may be that such measurements are not practically useful under some specified circumstances.

4.1.2 Descriptive Statistics for Cohesion Measures

Table 2 presents the descriptive statistics for the cohesion measures, using the same form as Table 1. We make the following observations:

- The largest maximum value is for LCOM1 (819) which also has the largest mean (62.1) and standard deviation (115.2).
- LCOM3 and LCOM4 both count common attribute references. However, LCOM4 also counts method invocations between methods in a class (cf. definitions in Section 2.3). The values calculated for LCOM3 and LCOM4 are very similar, at least in the data set we used. The reason is that there are only few invocations of methods within a class. This can be seen from the low median and 75% quartile of ICH, which is a count of method invocations within a class.
- All measures have more than five non zero values and are therefore considered for further analysis.

Measure	Max.	75%	Median	25%	Min.	Mean	Std. Dev.
LCOM1	819	61.50	24.50	8.25	0	62.10	115.20
LCOM2	818	36.75	11	1	0	43.10	105.80
LCOM3	40	7	4	2.250	1	5.20	4.80
LCOM4	40	6.750	4	2	1	5.10	4.90
LCOM5	2	0.9118	0.7727	0.6000	0	0.78	0.28
Coh	0.82	0.4964	0.3140	0.1847	0	0.34	0.19
Co	1	0.4667	0.3333	0.1452	0	0.36	0.28
LCC	1	0.8533	0.5224	0.0085	0	0.50	0.38
TCC	1	0.6222	0.3571	0.0085	0	0.40	0.35
ICH	72	8	2	0	0	6.36	11.35

Table 2: Descriptive statistics for cohesion measures

4.1.3 Descriptive Statistics for Inheritance Measures

Table 3 presents the descriptive statistics for each inheritance measure. The following observations can be made from the table:

- Distributions of the values of the measures show that inheritance has been used sparingly within the eight systems (i.e. low mean values and median for DIT, NOC). Similar results have also been found in other studies (Chidamber et al., 1998; Chidamber and Kemerer, 1994; Cartwright and Shepperd, 1999). There is, however, sufficient variance in all the measures to proceed with the analysis.
- There is the presence of multiple inheritance indicated by the different values returned for DIT, AID, and NOA, which would otherwise be identical. However, NOP only takes values of 1 or 0 and should have values greater than this to indicate the presence of multiple inheritance. On further inspection of the systems, the class with more than one parent, i.e., NOP=2 in this instance, is a library class. Therefore, this class is not considered in Table 3, but affects the measurement values for some of the system classes.

Measure	Max.	75%	Median	25%	Min.	Mean	Std. Dev.
DIT	4	1	0	0	0	0.850	1.197
AID	4	1	0	0	0	0.845	1.193
CLD	2	0	0	0	0	0.079	0.331
NOC	5	0	0	0	0	0.177	0.770
NOP	1	1	0	0	0	0.416	0.495
NOD	9	0	0	0	0	0.230	1.102
NOA	4	1	0	0	0	0.867	1.228
NMO	10	1	0	0	0	0.610	1.566
NMI	104	11	0	0	0	6.97	16.13
NMA	41	14	9	6	1	11.38	7.89
SIX	0.52	0.03571	0	0	0	0.047	0.095

Table 3: Descriptive Statistics for Inheritance Measures

4.1.4 Descriptive Statistics for Size Measures

Table 4 presents the descriptive statistics for the size measures.

Measure	Max.	75%	Median	25%	Min.	Mean	Std. Dev.
Stmts	554	147	100	52	2	112.0531	84.50896
NAI	22	8	5	2	0	5.477876	4.385645
NM	46	16	11	7	2	13.39823	9.332681
NMpub	46	15	9	4	0	11.81416	10.16168
NMNpub	10	1	0	0	0	1.584071	2.757217
NumPara	117	13	9	4	0	11.000	13.01304

Table 4: Descriptive Statistics for Size Measures

There are classes having no public methods (NMNpub has zero minimum). These are classes used to store records of customers, rentals, tapes, and videos, and are accessed by friend classes.

There are relatively few private or protected methods, more than half of the classes having no such methods at all. This may reflect the lack of experience of the programmers involved in this study.

4.2 Principal component analysis

In this section, we present the results from the principal component analysis. All measures with sufficient variance (six or more non-zero data points) were subjected to an orthogonal rotation as described in Section 3.1.3. A total of 67 measures was used. This figure includes the size measures, and the library- and non-library classes variants of the coupling measures.

We identified 16 orthogonal dimensions spanned by the 67 measures, indicating that, as expected, there is a large amount of redundancy present among these measures. The 16 PCs capture 88.6% of the variance in the data set.

The loadings of each measure in each rotated component is given in Table 5. Values above 0.7 are set in boldface, these are the measures we call into play when we interpret the PCs. For each PC, we also provide its eigenvalue, the variance of the data set explained by the PC (in percent), and the cumulative variance in the table. Based on the analysis of the coefficients associated with the measure within each of the rotated components, the PCs are interpreted as follows.

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13
Eigenvalue	11.2	9.54	7.29	5.29	3.79	3.56	2.86	2.60	2.39	2.24	2.06	1.67	1.42
Percent	16.7	14.2	10.9	7.90	5.66	5.31	4.27	3.89	3.56	3.35	3.08	2.49	2.13
Cum.%	16.7	30.9	41.8	49.7	55.4	60.7	64.9	68.8	72.4	75.7	78.8	81.3	83.4
CBO	-0.03	0.67	0.03	0.02	-0.01	0.03	-0.18	-0.13	0.07	0.18	0.10	-0.01	0.01
CBO'	-0.03	0.67	0.03	0.04	-0.01	0.04	-0.19	-0.13	0.07	0.08	0.11	-0.01	0.01
RFC_1	0.05	**0.73**	0.49	-0.07	0.00	-0.02	-0.13	0.04	0.07	0.19	-0.01	0.01	0.05
RFC_∞	0.06	**0.81**	0.31	-0.10	-0.01	-0.05	-0.11	0.04	0.05	0.10	-0.03	0.01	0.05
MPC	0.04	**0.96**	0.00	0.01	-0.05	0.12	-0.08	-0.04	0.03	0.08	-0.05	0.01	0.02
ICP	0.02	**0.96**	0.01	0.00	-0.04	0.15	-0.12	-0.05	0.02	0.10	-0.05	0.02	-0.01
IH-ICP	-0.02	0.09	0.00	-0.08	-0.02	-0.05	0.08	-0.05	0.01	**0.96**	0.01	0.04	0.00
NIH-ICP	0.02	**0.96**	0.01	0.01	-0.04	0.16	-0.13	-0.05	0.02	-0.05	-0.05	0.01	-0.01
DAC	0.04	0.13	-0.02	0.00	0.04	**0.92**	-0.16	0.09	0.03	-0.03	-0.05	-0.16	0.03
DAC'	0.05	0.21	-0.01	0.08	-0.01	**0.83**	-0.14	0.10	0.03	-0.06	-0.11	-0.10	-0.06
IFCAIC	0.04	0.04	0.06	-0.08	0.03	0.28	**-0.74**	-0.04	0.01	0.03	-0.10	-0.33	0.05
OCAIC	0.03	0.13	-0.03	0.02	0.03	**0.94**	-0.01	0.11	0.03	-0.04	-0.03	-0.10	0.02
FCAEC	-0.10	0.01	0.05	-0.03	-0.01	-0.12	-0.02	0.02	-0.04	-0.01	**0.86**	-0.01	-0.06
OCAEC	-0.03	-0.05	0.10	-0.05	**0.80**	-0.13	0.02	0.08	-0.02	-0.03	0.10	0.02	-0.08
OCMIC	-0.09	0.28	0.02	0.28	0.01	0.50	0.05	-0.13	-0.01	0.02	-0.04	0.19	-0.34
OCMEC	-0.08	-0.03	0.04	0.04	**0.91**	0.07	0.04	-0.01	0.04	0.00	-0.08	0.02	0.03
IFMMIC	0.03	0.43	-0.01	-0.10	0.01	-0.08	**-0.78**	0.02	-0.01	-0.04	-0.10	-0.12	0.09
AMMIC	-0.02	0.09	0.01	-0.08	-0.02	-0.04	0.07	-0.05	0.03	**0.97**	-0.04	0.04	0.00
OMMIC	0.04	**0.92**	0.00	0.06	-0.05	0.16	0.14	-0.04	0.03	-0.07	-0.02	0.04	-0.01
FMMEC	-0.07	-0.09	0.15	-0.06	-0.03	-0.09	-0.11	-0.04	0.05	-0.02	**0.88**	0.11	0.03
DMMEC	-0.02	-0.01	0.06	0.03	0.00	-0.03	0.04	-0.04	-0.19	-0.01	0.17	0.10	-0.01
OMMEC	-0.05	0.05	0.17	0.14	0.03	-0.11	0.05	-0.07	0.10	0.01	0.04	-0.08	0.04
CBO_L	0.31	0.01	0.04	0.11	-0.07	0.12	**-0.76**	0.06	0.15	-0.08	0.07	-0.06	-0.36
CBO'_L	0.17	0.03	0.05	0.18	-0.05	0.11	**-0.83**	0.02	0.16	-0.07	0.09	-0.02	-0.17
RFC_1_L	0.61	-0.02	-0.01	-0.17	0.33	-0.01	-0.09	0.24	0.03	-0.07	-0.02	-0.11	-0.58
RFC_∞_L	0.57	0.22	-0.03	-0.20	0.28	0.09	-0.13	0.20	0.05	-0.04	-0.05	-0.20	-0.56
MPC_L	**0.88**	0.03	0.04	0.21	-0.07	0.03	-0.14	-0.05	0.05	-0.05	-0.02	-0.18	-0.17
ICP_L	**0.95**	0.04	0.06	0.04	-0.09	0.00	-0.07	-0.02	0.08	-0.03	-0.02	-0.14	-0.13
IH-ICP_L	0.19	-0.09	-0.01	0.00	-0.07	0.02	-0.08	0.17	0.02	-0.04	0.05	-0.08	**-0.91**
NIH-ICP_L	**0.96**	0.05	0.06	0.04	-0.08	0.00	-0.07	-0.02	0.08	-0.02	-0.03	-0.14	-0.09
DAC_L	0.35	-0.03	0.03	0.11	-0.05	0.14	-0.14	-0.03	-0.02	-0.03	-0.02	**-0.86**	-0.12
DAC'_L	0.26	-0.06	0.03	0.12	-0.06	0.15	-0.16	-0.03	-0.02	-0.03	-0.04	**-0.86**	-0.17
OCAIC_L	0.40	-0.04	-0.06	0.18	-0.04	0.11	-0.13	-0.05	0.00	-0.03	-0.01	**-0.79**	-0.09
OCAEC_L	-0.01	-0.12	0.19	-0.09	0.56	0.36	-0.09	-0.05	0.04	-0.02	0.27	0.21	0.07
ACMIC_L	-0.13	-0.13	-0.01	0.06	0.13	0.09	-0.02	**0.85**	0.00	-0.03	0.01	0.09	-0.25
OCMIC_L	-0.15	0.04	-0.09	0.36	-0.08	-0.11	-0.12	0.28	0.02	0.49	-0.11	-0.07	0.04
OCMEC_L	-0.13	-0.08	0.07	-0.02	**0.82**	0.05	0.08	-0.02	0.12	-0.05	-0.14	0.03	0.00
AMMIC_L	0.36	-0.09	-0.02	-0.08	-0.07	0.05	-0.07	0.16	0.03	-0.06	0.02	-0.15	**-0.85**
OMMIC_L	**0.88**	0.04	0.04	0.22	-0.07	0.03	-0.13	-0.06	0.04	-0.04	-0.02	-0.18	-0.11
OMMEC_L	0.16	-0.01	0.39	0.07	-0.01	0.25	-0.13	-0.21	0.08	-0.03	0.25	0.45	0.02
LCOM1	-0.04	0.01	**0.95**	-0.07	0.00	0.01	0.02	-0.06	-0.10	-0.06	0.07	0.00	0.06
LCOM2	-0.04	-0.03	**0.94**	-0.11	0.01	-0.01	0.04	-0.05	-0.15	-0.03	0.08	0.01	0.04
LCOM3	0.03	0.07	**0.91**	-0.23	0.15	-0.04	-0.02	0.04	-0.10	0.00	0.07	0.02	-0.05
LCOM4	0.03	0.02	**0.91**	-0.22	0.14	-0.01	-0.01	0.02	-0.11	0.01	0.08	0.04	-0.04
LCOM5	0.02	-0.04	0.00	**-0.85**	-0.03	-0.06	0.01	0.03	-0.23	-0.04	0.04	0.03	-0.03
Coh	-0.13	-0.05	-0.34	**0.73**	-0.02	-0.01	0.06	-0.07	0.02	0.04	-0.16	-0.01	0.09
Co	0.09	0.09	-0.23	**0.82**	-0.04	0.03	-0.06	0.03	0.08	-0.07	-0.05	-0.13	0.04
LCC	0.11	-0.06	0.03	**0.77**	-0.06	0.09	-0.07	-0.12	0.13	-0.10	0.09	-0.06	0.12
TCC	0.18	-0.01	-0.15	**0.85**	-0.03	0.03	-0.12	-0.03	0.05	-0.03	-0.02	-0.17	0.05
ICH	0.10	0.20	0.12	0.13	-0.05	0.00	**-0.79**	-0.07	0.03	-0.02	0.23	0.01	0.04
DIT	0.56	0.01	-0.08	-0.40	-0.05	0.02	0.02	0.28	0.12	0.19	-0.08	-0.19	-0.49
AID	0.56	0.01	-0.08	-0.40	-0.05	0.02	0.02	0.28	0.12	0.19	-0.08	-0.19	-0.49
CLD	-0.13	-0.07	0.26	-0.10	-0.06	-0.05	0.09	-0.04	**-0.90**	-0.02	0.03	-0.03	0.03
NOC	-0.12	-0.08	0.10	-0.14	-0.05	-0.03	0.08	-0.05	**-0.94**	-0.02	-0.02	-0.01	0.05
NOP	0.42	0.08	-0.09	-0.38	0.07	-0.05	0.05	0.44	0.16	0.27	-0.15	-0.18	-0.31
NOD	-0.10	-0.07	0.13	-0.14	-0.05	-0.03	0.07	-0.04	**-0.94**	-0.02	-0.04	-0.02	0.05
NOA	0.56	0.01	-0.08	-0.40	-0.05	0.02	0.02	0.28	0.12	0.19	-0.08	-0.19	-0.49
NMO	0.04	-0.07	-0.03	-0.03	0.04	0.09	0.00	**0.96**	0.03	-0.02	-0.01	0.02	-0.14
NMI	0.13	0.14	0.01	-0.27	0.45	-0.13	-0.05	0.33	0.03	0.14	-0.02	-0.02	-0.63
NMA	0.04	0.10	**0.84**	0.13	0.03	0.07	-0.05	-0.18	0.04	-0.09	0.12	0.01	0.05
SIX	0.25	-0.10	-0.09	-0.20	-0.12	0.10	0.05	**0.84**	0.07	-0.04	-0.04	-0.03	-0.17
Stmts	0.51	0.56	0.18	0.10	-0.03	0.04	-0.15	-0.10	0.08	-0.18	0.02	0.05	-0.03
NAI	0.12	-0.13	0.16	-0.14	-0.07	0.31	-0.02	-0.23	0.22	-0.20	0.43	-0.05	0.20
NM	0.03	0.40	0.68	-0.05	0.03	-0.08	-0.07	0.07	0.08	0.21	0.02	0.00	0.05
NMpub	-0.14	0.36	0.63	0.01	0.05	-0.10	-0.01	-0.03	0.06	0.21	0.02	-0.01	0.04
NMnpub	0.59	0.00	-0.01	-0.21	-0.10	0.10	-0.18	0.32	0.06	-0.09	-0.01	0.06	0.01
NumPara	0.17	0.07	**0.81**	0.01	-0.03	0.01	-0.23	-0.01	-0.20	-0.02	-0.14	-0.03	-0.02

Table 5: Rotated Components

- PC1:MPC_L, ICP_L, NIH_ICP_L, and OMMIC_L measure the extent of import coupling from library classes through method invocations.
- PC2: CBO, CBO', RFC_1, RFC_∞, MPC,ICP, NIH-ICP, and OMMIC. These measures count import coupling from non-library classes through method invocation.

  It is interesting to observe that import coupling from library classes through method invocations (PC1) and import coupling from non-library classes (PC2) are two orthogonal dimensions. This indicates that a class with high import coupling from library classes does not necessarily have particularly high or low import coupling from non-library classes; the two forms of coupling are unrelated in our data set.
- PC3: LCOM1, LCOM2, LCOM3, LCOM4, NMA, and NumPara. The LCOM cohesion measures count pairs of methods of classes within a class that use attributes in common. These are not normalized measured, i.e., they have no upper bounds. As noted above, the fact that, e.g., LCOM4 additionally accounts for method invocations does not significantly affect the distribution of the measure to create a separate dimension. Also present in this PC are two size measures, NMA and NumPara. This indicates that the LCOM measures have a correlation to size: the more methods (and therefore method parameters) in a class, the higher the number of pairs of methods with no common attribute usage is likely to be.
- PC4: LCOM5, Coh, Co, LCC, TCC. These are normalized cohesion measures, i.e., measures having upper and lower bounds. LCOM5 is an inverse cohesion measure and ranges between 0 (maximum cohesion) and 2 (minimum cohesion). The other measure are 'straight' cohesion measures where high values indicate high cohesion and vice versa. This explains why LCOM5 is the only contributing measure with negative loading.
- PC5: OCAEC, OCMEC, and OCMEC_L measure export coupling to classes not related via friendship or inheritance, by types of coupling other than method invocations.
- PC6: DAC, DAC', and OCAIC measure import coupling from non-library classes through aggregation.
- PC7: IFCAIC, IFMMIC, CBO_L, and CBO'_L, ICH. This PC cannot easily be interpreted. First, we have mostly coupling measures, and one measure ICH that was suggested as a cohesion measure. We will later see that ICH is not likely to be measuring cohesion. Rather, it possesses properties of a complexity measure. The fact that all these measure show up in the same PC means that there are classes with high import coupling from friend classes, which also tend to be coupled to a large number of library classes, and have high complexity. This is an interesting observation, but may be peculiar to the data set we used.
- PC8: ACMIC_L, NMO, SIX. NMO and SIX are inheritance measures which look at method overriding. We therefore interpret this PC to measure changes to a parent class in its child class. The fact that ACMIC_L is also present here may be due to chance. ACMIC_L measures a certain type of inheritance-based coupling. A prerequisite for all three measures to be different from zero is that the class is not at the root, which may explain the positive relationship between these measures.
- PC9: CLD, NOC, and NOD measure, for a class *the depth in the hierarchy* which is below the class. A class with a non-zero CLD has at least one child class (NOC) and is likely to have more descendents.
- PC10: IH-ICP and AMMIC measure import coupling through method invocations of non-library ancestor classes.
- PC11: FCAEC and FMMEC measure export coupling to friend classes.
- PC12: DAC_L, DAC'_L, and OCAIC_L measure import coupling from library classes through aggregation. Again, with PC8 we also have a corresponding dimension which counts aggregation coupling to non-library classes only.
- PC13: IH-ICP_L and AMMIC_L measures inheritance-based import coupling from library classes through library classes. Again, this is the library-coupling counterpart of PC10.
- There are three more PCs with eigenvalues>1. These PCs could not be meaningfully interpreted, and we therefore did not provide their coefficients in Table 5.

From these interpretations, we can draw a number of observations.

- Most of the dimensions identified are coupling dimensions (9 of the 13 interpreted PCs). One dimension is entirely determined by cohesion measures (PC4), one is entirely determined by inheritance measures (PC9), and another one (PC8) is also likely to be measuring an inheritance-related dimension. The dominance of coupling dimensions may in part be due to the larger number of coupling measures considered here. However, it also suggests that from all structural class properties considered here, coupling is the one that has received the most attention in its investigation so far in the literature.

- There is one dimension that may be interpreted as size: PC3.
- PCs that are mixtures of measures capturing different structural attributes are rare (PC3: size and cohesion measures, PC7: coupling and cohesion, PC8: (inheritance-based) coupling and inheritance). This shows that by and large, these measures actually do capture distinct dimensions.
- However, we also have a number of measures that cannot be assigned to any of the PCs at all, or are somewhat visible in more than one PC. This affects some of the coupling measures (especially CBO and CBO' which are measuring both import and export coupling, and RFC_1_L and RFC_∞_L, which are a cross between size and import coupling.
  With respect to inheritance measures, DIT, AID, and NOA are "shared" between PC1 (method invocations of library classes) and PC13 (inheritance-based method invocations of library classes).
  Interestingly, except for NMA and NumPara in PC3, none of the size measures is clearly visible in any particular PC. These measures have loadings around 0.5 in two or more PCs.
- Among the measures considered here are variants of earlier measures capturing the same concept (for instance, RFC_1 and RFC_∞ in PC2, LCOM1-LCOM4 in PC3, TCC and LCC in PC4, etc.). These variants were mostly defined with the intention to improve existing measures by eliminating problems that were identified based on theoretical considerations (see (Briand et al., 1999; Briand et al., 1998a) for a summary of these discussions). From a practical perspective, these differences in the definitions do not seem to matter much, because the variants lie within the same PCs as the original measure.

For the coupling measures, we further observe:

- Most of the PCs are made up either from measures counting coupling to library classes, or from measures counting coupling to non-library classes only. Only a few PCs are constituted by a mix of library and non-library coupling measures.
- Because we observe multiple orthogonal dimensions of coupling, we conclude that a thorough analysis of coupling in an OO system requires multiple coupling measures. Defining one measure that would combine the various dimensions of coupling is not likely to be suitable to completely describe class coupling in an OO system.
- The results also show that some coupling categories (or combinations of thereof) in the classification suggested in (Briand et al., 1997b) seem to correspond to actual coupling dimensions, i.e., they match with a PC. However, the original classification contained 18 classes whereas we have identified less orthogonal dimensions. Therefore, several refinements are not supported by empirical evidence, e.g., the different types of interdependency do not appear to capture different dimensions in the context of export coupling to friend classes, since their measure are in the same PC.

For the cohesion measures, we can say that normalization clearly makes a difference dimension-wise: the measures are clearly separated into normalized or non-normalized dimensions, i.e., no dimension includes both normalized and non-normalized measures. Although it is still unclear whether cohesion measures should be normalized to reflect variations in size across classes (Briand et al., 1996b), it is definitely a decision which has an impact on the structural property captured by the measures.

ICH is a special case. It differs from all other cohesion measures in two aspects. First, the type of intra-class-connection considered are method invocations only, paying no attention to class attributes. All other cohesion measures investigated here consider references to attributes in some way. Second, ICH also possesses a theoretical property that sets it apart from the other measures: ICH is additive. If two *unrelated*, but *highly cohesive* classes *c* and *d* are merged into a single class *e*, the cohesion of the class *e* would be the sum of the cohesion of the separate classes *c* and *d*. That is, class *e* has an even higher cohesion than any of the separate classes. This is counter-intuitive, as an object of class *e* should represent two separate, semantic concepts and therefore be less cohesive. It is therefore unlikely that ICH is a measure of cohesion. However, since it has been proposed as a cohesion measure in (Lee et al., 1995), we have considered it as a such here.

4.3 Univariate logistic regression

In this subsection, we investigate the relationship of the individual measures to fault-proneness. Coupling, cohesion, inheritance, and size measures are treated in separate subsections below.

4.3.1 Univariate Analysis - Coupling results

The results of the univariate analysis are summarized in Table 6. For each measure, the regression coefficient and standard error is provided (Columns “Coeff.” and “Std Err”), the R^2 and $\Delta\psi$ value (as defined in Section 3.2), and the statistical significance (p-value), which is the probability that the coefficient is different from zero by chance.

Measure	Coupling to Non-Library Classes					Coupling to Library Classes Only				
	Coeff.	Std Err	p	R^2	$\Delta\psi$	Coeff.	Std Err	p	R^2	$\Delta\psi$
CBO	0.325	0.080	<.0001	0.088	2.012	0.896	0.173	<.0001	0.121	3.844
CBO'	0.338	0.083	<.0001	0.091	2.062	0.868	0.198	<.0001	0.082	3.133
RFC_1	0.085	0.018	<.0001	0.132	3.208	0.048	0.012	<.0001	0.093	2.797
RFC_∞	0.102	0.018	<.0001	0.213	8.168	0.059	0.011	<.0001	0.160	4.503
MPC	0.182	0.043	<.0001	0.158	5.206	0.109	0.022	<.0001	0.137	3.417
ICP	0.135	0.028	<.0001	0.189	7.710	0.041	0.007	<.0001	0.159	3.511
IH-ICP	0.044	0.082	0.5898	0.001	1.090	0.403	0.205	0.0491	0.033	1.938
NIH-ICP	0.149	0.031	<.0001	0.196	9.272	0.041	0.007	<.0001	0.159	3.482
DAC	0.212	0.100	0.0329	0.020	1.395	0.464	0.224	0.0386	0.017	1.428
DAC'	0.339	0.164	0.0389	0.020	1.385	0.345	0.261	0.1867	0.006	1.248
IFCAIC	0.748	0.956	0.4334	0.003	1.270	fewer than six non-zero values				
ACAIC	fewer than six non-zero values					fewer than six non-zero values				
OCAIC	0.250	0.116	0.0307	0.021	1.416	0.511	0.258	0.0476	0.016	1.398
FCAEC	0.586	0.591	0.3213	0.004	1.206	fewer than six non-zero values				
DCAEC	fewer than six non-zero values					fewer than six non-zero values				
OCAEC	-0.492	0.166	0.0030	0.043	0.260	0.096	0.147	0.5131	0.001	1.098
IFCMIC	fewer than six non-zero values					fewer than six non-zero values				
ACMIC	fewer than six non-zero values					-0.319	0.218	0.1441	0.007	0.782
OCMIC	0.0226	0.026	0.3384	0.0028	1.133	0.070	0.193	0.7168	0.000	1.051
FCMEC	fewer than six non-zero values					fewer than six non-zero values				
DCMEC	fewer than six non-zero values					fewer than six non-zero values				
OCMEC	-0.017	0.015	0.2520	0.004	0.816	-0.035	0.032	0.2762	0.004	0.830
IFMMIC	0.220	0.131	0.0922	0.021	1.575	fewer than six non-zero values				
AMMIC	0.0567	9.134	0.6735	0.001	1.067	0.747	0.267	0.0051	0.045	1.954
OMMIC	0.191	0.047	<.0001	0.152	4.937	0.110	0.023	<.0001	0.133	3.276
FMMEC	0.134	0.079	0.0891	0.017	1.333	fewer than six non-zero values				
DMMEC	0.924	0.844	0.2737	0.015	1.214	fewer than six non-zero values				
OMMEC	0.023	0.017	0.1664	0.007	1.180	0.026	0.023	0.2616	0.005	1.209

Table 6: Univariate analysis with coupling measures

None of the univariate outliers was found to be influential. Looking at the import coupling measures, we note that most measures which display sufficient variance also have a significant relationship to fault-proneness. This provides strong support for hypothesis H-IC that classes with high import coupling are more fault prone. In particular, all the measures that are counts of method invocations (e.g., RFC*, *ICP, OMMIC) show very strong $\Delta\psi$ values, far above the ones of the other measures, for both library and non-library coupling. Since $\Delta\psi$ is corrected for the standard deviation of each measure, we may conclude that method invocation is the main coupling mechanism having an impact on fault proneness in the systems under study.

The measures IH-ICP, IFCAIC, ACMIC, and IFMMIC, as well as DAC’_L, ACMIC,_L and OCMIC_L appear not be related to fault-proneness. Although these measures fulfill our minimum variance criterion, their mean values and standard deviations are relatively low as compared to the significant measures. This may explain why we failed to find a significant relationship for these measures.

Turning to export coupling measures, we observe that only OCAEC is significant at α=0.05. However, the negative regression coefficient indicates that classes high export coupling are likely to be less fault-prone than classes with low export coupling. Overall, there is no evidence for hypotheses H-EC. Apparently, how much a class is being used by other classes has little effect on the probability for the class to contain a fault.

In (Briand et al., 1997b), more export coupling measures have been found significant. In that study, which uses the same C++ systems and the same fault-data, the distinction between coupling to library classes and non-library classes has not been made: both types of coupling were added up in one measure. This indicates that, at least in the data set used, export coupling to both types of classes has to be counted together to find a relationship to fault-proneness. Other differences with (Briand et al., 1997b) are mainly due to the refinement and debugging of the static analyzer used to capture the various coupling measures.

4.3.2 Univariate Analysis - Cohesion results

The results of the univariate analysis are summarized in Table 7 which is presented in the same form as Table 6.

- The coefficients for the inverse cohesion measures LCOM1 to LCOM5 are positive, those for the straight cohesion measures Coh, Co, LCC, and TCC are negative (though the coefficients for some measures are likely to be different from zero by chance). In each case, this indicates an increase in the predicted probability of fault-detection as the cohesion of the class (as measured by the measures) decreases. The only exception to this is ICH, where the positive coefficient suggests that fault-proneness increases with cohesion. Again, this indicates that ICH is very likely not a cohesion measure. In fact, ICH can be shown to fulfill the properties for complexity measures defined in (Briand et al., 1996b). We can then interpret this result that the higher the complexity of a class, the more fault-prone it is.
- Only the measures LCOM3, Coh, and ICH are significant at α=0.05. ICH is very significant (p=0.0002), but unlikely to measure cohesion. Overall, these results only weakly support our hypothesis H-COH, that highly cohesive classes are less fault-prone.
- Measures LCOM1, LCOM3, LCOM4, LCOM5, Coh, and ICH are all significant at α=0.25 and, as specified in Section 3.1.3, will be considered for building multivariate prediction models in the next section.
- Measure LCOM2 is the least significant measure. LCOM2 has been criticized not to discriminate classes very well (Basili et al., 1996; Henderson-Sellers, 1996). As a result, it is unlikely to be a useful predictor. Our findings provide some empirical evidence that supports this.

Measure	Coeff	Std. Err.	p-value	R^2	$\Delta\psi$
LCOM1	0.0025	0.0020	0.2135	0.006	1
LCOM2	0.0004	0.0017	0.7898	0.000	1
LCOM3	0.1474	0.0615	0.0164	0.025	2.054
LCOM4	0.0915	0.0542	0.0914	0.012	1.554
LCOM5	0.9803	0.6373	0.1240	0.008	1.315
Coh	-2.2314	0.7855	0.0045	0.028	0.654
Co	-0.5434	0.6463	0.4005	0.002	0.859
LCC	-0.2743	0.3947	0.4870	0.002	0.902
TCC	-0.2859	0.4445	0.5202	0.001	0.906
ICH	0.1129	0.0300	0.0002	0.072	3.485

Table 7: Univariate analysis for cohesion measures

4.3.3 Univariate Analysis - Inheritance results

Univariate analysis is performed on inheritance measures. The results from Table 8 show that:

- All the inheritance measures are significant at α=0.05.
- Hypothesis H-Depth for measures DIT and AID is supported. The deeper the class in the inheritance hierarchy, the higher its fault-proneness.

- Hypothesis H-Ancestors for measures NOA, NOP, NMI is supported. The more parents or ancestors a class inherits from, and the more methods a class inherits, the higher its fault-proneness. Because these measures together with DIT and AID are all in PC1, H-Depth and H-Ancestors are difficult to distinguish.
- Hypothesis H-Descendents (measures NOC, NOD, CLD): The results indicate that classes with a high number of children or descendents are less fault-prone. Perhaps, a greater attention (e.g., through inspections) is given to them as they are developed since many other classes depend on them. As a result, fewer defects are found during testing. However, the relatively low $\Delta\psi$ shows that the impact on fault-proneness is not so strong for these measures.
- Hypothesis H-OVR for measures NMO and SIX is supported. The more overriding methods in a class, the higher its fault-proneness.
- Hypothesis H-SIZE is empirically supported by NMA results. The more methods added to a class, the higher its fault-proneness.

Measure	Coeff	Std. Err.	p-value	R^2	$\Delta\psi$
DIT	0.6993	0.1614	0.0001	0.086	2.311
AID	0.6931	0.1640	0.0001	0.081	2.285
CLD	-2.276	0.8020	0.0045	0.042	0.470
NOC	-1.468	0.6663	0.0276	0.053	0.322
NOP	1.572	0.3272	0.0001	0.089	2.177
NOD	-1.393	0.6884	0.0429	0.053	0.215
NOA	0.6811	0.1540	0.0001	0.093	2.307
NMO	0.5144	0.2296	0.0243	0.024	1.948
NMI	0.0254	0.0122	0.0373	0.025	1.496
NMA	0.0681	0.0221	0.0021	0.039	1.710
SIX	6.886	2.632	0.0089	0.029	1.337

Table 8: Univariate analysis with inheritance measures

The results also show that the measures belonging to PC1 and PC3 show the strongest impact on fault proneness. The interpretation is that what matters the most in terms of class fault proneness is its depth in the hierarchy and the extent of change from his parent class(es).

One influential univariate outlier was detected for measures SIX and NMO: removal of the outlier causes SIX and NMO to become significant at α=0.05 in univariate analysis. This class was derived from an abstract library class. Some of its methods declarations and implementations were entirely generated by C++ macros provided by the library class. Thus, these methods were not implemented manually, which may explain why fewer faults were detected in this class.

4.3.4 Univariate Analysis - Size Results

The results from univariate analysis on the size measures is summarized in Table 9.

Measure	Coeff	Std. Err.	p-value	R^2	$\Delta\psi$
Stmts	.0189313	.0030775	<0.0001	0.2133	4.952
NAI	0.1011222	0.0357943	0.005	0.0300	1.558
NM	0.0756322	0.0205779	<0.0001	0.0623	2.026
NMpub	0.0396085	0.145494	0.006	0.0288	1.496
NMNpub	0.1245331	0.0560172	0.026	0.0201	1.407
NumPara	0.06352	0.0207718	0.002	0.0362	2.286

Table 9: Univariate analysis with size measures

All measures are significant, with positive coefficients, supporting our hypothesis H-SIZE that larger classes are more likely to be fault-prone. Measure *Stmts*, which is a count of the executable and declaration statements and therefore available only after implementation, has the biggest impact on fault-proneness ($\Delta\psi$=4.952). The impact of the other measures, which rely on information available from the class interface, is weaker.

One univariate outlier for measure *NumPara* was removed. When the outlier is included, the measure is no longer significant at α=0.05. This outlier (the class with 113 parameters) consists of a number of methods with empty implementation bodies. This was dead code the developers forgot to delete.

4.4 Correlation with size

In this section we analyze the correlation of the coupling, cohesion, and inheritance measures to the size of the class. We use *Stmts*, the number of declaration and executable statements in the class, to measure size, as this is the size measure showing the strongest relationship with fault-proneness.

In Table 10, we indicate for each coupling measure the Spearman's Rho coefficient (r) and the corresponding p-value. Spearman's rho is well below 0.5 for most of these measures, which indicates that the correlation to size is at best moderate.

Measure	Coupling to Non-Library Classes only		Coupling to Library Classes only	
	rho	p-value	rho	p-value
CBO	0.3217	<.0001	0.3639	<.0001
CBO'	0.3359	<.0001	0.3550	<.0001
RFC_1	0.3940	<.0001	0.3125	<.0001
RFC_∞	0.4310	<.0001	0.3431	<.0001
MPC	0.3232	<.0001	0.5180	<.0001
ICP	0.3168	<.0001	0.4675	<.0001
IH-ICP	-0.1240	0.1082	0.1790	0.0176
NIH-ICP	0.3455	<.0001	0.4672	<.0001
DAC	0.1753	0.0163	0.1514	0.0443
DAC'	0.1958	0.0088	0.1370	0.0710
IFCAIC	0.1557	0.0438	n.a.	
ACAIC	-0.0167	0.8301	-0.0022	0.9777
OCAIC	0.1294	0.0785	0.1771	0.0194
FCAEC	0.0297	0.7010	0.0588	0.4478
DCAEC	-0.0429	0.5810	n.a.	
OCAEC	0.0765	0.3058	0.0922	0.2266
IFCMIC	0.0935	0.2240	n.a.	
ACMIC	-0.0167	0.8301	-0.1432	0.0629
OCMIC	0.0493	0.4913	-0.1082	0.1533
FCMEC	0.0484	0.5299	0.0548	0.4807
DCMEC	-0.0429	0.5810	n.a.	
OCMEC	-0.0855	0.2528	-0.0294	0.6884
IFMMIC	0.2365	0.0019	n.a.	
AMMIC	-0.1229	0.1115	0.1783	0.0181
OMMIC	0.2765	0.0001	0.5243	<.0001
FMMEC	-0.0057	0.9407	0.0566	0.4640
DMMEC	-0.0345	0.6553	-0.0167	0.8301
OMMEC	0.1732	0.0136	0.2680	<.0001

Table 10: Correlation of coupling measures to size

Table 11 shows Spearman's rho for the relationship between each cohesion measure and size. We see that none of the cohesion measures has a strong relationship to size. Even ICH, which possesses the additive property of a size measure, is not strongly correlated to size. However, it has the highest rho coefficient.

Measure	rho	p-value
LCOM1	0.270	<.001
LCOM2	0.134	0.042
LCOM3	0.209	0.002
LCOM4	0.173	0.010
LCOM5	-0.096	0.135
Coh	-0.077	0.231
Co	0.242	<.001
LCC	0.252	<.001
TCC	0.252	<.001
ICH	0.439	<.001

Table 11: Correlation of cohesion measures to size

From Table 12 we see that none of the inheritance measures has a strong correlation to size. NMA has the strongest correlation to size. NMA is the number of methods added to a class and a correlation to size is therefore to be expected.

Measure	rho	p-value
DIT	0.089	0.225
AID	0.088	0.229
CLD	-0.171	0.026
NOC	-0.170	0.027
NOP	0.054	0.490
NOD	-0.170	0.027
NOA	0.099	0.180
NMO	0.048	0.521
NMI	0.056	0.442
NMA	0.397	<.0001
SIX	0.021	0.774

Table 12: Correlation of Inheritance Measures to size

4.5 Multivariate Logistic Regression Model

We will now investigate a number of multivariate prediction models, built from different subsets of the measures we have analyzed so far. The main goals of these models is to build accurate prediction models for class fault-proneness by using all the design measures available together. We evaluate the accuracy of the best prediction model we found using a 10-cross-validation process and discuss possible applications of such a model.

4.5.1 Comparing Multivariate Models

In this section, we compare models built from size measures only, from coupling, cohesion, and inheritance design measures only, and one allowing all measures (design coupling, cohesion, inheritance, and size) to enter the model. With these models, we seek to find answers to the following questions:

- Are coupling, cohesion, and inheritance design measures fault-proneness predictors that are complementary to design size measures?

- How much more accurate is a model including the more difficult to collect coupling, cohesion, and inheritance measures? If it is not significantly better, then the additional effort of calculating these more expensive measures instead of some easily collected size measures would not be justified.

Design Size Model

Below is the model that resulted from running a stepwise selection process as described in Section 3.1.3, allowing only the design size measures to enter the model.

Measure	Coeff.	Std. Error	p
NM	.4493462	.0933279	0.000
NMpub	-.3145506	.0772522	0.000
NumPara	-.0598625	.0220541	0.007
Intercept	-.3882196	.3368118	0.249

Table 13: Model I - Size model

We will refer to this model as “Model I”. The model has three covariates, the loglikelihood of the model is -127.94, and R^2 of 0.1399 is very low, even lower than the R^2 of measure *Stmts* in univariate analysis (0.2133). Model I has one multivariate outlier, which is not influential and therefore was retained for the final model fitting. The conditional number for this model is $\sqrt{2.2396 / 0.0300} \approx 8.6381$, well below the critical threshold of 30. The signs of the coefficients of measures NMpub and NumPara are negative, though they were positive in univariate analysis. This is due to suppressor relationships between the variables, which are commonly observed in multivariate regression analysis (Darlington, 1968).

We applied this model to the 113 classes of the LALO system in order to compare the predicted and actual fault-proneness of the classes. A class was classified “predicted fault-prone”, if its predicted probability to contain a fault is higher than 0.75. This threshold was selected to roughly balance the number of actual and predicted fault-prone classes. The contingency table below summarizes these results:

		predicted		Σ
		π<=0.75	π>0.75	
actual	no fault	39 classes	21 classes	60 classes
	fault	21 classes, 76 faults	32 classes, 158 faults	53 classes 234 faults
Σ		60 classes	53 classes	113 classes

Table 14: Goodness of fit of model I

The model identifies 53 classes as fault-prone, 32 of which actually are fault-prone (60% correctness) and contain a total of 158 of all 234 faults (67% completeness). The Kappa value for the degree of agreement between actual and predicted fault-proneness is 0.25, which is rather low. If we used the predictions of this model to select classes for inspection, the low correctness of the model implies that a large number of classes (21) that do not contain any fault would have been inspected in vain.

Model built from coupling, cohesion, and inheritance measures

Next, we build a model allowing all coupling, cohesion, and inheritance measures to enter the model, following the forward selection process. Concerning the two versions of coupling measures (coupling to library and non-library classes) calculated for each system class, we again treat them as separate measures, and allow them individually to enter or exit the model. Since, as we have seen, coupling to library- and non-library classes are orthogonal dimensions related to fault-proneness, this will yield more accurate prediction models than a model built of measures which do not distinguish coupling from library and non-library classes.

This model (Model II) consists of seven measures: four coupling measures, three inheritance measures. Non of the cohesion measures was included, reflecting the results found in univariate analysis that many of these measures are not significant indicators of fault-proneness. There is one multivariate outlier with respect to

Measure	Coeff.	Std. Error	p
RFC_{∞}	.1748663	.0274879	0.000
NOP	3.152051	.6414347	0.000
RFC_1_L	.2206044	.0461345	0.000
NMI	-.2975254	.0549887	0.000
FMMEC	.5500034	.161769	0.001
NIHICP_L	-.0550239	.0163976	0.001
CLD	-2.572296	.8912236	0.004
Intercept	-3.332759	.6132243	0.000

Table 15: Model II - based on coupling, cohesion, and inheritance measures

these covariates, which is not influential and therefore retained. The conditional number is $\sqrt{2.9261/0.0405} \approx 8.5031$.

The measures NMI, RFC_1_L, and NOP cannot be clearly attributed to one PC. The remaining four measures cover PC1 (method invocations of library classes), PC2 (method invocations of non-library classes), PC9 (depth of inheritance tree below the class), and PC11 (export coupling to friend classes).

The model has a much better goodness of fit than Model I (loglikelihood = -70.62, R^2=0.53). Again, we applied the model to the 113 classes to compare the predicted and actual fault-proneness:

		predicted		Σ
		π<=0.65	π>0.65	
actual	no fault	50 classes	10 classes	60 classes
	fault	10 classes, 15 faults	43 classes, 219 faults	53 classes 234 faults
Σ		60 classes	53 classes	113 classes

Table 16: Goodness of fit of model II

We selected a threshold of 0.65 to balance the number of actual and predicted fault-prone classes. Model II performs much better: Off the 53 classes predicted fault-prone, 43 actually are fault-prone (81% correctness), which contain 219 of all 234 faults (94% completeness). The Kappa measure of agreement between actual and predicted fault-proneness is 0.64, much higher than Model I. The high goodness of fit indicates that the coupling and inheritance measures capture structural dimensions with a strong relationship to fault-proneness, which go beyond the size of classes. A model based on structural class properties can outperform models based on class size measures alone, which justifies the extra effort to collect these measures.

Model built from coupling, cohesion, inheritance, and design size measures

Finally, we will consider a model built using the forward selection process, allowing all coupling, cohesion, inheritance, and size measures to enter the model:

This model (Model III) consists of nine covariates: four coupling measures, three inheritance measures (though NMA rather should be considered measuring size), and two size measures. With nine covariates, the “ten data points per covariate” rule of thumb is still observed. This model has two multivariate outliers, which again were not influential and therefore retained to fit the final model. The conditional number of $\sqrt{3.1154/0.0271} \approx 10.720$ is higher than that of the previous to models, but still well below the critical threshold of 30.

Compared to Model II, dimension PC1 is no longer represented in the model. Instead, size measures (including NumPara of PC3) and another export coupling measure (OCAEC of PC5) are included.

Measure	Coeff.	Std. Error	p
CLD	-3.517571	1.315061	0.007
NOP	4.171345	.6731571	0.000
OCAEC	-1.048326	.405435	0.010
RFC_{∞}	.4639629	.1113023	0.000
FMMEC	.3506641	.1500296	0.019
NM	-.7262171	.1636992	0.000
NMA	.5256345	.1103538	0.000
NumPara	-.118577	.0430447	0.006
OMMIC	-.2693183	.0848974	0.002
Intercept	-2.836428	.6702554	0.000

Table 17: Model III - allowing all measures to enter the model

The results above could indicate that RFC_1_L and NIH_ICP_L in Model II were included because of their moderate relationship to size, which is related to fault-proneness. When size measures are allowed to enter the model, those coupling measures are then not selected as significant covariates anymore.

There is a higher amount of multicollinearity present than in Model II, as shown by the VIFs. OMMIC and RFC_{∞} of PC2 have coefficients with opposite signs, the size measures NM and NumPara have opposite signs. This makes the interpretation of the model difficult. However, it is not our goal here to come up with interpretable multivariate models, which is inherently difficult in the context of regression; first and foremost we are interested in their goodness of fit.

The loglikelihood of this model is -65.11, and the R^2 = 0.56, i.e., slightly higher than the model without size measures. The classification results of this model actually are, however, a bit worse than Model II, but they are still very good:

		predicted		Σ
		π<=0.7	π>0.7	
actual	no fault	49 classes	11 classes	60 classes
	fault	11 classes, 24 faults	42 classes, 210 faults	53 classes 234 faults
Σ		60 classes	53 classes	113 classes

Table 18: Goodness of fit of model III

A threshold of 0.7 was selected to better balance the number of actual and predicted fault-prone classes. 53 classes are predicted fault-prone, which contain 210 faults (90% completeness), and 11 of these classes are not actually fault-prone (79% correctness). Kappa=0.60 is slightly lower than that of Model II.

4.5.2 Model evaluation

Let us further investigate Model II which yielded the best classification for class fault-proneness. The accuracy of this model looks very good. However, it is somewhat optimistic since the prediction model was applied to the same data set it was derived from, i.e., we were assessing the goodness of fit of the model. To get an impression of how well the model performs when applied to different data sets, i.e., its prediction accuracy, we performed a 10-cross validation (Stone, 1974) of Model II. Then, more realistic values for completeness, correctness and Kappa can be devised.

For the 10-cross validation, the 113 data points were randomly split into ten partitions of roughly equal size. For each partition, we re-fitted the model using all data points not included in the partition, and then applied the model to the data points in the partition. We thus again obtain for all 113 classes a predicted probability of their fault-proneness. Classes with predicted probability π<=0.65 were classified “not predicted fault-

prone", the others "predicted fault-prone". The threshold 0.65 was retained from the model validation in the previous section.

		predicted		Σ
		π<=0.65	π>0.65	
actual	no fault	48 classes	12 classes	60 classes
	fault	11 classes, 17 faults	42 classes, 217 faults	53 classes 234 faults
Σ		59 classes	54 classes	113 classes

Table 19: Result from 10-cross-validation of Model II

The 54 classes predicted fault-prone contain 217 faults (92% completeness), 14 of these classes do not actually contain a fault (78% correctness). The Kappa value of 0.59 is not much lower than the 0.64 value obtained for Model II above.

The completeness and correctness figures are slightly lower than above. A prediction model is likely to perform better when it is applied to the data set it was derived from. Since with a 10-cross validation this is no longer the case, the model performs worse. These results, however, indicate that the model is still viable when applied to a comparable data set other than the one used for model fitting.

Model application

To illustrate how a prediction model such as the one presented above can be applied in practice, consider Figure 2. On the X-axis, we plotted the 113 system classes, sorted in decreasing order of their predicted fault-proneness. The values for the predicted fault-proneness were taken from the 10-cross-validation. The lower curve is the cumulative size of the classes (measured by *Stmts*, the number of executable and declaration statements), in percent. For example, at X=30 the value of the curve is 43, which means that the 30 classes with highest predicted fault-proneness constitute 43% of the code of the system. The upper curve is the accumulated number of *actual* faults in the classes, in percent. At X=30, this curve is at Y≈64, which means the first 30 classes contain 64% of the actual faults.

When planning inspections during the software development process, we would like to be able to make a trade-off between the resources spent on inspections on the one hand, and the effectiveness of inspections on the other hand. To optimize this trade-off, we can use the graph in Figure 2 to determine how many percent of the faults in the system we can expect to find by inspecting a certain percentage of the system code, and which classes should be inspected. For instance, assume we have resources available to inspect 30% of the code. From Figure 2 we can tell that the first 20 classes with highest predicted fault-proneness roughly constitute 30% of the code. If we select these classes for inspection, then we can expect to find a maximum of 52% of all faults in them.

A problem with the practical application of the graph in Figure 2 is that the upper curve, the cumulative percentage of *actual* faults, is, a priori, unknown. However, if the shape of the upper curve proves to be a constant across projects within a development environment, the graph could be used as a reference when planning inspections. Initial results indicate that the shape of the graph may indeed be similar in different projects. In a replication of the analysis described in this paper (Briand et al., 1998b), using data from an environment with very different characteristics (industrial setting with professional developers and a larger system), the analogous graph for that data set was very close to the one presented here. But in general, whether the shape of the upper curve is constant across projects in a given environment needs to be investigated.

Alternatively, a different analysis technique such as poisson regression (Long, 1997) could be used, which yields a model that predicts the number of faults in a class (as opposed to the predicted probability of fault detection in logistic regression). The predicted number of faults could then be used as a basis for planning inspections. The application of poisson regression to build software quality models will be part of our future work.

As is visible in Figure 2, the upper curve (percent of actual faults) shows a steeper slope than the lower curve (percent of system size), before reaching a plateau. The lower curve shows an approximately linear growth. This is another indication that our model does not simply assign higher fault-proneness to larger classes, but

that the coupling and inheritance measures capture a strong additional effect beyond size, that has an impact on the fault-proneness of classes.

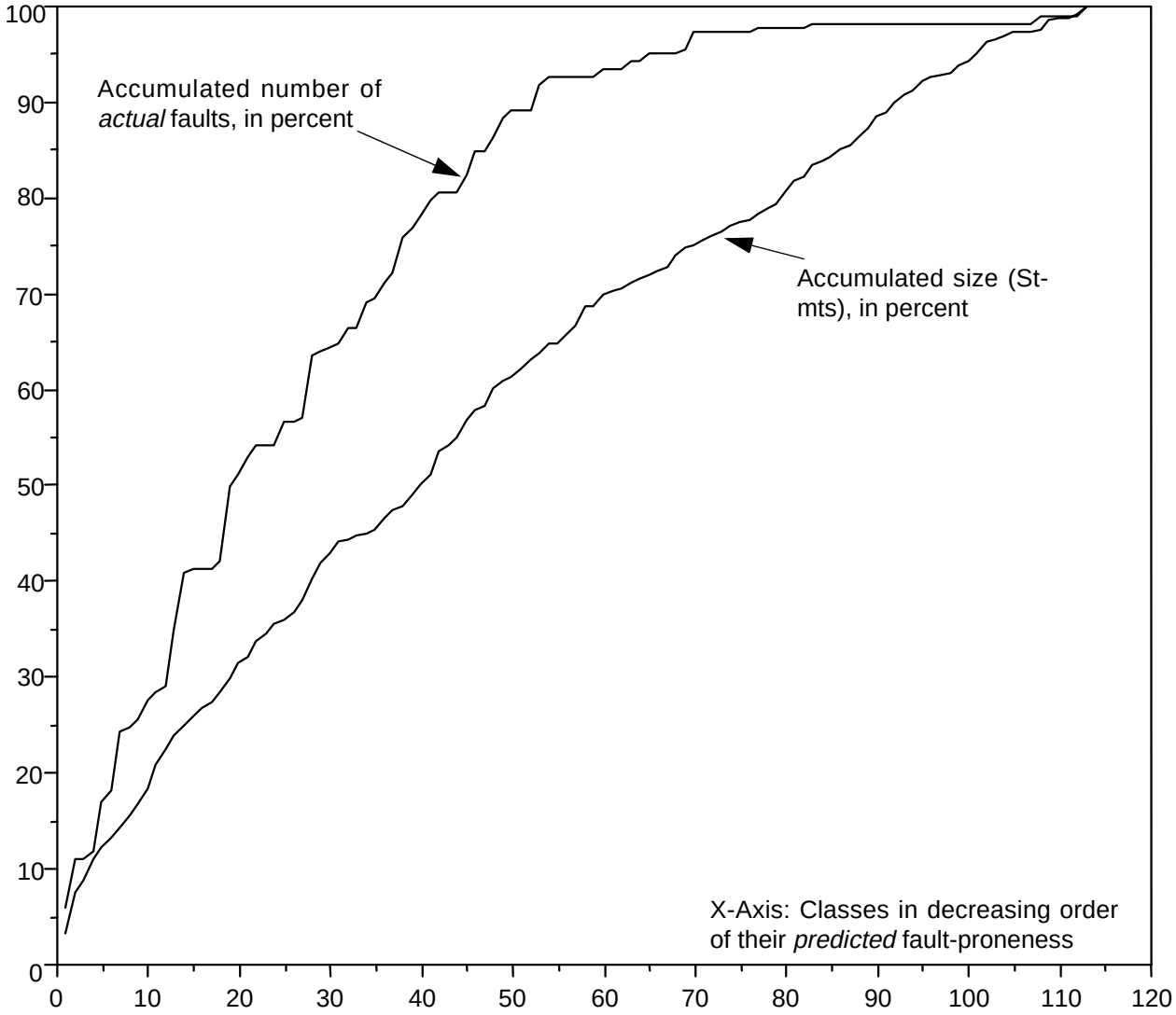

FIGURE 2. Application of the prediction model

4.6 Threats to validity

We distinguish between three types of threats to validity for an empirical study:

- Construct validity: The degree to which the independent and dependent variables accurately measure the concepts they purport to measure.
- Internal validity: The degree to which conclusions can be drawn about the causal effect of the independent variables on the dependent variables.
- External validity: The degree to which the results of the research can be generalized to the population under study and other research setting.

We now apply these criteria to assess the results described above.

4.6.1 Construct validity

The dependent variable we used is the probability that a fault is detected in a class during acceptance testing. Assuming testing was performed properly and thoroughly, the probability of fault detection in a class during acceptance testing should be a good indicator of its probability of containing a fault and, therefore, a valid

measure of fault-proneness. The construct validity of our dependent variable can thus be considered satisfactory

For construct validity of the independent variables, we have to address the question to which degree the coupling, cohesion, and inheritance measures used in this study measure the concepts they purport to measure. In (Briand et al., 1999; Briand et al., 1998a), the coupling and cohesion measures were theoretically validated against properties for coupling and cohesion measures proposed in (Briand et al., 1996b). These properties are one of the more recent proposals to characterize coupling and cohesion in an operational, and reasonably intuitive manner. The properties are considered necessary but not sufficient, as a measure that fulfills all properties is not guaranteed to make sense or be useful.

Of the coupling measures, the following measures fulfill all coupling properties: MPC, the ICP measures, and the measures of the suite by Briand et al. (Briand et al., 1997b). Of the remaining measures, some measures do not have a null value (the RFC measures, DAC and DAC'), some are not additive when two unconnected classes are merged (CBO, CBO', the RFC measures, DAC').

Of the cohesion measures, only Coh, TCC and LCC fulfill all cohesion properties. The other measures are not normalized (LCOM1-LCOM4, ICH), or not properly normalized as they make assumptions which may not be fulfilled for all classes (LCOM5, Co). In addition, LCOM2 is not monotonic, and, as already discussed, ICH is additive when unconnected classes are merged.

For the inheritance measures, the concepts they purport to measure are not clear. No empirical relation systems for these measures have been proposed; doing so would be desirable but it is beyond the scope of this paper.

At this point, it is interesting to compare the construct validity of the measures and their empirical relationship to fault-proneness, referred to as their "empirical validity". All four combinations of theoretical and empirical validity occur: From the theoretically valid measures, some are also empirically valid (e.g., OMMIC), some or not (e.g., LCC). Likewise, among the measures that were not successfully theoretically validated, some still are empirically valid (e.g., the RFC measures), others are not (LCOM1).

As we see, construct validity does not imply empirical validity, and vice versa. That is, enforcing construct validity does not help to finding measures that are also good quality indicators. However, it helps interpreting the results we obtain.Only when we use theoretically valid measures of coupling or cohesion can we say that we demonstrated the impact of coupling and cohesion on fault-proneness. Also, it adds more credibility to an underlying assumption that cannot be proven by statistical testing, namely, that the relationships we find are not only statistical ones, but *causal* ones. The problem with measures that are not theoretically valid, but still good quality indicators, is that we cannot be sure about what we are actually measuring.

The function of the measures as "design measures" is emphasized, i.e., these measures are applicable in early stages of the development process. If these measures are found to be useful, they can be used to detect problems in the design before implementation starts, thus potentially saving time and effort for rework of the design. However, since we needed testing fault data to perform the analysis, measurement was performed only after implementation was completed. If measurement had taken place, say, before implementation started, different measurement data could have been obtained because of the uncertainty inherent to early design information (e.g., the information about actual method invocations may be uncertain before implementation and method invocations are counted by CBO, RFC, MPC, OMMIC, IFMMIC, AMMIC, OMMEC, FMMEC, and DMMEC). This in turn could have led to different results in the statistical analyses.

4.6.2 Internal validity

The analysis performed here is correlational in nature. We have demonstrated that several of the measures investigated had a statistically and practically significant relationship with fault proneness during testing. Such statistical relationships do not demonstrate *per se* a causal relationship. They only provide empirical evidence of it. Only controlled experiments, where the measures would be varied in a controlled manner and all other factors would be held constant, could really demonstrate causality. However, such a controlled experiment would be difficult to run since varying coupling, cohesion, and inheritance in a system, while preserving its functionality, is difficult in practice. Some attempts to do so are reported in (Briand et al., 1996a; Briand et al., 1997a). On the other hand, it is difficult to imagine what could be alternative explanations for

our results besides a relationship between coupling, inheritance depth, and the cognitive complexity of classes.

4.6.3 External validity

The following facts may restrict the generalizability of our results.

- The systems developed are rather small: they lie between 5000 and 14000 source lines of C++ code.
- The systems developed have a limited conceptual complexity.
- The developers are not as well trained and experienced as average professional programmers.

5.0 Conclusions

Our main goal was to perform a comprehensive empirical validation of all the object-oriented (OO) design measures found in the literature. We wanted to understand their interrelationships, their individual impact on class fault proneness, and, when used together, their capability to predict where faults are located. To do so, a repeatable, complete analysis procedure is proposed for future replications and can help the comparison of results coming from different data sets. This is a fundamental requirement if we want to build a cumulative body of knowledge through replicated studies.

Many of the coupling, cohesion, and inheritance measures studied in this paper appear to capture similar dimensions in the data. In fact, the number of dimensions actually captured by the measures is much lower than the number of measures itself. This simply reflects the fact that many of the measures proposed in the literature are based on comparable ideas and hypotheses, and are therefore somewhat redundant.

Univariate analysis results have shown that many coupling and inheritance measures are strongly related to the probability of fault detection in a class. In particular, coupling induced by method invocations, the rate of change in a class due to specialization, and the depth of a class in its inheritance hierarchy appear to be important quality factors. On the other hand, cohesion, as currently captured by existing measures, does not seem to have a significant impact on fault proneness. This is likely to reflect two facts: (1) the weak understanding we currently have of what this attribute is supposed to capture, (2) the difficulty to measure such a concept through syntactical analysis only.

Multivariate analysis results show that by using some of the coupling and inheritance measures, very accurate models can be derived to predict in which classes most of the faults actually lie. When predicting fault-prone classes, the best model shows a percentage of correct classifications about 80% and finds more than 90% of faulty classes.

The study performed in this paper should be replicated across many environments and systems in order for our community to draw general conclusions about what OO measurement can do to help assess the quality of early designs and systems. It would also be interesting to investigate, in a similar fashion, the relationship between OO design measures and other external quality attributes, such as maintainability (Briand et al., 1996a; Briand et al., 1997a). The results above still support the idea that measurement of OO designs can still shed light on their quality. However, as long as empirical studies remain as scarce or incomplete as they are today, product quality measurement for OO development is likely to remain an elusive target.

Acknowledgments

The authors would like to thank Michael Ochs for developing the M-System, by which our measurement was performed. We also want to thank the anonymous reviewers and Dr. Khaled El Emam, Bernd Freimut, and Isabella Wieczorek for their helpful comments on drafts of this report. Special thanks also go to Drs. Basili and Melo who were, with Lionel Briand, involved in the original studies from which we reused the defect data used here.

Appendix A: Measures used in this study

Tables 20 to 22 describe the coupling, cohesion, and inheritance measures used in this study. In each table, the column “Name” states the acronym of each measure and what it stands for. In “Definition”, we provide

an informal natural language definition of the measures. These should give the reader a quick insight into the measures. However, such definitions tend to be ambiguous. Formal definitions of the measures using a uniform and unambiguous formalism are provided in (Briand et al., 1999; Briand et al., 1998a). Column “Source” indicates the literature reference where the measure has originally been proposed.

Name	Definition	Source
CBO (coupling between object classes)	According to the definition of this measures, a class is coupled to another, if methods of one class use methods or attributes of the other, or vice versa. CBO for a class is then defined as the number of other classes to which it is coupled. This includes inheritance-based coupling (coupling between classes related via inheritance).	(Chidamber and Kemerer, 1994)
CBO’	Same as CBO, except that inheritance-based coupling is not counted.	(Chidamber and Kemerer, 1991)
RFC_{∞} (response set for class)	The response set of a class consists of the set M of methods of the class, and the set of methods directly or indirectly invoked by methods in M. In other words, the response set is the set of methods that can potentially be executed in response to a message received by an object of that class. RFC is the number of methods in the response set of the class.	(Chidamber and Kemerer, 1991)
RFC_1	Same as RFC_{∞}, except that methods indirectly invoked by methods in M are not included in the response set this time.	(Chidamber and Kemerer, 1994)
MPC (message passing coupling)	The number of method invocations in a class.	(Li and Henry, 1993)
DAC (data abstraction coupling)	The number of attributes in a class that have as their type another class.	
DAC´	The number of different classes that are used as types of attributes in a class.	
ICP (information-flow-based coupling)	The number of method invocations in a class, weighted by the number of parameters of the invoked methods.	(Lee et al., 1995)
IH-ICP (information-flow-based inheritance coupling)	As ICP, but counts invocations of methods of ancestors of classes (i.e., inheritance-based coupling) only.	(Lee et al., 1995)
NIH-ICP (information-flow-based non-inheritance coupling)	As ICP, but counts invocations to classes not related through inheritance.	(Lee et al., 1995)

Table 20: Coupling measures

Name	Definition	Source
IFCAIC	These coupling measures are counts of interactions between classes. The measures distinguish the relationship between the classes (friendship, inheritance, none), different types of interactions, and the locus of impact of the interaction. The acronyms for the measures indicates what interactions are counted: • The first or first two letters indicate the relationship (A: coupling to ancestor classes, D: Descendents, F: Friend classes, IF: Inverse Friends (classes that declare a given class *c* as their friend), O: Others, i.e., none of the other relationships). • The next two letters indicate the type of interaction: CA: There is a Class-Attribute interaction between classes *c* and *d*, if *c* has an attribute of type *d*. CM: There is a Class-Method interaction between classes *c* and *d*, if class *c* has a method with a parameter of type class *d*. MM: There is a Method-Method interaction between classes *c* and *d*, if *c* invokes a method of *d*, or if a method of class *d* is passed as parameter (function pointer) to a method of class *c*. • The last two letters indicate the locus of impact: IC: Import coupling, the measure counts for a class *c* all interactions where *c* is using another class. EC: Export coupling: count interactions where class *d* is the used class.	(Briand et al., 1997b)
ACAIC		
OCAIC		
FCAEC		
DCAEC		
OCAEC		
IFCMIC		
ACMIC		
OCMIC		
FCMEC		
DCMEC		
OCMEC		
OMMIC		
IFMMIC		
AMMIC		
OMMEC		
FMMEC		
DMMEC		

Table 20: Coupling measures

Name	Definition	Source
LCOM1 (lack of cohesion in methods)	The number of pairs of methods in the class using no attribute in common.	(Chidamber and Kemerer, 1991; Henderson-Sellers, 1996)
LCOM2	LCOM2 is the number of pairs of methods in the class using no attributes in common, minus the number of pairs of methods that do. If this difference is negative, however, LCOM2 is set to zero.	(Chidamber and Kemerer, 1994)
LCOM3	Consider an undirected graph *G*, where the vertices are the methods of a class, and there is an edge between two vertices if the corresponding methods use at least an attribute in common. LCOM3 is then defined as the number of connected components of *G*.	(Hitz and Montazeri, 1995)
LCOM4	Like LCOM3, where graph *G* additionally has an edge between vertices representing methods *m* and *n*, if *m* invokes *n* or vice versa.	(Hitz and Montazeri, 1995)
Co (connectivity)	Let V be the number of vertices of graph G from measure LCOM4, and E the number of its edges. Then $Co = 2 \cdot \frac{|E| - (|V| - 1)}{(|V| - 1) \cdot (|V| - 2)}$.	(Hitz and Montazeri, 1995) (named "C")
LCOM5	Consider a set of methods $\{M_i\}$ ($i=1,...,m$) accessing a set of attributes $\{A_j\}$ ($j=1,...,a$). Let $\mu(A_j)$ be the number of methods which reference attribute A_j. Then $LCOM5 = \frac{\frac{1}{a}\left(\sum_{j=1}^{a} \mu(A_j)\right) - m}{1 - m}$	(Henderson-Sellers, 1996)
Coh	A variation on LCOM5: $Coh = \frac{\sum_{j=1}^{-} \mu(A_j)}{m \cdot a}$	(Briand et al., 1998a)
TCC (tight class cohesion)	Besides methods using attributes directly (by referencing them), this measure considers attributes "indirectly" used by a method. Method *m* uses attribute *a indirectly*, if *m* directly or indirectly invokes a method m' which directly uses attribute *a*. Two methods are called *connected*, if they directly or indirectly use common attributes. TCC is defined as the percentage of pairs of public methods of the class which are connected, i.e., pairs of methods which directly or indirectly use common attributes.	(Bieman and Kang, 1995)
LCC (loose class cohesion)	Same as TCC, except that this measure also considers pairs of "indirectly connected" methods. If there are methods $m_1,..., m_n$, such that m_i and m_{i+1} are connected for $i=1,...,n-1$, then m_1 and m_n are indirectly connected. Measure LCC is the percentage of pairs of public methods of the class which are directly or indirectly connected.	(Bieman and Kang, 1995)
ICH (information-flow-based cohesion	ICH for a method is defined as the number of invocations of other methods of the same class, weighted by the number of parameters of the invoked method (cf. coupling measure ICP above). The ICH of a class is the sum of the ICH values of its methods.	(Lee et al., 1995)

Table 21: Cohesion measures

Name	Definition	Source
DIT (depth of inheritance tree)	The DIT of a class is the length of the longest path from the class to the root in the inheritance hierarchy.	(Chidamber and Kemerer, 1991; Chidamber and Kemerer, 1994)
AID (average inheritance depth of a class)	AID of a class without any ancestors is zero. For all other classes, AID of a class is the average AID of its parent classes, increased by one.	(Henderson-Sellers, 1996)
CLD (class-to-leaf depth)	CLD of a class is the maximum number of levels in the hierarchy that are below the class.	(Tegarden et al., 1992)
NOC (number of children)	The number of classes that directly inherit from a given class.	(Chidamber and Kemerer, 1991; Chidamber and Kemerer, 1994)
NOP (number of parents)	The number of classes that a given class directly inherits from.	(Lake and Cook, 1994; Lorenz and Kidd, 1994)
NOD (number of descendents)	The number of classes that directly or indirectly inherit from a class (i.e., its children, 'grand-children', and so on).	(Lake and Cook, 1994; Tegarden et al., 1992)
NOA (number of ancestors)	The number of classes that a given class directly or indirectly inherits from.	(Tegarden et al., 1992)
NMO (number of methods overridden)	The number of methods in a class that override a method inherited from an ancestor class.	(Lorenz and Kidd, 1994)
NMI (number of methods inherited)	The number of methods in a class that the class inherits from its ancestors and does not override.	(Lorenz and Kidd, 1994)
NMA (number of methods added)	The number of new methods in a class, not inherited, not overriding.	(Lorenz and Kidd, 1994)
SIX (specialization index)	SIX is NMO * DIT / (NMO+NMA+NMI)	(Lorenz and Kidd, 1994)

Table 22: Inheritance related measures

In order to compare how these measure relate to size we also consider a number of size measures defined in Table 23.

Name	Definition
Stmts	The number of declaration and executable statements in the method of a class.
NM (Number of methods)	The number of all methods (inherited, overriding, and non-inherited) methods of a class. This
NAI (Number of attributes)	The number of attributes in a class (excluding inherited ones). Includes attributes of basic types such as strings, integers.
NMpub (Number of public methods)	The number of public methods implemented in a class.
NMNpub (Number of non-public methods)	The number of non-public (i.e., protected or private) methods implemented in a class.
NumPara (Number of parameters)	The sum of the number of parameters of the methods implemented in a class.

Table 23: Size measures

References

All ISERN technical reports are available from http://www.iese.fhg.de/ISERN/pub/isern_biblio_tech.html.

Basili, V., Briand, L., Melo, W., 1996. A Validation of Object-Oriented Design Metrics as Quality Indicators. IEEE Transactions on Software Engineering, 22 (10), 751-761.

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

Bieman, J., Kang, B., 1995. Cohesion and Reuse in an Object-Oriented System. Proc. ACM Symp. Software Reusability (SSR'94), 259-262.

Briand, L., Bunse, C., Daly, J., Differding, C., 1996. An Experimental Comparison of the Maintainability of Object-Oriented and Structured Design Documents, Empirical Software Engineering Journal, 2 (3), 291-312.

Briand, L., Bunse, C., Daly, J., 1997a. An Experimental Evaluation of Quality Guidelines on the Maintainability of Object-Oriented Design Documents. Technical Report ISERN-97-02, Fraunhofer Institute for Experimental Software Engineering, Germany. Also published in the proceedings of Empirical Studies of Programmers (ESP'97).

Briand, L., Devanbu, P., Melo, W., 1997b. An Investigation into Coupling Measures for C++. Technical Report ISERN-96-08, Proceedings of ICSE '97, Boston, USA.

Briand, L., Daly, J., Wüst, J., 1999. A Unified Framework for Coupling Measurement in Object-Oriented Systems. IEEE Transactions on Software Engineering 25 (1), 91-121.

Briand, L., Daly, J., Wüst, J., 1998a. A Unified Framework for Cohesion Measurement in Object-Oriented Systems. Empirical Software Engineering Journal, 3 (1), 65-117.

Briand, L., Ikonomovski, S., Lounis, H., Wüst, J., 1998b. A Comprehensive Investigation of Quality Factors in Object-Oriented Designs: an Industrial Case Study", Technical Report ISERN-98-29, Fraunhofer Institute for Experimental Software Engineering, Germany. A short version is to be published in the proceedings of ICSE'99.

Briand, J., Morasca, S., Basili, V., 1996b. Property-Based Software Engineering Measurement. IEEE Transactions of Software Engineering, 22 (1), 68-86.

Barnett, V., Price, T., 1995. Outliers in Statistical Data. 3rd ed., John Wiley & Sons.

Chatterjee, S., Price, B., 1991. Regression Analysis by Example. 2nd ed., John Wiley & Sons.

Chidamber, S., Darcy D., Kemerer, C., 1998. Managerial use of Metrics for Object-Oriented Software: An Exploratory Analysis. IEEE Transactions on Software Engineering, 24 (8), 629-639.

Chidamber, S., Kemerer, C., 1991. Towards a Metrics Suite for Object Oriented design". Proc. Conference on Object-Oriented Programming: Systems, Languages and Applications (OOPSLA'91). Published in SIGPLAN Notices, 26 (11), 197-211.

Chidamber, S., Kemerer, C., 1994. A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering, 20 (6), 476-493.

Cohen, J., 1960. A Coefficient of Agreement for Nominal Scales. Educational and Psychological Measurement, Vol. XX, No. 1.

Cartwright, M., Shepperd, M., 1999. An Empirical Investigation of an Object-Oriented Software System. IEEE Transactions of Software Engineering, to appear.

Darlington, R., 1968. Multiple Regression in Psychological Research and Practice. Psychological Bulletin, 69 (3), 161-182.

Devanbu, P., 1992. "A Language and Front-end Independent Source Code Analyzer. Proceedings of ICSE '92, Melbourne, Australia.

Dunteman, G., 1989. Principal Component Analysis. Sage University Paper 07-69, Thousand Oaks, CA.

Henderson-Sellers, B., 1996. Software Metrics, Prentice Hall, Hemel Hempstaed, U.K.

Hosmer, D., Lemeshow, S., 1989. Applied Logistic Regression, John Wiley & Sons.

Hitz, M., Montazeri, B., 1995. Measuring Coupling and Cohesion in Object-Oriented Systems. Proc. Int. Symposium on Applied Corporate Computing, Monterrey, Mexico. http://www.pri.univie.ac.at/~hitz/papers/ESEC95.ps

Khoshgoftaar, T., Allen, E., 1997. Logistic Regression Modeling of Software Quality, TR-CSE-97-24, Florida Atlantic University.

Lake, A., Cook, C., 1994. Use of factor analysis to develop OOP software complexity metrics. Proc. 6th Annual Oregon Workshop on Software Metrics, Silver Falls, Oregon.

Li, W., Henry, S., 1993. Object-Oriented Metrics that Predict Maintainability. Journal of Systems and Software, 23 (2), 111-122.

Long, S., 1997. Regression Models for Categorical and Limited Dependent Variables. Advanced Quantitative Techniques in the Social Sciences Series, Sage Publications.

Lorenz, M., Kidd, J., 1994. Object-Oriented Software Metrics. Prentice Hall Object-Oriented Series, Englewood Cliffs, N.J.

Lee, Y., Liang, B., Wu, S., Wang, F., 1995. Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow, Proc. International Conference on Software Quality, Maribor, Slovenia, 1995.

Mayrand, J., Coallier, F., 1996. System Acquisition Based on Software Product Assessment. Proceedings of ICSE'96, Berlin, Germany, 210-219.

Menard, S., 1995. Applied Logistic Regression Analysis. Sage University Paper 07-106, Thousand Oaks, CA.

Rosenberg, J., 1997. Some Misconceptions About Lines of Code, Proceedings of the 4th International Software Metrics Symposium (Metrics '97), 137-142.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., 1991. Object-Oriented Modeling and Design, Prentice-Hall.

Stone, M., 1974. Cross-validatory choice and assessment of statistical predictions, J. Royal Stat. Soc., Ser. B 36, 111-147.

Tegarden, D., Sheetz, S., Monarchi, D., A Software Complexity Model of Object-Oriented Systems. Decision Support Systems, 13(3-4), 241-262.

About the authors

Lionel C. Briand received the B.S. degree in geophysics and the M.S. degree in Computer Science from the University of Paris VI, France. He received the Ph.D. degree, with high honors, in Computer Science from the University of Paris XI, France.

Lionel is currently the head of the Quality and Process Engineering Department at the Fraunhofer Institute for Experimental Software Engineering (FhG IESE), an industry-oriented research center located in Rheinland-Pfalz, Germany. His current research interests and industrial activities include measurement and modeling of software development products and processes, software quality assurance, domain specific architectures, reuse, and reengineering. He has published numerous articles in international conferences and journals and has been a PC member or chair in several conferences such as ICSE, ICSM, ISSRE, METRICS, and SEKE. Before that, Lionel started his career as a software engineer at CISI Ingénerie, France. He then joined, as a research scientist, the NASA Software Engineering Laboratory, a research consortium: NASA Goddard Space Flight Center, University of Maryland, and Computer Science Corporation. Before going to FhG IESE, he held the position of lead researcher of the software engineering group at CRIM, the Computer Research Institute of Montreal, Canada.

John W. Daly received the B.Sc. and Ph.D. degrees in Computer Science from the University of Strathclyde, Glasgow, Scotland in 1992 and 1996, respectively. Between 1996 and 1998, John was a software engineering researcher and then a research project manager in the Quality and Process Engineering Department at the Fraunhofer Institute for Experimental Software Engineering, Germany. In April 1998, he joined the Quality Assurance Department at Hewlett Packard Ltd., South Queensferry, Scotland as a software process engineer.

John's industrial activities and current research interests include software measurement, software process and improvement, software quality, and object-oriented development techniques.

D. Victor Porter received the B.Sc. degree in Mathematics and Computer Science from the University of Newcastle upon Tyne in 1992. He received the M.Sc. degree, with distinction, in Information Systems from the University of Stirling in 1997. He was a visiting researcher at the Fraunhofer IESE, Kaiserslautern in 1997. In 1998 he joined IBM Global Services at Perth, Scotland, where he is developing OO software for mainframe environment solutions using the RAD tool Synthesys.

Jürgen Wüst received the degree Diplom-Informatiker (M.S.) in Computer Science with a minor in Mathematics from the University of Kaiserslautern, Germany, in 1997. He is currently a researcher at the Fraunhofer Institute for Experimental Software Engineering (IESE) in Kaiserslautern, Germany. His current research activities and industrial activities include software measurement, software architecture evaluation, and object-oriented development techniques.