Empirical Software Engineering, 6, 11–58, 2001.

Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs

LIONEL C. BRIAND briand@sce.carleton.ca
Carleton University Systems and Computer Engineering 1125 Colonel By Drive Ottawa, ON, K1S 5B6, Canada

JÜRGEN WÜST wuest@iese.fhg.de
Fraunhofer Institute for Experimental Software Engineering Sauerwiesen 6 67661 Kaiserslautern, Germany

HAKIM LOUNIS lounis.hakim@uqam.ca
Université du Québec à Montréal Département d'informatique Case postale 8888, succursale Centre-ville, Montréal, Canada H3C 3P8

Abstract. This paper aims at empirically exploring the relationships between most of the existing design coupling, cohesion, and inheritance measures for object-oriented (OO) systems, and the fault-proneness of OO system classes. The underlying goal of this study is to better understand the relationship between existing design measurement in OO systems and the quality of the software developed. in addition, we aim at assessing whether such relationships, once modeled, can be used to effectively drive and focus inspections or testing.

The study described here is a replication of an analogous study conducted in a university environment with systems developed by students. In order to draw more general conclusions and to (dis)confirm the results obtained there, we now replicated the study using data collected on an industrial system developed by professionals.

Results show that many of our findings are consistent across systems, despite the very disparate nature of the systems under study. Some of the strong dimensions captured by the measures in each data set are visible in both the university and industrial case study. For example, the frequency of method invocations appears to be the main driving factor of fault-proneness in all systems. However, there are also differences across studies, which illustrate the fact that, although many principles and techniques can be reused, quality does not follow universal laws and quality models must be developed locally, wherever needed.

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

1. Introduction

A large number of object-oriented (OO) measures have been proposed in the literature (Briand et al, 1997) Bieman and Kang, 1995; Chidamber and Kemerer, 1991; Chidamber and Kemerer, 1994; FAST, 1997; Hitz and Montazeri, 1995; Li and Henry, 1993; Lake and Cook, 1994; Lorenz and Kidd, 1994; Lee et al,1995). A particular emphasis has been given to the measurement of design artifacts, in order to help assess quality early on during the development process. 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 large software systems can be thus done very fast, at a low cost, with little human involvement. But how do we know what measures actually capture important quality aspects? Many of the measures proposed and their relationships to external quality attributes of OO designs, have been the focus of little empirical investigation (Basili et

al, 1996; Briand et al, 1997; Briand et al, 1998; Chidamber et al, 1998; Cartwright and Shepperd, 1997; Li and Henry, 1993). It is therefore difficult to assess whether these measures capture complementary dimensions or are indicators of any relevant external quality attributes such as reliability or maintainability. In this context, it is interesting to note that the new ISO/IEC international standard (14598) (ISO/IEC DIS 14598-1) on software product quality states that "Internal metrics are of little value unless there is evidence that they are related to external quality" and that "Internal metrics are of little value unless there is evidence that they are related to external quality" and that "Internal metrics should also have predictive validity, that is that they should correlate with some desired external criterion". The general opinion of the practitioners involved in the making of this ISO standard seems to be that empirical investigations regarding product internal measurement and its relationship to external quality measures are, from a practical perspective, a crucial issue for the betterment of quality control early on n the life cycle. This is, however, in sharp contrast to the lack of empirical results published to date.

Recently, some of the authors performed an in-depth, comprehensive analysis of most of the published OO measures on students' projects (Briand et al, 1998) of rather small sizes. Based on data collected in an experiment in a university setting, three questions were addressed:

- Are existing OO design measures capturing different dimensions and structural aspects? If not, what are the underlying structural dimensions they actually capture? Analyzing the structural dimensions covered by the design measures sheds some light on the amount of redundancy that is present among existing measures, and helps to better interpret what individual measures are really capturing.

- What is the relationship between these measures and the likelihood of detecting a fault in a class, i.e., its fault-proneness? Which ones strongly affect fault-proneness? The high cognitive complexity of classes may result in many different types of problems such as low maintainability or high fault-proneness. However, fault-proneness is not only an important quality aspect related to class cognitive complexity but also the easiest one to observe and measure, hence its use in our studies. 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.

- How accurate are the existing measures in predicting faulty classes? To what extent can they be used to drive code and design inspections? It is important to 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.

In order to draw more general conclusions and (dis)confirm the results obtained in our student experiments, we replicate here this analysis on data we collected on an industrial project which is currently in use and under maintenance. In Briand et al. (1998), we proposed and applied a precise, complete, and repeatable analysis procedure, which was designed to facilitate such replications. In this paper, we repeat this procedure to demonstrate how each of the steps in the analysis procedure contributes towards making clear, systematic comparisons across studies.

We identified a number of structural dimensions in OO designs that appear to be related to class fault-proneness across the two data sets. Considering the significant differences between the students' systems and the industrial system studied here (e.g., in terms of size, distribution of the measures, domain, programmer experience), we hope to draw conclusions that should be robust across many systems. Further replication is of course necessary to build an adequate body of knowledge regarding the use of OO design measures.

The paper is organized as follows. Section 2 describes the goals and setting of the empirical study, and the data collected. Section 3 describes the methodology used to analyze the data. The results of this analysis are then presented in Section 4, where we also compare the results to those obtained for the students' systems in (Briand et al, 1998). We draw our conclusions in Section 5.

2. 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 system used for this study is an open multi-agent system development environment, called LALO (Langage d'Agents Logiciel Objet). This system has been developed and maintained since 1993 at CRIM (Centre de Recherche Informatique de Montréal). The users of LALO are mainly universities and research centers.

The LALO system consists of 83 classes that were developed from scratch, and seven additional classes that were automatically generated by code generators. These 90 C++ classes have a total of approximately 40K source lines of code (SLOC). The automatically generated classes were not investigated since they are much less likely to contain faults than classes implemented manually. In fact, the use of these classes could have biased the results.

In addition to the 90 application-specific classes, a number of standard library classes for IO, threading, socket communication, etc., are used in the LALO system.

LALO was mostly developed under Windows NT using Visual C++ and then ported to Sun OS and Solaris. We have investigated only the Sun OS version. Porting the LALO system to different platforms was an easy task that was not the source of additional faults.

Six developers have worked on the LALO system over its lifetime, with at most three developers working on the system in parallel. All developers had several years of experience in system development, and four developers had worked on OO systems before.

The following relevant items were collected for the empirical study:

- The source code of the LALO system, version 1.3
- Fault data related to failures detected by world-wide users of LALO, over a period of about one year.

A tool developed at the Frauhofer IESE, and based on the FAST parser technology of the SEMA group's Concerto2/AUDIT tools (FAST, 1997), was used to extract the values for the object-oriented measures directly from the source code of LALO version 1.3. To collect the fault data, change report forms (CRF) were used to document the nature of each problem reported by a LALO user, the names of the faulty C++ classes, and the type and location of the maintenance change. The CRF's were registered in a revision control system, which could then be used to generate statistics about the number of faults traced back to each individual class.

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, and event is a detection of a fault in a class due to a failure reported by a user. 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 3.1, 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. Even when there is no causal relationship between size and number of faults, a negative correlation between size and fault density can be found (Rosenberg, 1997). This is, however, a pure mathematical artifact, which makes any analysis of the impact of size on fault-proneness difficult.

2.3. Independent Variables

The measures of coupling, cohesion, and inheritance identified in a literature survey on object-oriented design measures (Briand et al, 1998; Briand et al, 1998) are the independent variables used in this study. We focus on design measurement since we want the measurement-based models investigated in this paper to be usable at early stages of software development. Furthermore, we only use measures defined at the class level since this is also the granularity at which the fault data could realistically be collected.

We consider a total of 28 coupling measures, 10 cohesion measures and 11 inheritance measures. Another important aspect to consider is the size of classes. We will investigate the relationship of coupling, cohesion, and inheritance measures to size measures and whether all the former, more complex measures help build better predictive models than size alone. This will be further described and justified in Section 3.2.3. The definitions of all measures used are summarized in the Appendix.

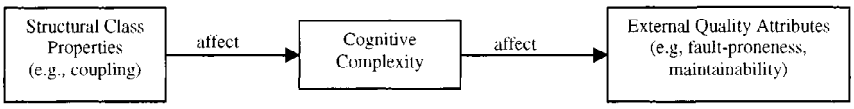

Figure 1. Relationships between structural class properties, cognitive complexity, and external quality.

In order to make possible the use of these measures at the design stage, we adapted some of the measures involving counts of method invocations as follows. Measures that are based on counts of multiple invocations of pairs of methods (say methods m' and m) were changed to be solely sensitive to the fact a given method invokes another one at least once. The rationale for this decision is that the precise number of times a given method m' invokes m is information which is available only after implementation is completed, whereas the information that m' invokes m is usually available earlier during the design phase. The measures affected by this simplification are MPC, the ICP measures, the method-method interaction measures by Briand et al (1997), and ICH.[1]

The values for each of the design measures were collected for each of the 83 LALO classes that were developed from scratch. At this stage, it is pertinent to consider the influence of the library classes and the automatically generated classes (for simplicity, we collectively refer to these classes as 'library classes' hereafter). 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 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 coupling measure were calculated for each application class twice: counting coupling to other application classes only, and counting coupling to library classes only. Analysis was then performed on both variants of the measures.

Similarly, for inheritance we could hypothesize that deriving a class from a library class has a different effect on the derived class than deriving it from a system-specific non-library class. However, in the LALO system we found inheritance relationships only between non-library classes, but not between library- and non-library classes. Therefore, it was not necessary to measure inheritance with library- and non-library classes separately.

2.4. Hypotheses

In this study, we want to test a number of hypotheses, which relate various design measures to fault-proneness. Our hypotheses are mostly derived from the causal chain depicted in Figure 1: The structural properties of a class, measured by the coupling, cohesion, and inheritance measures, affect the cognitive complexity of the class. By cognitive complexity we mean the mental burden of the persons who have to deal with the class (developers, inspectors, testers, maintainers, etc). We believe that it is the, sometimes necessary, high cognitive complexity of a class which causes it to display undesirable external qualities, such as increased fault-proneness, or decreased maintainability and testability. Figure 1

thus illustrates why, in our opinion, structural properties affect external class quality. Note, however, that due the difficulty involved in directly measuring cognitive complexity (subjectivity, unavailability of most developers a the time this study was performed) we do not formally test this explanation, but only look at the relationship between the more easily measured structural class properties, and fault proneness as one example of external quality.

The structural properties of a class are not, however, the only factors which affect a class' cognitive complexity. For instance, the completeness and traceability of the documentation will also have an impact. Cognitive complexity is also dependent on the individuals who deal with the class and is to some degree subjective. Even if we restrict our attention to structural properties, their relationship to external quality attributes will be affected (i.e., interactions) by other factors such as the software engineer's capability and experience and the development methodology in place. So, although we attempt here to identify general trends, we have to expect variability, whose extent is not known. In the context of the more general program of research depicted in Figure 1, we focus, in this study, on an external class quality which can be measured easily and objectively: fault-proneness.

The following hypothesis will be tested in this study:

H-IC (for all important 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, NMINH): 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.

2.5. Comparison to Students' Systems

The results of the current study will be compared to those obtained in a previous study (Briand et al, 1998), where we used data collected from a development project performed at the University of Maryland (UMD) over a four-month period. Eight different groups of developers, composed of undergraduate and postgraduate students, were each asked to develop an information system. The systems were implemented in C++, and ranged in size from 4 to 15 KSLOC. The eight systems contained a total of 113 non-library classes.

The independent variables were the same as described in Section 2.3. However, the fault data used in that study were of different nature since they came from some acceptance testing activity performed on each system by an independent group composed of experienced software professions. See (Briand et al, 1998) for more details on the setting of that study.

3. Data Analysis Methodology

In this section we describe the methodology used to analyze the design measurement data. In Section 3.1, we provide an introduction to our primary analysis technique, logistic regression. In Section 3.2, we describe the overall analysis procedure and the techniques used.

3.1. Logistic Regression

Logistic regression is a standard classification 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 relationship equation (the univariate model is a special case of this, where only one independent variable appears):

$$\pi(X_1, \ldots, X_n) = \frac{e^{(c_0 + c_1 X_1 + \cdots + c_n X_n)}}{1 + e^{(c_0 + c_1 X_1 + \cdots c_n X_n)}}$$

π is the probability that a fault was traced back to a class after a failure during operation, and the $X_i s$ are the design measures included as independent variables in the model (also called covariates).

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 illustrate this concept, consider the example of a (univariate) prediction model for the fault-proneness of classes, as a function of its depth in the inheritance tree (inheritance measure DIT). If, for instance, for DIT $= 3$ we obtain $\pi(3) = 0.4$, we could interpreted this that "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 a single X_i—assuming that all other $X_j s$ are constant—takes a flexible S shape which ranges between two extreme cases:

- When X_i is not significant, then the curve approximates a horizontal line, i.e., π does not depend on X_i.
- When X_i entirely differentiates fault-prone software parts from the ones that are not, 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 the regression equation.

The coefficients c_i are estimated through the maximization of a likelihood function L, 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, $1 = \ln[L]$, 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 obtained our fault data, where, in a given class, each fault instance stems from a distinct failure report form. Each data vector in the data set describes an observation and has the following components: an event category (fault, no fault) and a set of OO design measures (introduced 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 on 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 a logistic regression, the regression coefficients Ci cannot be easily interpreted for this purpose. Instead, we use a measure $\Delta\Psi$, which is based on the notion of the odds ratio (Hosmer and Lemeshow, 1989). More specifically, the odds ration $\Psi(X) = \pi(X)/(1 - \pi(X))$ represents the ration between the probability of having a fault an the probability of not having a fault when the value of the measure is X. As and 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 formula

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

where σ is the standard deviation of measure X. Therefore, $\Delta\Psi$ represents the reduction/increase in the odds ration when the value X increases by one standard deviation.

This is designed to provide an intuitive insight into the impact of independent variables. However, as we will see in Section 4, some measures display very extreme outliers which inflate the standard deviation of those measures. The $\Delta\Psi$s then can no longer be reasonably interpreted. Therefore, such outliers were excluded for the calculation of the $\Delta\Psi$s.

To assess the statistical significance of each independent variable in the model, a likelihood ratio chi-square tests is used. Let $l = \ln[L]$ be the loglikelihood of the model given by our regression 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 $\chi^2(l)$). We test $p = P(\chi^2(l) > G)$. If p is larger than some level of significance α (typically, $\alpha = 0.05$), the observed change in the loglikelihood may well be due to chance, and X_i is not considered significant. If $p <= \alpha$, 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—not to be confused with the least-square regression 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 R^2 of least-square regression, high R^2s are rare for logisitic regression. For this reason, the reader should not interpret logistic regression R^2s using the usual heuristic for least-square regression R^2s. Logistic regression R^2 is defined by the following ratio:

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

LL is the loglikelihood obtained by Maximum Likelihood Estimation of the model described by the regression equation, and LL_S is the loglikelihood obtained by Maximum Likelihood Estimation of a model with the intercept c_0 only. 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.

3.2. Procedure for Data Analysis

The procedure used to analyze the data collected for each measure is described in five stages: (i) descriptive statistics and outlier analyses, (ii) principal component analysis, (iii) corre-

lation to size, (iv) univariate analysis, and (v) multivariate regression analysis (prediction model construction and evaluation). The procedure is aimed at making the results of our study repeatable and comparable to future replications across different environments.

3.2.1. Descriptive Statistics and Outlier Analysis

The data set consists of 83 classes along with the relevant values for each coupling, cohesion, inheritance, and size measure. The distribution (mean, median, and interquartile ranges) and variance (standard deviation) of each measure is examined. Low variance measures do not differentiate classes very well and therefore are not likely to be useful predictors in our data set. Measures with less than five non-zero data points were not considered for subsequent analysis. 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.

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. 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. 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. For this reason it is necessary to identify outliers, test their influence, and possibly remove them to obtain stable results. 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 are not considered in the univariate analysis results.
- 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 measure that takes correlations between measures into account. Multivariate outliers are data points with a large distance from the sample space centroid. Again, a multivariate outlier may be over-influential and therefore removed, if the significance of any of the n 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).

3.2.2. 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 the data set (Munson and Khoshgoftaar, 1989).

Principal components (PCs) are linear combinations of the standardized variables, i.e., design measures. The sum of the square of the weights in each linear combination is equal to one. PCs are calculated as follows. The first PC is the linear combination of all standardized variables that 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 shows large weights and therefore contributed significantly to the variance of each PC. To better identify these variables, the *loadings* of the variables in a given PC can be considered. The loading of a variable is its correlation with the 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 further ease interpretation of 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 themselves 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 replicated studies, it is interesting to see which dimensions are also 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. From such observations, we can also derive recommendations regarding which measures appear to be redundant and need not be collected, without losing a significant amount of design information.

3.2.3. Univariate Regression Analysis

Univariate logistic regression is performed for each individual measure (referred to as independent variable in the context or regression analysis) against the dependent variable, i.e., no fault/fault detection, in order to determine if the measure is a useful predictor of fault-proneness. An introduction to logistic regression was given in Section 3.1. Univariate regression analysis is conducted for two purposes:

1. to test the hypotheses stated in Section 2.4, and

2. to screen out measures not significantly related to fault-proneness. Only measures that are significant at significance level $\alpha = 0.25$ are considered for the subsequent multivariate analysis, as described in Section 3.2.4.

3.2.4. Multivariate Regression Analysis

Multivariate logistic regression is performed to build prediction models of 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

1. 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.

2. reduces multicollinearity, i.e., independent variables which are highly correlated. This makes the model more interpretable, and accurate as the coefficient estimates are improved.

Prediction Model Construction

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

The two major stepwise selection processes used for regression model fitting 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 criterion 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 criterion is fulfilled.

Because of the large number of independent variables used in this study, the initial model in a backward selection process would contain too many variables and could not be interpreted in a meaningful way. Therefore, we opted for the forward selection procedure to build the prediction models.[2] 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 $\alpha_{\text{Exit}} = 0.10$), it is deleted from the model. The process stops when adding the best variable no longer improves the model significantly (at $\alpha_{\text{Enter}} = 0.05$).

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 that violated the rule of thumb to have at least ten data points per independent variable.

Test for Multicollinearity

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 Belsey 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 3.2.2.

Let $X_1, \ldots, X_n$ be the covariates of our model. We perform a principal component analysis on these variables, and set $l_{\max}$ to be the largest eigenvalue, $l_{\min}$ the smallest eigenvalue of the principal components. The conditional number is then defined as $\lambda = \sqrt{l_{\max}/l_{\min}}$ 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).

Prediction Model Evaluation: Goodness of Fit

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 $_0$. 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 their 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. As was the case with the UMD study, we had no reliable data available concerning the severity of the faults.

- *Correctness*: We can always increase the completeness of our prediction model be lowering the threshold p_0 used to classify classes as fault-prone ($\pi > p_0$). This causes more classes to be classified as 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.

Increased correctness comes at the cost of lower completeness, and vice versa. The question then is, what is an optimal threshold p_0 that provides a good tradeoff between correctness and completeness. This depends very much on the situation in which the model is to be used, e.g., the cost of performing an inspection, and the cost of a fault going undetected during inspection. In the absence of assumptions about such external conditions, it is useful to calculate correctness and completeness for all possible thresholds p_0, and present the results in a graph.

A third measure of the goodness of fit is the R^2 statistic, as explained in Section 3.1. unlike completeness, correctness, the definition of R^2 is specific to regression techniques based on maximum-likelihood estimation.

3.2.5. The Impact of Design Size

The size of designs (e.g., class designs) is a necessary part of any predictive model. This is mostly justified by the fact that the size of any software artifact determines, to some extent, many of its external properties such as fault-proneness or effort. On the one hand, we want our predictive models to account for size. But in many cases, e.g., in the case of

fault-proneness models, and for practical reasons, we need them to capture more than size effects. A model that systematically identifies bigger classes as more fault-prone would a priori be less useful: the predicted fault-prone classes are likely to cover a larger part of the system, the model thus could not help to focus inspection and testing efforts very well. For the purpose of assessing the usefulness and practicality of all the predictive models we obtain, we will compare the increase in potential fault detection rate and the increase in size to be tested or inspected.

In addition, for each measure, we analyze its relationship to the size of the class, as measured based on design information. This is to determine empirically whether the measure, even though it is assumed to be 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, this might shed light on its relationship with fault-proneness: larger classes are more likely to contain faults. Recall that we are also interested in increasing our understanding of OO code and design quality, independently of its size. We select the size measures showing the strongest relationship to fault-proneness and most of its variance due to size: NMImp, the number of methods implemented in the class (this corresponds to the simpler version of the well-known WMC measure (Chidamber and Kemerer, 1994)). We then calculate Spearman's Rho coefficient between each design measure and size. We chose a non-parametric measure of correlation, given the skewed distributions of the deign measures that we usually observe. This analysis is more refined but partially redundant with principal component analysis that will show us the measures being part of the same dimension as size measures.

3.3. Comparison to Previous Study

In order to build on previous experience and attempt to generalize further our results, we compare the results obtained from LALO with those obtained from the systems analyzed in (Briand et al, 1998) referred to hereafter as the "UMD systems"). Therefore, in the analyses below, we include a systematic comparison of the results with this previous study and try to explain differences and similarities. Since the systems studied are very different in nature, this should allow us to identify what results are more likely to be generalizable. One difference between the two studies should however be recalled. The UMD systems' dataset contains fault data collected during acceptance testing whereas the LALO fault data comes from operation. Although it is difficult to assess the extent to which this could blur the comparison results, it is important that this fact be stated.

4. Analysis Results

The analysis results are partitioned into five subsections. In Section 4.1 though Section 4.4, we discuss, respectively, the results of the data distribution and outlier analyses, the principal component analysis, correlation to size, and the univariate analysis. Section 4.5 then

Table 1. Descriptive statistics for coupling measures.

	Coupling to non-library classes only							Coupling to library classes only						
Measure	Max	75%	Med	25%	Min	Mean	Std Dev	Max	75%	Med	25%	Min	Mean	Std Dev
CBO	31	9	5	3	0	7.181	6.659	2	1	0	0	0	0.422	0.587
CBO′	31	8	4	3	0	6.614	6.648	2	1	0	0	0	0.422	0.587
RFC_1	367	53	32	16	3	46.313	53.41	27	2	0	0	0	2.458	5.61
RFC_∞	638	142	57	20	3	100.96	124.76	118	88	5	0	0	31.94	41.10
MPC	274	14	5	1	0	16.205	37.448	73	2	0	0	0	3.47	10.233
ICP	676	46	12	2	0	49.374	105.37	201	7	0	0	0	9.446	27.88
IH-ICP	50	5	2	0	0	4.916	8.857	0	0	0	0	0	0	0
NIH-ICP	626	33	9	1	0	44.458	101.277	201	7	0	0	0	9.446	27.88
DAC	8	2	1	0	0	1.193	1.477	0	0	0	0	0	0	0
DAC′	7	1	1	0	0	1	1.22	0	0	0	0	0	0	0
IFCAIC	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ACAIC	2	0	0	0	0	0.024	0.22	0	0	0	0	0	0	0
OCAIC	8	2	1	0	0	1.169	1.48	0	0	0	0	0	0	0
FCAEC	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DCAEC	2	0	0	0	0	0.024	0.22	0	0	0	0	0	0	0
OCAEC	16	1	0	0	0	1.169	2.603	6	0	0	0	0	0.084	0.666
IFCMIC	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ACMIC	8	1	0	0	0	0.807	1.477	0	0	0	0	0	0	0
OCMIC	205	8	4	2	0	9.386	23.644	2	0	0	0	0	0.217	0.443
FCMEC	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DCMEC	38	0	0	0	0	0.807	4.432	0	0	0	0	0	0	0
OCMEC	135	4	1	0	0	9.386	23.247	62	1	0	0	0	1.735	8.412
IFMMIC	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AMMIC	24	3	1	0	0	2.108	4.024	0	0	0	0	0	0	0
OMMIC	250	9	4	0	0	14.096	35.343	73	2	0	0	0	3.47	10.233
FMMEC	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DMMEC	69	0	0	0	0	2.108	8.528	0	0	0	0	0	0	0
OMMEC	124	15	4	1	0	14.096	23.705	26	0	0	0	0	0.542	2.91

presents and evaluates a number of multivariate prediction models built from the design measures. In Section 4.6, we consider threats to the validity of this study.

4.1. Descriptive Statistics

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 "StdDev" state for each measure the maximum value, 75% quartile, median, 25% quartile, minimum, mean value, and standard deviation, respectively.

From this table, we can make the following observations:

- The measures that count coupling to friend classes (the F***C and IF***C measures) are all zero. That is, there are no friendship relationship in the system.

- There is little inheritance coupling, as can be seen by the low mean and standard deviation of the measures which count this type of coupling: IH-ICP, the A***C and D***C measures. Measures ACAIC and DCAEC have only one class with a non-zero value. These measures count instances where a class has an attribute whose types is an ancestor class. That is, there is both an "is-a" and a "has-a" relationship between two classes. Of course we expect this to occur infrequently (this was also observed in the UMD systems).
- There is no inheritance-based coupling to library classes. Therefore, ICP and NIH-ICP yield identical values, as do MPC and OMMIC.
- Overall, there is only very little coupling to library classes. 60% of the LALO classes exclusively interact with other LALO classes only, without having to rely on library classes at all.
- The measures OCAEC, OCMEC, and OMMEC counting export coupling to library classes are non-zero for some classes. This indicates that some library classes use non-library classes, something one normally would not expect to happen. In our case, the automatically generated classes in the LALO system, which we treat as library classes, implement a parser and use some of the 83 non-library classes.

Comparison to UMD Systems

For the UMD study, we investigated eight independent, smaller systems, whereas in the current study, we have one large system. Thus, because the LALO classes are embedded in a larger context, there is overall more coupling present in the LALO system. Especially the measures that involve method invocations have higher means and standard deviations than in the UMD systems. However, there are two exceptions to this:

- There is less aggregation coupling in the LALO system (in this paper, by aggregation we mean instances where a class has an attribute whose type is another class). In particular, there is no aggregation coupling to library classes, which is to be expected for the kinds of libraries used (IO, threading).
- Unlike in the UMD systems, there is no friendship coupling in the LALO system. This was considered bad practice and was avoided, e.g., by introducing access methods to set and retrieve values of class attributes, whenever this was required.

4.1.2. Descriptive Statistics for Cohesion Measures

Table 2 presents the descriptive statistics for the cohesion measures. We make the following observations:

- ICH, LCOM1 and LCOM2 have extreme outliers. For the LCOM measures, this is due to the presence of access methods. These methods usually only reference one

Table 2. Descriptive statistics for cohesion measures.

Measure	Max	75%	Median	25%	Min	Mean	Std Dev
LCOM1	5437	79	45	17	0	139.5904	598.25
LCOM2	4988	46	22	0	0	99.5904	547.74
LCOM3	56	7	5	3	1	5.867	6.32
LCOM4	12	6	5	3	1	4.952	2.837
LCOM5	1.25	0.750	0.620	0.489	0	0.62071	0.257
Coh	1	0.556	0.438	0.281	0	0.42850	0.231
Co	0.5	0.234	0.097	0	−1	0.11246	0.201
LCC	1	0.711	0.524	0.222	0	0.48434	0.300
TCC	1	0.574	0.333	0.181	0	0.38255	0.254
ICH	343	3	0	0	0	9.615	42.54

attribute, and therefore increase the number of pairs of methods in the class that do not use attributes in common.

- The low median and 75% quartile of ICH indicate that there are relatively few method invocations within classes: only one third of all classes have a non-zero value for ICH. This explains why LCOM3 and LCOM4 have similar distributions (similar mean and quartiles below 75%, the maximum of LCOM3 is an extreme outlier with a large impact on the standard deviation). In addition to counting pairs of methods using attributes in common like LCOM3 does, LCOM4 also takes method invocations within a class into account. Because for most classes there are no within-class method invocations, LCOM4 produces similar values to LCOM3. This was also observed in the UMD systems.

Comparison to UMD Systems

Overall, the distributions of the individual measures in terms of their mean and standard deviations are very similar to the UMD systems. Unlike the coupling measures, the cohesion measures are concerned with the internal structure of each individual class and are not strongly affected by the overall size of the systems. It is interesting to see that both studies' results show normalized cohesion measures (i.e., Coh, Co, LCOM5, LCC, TCC) with mean and median values far below 1. In light of the relatively high experience of the LALO developers, the question is now whether achieving values near 1 is a realistic expectation for such measures and, consequently, how should we interpret them?

4.1.3. Descriptive Statistics for Inheritance Measures

Table 3 summarizes the descriptive statistics for all inheritance measures. We make the following observations:

Table 3. Descriptive statistics for inheritance measures.

Measure	Max	75%	Median	25%	Min	Mean	Std Dev
DIT	3	1	1	0	0	0.831	0.852
AID	3	1	1	0	0	0.831	0.852
CLD	3	0	0	0	0	0.277	0.611
NOC	9	0	0	0	0	0.590	1.415
NOP	1	1	1	0	0	0.590	0.495
NOD	12	0	0	0	0	0.831	2.235
NOA	3	1	1	0	0	0.831	0.853
NMO	13	4	0	0	0	2.470	3.586
NMINH	127	8	3	0	0	6.542	15.99
NMA	104	15	9	8	1	12.81	12.16
SIX	1.1818	0.3333	0	0	0	0.166	0.256

- The maximum value of NOP is 1, no class has two or more parents, i.e., multiple inheritance was not used. Therefore the measures DIT, AID, and NOA produce identical values, and we removed AID and NOA from all subsequent analyses.
- NMA and NMNH show an extreme outlier (104 and 127 respectively). The two classes involved are central to the architecture of LALO. The class with NMINH=127 inherits from the class with NMA=104.
- Overall, little use of inheritance was made. This is observed in most data sets reported (Chidamber and Kemerer, 1994; Chidamber et al, 1998; Cartwright and Shepperd, 1997).

Comparison to UMD Systems

The distribution of measures are comparable between the two systems, most measures have similar means and standard deviation. Notable differences:

- NOC and NOD do vary more in the LAO systems.
- Method overriding is much more used in the LALO system. In the UMD study, the average NMO was 0.61, which is one fourth of the value of the LALO system. This may be explained by the larger experience of the LALO developers with OO and the inheritance mechanism: they are better able to design class inheritance hierarchies that exploit the opportunities for polymorphism to a larger degree.

4.1.4. Descriptive Statistics for Size Measures

Table 4 presents the descriptive statistics for size measures.

The descriptive statistics for the size measures do not show any interesting or surprising trends, the extreme outliers for measures NMImp and NMInh were already discussed in the inheritance section.

Table 4. Descriptive statistics for size measures.

Measure	Max	75%	Median	25%	Min	Mean	Std Dev
NumPara	233	18	10	7	0	16.60	28.07
NAImp	16	3	2	1	0	2.578	3.029
NMImp	108	16	11	9	1	14.61	12.61
NAInh	27	3	1	0	0	2.337	4.435
NMInh	127	8	3	0	0	6.542	15.98

Comparison to UMD Systems

In terms of the number of methods and number of parameters, the individual classes in UMD were smaller (average 12 implemented methods per class, with 11 parameters in UMD; average 16 methods per class with 17 parameters in LALO). However, the classes of the UMD systems have more attributes (average 5.4, as opposed to 2.5 in LALO).

4.2. Principal Component Analysis

For the PCA, measures that did not vary were discarded. For pairs of measures with identical values, one redundant measure was removed. PCA using the remaining measures identified ten PCs which capture 87% of the data set variance. Table 5 shows for each rotated component the loadings of the measure, with loadings larger than 0.7 set in boldface. The eigenvalue, the percentage of the data set variance each PC captures, and the cumulative variance percentage are also provided.

Since each coupling measure was 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 measures. To distinguish them, we denote the version counting coupling to library classes by appending "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 method from non-library classes.

Based on the analysis of the loadings associated with each measure within each of the rotated components, the PCs are interpreted as follows:

PC1: NumPara, NMImp, DAC, OCAIC; LCOM1, ICH, NMA: Measures NumPara, NMImp, and NMA show the highest loadings in this PC and measure the class size in terms of its methods and method parameters. This is our interpretation of this PC. LCOM1 is a count of the number of pairs of methods that do not use attributes in common; the more methods, the larger the potential number of such pairs, hence a correlation with the number of methods is plausible. Likewise, ICH, which counts within-class method invocations, is likely to increase with the number of methods. For DAC and OCAIC, which measure import aggregation coupling, the correlation to the number of methods is unexpected, and we have found no explanation for this. However, the loadings of these measures are somewhat lower than those of NumPara, NMImp, and NMA.

Table 5. Rotated component.

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10
Eigenvalue:	10.912	8.229	6.247	4.608	4.075	2.930	2.708	1.833	1.466	1.313
Percent:	21.396	16.135	12.249	9.035	7.991	5.744	5.310	3.594	2.875	2.575
CumPer.	21.396	37.531	49.779	58.814	66.805	72.549	77.859	81.453	84.328	86.903
NumPara	**0.856**	−0.083	−0.114	0.014	0.254	0.058	0.093	0.126	0.195	0.08
NMA	**0.846**	−0.119	0.079	0.065	0.274	0.143	−0.303	0.116	0.058	0.004
NMImp	**0.776**	−0.013	0.219	0.067	0.248	−0.281	−0.371	0.072	0.078	0.009
OCAIC	**0.754**	0.126	0.281	−0.4	−0.009	0.026	0.026	−0.015	−0.283	−0.132
ICH	**0.75**	−0.181	−0.105	0.09	0.173	−0.007	−0.242	−0.087	0.221	0.055
DAC	**0.749**	0.128	0.297	−0.392	−0.001	−0.033	0.022	0.007	−0.275	−0.165
LCOM1	**0.737**	−0.103	0.094	0.053	0.165	−0.189	−0.535	−0.161	0.113	0.041
NOC	0.041	−**0.891**	0.088	0.09	−0.043	0.115	0.033	−0.278	−0.04	−0.001
DMMEC	0.192	−**0.873**	0.101	0.027	0.01	0.044	−0.15	−0.079	−0.027	0.131
DCMEC	−0.148	−**0.862**	0.243	0.012	−0.046	−0.003	0.061	−0.096	−0.053	0.094
NOD	−0.04	−**0.857**	0.054	0.088	−0.041	0.129	0.016	−0.292	−0.028	−0.042
CLD	0.047	−**0.741**	−0.068	0.091	−0.03	0.163	−0.187	−0.336	−0.025	−0.076
OMMEC	0.143	−0.147	**0.828**	0.15	0.01	−0.011	−0.041	0	0.022	0.11
OCMEC	0.107	−0.111	**0.75**	0.042	0.004	−0.235	−0.109	−0.097	0.123	0.389
NMINH	0.045	0.048	−0.043	−**0.944**	−0.008	−0.018	0.039	−0.064	0.053	−0.012
NAInh	0.014	0.087	−0.105	−**0.912**	−0.072	−0.024	0.088	−0.051	0.183	0.002
RFC1	0.235	0.051	0.12	−**0.828**	0.327	−0.127	−0.087	−0.036	0.213	−0.003
NIH-ICP	0.183	0.038	0.109	−0.032	**0.959**	0.002	−0.034	0.052	0.044	0.016
ICP	0.178	0.047	0.116	−0.071	**0.95**	−0.009	−0.034	0.048	0.119	0.015
MPC	0.218	0.029	0.037	−0.104	**0.942**	0.066	−0.031	0.073	0.019	−0.023
OMMIC	0.217	0.008	−0.013	−0.064	**0.939**	0.115	−0.036	0.073	−0.077	−0.017
SIX	−0.037	0.144	0.027	0.075	−0.043	−**0.925**	−0.018	−0.045	−0.077	−0.017
NMO	0.044	0.196	0.313	0.017	0.006	−**0.863**	−0.21	−0.065	0.055	0.011
ACMIC	0.014	0.159	0.359	0.119	0.078	−**0.814**	−0.063	0.017	0.193	−0.016
DIT	−0.028	0.174	−0.186	−0.335	−0.183	−**0.724**	0.139	−0.088	−0.033	−0.002
MPC_L	0.129	−0.016	0.02	0.029	0.006	−0.02	−**0.95**	−0.006	0.028	0.067
ICP_L	0.131	−0.013	0.026	0.037	0.008	−0.027	−**0.949**	−0.009	0.036	0.07
RFC1_L	0.03	−0.017	0.108	−0.45	0.027	−0.036	−**0.801**	0.043	−0.145	0.01
TCC	0.084	0.174	0.037	0.096	0.095	0.241	−0.042	**0.914**	−0.006	−0.059
LCOM5	0.076	−0.216	0.166	−0.022	0.002	0.066	−0.139	−**0.896**	−0.048	−0.074
Co	0.19	0.093	0.052	0.035	0.109	0.03	−0.021	**0.878**	−0.023	−0.021
Coh	−0.151	0.228	−0.186	0.035	−0.026	−0.014	0.177	**0.86**	0.055	0.066
LCC	0.261	0.209	0.193	0.024	0.129	0.324	−0.196	**0.78**	−0.096	−0.131
IH-ICP	−0.028	0.096	0.095	−0.436	0.052	−0.122	−0.005	−0.034	**0.824**	−0.011
AMMIC	0.03	0.158	0.376	−0.31	0.141	−0.349	0.031	0.014	**0.709**	−0.045
OMMEC_L	−0.008	0.084	0.252	0.013	−0.006	−0.044	−0.159	−0.034	−0.019	**0.892**
OCMEC_L	−0.107	−0.367	0.363	0.01	−0.03	−0.051	−0.021	−0.009	−0.079	**0.784**
OCAEC	0.03	0.099	0.658	0.008	0.103	−0.148	−0.063	−0.056	0.18	0.575
CBO_L	−0.071	−0.126	0.555	0.028	−0.028	−0.279	−0.444	−0.01	−0.089	0.254
DAC′	0.641	0.08	0.152	−0.535	0.065	0.057	0.001	0.093	−0.263	−0.182
LCOM3	0.234	−0.35	−0.033	0.059	0.022	−0.556	−0.169	−0.556	0.142	0.169
LCOM4	−0.057	−0.277	0.059	0.082	−0.039	−0.637	0.089	−0.549	0.103	0.161
RFC∞_L	−0.068	−0.213	0.173	−0.286	0.102	−0.597	−0.288	−0.118	0.055	0.135
OCMIC_L	0.082	−0.138	0.392	0.15	0.046	−0.494	−0.582	−0.115	0.157	0.133
NAImp	0.626	0.025	0.092	0.047	0.141	0.082	−0.647	−0.041	−0.049	−0.087
LCOM2	0.613	−0.189	−0.029	−0.015	0.036	−0.124	−0.61	−0.285	−0.026	0.085
CBO	0.198	−0.458	0.632	−0.16	0.402	−0.137	−0.002	−0.014	0.084	0.074
CBO′	0.196	−0.47	0.635	−0.13	0.414	−0.073	−0.024	−0.011	0.059	0.069
NOP	−0.031	0.213	−0.072	−0.366	−0.226	−0.649	0.07	−0.088	0.256	0.068
RFC∞	0.1	−0.023	−0.183	−0.69	0.336	−0.056	−0.285	0.01	0.142	0.033
OCMIC	0.649	0.072	−0.026	−0.185	0.249	0.159	0.184	0.239	−0.141	0.011

PC2: DCMEC; DMMEX, CLD, NOC, NOD: DCMEC and DMMEC measure export coupling to descendent classes, NOC, NOD, and CLD are counts of the number of children, descendents, and depth of inheritance hierarchy below the class. Naturally, the more descendents a class has, the higher the export coupling to these descendents is likely to be. We interpret this PC to measure the depth of the inheritance tree below a class.

PC3: OCMEC and OMMEC: these are measures of export coupling to "other" classes, i.e., classes not related via friendship or inheritance.

PC4: NAInh, RFC_1, NMINH: NAInh, and NMINH count the number of inherited attributes and methods, respectively, and we interpret this PC to measure the inherited size of the class. RFC_1 counts the number of methods (including inherited ones) of a class, plus the number of methods involved by these. Hence, RFC_1 is expected to be larger for descendent classes.

PC5: MPC, ICP, NIH-ICP, OMMIC: These are measures of method invocations to non-library classes.

PC6: ACMIC, DIT, NMO, SIX: This PC is not easy to interpret. The number of overriding methods (NMO), and the specialization index (SIX) are concerned with method overriding. Method overriding is more likely to occur in classes situated deep in the inheritance hierarchy, as is the type of ancestor import coupling measured by ACMIC. As NMO and SIX have the highest loadings, we interpret this PC to measure the amount of method overriding.

PC7: RFC_1_L, MPC_L, ICP_L: Thee measures count import coupling thorough method invocations to library classes.

PC8: LCOM5, Coh, Co, LCC, TCC: These are normalized cohesion measures.

PC9: IH-ICP, AMMIC: Measure method invocation import coupling to ancestor classes.

PC10: OCMEC_L, OMMEC_L: Measures of export-coupling to library classes.

Comparison to UMD Systems

Six of the ten PCs in LALO can be matched to PCs with identical interpretation in UMD: PC2, and PC5-PC9. PC3 and PC10 in LALO, counting export coupling to library- and non-library classes, respectively, have one corresponding PC in UMD counting export coupling to both library- and non-library classes. Similarly, the measures of PC1 in LALO formed two separate PCs in UMD: class size and import aggregation coupling to non-library classes. For PC4 (inherited size), no similar PC was observed in UMD.

For coupling measures, we again see the separation between coupling to library and coupling to non-library classes, that is, no PC contains a mixture of the two (in UMD such mixtures occurred in two minor PCs only).

For cohesion, we again have the separation into normalized and non-normalized measures. However, in LALO, the "non-normalized cohesion measures" are more strongly correlated to size, which was not the case in UMD.

Further observations:

- As in UMD, most of the PCs identified are coupling dimensions (5 out of 10), we have one cohesion dimension (PC8), two inheritance dimensions, (PC2 and PC6), two size

Table 6. Univariate analysis results for coupling measures.

	Coupling to non-library classes only					Coupling to library classes only				
Measure	R^2	Coeff.	Std Err	$\Delta\Psi$	p-value	R^2	Coeff.	Std Err	$\Delta\Psi$	p-value
CBO	0.3171	0.404	0.084	5.493	**<.0001**	0.1688	2.108	0.47	3.445	**<.0001**
CBO′	0.3421	0.447	0.091	4.439	**<.0001**	0.1688	2.108	0.47	3.445	**<.0001**
RFC_1	0.296	0.024	0.007	2.142	**0.0005**	0.2037	0.535	0.165	2.005	**0.0013**
RFC_∞	0.678	0.0051	0.0018	1.568	**0.0070**	0.0801	0.016	0.005	1.645	**0.0006**
MPC	0.2159	0.099	0.03	1.73	**0.0008**	0.2349	0.687	0.231	3.34	**0.0029**
ICP	0.3121	0.051	0.012	2.569	**0.0001**	0.2377	0.257	0.087	3.612	**0.0031**
IH-ICP	0.0662	0.080	0.030	1.174	**0.0078**			All zero		
NIH-ICP	0.3358	0.071	0.017	2.741	**<.0001**	0.2377	0.257	0.087	3.612	**0.0031**
DAC	0.0308	0.257	0.124	1.461	**0.0386**			All zero		
DAC′	0.0312	0.304	0.147	1.448	**0.0391**			All zero		
OCAIC	0.0281	0.241	0.121	1.429	**0.0467**			All zero		
OCAEC	0.0925	0.588	0.242	1.462	**0.0152**			All zero		
ACMIC	0.0376	0.339	0.152	1.226	**0.0261**			All zero		
OCMIC	0.0618	0.1015	0.036	2.009	**0.0052**	0.1689	3.179	1.019	4.089	**0.0018**
DCMEC	0.0021	0.0345	0.0661	1.044	0.6019			All zero		
OCMEC	0.1320	0.093	0.035	1.365	**0.0086**	0.1355	1.82	0.458	2.395	**<.0001**
AMMIC	0.0758	0.19	0.076	1.395	**0.0121**			All zero		
OMMIC	0.1938	0.096	0.03	1.562	**0.0014**	0.2349	0.687	0.231	3.34	**0.0029**
DMMEC	0.0192	0.138	0.092	1.204	0.1349			All zero		
OMMEC	0.1034	0.051	0.017	1.542	**0.0025**	0.0732	0.826	0.377	1.454	**0.0282**

dimensions (PC1 and PC4). This is almost identical to UMD, where we had 9 out of 13 coupling dimensions and only one size dimension.

- The smaller number of coupling dimension identified is explained by the fact that several types of coupling in UMD do not occur in LALO, for instance, friendship coupling, inheritance-based or aggregation coupling to library classes.

As in UMD, we again see that measures purporting to measure different structural attributes end up in the same PC (inheritance-based coupling and inheritance in PC2 and PC6), size and cohesion and coupling in PC1.

4.3. Univariate Logistic Regression Analysis

4.3.1. Univariate Regression Analysis for Coupling Measures

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 3.1), and the statistical significance (p-value), which is the probability that the coefficient is different from zero by chance. Measures with little or no variance are not considered in the table.

Table 7. Univariate analysis results for cohesion measures.

Measure	R^2	Coeff.	Std Err	$\Delta\Psi$	p-value
LCOM1	0.2078	0.02	0.006	3.942	**0.0004**
LCOM2	0.0949	0.011	0.005	1.613	**0.0249**
LCOM3	0.0499	0.135	0.063	1.499	**0.0316**
LCOM4		Not significant			0.7159
LCOM5		Not significant			0.8061
Coh		Not significant			0.3786
Co		Not significant			0.2619
LCC	0.0357	1.712	0.696	1.672	**0.0140**
TCC		Not significant			0.3726
ICH	0.3000	0.700	0.214	3.525	**0.0010**

As we can see in Table 6, most of the measures have a significant relationship to fault-proneness (at $\alpha = 0.05$). The exceptions are DCMEC and DMMEC of PC2, which count export coupling to descendent classes.

For all significant measures, the regression coefficients are positive. This confirms hypotheses H-IC and H-EC stated in Section 2.4 that classes with higher import or export coupling are more likely to be fault-prone.

The impact of export coupling on fault-proneness is weaker than that of import coupling: the export coupling measures mostly have lower coefficient and $\Delta\Psi$s than their import coupling counterparts.

The CBO measures are the only measures that count both import and export coupling: Their relationship to fault-proneness is particularly strong (high coefficients and $\Delta\Psi$s).

Comparison to UMD Systems

Similar to the results obtained with the UMD systems, all import coupling measures with sufficient variation were found to be significant predictors of fault-proneness.

However, in the UMD systems, none of the export coupling measures were found to be significantly related to fault-proneness in the expected direction. In the LALO system, most of the export coupling measures that do vary also are indicators of fault-proneness. Maybe, because of the weaker impact of export coupling, and because there was overall less coupling in the UMD systems, thus resulting in less variation, we failed to find a statistically significant relationship to fault-proneness in those systems.

4.3.2. Univariate Regression Analysis for Cohesion Measures

The results from univariate analysis with cohesion measures are summarized in Table 7.

Five of the ten cohesion measures are significant at $\alpha = 0.05$:

- The significant measures LCOM1, LCOM2, and LCOM3 show positive correlation coefficients. This indicates that the higher the values of these measures, the more fault-

prone the class is likely to be. This is consistent with our Hypothesis H-Coh that low cohesion is bad design.

- As was discussed in (Briand, et al. 1998), the mathematical properties of ICH make it unlikely to be measuring cohesion. ICH possesses properties of a complexity measure. With this information, the positive coefficient of ICH is reasonable: the higher ICH (and thus class complexity), the more fault-prone the class.

- For LCC, the positive coefficient indicates that fault-proneness increases with class cohesion, which is counterintuitive. As we will see below, LCC is also positively correlated to size, which may explain this unexpected relationship to fault-proneness.

All other measures (LCOM4, LCOM5, Coh, Co, TCC) are not significant predictors. This includes all normalized cohesion measures, except LCC. We consider normalization a necessary property of a cohesion measure (Briand, et al. 1996), but it appears not to be effective from a prediction point of view.

Comparison to UMD Systems

From the cohesion measures found significant in the LALO system, LCOM3 and ICH were also significant in the UMD systems. Coh was significant in the UMD systems, but not in the LALO system. All other measures, including LCOM1, LCOM2, and LCC, were not significant in the UMD systems.

4.3.3. Univariate Regression Analysis for Inheritance Measures

The results from univariate analysis for the inheritance measures are summarized in Table 8. We make the following observations:

- Only three measures are significant indicators of fault-proneness: DIT, NMO, and NMA.

- The negative coefficient of DIT indicates that fault-proneness decreases with the depth of the class. That is, classes located deeper in the inheritance hierarchy are less fault-prone. An explanation for this is that, in the LALO system, classes situated deeper in the inheritance hierarchy provide only implementations for a few specialized methods, and are therefore less likely to contain faults than classes at the root. This stems from a deliberate strategy, from the LALO development team, to place as much functionality as possible near the root of the inheritance hierarchies.

- For NMO and NMA, the positive coefficient indicates that fault-proneness increases with the number of overriding or adding methods. This is consistent with our hypotheses H-OVR and H-NMA.

Table 8. Univariate analysis results for inheritance measures.

Measure	R^2	Coeff.	Std Err	$\Delta\Psi$	p-value
DIT	0.0414	−0.656	0.245	0.572	**0.0074**
CLD	0.0085	0.366	0.314	1.251	**0.2431**
NOC		Not significant			**0.7064**
NOP		Not significant			0.5587
NOD		Not significant			0.8280
NMO	0.0479	0.152	0.057	1.724	**0.0082**
NMINH		Not significant			0.3254
NMA	0.1075	0.112	0.037	3.925	**0.0021**
SIX		Not significant			0.7766

Comparison to UMD Systems

Measures NMO and NMA appear to be consistently good indicators of fault-proneness in both systems.

In the UMD-Systems, all inheritance measures were significant indicators of fault-proneness. Besides NMO and NMA, only DIT is a significant predictor in LALO. However, its relationship to fault-proneness is in the opposite direction. In the UMD systems, classes deeper down in the inheritance hierarchy were likely to be more fault-prone, as stated in hypothesis H-Depth. The student developers of the UMD systems had no previous experience with object-orientation, most of the LALO developers have worked on OO systems before. This could indicate that the added cognitive complexity of using inheritance may cause critical problems to beginners (such as the UMD systems' student developers) but is not an issue for an experienced team using a well-defined inheritance strategy, such as the LALO developers.

A related result is that NOC was found significant in the UMD systems, but not in the LALO system, even though the measure has higher variance in the LALO system.

To summarize, the inheritance measures do not appear to be stable quality indicators of OO design, but seem to rather indicate how well the development team can deal with inheritance. In conjunction with fault data, the inheritance measures help to determine if the way inheritance is used in the system causes the developers problems. On their own, however, the existent inheritance measures are not able to distinguish, in an objective manner, proper use of inheritance from improper use.

4.3.4. Univariate Regression Analysis for Size Measures

Table 9 shows the univariate analysis results for the size measures.

Except for the NAInh and NMInhy counting inherited methods and attributes, all size measures have a significant relationship to fault-proneness, NMImp showing the strongest impact.

In UMD, all size measures were significant, including NAInh and NMInh, which we explain by the lack of experience of the students with the inheritance mechanism.

Table 9. Univariate analysis results for size measures.

Measure	R^2	Coeff.	Std Err	$\Delta\Psi$	p-value
NumPara	0.1191	.0916	.0265	2.562	**0.001**
NAImp	0.1306	.3845	.1112	2.412	**0.001**
NMImp	0.1487	.1335	.0347	2.750	**0.000**
NAInh		Not significant			0.352
NMInh		Not significant			0.325

Table 10. Correlation of coupling measures to size.

	Coupling to non-library classes only		Coupling to library classes only	
Measure	Rho	p-value	Rho	p-value
CBO	0.3937	**0.0002**	0.3152	**0.0037**
CBO′	0.3939	**0.0002**	0.3152	**0.0037**
RFC_1	0.5198	**<.0001**	0.3101	**0.0043**
RFC_∞	0.3162	**0.0036**	0.1982	**0.0724**
MPC	0.3969	**0.0002**	0.3569	**0.0009**
ICP	0.4237	**<0.0001**	0.3601	**0.0008**
IH-ICP	0.0901	0.4178	All zero	
NIH-ICP	0.4645	**<.0001**	0.3601	**0.0008**
DAC	0.5208	**<.0001**	All zero	
DAC′	0.5083	**<.0001**	All zero	
OCAIC	0.4866	**<.0001**	All zero	
OCAEC	0.1331	0.2303	All zero	
ACMIC	0.2826	**0.0096**	All zero	
OCMIC	0.5167	**<.0001**	0.4201	**<.0001**
DCMEC	−0.1664	0.1326	All zero	
OCMEC	0.1984	**0.0722**	0.3455	**0.0014**
AMMIC	0.0996	0.3701	All zero	
OMMIC	0.3591	**0.0009**	0.3569	**0.0009**
DMMEC	0.0498	0.6548	All zero	
OMMEC	0.4688	**<.0001**	0.2343	**0.0330**

4.4. Correlation to Size

In this section we analyze the correlation of the design measures to the size of the class. We measure the size of the class design as the number of methods that are implemented in the class. The correlation of each design measure with size is expressed in terms of its Spearman Rho coefficient with size, and the corresponding p-value.

4.4.1. Correlation to Size for Coupling Measures

The correlations of the coupling measures to class size are shown in Table 10.

The non-library versions of measures DAC, DAC′, OCAIC, and OCMIC (all in PC4), and RFC_1, have the strongest relationship to size. For OCMIC this may be explained because the

Table 11. Correlation of cohesion measures to size.

Measure	Rho	*p*-value
LCOM1	0.8069	**<.0001**
LCOM2	0.3087	**0.0045**
LCOM3	0.2416	**0.0277**
LCOM4	0.1036	0.3515
LCOM5	0.0870	0.4373
Coh	−0.1259	0.2569
Co	0.3491	**0.0012**
LCC	0.4093	**0.0001**
TCC	0.2013	0.0680
ICH	0.5431	**<.0001**

more methods a class has, the more method parameters there are, the higher OCMIC is likely to be (which counts the number of method parameters that have an "other" class as their type). RFC_1 includes a count of the number of methods; therefore a correlation to size is expected here. However, the Rho coefficients for all these measures are only barely above 0.5, i.e., the relationship to size is not very strong. For all other coupling measures, significant Rho coefficients are below 0.5, that is, there is at most a moderate correlation to size for these measures, if any. Some of these correlations, however, may partly explain some of the relationship to fault-proneness reported below in Section 4.3. When building multivariate models, in Section 4.5, we will verify whether coupling measures add a predictive power and show practical usefulness.

Comparison to UMD Systems

Common to the LALO and UMD systems is that overall a statistically significant correlation to size is present, but it is weak.

4.4.2. Correlation to Size for Cohesion Measures

Table 11 shows, for each cohesion measure, the Spearman's Rho coefficient with size (the number of methods implemented in the class).

LCOM1 is very strongly correlated to size, which may explain its significant relationship to fault-proneness. The correlation to size is due to presence of access methods in some classes. As discussed in (Briand, et al. 1998), access methods typically reference one class attribute only. Thus, a large number of pairs of methods that use not attributes in common can be formed with these access methods. The more methods in a class (i.e., the larger the class), the larger the potential number of such pairs.

We saw that the presence of access methods has strong impact on some of the measures in PCA, univariate analysis and correlation to size. Several authors suggested modifications to the definitions of these measures, to better account for the presence of access methods

Table 12. Correlation of inheritance measures with size.

Measure	Rho	*p*-value
DIT	0.0005	0.9966
AID	0.0005	0.9966
CLD	−0.0346	0.7563
NOC	−0.0434	0.6972
NOP	0.0200	0.8575
NOD	−0.0410	0.7130
NOA	0.0005	0.9966
NMO	0.3748	**0.0005**
NMINH	−0.1488	0.1793
NMA	0.7740	**<.0001**
SIX	0.2445	0.0259

(see (Briand, et al. 1998) for a summary of these discussions). However, for a completely automated static analysis of the systems, there must be a way to automatically recognize access methods (e.g., a consistently applied naming scheme that uniquely identifies access methods by "set_" and "get_" name prefixes, which must not be used for any other methods). If there is no way to safely identify access methods, as this is the case in LALO, these modified measures are difficult to implement and use in practice.

ICH and LCC have a moderate positive correlation to size. For ICH, which is a count of method invocations within class, this is understandable: the more methods a class has, the more method invocations are likely to occur within the class (even though this could not be observed in the UMD systems). For LCC, however, we have no explanation for the correlation to size. Most of the other measures have a significant, but weak correlation to size.

Comparison to UMD Systems

In the UMD systems, we also found that many measures had a significant, but weak correlation to size (especially Co, TCC, and LCC). ICH displays the strongest correlation among all cohesion measures in both the UMD and LALO systems. Such a relationship is explained, as discussed above, but the way the measure is defined. Such correlations just confirm the argument that ICH cannot probably be considered as a valid cohesion measure.

4.4.3. Correlation to Size for Inheritance Measures

Table 12 shows Spearman's Rho correlation coefficients with size for all inheritance measures. Only NMA, the number of methods added to a class, is strongly correlated to size (which is to be expected). All other measures are at most weakly correlated to size. This is identical to what was observed in the UMD systems.

4.5. Multivariate Regression Analysis

In this section we investigate a number of multivariate prediction models, built from different subsets of the measures we have analyzed so far (Section 4.5.1). The main goal of these models is to build accurate prediction models for class fault-proneness by using all the design measures available together. We first build a model based on size measures only. We then allow coupling, cohesion, and inheritance measures to enter the model and compare its goodness of fit with the size-only model. It is important to note that coupling and cohesion measures are usually available at a later stage in the design process. We evaluate the accuracy of the most plausible prediction model we found using a 10-cross-validation process (Section 4.5.2), and discuss possible applications of such a model in Section 4.5.3.

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 question:

- 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.

Size Model

Because the number of size measures considered is small, a backward stepwise selection process was used to build a prediction model based solely on size measures:

Table 13. Model I based on design size measures only.

Measure	Coefficient	Std. Err.	p-value
NumPara	.0577	.0267	0.031
NAImp	.2655	.1154	0.021
Intercept	−.3016	.3847	0.433

We will refer to this model as “Model I”. Because there are strong correlations present among the size measures we considered, only two measures ended up in this model. Thus, the goodness of fit of the model cannot be expected to be very high: the loglikelihood

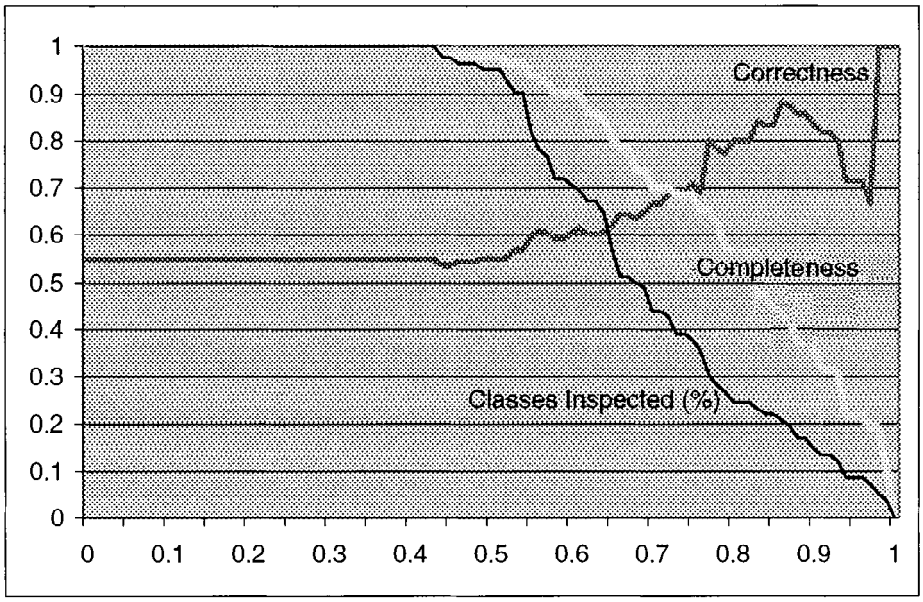

Figure 2. Correctness/completeness graph for Model I.

of the model is -74.05, and $R^2 = 0.1639$. The conditional number for this model is $\sqrt{1.446672/0.5533} \approx 1.61$, which does not indicate any problems.

We applied this model to the 83 classes of the LALO system in order to compare the predicted probability 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 a certain threshold, p_0. We can then calculate correctness and completeness of this classification, as described in Section 3.2.4. The graph in Figure 2 shows the model's correctness and completeness (vertical axis), as a function of the classification threshold p_0, which ranges from 0 to 1 on the horizontal axis. The graph additionally shows the number of classes that are classified fault-prone, as a percentage of the total number of classes (i.e., classes inspected).

A good tradeoff between correctness and completeness is achieved at a cutoff value of 0.73, where completeness and correctness values are both about 68%. At that threshold, 32 of the 83 classes (39%) are selected for inspection, of which 22 classes are actually faulty and contain 89 out of 131 faults found.

We will now compare the results of Model I with the comparable model in the UMD study. We put in the first column the covariates that were selected in LALO and UMD, respectively. We then provide an interpretation of what dimensions these measures capture based, when possible, on principal component analysis. It is to be expected that covariates will vary across models, when built in different environments. But we want to determine whether some common dimensions seem to be important, regardless of the environment. Though this just represents two studies, they are sufficiently different (see Section 3.3)

Table 14. Comparison of UMD and LALO studies for Model I

LALO	UMD	Interpretation of PC
NumPara	NumPara	Size in terms of methods and their parameters
NAImp		No clear PC, data members locally defined (not inherited)
	NM	No clear PC, number of methods locally defined and inherited
	NMpub	No clear PC, Number of public methods

to make any commonality worth noticing. From Table 14, we can see that design size measures related to methods and their parameters seem to play a major role in predicting fault-proneness. A related study looking at the cost of development (Briand and Wüst, 1999) finds similar conclusions.

The goodness of fit achieved by the UMD model for the UMD systems was 60% correctness and 67% completeness (LALO: 68% correctness, 68% completeness). Thus, we have about the same level of completeness as the LALO model, and a slightly lower correctness.

Model Built from Coupling, Cohesion, and Inheritance Measures

Next, we build a model allowing all coupling, cohesion, and inheritance measures significant at $\alpha = 0.25$ to enter the model, following a forward selection process as described in Section 3.2.4. Here it is:

Table 15. Model II—Based on coupling, cohesion, and inheritance measures.

Measure	Coeff.	Std. Err.	p-value
CBO′	.379	.174	0.030
ICP_L	.272	.111	0.014
ICH	.466	.241	0.053
NIH-ICP	.069	.0262	0.009
OCAIC	−1.91	.658	0.004
OCMIC	.136	.0554	0.014
Intercept	−2.56	.791	0.001

This model (Model II) consists of six measures: five import coupling measures, and the "cohesion" measure ICH, which, as discussed, rather has properties of a class complexity measure. None of the other cohesion and inheritance measures are included, reflecting the results found in univariate analysis that many of these measures are not significant indicators of fault-proneness in the LALO system. Of the six model covariates, two covariates could not be clearly attributed to any PC. The remaining four covariates cover the three PCs PC1, PC5, and PC7. There are two measures from PC1 present (ICH and OCAIC), and they are somewhat correlated. This may explain the negative sign of OCAIC, which was positive in the univariate analysis. Changes in coefficient magnitude or sign are common in multivariate regression analysis and can be due to multicollinearity and adjustment effects between

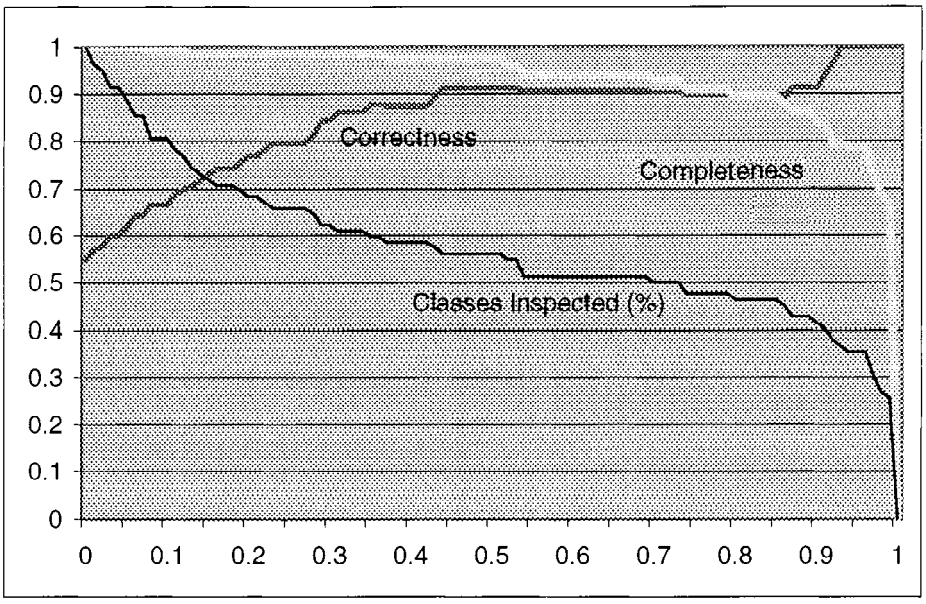

Figure 3. Correctness/completeness graph for Model II.

covariates. However, the conditional number of the model is $\sqrt{3.423092/0.118273} \approx 5.38$, still well below the critical threshold of 30.

The model has a high goodness of fit (loglikelihood = -27.94, $R^2 = 0.69$). Figure 3 shows the correctness and completeness graphs for model II.

Model II performs very well: correctness and completeness values above 90% are achieved for thresholds in the range 0.45 to 0.85. For instance, at $p_0 = 0.7$, of the 42 classes predicted fault-prone, 38 actually are fault-prone (90% correctness), which contain 122 faults (93% completeness, there are 131 faults in total). The high goodness of fit indicates that the design 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.

Like for Model I, we now compare the models of the LALO and UMD studies. We can see from Table 16 that both studies, despite their differences, showed that import coupling, through method invocations, from library and non-library classes have a significant role on fault-proneness. This is therefore likely to be a structural property that should be considered as a potential fault-proneness factor when building such models. Inheritance properties appear as significant covariates in the UMD studies and not in the LALO data, as expected and discussed in Section 4.3.3.

The goodness of fit of the UMD model for the UMD systems was 81% correctness and 94% completeness. Like for the LALO study, the inclusion of coupling and inheritance

Table 16. Comparison of UMD and LALO studies for Model II.

LALO	UMD	Interpretation of PC
CBO′		No clear PC
ICP_L	NIHICP_L	Import coupling through method invocations from library classes
ICH		Class size in terms of methods and their parameters
NIH-ICP	RFC_∞	Import coupling through method invocations to non-library classes
OCAIC		Class size in terms of methods and their parameters
OCMIC		No clear PC—coupling
	NOP	No clear PC
	RFC_1_L	No clear PC
	NMI	No clear PC
	FMMEC	Export coupling to friend classes
	CLD	Depth of inheritance hierarchy

measures clearly improved the model fit compared to the model based on design size measures only.

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 design size measures (significant at 0.25) to enter the model:

Table 17. Model III—Allowing all measures to enter the model.

Measure	Coefficient	Std. Err.	*p*-value
CBO′	.683	.176	0.000
ICP_L	.379	.124	0.002
OCAIC	−3.91	1.12	0.000
NumPara	.190	.0567	0.001
NAImp	1.61	.652	0.014
ICH	.484	.240	0.045
Intercept	−5.41	1.47	0.000

This model (Model III) consists of six covariates: three import coupling measures, two size measures, and ICH. Compared to Model II, the two coupling measures OCMIC and NIH-ICP were replaced by two size measures: NumPara and NAImp (which happen to be exactly the size measures that were selected in Model I).

The results above could indicate that some of the coupling measures 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.

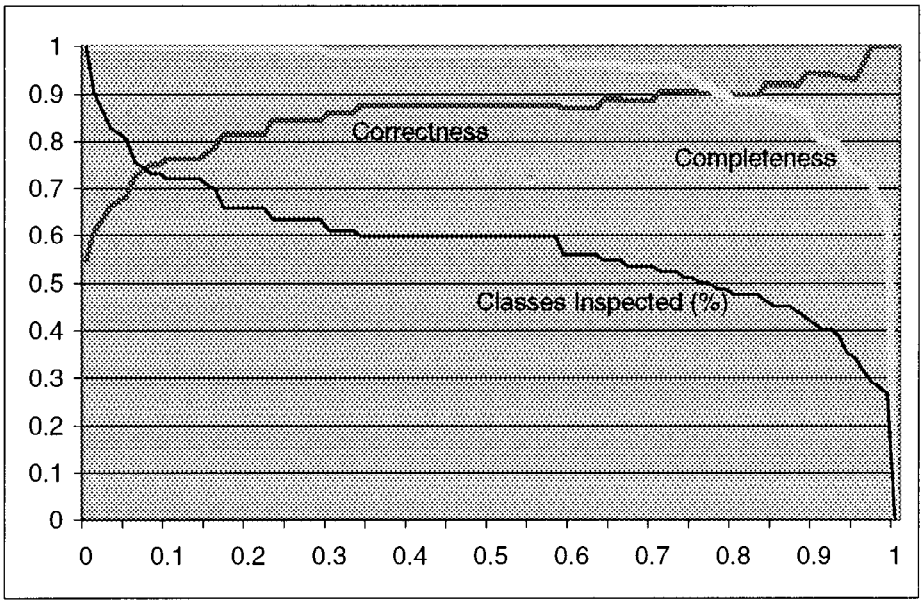

Figure 4. Correctness/completeness graph for Model III.

The loglikelihood of this model is $= -24.99$, and the $R^2 = 0.72$, i.e., slightly higher than the model without size measures. The conditional number of $\sqrt{3.23936/0.057749} \approx 7.59$ still does not indicate a problematic amount of multicollinearity.

The correctness and completeness curves for Model III are very similar to that of Model II. At the threshold of 0.73, 44 of the 83 classes are selected for inspection, of which 40 are actually faulty (91% correctness), and we achieve 95% completeness (124 out of 141 faults). The accuracy of the corresponding model in the UMD study was 90% correctness and 70% completeness. Overall, the results we obtained in the LALO study confirm the trends observed in the UMD study: The two main fault-proneness factors are size and import coupling and very high levels of correctness and completeness can be achieved.

4.5.2. Model Evaluation

Based on our discussions in Section 3.2.5, and therefore for all practical reasons regarding the usefulness of our predictive model, we will further investigate Model II. Recall that this model is not based on size measures and shows a similar accuracy to Model III. However, its accuracy 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) of model II. Then more realistic values for completeness and correctness can be devised.

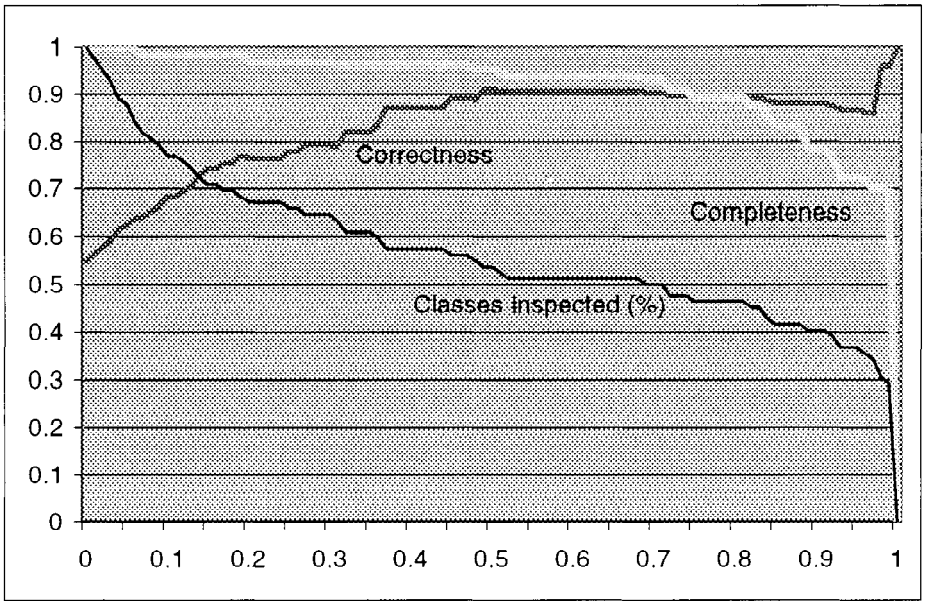

Figure 5. Correctness/completeness graph for Model II in 10-cross validation

For the 10-cross validation, the 83 data points were randomly split into ten partitions of roughly equal size (seven partitions of 8 data points each, three partitions of 9 data points). 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 83 classes a predicted probability of their fault-proneness. Figure 5 shows the resulting correctness and completeness graphs.

The correctness and completeness graphs have not changed much more from Figure 3. The difference lies mostly with the lower correctness curve for thresholds larger than 0.85, i.e., outside the range of practically useful values of p_0. For thresholds between 0.5 and 0.75, the model achieves correctness and completeness values above 90%, where about 50% of the classes would be selected for inspections. Hence, the accuracy of the model in cross-validation is not much lower than the model goodness-of-fit determined in Section 4.5.1. This indicates a high stability of the model.

In the UMD study, the best model achieved 92% completeness and 78% correctness when performing cross-validation. Like for the LALO study, the loss in accuracy was small when compared to the model goodness-of-fit.

4.5.3. Model Applications

They are many ways in which the prediction models above can be used. However, before one can do so, it is important to demonstrate that, in a given application domain and environment,

such models can be stable. Thus, from project to project, the models can be applied with a reasonable level of confidence. Such stability can only be obtained if enough project data have been collected to provide stable distributions on the design measures' ranges. As discussed above, many of the differences across regression results between the UMD systems and the LALO system came from significant differences in data distributions, some of the design measures being more or less variable in one data set or the other. However, despite the sharp differences with respect to almost every project factor, e.g., application domain, experience, it is interesting to note that some strong commonalities have been clearly identified.

One important application is to use the fault-proneness models to help focus inspections or testing. Taking the example of inspection planning 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 a graph such as the one in Figure 6 to determine how many percent of faults in the system we can expect to find by inspecting a certain percentage of the system code, and which classes should be inspected. On the X-axis, we plotted the 83 system classes, sorted in decreasing order of their predicted fault-proneness (the predicted probabilities were taken from the 10-cross validation described above). The lower curve is the cumulative size of the classes, in percent (measured in terms of the number of methods in a class). At $X = 30$ the value of the curve is 50%, which means that the 30 classes with highest predicted fault-proneness contain 50% of the methods of the system. The upper curve is cumulative number of actual faults in the classes, in percent. At $X = 30$, this curve is at $Y \approx 75\%$, which means the fist 30 classes contain 75% of the actual faults.

As an example of an application of this graph, assume we have resources available to inspect 40% of all methods. From Figure 6 we can tell that the first 12 classes with highest predicted fault-proneness roughly contain 40% of all methods. If we select these classes for inspections, then we can expect to find a maximum of about 57% of all faults in them. The graph also tells us that by increasing the resources by 5 points to inspect 45% of all methods, the maximum percentage of faults we can find grows to about 67%.

A practical problem with the application of the graph in Figure 6 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 stable 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 the UMD study, the analogous graph for that data set had a very similar form to the one presented here, a result of the fact that the models in the UMD study and the present study have a similar predictive power. But in general, whether the shape of the upper curve is stable across projects in a given environment needs to be verified individually for each environment.

In Figure 6, 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, except that the slope for the first ten classes with highest predicted fault-proneness is steeper than for the remaining classes (which means that these ten classes have an above-average size). But overall, the discrepancy between the upper and lower curve indicates that our model does not simply assign higher fault-proneness to

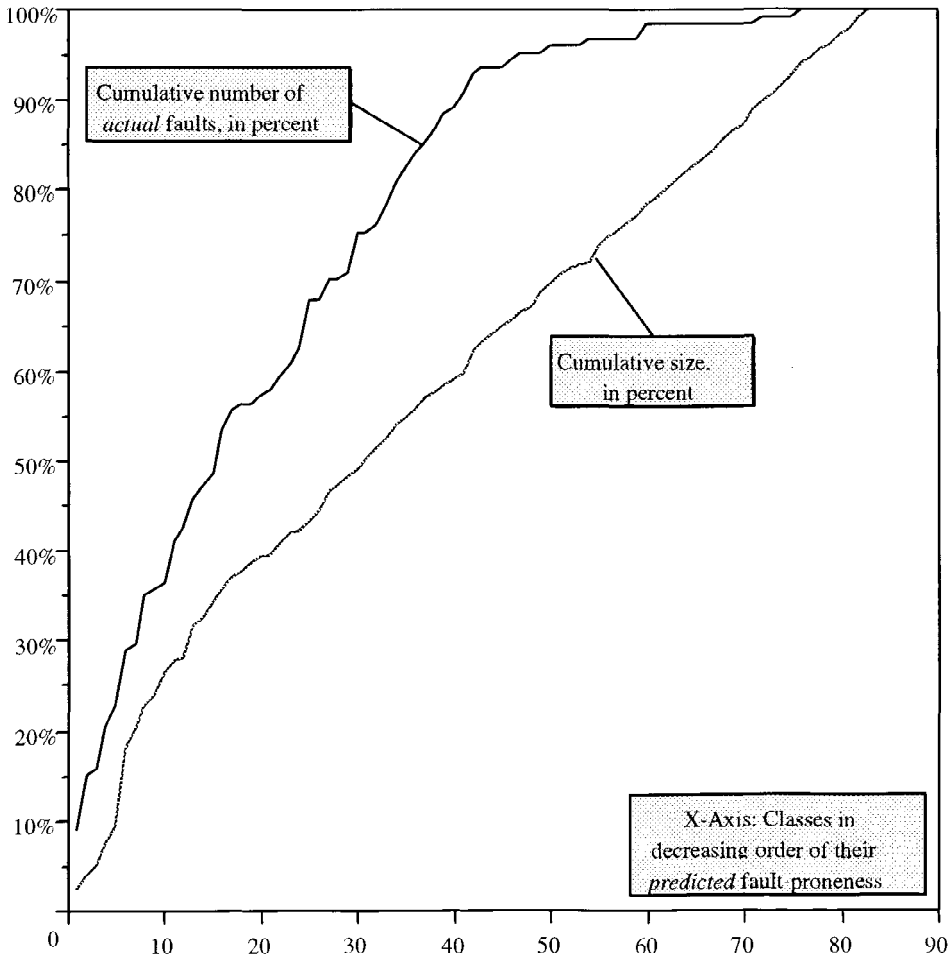

Figure 6. Application of the prediction model.

larger classes, but that the design measures capture an additional effect beyond size that has an impact on the fault-proneness of classes. This confirms the results obtained from multivariate analysis which showed that the design measures can help model an effect on fault-proneness which is complementary to the one of class size.

Last, as another example of application, once one knows the design properties that matter with respect to OO software quality in a given organization and application domain, then such measurement can be used as quality benchmark to assess and compare products. For example, such experiences are reported in (Mayrand and Coallier, 1996) where the code of acquired products is systematically measured to identify systems or subsystems that strongly depart from previously established measurement benchmarks. Such benchmarks

can be built based on existing operational software, which has shown to be of good quality. New software can then be analyzed by comparing, for example, import coupling class distributions with the established benchmark. Any system part that would show a strong departure from the benchmark distribution could, for example, be further inspected to investigate the cause of the deviation. If no acceptable justification can be found then the acquisition manager may decide to require some corrective actions before the final delivery or for future releases of the software. This is particularly important when systems will be maintained over a long period of time and new versions produced on a regular basis.

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 settings.

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

4.6.1. Construct Validity

The construct validity of the dependent variable has been demonstrated in Section 2.1. We still have to address the question to which degree the coupling, cohesion, and inheritance measures used in this study measure the concepts they purport to measures. In (Briand et al, 1998) and (Briand et al, 1998), the coupling and cohesion measures were theoretically validated against properties for coupling and cohesion measures proposed in (Briand et al, 1996). These properties are one of the more recent proposals to characterize coupling and cohesion in a reasonably intuitive but rigorous 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. (1997). 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.

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 operation. 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, 1997; Briand et al, 1997) 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 present study is a case study, therefore, we have the usual threats to the external validity: The models we built are validated for the LALO system and development environment only. However, LALO is a midsize system developed by experienced, professional developers and is therefore somewhat representative of some of the OO development in industry.

5. Conclusions

Based on the comparison of the results from the two studies performed so far, we provide a number of recommendations. If one intends to build quality models of OO designs, coupling will very likely be an important structural dimension to consider. More specifically, a strong emphasis should be put on method invocation, import coupling since it has shown to be a strong, stable indicator of fault proneness. We also recommend that the following aspects be measured separately since they capture distinct dimensions in our data sets: import versus export coupling, coupling to library classes versus application classes, method invocation versus aggregation coupling. As far as cohesion is concerned and measured today, it is very likely not a very good fault-proneness indicator. This stems mainly from the current difficulty to define clearly the concept and measure it. One illustration of this problem is that two distinct dimensions are captured by existing cohesion measures: normalized versus non-normalized cohesion measures. As opposed to the various coupling dimensions, these do not look like components of a vector characterizing class cohesion, but rather as two fundamentally different ways of looking at cohesion. Which one is actually measuring the concept of cohesion, if any? Similarly, inheritance measures appear not to be consistent indicators of class fault-proneness. Their significance as indicators strongly depends on the

experience of the system developers and the inheritance strategy in use on the project. The combined use of inheritance measures and fault data to assess the use of inheritance in a project/organization is an important topic of further research.

As discussed, one important application OO design measures is to build quality benchmarks to assess OO software products that are newly developed or under maintenance, e.g., in the context of large scale software acquisition and outsourcing (Mayrand and Coallier, 1996). Ideally, the measures used for this purpose should be consistent indicators of software design problems, and their relationships to external quality attributes, e.g., fault-proneness, should not strongly depend on the characteristics of the environment where the software was developed. We therefore conclude that, in the current stage of knowledge, measures of import coupling appear to be particularly well suited to develop simple and practical quality benchmarks.

Since import coupling appears to be a consistent driver for fault-proneness, one might conclude that a system built of only one—big—class, displaying no import coupling whatsoever, should have to be not fault-prone at all. like in all empirical work, the relationships we established here are valid only for a certain population, which is usually unknown. In our case, we would vaguely characterize this population as 'reasonably object-oriented, mid-sized systems'. A system consisting of only one class obviously cannot be called object-oriented. Therefore, we would not expect our prediction models to be valid for such systems. In addition, it should be recalled that import coupling is only one of many drivers of fault-proneness. Other product properties such as size, and also process and resource related issues like experience of people affect fault-proneness.

When using design measures to build predictive models of fault-prone classes, we have consistently obtained, across two studies, high levels of classification accuracy (i.e., around 90% correctness and completeness). This suggests that design measurement-based models may be very effective instruments for quality evaluation and control of OO systems by aggregating fault-proneness predictions for classes. However, the overall results of our studies also tell us that the validity of fault-proneness models may be very context-sensitive. In a given environment, their stability has to be assessed and analyzed across systems so that conditions (e.g., application domain) under which models are stable can be identified. Although some of the patterns and relationships presented above seem stable across very different study settings and systems, the replication of such studies is necessary in order to build over time a credible body of empirical knowledge on which to base the quality assessment of OO designs and systems.

In our future work we will also investigate alternative modeling techniques aimed at predicting counts of defects in classes (as opposed to the predicted probability of fault detection in logistic regression) and thereby facilitate the use of design measurement-based prediction models. Techniques such as Poisson or negative-binomial regression analysis (Long, 1997) should be investigated, depending on the defect distributions at hand.

Acknowledgments

We would like to thank Michael Ochs for developing the analyzers used in this study, Michel Lavallée for making the LALO study possible, and the LALO developers without whom

such quality data could not have been collected. The Concerto2/AUDIT tools and the FAST technology were developed by SEMA Group, France, and are now marketed by Verilog. We also want to thank SEMA group for providing us with this tool suite.

Appendix

Table 18 to Table 20 describe the coupling, cohesion, and inheritance measures used in this study. We list the acronym used for each measure, informal definitions of the measures, and literature references where the measures originally have been proposed. The informal nature language definitions of the measures 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, 1998; Briand et al, 1998), where we also perform a systematic comparison of these measures, and analyze their mathematical properties.

Table 18. Coupling measures.

Name	Definition	Source
CBO	Coupling between object classes. According to the definition of this measure, a class is coupled to another, if methods of one class use methods or attributes of the other, or vice versa. CBO is then defined as the number of other classes to which a class 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.	(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 another class as their type.	(Li and Henry, 1993)
DAC′	The number of different classes that are used as types of attributes in a class.	(Li and Henry, 1993)
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	As ICP, but counts invocations of methods of ancestors of classes (i.e., inheritance-based coupling) only.	(Lee et al, 1995)

(*continued*)

Table 18. Coupling measures (*continued*).

Name	Definition	Source
NIH-ICP	As ICP, but counts invocations to classes not related through inheritance.	(Lee et al, 1995)

IFCAIC
ACAIC
OCAIC
FCAEC
DCAEC
OCAEC
IFCMIC
ACMIC
OCMIC
FCMEC
DCMEC
OCMEC
IFMMIC
AMMMIC
OMMIC
FMMEC
DMMEC
OMMEC
	These coupling measures are counts of interactions between classes. The measures distinguish the relationship between 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* issuing another class.
	  • EC: Export coupling: count interactions where class *d* is the used class.
		(Briand et al, 1997)

Table 19. Cohesion 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)
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 defined as the number of connected components of G.	(Hitz and Montazeri, 1995)

(*continued*)

Table 19. Cohesion measures (*continued*).

Name	Definition	Source
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 $\text{Co} = 2(\lvert E\rvert - (\lvert V\rvert - 1))/((\lvert V\rvert - 1)(\lvert V\rvert - 2))$.	(Hitz and Montazeri, 1995)
LCOM5	Consider a set of methods $\{M_i\}(i = 1, \ldots, m)$ accessing a set of attributes $\{A_j\}(j = 1, \ldots, a)$. Let $\mu(A_j)$ be the number of methods which reference attribute A_j. Then $\text{LCOM5} = \frac{1}{a}\left(\left(\sum_{j=1}^{a} \mu(A_j)\right) - m\right)/(1 - m)$.	(FAST, 1997)
Coh	A variation on LCOM5: $\text{Coh} = \left(\sum_{j=1}^{a} \mu(A_j)\right)/(m \cdot a)$	(Briand et al, 1998)
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 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 $m_1, \ldots, m_n$, such that m_i and m_{i+1} are connected for $i = 1, \ldots, n-1$, 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 20. Inheritance 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)
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.	(FAST, 1997)

(*continued*)

Table 20. Inheritance measures.

Name	Definition	Source
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, 1995)
NOC (number of children)	The number of classes that directly inherit from a given class	(Chidamber and Kamerer, 1991; Chidamber and Kemerer, 1994)
NOP (number of parents)	The number of classes that 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, 'grandchildren', and so on)	(Lake and Cook, 1994; Tegarden et al, 1995)
NOA (number of ancestors)	The number of classes that given class directly or indirectly inherits from.	(Tegarden et al, 1995)
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)
NMINH (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 methods in a class that the class inherits from its ancestors and does not override.	(Lorenz and Kidd, 1994)
SIX (specialization index)	SIX is NMO * DIT / (NMO + NMA + NMINH)	(Lorenz and Kidd, 1994)

Table 21. Size measures.

Name	Definition
NMIMP	The number of methods implemented in a class (non-inherited or overriding methods)
NMINH	The number of inherited methods in a class, not overridden
NAIMP	Then number of attributes in a class (excluding inherited ones). Includes attributes of basic types such as strings, integers.
NAINH	The number of inherited attributes in a class
NUMPAR	Number of parameters. The sum of the number of parameters of the methods implemented in a class.

In order to compare how these measure relate to size we also consider a number of size measures defined in Table 21. These measurements are frequently used in publications and available tools, so no definite source or author can be given.

Notes

1. This simplification was also performed for the UMD study, though we neglected to mention it explicitly there.

2. We also attempted to use a backward selection process by pre-selecting variables using principal component analysis. Following a commonly used procedure, the highest loading variables for each principal component was selected and then the backward selection process run on this set of variables. The results, however, showed the goodness of fit of the models thus obtained to be poorer than the models obtained from the forward stepwise procedure. We will therefore stick with the forward procedure in this paper.

References

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

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

Briand, L., Bunse, C., Daly, J., and Differding, C. 1997. An experimental comparison of the maintainability of object-oriented and structured design documents. *Empirical Software Engineering* 2(3). Also available as Technical Report ISERN-96-14.

Briand, L., Bunse, C., and Daly, J. 1997. An experimental evaluation of quality guidelines on the maintainability of object-oriented design documents. *Proceedings of Empirical Studies of Programmers: Seventh Workshop (ESP 7)*. Also available as Technical Report ISERN-97-02.

Briand, L., Devanbu, P., and Melo, W. 1997. An investigation into coupling measures for C++. *Proceedings of ICSE '97*. Boston, USA.

Briand, L., Daly, J., Porter, V., and Wüst, J. 2000. Exploring the relationships between design measures and software quality in object-oriented systems. *Journal of Systems and Software*, 51:245-273. Also available as Technical Report ISERN-98-07.

Briand, L., Daly, J., and Wüst, J. 1999. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering*, 25(1). Also as Technical Report ISERN-96-14.

Briand, L., Daly, J., Wüst, J. 1998. A unified framework for cohesion measurement in object-oriented systems. *Empirical Software Engineering Journal* 3(1): 65–117. Also available as Technical Report ISERN-97-05.

Briand, L., Morasca, S., and Basili V. 1996. Property-based software engineering measurement. *IEEE Transactions of Software Engineering* 22(1): 68–86.

Briand, L., and Wüst, J. 2001. The impact of design properties on development cost in object-oriented systems. Technical Report ISERN 99 16. *Forthcoming in the proceedings of IEEE Metrics*, London, England.

Bieman, J. M., and Kang, B.-K. 1995. Cohesion and reuse in an object-oriented system. *Proc, ACM Symp. Software Reusibility (SSR '94)*, 259–262.

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

Chidamber, S. R., and Kemerer, C. F. 1991. Towards a metrics suite for object oriented design. A. Paepcke, (ed.) *Proc. Conference on Object-Oriented Programming: Systems, Languages and Applications (OOPSLA '91)*. Published in *SIGPLAN Notices* 26(11): 197–211.

Chidamber, S. R., and Kemerer, C. F. 1994. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering* 20(6): 476–493.

Chidamber, S., Darcy, D., and Kemerer, C. 1998. Managerial use of metrics for object-oriented software: An exploratory analysis. *IEEE Transactions on Software Engineering* 24(8): 629–639.

Cartwright, M., and Shepperd, M. 1997. An empirical investigation of an object-oriented software: An exploratory analysis. *Technical Report, Department of Computing.* Bournemouth University, UK.

Dunteman, G. 1989. *Principal Component Analysis.* SAGE Publications.

FAST Programmer's Manual. 1997. SEMA Group, France.

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

Hosmer, D. W. and Lemeshow, S. 1989. *Applied Logisitc Regression.* John Wiley & Sons.

Hitz, M., and Montazeri, B. 1995. Measuring coupling and cohesion in object-oriented systems. *Proc. Int. Symposium on Applied Corporate Computing*. Monterray, Mexico.

Khoshgoftaar, E., and Allen, E. 1997. Logistic regression modeling of software quality. TR-CSE-97-24. Florida Atlantic University.

Li, W., and Henry, S. 1993. Object-oriented metrics that predict maintainability. *J. Systems and Software* 23(2): 111–122.

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

Lorenz, M., and Kidd, J. 1994. Object-oriented software metrics. *Prentice Hall Object-Oriented Series*. Englewood Cliffs, N.J.

Lee, Y.-S., Liang, B.-S., Wu, S.-F., and Wang, F.-J. 1995. Measuring the coupling and cohesion of an object-oriented program based on information flow. *Proc. International Conference on Software Quality*. Maribor, Slovenia.

Long, S. 1997. Regression models for categorical and limited dependent variables. *Advanced Quantitative Techniques in the Social Sciences Series*. Sage Publications.

Mayrand, J., and Coallier, F. 1996. System acquisition based on software product assessment. *Proceedings of ICSE '96*. Berlin, Germany, 210–219.

Munson, J., and Khoshgoftaar, T. 1989. The dimensionality of program complexity. *Proceedings of ICSE'89*. Pittsburgh, 245–254.

Rosenberg, J. 1997. Some misconceptions about lines of code. *Proceedings of the 4th International Software Metrics Symposium (Metrics '97)*. 137–142.

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

Tegarden, D. P., Sheetz, S. D., and Monarchi, D. E. A software complexity model of object-oriented systems. *Decision Support Systems* 13(3–4): 241–262.

ISO/IEC DIS 14598-1. Information technology—Product evaluation, Part 1: General Overview.

Hakim Lounis is professor at the computer science department of the University of Quebec at Montreal (UQAM). He holds a Ph.D in computer science from Université de Paris-Sud (Orsay, France). From 1996 to June 2000, he was researcher at the Computer Research Institute of Montreal (CRIM), where he led since august 1999 the software engineering and knowledge engineering team. He was also, since 1997, adjunct professor at the university of Laval (Quebec city). During the ten last years, he has been involved in different research, development and technology transfer projects in Europe and Canada.

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.

Lionel Briand joined the Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, where he is now associate professor. Before that he was the software quality engineering department head at the Fraunhofer Institute for Experimental Software Engineering, Germany. Lionel also worked as a research scientist for the Software Engineering Laboratory, a consortium of the NASA Goddard Space Flight Center, CSC, and the University of Maryland. But his first experiences are in the trenches, designing and developing large software systems, and he has, over the years, acted as a consultant to many industrial and government organizations. He holds a Ph.D. degree in Computer Science, with high honors, from the University of Paris XI, France.

Lionel has been on the program, steering, or organization committees of many international, IEEE conferences. He also belongs to the steering committee of METRICS and is on the editorial boards of Empirical Software Engineering: An International Journal and IEEE Transactions on Software Engineering. His research interests include: object-oriented analysis and design, inspections and testing in the context of object-oriented development, quality assurance and control, project planning and risk analysis, technology evaluation.