Modeling Development Effort in Object-Oriented Systems Using Design Properties

Lionel C. Briand, *Member*, *IEEE*, and Jürgen Wüst

Abstract—In the context of software cost estimation, system size is widely taken as a main driver of system development effort. But, other structural design properties, such as coupling, cohesion, and complexity, have been suggested as additional cost factors. In this paper, using effort data from an object-oriented development project, we empirically investigate the relationship between class size and the development effort for a class and what additional impact structural properties such as class coupling have on effort. This paper proposes a practical, repeatable, and accurate analysis procedure to investigate relationships between structural properties and development effort. This is particularly important as it is necessary, as for any empirical study, to be able to replicate the analysis reported here. More specifically, we use Poisson regression and regression trees to build cost prediction models from size and design measures and use these models to predict system development effort. We also investigate a recently suggested technique to combine regression trees with regression analysis which aims at building more accurate models. Results indicate that fairly accurate predictions of class effort can be made based on simple measures of the class interface size alone (mean MREs below 30 percent). Effort predictions at the system level are even more accurate as, using Bootstrapping, the estimated 95 percent confidence interval for MREs is 3 to 23 percent. But, more sophisticated coupling and cohesion measures do not help to improve these predictions to a degree that would be practically significant. However, the use of hybrid models combining Poisson regression and CART regression trees clearly improves the accuracy of the models as compared to using Poisson regression alone.

Index Terms—Cost estimation, object-oriented measurement, empirical validation.

1 Introduction

The problem of cost estimation in software engineering has been addressed by many articles and books [3]. Research shows that many factors can affect the development cost of software projects, including human, organization, and process factors. However, one important factor is the size of the product to be developed [3]. Many measures of size have been proposed, ranging from source lines of code to functional size measures, such as function points. Size may not only be measured in various ways, counting different types of artifacts, but at different points in time ranging from requirements analysis to coding. In addition to size, many other product properties have been mentioned as potential fault-proneness and cost factors, such as complexity, cohesion, and coupling [5], [6]. Choosing appropriate measurement for size or any other properties depends on the objective of measurement, the time at which measurement will be taken during the development process, and the type of information available at that time in a given organization. We focus on the impact of object-oriented design properties on development effort.

- *L.C. Briand is with the Systems and Computer Engineering Department, Carleton University, 1125 Colonel By Drive, Ottawa, ON, K1S 5B6, Canada. E-mail: briand@sce.carleton.ca.*
- *J. Wüst is with the Fraunhofer Institute for Experimental Software Engineering, Sauerwiesen 6, 67661 Kaiserslautern, Germany. E-mail: wuest@iese.fhg.de.*

Manuscript received 1 Jan. 2001; revised 1 Apr. 2001; accepted 15 May 2001. Recommended for acceptance by S. Pfleeger.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 114697.

This paper has two related but distinct objectives. First, we simply want to better understand the nature of the relationships, if any, between design product properties and development effort in object-oriented (OO) systems. Does size, based on design information, show a strong relationship to effort? What additional impact is shown by properties such as complexity, cohesion, or coupling? Is their impact of any practical significance? Second, we would like to get an idea of how accurately effort predictions can be based on such design information, at different stages of the design. Although that answer depends on the specific context of application, e.g., application domain or stability of development process, our aim here is to determine whether this is likely to be feasible at all. We will base our results on a system developed in C++ (described in [28]) for which effort data at the class level has been collected. The fine granularity level of the data collection allows us, as described below, to provide initial answers to the questions stated above. Another contribution of this paper is the proposed analysis and modeling procedure. As further described below, investigating the above-mentioned issues is a complex endeavor. Our approach is practical, precisely defined so as to lead to easy replications of the study, and accurate in terms of modeling complex relationships between effort and design structural properties.

The paper starts by a description of the research method we use here, also including the data set from which the results are drawn. Then, related work is briefly presented and compared to the objectives of our paper. Then, Section 5

provides the analysis results in a structured manner. Conclusions and future work are presented in Section 6.

2 Research Method

2.1 General Method

Answering the questions stated above regarding the relationships between design properties and development effort requires a careful analysis of real-project data. No theory regarding the many design factors that can potentially impact effort can fully address such research objectives. They can only be tackled through empirical studies, such as the one presented here. As a basis for this paper, we will use effort data collected on a C++ system (Section 2.3) and will analyze the source code of that system to extract design information using an analyzer we have developed. We will then seek to analyze the relationships between design properties and effort. Though we have initial hypotheses [5], [6] regarding how design properties impact development effort, we cannot a priori make any assumption regarding the general form of the relationships between properties and effort, the interactions of these properties, and their relative impact on effort. Because of the exploratory nature of the study, we will use a number of modeling techniques to investigate the issues previously mentioned.

2.2 Study Variables and Unit of Analysis

For each class, effort was collected and accounted for designing and coding the class, documenting, testing, and correcting it. So, only unit and integration testing effort was collected for each class, but system or acceptance testing effort could not be easily associated with any particular class.

It makes sense to see system and acceptance testing effort as dependent on the whole system's properties rather than any particular class subset. Looking at testing effort involving the whole system would force us to use the system as a unit of analysis and require data on a large number of systems, which would be extremely difficult to obtain. As a consequence, our analysis will be incomplete in the sense that some development effort (i.e., system and acceptance test) will not be taken into consideration. However, such an analysis is still useful as it determines the impact of design properties on a substantial part of the development effort and does not require data on a large number of projects.

As a consequence of our choice of unit of analysis, the impact of commonly considered cost factors, such as programming language, application domain, experience of developers, etc., cannot be considered in this study. Such factors vary across projects, but not within projects of limited size such as this one. Therefore, the scope of the study is confined to the impact of design properties on effort as described above.

2.3 Data Collection

The product under study is a graphical and interactive editor for music scores, called LIOO [11]. The system was implemented in C++ on a Linux platform. It was developed by staff members at the University of Florence using an iterative development process described in [28]. The total development effort for the system was 30 person-months and involved 11 developers. In this study, we have analyzed the version resulting from the first iteration. Seven developers were involved in the creation of this first iteration. This version is composed of 103 classes, totaling 17KLOC (nonblank, noncomment lines of code). These figures exclude reused libraries (XVGAlib for the GUI) and automatically generated code (yacc/lex) that are not within the scope of this study.

For the effort data collection, the developers were instructed to maintain a list of the classes they worked on and record the effort spent on each class. This includes all effort for designing and implementing, testing, debugging, and documenting that can be specifically allocated to individual classes. Developers updated their effort logs at least once each day. The total effort thus recorded was 393.3 person-hours. As mentioned above, this figure excludes system/acceptance testing effort that cannot be assigned to individual classes (data on the amount of system/acceptance testing effort is not available to the authors). Design properties were collected through a static analyzer developed at Fraunhofer IESE, which collects most of the design measures proposed to date in the literature from source code (see [5], [6] for an in-depth discussion and the short definitions for the measures in the Appendix). We classified measures as measuring one of the following attributes: size, coupling, cohesion, or inheritance.

These measures are based on the following design information:

- for size measures: system classes, their attributes (including their types) and methods (including parameter arguments and their types);
- additionally, for inheritance and coupling measures: relationships between classes (inheritance, association, aggregation), relationships between methods (method invocations);
- additionally, for cohesion measures: relationships between methods and class attributes (which method reads or modifies which class attributes).

It is clear that such information is inherently different from what is usually associated with source code measurement (counts of statements, depth of nesting level, cyclomatic complexity, ratio of comment lines, etc.). As stated above, the design information we use here to quantify design properties can potentially be extracted from various UML diagrams, some of which are available at early analysis stages (e.g., class diagrams), while other diagrams such as collaboration or sequence diagrams showing detailed interactions between objects are typically artifacts of late low-level design.

For the sake of convenience, we used the source code as an analyzer because it was readily available and since, like for any postmortem study of that type, the source code is readily available and complete. This has two main consequences on the research presented here. On the one hand, our results may be cleaner as we get the final information regarding the design properties of the system classes, whereas early design diagrams may have shown discrepancies with the final product. On the other hand, it is clear

that any prediction capability of the models presented here will be optimistic as they will not account for the uncertainty and changes occurring during a project where design decisions may be constantly changing and under refinement. In typical project situations, however, such changes in design would warrant updating predictions and imply project replanning. To conclude, the results presented here should be interpreted as a feasibility study assessing what can possibly be achieved under optimal conditions (e.g., stable development process and methodologies, stable early design artifacts) rather than realistic figures of effort estimation accuracy. This is, however, a necessary first step.

A final issue concerning the data collection is that the study presented in this paper is a postmortem analysis. The data collection procedure was designed with the goal to collect class level development effort, which suits our research goals. In a setting where the data collection procedures can be planned upfront, additional cost factors such as developer experience could have been taken into account.

2.4 Data Analysis Procedure

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

1. We will first look at the frequency distributions of our design measures across the classes of the system. This is necessary in order to explain some of the results that follow and differences across studies, including future replications of this study.
2. We will then use Principal Component Analysis (PCA) [16] to determine the dimensions captured by our design measures. It is common in software engineering, like in other fields, to have much collinearity between measures capturing similar underlying phenomena. In particular, PCA will help us better interpret the meaning of our results in the subsequent steps.
3. Univariate regression analysis looks at the relationships between each of the design measures we investigate and effort. This is a first step to identifying potential effort predictors to be used in the next step and to identifying what types of measurements are significantly related to effort. Ordinary Least Squares (OLS) regression is the most commonly used regression technique for cost modeling. However, since our dependent variable is always positive and is highly nonnormal in its distribution (skewed), we will use a log-linear model assuming Poisson distributed effort predictions: Poisson regression. This choice will be further discussed in the next section. Effort predictions will then be Poisson distributed and always positive. All bivariate relationships will be checked for overinfluential observations that could bias the result, e.g., the significance of the result depends on the presence of one observation. This is important as we want our results to be stable.
4. Multivariate analysis also looks at the relationships between design measurement and effort, but considers design measures in combinations, as covariates in a multivariate model predicting effort. Here, we will also use Poisson regression combined with a forward stepwise variable selection procedure[1] [17]. In addition, here we use a method to improve the models' predictive accuracy by combining Poisson regression and regression trees (an adaptation of [30]). The goal of multivariate analysis is to determine what levels of predictive accuracy can be achieved in terms of effort estimation during subsequent stages of design. In addition, it tells us what kind of design measurement seems to play a more predominant, practically significant role for effort prediction. All results will be checked for the presence of overinfluential observations.
5. Apply cross-validation [31] in order to get a more realistic estimate of the predictive power of the multivariate prediction models when they are applied to data sets other than those the models were derived from. In short, a V-cross-validation divides the data set into V parts, each part being used to assess a model built on the remainder of the data set. In our study, we will divide our 103 observations data set into 10 randomly selected parts and perform a 10-cross-validation.
6. When we apply the prediction model to estimate project effort, we obtain just one predicted effort value. Though we may be particularly lucky in our case study and get a good project effort prediction, this is unsatisfactory as we know there is uncertainty in such a prediction. We use a simulation technique called Bootstrapping [27] to build, for example, a 95 percent confidence interval for the predicted system effort. As discussed below, this will shed more light on the uncertainty attached to the point estimates that could be expected from our effort model.

3 Modeling Techniques

We will here briefly introduce and motivate the modeling techniques used in this study and aforementioned.

3.1 Poisson Regression

Our dependent variable, effort, is always positive and, although the average effort is modest (3.86 hours), the distribution is skewed to the right (the 10 percent and 90 percent percentiles are 0.2 and 12.7, respectively), beyond what a normal distribution could reasonably model. For that reason, log-linear ordinary least-squares (OLS) regression models are frequently used in software engineering cost-estimation models [3]. For example, one can fit the regression equation $\ln y = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$. Thus, the logarithmic transformation can address the distribution problem mentioned above and also decreases the weight of extreme observations with high values in the dependent variable. However, this logarithmic transformation introduces a systematic bias in the predictions, i.e., the sum of predicted and actual effort values over all observations are not equal. This is due to the fact that $E(ln\ y)$, the expected value of $ln\ y$, tends to be lower than

1. We tried different variable selection strategies, e.g., backward selection strategies using only variables with highest loadings in PCA (each capturing a specific dimension). We observed no notable improvement.

$ln\ E(y)$ (Jensens inequality).[2] But, a log-linear model predicts $E(ln\ y)$, whereas we wish to obtain an unbiased prediction of $E(y)$. Unbiased prediction is a crucial property in our application context since the sum of the predicted effort over all classes is used as an estimate for the total project effort.

To compensate for this bias, we have to model y directly using a nonnormal distribution, e.g., Poisson distribution. The Poisson distribution is usually used to model rare events generated by a Poisson process, as measured by nonnegative integers. Though effort is, in theory, not discrete, it is measured in a discrete form (i.e., in our case, the minimum time measurement unit was three minutes (0.05 person hours—for some derived but otherwise empty classes whose full implementation was deferred to later versions). In addition, large effort values are rare and, as mentioned above, the effort distribution shows similar characteristics to the Poisson distribution. It is important to note that those are typical characteristics of effort distributions and are not specific to this data set [26]. From a practical perspective, as we will see below, this unusual approach to modeling effort predictions yields accurate, unbiased results based on cross-validation.[3]

A set of modeling techniques, known as Generalized Linear Models, has been devised (see [25] for a thorough discussion) in order to cope with situations where relationships are not linear and the error distributions are not normal. In our study, we will use a generalized linear model (GLM) with a logarithm link function (log *effort* is linearly related to design measures) and a Poisson error distribution. This is also known as the Poisson regression model [23].

In Poisson regression, the dependent variable y is assumed to have Poisson distribution with parameter μ, i.e.,

$$\Pr(y \mid \mu) = \frac{e^{-\mu}\mu^{y}}{y!}.$$

The parameter μ is both the expected value and variance of y: $\mu = \mathrm{E(y)} = \mathrm{Var(y)}$, an important characteristic of the Poisson distribution known as equidispersion.

In the Poisson regression model, y is assumed to be Poisson distributed with the conditional mean[4] being a function of the independent variables $\mathrm{X} = \mathrm{x_1}, \ldots, \mathrm{x_n}$, given by the regression equation

$$\hat{\mu} = e^{\beta_0+\beta_1 x_1+\cdots+\beta_n x_n}.$$

The probability distribution of y, given $\mathrm{x_1}, \ldots, \mathrm{x_n}$, is therefore

$$\Pr(y \mid x_1, \cdots, x_n) = \frac{e^{-\hat{\mu}}\hat{\mu}^{y}}{y!}.$$

For a given data set, the regression coefficients $\beta_0, \ldots, \beta_n$ can be estimated through maximization of a likelihood function based on the above probability distributions.

3.2 Regression Trees

CART Regression tree analysis [10], as opposed to regression analysis, is a data mining technique that does not require any functional form assumption. It simply builds a partition tree of the data set. The partition tree is built in such a way that its terminal nodes are more and more consistent with respect to the dependent variable (our study: effort). Consistency is measured in terms of effort variability (least absolute deviation [10]). The tree is also built so that a minimal number of observations are present in the terminal nodes (our study: 10) so that we can implement the procedure presented in Section 3.3 to combine regression trees and Poisson models. Each path of the tree represents a prediction rule that, in our case, relates the design measures to effort.

For example:

NMIMP>5.5 AND NMIMP≤9.5 AND NA≤6 AND
NM>24.5 => Median Effort = 1.0

This rule states that, when the number of methods locally defined (NMIMP), the total number of methods (NM), and the number of attributes (NA) are within a certain range, then the median effort can be predicted to be of a certain value (the mean can be derived as well). Regression trees are good at modeling nonlinear structures in the data but are not particularly good at modeling linear relationships. Other specificities are that they can model local trends, i.e., trends only present in a subset of the data, and interactions between covariates, i.e., design measures, in a simple manner. In a way, regression trees are complementary to regression models in terms of the structures they model. Therefore, we will attempt to combine Poisson regression and regression trees as specified below.

Absolute deviation of a tree T (AD(T)) is mentioned above as being the main accuracy criterion used to assess regression trees. It is defined, in our application context, as follows:

$$AD(T) = \frac{1}{N}\sum_{n=1}^{N} \mid e_n - T(x_n) \mid,$$

where e_n is the actual effort of observation n, x_n is the vector of design measures (used as predictors) for observation n, $T(x_n)$ is the predicted effort for observation n, assuming the regression tree T is used as a prediction model, and N is the total number of observations in the learning sample used to build the tree. There are different ways to use this definition of AD to assess the accuracy of a tree. As our goal here will be to use AD to compare trees, as opposed to assessing their overall accuracy in a perfectly realistic manner, we will use a simple evaluation procedure called resubstitution [10] to assess the tree goodness-of-fit. This simply computes the average absolute deviation for each one of the observations x_n that were used to build the tree using, as a prediction $T(x_n)$, the mean value of the terminal node where it belongs. The average AD across all observations in T will then be computed and compared with the average AD of the data set before building the tree.

2. Jensen's inequality: For any random variable X, if g(X) is a convex function, then $\mathrm{E(g(X))} \geq \mathrm{g(E(X))}$.

3. Most of the software engineering cost estimation literature is concerned with sizing entire projects. When there is no need to add up predicted values having an unbiased estimator is not a crucial property, and the use of log-linear models with normal errors may be appropriate.

4. Poisson regression assumes: $\mathrm{E(y|x_1, \ldots, x_n)} = \mathrm{Var(y|x_1, \ldots, x_n)}$. This assumption was tested on our data set and no severe overdispersion was noted for the multivariate models (for univariate models, overdispersion is present, as will be shown in Section 5.4). When this is the case in other data sets, the reader is invited to use a generalization of Poisson regression referred to as negative binomial regression [23].

Another interesting feature of regression trees is that procedures to assess variable importance have been defined. Though we do not wish here to go into details (provided in [10]), these procedures look at the *relative* importance of all the predictor variables that are being used in terms of their capability to explain variations in effort in a tree. The principle is to look at the improvement in goodness-of-fit (i.e., AD) that are obtained through a variable while attempting to build a regression tree, taking into account every node of the tree where the variable is used for splitting.

3.3 Combining Regression Trees and Poisson Regression Analysis

Adapting some of the recommendations in [30], we will combine the two modeling techniques in a hybrid model by following the process below:

- Run regression trees by specifying that terminal nodes should contain at least 10 observations—this number is somewhat arbitrary but it should be sufficiently large to capture significant trends. For larger data sets, one can afford to use larger numbers.
- Add dummy variables (binary) to the data set by assigning observations to terminal nodes in the regression trees, i.e., assign 1 to the dummy variable for observations falling in its corresponding terminal node. There are as many dummy variables as terminal nodes in the tree.
- Run stepwise Poisson regression using both design measures and the dummy variables as potential covariates.

This process takes advantage of the modeling power of Poisson regression while still using the specific structures that regression trees can capture. As shown in [3], there exist other ways to combine regression trees and regression analysis, but experience has shown they are not nearly as effective.

3.4 Assessing and Comparing Models' Accuracy

A usual way to compare cost models is to look at the Magnitude of Relative Error (MRE) or the Absolute Relative Error (ARE) of their effort predictions.[5] In this study, we will look at these criteria at two distinct levels: class effort prediction and system effort prediction. For each model investigated, we will have 103 class effort MREs and AREs, either resulting from the fit of the model or its predictions from a 10-cross-validation. They are calculated as follows: For a class i, $i = 1, \ldots, 103$, let eff_i be its actual effort and $\hat{eff}_i$ the predicted effort. Then,

$$ARE_i = \mid eff_i - \hat{eff}_i \mid, \text{ and } MRE_i = \mid eff_i - \hat{eff}_i \mid / eff_i.$$

5. Here, we prefer to use the MRE and ARE to assess the predictive power of our models over other goodness-of-fit statistics applicable to ML estimation such as the loglikelihood or pseudo-R^2. The MRE and ARE are independent of the modeling techniques used; they are expressed in the units of the dependent variable and are therefore easier to interpret and allow for straightforward comparisons between models.

Class MREs/AREs can be easily compared across models by performing a paired statistical test checking whether differences between models are statistically significant. Both parametric (paired t-test) or nonparametric (Wilcoxon T test) tests can be used [17]. The paired t-test typically works better when population distributions are approximately normal and tends to be conservative otherwise. Because we do not enforce the models we compare to be nested, we will perform 2-tailed tests at a level of significance of $\alpha = 0.05$.

At the system level, after performing cross-validation, we will only have one MRE value per model since the predicted system effort is the result of summing up class predicted efforts as follows:

$$ARE = \left| \sum_i (eff_i - \hat{eff})_i \right| \text{ and } MRE = ARE / \sum_i eff_i.$$

The expectation is that, for unbiased models, over and underprediction of individual class estimates mostly cancel each other out and yield an unbiased system effort estimate. However, having one predicted effort value is unsatisfactory as we know there is uncertainty in such a prediction. We would like, for example, to have a 95 percent confidence interval for such predictions so that model users would be able to make more informed decisions knowing the uncertainty of the prediction. One way to obtain such a confidence interval is to use bootstrapping [27], a nonparametric, simulation-based technique to draw statistical inferences from (small) samples. In our case, we build the 95 percent error interval of the MRE/ARE for system effort prediction, following the bootstrapping procedure below:

1. Repeat 1,000 times: randomly sample, *allowing replacement*, 103 class MRE/ARE values out of our 103 class MRE/ARE values we have for each model (obtained from cross validation). This step yields 1,000 samples of 103 observations, each randomly and slightly different from the original sample as some of the original observations will not have been sampled, whereas others will have been sampled several times. That form of sampling is sometimes called "resampling."
2. For each of the 1,000 samples generated through Step 1, we compute the system MRE/ARE value, which is computed using the definitions above.
3. The 1,000 system-level MRE/ARE values obtained from the previous step form an empirical distribution. Compute the 2.5 percent and 97.5 percent percentiles of this distribution which represent, based on bootstrapping theory, a good estimate of the 95 percent error interval for the system effort prediction MRE/ARE.

In short, through the resampling of existing observations, bootstrapping enables the estimation of typical sample statistics' distributions. The sample mean, median, and standard deviations are common examples of such statistics. The basic assertion behind bootstrapping is that the relative frequency distribution of a sample statistic (MRE/ARE in our case) calculated from the "resamples" is an

estimate of the sampling distribution of this statistic. Theoretical work and simulations have shown this is the case when the number of resamples is large (1,000 is a standard number). In other words, if the sample is a good approximation of the population under study, it will provide a good approximation of the sampling distribution of any selected sample statistic. However, bootstrapping may fail for statistics that depend on "narrow features of the sampling process," e.g., estimating the sampling maximum distribution based on the maximum observation of each sample. Besides this limitation and though computationally intensive, bootstrapping works well with small samples [27]—unfortunately a common feature in software engineering research.

4 Related Work

There exists a substantial body of work on project cost estimation based on various project characteristics (references are provided in [3]). However, studies such as the present one, which investigate cost drivers for individual modules or classes within a project, are rare. There may be several reasons for this:

- Such studies require effort be collected on a per-class-basis in a consistent and reliable manner. This is more difficult than accounting only for the total project cost.
- From a practical perspective, such a fine granularity may not be needed as the typical application of a cost model is to estimate, at an early stage, the cost and risk associated with entire projects.

However, as has been discussed in Section 2.2, such fine-grained analyses allow us to learn about product-related cost drivers at a faster pace, with fewer projects. We are aware of two such studies:

- Nesi and Querci [29] propose a set of complexity and size code measures (based on counts of methods, attributes, and LOC or cyclomatic complexity) and aim to build effort prediction models from these. They classify their measures into "a posteriori" measures that are available only after implementation and "predictive" measures that can be collected earlier. Using industry data, and OLS regression, they show that models explaining about 80 percent of the variation in class effort can be built from either a posteriori or predictive measures, i.e. the predictive measures can explain variation in effort as well as a posteriori measures. This result is consistent with the findings in this paper.
- Chidamber et al. [14] have investigated the six design measures of their measurement suite proposed in [13] (a subset of the measures used in our study). Their aim was not to build accurate prediction models, but rather to test the ability of the measures to identify high effort and low productivity classes. They find dichotomized versions of CBO and LCOM2 to be good indicators of such classes, with small loss in predictive power when the measures are applied during early phases.

Both studies used a different modeling technique (OLS), no measures of goodness of fit other than the R^2 were given,[6] and no cross-validation was attempted. It is therefore not possible to quantitatively compare the results with the current study.

5 Analysis Results

This section presents the results obtained by following the procedure described in Section 2. Additional details regarding the analysis procedure itself are provided.

5.1 Descriptive Statistics

Looking at the distribution statistics is important in order to interpret the subsequent results of the analysis. For example, it is important to know that some measures show little variation, meaning, for instance, that a design mechanism is seldom used. This might explain some missing trends in the data. In addition, frequency distributions can tell us whether the system we look at is somewhat representative and shows unusual patterns, e.g., no inheritance, extremely deep inheritance hierarchies. Again, this may be important to determine whether we can generalize our results or to explain any particular result.

Based on Table 1, we notice a number of interesting results. First, as opposed to what was observed in previous publications ([4], [8], [13], [14]), the use of inheritance mechanisms is here substantial. The average depth of classes in the inheritance hierarchy is two and only 11 out of 103 classes are not derived from another class. For most classes, the number of inherited methods and attributes surpasses the number of locally defined methods and attributes. Furthermore, the amount of inheritance-based coupling is about the same as noninheritance-based coupling, whereas it was only a fraction of it in previous studies. All this tells us that we are dealing here with a design that is really object-oriented in nature. Surprisingly, some classes show absolutely no coupling based on direct method invocation (minimum value of CBO = 0) and seem only to be invoked through polymorphic method invocations. This shows how important it is to take into account polymorphism when measuring coupling.

There are no friendship relationships in the LIOO system. Consequently, the measures of coupling between such classes (the F**EC and IF**IC measures of the suite by Briand et al. [7]) do not vary and are removed from further analysis. There is no aggregation coupling between classes related via inheritance (measured by ACAIC and OCAEC), these measures were also discarded. Because there is only aggregation between "other" classes, DAC and OCAIC yield identical values, and DAC was removed from further analysis to avoid redundancy.

6. We have not provided R^2 goodness-of-fit values here as the R^2 for Poisson regression does not have a simple, straightforward interpretation as the OLS R^2 and conclusions we would draw from such values would be similar to the ones we draw from MRE values.

TABLE 1
Descriptive Statistic for All Measures

Measure	Mean	StdDev	Max	P75	Med	P25	Min
CBO	3.883	8.127	72	3	2	1	0
CBO'	3.204	8.063	72	2	1	1	0
RFC_1	340.204	139.801	607	359	339	334	0
RFC_∞	450.223	176.127	783	511	415	401	0
MPC	12.214	25.881	189	10	4	3	0
PIM	51.592	174.731	1202	36	5	3	0
PIM_EC	51.592	47.161	311	56	48	27	0
ICP	73.592	221.020	1557	57	11	6	0
IHICP	15.379	53.949	392	7	0	0	0
NIHICP	58.214	180.780	1287	48	8	6	0
DAC	0.495	1.145	5	0	0	0	0
DAC'	0.408	0.964	5	0	0	0	0
OCAIC	0.495	1.145	5	0	0	0	0
OCAEC	0.495	1.037	8	1	0	0	0
ACMIC	0.320	0.782	4	0	0	0	0
OCMIC	1.981	4.824	31	1	0	0	0
DCMEC	0.320	2.766	28	0	0	0	0
OCMEC	1.981	8.882	83	0	0	0	0
AMMIC	6.078	19.734	165	5	0	0	0
OMMIC	6.136	10.908	67	4	3	1	0
DMMEC	6.078	31.670	250	0	0	0	0
OMMEC	6.136	31.056	311	4	0	0	0
LCOM1	80.320	239.885	1676	36	15	10	0
LCOM2	70.320	229.982	1641	36	15	10	0
LCOM3	6.369	6.155	40	7	5	4	0
LCOM4	4.650	3.044	22	6	5	2	0
LCOM5	0.974	0.323	1.25	1.2	1.125	.7888	0
COH	0.137	0.243	1	.21666	0	0	0
CO	-0.107	0.229	1	0	-.143	-.25	-.5
LCC	0.218	0.353	1	.34736	0	0	0
TCC	0.148	0.254	1	.30303	0	0	0
ICH	9.136	28.964	206	6	0	0	0
DIT	1.981	1.029	4	3	2	1	0
AID	1.947	1.034	4	3	2	1	0
CLD	0.379	0.806	4	0	0	0	0
NOC	0.922	2.295	16	0	0	0	0
NOP	0.922	0.362	2	1	1	1	0
NOD	2.029	9.129	89	0	0	0	0
NOA	2.029	1.089	5	3	2	1	0
NMO	4.447	2.950	18	6	5	1	0
NMA	5.184	9.767	51	5	1	0	0
SIX	0.298	0.277	.8333	.445	.207	.0909	-1
NA	8.194	7.606	30	13	4	3	1
NAIMP	1.728	3.335	16	2	0	0	0
NAINH	6.466	7.200	28	5	4	3	0
NM	38.699	31.070	158	40	25	24	0
NMIMP	9.631	10.263	59	9	6	5	0
NMINH	29.068	28.816	130	24	19	17	0
NUMPAR	4.990	9.439	42	4	1	0	0

5.2 Principal Component Analysis

5.2.1 PCA with Coupling Measures

The results of principal component analysis on the coupling measures are summarized in Table 2. In order to be better able to interpret the PCs, we consider the rotated components. This is a technique where the PCs are subjected to an orthogonal rotation. As a result, the rotated components show a clearer pattern of loadings, where the variables either have a very low or very 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. See [16] for more details on PCA and rotated components. The first three rows in Table 2

TABLE 2
Rotated Components for Coupling Measures

	PC1	PC2	PC3	PC4	PC5
EigenValue:	8.984	3.956	2.381	1.543	1.432
Percent:	42.782	18.837	11.340	7.347	6.819
CumPercent:	42.782	61.618	72.958	80.305	87.124
CBO	0.223	**-0.929**	0.002	0.186	0.147
CBO'	0.156	**-0.945**	0.045	0.175	0.158
RFC_1	0.412	0.212	**-0.713**	-0.219	0.109
RFC_∞	0.383	0.238	**-0.731**	0.062	0.018
MPC	**0.905**	-0.055	-0.109	0.371	-0.044
PIM	**0.930**	-0.117	-0.015	0.137	0.268
PIM_EC	0.302	-0.685	-0.198	0.122	0.458
ICP	**0.929**	-0.115	-0.028	0.185	0.260
IHICP	**0.939**	-0.110	-0.038	0.134	0.241
NIHICP	**0.923**	-0.115	-0.026	0.196	0.263
DAC'	0.247	-0.235	0.114	**0.813**	0.314
OCAIC	0.206	-0.206	0.131	**0.853**	0.245
OCAEC	0.077	0.100	**0.886**	0.089	0.002
ACMIC	0.616	-0.078	-0.102	0.436	0.191
OCMIC	0.070	-0.506	0.178	0.636	0.123
DCMEC	0.209	-0.124	0.093	0.205	**0.876**
OCMEC	0.112	0.082	**0.808**	-0.036	0.141
AMMIC	**0.973**	-0.082	-0.067	0.062	-0.027
OMMIC	0.387	0.019	-0.138	**0.770**	-0.056
DMMEC	0.313	-0.132	0.015	0.188	**0.889**
Ommec	-0.091	**-0.970**	0.089	0.075	-0.111

report the eigenvalue, percentage of variation explained for each PC, and the cumulative percentage of variation explained. The remaining rows then show the loading of each measure in each PC. Loadings with absolute values larger than 0.7 indicate a substantial correlation with the PC and are set in boldface.

Based on Table 2, we can see coupling measures capture roughly five dimensions, i.e., principal components. Based on the highest loadings within each principal component, they can be interpreted as follows:

- *PC1*. All the measures with high loadings are import coupling measures, where coupling based on method invocations and where inheritance-based coupling and/or polymorphic coupling are taken into account. These two kinds of coupling tend to occur together since, in C++, method invocations in one's own inheritance hierarchies are polymorphic by default.
- *PC2*. This dimension is driven by both general coupling measures (CBO, CBO') and export coupling through method invocations. For some reason, the number of coupled classes for a given class is mainly driven by the number of method invocations to that class.
- *PC3*. This PC contains measures OCAEC and OCMEC counting export coupling by unrelated classes (unrelated by inheritance). The RFC measures, measuring size (inherited and noninherited) and coupling by method invocations, are inversely correlated to export coupling. This indicates that there are some relatively small and self-contained classes (=> low RFC) which provide small, universal services that are being used a lot (=> high OCAEC, OCMEC).
- *PC4*. This is mainly driven by static method invocation and aggregation import coupling. So, no inheritance or polymorphism-based coupling is taken into account here and that is what probably brings these measures together. Again, this shows that inheritance brings orthogonal coupling dimensions that are important to measure.
- *PC5*. This clearly captures export coupling to descendent classes.

5.2.2 Principal Component Analysis for Cohesion Measures

As already noticed in previous studies [4], [8], there are two clearly interpretable cohesion dimensions based on Table 3:

- *PC1*. All normalized cohesion measures are in this dimension. These measures are independent of the size of the class, although their normalization procedures differ. Again these results show that most of the variations in the measures that have been proposed to date do not make a substantial difference.
- *PC2*. All nonnormalized measures are captured here. Many of these measures have shown to be related to the size of the class in past studies (see also Section 5.3). We have discussed in [6] whether these measures can be considered valid measures of cohesion.

TABLE 3
Rotated Components for Cohesion Measures

	PC1	PC2
EigenValue:	4.440	3.711
Percent:	44.398	37.108
CumPercent:	44.398	81.506
LCOM1	0.084	**0.980**
LCOM2	0.041	**0.983**
LCOM3	-0.218	**0.929**
LCOM4	-0.604	0.224
LCOM5	**-0.878**	0.057
COH	**0.872**	-0.113
CO	**0.820**	0.139
LCC	**0.869**	0.320
TCC	**0.945**	0.132
ICH	0.148	**0.927**

TABLE 4
Rotated Components for Inheritance Measures

	PC1	PC2	PC3
EigenValue:	4.995	1.912	1.075
Percent:	49.947	19.119	10.751
CumPercent:	49.947	69.066	79.817
DIT	**0.905**	0.155	-0.298
AID	**0.851**	0.145	-0.371
CLD	-0.240	**-0.853**	0.010
NOC	-0.129	**-0.927**	0.111
NOP	**0.776**	0.051	0.104
NOD	-0.099	**-0.884**	0.116
NOA	**0.925**	0.161	-0.173
NMO	**0.720**	0.253	0.512
NMA	-0.244	-0.221	**0.806**
SIX	**0.712**	0.220	-0.070

5.2.3 PCA for Inheritance Measures

Regarding Table 4, three clear dimensions, which already had been identified in previous studies [4], [8] can be identified among all inheritance measures:

- *PC1*. This dimension captures the depth of a class. The deeper the class (DIT, AID), the more ancestors/parents it has, the more likely method overriding (captured with measure NMO) is to occur.
- *PC2*. This captures the depth of inheritance below the class.
- *PC3*. Only the number of methods added seems to drive this dimension. It captures the size of the functionality increment brought by the class.

5.2.4 PCA for Size Measures

Regarding size, in Table 5, two main dimensions are clearly identified:

- *PC1*. This mainly captures the inherited size (i.e., attributes, classes) from ancestor classes.
- *PC2*. This dimension relates to locally defined attributes and methods.

5.2.5 Discussion

From a general perspective, many of the dimensions outlined by principal component analysis are consistent with previous studies [4], [8]. Variations across studies are mainly observed among the coupling dimensions. Based on our comparisons of the system analyzed, they often reflect variations in the use of inheritance mechanisms. In an environment where the use of inheritance follows a clear strategy, we would expect the dimensions of coupling to be more stable across systems. But, this still remains to be confirmed.

From a practical perspective, the results of PCA show a large amount of redundancy among the proposed measures. This confirms the results from previous studies and implies that design measurement could be limited to a much smaller number of measures than the one we used here.

5.3 Correlation to Size

An issue related to PCA is to look at the relationship between the design measures and size. This is important as such a relationship is a possible explanation why certain design measures are related to effort.

One way to investigate the relationship of design measures to size is to perform a principal component analysis with design and size measures simultaneously. However, given the large number of measures considered here (over 40 measures) and the relatively small number of observations (103), this approach seems questionable. Therefore, we look at the pairwise relationship between design measures and a representative size measure, NMIMP (the number of implemented methods in a class). The measure of correlation we use is Spearman Rho. Given the skewed distribution of measures, this measure is preferred over Pearson's r.

From Table 6, we observe that many measures are very strongly correlated to size, for instance, Spearman Rho for ICP, OCMIC, LCOM1, LCOM2, ICH, with NMIMP ranges between 0.7 and 0.9 (see Appendix B). Such strong correlations were rare in the previous systems we analyzed [4], [8]. Given the definition of the measures, a certain correlation to size is to be expected, but we have no explanation why it is particularly strong in this system.

TABLE 5
Rotated Components for Size Measures

	PC1	PC2
EigenValue:	4,004	2.524
Percent:	57.20	36.07
CumPercent:	57.20	93.27
NA	**0.927**	0.314
NAIMP	-0.031	**0.880**
NAINH	**0.993**	-0.076
NM	**0.970**	0.215
NMIMP	0.155	**0.948**
NMINH	**0.990**	-0.106
NUMPAR	0.079	**0.948**

TABLE 6
Correlation of Design Measures to Size

Measure	Spearman's Rho	p-value
CBO	0.8097	<0.0001
CBO'	0.6731	<0.0001
RFC_1	0.3706	0.0001
RFC_∞	0.4414	<0.0001
MPC	0.6616	<0.0001
PIM	0.7331	<0.0001
PIM_EC	0.5797	<0.0001
ICP	0.7789	<0.0001
IHICP	0.6033	<0.0001
NIHICP	0.7453	<0.0001
DAC'	0.5525	<0.0001
OCAIC	0.5577	<0.0001
OCAEC	0.2045	0.0383
ACMIC	0.5533	<0.0001
OCMIC	0.7650	<0.0001
DCMEC	0.3118	0.0013
OCMEC	0.1009	0.3106
AMMIC	0.5623	<0.0001
OMMIC	0.5671	<0.0001
DMMEC	0.3512	0.0003
OMMEC	0.4075	<0.0001
LCOM1	0.9540	<0.0001
LCOM2	0.7960	<0.0001
LCOM3	0.6004	<0.0001
LCOM4	0.2582	0.0084
LCOM5	-0.3463	0.0003
COH	0.4010	<0.0001
CO	0.5167	<0.0001
LCC	0.5455	<0.0001
TCC	0.5083	<0.0001
ICH	0.8221	<0.0001
DIT	-0.1903	0.0541
AID	-0.2391	0.0150
CLD	-0.0635	0.5242
NOC	-0.0814	0.4135
NOP	-0.0333	0.7386
NOD	-0.0718	0.4709
NOA	-0.1383	0.1636
NMO	0.4029	<0.0001
NMA	0.8190	<0.0001
SIX	-0.0797	0.4235
NA	0.3525	0.0003
NAIMP	0.5695	<0.0001
NAINH	-0.0097	0.9228
NM	0.5623	<0.0001
NMINH	-0.1103	0.2675
NUMPAR	0.8184	<0.0001

5.4 Univariate Analysis

This section looks at the individual relationships between design measures and the effort allocated to each individual class. As was discussed in Section 3.1, an important assumption of the Poisson regression model is the equidispersion, i.e., that the conditional variance Var(y | X) equals conditional mean E(y | X). In practice, we more commonly find that Var(y | X) > E(y | X), which is known as overdispersion. In the presence of overdispersion, the significance of covariates can be overestimated. In that case, a negative binomial regression model should be used, which is designed to take this overdispersion into account.

In the negative binomial model, we still have the regression equation $\hat{\mu} = e^{\beta_0+\beta_1 x_1+\cdots+\beta_n x_n}$, however, we assume a different probability distribution of y, given $x_1, \ldots, x_n$:

$$\Pr(y \mid x_1, \cdots, x_n) = \frac{\Gamma(y+\nu)}{y!\Gamma(\nu)} \left(\frac{\nu}{\nu+\hat{\mu}}\right)^{\hat{\mu}} \left(\frac{\hat{\mu}}{\nu+\hat{\mu}}\right)^{\nu},$$

with a parameter $\nu > 0$ that is estimated along with the regression coefficients $\beta_0, \ldots, \beta_n$ in a maximum likelihood estimation based on the above probability distribution. It can be shown that we still have $\hat{\mu} = E(y \mid X)$, but $Var(y \mid X) = \hat{\mu} + \alpha\hat{\mu}$, where $\alpha = \nu^{-1}$. α is called the dispersion parameter. For α -> 0, the negative binomial model converges toward the Poisson model. For more details on Poisson and negative binomial regression, see [23].

Table 7 shows the results from applying univariate negative binomial regression to our data set. Columns "Coeff," "StdErr," and "p(coef)" indicate the estimated regression coefficient, its standard error, and p-value (i.e., probability that the coefficient is different from zero by chance). The columns alpha and p(alpha) show the estimated dispersion parameter and its p-value.

As a first result, we see that the alphas for all measures are significantly different from zero, i.e., overdispersion is present here. Overdispersion is a not a property of the dependent variable, but a property of a particular combination of dependent/independent variables. In multivariate analysis, due to the higher number of independent variables, the data can be better fitted and overdispersion is less likely to be a problem.

Looking at the significance of coupling measures, we can see that most import coupling measures are significant and show coefficients in the expected direction. Inheritance-based coupling measures seem to be associated with lower coefficients than coupling measures involving classes in different inheritance hierarchies. One explanation to further investigate is that classes within hierarchies are often developed by the same programmer, whereas different hierarchies are more likely to be developed by different people. Also, export coupling measures show a much weaker impact than import coupling on effort. It is interesting to note that previous studies showed similar results when using the number of detected faults as a dependent variable [4], [8].

With respect to cohesion measures, we can see that all cohesion measures are significant except LCOM4 and Coh. Because they are not normalized, LCOM1 and LCOM2, like in previous studies [4], [8], show a strong (quadratic) relationship with the number of methods locally defined in classes, that is a size measure. This may explain their relationship with effort. More importantly, normalized cohesion measures have coefficients with a sign opposite to what was expected. Our interpretation is that many classes deep in the hierarchies redefine a few methods without defining new attributes. They access inherited attributes, which are not taken into account by existing cohesion measures. Therefore, they are small and relatively

TABLE 7
Univariate Analysis Results

Msr	Coef.	Std.Err.	p(coef)	Alpha	P(alpha)
CBO	0.1703	0.0345	<0.0001	1.2650	<0.0001
CBO'	0.1634	0.0366	<0.0001	1.3399	<0.0001
RFC_1	-0.0001	0.0008	0.9030	1.7699	<0.0001
RFC_∞	0.0013	0.0007	0.0640	1.7162	<0.0001
MPC	0.0373	0.0064	<0.0001	1.0272	<0.0001
PIM	0.0035	0.0014	0.0120	1.5025	<0.0001
PIM_EC	0.0148	0.0034	<0.0001	1.3891	<0.0001
ICP	0.0041	0.0014	0.0020	1.3975	<0.0001
IHICP	0.0100	0.0037	0.0070	1.5506	<0.0001
NIHICP	0.0052	0.0017	0.0020	1.3880	<0.0001
DAC'	0.8633	0.1119	<0.0001	0.8465	<0.0001
OCAIC	0.7020	0.0848	<0.0001	0.7632	<0.0001
OCAEC	0.4377	0.2218	0.0480	1.7047	<0.0001
ACMIC	0.8765	0.1370	<0.0001	0.9605	<0.0001
OCMIC	0.2039	0.0310	<0.0001	0.9586	<0.0001
DCMEC	0.0289	0.0596	0.6280	1.7654	<0.0001
OCMEC	0.0428	0.0352	0.2240	1.7406	<0.0001
AMMIC	0.0286	0.0117	0.0140	1.5292	<0.0001
OMMIC	0.0767	0.0123	<0.0001	0.9242	<0.0001
DMMEC	0.0072	0.0045	0.1110	1.7047	<0.0001
OMMEC	0.0404	0.0172	0.0190	1.6540	<0.0001
LCOM1	0.0048	0.0009	<0.0001	1.0675	<0.0001
LCOM2	0.0046	0.0010	<0.0001	1.2402	<0.0001
LCOM3	0.0771	0.0233	0.0010	1.5154	<0.0001
LCOM4	-0.0383	0.0327	0.2420	1.7478	<0.0001
LCOM5	-2.2934	0.7496	0.0020	1.6453	<0.0001
COH	1.4052	0.9739	0.1490	1.7394	<0.0001
CO	5.3406	0.7589	<0.0001	1.1643	<0.0001
LCC	2.8036	0.3105	<0.0001	0.8101	<0.0001
TCC	4.1435	0.6990	<0.0001	1.2612	<0.0001
ICH	0.0419	0.0079	<0.0001	1.0882	<0.0001
DIT	-0.5108	0.1325	<0.0001	1.5416	<0.0001
AID	-0.7283	0.1331	<0.0001	1.3520	<0.0001
CLD	0.0034	0.1635	0.9840	1.7702	<0.0001
NOC	0.0123	0.0564	0.8270	1.7693	<0.0001
NOP	0.1906	0.2618	0.4670	1.7608	<0.0001
NOD	0.0078	0.0163	0.6320	1.7657	<0.0001
NOA	-0.1470	0.0948	0.1210	1.7296	<0.0001
NMO	0.1111	0.0321	0.0010	1.5550	<0.0001
NMA	0.1144	0.0135	<0.0001	0.6990	<0.0001
SIX	-2.7423	0.6768	<0.0001	1.5627	<0.0001
NA	0.0935	0.0189	<0.0001	1.3553	<0.0001
NAIMP	0.2847	0.0343	<0.0001	0.7303	<0.0001
NAINH	0.0140	0.0176	0.4270	1.7590	<0.0001
NM	0.0185	0.0052	<0.0001	1.5185	<0.0001
NMIMP	0.1112	0.0115	<0.0001	0.5437	<0.0001
NMINH	0.0028	0.0041	0.4980	1.7620	<0.0001
NUMPAR	0.1089	0.0111	<0.0001	0.5166	<0.0001

easy to develop but show a cohesion of zero. Even more surprisingly, when modifying the measures to take inherited attributes into account, the results do not show substantial differences. Two factors can contribute to this phenomenon: First, the noninherited methods may actually make only limited use of inherited attributes. Second, by taking inherited methods into account, the denominator (counting a maximum possible number of connections) for some cohesion measures grows proportionally faster than the numerator (actual number of connections).

ICH shows a strong correlation with effort, but, as discussed in previous papers [4], [8], this measure is conceptually and statistically strongly related to the size of the class. To conclude, the relationships between cohesion measures and effort are due to other phenomena unrelated to the internal cohesion of classes. However, this

TABLE 8
Poisson Model, Size Measures Only

Measure	Coef.	Std.Err.	p
NA	0.048	0.007	0.000
NAIMP	0.106	0.013	0.000
NMIMP	-0.066	0.010	0.000
NUMPAR	0.115	0.010	0.000
Intercept	0.157	0.108	0.146

TABLE 9
Goodness of Fit: Distribution of Class Level MREs for All Models

	Size Only		Size & Coupling		All measures	
	Poisson	Hybrid	Poisson	Hybrid	Poisson	Hybrid
Mean	1.706	0.724	1.504	0.693	0.965	0.744
StdDev	3.155	1.405	2.627	1.959	1.571	1.582
P25	1.660	0.559	1.261	0.416	0.729	0.642
Median	0.702	0.194	0.586	0.225	0.447	0.317
P75	0.185	0.074	0.226	0.152	0.291	0.093

does not come as a full surprise as we did not expect cohesion to show a direct strong correlation with effort, but rather play the role of an adjustment factor. This will be investigated in the section on multivariate analysis.

DIT and AID indicate that deeper classes require less effort. Again, this may be explained by the presence of simpler classes deep in the hierarchies, with no locally defined attributes, that redefine existing methods.

Most size measures are significant. Measures counting the number of methods locally defined or their number of parameters show the strongest correlation with effort. On the other hand, size measures counting the amount of inheritance are not significant (because of the strong use of inheritance, NA and NM are mostly driven by inherited elements).

5.5 Multivariate Analysis

Multivariate Analysis involves looking at the combined impact of all design measures (more precisely, we selected the ones that showed a p-value below 0.25 in the univariate analysis). We have three main interrelated objectives in performing such an analysis:

- Assess what goodness of fit such effort models can achieve. This gives us an optimistic maximum bound of what can be achieved if such models were to be developed for real use.
- Assess the additional impact of cohesion and coupling as compared to size alone. Are they of any practical significance in terms of effort estimation?
- Assess the gain in estimation accuracy at different stages of design.

We will first build a model based on size measures only, e.g., number of attributes, methods, or parameters in the class interface. This information is available earlier in the design process than coupling, cohesion, or structural complexity information. Then, we move to build models based on both size and coupling measures. Last, we make use of all available design measures, also including cohesion and complexity measures requiring detailed knowledge about the internal structure of classes. The three categories of models we build correspond to successive stages of object-oriented design where 1) classes and their public interfaces are identified, 2) their dependencies and relationships are established, and 3) their internal structure (i.e., private elements and internal invocations) is determined.

Stemming from just one case study, the particular models built here do not purport to have any general validity outside the bounds of our case study environment. But, recall that our goal here is to assess the feasibility of building such models and their relative predictive power at different stages of the development life cycle. The focus of our multivariate analysis is therefore to obtain an initial assessment of the feasibility of building effort prediction models based on analysis and design information.

5.5.1 Poisson Model Based on Size Measures

Table 8 shows the results of applying stepwise Poisson regression on the subset of design size measures preselected based on univariate analysis. The columns show, from left to right, the design measures selected as significant covariates in the model (definitions of all measures are summarized in the appendix), the coefficients estimated through maximum likelihood estimation, the associated standard error, and the p-value telling us about the significance of these coefficients (i.e., their probability to be different from zero by chance). All the tables reporting regression results in the paper will follow the same structure.[7]

The negative coefficient of measure NMIMP is counter-intuitive, as we expect effort to increase with the number of implemented methods. This is due to a suppressor relationship with variable NUMPAR, the number of method parameters in the class. Such suppressor relationships between correlated variables are commonly observed in multivariate models [15] and not of concern as long as there is no strong multicollinearity among the variables. Multicollinearity [1] was determined to be negligible in this model (Conditional number = 2.63).[8]

We also need to look at the goodness of fit of the model. Although there are many ways to look at it, a simple and intuitive way is to consider the magnitude of relative error (MRE) based on the comparisons between the model's expected class effort values and the actual class effort values. Table 9 provides us with the distributions of MRE values in the data set, for this model, and, to ease comparisons, all subsequent models built in the following

7. All multivariate models presented in this paper were also fitted using negative-binomial regression. In no case did we observe an overdispersion parameter significantly different from zero (at p = 0.05), justifying the use of Poisson regression.

8. The conditional number is defined as $C = \sqrt{\lambda_{max}/\lambda_{min}}$, where λ_{max} and λ_{min} are the maximum and minimum eigenvalues of the correlation matrix of the measures in the model. Experiments showed that values of C above 30 indicate high presence of multicollinearity for which remedial action should be taken. The experiments and the theory underlying the conditional number are described in [1]. High multicollinearity does not affect the goodness of fit, but it is expected to impact the predictive accuracy of the model as it results in coefficients with higher standard errors.

TABLE 10
Goodness of Fit: Distribution of Class Level AREs for All Models

	Size Only		Size & Coupling		All measures	
	Poisson	Hybrid	Poisson	Hybrid	Poisson	Hybrid
Mean	1.892	1.386	1.517	1.151	1.223	1.098
StdDev	3.296	2.498	2.724	2.159	1.959	1.989
P25	1.856	1.573	1.069	1.307	0.900	1.319
Median	0.521	0.329	0.631	0.286	0.452	0.495
P75	0.383	0.073	0.429	0.076	0.325	0.117

subsections will follow the same format. We indicate the mean, standard deviation, 25th percentile (row P25), median, and 75th percentile (P75) for Poisson and hybrid models based on various subsets of the measures investigated here (indicated by the columns). Table 10 shows the same for the AREs.

The MRE of the *system* predicted effort (i.e., sum of predicted class efforts) is not an interesting piece of information as we are dealing with the goodness of fit of an unbiased model—therefore, it is certain to be exact. We will look at the system prediction accuracy when performing cross-validation in Section 5.6.

5.5.2 Hybrid Model Based on Size Measures

As a first intermediate step to build our hybrid model, we have to build a regression tree as described in 3.2, using size measures only. The resulting tree (Fig. 1) consists of seven terminal nodes (STN_i). From the top down, the data set is split in a stepwise manner so that absolute deviation (See Section 3.2) is minimized. Each edge in the tree represents one subset resulting from a split and provides information regarding the condition under which an observation belongs to that subset. Here, a condition is always a threshold on the value of a specific design measure. Readers are referred to [10] for further details. The tree in Fig. 1 shows, for each terminal node, its label, its number of observations, its average absolute deviation (AD), and its corresponding median effort. When following a path from the top of the tree to any terminal node, AD decreases at each level. Across terminal nodes, as visible in Fig. 1, AD also tends to increase from left to right. This is due to the fact that the mean effort increases from left to right, thus allowing for larger sample variations.

Let us now turn our attention to the interpretation of the regression tree shown in Fig. 1. One standard procedure in regression trees to estimate the "importance" of variables as predictors is provided in [10]. One problem this procedure addresses is the one of "surrogate measures." It is common in data sets such as the one considered here that several measures lead to an identical or similar split for any given node in the tree. Though some measures may not appear in the trees that we obtain, they may actually be good predictors of effort. It is unnecessary here to show surrogates for each split and clutter the tree diagram with more information, but it is important to understand that, following the procedures in [10], surrogate measures are accounted for to estimate the relative importance of design measures in Table 11. The general principle to compute variable importance is as follows: The improvement in absolute deviation in each tree node when using a variable is estimated for both the variables selected for the splits and its surrogates. These improvements are summed over each node and totaled and are scaled relative to the best performing variable.

In Table 11, the best performing variable is NMIMP and it is assigned, following the procedure above, an importance of 100 percent. We can see, for example, that though NUMPAR never appears in the tree, it is ranked second in importance. This is due to the fact that it commonly is a very good surrogate for NMIMP. An important result, which confirms the Poisson Regression results, is that local

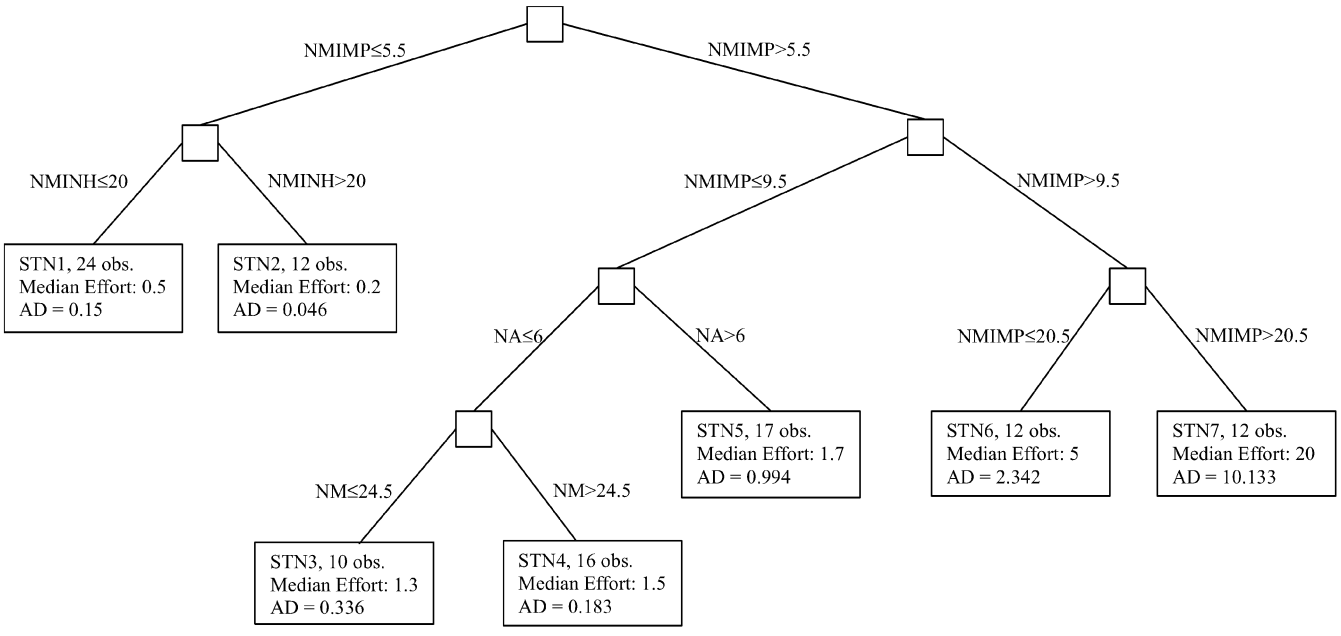

Fig. 1. Regression tree using size measures only.

TABLE 11
Variable Importance in Size Regression Tree

Design Size Measure	Relative Importance
NMIMP	100.000
NUMPAR	78.287
NAIMP	60.917
NMPUB	39.572
NM	39.572
NA	34.679
NAINH	5.810
NMINH	3.609
NMNPUB	1.743

(implemented) class features are more strongly related to effort than inherited features. That is, inherited features do not add significant effort overhead in our data set. If an opposite result were to be obtained in another context and data set, it would suggest that the use of inheritance is potentially a problem, leading to inefficiencies and it should be investigated.

To assess the overall goodness-of-fit of the regression tree in Fig. 1, we compute the mean AD over all the data set using resubstitution, as described in [10] and summarized in Section 3.2. The tree mean AD = 1.72, that is, roughly half the original data set mean AD of 3.356. In other words, the resubstitution relative error = 0.513. This only shows the improvement in goodness-of-fit due to using the regression tree and demonstrates its potential utility in explaining effort variation.

If we now focus on the use of regression trees to improve Poisson regression estimation accuracy, each terminal node is then transformed into additional binary variables in the data set. In this particular case, these new variables are also denoted $STN1, \ldots, STN7$ (after terminal nodes), indicating, for each observation, if it belongs to the respective terminal node or not. Again, a stepwise Poisson regression was run, this time allowing the size measures and the dummy variables to enter the model. From the results of Poisson regression in Table 12, we can see that three of the terminal node dummy variables got selected as significant covariates: STN1, STN2, and STN7. In addition, we still have the four size measures selected in the previous

TABLE 12
Hybrid Model with Size Measures

Measure	Coef.	Std.Err.	P>|z|
NA	0.030	0.007	0.000
NAIMP	0.115	0.013	0.000
NMIMP	-0.084	0.011	0.000
NUMPAR	0.089	0.011	0.000
STN1	-0.979	0.284	0.001
STN2	-2.173	0.624	0.000
STN7	1.480	0.177	0.000
Intercept	0.623	0.124	0.000

TABLE 13
Poisson Model with Size, Coupling

Measure	Coef.	Std.Err	p-value
OCMIC	0.069	0.008	0.000
MPC	0.013	0.004	0.001
ICP	-0.001	0.001	0.014
NAIMP	0.146	0.015	0.000
ACMIC	0.204	0.058	0.000
IHICP	0.005	0.001	0.001
DMMEC	-0.004	0.002	0.034
Intercept	0.092	0.094	0.331

model. Multicollinearity was tested and does not present a problem in this model (Conditional number: 8.5).

The goodness of fit of the model has improved significantly over the Poisson regression model (Table 9). The median and mean MRE went from 0.70 and 1.7 to 0.19 and 0.73, respectively. A paired t-test for comparing the mean MREs across the two models shows a significant difference (t = 3.64, p = 0.0004). The mean ARE went from 1.84 to 1.39 (t = 2.37, p = 0.02). These results suggest a major improvement and we have supporting evidence here that the way we combine Poisson regression and regression trees can help to improve model building in terms of goodness of fit. This will be confirmed in Section 5.6 when we investigate cross-validation results.

5.5.3 Poisson Model Based on Size and Coupling

In this model, we use both size and coupling measures to build a multivariate effort model. One important question is whether coupling information, which can be measured at a later stage of OO design, can help improve effort predictions. As a first step, we look here at the improvement in goodness of fit, as shown by MRE.

Except for NAIMP, no size measures have been selected in the model shown in Table 13. This reflects that some coupling measures have a strong correlation to size (see Section 5.3) and are included in the model partly for this reason. However, since coupling measures are selected (not the size measures), additional information in the coupling measures makes them better covariates. The conditional number is 10.2. The model's fit (see Table 9) is slightly better than the size only model: Median/ mean MRE decreased from 0.70/1.70 to 0.59/1.50. Though this difference is small, a t-test indicates it is significant: t = 2.379, p = 0.0192. For the mean ARE, we have a decrease from 1.84 to 1.51 (t = 2.06, p = 0.042). Therefore, this suggests that coupling information contributes—but to a lesser extent than size—to explaining class development effort.

5.5.4 Hybrid Model Based on Size and Coupling Measures

Like for the size only model, we then attempt to build a hybrid model (regression tree + Poisson regression) using both size and coupling measures. The rules of the terminal nodes (SCTNi) are as follows: The regression tree we obtain differs only slightly from the Size-only tree in Fig. 1, thus

TABLE 14
Variable Importance in Size + Coupling Regression Tree

Design Size Measure	Relative Importance
NMIMP	100.000
NUMPAR	76.025
PIM_EC	48.571
OCMIC	46.957
ICP	44.410
CBOD	40.745
OMMIC	12.919

confirming the limited impact of coupling on effort in addition to size. In particular, SCTN1, 2, 6, and 7 are identical. PIM_EC and IH-ICP are new to the tree, where IH_ICP shows three close surrogates: AMMIC, PIM, and ICP. These results show that export coupling and import coupling, when accounting for polymorphism and dynamic binding, are playing a role, albeit limited, in a subpart of the tree. This is confirmed by Table 14, where only the most important variables are listed. We can see other import coupling measure such as OCMIC and OMMIC can be important in terms of effort explanation. They do not show in the tree but are weak surrogates of NMIMP and PIM_EC. However, despite the results in the importance table, the mean absolute deviations in the new terminal nodes do not seem to improve overall over the size-only regression tree (0.996, 0.336, and 0.183 versus 0.817, 0.527, 0.183).

When looking at the overall goodness-of-fit of the regression tree in Fig. 2 (mean AD = 1.68, resubstitution relative error = 0.501), we realize there is only a small improvement over the size model. So, despite the fact that coupling measures are being selected and deemed important in the tree construction, they do not make any substantial difference in terms of the overall goodness-of-fit of the regression tree.

TABLE 15
Hybrid Model, Size, and Coupling

Measure	Coef.	Std.Err.	P>|z|
MPC	0.016	0.003	0.000
OCMIC	0.033	0.009	0.000
ICP	-0.001	0.000	0.000
NAIMP	0.082	0.016	0.000
SCTN2	-2.676	0.631	0.000
SCTN1	-1.727	0.299	0.000
SCTN5	-1.472	0.284	0.000
ACMIC	0.343	0.067	0.000
SCTN4	-0.791	0.280	0.005
NA	-0.031	0.011	0.006
Intercept	1.260	0.173	0.000

Like for the size only model, the combined use of regression trees and Poisson regression (Table 15) substantially improves the goodness of fit of the model (see Table 9 and Table 10, where the mean and median MREs go from 0.58/1.5 to 0.22/0.69, t-test: t = 3.250, p = 0.0016; the mean ARE goes from 1.51 to 1.15, which is also significant according to a t-test: t = 2.57, p = 0.01). However, the goodness of fit is not very different from the hybrid model with only size measures (the mean and median MREs were 0.72 and 0.19 for the Hybrid size model; t-test: t = 0.263, p = 0.7931; decrease of mean ARE is also not significant). Using coupling measures in addition to size does not help to increase the predictive power of the hybrid model. The conditional number for this model is 6.3.

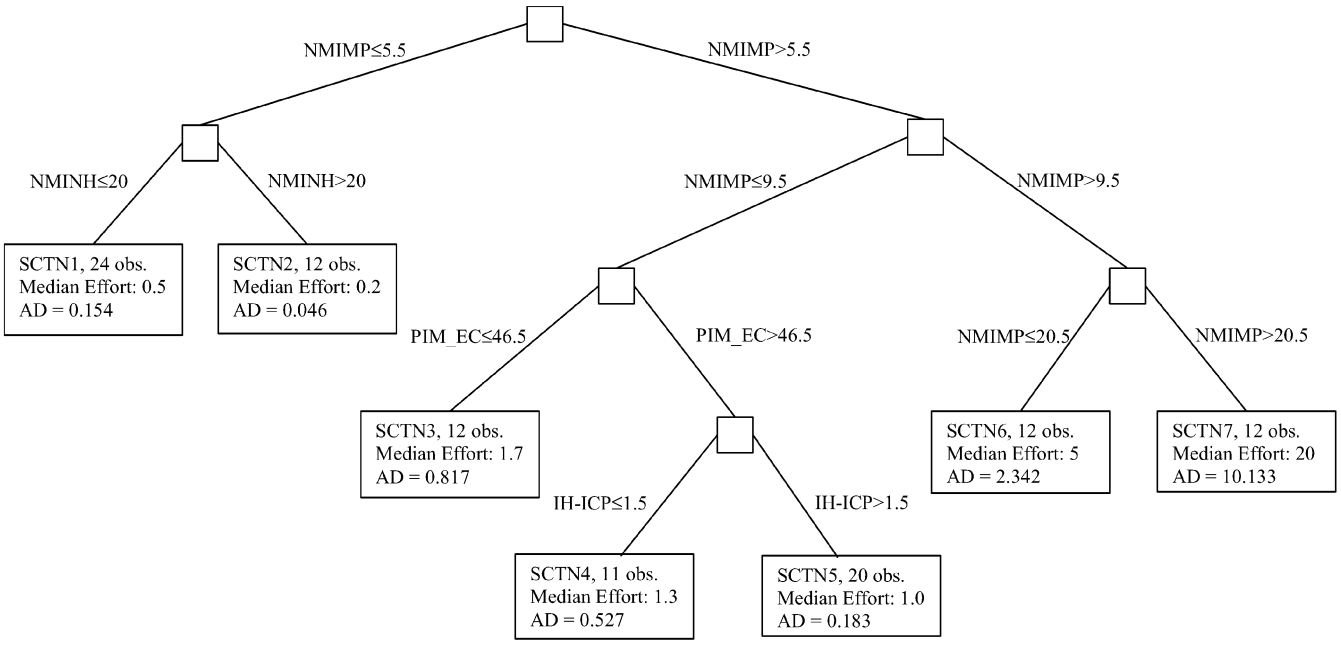

Fig. 2. Regression tree using size and coupling measures only.

TABLE 16
Poisson Model with All Measures

Measure	Coef.	Std.Err.	P>|z|
OCMIC	0.028	0.013	0.031
LCOM2	-0.002	0.001	0.000
ICH	-0.006	0.003	0.040
COH	-1.731	0.432	0.000
ACMIC	0.243	0.063	0.000
LCC	2.414	0.244	0.000
LCOM3	0.077	0.013	0.000
NUMPAR	0.058	0.010	0.000
Intercept	-0.505	0.160	0.002

TABLE 17
Variable Importance in All-Measures Regression Tree

Design Size Measure	Relative Importance
LCOM1	100.000
NMIMP	99.787
LCOM2	95.319
ICH	70.496
NAIMP	59.149
PIM_EC	53.688
OMMIC	16.099

5.5.5 Poisson Model Based on All Measures

We now use all design measures available to us, including cohesion and complexity measures, which are available during the late stages of the design. From Table 16, we can see that two coupling measures are selected and both normalized and nonnormalized cohesion measures [6]. A size measure is still in the model: NUMPAR. The conditional number is 10.6.

The goodness of fit (Table 9) of the model is clearly better than the Poisson model using size and coupling measures only: The mean/median MREs go from 1.50/0.48 to 0.96/0.44. A paired t-test testing the differences in mean MREs yields t = 3.998 and p < 0.0001. The decrease of the mean ARE from 1.52 to 1.22 is not significant when using the t test (t = 1.52 and p = 0.13). However, this is just the result of the distributions not being normal. A nonparametric Wilcoxon T test yields a p value equal to 0.0042 (z = 2.864). Therefore, although many of the cohesion measures show some correlation to size, they help significantly decrease both MRE and ARE. The latter does not improve as much as low effort observations, which are more sensitive to deviations from actual values in terms of MRE, tend to benefit further from using cohesion and complexity measures.

5.5.6 Hybrid Model Based on All Measures

Again, we built a regression tree from all measures, resulting in eight terminal nodes (denoted by $TN1, \ldots, TN8$ in Fig. 3). This time, the tree looks pretty different from the two previous ones. In order to simplify the interpretation and comparison of this tree, let us look first at the importance table in Table 17, where we list only the subset of the most important predictors. We see that NMIMP is still a very important variable. But, other cohesion measures (LCOM1, LCOM2) appear as well, along with previously mentioned coupling measures. As discussed earlier, these measures also have a strong quadratic relationship with NMIMP; therefore, they essentially capture class size and, thus, explain effort very well.

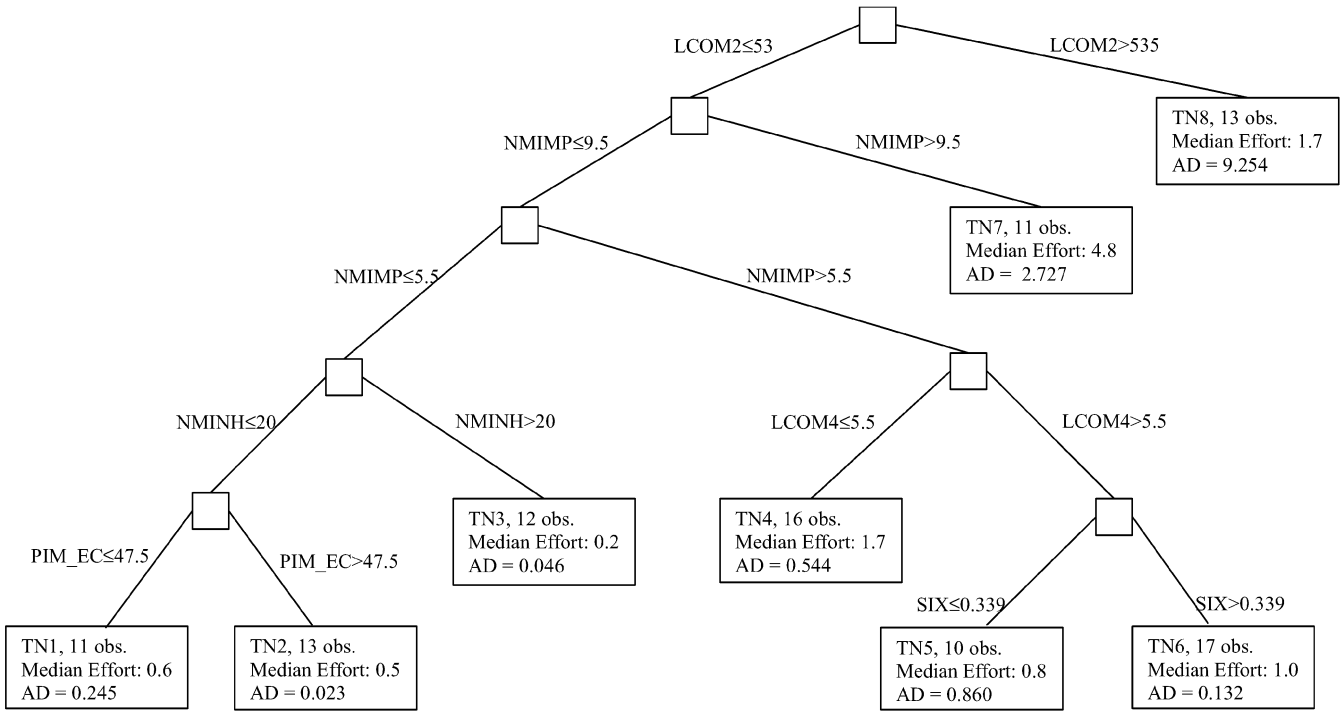

Fig. 3. Regression tree using all design measures.

TABLE 18
Hybrid Model with All Measures

Measure	Coef.	Std.	p
NUMPAR	0.038	0.012	0.001
NMIMP	0.050	0.021	0.017
ACMIC	0.206	0.048	0.000
LCC	0.763	0.173	0.000
TN8	0.774	0.159	0.000
TN3	-1.512	0.631	0.017
LCOM1	-0.002	0.000	0.000
TN2	-0.812	0.418	0.052
Intercept	-0.157	0.167	0.348

When looking at the overall goodness-of-fit of the regression tree in Fig. 3 (resubstitution relative error = 0.502), we realize there is no improvement over the size + coupling model. So, despite the fact that cohesion measures are being selected and deemed important in the tree construction, they do not make any substantial difference in terms of the overall goodness-of-fit of the regression tree.

When using all design measures with the hybrid model combining Poisson regression and regression trees (Table 18), we obtain only a slight improvement of the mean/median MRE over the Poisson model in Section 5.5.5 (for MRE, t = 1.892, p = 0.0613 not significant at the 5 percent level). This is even more apparent for the decrease in the mean ARE from 1.22 to 1.10, which is also not significant (t = 1.09, p = 0.29). In addition, there is no improvement over the hybrid model using size and coupling information (t-test for equal mean MREs t = -0.34, p = 0.7347; for the AREs, we have t = -0.54, p = 0.59). The conditional number of the model in Table 18 is 13.0.

5.5.7 Conclusions

From the goodness-of-fit results discussed above, we clearly see that class size seems to be the main driver in explaining effort. Although other attributes, such as coupling or cohesion, play a role, their impact in explaining effort variations is limited. If such results would be confirmed by further studies, that would mean that simple counts in class diagrams could lead to accurate effort estimates during analysis and high-level design, within certain limitations discussed in Section 5.7. A number of questions remain to be investigated. For example, although seven developers were involved in the system development under study, the variations due to human factors might be more important across systems, even within an application domain, thus leading to less accurate models.

Another important result is that the way we combine Poisson regression and regression trees seems to be very effective at improving the goodness-of-fit of some of our models. Now, in order to get a more realistic assessment of our Poisson models' predictive power, we will perform a cross-validation analysis.

5.6 10-Cross-Validation

Here, we perform a 10-cross-validation for each model. We randomly partition the data set into 10 subsets. For each of them, we refit each model on the remainder of the data set and assess the model by applying it to the held-out subset. Thus, we obtain new effort predictions for each observation in the data set, compute new MRE/ARE values for each model, and compare them using a two-tailed paired t-test.

TABLE 19
Distribution of Class Level MREs Using 10-Cross-Validation

	Size only		Size&Coupling		All measures	
	Pois.	Hyb.	Pois.	Hyb.	Pois,	Hyb.
Mean	1.712	0.779	1.545	0.770	1.013	0.851
P25	0.223	0.085	0.275	0.130	0.298	0.125
Med.	0.723	0.258	0.743	0.250	0.504	0.347
P75	1.295	0.571	1.286	0.464	0.759	0.708

The distribution of class MREs and AREs are summarized in Table 19 and Table 20, which are structured in the same way as Table 9.

The models we built in Section 5.5 seem to be stable as the MRE values did not increase significantly with the cross-validation procedure. When we perform t-tests to compare the class level MREs from model fit and cross-validation, no p-value is significant at $\alpha = 0.05$.

5.6.1 Impact of Design Measures

- For the three Poisson models, the "all measures" model has significantly lower MREs than the two other models. The difference between "size only" and "size & coupling" is not significant. In terms of AREs, there is no significant difference between any of the Poisson models.
- For the three hybrid models, there is no significant difference either in MRE or ARE between any of the hybrid models.
- Cross-validation therefore confirms that, when using a hybrid model that best captures the structure of the data, coupling measures do not improve MRE/ARE.

5.6.2 Use of Hybrid Models

The transition from Poisson regression to a hybrid model brings a significant improvement in MRE for the size-only and the size-and-coupling models, but not for the model using all measures. For AREs, only the improvement for the size-and-coupling models is statistically significant. From a general perspective, it seems that such hybrid models should be systematically tried out as they are easy to implement and can potentially bring significant improvements.

TABLE 20
Distribution of Class Level AREs Using 10-Cross-Validation

	Size only		Size&Coupling		All measures	
	Pois.	Hyb.	Pois.	Hyb.	Pois,	Hyb.
Mean	2.286	1.805	2.209	1.501	2.108	1.804
P25	0.368	0.059	0.483	0.087	0.328	0.122
Med	0.622	0.349	0.633	0.346	0.465	0.496
P75	1.697	1.679	1.093	1.374	1.306	1.450

TABLE 21
Distribution of System Level MREs Using Bootstrapping

	Size Only		Size & Coupling		All measures	
	Poisson	Hybrid	Poisson	Hybrid	Poisson	Hybrid
Mean	0.101	0.081	0.119	0.073	0.127	0.098
StdDev	0.082	0.061	0.092	0.056	0.090	0.079
P25	0.040	0.032	0.048	0.028	0.054	0.038
Median	0.083	0.069	0.103	0.060	0.112	0.083
P75	0.143	0.113	0.166	0.107	0.185	0.135
P2.5	0.004	0.003	0.005	0.002	0.005	0.004
P97.5	0.323	0.228	0.341	0.210	0.329	0.298

TABLE 22
Distribution of System Level AREs Using Bootstrapping

	Size Only		Size & Coupling		All measures	
	Poisson	Hybrid	Poisson	Hybrid	Poisson	Hybrid
Mean	38.594	32.104	46.194	29.016	50.432	37.870
StdDev	29.582	25.384	34.377	23.161	39.401	29.331
P25	15.187	12.332	18.468	10.681	20.057	15.091
Median	31.706	26.960	41.130	23.550	42.908	31.560
P75	54.655	44.820	64.184	41.398	70.629	53.594
P2.5	1.604	1.150	1.846	0.826	2.125	1.465
P97.5	107.345	95.848	135.495	84.724	142.270	106.167

5.6.3 Looking at the System Effort MRE/ARE Using Bootstrapping

Since we can, based on our predictions, compute only one system effort MRE/ARE value per model, we cannot perform a straight statistical comparison of the system effort prediction accuracy across models. However, this is one of the main goals of our study. In order to perform such statistical inferences, we are going to use bootstrapping, as specified in Section 3.4. One thousand resamples were generated from the original data set. For each resample, the system effort and corresponding MREs/AREs were calculated. The distribution of the system-level MREs/AREs thus obtained are given in Table 21 and Table 22 (structured in the same way as Table 9). We additionally indicate the 2.5th and 97.5th percentiles, which define the boundaries of the 95 percent confidence intervals.

The median MREs range between 5 and 11 percent and the 95 percent confidence intervals upper bounds are below 35 percent. This result shows that system effort predictions, in most cases, are expected to be relatively accurate by usual software engineering effort estimation standards. The system-level mean MREs from bootstrapping in Table 21 are one order of magnitude lower than the class-level mean MREs resulting from the 10-cross-validation in Table 19. This indicates that the over and underprediction of effort for individual classes cancel each other out when taking the sum of predicted effort over all classes, leading to more accurate estimates for the system effort. This is only possible because, as discussed above, our models are unbiased. From a practical standpoint, this means that our effort models are more suitable for the purpose of system effort prediction but would be more difficult to use, for example, to assign classes and effort to developers. Note that the absolute errors of course are much larger now as these numbers now pertain to sets of 100 classes.

An important application of such system effort Bootstrap distributions is risk analysis. In the area of cost prediction, we do need more than just point estimates of effort. We require predicted effort distributions to select budgets associated with acceptable levels of risks. For example, if one wishes (and is in a position) to select a budget that minimizes the risk of budget overrun, one can use the 95 percent upper MRE bound (e.g., 35 percent) and increase the predicted system effort using our models by adding a percentage corresponding to this MRE value (e.g., 35 percent overhead). Many such uses of the uncertainty associated with the system effort prediction can be devised.

As expected, the median and mean MREs and AREs are also significantly smaller for hybrid models according to unpaired t-tests comparing Bootstrap distributions. Another important result is that, in each case, transitions to hybrid models decrease the bounds of the confidence intervals, especially the upper bounds, where there is more room for improvement, thus reducing the risk associated with a prediction.

On the other hand, the use of coupling, cohesion, or complexity measures (late design measures) does not play a significant role in improving system effort predictions. The lack of effect at the system level is evident from the Bootstrapping results. Confidence intervals boundaries, means, and medians do not decrease. ARE and MRE values even seem to grow, though these effects are small.

5.7 Threats to Validity

We discuss, in turn, threats to the construct, internal, and external validity to this study. Our goal here is to help 1) the readers qualify the results that are presented in this paper and 2) future research by highlighting some of the issues associated with our study.

5.7.1 Construct Validity

Construct validity is the degree to which the independent and dependent variables accurately measure the concepts they purport to measure. The dependent variable class development effort is subject to measurement error as it is measured only at a certain granularity and developers working on several classes simultaneously can at most estimate how much of their total effort went into each class.

For an investigation as to which degree the coupling and cohesion measures used in this study measure the concepts they purport to measure, the reader is referred to [5], [6].

Another issue is that measurement of the independent variables was performed from source code, while, for the practical application of the models, this measurement must be taken from early designs (e.g., represented using the Unified Modeling Language). For the current study, no complete design documentation was available from which the measures could have been obtained. It therefore remains to be investigated how much the measures change when taken from design artifacts as compared to code and how much this affects accuracy of the prediction models. This will in large part depend on the development and design methodology that was followed.

5.7.2 Internal Validity

Internal validity is the degree to which conclusions can be drawn about the causal effect of the independent variables on the dependent variables. The analysis performed here is correlational in nature. We have demonstrated that several of the measures investigated have a statistically and practically significant relationship with class development effort. Such statistical relationships do not demonstrate, per se, a causal relationship. They only provide supporting 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 size, coupling, and cohesion in a system, while preserving its functionality, is difficult in practice. However, keeping these limitations in mind, it is difficult to imagine what could be confounding factors that are not measured in our study and that could explain the results we have obtained, other than the typical hypotheses relating size, coupling, and effort [6], [5].

5.7.3 External Validity

External validity is the degree to which the results of the research can be generalized to the population under study and other research settings.

The models we built are validated for the LIOO system and development environment only. We used bootstrapping to derive confidence intervals for MREs/AREs that we expect to achieve for such systems. However, there are other project and human factors, such as developer experience, application domain, development process, tools, languages used, time, and budget constraints. Variability introduced by such factors is not accounted for by our models nor the bootstrapping validation. In particular, LIOO was developed in an academic environment and is a relatively small system and, thus, is not fully representative of industrial OO development.

6 Conclusions

From the results presented in this study, we may conclude that there is a reasonable chance that useful effort estimation models could be built during the analysis and design of object-oriented systems. The best model shows system effort prediction MREs that lie, in 95 percent of the cases, within a 3 to 23 percent interval when using cross-validation and Bootstrapping to obtain realistic results. This result is obtained because, in part, we used a regression technique that yielded unbiased predictions: Poisson regression. OLS regression with transformed variables has been shown to be severely biased and yielded severely underestimated system effort predictions.

These results must be seen as a maximum bound rather than a realistic figure for cross project predictions since the data comes from one system. Although seven developers were involved, if models had been built using data from several systems, additional human and process factors would have likely introduced more variation in the trends we have observed here. But, this remains to be investigated.

Another important result is that simple size measures, which can be obtained from class diagrams or code, explain most of the effort variance. More sophisticated coupling measures do not bring substantial gains in terms of goodness of fit and effort estimation accuracy. The investigated cohesion measures do not help at all.

Last but not least, the combination of Poisson regression and regression trees has helped to significantly improve predictions, especially predictions based on size measures only that can be performed early on during the design stages. This can be explained by the fact that regression trees tend to capture complementary structures in the data since they help define new predictors capturing interactions between the original predictors. Though they will not systematically help, such hybrid models are worth trying.

An important problem remains to be addressed. Recall that our effort predictions do not include system and acceptance test effort, as this cannot be considered when building models at the class granularity level. Then, how to perform complete effort predictions? Building models at the system or subsystem level may, on the other hand, be an unrealistic objective as collecting the required number of project data points may take a prohibitive time. An alternative is for organizations to design common data repositories to speed up the data collection process [3]. Another possibility that should be investigated is to devise an overhead model that systematically adds some system and acceptance test effort overhead to the predicted effort of our models.

Appendix A

Comparison with Linear and Loglinear Models

In this appendix, we compare the performance of more traditional linear and loglinear models used in software engineering cost estimation to the performance of the Poisson models presented in the paper. This is important to ensure that our choice of more complex Poisson models—and the increased difficulty of building and applying such models—actually pays off and is not easily replaced by simpler models that work just as well.

We will perform this comparison for the size-only models, which are the simplest and should therefore show the results in the clearest way. Note that, at this point, the focus is on comparing modeling techniques, not on building accurate models based on different subsets of measures. We first present the linear and loglinear models and then compare their goodness-of-fit, class level errors in 10-cross-validation, and system level errors with bootstrapping to the Poisson size-only model in Section 5.5.1.

A.1 Linear Model

The linear model expresses effort as a direct linear combination of the dependent variables:

$$eff = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n.$$

Table 23 shows the linear model obtained by stepwise linear regression.

TABLE 23
Linear Model

Effort	Coef.	Std.Err.	P>|t|
NUMPAR	.5213521	.0833277	0.000
NAIMP	.6565265	.2358405	0.006
Intercept	.1307116	.6343919	0.837

TABLE 24
Loglinear Model

Measure	Coef.	Std.Err.	P>|t|
NUMPAR	.0494668	.0066604	0.000
NAIMP	.0901252	.0189597	0.000
NA	.0194166	.0061861	0.002
Intercept	.439571	.0650308	0.000

TABLE 25
Goodness of Fit: Distribution of Class Level AREs for Size-Only Models

	Poisson	Linear	Loglinear
Mean	1.892	2.446	2.359
StdDev	3.296	4.965	5.381
P25	1.856	2.471	1.180
Median	0.521	0.869	0.477
P75	0.383	0.369	0.184

The model contains two covariates, the number of method parameters NUMPAR, and the number of non-inherited class attributes, NAIMP. The adjusted R-square of this model is 0.5825, i.e., 58 percent of the variation in class level effort is explained by the model.

A.2 Loglinear Model

Log-linear regression models are frequently used in software engineering cost-estimation models [3]. The regression equation is $\ln(eff + 1) = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$. The logarithmic transformation decreases the weight of extreme observations with high values in the dependent variable.

Table 24 shows the loglinear model obtained through the stepwise regression heuristic. As the linear model, the loglinear model contains NUMPAR and NAIMP as covariates and NA contains the number of both inherited and noninherited class attributes. The adjusted R-squared is 0.7522, which would be an acceptable level of accuracy for such a model.

A.3 Goodness of Fit

Table 25 and Table 26 show the distribution of class level AREs and MREs for the above linear and loglinear model and the Poisson model from Section 5.5.1.

The mean absolute error of the Poisson model is lower than that of the linear and loglinear models. Statistically, however, these differences are not significant (paired t-test comparing Poisson and linear AREs: t = -1.28, p = 0.20; for Poisson and loglinear: t = -1.33, p = 0.19).

TABLE 26
Goodness of Fit: Distribution of Class Level MREs for All Models

	Poisson	Linear	Loglinear
Mean	1.706	1.375	1.011
StdDev	3.155	2.690	1.595
P25	1.660	0.869	1.230
Median	0.702	0.739	0.379
P75	0.185	0.346	0.254

TABLE 27
Distribution of Class Level AREs Using 10-Cross-Validation

	Poisson	Linear	Loglinear
Mean	2.286	2.786	2.744
P25	0.368	0.353	0.208
Med	0.622	0.881	0.528
P75	1.697	2.852	1.237

TABLE 28
Distribution of Class Level MREs Using 10-Cross-Validation

	Poisson	Linear	Loglinear
Mean	1.712	1.560	1.042
P25	0.223	0.441	0.283
Med.	0.723	0.804	0.387
P75	1.295	1.055	1.271

In terms of MRE, the Poisson model is worse than both the linear and loglinear model. Only the difference between loglinear and Poisson model is significant (t = 4.17, p < 0.01).

Being a biased model, the sum of predicted effort of the loglinear model over all classes is 361.6, hence, system level ARE = 393.3-361.6 = 31.7 and the system level MRE 31.7/393.3 = 0.08. System level ARE and MRE for the nonbiased Poisson and linear models are zero.

A.4 10-Cross-Validation

Subjecting the above models to the 10-cross-validation process yields the distributions for class level AREs and MREs illustrated in Tables 27 and 28.

The results reflect those from the goodness-of-fit analysis. The Poisson model has lower absolute errors than both the linear and loglinear model; both times the difference is not significant (paired t-test comparing Poisson and linear AREs: t = -1.10, p = 0.27; for loglinear: t = -1.24, p = 0.22). The Poisson MREs are worse than linear and loglinear MREs; for the loglinear model, this difference is even statistically significant.

A.5 Bootstrapping for System Level Errors

Now, we again apply the bootstrapping algorithm described in Section 3.4 to the predicted class level efforts obtained in the above 10-cross-validation, summing up the predicted class level efforts to obtain a system level error. Table 29 shows the distribution of system level AREs; Table 30 shows the same for MREs.

Not surprisingly, from what has gone before, the system level AREs for the Poisson model are clearly lower than those of the linear and loglinear model (unpaired t-tests statistically significant at p < 0.001).

TABLE 29
Distribution of System Level AREs Using Bootstrapping

	Poisson	Linear	Loglinear
Mean	38.594	54.224	63.871
StdDev	29.582	41.493	46.045
P25	15.187	22.265	28.613
Median	31.706	45.440	55.665
P75	54.655	76.942	89.643
P2.5	1.604	1.856	3.308
P97.5	107.345	149.902	168.916

TABLE 30
Distribution of System Level MREs Using Bootstrapping

	Poisson	Linear	Loglinear
Mean	0.101	0.137	0.160
StdDev	0.082	0.102	0.108
P25	0.040	0.056	0.075
Median	0.083	0.121	0.145
P75	0.143	0.193	0.230
P2.5	0.004	0.005	0.010
P97.5	0.323	0.378	0.400

Interestingly, the system level MRE of the Poisson model is now also lower than those of the linear and loglinear models. The loglinear model has the worst system level MRE of the models considered here, which is clearly a consequence of the biased nature of this model. For the linear model, this is a bit of a surprise and might be explained by the fact that the mean MRE in 10-cross-validation was not significantly better than that of the Poisson model and the median and 25th percentile were worse than those of the Poisson model.

A.6 Conclusions

We can summarize these results as follows: In our particular context of application, where we use the sums of predicted class level efforts to obtain an overall predicted system level effort, the use of Poisson models clearly shows an advantage over the traditional (log)linear models (in our case, ca. 40 percent lower absolute and relative errors are achieved). This justifies our choice of using Poisson models for our context of application.

In the traditional software engineering applications of cost estimation, we typically have models to predict the cost of entire projects, based on a number of system-wide project cost factors. In that context of application, there is no need to take the sum of predicted efforts over a number of observations, hence, biased models are not a problem. In that case, the use of loglinear models, which yield better MREs for individual observations, may be perfectly appropriate.

TABLE 31
Cohesion Measures

Name	Definition	Src.
LCOM1	Lack of cohesion in methods. The number of pairs of methods in the class using no attribute in common.	[12]
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.	[13]
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.	[19]
LCOM4	Like LCOM3, where graph G additionally has an edge between vertices representing methods *m* and *n*, if *m* invokes *n* or vice versa.	[19]
Co	Connectivity. Let V be the number of vertices of graph G from measure LCOM4, and E the number of its edges. Then $Co = 2(\|E\| - (\|V\| - 1))/((\|V\| - 1)(\|V\| - 2))$.	[19]
LCOM5	Consider a set of methods $\{M_i\}$ (i=1,...,m) accessing a set of attributes $\{A_j\}$ (j=1,...,a). Let $\mu(A_j)$ be the number of methods which reference attribute A_j. Then $LCOM5 = \frac{1}{a}((\sum_{j=1}^{a} \mu(A_j)) - m)/(1-m)$.	[18]
Coh	A variation on LCOM5: $Coh = (\sum_{j=1}^{a} \mu(A_j))/(m \cdot a)$	[6]
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.	[2]
LCC	Loose class cohesion. Same as TCC, except that this measure also considers pairs of *indirectly connected* methods. If there are methods m_1,..., m_n, such that m_i and m_{i+1} are connected for i=1,...,n-1, then m_1 and m_n are indirectly connected. Measure LCC is the percentage of pairs of public methods of the class which are directly or indirectly connected.	[2]
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.	[21]

TABLE 32
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).	[13]
CBO'	Same as CBO, except that inheritance-based coupling is not counted.	[12]
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.	[12]
RFC_1	Same as RFC_∞, except that methods indirectly invoked by methods in M are not included in the response set.	[13]
MPC	Message passing coupling. The number of method invocations in a class.	[20]
DAC	Data abstraction coupling. The number of attributes in a class that have another class as their type.	[20]
DAC'	The number of different classes that are used as types of attributes in a class.	[20]
ICP	Information-flow-based coupling. The number of method invocations in a class, weighted by the number of parameters of the invoked methods. Takes polymorphism and dynamic binding into account.	[21]
IH-ICP	As ICP, but counts invocations of methods of ancestors of classes (i.e., inheritance-based coupling) only.	[21]
NIH-ICP	As ICP, but counts invocations to classes not related through inheritance.	[21]
PIM	Polymorphically invoked methods. The number of invocations of methods of a class c by other classes (regardless of the relationship between classes). Also takes polymorphism and dynamic binding into account. Same as ICP, except that no weighting by the number of parameters is performed.	---
PIM_EC	Export coupling version of PIM. The number of invocations of methods of a class c by other classes (regardless of the relationship between classes). Also takes polymorphism and dynamic binding into account.	---
IFCAIC ACAIC OCAIC FCAEC DCAEC OCAEC IFCMIC ACMIC OCMIC FCMEC DCMEC OCMEC IFMMIC AMMIC 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 is using another class. • EC: Export coupling: count interactions where class d is the used class.	[7]

APPENDIX B
DEFINITION OF DESIGN MEASURES

The measures of coupling, cohesion, and inheritance identified in a literature survey on object-oriented design measures [5], [6], as well as two new variations of coupling measures, 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 effort data could realistically be collected.

The Tables 31, 32, 33, and 34 describe the 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 natural language definitions of the measures should give the reader 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 [5], [6].

In this paper, measures are classified as coupling, cohesion, or inheritance measures based on what their authors labeled them to be. There are a number of approaches to defining these attributes in an objective manner; a recent and practical proposal is [9], which defines, for each attribute, a set of mathematical properties that measures of the attribute should possess. As is shown in [5], [6], not all coupling and cohesion measures investigated here fulfill these properties. All size measures used here fulfill the size properties postulated in [9].

TABLE 33
Inheritance Measures

Name	Definition	Src
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.	[12]
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.	[18]
CLD	Class-to-leaf depth. CLD of a class is the maximum number of levels in the hierarchy that are below the class.	[32]
NOC	The number of children, parent, descendent, or ancestor classes of a class.	[12], [20], [23], [32]
NOP		
NOD		
NOA		
NMO	Number of methods overridden The number of methods in a class that override a method inherited from an ancestor class.	[23]
NMA	Number of methods added. The number of new methods in a class, not inherited, not overriding.	[23]
SIX	Specialization Index. SIX is defined as NMO * DIT / (NMO+NMA+NMINH)	[23]

TABLE 34
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
NM	The number of methods in a class (both inherited and non-inherited)
NAIMP	The number of attributes in a class (excluding inherited ones). Includes attributes of basic types such as strings, integers.
NA	The number of attributes in a class (both inherited and non-inherited)
NUMPAR	(Number of parameters)The sum of the number of parameters of the methods implemented in a class.

More importantly, from a practical point of view, the measures are roughly distinguished by the information required to compute them. Size measures rely on information available from the class interface only, coupling measures additionally require information about method calls and internal class attributes, and cohesion and complexity measures additionally require information about attribute usage and within-class method invocations. Hence, the progressive stages at which these measures become available are size and inheritance measures first, then coupling, then cohesion and complexity.

In order to make use of these measures at the design stage possible, 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 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. [7], and ICH.

ACKNOWLEDGMENTS

The authors wish to thank Paolo Nesi, Fabrizio Fioravanti, and their colleagues at the University of Florence for providing them with the LIOO system and effort data. Also, they thank Audris Mockus for his advice on statistical matters. The Concerto2/AUDIT tools and the FAST technology were develop by SEMA Group, France, and are now marketed by Verilog. We want to thank the SEMA group for providing us with this tool suite. Lionel Briand was in part supported by the US National Science Foundation under grant number 0082574 and by the Canadian Science and Engineering Research Council. All referenced ISERN reports are available at http://www.iese.fhg.de/ISERN.

REFERENCES

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

[2] J.M. Bieman and B.-K. Kang, "Cohesion and Reuse in an Object-Oriented System," *Proc. ACM Symp. Software Reusability (SSR '94),* pp. 259-262, 1995.

[3] L. Briand, K. El Emam, K. Maxwell, D. Surmann, and I. Wieczorek, "An Assessment and Comparison of Common Software Cost Estimation Models," *Proc. 21st Int'l Conf. Software Eng., (ICSE '99),* 1999.

[4] L. Briand, J. Daly, V. Porter, and J. Wüst, "Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems," *J. Systems and Software,* vol. 51, pp. 245-273, 2000.

[5] L. Briand, J. Daly, and J. Wüst, "A Unified Framework for Coupling Measurement in Object-Oriented Systems," *IEEE Trans. Software Eng.,* vol. 25, no. 1, pp. 91-122, Jan. 1999.

[6] L. Briand, J. Daly, and J. Wüst, "A Unified Framework for Cohesion Measurement in Object-Oriented Systems," *Empirical Software Eng. J.,* vol. 3, no. 1, pp. 65-117, 1998.

[7] L. Briand, P. Devanbu, and W. Melo, "An Investigation into Coupling Measures for C++," *Proc. 19th Int'l Conf. Software Eng., (ICSE 97),* 1997.

[8] L. Briand, J. Wüst, and H. Lounis, "Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs," *Empirical Software Eng.: An Int'l J.,* vol 6, no. 1, pp. 11-58, 2001.
[9] L. Briand, S. Morasca, and V. Basili, "Property-Based Software Engineering Measurement," *IEEE Trans. Software Eng.,* vol. 22, no. 1, pp. 68-86, Jan. 1996.
[10] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, *Classification and Regression Trees.* Wadsworth & Books/Cole Advanced Books & Software, 1984.
[11] LIOO Homepage (in Italian), http://aguirre.ing.unifi.it/~lioo, 1995.
[12] S.R. Chidamber and C.F. Kemerer, "Towards a Metrics Suite for Object Oriented Design," *Proc. Conf. Object-Oriented Programming: Systems, Languages and Applications (OOPSLA '91),* A. Paepcke, ed., pp. 197-211, Oct. 1991.
[13] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Software Eng.,* vol. 20, no. 6, pp. 476-493, June 1994.
[14] S. Chidamber, D. Darcy, and C. Kemerer, "Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis," *IEEE Trans. Software Eng.,* vol. 24, no. 8, pp. 629-639, Aug. 1998.
[15] R. Darlington, "Multiple Regression in Psychological Research and Practice," *Psychological Bulletin,* vol. 69, no. 3, pp. 161-182, 1968.
[16] G. Dunteman, *Principal Component Analysis.* SAGE Publications, 1989.
[17] W. Hayes, *Statistics,* fifth ed. Hartcourt Brace College Publishers, 1994.
[18] B. Henderson-Sellers, *Software Metrics.* Hemel Hempstaed, UK: Prentice Hall, 1996.
[19] M. Hitz and B. Montazeri, "Measuring Coupling and Cohesion in Object-Oriented Systems," *Proc. Int'l Symp. Applied Corporate Computing,* Oct. 1995.
[20] A. Lake and C. Cook, "Use of Factor Analysis to Develop OOP Software Complexity Metrics," *Proc. Sixth Ann. Oregon Workshop Software Metrics,* 1994.
[21] Y.-S. Lee, B.-S. Liang, S.-F. Wu, and F.-J. Wang, "Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow," *Proc. Int'l Conf. Software Quality,* 1995.
[22] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability," *J. Systems and Software,* vol. 23, no. 2 pp. 111-122, 1993.
[23] S. Long, "Regression Models for Categorical and Limited Dependent Variables," *Advanced Quantitative Techniques,* Sage Publications, 1997.
[24] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics.* Englewood Cliffs, N.J.: Prentice Hall, 1994.
[25] P. McCullagh and J. Nelder, *Generalized Linear Models.* London: Chapman & Hall, 1989.
[26] T. Graves and A. Mockus, "Identifying Productivity Drivers by Modeling Work Units Using Partial Data," Technical Report BL0113590-990513-09TM, Bell Laboratories, 2000.
[27] C. Mooney and R. Duval, "Bootstrapping. A Nonparametric Approach to Statistical Inference," *Quantitative Applications in the Social Sciences,* vol. 95, Sage Publications, 1993.
[28] P. Nesi, "Managing OO Projects Better," *IEEE Software,* pp. 50-60, July/Aug. 1998.
[29] P. Nesi and T. Querci, "Effort Estimation and Prediction of Object-Oriented Systems," *J. Systems and Software,* vol. 42, pp. 89-102, 1998.
[30] D. Steinberg and N. Cardell, "The Hybrid CART-Logit Model in Classification and Data Mining," *Salford Systems,* 1999, http://www.salford-systems.com.
[31] M. Stone, "Cross-Validatory Choice and Assessment of Statistical Predictions," *J. Royal Statistical Soc., Series B-36,* pp. 111-147, 1974.
[32] D.P. Tegarden, S.D. Sheetz, and D.E. Monarchi, "A Software Complexity Model of Object-Oriented Systems," *Decision Support Systems,* vol. 13, no. 3-4, pp. 241-262, 1995.

Lionel C. Briand received the PhD degree in computer science with honors from the University of Paris, XI, France. He is currently heading the Software Quality Engineering Laboratory in the Department of Systems and Computer Engineering, Carleton University, Canada. Before that, he was the Software Quality Engineering Department head at the Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany. He 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, College Park, 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 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 board 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, and technology evaluation. He is a member of the IEEE.

Jürgen Wüst received the Diplom-Informatiker MS degree in computer science 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 research activities and industrial activities include software measurement, software product evaluation, and object-oriented development techniques.