The Impact of Design Properties on Development Cost in Object-Oriented Systems

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

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

Technical Report ISERN-99-16

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, 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.

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%). Effort predictions at the system level are even more accurate as, using Boostrapping, the estimated 95% confidence interval for MREs is 3%-23%. 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.

Keywords

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.

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 cost in object-oriented (OO) systems. Does size, based on design information, show a strong relationship to cost? What additional impact do properties show such as complexity, cohesion, or coupling? Is their impact of any practical significance? Second, we would like to get an idea of how accurate can cost predictions 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 is here to determine whether this is likely to be feasible at all. We will base our results on a system developed in C++ (described in [20]) for which cost 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.

The paper starts by a description of the research method we use here, including also the data set from which the results are drawn. Then related work is briefly presented and compared to the objectives of our paper. Section 5 then 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 cost requires a careful analysis of real-project data. No theory regarding the many design factors that can potentially impact cost 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 regarding how design properties impact development effort, we cannot a priori make any assumption regarding 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 it, 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 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.

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 [27]. In this study, we have analyzed the version resulting from the first iteration. Seven developers were involved in creation of this first iteration. This version of the system is composed of 103 classes, totaling 23KLOC. Reused libraries (XVGAlib for the GUI) and automatically generated code (yacc/lex) that are not within the scope of this study and are not included in these figures.

Effort data was collected through questionnaires distributed to the seven developers on the project that worked in the first iteration. Throughout development, developers continuously kept a log of their activities to track the development effort per class. The total effort recorded was 393.3 person-hours. Design properties were collected through a set of design measures including most of the design measures proposed to date in the literature (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.

As stated above, the design information we use here to quantify design properties can potentially be extracted from UML diagrams. But, for the sake of convenience, we used the source code as an analyzer was readily available and since, like for any post-mortem 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 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 cost estimation accuracy. This is, however, a necessary first step.

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) [15] to determine the dimensions captured by our design measures. It is common in software engineering, like in other fields, to have many collinearities 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 cost. This is a first step to identify potential cost predictors to be used in the next step and to identify what types of measure are significantly related to cost. 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 non-normal 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 over-influential 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 cost, but considers design measures in combinations, as covariates in a multivariate model predicting cost. We will also use Poisson regression here combined with a forward stepwise variable selection procedure[1] [16]. In addition, we use here a method to improve the models' predictive accuracy by combining Poisson regression and regression trees (an adaptation of [29]). The goal of multivariate analysis is to determine what levels of predictive accuracy can be achieved in terms of cost estimation, during subsequent stages of design. In addition, it tells us about what kind of design measurement seems to play a more predominant, practically significant role for cost prediction. All results will be checked for the presence of over-influential observations.

5 Apply cross validation [30], 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 dataset into V parts, each part being used to assess a model built on the remainder of the dataset. In our study, we will divide our 103 observations dataset 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 to build, for example, a 95% 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% and 90% percentiles are 0.2 and 12.7, respectively), beyond what a normal distribution could 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 $\ln y = b_0 + b_1 x_1 + \cdots + b_n x_n$. Thus the logarithmic transformation can address the distribution problem mentioned above, and also decreases the weigh 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 *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 non-normal 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 3 minutes (0.05 person hours)). In addition, large effort values are rare and, as mentioned above, the effort distribution show 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 [25]. 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 [24] 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

[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.

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

[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.

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 [22].

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

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

The parameter μ is both the expected value and variance of y: μ=E(y)=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 X=$x_1,\ldots,x_n$, given by the regression equation

$$\hat{m} = e^{b_0 + b_1 x_1 + \cdots + b_n x_n} .$$

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

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

For a given data set, the regression coefficients $b_0,\ldots,b_n$ can be estimated through maximization of a likelihood function based on the above probability distributions.

3.2 Regression Trees

Regression tree analysis [10] is a data mining technique that does not require any functional form assumption, as opposed to regression analysis. 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 deviance [10]). The tree is also built so that a minimal number of observations are present in the terminal nodes (our study: 10). Each path of the tree represents a 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 non-linear 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. We will therefore attempt to combine Poisson regression and regression trees as specified below.

3.3 Combining Regression Trees and Poisson Regression Analysis

Adapting some of the recommendations in [29], 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 they are not nearly as effective.

[4] Poisson regression assumes: E(y | x_1, …, x_n)=Var(y | x_1, …, x_n). This assumption was tested on our dataset 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 datasets, the reader is invited to use a generalization of Poisson regression referred to as negative binomial regression [22].

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, to a lesser extent, 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,…,103, let eff_i be its actual effort, and $\hat{eff}_i$ the predicted effort. Then,

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

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 non-parametric (Wilcoxon T test) tests can be used [16]. 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:

$$ARE = \left| \sum_i (eff_i - \hat{eff})_i \right| \text{ and } MRE = ARE / \sum_i eff_i . \quad (1)$$

The expectation is that, for unbiased models, over-and under-prediction 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% 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 [26], a nonparametric, simulation-based technique to draw statistical inferences from (small) samples. In our case, we build the 95% error interval of the MRE/ARE for system effort prediction, following the bootstrapping procedure below:

1. Repeat 1000 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 steps yields 1000 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 "re-sampling".
2. For each of the 1000 samples generated through Step 1, we compute the system MRE/ARE value, which is computed using equation (1).
3. The 1000 system-level MRE/ARE values obtained from the previous step form an empirical distribution. Compute the 2.5% and 97.5% percentiles of this distribution, which represent, based on bootstrapping theory, a good estimate of the 95% error interval for the system effort prediction MRE/ARE.

In short, through the re-sampling of existing observations, bootstrapping enables the estimation of any sample statistic distribution, e.g., mean, median, standard deviations are common examples. Although computationally intensive, it has been shown bootstrapping works well with small samples [26]. 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 (1000 is a standard number).

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:

[5] We prefer here 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.

- Nesi and Querci [28] 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% 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 about as well as a-posteriori measures. This result is consistent with the findings in this paper.
- Chidamber, Darcy, and Kemerer [14] have investigated the six of the design measures 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, again 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 can 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 2 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 non-inheritance 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.

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_8	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

[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

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

Table 1: Descriptive statistic for all measures

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.

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 better be 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 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 [15] for more details on PCA and rotated components. The first three rows in Table 2 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 0.8 indicate a substantial correlation with the PC, and are set in boldface.

Based on Table 2, we can see coupling measures capture roughly 5 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 non-inherited) and coupling by method invocations, are inversely correlated to export coupling. This indicates that there 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 bring these measures together. Again, this show that inheritance bring orthogonal coupling dimensions that are important to measure.

PC5 This clearly captures export coupling to descendent classes.

	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_8	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

Table 2: Rotated Components for coupling measures

5.2.2 Principal Component Analysis for size 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 non-normalized 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.

	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 3: Rotated components for cohesion measures

5.2.3 PCA for Inheritance measures

Regarding Table 4, three clear dimensions, which already had been identified in previous studies [REF], 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 of drive this dimension. It captures the size of the functionality increment brought by the class.

	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

Table 4: Rotated components for inheritance measures

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.

	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 5: Rotated components for size measures

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 a 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. We therefore 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, e.g. Pearson's r.

Measure	Spearman's Rho	p-value
CBO	0.8097	<0.0001
CBO'	0.6731	<0.0001
RFC_1	0.3706	0.0001
RFC_8	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

Table 6: Correlation of design measures to size

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). Such strong correlations were rare in the previous systems we analyzed [3][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.

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 $\hat{m} = e^{b_0+b_1x_1+\cdots+b_nx_n}$, however, we assume a different probability distribution of y, given $x_1,\ldots,x_n$:

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

with a parameter $v>0$ that is estimated along with the regression coefficients $b_0,\ldots,b_n$ in a maximum likelihood estimation based on the above probability distribution. It can be shown that we still have $\hat{m} = E(y \mid X)$, but $Var(y \mid X) = \hat{m} + a\hat{m}$, where $a = n^{-1}$. α is called the dispersion parameter. For α–>0, the negative binomial model converges towards the Poisson model. For more details on Poisson and negative binomial regression, see [22].

Table 7 shows the results from applying univariate negative binomial regression to our model. 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 α and its p-value.

As a first result, we see 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 difference 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. They are therefore small and relatively easy to develop but show a cohesion of 0. Even more surprising, 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 non-inherited 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 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).

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_8	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

Table 7: Univariate Analysis Results

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 is the goodness of fit that such cost 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 cost 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 pre-selected 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 0 by chance). All the tables reporting regression results in the paper will follow the same structure[7].

Multicollinearity [1] was determined to be negligible in this model (Conditional number = 2.63)[8].

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 8: Poisson Model, size measures only

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 dataset, for this

[7] All multivariate models presented in this paper where 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=$\sqrt{l_{max} / l_{min}}$, where l_{max} and l_{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.

model, and, to ease comparisons, all subsequent models built in the following 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). In order to be complete, Table 10 shows the same for the AREs. This is not discussed in the remainder of the paper as the conclusions would be similar to the ones drawn based on MREs.

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

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	1.892	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

Table 10: Goodness of fit: Distribution of class level AREs for all models

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 – it is therefore 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. This tree consists of seven terminal nodes. Each derived rule characterizes a terminal node (STN_i) of the tree and is associated with a sample from which the median and mean can be estimated. Each observation in the dataset belongs to exactly one terminal node. We provide rules below (alternatively a tree could have been displayed) and their corresponding median effort value.

STN1: NMIMP=5.5 AND NMINH=20 => Median Effort = 0.5

STN2: NMIMP=5.5 AND NMINH>20 => Median Effort = 0.2

STN3: NMIMP>5.5 AND NMIMP=9.5 AND NA=6 AND NM=24.5 => Median Effort = 1.3

STN4: NMIMP>5.5 AND NMIMP=9.5 AND NA=6 AND NM>24.5 => Median Effort = 1.0

STN5: NMIMP>5.5 AND NMIMP=9.5 AND NA>6.000000 => Median Effort = 1.7

STN6: NMIMP>9.5 AND NMIMP=20.5 => Median Effort = 5

STN7: NMIMP>20.5 => Median Effort = 20

Each terminal node was then transformed into additional binary variables in the dataset. In this particular case, they are also denoted *STN1*, …, *STN7* (after size terminal node), indicating for each observation if it belongs to the respective terminal node, or not.

Again, as 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 11, we can see that three of the terminal node dummy variables got selected as significant covariates: STN1, STN2, STN7 (whose respective rules are given above). In addition, we still have the four size measures selected in the previous model. Multicollinearity was tested and does not present a problem in this model (Conditional number: 8.5).

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 11: Hybrid Model with size measures

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). This is a major improvement and we have here supporting evidence that the way we combine Poisson regression and regression trees can help to improve model building from a predictive point of view. This will be confirmed in Section 5.6, when we will 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 cost 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 12. 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, and 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/MRE increased 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. This therefore suggests that coupling information contributes–but not substantially more than size–to explaining class development effort.

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

Table 12: Poisson model with size, coupling

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:

SCTN1: NMIMP=5.5 AND NMINH=20 => Median Effort = 0.5

SCTN2: NMIMP=5.5 AND NMINH>20 => Median Effort = 0.2

SCTN3: NMIMP>5.5 AND NMIMP=9.5 AND PIM_EC=46.5 => Median Effort = 1.7

SCTN4: NMIMP>5.5 AND NMIMP=9.5 AND PIM_EC>46.5 AND IH-ICP=1.5 => Median Effort = 1.3

SCTN5: NMIMP>5.5 AND NMIMP=9.5 AND PIM_EC>46.5 AND IH-ICP>1.5 => Median Effort = 1.0

SCTN6: NMIMP>9.5 AND NMIMP=20.5 => Median Effort = 5

SCTN7: NMIMP>20.5 => Median Effort = 20

The regression tree we obtain differs only slightly from the Size-only tree, another indicator of the limited impact of coupling on effort in addition to size. In particular, SCTN1,2,6 and 7 use the same rules.

Like for the size only model, the combined use of regression trees and Poisson regression substantially improves the goodness of fit of the model (see Table 9, the mean and median MREs go from 0.58/1.5 to 0.22/0.69 ; t-test: t=3.250, p=0.0016). 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). 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.

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

Table 13: Hybrid model, size and coupling

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 14 we can see that two coupling measures are selected and both normalized and non-normalized cohesion measures [6]. A size measure is still in the model: NUMPAR. The conditional number is 10.6.

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 14: Poisson model with all measures

The goodness of fit (Table 9) of the model is clearly better than the Poisson regression 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. Although many of these measures show some correlation to size, they bring more information that helps explain additional class effort variation.

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, …, TN8). The rules for the TNs are:

TN1: LCOM2=53 AND NMIMP=5.5 AND NMINH=20 AND PIM_EC=47.5 ⇒ Median effort = 0.6

TN2: LCOM2=53 AND NMIMP=5.5 AND NMINH=20 AND PIM_EC>47.5 ⇒ Median effort = 0.5

TN3: LCOM2=53 AND NMIMP=5.5 AND NMINH>20 ⇒ Median effort = 0.2

TN4: LCOM2=53 AND NMIMP=9.5 AND NMIMP>5.5 AND LCOM4=5.5 ⇒ Median effort = 1.7

TN5: LCOM2=53 AND NMIMP=9.5 AND NMIMP>5.5 AND LCOM4>5.5 AND SIX=0.339 ⇒ Median effort = 0.8

TN6: LCOM2=53 AND NMIMP=9.5 AND NMIMP>5.5 AND LCOM4>5.5 AND SIX>0.339 ⇒ Median effort = 1.0

TN7: LCOM2=53 AND NMIMP>9.5 ⇒ Median effort = 4.8

TN8: LCOM2>53 ⇒ Median effort = 14.6

When using all design measures, the hybrid use of Poisson regression and regression trees only slightly improve the mean/median MRE (for MRE, t=1.892, p=0.0613 not significant at the 5% level). In addition, there is no improvement over the hybrid model using size and coupling information (t=-0.34, p=0.7347). The conditional number is 13.0.

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

Table 15: Hybrid model with all measures

From the goodness of fit results discussed above, we clearly see that class size seems to be the main driver in explaining cost. Although other attributes such as coupling or cohesion play a role, their impact on cost is limited. If such results would be confirmed by further studies, that would mean that simple counts in class diagrams could lead to accurate cost estimates during analysis and high-level design. Of course, 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 models' accuracy we will perform their cross validation.

5.6 10-Cross validation

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

The distribution of class MREs and AREs are summarized in Table 16 and Table 17, which are structured in the same way as Table 9.

	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

Table 16: Distribution of class level MREs 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 17: Distribution of class level AREs using 10-cross validation

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 α=0.05.

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.
- For the three hybrid models, there is no significant difference in MRE between any of the models.
- Cross-validation therefore confirms that, when using a hybrid model that captures best the structure of the data, coupling measures do not improve MRE.

Use of Hybrid Models

The transition from Poisson regression to a hybrid model brings a significant improvement for the size-only, and the size-and-coupling models, but not for the model using all measures. 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.

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.

1000 resamples were generated from the original dataset. 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 18 and Table 19 (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% confidence intervals.

	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 18: Distribution of system level MREs 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

Table 19: Distribution of system level AREs using Bootstrapping

The median MREs range between 5% and 11%, and the 95% confidence intervals upper bounds are below 35%. This result shows that system effort predictions, in most cases, are expected to be relatively accurate by usual software engineering cost estimation standards. The system-level mean MREs from bootstrapping in Table 18 are one order of magnitude lower than the class-level mean MREs resulting from the 10-cross validation in Table 16. 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.

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% upper MRE bound (e.g., 35%) and increase the predicted system effort using our models by adding a percentage corresponding to this MRE value (e.g., 35% overhead). Many such uses of the uncertainty associated with the system effort prediction can be devised.

As expected, the median and mean MREs 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. MRE values even seem to grow, but these effects are very small and not significant.

6 CONCLUSIONS

From the results presented in this study, we may conclude that there is a reasonable chance that useful cost 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% of the cases, within a 3%-23% 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 with transformed variables has shown to be severely biased and yielded severely underestimated son regression. OLS with transformed variables has 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 introduce 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, explain most of the effort variance. More sophisticated coupling measures do not add substantial gains in terms of goodness of fit and therefore in terms of cost estimation accuracy. The investigated cohesion and complexity measures do not help at all.

Last but not least, the combination of Poisson regression and regression trees has helped to improve significantly predictions, especially early predictions based on size measures only. This can be explained by the fact that they tend to capture complementary structures in the data. Although different attempts have been made in the past to combine trees and regression analysis [3], they were not successful for reasons beyond the scope of this paper.

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.

ACKNOWLEDGEMENTS

We thank Paolo Nesi and his colleagues at the University of Florence for providing us with the LIOO system and effort data, and 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 SEMA group for providing us with this tool suite.

REFERENCES

All referenced ISERN reports are available at http://www.iese.fhg.de/ISERN.

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

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

[3] L. Briand, K. El Emam, K. Maxwell, D. Surmann, I. Wieczorek, "An Assessment and Comparison of Common Software Cost Estimation Models". *In: Proceedings of the 21st International Conference on Software Engineering, ICSE 99*, (Los Angeles, USA 1999) 313-322.

[4] L. Briand, J. Daly, V. Porter, J. Wüst, "A Comprehensive Empirical Validation of Product Measures for Object-Oriented Systems", Journal of Systems and Software, to appear. Technical Report ISERN-98-07, 1998.

[5] L. Briand, J. Daly, J. Wüst, "A Unified Framework for Coupling Measurement in Object-Oriented Systems", IEEE Transactions on Software Engineering 25 (1), 91-122, 1999.

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

[7] L. Briand, P. Devanbu, W. Melo, "An Investigation into Coupling Measures for C++", Proceedings of ICSE '97, Boston, USA, 1997.

[8] L. Briand, J. Wüst, H. Lounis, S. Ikonomovski, "Investigating Quality Factors in Object-Orienbted Designs: an Indusrial Case Study", *Proceedings of the 21st International Conference on Software Engineering, ICSE 99*, (Los Angeles, USA 1999) 345-354.

[9] L. Briand, S. Morasca, V. Basili, "Property-Based Software Engineering Measurement", IEEE Transactions of Software Engineering, 22 (1), 68-86, 1996.

[10] Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J. Classification and Regression Trees. *Wadsworth & Books/Cole Advanced Books & Software* (1984).

[11] LIOO Homepage (in Italian): http://aguirre.ing.unifi.it/~lioo

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

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

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

[15] G. Dunteman, "Principal Component Analysis", SAGE Publications, 1989.

[16] Hayes W. Statistics. *Fifth Edition, Hartcourt Brace College Publishers* (1994).

[17] B. Henderson-Sellers, “Software Metrics”, Prentice Hall, Hemel Hempstaed, U.K., 1996.
[18] M. Hitz, B. Montazeri, “Measuring Coupling and Cohesion in Object-Oriented Systems”, in Proc. Int. Symposium on Applied Corporate Computing, Monterrey, Mexico, October 1995.
[19] A. Lake, C. Cook, “Use of factor analysis to develop OOP software complexity metrics”, Proc. 6th Annual Oregon Workshop on Software Metrics, Silver Falls, Oregon, 1994.
[20] Y.-S. Lee, B.-S. Liang, S.-F. Wu, F.-J. Wang, “Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow”, in Proc. International Conference on Software Quality, Maribor, Slovenia, 1995.
[21] W. Li, S. Henry, “Object-Oriented Metrics that Predict Maintainability”, J. Systems and Software, 23 (2), 111-122, 1993.
[22] S. Long, “Regression Models for Categorical and Limited Dependent Variables”, Advanced Quantitative Techniques, Sage publications, 1997.
[23] M. Lorenz, J. Kidd, “Object-Oriented Software Metrics”, Prentice Hall Object-Oriented Series, Englewood Cliffs, N.J., 1994.
[24] P. McCullagh and J. Nelder, “Generalized Linear Models”, London: Chapman & Hall, 1989
[25] T. Graves, A. Mockus, Identifying Productivity Drivers by Modeling Work Units Using Partial Data, Bell Laboratories Technical Report BL0113590-990513-09TM, 2000
[26] C. Mooney, R. Duval, “Bootstrapping. A Nonparametric Approach to Statistical Inference”, Quantitative Applications in the Social Sciences, 95, Sage Publications, 1993.
[27] P. Nesi, “Managing OO Projects Better”, IEEE Software, July/August 1998, p. 50-60.
[28] P. Nesi, T. Querci, “Effort estimation and prediction of object-oriented systems”, Journal of Systems and Software 42, p. 89-102, 1998.
[29] D. Steinberg, N. Cardell, “The Hybrid CART-Logit Model in Classification and Data Mining”, Salford Systems, 1999 http://www.salford-systems.com
[30] M. Stone, “Cross-validatory choice and assessment of statistical predictions”, J. Royal Stat. Soc., Ser. B 36, 111-147.
[31] D.P. Tegarden, S.D. Sheetz, D.E. Monarchi, “A Software Complexity Model of Object-Oriented Systems”, Decision Support Systems, 13(3-4), 241-262, 1995.

APPENDIX: 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 variation 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 following tables 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 a quick insight into the measures. However, such definitions tend to be ambiguous. Formal definitions of the measures using a uniform and unambiguous formalism are provided in [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 define these attributes in an objective manner, a recent and practical proposal is [9], which defines for each attribute a set of mathematical properties that a 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].

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, 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: size and inheritance measures first, then coupling, then cohesion and complexity.

In order to make possible the use of these measures at the design stage, we adapted some of the measures involving counts of method invocations as follows. Measures that are based on counts of multiple invocations of pairs of methods (say methods *m'* and *m*) were changed to 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 an 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.

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.	[18]
LCOM4	Like LCOM3, where graph G additionally has an edge between vertices representing methods *m* and *n*, if *m* invokes *n* or vice versa.	[18]
Co	Connectivity. Let *V* be the number of vertices of graph G from measure LCOM4, and *E* the number of its edges. Then $Co = 2(\lvert E\rvert - (\lvert V\rvert - 1))/((\lvert V\rvert - 1)(\lvert V\rvert - 2))$.	[18]
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 $LCOM5 = \frac{1}{a}((\sum_{j=1}^{a} m(A_j)) - m)/(1-m)$.	[17]
Coh	A variation on LCOM5: $Coh = (\sum_{j=1}^{a} m(A_j))/(m \cdot a)$	[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.	[20]

Table 20: Cohesion 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_8	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_8, 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.	[19]
DAC	Data abstraction coupling. The number of attributes in a class that have another class as their type.	[19]
DAC'	The number of different classes that are used as types of attributes in a class.	[19]
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.	[20]
IH-ICP	As ICP, but counts invocations of methods of ancestors of classes (i.e., inheritance-based coupling) only.	[20]
NIH-ICP	As ICP, but counts invocations to classes not related through inheritance.	[20]
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]

Table 21: Coupling 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.	[17]
CLD	Class-to-leaf depth. CLD of a class is the maximum number of levels in the hierarchy that are below the class.	[31]
NOC	The number of children, parent, descendent, or ancestor classes of a class.	[12],
NOP		[19],
NOD		[22],
NOA		[31]
NMO	Number of methods overridden The number of methods in a class that override a method inherited from an ancestor class.	[22]
NMA	Number of methods added. The number of new methods in a class, not inherited, not overriding.	[22]
SIX	Specialization Index. SIX is defined as NMO * DIT / (NMO+NMA+NMINH)	[22]

Table 22: Inheritance 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.

Table 23: Size Measures