Fraunhofer Einrichtung Experimentelles Software Engineering

Quality Modeling based on Coupling Measures in a Commercial Object-Oriented System

Authors:
Lionel C. Briand
Premkumar Devanbu
Walcélio L. Melo

IESE-Report No. 001.98/E
Version 1.0
January, 1998

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competetive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach
Sauerwiesen 6
D-67661 Kaiserslautern

Abstract

This paper proposes a comprehensive suite of measures to quantify the level of class coupling during the design of object-oriented (OO) systems. This suite takes into account different OO design mechanisms, such as usage, specialization, and aggregation, thus capturing different kinds of coupling in OO systems. Based on data about operational failures of a commercial software system, our coupling measures are empirically investigated by analyzing their relationship with the probability of fault detection across classes. The results demonstrate that some of these coupling measures, along with some of Chidamber and Kemerer's measures, may be useful early quality indicators of the design of OO systems. In addition, principal component analysis shows that the underlying theory on which are based our coupling measures is partially supported by evidence. The results are then compared with a previous university study: we found a significant level of consistency and stability in the results.

Keywords: coupling in object-oriented designs, C++ programming language, prediction model of fault-prone classes.

1 Introduction

Software design is defined as the process of designing system components, and the interactions between those components, to solve a specific problem. Errors in design are costly, more so than errors in coding. We should, therefore, eliminate them early, before they propagate to other phases. One well-known quality heuristic is that good software design should obey the principle of low coupling. Stevens, Myers and Constantine [27] define coupling as "the measure of the strength of association established by a connection from one module to another". Strong coupling makes a software system more complex; highly inter-related modules are harder to understand, change or correct [14]. Complexity can be reduced by designing the systems with the weakest possible coupling between modules. A software designer must, therefore, strive to minimize coupling to reduce the risk of error propagation across modules.

However, the goal of minimizing coupling contradicts some aspects of OO design, particularly the use of inheritance: Inheritance couples a class to its descendants and ancestors. Thus one might ask the following questions:

- Is inheritance coupling comparable to the standard notion of coupling?
- What is the impact of inheritance coupling on quality?

To settle these questions, we have investigated coupling in the design of OO systems and quantitativelyevaluated the impact of coupling on software quality.

The goals of this work are:

1. to define a suite of measures that quantify the level of coupling between classes in OO software systems. The suite will differentiate among different linguistic and design mechanisms.
2. to empirically evaluate the capability of this suite to predict fault-prone classes.

We have created a metrics suite designed to investigate the quality impact of the different design mechanisms in C++. We have called this suite C-FOOD (Coupling measures For Object-Oriented Design). Classes can be coupled in many different ways (e.g., specialization, generalization, aggregation, usage, and friendship [4]). The C-FOOD suite of measures distinguish these types of coupling during the design phase. We can thus provide early feedback to the software developers regarding, for example, where to focus design/code inspections or which design alternative is more appropriate.

This paper is organized as follows. Section 2 presents related work. Section 3 first discusses different forms of interactions which can be statically captured from an OO design. Then, a suite of coupling measures capturing different forms of interactions is introduced. Section 4 addresses the empirical evaluation of our OO design coupling measures. Via a standard and statistically-based classification technique, logistic regression, it is demonstrated that these measures can be used to predict fault-prone classes at the early design phases. To do so, we use process (e.g., detected faults) and product (i.e., OO design artifacts) data from a medium-size industrial OO software system. Then, our suite of OO design coupling measures is used in combination to Chidamber and Kemerer's (CK) suite thus showing that our suite of measures is, with respect to coupling, complementary to what can be considered the de facto standard metric suite in OO software development. Finally, Section 5 concludes the paper by presenting lessons learned and future work.

2 Related Work

Despite the importance of evaluating and predicting the quality of software products based on design properties such as coupling, there is little work in this area. Most existing measures capture coupling between modules using source code which is only available after implementation, e.g., [3]. Some research has addressed this issue, e.g., [11], [22], and [5], which measure coupling using OO design documents usually available before implementation. In [11], a coupling measure named Coupling Between Object classes (CBO) is defined, and empirically validated in [2]. With the CBO measure, class A is coupled to class B if A uses B's member functions and/or instance variables. CBO counts the number of classes to which a given class is coupled.

Chidamber and Kemerer's measures [11] do not distinguish between the different types of interactions two C++ classes can have and do not take into account the extent of the dependency between classes. These issues have been partially addressed by [5] where ratio scale coupling measures have been defined. These measures are used to detect difficult to maintain and fault-prone Ada packages [5]. Abstract Data Types (ADT) is used as a unit of analysis. They define different types of interactions between pairs of Ada packages: *import coupling* and *export coupling,* which capture, respectively, the impact of changes performed in external packages on a given package and the impact on external packages when changes are performed in a given package.

Our suite of C++ coupling measures is based on the suite of cohesion and coupling measures proposed in [5], e.g., the concepts of import and export coupling. To handle OO language features, in [23] the coupling measures defined in [5] have been extended to handle inheritance, usage and aggregation. However, they do adopt the coupling concepts and the underlying product abstraction representation proposed in [5].

In a previous study [9], we have further formalized the measurement framework proposed in [5] and [23] and performed a first empirical evaluation of our suite of measures by studying their relationship to the probability of fault detection across C++ classes. To do so, we have used data from an empirical study conducted at the University of Maryland with graduate and senior students. This paper complements the work described in [9] in several ways. We investigate a medium size industrial project fully designed and implemented in an OO software development environment by a team composed of senior software engineers. We analytically compare the results obtained from these two studies and explain differences between the results. We also show how well the C-FOOD suite complements already existing OO design measures: accurate predictive software quality models combining [9] and the CK suite of measures are constructed and validated. In [9], the use of friend classes was investigated and, consequently, 18 coupling measures were defined and investigated, 6 of them being related to coupling with friend classes. Since the industry system used here does not contain friend classes, these results cannot be investigated, compared with [9], or replicated.

3 A suite of coupling measures for object-oriented design

We define here fine-grained measures which capture different types of interactions between classes. Based on them, we can provide more precise guidance and feedback to software designers, e.g., which type of coupling may affect error-proneness, maintenance costs, and reusability.

There are 3 different facets, or modalities, of coupling between classes in OO systems developed with C++. We refer to them as *locus, type,* nd *relationship.* Coupling between classes in C++ can be due to any combination of these facets. Using measures that can account for all different types of interactions, we can evaluate the actual impact of each coupling dimension on the quality of the resulting artifact.

- **Relationship** refers to the type of relationship: inheritance or other (neither). Clearly, a class C is most closely coupled with all its descendants or ancestorsWe would like to measure the quality impact of coupling due to each type of relationship.
- **Locus** refers to expected locus of impact; i.e., whether the impact of change flows towards a Class (import) or away from a Class (export) . Thus changes to an ancestor flows towards a class (import) and changes to a class flows towards its descendants (export). Which direction is more important for predicting the number of faults in a class? We would like the C-FOOD measures to distinguish between them (see [5] for more details and justifications). With respect to the relationship dimension above, notice that a Class C exports impact to its descendants, and imports impact from its ancestors.
- **Type** refers to the type of interactions between classes (or their elements): It may be Class-Attribute interaction , Class-Method interaction, or Method-Method interaction. In the following, when we discuss attributes and methods of a class C, we only mean newly defined or overriding methods and attributes of C, (not ones inherited from C's ancestors in the inheritance hierarchy).

Based on these are 3 different facets, or modalities, of coupling between classes in OO systems developed with C++, we can have the following types of interactions between classes.

1. **Class Attribute (CA) interaction**
   From Figure 1, notice the CA interaction between classes A and B through the attributes **public_ab1** and **public_ab2**. Clearly, if class A is changed, class B is impacted via the two public attributes of class B that depend on class A (more precisely: its data type identified by the class name). By definition, there is no CA interaction between class B and class A directly**,** but only between elements of class B and class A. According to Booch this type of interaction is defined as "aggregation". Class B objects can aggregate Class A objects.

2. **Class-Method (CM) interaction**
   The signature of a method m_i of class c_i, can have a reference to another class c_j. Here, c_i is coupled with c_j via the method m_i. In addition, the method m_i can be a function which returns an instance (or a pointer to an instance) of c_j. Consider the declarations of classes A and B presented in Figure 1; there is an CM interaction between class A and mb1, a method of class B. According to BOOCH this type of interaction is defined as "usage".

3. **Method-Method (MM) interaction**
   Let us consider two methods, m_i and m_j, which belong, respectively, to classes c_i and c_j. If a method m_i calls a method m_j, or if m_j is passed as pa-

rameter (function pointer) to m_i, we say that there is a MM interaction between c_i and c_j through the methods m_i and m_j. For instance, consider the declarations of classes A and B in Figure 1. There is a **MM** interaction between class A and class B through the method **mb2** and **ma1**, since **ma1** is used as a parameter by the method **mb2**. This kind of interaction occurs frequently in the design of graphical user interfaces [25], e.g., call-back procedures; such low-level design interactions also occur in the design patterns literature (e.g*., Bridge, Adapter, Observer* etc). Indeed, the OMT-based notation (See *pp* 16-17, [18]) used for design patterns indicates these MM interactions.

Which type of interaction more accurately indicates fault likelihood? We would like the C-FOOD measures to distinguish these three kinds of coupling.

```
1  class B{
2       public:
3               A* public_ab1;
4               A public_ab2;
5       private:
6
7               void mb1(A &);
8               A mb2((void (A::*)());
9       …
10 };
11 void B::mb1(A& a1)
12 {    A a2;
13      …
14      a2 = mb2(&A::ma1);
15 };

class A{
        public:
...
   };
```

Figure 1: Examples of interactions between two classes.

As can be seen above, we have three types of relationship, two loci, and three types of interactions. Considering all combinations, we have 12 different possible types of coupling measures such as descendent attribute interaction export, ancestor method interaction import, and so on. This certainly leads to a large number of measures, and a corresponding increase in the difficulty of constructing tools, data gathering, and also in the analysis; however, since these types of coupling are different, arise from distinct language features, and presumably cause varying "cognitive loads" on programmers, it is important to evaluate them separately. Generally, we use the letters **"CA"** for **Class-Attribute interaction, "CM"** for **Class-Method interaction**, and "**MM**" for **Method-Method** interaction. Our goal is to define each of the 12 measures to gauge the level of coupling along the appropriate dimensions; in general, when the values of coupling are higher, we would expect more interactions. We are now ready to state the hypotheses that we intend to test in this study:

- **Hypothesis 1:** The higher the export coupling of a class C, the greater the impact of a change to C on other classes. If many classes depend on the de-

sign of C, and thus there is greater likelihood of failures being traced back to faults in C.

- **Hypothesis 2:** The higher the import coupling of a class C, the greater the impact of a change in other classes on C itself. Thus if C depends on many other classes, and the consequences are two-fold: (1) understanding C may be more difficult and therefore more fault-prone, (2) coupled classes are more likely to be misunderstood and therefore misused by C.
- **Hypothesis 3:** Identical to Hypothesis 1, but for (export) inheritance coupling to descendants.
- **Hypothesis 4:** Identical to Hypothesis 2, but for (import) inheritance coupling from ancestors.

The above 4 hypotheses are consistent with current beliefs about good OO design; there may be other hypotheses to be tested about coupling, of course. On some issues, there appears to be no popular consensus; thus for example, there seems to be no folklore on whether attribute coupling is better or worse than method coupling. In any case, the measures defined above measure coupling along several dimensions, including the import/export locus and the friend relationship, and we would expect to be able to shed light on the above hypotheses. We can now introduce the formalism to define the 12 measures. First, we present some definitions:

Definition: System

A system is defined as a collection of OO classes. Let us assume a function called **Classes** which when applied to a system S, gives the distinct classes of S, such that:

Classes(S) = { c_1, c_2, c_3, …, c_n} such that if $c_i = c_j$ then i = j where i, j = 1, …, n.

In addition, the following functions need to be specified to enable the definition of our metrics:

Ancestors(c) is a function that returns the set of classes that are the ancestors of c. Ancestors(c) refers to the base classes of c, and their base classes, and so on (closure).

Descendants(c) is a function that returns the set of classes that are the descendants of c.

Others(c) = System(S) - Descendants(c) - Ancestors(c) - {c}.

All the metrics presented in the sections below correspond to particular counts of interactions and are of the generic form:

$$\textbf{Metric}(c_i) = \sum_{c_j \in Relationship(c_i)} Interactions(c_i, c_j)$$

where the two sources of variation across metrics: Interaction(c_i, c_j) and relationship(c_i) in the formula above, corresponds to a particular type of interaction in a certain direction and a particular type of relationship, respectively, between c_i and c_j.

The acronyms for the metrics follow the rationale below:

- The first letter(s) represent the type of relationship considered (i.e, D for descendant, A for ancestor, O for others).
- The 2 letters afterwards capture the type of interaction (i.e., CA, CM, MM).
- The last 2 letters says whether this is import (IC) or export coupling (EC).

3.1 Class coupling through Class-Attribute Interaction

3.1.1 Function: Actual CA interaction(ACA)

ACA(c_i, c_j) is defined as the number of Actual Class-Attribute interactions that are present among the attribute declarations of class c_i and the class c_j. For example, from Figure 1, the Actual Attribute Interaction between class A and B is 2.

However, it should be noted that **ACA**(B,A) = 0.

3.1.2 Measures for import coupling based on CA interactions

	Interactions(C_i, C_j)	**Relationship**(C_i)
ACAIC: Ancestors CA Import Coupling	ACA(C_i, C_j)	Ancestors
OCAIC: Others CA Import Coupling	ACA(C_i, C_j)	Others

3.1.3 Measures for export coupling based on CA interactions

	Interactions(C_i, C_j)	**Relationship**(C_i)
DCAEC: Descendant CA Export Coupling	ACA(C_j, C_i)	Descendants
OCAEC: Others CA Export Coupling	ACA(C_j, C_i)	Others

3.2 Class coupling through Class-Method interaction

3.2.1 Definition: Actual CM interaction(ACM)

ACM(c_i, c_j) is defined as the number of Actual Class-Method interactions between the methods of the class c_j and the class c_i. For instance, from Figure 1, **ACM**(A,B) = 2. However, **ACM**(B,A) = 0.

3.2.2 Measures for Import Coupling based on CM interaction

	Interactions(C_i, C_j)	**Relationship**(C_i)
ACMIC: Ancestors CM Import Coupling	ACM(C_i, C_j)	Ancestors
OCMIC: Others CM Import Coupling	ACM(C_i, C_j)	Others

3.2.3 Measures for Export Coupling based on CM interactions

	Interactions(C_i, C_j)	**Relationship**(C_i)
DCMEC: Descendant CM Export Coupling	ACM(C_j, C_i)	Descendants
OCMEC: Others CM Export Coupling	ACM(C_j, C_i)	Others

3.3 Class coupling through method-method interaction

3.4 Definition: Actual MM interaction (AMM)

AMM(c_i, c_j) is defined as the number of Actual **Method-Method** interactions that are present among the methods of class c_j and the methods of class c_j.

For example, from Figure 1, **AMM**(A,B) = 1 whereas **AMM**(A,B) = 0.

3.4.1 Measures for Import Coupling based on MM interactions

	Interactions(C_i, C_j)	**Relationship**(C_i)
AMMIC: Ancestors MM Import Coupling	AMM(C_i, C_j)	Ancestors
OMMIC: Others MM Import Coupling	AMM(C_i, C_j)	Others

3.4.2 Measures for Export Coupling based on MM interactions

	Interactions(C_i, C_j)	**Relationship**(C_i)
DMMEC: Descendant MM Export Coupling	AMM(C_j, C_i)	Descendants
OMMEC: Others MM Export Coupling	AMM(C_j, C_i)	Others

4 Empirical Validation of Coupling Measures

4.1 Data Set

The data were collected from an open multi-agent system development environment, called LALO. This system has been developed and maintained since 1993 at CRIM (Centre de Recherche Informatique de Montreal); it includes 87 C++ classes with approximately 40K source lines of code (SLOC). Classes automatically generated by software tools, e.g., OO lex/yacc, or those verbatim reused from libraries, are included in this amount. Therefore, in the analysis below, these classes were not investigated since verbatim reused classes or classes generated automatically are much less likely to contain faults than those implemented or modified by software engineers [8]. In fact, the use of these classes could have biased the results.

Mostly, LALO is developed in Windows NT using Visual C++ and then ported to Sun OS and Solaris. We have investigated only the Sun OS version. Before being delivered to its 50 world-wide users, LALO is tested using structural and functional tests. During testing phases, tools are used to help the software engineers detect errors from memory manipulation which are common on C and C++ programs

In order to validate our coupling measures, we use the following information: (1) the source code of the C++ system, (2) design measurement of its classes, (3) fault data. The fault data collected report concrete manifestations of errors found by the 50 beta-testers of LALO, versions 1.0, delivered on October 1996. The faults we have investigated have been detected from October 1996 to November 1997. After these errors have been fixed, Version 1.1 of LALO was delivered.

The actual data for the suite of measures we have proposed were collected directly from the source code of the version 1.1 of LALO. This version is the product of changes performed to fix the errors found on version 1.0. To extract the design coupling measures, we have used a tool set consisting of a source code analyzer built with GEN++ [15] and some simple shell scripts. It is important to note here that the metrics were derived purely by *static* analysis; thus, MM interactions resulting via *dynamic* interactions due to run-time virtual function bindings are *not* captured. Although our results were obtained by analyzing the source code, the empirical relations that were used in our data gathering may equally well be obtained from a standard design notation such as UML [28].

4.2 Validation Strategy

When validating a product measure, there are at least four questions to be considered [7]: (1) is the measure adequately capturing the attribute it purports to measure (i.e., construct validity)? (2) is the attribute itself well-defined based on an explicit empirical model (i.e., empirical relational system) (3) is there any empirical evidence supporting the underlying hypotheses of the empirical model? (4) Is the measure useful from a practical perspective? (1) and (2) have been already addressed in the previous sections, (3) is addressed by applying univariate logistic regression analysis, and (4) is addressed by building multivariate logistic regression models for prediction. Logistic regression analysis [20] was selected here as a means to empirically validate the coupling measures we defined. Logistic regression has shown to have better properties than discriminant analysis, e.g., no distributional assumptions [M95], and is a standard statistical classification technique for which tools are broadly available.

4.2.1 Logistic Regression: a brief overview

To validate the OO design measures as quality indicators, we use a binary dependent variable aimed at capturing the fault-proneness of classes: was a fault detected in a class during testing phases? We used logistic regression, a standard technique based on maximum likelihood estimation, to analyze the relationships between measures and the fault-proneness of classes. Logistic regression has already been used in several instances to predict error-prone components [2] [5].

Other classification techniques such as classification trees [20], Optimized Set Reduction [6], or neural networks [21] could have been used. However, our goal here is not to compare multivariate analysis techniques (see [6] for a comparison) but, based on a suitable and standard technique, to validate empirically a set of product measures. We first used univariate logistic regression, to evaluate the relationship of each of the measures in isolation with fault probability. We then performed multivariate logistic regression to evaluate the predictive capability of those measures that had been assessed as sufficiently significant in the univariate analysis.

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

$$\pi(X_1, \ldots, X_n) = \frac{e^{(C_0 + C_1.X_1 + \ldots + C_n.X_n)}}{1 + e^{(C_0 + C_1.X_1 + \ldots + C_n.X_n)}}$$

where p is the probability that a fault was found in a class during the validation phase, X_i's are the design coupling measures included as explanatory variables in the model (called *covariates* of the logistic regression equation), and the C_i's are regression coefficients to be estimated. The curve between p and any single X_i—i.e., assuming that all other X_j's are constant takes a flexible S shape which ranges between two extreme cases:

(1) when a variable is not significant, then the curve approximates a horizontal line, i.e., p does not depend on X_i

(2) when a variable entirely differentiates error-prone software parts, then the curve approximates a step function.

The coefficients C_i's will be estimated through the maximization of a likelihood function, built in the usual fashion, i.e., as the product of the probabilities of the single observations, which are functions of the covariates (whose values are known in the observations) and the coefficients (which are the unknowns). This procedure assumes that all observations are statistically independent**.**

In our context, an observation is the (non) detection of a fault in a C++ class. Each (non) detection of a fault is assumed to be an event independent from other fault (non) detection. Each data vector in the data set describes an observation and has the following components: an event category (fault, no fault) and a set of OO design measures characterizing either the class where the fault was detected or a class where no fault was detected. For each measure, we provide the following statistics:

1. *Coefficient:* the estimated regression coefficient. The larger the coefficient in absolute value, the stronger the impact (positive or negative, according to the sign of the coefficient) of the explanatory variable on the probability π of a fault to be detected in a class.

2. Dy which is based on the notion of odds ratio, and provides an evaluation of the impact of the measure on the response variable. More specifically, the odds ratio $\psi(X)$ represents the ratio between the probability of having a fault and the probability of not having a fault when the value of the measure is X. As an example, if, for a given value X, $\psi(X)$ is 2, then it is twice as likely that the class does contain a fault than that it does not contain a fault. The value of Dy is computed by means of the following formula:

$$\Delta\psi = \frac{\psi(X+1)}{\psi(X)}$$

   Therefore, $\Delta\psi$ represents the reduction/increase in the odds ratio when the value X increases by 1 unit. $\Delta\psi$ provides an insight into the impact of explanatory variables and is more interpretable than logistic regression coefficients. In this study, use $\Delta\psi$'s to assess quantitatively the impact of coupling measures on π.

1. The statistical significance provides an insight into the accuracy of the coefficient estimates. It tells the reader about the probability of the coefficient being different from zero by chance. Historically, a significance threshold of α = 0.05 (i.e., 5% probability) has often been used to determine whether an explanatory variable was a significant predictor. However, the choice of a particular level of significance is a subjective decision and other levels such as α = 0.01 or 0.1 are common. Also, the larger the level of significance, the larger the standard deviation of the estimated coefficients, and the less believable the calculated impact of the explanatory variables. The significance test is based on a likelihood ratio test [20] commonly used in the framework of logistic regression.

The goodness of fit of a multivariate classification model based on logistic regression can be evaluated in several ways. First, the correctness and completeness of classification based on the model can be computed. Correctness is defined here as the percentage of classes predicted as faulty that are actually faulty. Completeness is computed as either the percentage of faulty classes that are identified as such or the percentage of faults they contain, depending on whether or not we want the completeness figure to take into account the number of faults in each class identified as faulty. In an application context where classes classified as faulty would undertake (additional) inspections, the latter measure makes more sense and we will use it in the remainder of the paper. Completeness and correctness figures have the advantage to be intuitive

and practical to compute the benefits of using a given classification model. However, completeness and correctness have to be used together to allow meaningful comparisons. This is not always practical and may lead to ambiguous comparison results.

Another way to assess the goodness of fit of a model is to use the R^2 statistic for logistic regression—not to be confused with the least-square regression R^2 [10]—they are built upon very different formulae, even though they both range between 0 and 1 and are similar from an intuitive perspective. The higher R^2, the higher the effect of the model's explanatory variables, the more accurate the model. However, as opposed to the R^2 of least-square regression, high R^2s are rare for logistic regression. For this reason, the reader should not interpret logistic regression R^2s using the usual heuristics for least-square regression R^2s. (The interested reader may refer to [24] for a detailed discussion of this issue.) Logistic regression R^2 is defined by the following ratio:

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

where:

- LL is the log likelihood obtained by Maximum Likelihood Estimation of the model for computing π above
- LL_S is the log likelihood obtained by Maximum Likelihood Estimation of a model without any variables, i.e., with only C_0. By carrying out all the calculations, it can be shown that LL_S is given by

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

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

Another way to assess the performance of a classifier is to use the Kappa statistic defined by Cohen [13]. The kappa statistic is a degree of agreement be-

tween 0 and 1 for nominal scales, 1 being a perfect agreement between predicted and actual classifications. It assumes that, in a contingency table whose columns and rows capture identical response categories coming from two different sources (i.e., predicted and actual class category: faulty, not faulty), most of the counts will be on the diagonal if the responses, i.e., predicted and actual categories, tend to agree. A Kappa values of 0 indicates that the predictions are not more diagonal than expected from chance alone. Negative values can only occur if agreement is weaker than expected by chance, but this is rare. Kappa has the advantage to have a more intuitive definition than R^2 while being also amenable to statistical testing, i.e., checking whether the association between predicted and actual classifications may be due to chance.

In the remainder of the paper, we will use the three types of indicators presented above to measure and compare the goodness of fit of the various multivariate classification models we have developed.

4.2.2 Principal Components Analysis

Principal components analysis (PCA) is a statistical techniques that can be used to analyse a set of explanatory variables to try to identify *components* Components are groups of variables can be usefully thought of as capturing some common factor or trait; they represent an underlying process or principle that creates correlations among the members of the group. The PC analysis iteratively searches for such correlated groups of variables which also show independence from other groups. One reason to use principal component analysis (PCA) [16] is to reduce the dimensionality of the sample space in which the data analysis is taking place, i.e., the number of explanatory variables to take into account. When one deals with a large number of explanatory variables, it is usually interesting to determine whether these variables capture common underlying factors or dimensions. In addition, it provides a more solid ground to interpret the measures and identify the factors they are really capturing.

PCA attempts to identify orthogonal dimensions which are the result of different linear combinations of the explanatory variables and which explain the variance of the data set in the sample space. It is hoped that the main dimensions capture most of the variance in the data set and can be used instead of the explanatory variables. This would allow us to deal with a smaller set of explanatory variables which are moreover all orthogonal, thus facilitating the contruction of multivariate regression models [16].

4.3 Analysis Results for the LALO system

This analysis investigates the coupling measures presented above but also the CK suite of measures. It is composed of the following steps. First, we look at the distributions of the measures in the LALO dataset. This is necessary to help

interpret any subsequent result in the analysis. Then, as suggested in [25], we perform a principal component analysis (PCA) to determine the actual underlying dimensions of the dataset. Indeed, despite differences in their definitions, many measures may capture similar underlying dimensions. A small number of dimensions may be used instead of the measures as potential explanatory variables, thus simplifying the subsequent analysis. PCA is also useful to help interpreting subsequent results and tells us whether our theories regarding the dimensions captured by the the CK suite and our coupling measures seem to be supported by evidence. Using logistic regression, we then analyze the relationships between fault-proneness and the principal components resulting from PCA and the measures themselves, respectively. Multivariate analysis is then performed in order to build a classification model based on principal components and the measures, respectively. Results are then compared to determine the impact of PCA on classification results and comparisons with a previous academic study (performed at the University of Maryland, referred to as UMD) are then performed to assess the stability and external validity of the results.

4.3.1 Descriptive Statistics

Looking at the measurement distributions will allow us to make more informed comparisons with the UMD study. Very often, differences in results across data sets are the results of varying distributions. We present in Tables 1 and 2 all the descriptive statistics regarding the LALO and UMD data sets, respectively.

DCAEC and DCMEC show little variance in the LALO system and are therefore not likely to show any relationship with fault-proneness. Other industrial datasets would be necessary to study their effect on fault-proneness. Many differences may be noticed between the distributions in the LALO data set and the UMD data set. OCMEC, OCMIC, OCAEC, OMMIC, and AMMIC show much larger variance, mean, and median in the LALO dataset. Since the LALO system is larger than the UMD systems, we expect overall higher class import, export, and inheritance coupling. On the other end, DIT shows a smaller variance, mean, and median in the LALO dataset, i.e., the depth of inheritance is in average not as large in the LALO system. Since the LALO programmers were trained professionals, it may be assumed they used inheritance more properly than the students who developed the UMD system. This might also explain why friend classes were not used in the LALO system.

4.3.2 Principal Component Analysis

Overview of results

On the LALO data set, when using our coupling measures and the CK suite, PCA yields 4 principal components whose Eigenvalue is above one, a usual criterion in PCA to select principal components (PCs) [16]. The Eigenvalue is the

variance of the standardized explanatory variables explained by the PC, multiplied by the number of variables and divided by one hundred. The four selected principal components (PC) represent four orthogonal dimensions in the sample space formed by all the measures. Table 3 shows, for these four principal components, what are the weightings of each explanatory variable (design measure) in the linear expression forming each PC. In addition, it shows the Eigenvalue of each PC, the percentage of variance of the standardized variables that is explained by the PC, and the cum ulative variance explained from left to right. In fact, although this is out of the scope of this paper, the principal components shown in Table 3 are rotated [16] in order to make the weightings more interpretable. The basic principle is that principal components are rotated in the sample space to minimize or maximize the weights corresponding to each explanatory variable. Rotated PCs capture the same information as non-rotated ones and are sometimes referred to as factors. The larger the absolute weight associated with a design measure, the larger the impact of this measure on the principal component.

Interpretation of results

The first (rotated) principal component (PC1) seems to capture import coupling since OCAIC, OCMIC, OMMIC, and RFC are the measures providing most of the weight for PC1. Although it also takes into account the public methods of the class to be measured, RFC can be expected to strongly influence PC1 since it mainly captures import coupling through external method invocations. Since the weights are negative, PC1 is actually inversely related to import coupling (This is just a result of the PC rotation mentioned above). PC1 actually explains 40% of the variance in the sample space. Based on a similar analysis of weights, PC2 and PC3 seem to capture export coupling and inheritance import coupling, respectively. The interpretation of PC4 is unclear. The main weights are from LCOM and, to a lesser extent, WMC and DMMEC. This dimension might capture (lack of) cohesion, which is itself somewhat related to the size of the class (WMC) and the number of method invocations from descendant classes (DMMEC).

From a general perspective, PCA confirms the existence of at least three dimensions in the measurement of design quality: import coupling, export coupling, and inheritance import coupling. This supports in part the basic theory on which is based the definition of our coupling measures since the locus and relationship facets appear to capture different visible dimensions of coupling in the LALO dataset. From a practical perspective, this suggests that coupling should be measured independently along the three dimensions above and that measures aggregating them are not likely to be optimal. On the contrary, the type facet of coupling does not seem to capture different dimensions since CA, CM, and MM interactions are systematically merged into common principal components.

However, it is important to note that PCA systematically generates low weights for low variance variables. Therefore, although there may be additional important coupling dimensions, we might not be able to see them in the LALO dataset.

4.3.3 Univariate Analysis

In this section, using logistic regression, we provide the results regarding the analysis of the individual relationship of each principal component, but also each measure, with fault-proneness.

Using Factors (rotated principal components)

Table 4 shows that the four factors identified during PCA are significant univariate predictors of fault-proneness. This basically means that all four dimensions capture underlying attributes that are relevant predictors of fault-proneness. In other words, all kinds of coupling (i.e., import, export, inheritance) are indicators of fault-proneness. Based on the regression coefficients and their standard error, it seems that Factor 2, i.e., export coupling, shows a particularly strong effect. We did not show the factors' $\Delta\psi$s here since they are not really interpretable for factors, which are based on standardized variables and do not have a measurement unit. Univariate analysis was performed by looking carefully at the impact of extreme observations or outliers. These may have a strong impact on the resulting estimated coefficients and it is therefore important to check whether an observed significant relationship is not only due to one or a few outliers, a result which would cast doubt on the validity of the observed relationship. Consequently, the results in Table 4 are not always based on the whole set of observations. In some cases, one observation was removed when it was deemed overly influential on the univariate analysis results.

Using Design Measures Directly

Table 5 shows all the design measures that appear statistically significant (α = 0.05) during univariate analysis. Many of the measures that do not appear significant do not show much variability in the LALO dataset. No trend can therefore be detected regarding these measures when using this dataset. In particular, no friend classes were used in the LALO system so that all the metrics defined in [9] for coupling between friend classes are not relevant here.

The results indicate that increased import coupling (OCAIC, OCMIC, ACMIC, AMMIC, OCMIC, OMMIC, RFC), export coupling (OCMEC, OCAEC, OMMEC, DMMEC, CBO), lack of cohesion (LCOM), and class size (WMC) increases the likelihood of a defect detection in a class. Inheritance coupling through method-method interactions, whether it is import (AMMIC, ACMIC) or export (DMMEC), also appears to have an impact on fault-proneness. This would sug-

gest that inheritance coupling with ancestors and descendants, like regular coupling, is fault-prone. Again, its impact seems significantly stronger than for regular coupling. This can be explained in part by the fact that measures like ACMIC can indicate misuses of inheritance. When inheritance is used as a means to specialize classes, then we do not expect methods in descendent classes to frequently access objects of ancestor classes. Otherwise, this might be an indication of inheritance whose purpose is only internal reuse, not specialization (also referred to as implementation inheritance).

Based on an analysis of the $\Delta\psi$s, aggregation-based coupling with/from unrelated classes and inheritance coupling seem to have a strong relationship with fault-proneness whereas other forms of coupling, such as method invocation and parameter passing, show weaker effects. These results show that Hypotheses 1-4 are supported by the results, although not all forms of coupling are equivalent with regard to their impact on fault-proneness. LCOM, although significant, has a weak effect on fault-proneness. This might also be due to the flaws identified and discussed in [8,12, 19].

4.3.4 Multivariate Analysis

This section focuses on the construction of multivariate models for the purpose of classification. Both factor and raw design measures will be used and the results compared. Factors or measures will be selected according to a typical backward regression process (as recommended in [20]) using a α value of 0.05 as a selection threshold.

Using principal components

All factors contribute to the goodness of fit of the multivariate model that predicts fault-proneness. All p-values are below $\alpha = 0.05$, but the intercept is, as expected, not significantly different from 0, i.e., if all coupling, cohesion, and size measures are equal to 0, then the probability to detect a fault is 0. This means that import, export, inheritance, and cohesion/size (the last factor is unclear) are all contributing factors to fault-proneness.The obtained R^2 and Kappa values are 0.40 and 0.472, respectively. From Table 6, we can compute that this classification model yields a completeness of 70% (of faults in classes identified as faulty) and a correctness of 73% (of actual faulty classes classified as such) when using a $\pi = 0.4$ threshold for classification. Other thresholds could have been selected, depending on how completeness and correctness are weighted for a given application. In our case, the one yielding the most balanced completeness and correctness results is used.

Using Metrics Directly

The best model is a seven measures model (WMC, CBO, LCOM, OCAEC, ACMIC, AMMIC, DMMEC in Table 7), yields a R^2 of 0.60 and a kappa value of 0.595. Again and for the same reasons, the intercept is not significantly different from 0. We can conclude that, in terms of goodness of fit and based on the resuls of the previous section, PCA does not seem to help since the model based on PCs only yielded a R^2 of 0.40. Moreover, PCA leads to a more expensive data collection since a lot of more measures need to be collected to compute the factors and use the PCA-based model. It is interesting to note that the measures selected in the measures-based model cover PC2, PC3 and PC4, three of the 4 principal components that capture most of the dataset variance. In Tables 4 and 6, we can see that PC1 shows the largest p-values (0.035 and 0.061), that is the weakest significance, and it is therefore consistent with the fact that no variable from PC1 was selected in this multivariate model.

Table 8 shows the actual count of (non) faulty classes versus their respective classifications when using the two models presented in Section 4.3.4. The counts between parentheses show the number of faults detected in the faulty classes within each cell. The design measures-based model yields a completeness of 79% and a correctness of 75%, for a threshold $\pi = 0.4$. Again, the results seem to indicate that the design measures-based model is significantly better. This is further confirmed when using the Kappa statistic [13] to measure the degree of agreement (between 0 and 1) between actual classification and the model classification. The PCA-based and design measures-based models yields, respectively, a Kappa value of 0.472 and 0.6. Both levels of agreement show a p-value = 0.0000. Therefore, despite the fact that PCA helped us identify the actual factors captured by the measures and thus facilitated the interpretation of the results, it leads to a poorer and less practical classification model.

4.3.5 Comparison with the UMD Study

The CK suite and the coupling measures presented above have already been validated by the authors on a dataset resulting from a student experiment (the UMD dataset). Further details can be found in [8,9]. We compare below the results we obtain with the LALO system with the results we obtained in these two previous validation studies. Since these two datasets come from very different environments, such a comparison should give us some insight into the external validity and robustness of the LALO results regarding the impact of design measures on fault-proneness.

We consider the univariate analysis results and compare the size effect ($\Delta\psi$) of the various coupling measures across the two datasets. Table 9 shows the $\Delta\psi$s for the UMD and LALO datasets, respectively. This allows us to compare the size effects of each design measure across datasets. Significant differences

across the two columns are investigated and tentative explanations are provided.

There are two main types of differences: (1) Some measures appear significant in one dataset only, (2) They are significant and have an effect in the same direction in the two datasets, but show significantly different impacts on fault-proneness (say a difference >20% in $\Delta\psi$). Both (1) and (2) can be at least in part explained in terms of differences between distributions across the datasets.

In category (1), we find DIT, LCOM, OCAEC, AMMIC, and DMMEC. All of these can be easily explained by large differences in means, median, and standard deviations (larger in UMD for DIT and larger in LALO for the other measures). LALO being a larger system, all these differences but the one regarding DIT are to be expected. The fact that the inheritance structure is shallow in the LALO system is consistent with other studies showing that inheritance is not extensively used in many commercial OO systems [11]. When a variable shows little variation in a data set, then there is little chance any relationship will be visible or appear to have a strong impact.

In category (2), we find ACMIC since it shows a much smaller mean, median, and standard deviation in the UMD dataset, we can easily explain the large difference in impact (39% in $\Delta\psi$).

When no strong differences in distribution are present, all the coupling measures presented in this paper have a consistent relationship with fault-proneness across the UMD and LALO studies. This shows, despite small differences, a remarkable stability of results across two datasets despite different data collection settings, i.e., students vs. professional programmers, academic experiment versus commercial system. The differences in system types are still visible in the descriptive statistics of the design measures. LALO is a much larger system and shows larger scale coupling, as expected. We did notice, however, that (perhaps because of more professionalism and rigor) the use of friend classes have been avoided and inheritance seems to be used with more caution.

5 Conclusions

This paper has proposed a comprehensive suite of design coupling measures that capture what we believe to be distinct dimensions of coupling. The analysis of an industrial C++ system has shown that these design coupling measures are, when they show variability in the dataset, good indicators of fault-

proneness, i.e., the likelihood of fault detection in a class. In addition, the analysis shows that the underlying theory on which is based our suite of coupling measures is in part supported by the results since principal component analysis shows that the underlying, orthogonal dimensions of the dataset match in part our distinct coupling dimensions. A particularly interesting point is that certain types of coupling in the inheritance hierarchies seem to be strongly fault-prone. They may capture (may be necessary) "misuses" of inheritance, e.g., implementation inheritance [27].

When, based on our coupling measures and Chidamber & Kermerer's (CK) suite of measures, multivariate logistic regression analysis is used to build an optimal classifier, then good results are obtained in terms of correctness (75%) and completeness (79%), that is the percentage of faulty classes correctly classified and the percentage of faults in the classes classified as faulty. From these two figures, the cost/benefit of using a classifier can be estimated in a given environment based on the average costs of defect detection, unecessary inspections and testing, and undetected defects. In this case, 29 classes out of 85 have been classified as faulty and might undertake additional inspections. These inspections are useless for 7 of these 28 classes since they do not contain any fault. However, 79% of all faults may be detected by inspecting 28 classes, that is 33% of all the system's classes. From these results, we may also conclude that the suite of coupling measures proposed in this paper is complementary to the Chidamber and Kemerer's suite since both types of measures appear as significant covariates in multivariate classification models.

When comparing the results obtained based on the industrial system studied in this paper and previous results obtained by the authors on student projects, many similarities are visible with respect to the impact of design measures on fault-proneness. The differences between the studies results can mostly be explained by the differences in dataset distributions, themselves being the consequence of the different nature of the compared systems, i.e., small student systems versus a significantly larger commercial system. This provides us with additional confidence regarding the use of the coupling measures presented above, as well as some of the CK measures, as design quality indicators. The results show remarkable consistency in light of the very different origins of the two datasets.

As in [5], principal component analysis did not show to be useful to improve the prediction capability of the classification models. On the other hand, principal components helped interpreting the results and assess the validity of the coupling theory presented in this paper. However, on datasets where the number of variables is larger when compared with the dataset size, PCA may still increase the chances to develop a better multivariate model when using a stepwise regression procedure, thus leading to better classification results.

Overall, the results in this paper show that useful quality models can be developed based on locally collected design measures and defect data. Although there is still room for improvement and such models are likely not to be portable across environments, they should, in the environment where they are developed, help focus inspections and testing on fault-prone parts, help control the decay of system structure during maintenance, and help assess the quality of object-oriented designs.

Our plans for the future include the refinement of the C-FOOD suite, the investigation of other measurement concepts such as cohesion, and the replication of our study on additional industrial object-oriented software systems.

Acknowledgments. We are grateful to Jürgen Wüst, from the Fraunhofer IESE, who built tools to gather the data. During this work W. Melo was, in part, supported by NSERC operation grant #OGP 0197275 and the Bell Canada Software Quality Group; his work was performed in the context of the MODL unit at CRIM. This work would not be possible without the help of the CRIM SBC Unit, especially H. Marchal and C. Saldanha.

6 References

[1] V. Basili, L. Briand et W. Melo, "How reuse influences productivity in object-oriented systems", *Communications of ACM*, 39(10):104-116, October 1996.

[2] V. Basili; L. Briand; W. Melo, "A validation of object-oriented design metrics as quality indicators.", *IEEE Trans. Software Eng.*, 22(10):751-761, 1996.

[3] J. M. Bieman and L. M. Ott, "Measuring Functional Cohesion". Computer Science Dept., Colorado State Univ., June 1993, Fort Collins, CO, USA. TR#: CS-93-109.

[4] G. Booch. "OO Analysis and design with applications". 2nd edition, Benjamin Cunnings, 1994.

[5] L. Briand; S. Morasca; V. Basili; "Defining and validating high-level design metrics". UMD-CSD, College Park, MD, USA, TR#: CS-TR-3301, 1994.

[6] L. Briand, V. Basili and C. Hetmanski, "Developing Interpretable Models with Optimized Set Reduction for Identifying High Risk Software Components," *IEEE Trans. Software Eng.*, SE-19 (11):1028-1044, 1993.

[7] L. Briand, K. El Emam, S. Morasca, "Theoretical and Empirical Validation of Software Product Measures." ISERN technical report 95-03, 1995.

[8] L. Briand, S. Morasca, and V. Basili, "Property Based Software Engineering Measurement." *IEEE Trans. on Software Eng.*, 22(1): 68-86, Jan. 1996.

[9] L. Briand, P. Devanbu, W. Melo, "An Investigation into Coupling Measures for C++". Proc. of the *19th Int'l Conf. on S/W Eng.*, Boston, USA, 1997.

[10] J. Capon, "*Elementary Statistics for the Social Sciences*", Wadworth Publishing Company, 1988.

[11] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object-oriented design.", *IEEE Trans. Software Eng.*, 20(6):476–493, 1994.

[12] N. I. Churcher and M. J. Shepperd, "Comments on 'A Metrics Suite for Object-Oriented Design'", *IEEE Trans. Software Eng.*, 21(3):263–265, 1995.

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

[14] L. Constantine, E. Yourdon, "Structured Design," Prentice Hall, 1979

[15] P. Devanbu, "A language and front-end independent source code analyzer", *Proc. of the 12th Int'l Conf. on S/W Eng.,* Melbourne, Australia, 1992.

[16] G. Dunteman, "Principal Component Analysis", SAGE publications, 1989.

[17] W. Everitt, "Cluster Analysis.", Edward Arnold, 1993.

[18] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *"Design Patterns: Elements of Reusable Object-Oriented Software*". Addison Wesley. October 1994.

[19] M. Hitz and B. Montazeri, "Chidamber and Kemerers's metrics suite: a measurement theory perspective.", *IEEE Trans. Software Eng.,* 22(4):267-271, April, 1996.

[20] D. Hosmer and S. Lemeshow, *"Applied Logistic Regression."* Wiley-Interscience. 1989.

[21] T.M. Khohgoftaar, A.S. Panday, and H.B. More, "A Neural Network Approach for Predicting Software Development Faults",*Proc. of the 3rd Int'l IEEE Symp. on S/W Reliability Engineering*, NC. 1992

[22] W. Li and S. Henry, "Object-oriented metrics that predict maintainability.", *JSS*. 23(2):111–122, 1993.

[23] W. Melo and A. Rajput, "Definition and Validation of metrics for Coupling in OO Design Metrics" , 1996, Unpublished manuscript.

[24] S. Menard. "Applied Logistic Regression Analysis", SAGE publications, 1995.

[25] J. Munson and K. Khoshgoftaar, "The Detection of Fault-Prone Programs," *IEEE Trans. Software Eng.*, SE-18 (5):423-433, 1992.

[26] R. Selby and A. Porter, "Learning from Examples: Generation and Evaluation of Decision Trees for Software Resource Analysis.", *IEEE Trans. Software Eng., 14(2): 1743-1747,* 1988.

[27] W. Stevens, G. Myers, L. Constantine, "Structured Design", IBM Systems Journal, Vol.13, 1974, 115-139

[28] Unified Modeling Language Resources, `http://www.rational/com/uml`

[29] D. A. Young, "Object-Oriented Programming with C++ and OSF/Motif", Prentice-Hall, 1992.

	Minimum	Maximum	Median	Mean	Std Dev

WMC	0.00	126.00	12.00	15.89655	17.41760
DIT	0.00	3.00	1.00	0.79310	0.85096
NOC	0.00	8.00	0.00	0.57831	1.34455
CBO	1.00	40.00	9.00	12.93103	9.26207
RFC	0.00	229.00	24.00	34.39080	35.17576
LCOM	0.00	2214.0	0.0	60.5513	343.4843
OCAIC	0.00	12.00	1.00	1.68966	2.02491
OCAEC	0.00	34.00	1.00	1.68966	4.07577
DCAEC	0.00	2.00	0.00	0.02299	0.21442
OCMIC	0.00	217.00	11.00	17.87356	32.58720
ACMIC	0.00	9.00	0.00	0.87356	1.83509
OCMEC	0.00	233.00	6.00	17.22989	35.25776
DCMEC	0.00	49.00	0.00	0.87356	5.42133
OMMIC	0.00	505.00	10.00	32.11494	76.65453
AMMIC	0.00	350.00	1.00	6.17241	37.52246
OMMEC	0.00	382.00	12.00	31.33333	64.21687
DMMEC	0.00	73.00	0.00	2.14943	8.77699

Table 1: Descriptive Statisitcs for the LALO System

	Minimum	Maximum	Median	Mean	Std Dev
WMC	1	99	9.5	13.4	14.9
DIT	0	9	0	1.32	1.99
NOC	0	13	0	0.23	1.54
CBO	0	30	5	6.8	7.56
RFC	0	105	19.5	33.91	33.37
LCOM	0	426	0	9.7	63.77
OCAIC	0	10	0	0.97	1.55
ACAIC	0	3	0	0.07	0.35
OCAEC	0	33	0	0.99	3.34
DCAIC	0	3	0	0.02	0.28
OCMIC	0	50	3	4.9	6.42
ACMIC	0	2	0	0.09	0.41
OCMEC	0	84	1	4.61	10.62
DCMIC	0	0	0	0	0
OMMIC	0	112	4	9.14	14.31
AMMIC	0	11	0	0.72	1.85
OMMEC	0	59	4	8.12	11.1
DMMEC	0	26	0	0.27	2.45

Table 2: Descriptive Statistics for the UMD systems

	PC1	PC2	PC3	PC4
EigenValue	5.206	3.227	1.528	1.103
Percent	40.048	24.822	11.751	8.488
CumPercent	40.048	64.870	76.621	85.109
WMC	-0.536	0.159	0.364	**0.586**
CBO	-0.553	**0.644**	0.289	0.181
RFC	**-0.873**	0.043	0.241	0.342
LCOM	-0.115	0.056	-0.027	**0.923**
OCAIC	**-0.745**	0.073	-0.094	0.379
OCAEC	0.015	**0.846**	0.230	-0.016
OCMIC	**-0.983**	0.014	0.041	0.093
ACMIC	0.019	0.252	**0.917**	0.022
OCMEC	0.055	**0.961**	0.206	0.027
OMMIC	**-0.954**	-0.008	-0.017	0.051
AMMIC	-0.114	0.121	**0.910**	-0.100
OMMEC	-0.075	**0.885**	-0.040	0.030
DMMEC	-0.437	-0.056	-0.190	**0.680**

Table 3: Rotated Principal Components for the LALO dataset

	Est. Coeff.	Std. Error	p-value
PC 1	- 0.44	0.24	0.0354
PC 2	1.797	0.559	0.0000
PC 3	0.739	0.232	0.0000
PC 4	0.809	0.37	0.0035

Table 4: Univariate Logistic Regression Analysis with Factors

	Est. Coeff.	Std. Error	p-value	Δψ (%)
WMC	0.1273	0.0315	0.0000	14%
CBO	0.2128	0.0389	0.0000	23%
RFC	0.0338	0.0096	0.0000	3%
LCOM	0.0361	0.0188	0.0010	4%
OCAIC	0.2992	0.1133	0.0025	35%
OCAEC	0.6945	0.1839	0.0000	100%
OCMIC	0.0423	0.0184	0.0009	4%
ACMIC	0.6469	0.1634	0.0000	91%
OCMEC	0.0620	0.0187	0.0000	6%
OMMIC	0.0112	0.0053	0.0033	1%
AMMIC	0.2327	0.0759	0.0000	26%
OMMEC	0.0327	0.0085	0.0000	3%
DMMEC	0.1348	0.0749	0.0033	14%

Table 5: Significant Univariate Logistic Regression Results with the Design Measures

Term	Estimate	Std Error	p-value
Intercept	-0.0455	0.3211	0.8873
PC 1	0.4959	0.2650	0.0613
PC 2	-2.0943	0.8132	0.0100
PC 3	-1.3858	0.4391	0.0016
PC 4	-0.8374	0.4032	0.0378

Table 6: Multivariate Logistic Regression Model with Factors

Term	Estimate	Std Error	p-value
Intercept	1.144	1.024	0.264
WMC	- 0.378	0.142	0.008
CBO	0.264	0.097	0,006
LCOM	0,149	0,067	0,0272
OAEC	0,983	0,468	0,036
ACMIC	2,020	0,558	0,0003
AMMIC	-0,416	0,213	0,051
DMMEC	0.588	0.241	0.014

Table 7: Multivariate Logistic Regression Model with Design Measures

	Factors Based Model		Measures Based Model	
Classification	Non-Faulty	Faulty	Non-Faulty	Faulty
Actually Non-Faulty	50	6	49	7
Actually Faulty	13(17 faults)	16(40 faults)	8(12 faults)	21(45 faults)

Table 8: Classification results for factors- and design measures-based models

	$\Delta\psi$ UMD	$\Delta\psi$ LALO
WMC	2%	14%
CBO	15%	23%
RFC	9%	3%
DIT	62%	NS
LCOM	NS	4%
OCAIC	38%	35%
OCAEC	NS	100%
OCMIC	10%	4%
ACMIC	52%	91%
OCMEC	12%	6%
OMMIC	12%	1%
AMMIC	NS	26%
OMMEC	6%	3%
DMMEC	NS	14%

Table 9: Size Effects Comparisons between Studies

Document Information

Title:	Quality Modeling based on Coupling Measures in a Commercial Object-Oriented System
Date:	January, 1998
Report:	IESE-001.98/E
Status:	Final
Distribution:	Public
	also published as ISERN-98-01