Proc. of the 19th International Conference on Software Engeneering, 18-23 May, 1997, Boston, MA.

An Investigation into Coupling Measures for C++

Lionel Briand
Fraunhofer IESE
Technologie Park
Sauerwiesen 6, D-67661
Kaiserslautern, Germany
briand@iese.fhg.de

Prem Devanbu
AT&T Labs Research
600, Mountain Ave
Murray Hill,
New Jersey 07974, USA
prem@research.att.com

Walcelio Melo
CRIM
1801 McGill College Ave
Montreal,
Canada H3A24
wmelo@crim.ca

Abstract

This paper proposes a comprehensive suite of measures to quantify the level of class coupling during the design of object-oriented systems. This suite takes into account the different OO design mechanisms provided by the C++ language (e.g., friendship between classes, specialization, and aggregation) but it can be tailored to other OO languages. The different measures in our suite thus reflect different hypotheses about the different mechanisms of coupling in OO systems. Based on actual project defect data, the hypotheses underlying our coupling measures are empirically validated by analyzing their relationship with the probability of fault detection across classes. The results demonstrate that some of these coupling measures may be useful early quality indicators of the design of OO systems. These measures are conceptually different from the OO design measures defined by Chidamber and Kemerer; in addition, our data suggests that they are complementary quality indicators.

Key-words: Coupling on Object-Oriented Design; C++ programming language; prediction model of fault-prone components.

INTRODUCTION

Coupling refers to the degree of interdependence among the components of a software system. Good software design should obey the principle of low coupling. Strong coupling makes a system more complex; highly inter-related modules are harder to understand, change or correct. By minimize coupling, one can avoid propagating errors 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: Should one discourage the use of inheritance and other aspects of OO technology that introduce coupling in software? Or does the old notion of coupling not hold in the OO paradigm? There are also other facilities in OO languages like C++, such as friendship, which may introduce stronger coupling. Should one discourage the use of these facilities as well? To settle these questions, we have investigated coupling in the design of OO systems and quantitatively evaluated the impact of coupling on software quality.

The goals of this work are:

a) 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 mechanisms, and between different approaches to design.

b) 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++. Classes can be coupled in many different ways (e.g., specialization, generalization, aggregation, and friendship). Our 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.

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., [8] and [4], which measure coupling using OO design documents usually available before implementation. In [8], 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 [8] 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 [4] where ratio scale coupling measures have been defined. These measures are used to detect difficult to maintain and fault-prone Ada packages [4]. 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 [4], i.e., the concepts of import and export coupling. However, to handle OO language features, we enhanced those metrics to handle inheritance and aggregation. In addition, the measures proposed in [4] do not handle coupling by friendship**,** which is specific to C++. *Friendship* is a mechanism to selectively allow the body of a class to be accessed by other classes; it reduces information hiding, and increases coupling. Thus, it is important to measure the impact of friendship on software quality. However, we do adopt the coupling concepts and the underlying product abstraction representation proposed in [4].

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 is likely to increase error density, increase maintenance costs, and/or reduce 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,* and *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: friendship, inheritance, or other (neither). Clearly, a class C is most closely coupled with all its descendants, ancestors, friends. We 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 our measures to distinguish between them. With respect to the relationship dimension above, notice that a Class C exports impact to its friends and descendants, and imports impact from its ancestors and classes that have C as their friend.

• **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).

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.

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.

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 parameter (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 [18], e.g., call-back procedures; such low-level design interactions also occur frequently in the design patterns literature (e.g., *Bridge, Adapter, Observer* etc). Indeed, the OMT-based notation (See *pp* 16-17, [12]) used for design patterns indicates these MM interactions.

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

```
1 class B{
2       public:
3               A* public_ab1;
4               A public_ab2;
5       private:
6               int i;
7               float r;
8
9               void mb1(A &);
10              A mb2((void *)());
11      …
12};
13 void B::mb1(A& a1)
14 {    A a2;
15      …
16      a2 = mb2(&A::ma1);
17 };
```

```
class A{
        public:
                int aa;
                void ma1();
};
```

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 18 different possible types of coupling measures such as friendship 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 18 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. Many classes depend critically on the design 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 C depends critically 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: Coupling based on friendship between classes is in general likely to increase the likelihood of a fault even more than other types of coupling, since friendship violates modularity in OO design.

The above 3 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 18 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:

- **Friends^{-1}(c)** is a function that returns the set of classes that have class c as a friend.
- **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).
- **Friends(c)** is a function that returns the set of classes that are the friends of c.
- **Descendants(c)** is a function that returns the set of classes that are the descendants of c.
- **Others(c)** = System(S) - Friends(c) - Descendants(c) - Friends^{-1} (c) - Ancestors(c) - {c}.

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

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

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 (two) first letter(s) represent the type of relationship considered (i.e., IF for Inverse Friend, F for Friend, 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).

Class coupling through Class-Attribute Interaction

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.

Measures for import coupling based on CA interactions

	Interactions(C_i, C_j)	**Relationship**(C_i)
IFCAIC: Inverse Friend CA Import Coupling	ACA(C_i, C_j)	Friends^{-1}
ACAIC: Ancestors CA Import Coupling	ACA(C_i, C_j)	Ancestors
OCAIC: Others CA Import Coupling	ACA(C_i, C_j)	Others

Measures for export coupling based on CA interactions

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

Class coupling through Class-Method interaction
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.

Measures for Import Coupling based on CM interactions

	Interactions(C_i, C_j)	**Relationship**(C_i)
IFCMIC: Inverse Friend CM Import Coupling	ACM(C_i, C_j)	Friends^{-1}
ACMIC: Ancestors CM Import Coupling	ACM(C_i, C_j)	Ancestors
OCMIC: Others CM Import Coupling	ACM(C_i, C_j)	Others

Measures for Export Coupling based on CM interactions

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

Class coupling through method-method interaction

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.

Measures for Import Coupling based on MM interactions

	Interactions(C_i, C_j)	**Relationship**(C_i)
IFMMIC: Inverse Friend MM Import Coupling	AMM(C_i, C_j)	Friends^{-1}
AMMIC: Ancestors MM Import Coupling	AMM(C_i, C_j)	Ancestors
OMMIC: Others MM Import Coupling	AMM(C_i, C_j)	Others

Measures for Export Coupling based on MM interactions

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

EMPIRICAL VALIDATION OF COUPLING MEASURES

Validation data

In order to validate the three hypotheses stated in the previous section, we used the data from an empirical study performed at University of Maryland (for further details see [1] and [2]).

This empirical study is not what could be called formally a *controlled* experiment since the levels of the independent variables (i.e., OO design coupling measures) are not controlled for and not assigned randomly to classes. Such a design is not practical. Our study is more observational. However, we have tried to make the results of our study as general as possible (i.e., maximizing external validity) by a careful selection of the study participants, the study material, and the development process.

We collected: (1) the source code of the C++ programs delivered at the end of the implementation phase, (2) data about these programs, (3) data about errors found during the testing phase and fixes during the repair phase. To collect items (2) and (3), we used the following forms:

- Fault Report Form.
- Component Origination Form.

A fault report form was used to gather data about (1) the faults found during the testing phase, (2) classes changed to correct such faults [1][2].

A component origination form was used to capture whether the class was developed from scratch or was derived from an existing class. In the latter case, we collected the amount of modification needed to meet the system requirements and design**:** none, slight (less than 25% of code changed) or extensive (more than 25% of code change) as well as the name of the reused class. Classes reused without modification were labeled: *verbatim reused.*

The actual data for suite of measures we have proposed were collected directly from the source code by a tool set consisting of a source code analyzer built with GEN++ [10] 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.

Validation Strategy

Logistic regression analysis [14] is used here as a means to empirically validate the coupling measures we defined. When validating a product measure, there are at least four questions to be considered [6]: (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 prediction models.

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] [4].

Other classification techniques such as classification trees [17], Optimized Set Reduction [5], or neural networks [15] could have been used. However, our goal here is not to compare multivariate analysis techniques 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):

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

where π 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**),** Y is a binary variable capturing whether or not class contains one or several faults, and the C_i's are regression coefficients to be estimated. The curve between π 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., π 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:

- *Coefficient* (appearing in Tables 2, 6 and 7, 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.
- $\Delta\psi$ (appearing in Table 2 only), 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 $\Delta\psi$ 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.
- The statistical significance (p-value, appearing in Tables 2, 6 and 7) 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 $\alpha = 0.05$ (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 $\alpha = 0.01$ 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 [14] commonly used in the framework of logistic regression.

Univariate Analysis

Descriptive Statistics

We look first at the distribution and variance of the various coupling measures we intend to validate. This will allow us to interpret more accurately the results of the next sections. In addition, it will facilitate the comparison of results in future studies and across different systems.

Table 1 shows that many coupling measures have a limited variance in our data set (which excludes verbatim reused classes, for reasons explained in the next section). For instance, measures IFCAIC, ACAIC, FCAIC, DCAIC, IFMIC, ACMIC, FCMIC, DCMIC, and AMMIC show a low standard deviation and mean below 2 interactions (the measurement units are all expressed in terms of number of interactions). As a consequence, at least in our data set, the measures with low variance are not likely to be useful predictors. This issue will be further discussed in the next section.

Univariate Logistic Regression

Table 2 shows the results when performing univariate logistic regressions with the coupling measures defined in a previous section. The analysis is only performed with classes that are either new or modified. Verbatim reused classes are typically from well-tested libraries and are usually defect-free [1]. To prevent bias in our validation study, they will not be considered. In this context, seven measures show a significant relationship with the probability π of fault detection: four import coupling measures and three export coupling measures.Two of these seven measures capture coupling with friend classes.

For example, OCAIC shows a regression coefficient of 0.32 and a $\Delta\psi$ of 1.38, which represents an increase of 38% of the odds ratio (ψ) when OCAIC increases by one unit. In addition, the regression coefficient p-value is 0.0056 and is far below the threshold $\alpha = 0.05$. The relationship between OCAIC and π is therefore significant and this result supports hypothesis 1. A detailed discussion of the results is provided below.

From Table 1 we can see that most of the coupling measures we have defined do not appear in Table 2 because they are not statistically significant (p-value > α, where α = 0.05). Therefore, none of these measures can realistically be empirically validated with our data set. Whether these measures are too rough or show more variability in other systems remains an open question. Certain relationships were made significant and /or stronger because of an unique outlier in the dataset. Therefore, in order to obtain more realistic results, they were not considered to compute the results shown in Table 2. Because of space limitations, we

cannot present a detailed outlier analysis here.

Table 3 presents rank correlations between our coupling measures by computing Spearman Rho (r_s), a well known non-parametric, ordinal measure of association. Based on these results, we can see that seven rank correlations (r_s) between coupling measures are significant (Table 3 shows them in bold characters). In order to interpret these correlation results more precisely, it is better to consider r^2_s instead of r_s since the former is a PRE (Proportionate Reduction in Error) measure of association, i.e., it indicates the degree to which errors in predicting the categories of one variable may be reduced by knowing the categories of the other variable [7]. Based on Table 3, we can see clearly that none of these correlations shows a r^2_s value approaching 1 and therefore these measures do not appear to capture similar dimensions. Although all significant, these measures seem to actually capture different dimensions of coupling.

In the paragraphs below, we will discuss and interpret the results we obtained in Table 2:

Import Coupling from "other" classes: OCAIC, OCMIC, OMMIC

When a high number of attribute interactions exist between a class C and classes which are not an ancestor, descendant, or friend of C (as measured by OCAIC), C appears to be more fault-prone. This can be explained by the fact that the class has to be developed and maintained while considering numerous dependencies between its attributes and other classes.

Similarly, when a high number of class-method or method-method interactions exist between classes which are not ancestor, descendant, or friend of a class and this class (OCMIC and OMMIC, respectively), it appears to be more fault-prone prone**.** This can be explained by the fact that designing and implementing the class' methods requires the use and understanding of different classes which are not related through inheritance or friend dependencies. Hypothesis 2 is therefore supported by empirical evidence.

Export Coupling to "other" classes: OCMEC, OMMEC

When a class interacts with many methods in classes that are not ancestors, descendants, or friends (OCMEC), then it seems to be more fault-prone. This class has therefore to be designed to satisfy the requirements of many methods.

Similarly, when many methods (defined in classes which are not hierarchically related or friend) use and depend on a class' methods (OMMEC), then the class appears to be more fault-prone. In this case, the class' methods have to fulfill the requirements of many other methods and this makes it more difficult to specify, design, and code the class. Hypothesis 1 is therefore supported by empirical evidence.

Import Coupling from friend classes: IFMMIC

When the methods of a particular class depends on many methods of friend classes, this class tends to be fault-prone. In addition, there is evidence that import coupling from friend classes makes classes even more fault-prone than import coupling from "other" classes. This is visible in Table 2 since the coefficient associated with IFMMIC is significantly higher than the one associated with OMMIC. This result supports Hypothesis 3.

Export Coupling to friend classes: FMMEC

When many external methods depend on the methods of a particular class and that these external methods belong to friend classes, this class tends to be fault-prone. Again, there is evidence that export coupling to friend classes makes classes even more fault-prone than export coupling to "other" classes. Table 2 shows a higher coefficient for FMMC as compared to OMMIC. This result further supports Hypothesis 3.

Relationships between Coupling Measures and Class Sizes

Table 4 shows rank correlations between simple size measures (i.e., source lines of code without blanks, executable number of statements) and our coupling measures. This is important in the context of certain applications of product measurement such as targeting inspections on fault-prone parts. If the model tells the developers to inspect bigger classes then this model is not of much help. On the other hand, if the model indicates fault-prone classes which are not systematically bigger than average, then the model can really help concentrate inspections on fault-prone system parts without creating an important cost overhead. Based on Table 4, we can see that OMMIC and OMMEC are strongly associated with the number of source lines of code and executable statements in the class. Therefore, larger classes may have a tendency to show more import and export coupling between methods. However, it is important to notice that, although statistically related, they still capture different dimensions (r^2_s values are not approaching 1).

Relationships between Coupling Measures and Chidamber&Kemerer's (C&K) Measures

Given the current interest, e.g. [9],[13], and [2], on the measures proposed in [8], it is also important to compare our measures with those measures. Table 5 shows that, even though several statistically significant associations exist between the coupling measures we defined and C&K measures, none of them is sufficiently strong (i.e., near $r^2_s = 1$) to claim that they capture identical or similar phenomena. In the next section, the multivariate analysis results show that our coupling measures complement the C&K measures. Many of these associations can be easily explained; but this is beyond the scope of this paper.

Multivariate Analysis

Table 6 shows the resulting regression coefficients and statistical significance when running a backward stepwise multivariate logistic regression on the whole data set minus verbatim reused classes. In addition, multivariate outlier analysis was performed (i.e., Mahalanobis distance

[11] was computed for each observation) and 7 data points (out of 113), showing to be far away in the sample space from the sample centroid (multivariate mean), were removed in order to ensure more realistic and stable results. Results when considering the outliers will also be discussed. We used, as a starting set for the stepwise regression analysis, metrics that were significant in the univariate analysis and belonging to either C&K's or our suite of coupling metrics. In addition, a dummy variable indicating whether or not the class was slightly modified (Class Origin = 0, otherwise 1) was also considered but did not appear to be a significant covariate. The multivariate model in Table 6 shows 2 covariates coming from our suite of metrics and 2 covariates coming from C&K's suite. This shows that some of our coupling metrics (OCMEC, FMMEC) are complementary quality predictors to 2 C&K metrics (DIT, RFC). OCMEC and FMMEC therefore help improve the goodness of fit of the best multivariate regression model obtained with backward stepwise regression. Therefore, they appear to be useful predictors of fault-proneness. When considering the outliers, 3 additional variables appear significant in the multivariate model (OMMEC, WMC, CBO). However, the validity of this result is uncertain and it does not change the conclusions stated above regarding the usefulness of our coupling metrics.

From a practical perspective, some measures require high-level design information only (i.e., class interface) whereas others require the program call graph which is usually (depending on the design method adopted) available later on during low-level design. In particular, some MM interactions are not visible from the class interface (although they are described in some high-level design formalisms). Several metrics that showed to be significant during univariate analysis can be completely derived from class interface information: WMC, NOC, DIT, OCAIC, OCMIC, and OCMEC. From later design stages, the remaining metrics can be derived: RFC, CBO, OMMIC, OMMEC, and IFMMIC. Table 7 shows the results of multivariate analysis when using exclusively early design metrics as covariates: OCAIC, DIT, and WMC are selected as significant covariates. Therefore, one of our coupling metrics appears to contribute significantly to a better fit of an early design quality model. The models presented in Tables 6 and 7 are the results of a backward stepwise regression analysis and slightly different models with different subsets of covariates could have been obtained if we had used a different heuristic. However, results and conclusions were in general similar.

CONCLUSIONS

Empirical results show that some of the coupling measures we defined are:

- significant predictors of fault detection probability, a reasonable measure of fault-proneness.
- complementary to Chidamber and Kemerer's measures as quality predictors.

In particular, hypotheses 1 and 2 are supported by the results since several import and export coupling measures appear to be significant predictors of fault-proneness. However, some measures did not seem significant, very likely because of their small variance in our sample. This suggests the need for replications of this study on other systems, with different distributions, to improve understanding of the relationships of these coupling measures with fault-proneness.

In addition, because measures of coupling by *friendship* show higher regression coefficients, there seems that the use of "friend" classes in C++ increases fault-proneness of classes even more than other types of coupling. This supports hypothesis 3. Again, further studies are needed in order to confirm this result .

Our plans for the future include the refinement of our measures, the investigation of other measurement concepts such as cohesion, and the replication of our study in industrial object-oriented software systems.

Acknowledgments. We are grateful to Jürgen Wüst, who built tools to gather the data and Hakim Lounis for helping us prepare the final version of this paper. During this work W. Melo was, in part, supported by Bell Canada.

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 TSE*, 22(10), 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] L. Briand; S. Morasca; V. Basili; "Defining and validating high-level design metrics", UMD-CSD, College Park, MD, USA, TR#: CS-TR-3301, 1994.

[5] 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

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

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

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

[9] N. I. Churcher and M. J. Shepperd. “Comments on ‘A Metrics Suite for Object-Oriented Design’”. *IEEE TSE*, 21(3):263–265, 1995.

[10] P. Devanbu, “A language and front-end independent source code analyzer”, in Proc. of the *Twelfth Int’l Conference on Software Engineering*, Melbourne, Australia, 1992.

[11] Everitt, “Cluster Analysis.”, Edward Arnold, 1993.

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

[13] M. Hitz and B. Montazeri. “Chidamber and Kemerers’s metrics suite: a measurement theory perspective.”, *IEEE TSE*, 22(4):267-271, April, 1996.

[14] D. Hosmer and S. Lemeshow. *“Applied Logistic Regression.”* Wiley-Interscience. 1989.

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

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

[17] Selby and A. Porter. “Learning from Examples: Generation and Evaluation of Decision Trees for Software Resource Analysis.”, *IEEE TSE, 14(2): 1743-1747.* 1988.

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

	Maximum	Minimum	Median	Mean	Std Dev
OCAIC	10	0	0	0.97	1.55
IFCAIC	2	0	0	0.07	0.32
ACAIC	3	0	0	0.07	0.35
OCAEC	33	0	0	0.99	3.34
FCAIC	4	0	0	0.10	0.47
DCAIC	3	0	0	0.02	0.28
OCMIC	50	0	3	4.9	6.42
IFMIC	8	0	0	0.46	1.47
ACMIC	2	0	0	0.09	0.41
OCMEC	84	0	1	4.61	10.62
FCMIC	8	0	0	0.49	1.5
DCMIC	0	0	0	0	0
OMMIC	112	0	4	9.14	14.31
IFMMIC	13	0	0	0.57	2.08
AMMIC	11	0	0	0.72	1.85
OMMEC	59	0	4	8.12	11.1
FMMEC	16	0	0	0.69	2.5
DMMEC	26	0	0	0.27	2.45

Table 1 Descriptive Statistics (excluding verbatim reused classes)

Measures	Coefficient	Δψ	p-value	Type	From/To
OCAIC	0.32	1.38	0.0056	Import	Others
OCMIC	0.093	1.10	0.001	Import	Others
OCMEC	0.11	1.12	0.0001	Export	Others
OMMIC	0.115	1.12	0.0000	Import	Others
IFMMIC	0.485	1.62	0.0000	Import	Friend
OMMEC	0.06	1.06	0.0000	Export	Others
FMMEC	0.868	2.38	0.0000	Export	Friend

Table 2 Significant Univariate Relationships (excluding verbatim reused classes)

	Spearman Rho (r^2_s)						
	OCAIC	OCMIC	OCMEC	OMMIC	IFMMIC	OMMEC	FMMEC
OCAIC	1	**0.40**	0.13	**0.47**	0.03	0.16	-0.17
OCMIC		1	**0.36**	**0.52**	0.23	**0.35**	-0.05
OCMEC			1	0.07	-0.04	0.06	-0.06
OMMIC				1	**0.35**	**0.61**	0.10
IFMMIC					1	0.31	-0.02
OMMEC						1	0.14

Table 3 Rank correlation's between Significant Coupling Measures

	Spearman Rho (r^2_s)					
	DIT	RFC	NOC	CBO	LCOM	WMC
OCAIC	0.17	**0.43**	-0.17	**0.24**	-0.002	0.17
OCMIC	0.05	**0.45**	**-0.30**	**0.31**	-0.09	**0.33**
OCMEC	0.23	0.07	-0.18	-0.07	0.07	**0.32**
OMMIC	0.10	0.70	**-0.27**	**0.37**	-0.03	**0.29**
IFMMIC	-0.04	**0.31**	-0.19	0.16	-0.03	0.11
OMMEC	-0.08	**0.57**	0.02	**0.38**	0.065	0.14
FMMEC	-0.15	-0.05	0.02	-0.03	0.13	0.14

Table 4 Associations between Coupling Measures and C&K Measure

	Spearman Rho (r^2_s)	
	SLOC	STMT
OCAIC	**0.24**	**0.24**
OCMIC	**0.28**	**0.42**
OCMEC	0.15	0.06
OMMIC	**0.50**	**0.63**
IFMMIC	**0.28**	**0.32**
OMMEC	**0.44**	**0.59**
FMMEC	-0.06	-0.03

Table 5 Associations between Coupling Measures and Size Measures

	Coefficient	p-value
Intercept	3.32	0.0000
DIT	1.15	0.0000
RFC	0.10	0.0000
OCMEC	0.15	0.0004
FMMEC	0.28	0.0054

Table 6 Multivariate Logistic Regression Model (see [8] for further details about C&K metrics). DIT stands for Depth of Inheritance Tree of a class and RFC for Response For a Class.

	Coefficient	p-value
Intercept	0.83	0.0320
DIT	0.86	0.0000
WMC	0.10	0.0003
OCAIC	0.1	0.0050

Table 7 Multivariate Logistic Regression Model (high-level design measures only). WMC [8] stands for Weighted Method per Class.