

ULRR

A friend in need is a friend indeed

Item Type	Meetings and Proceedings
Authors	English, Michael;Buckley, Jim;Cahill, Tony
Citation	2005 International Symposium on Empirical Software Engineering;pp. 469-478
Publisher	IEEE Computer Society
Download date	2026-05-16 23:21:36
Item License	https://creativecommons.org/licenses/by-nc-sa/1.0/
Link to Item	https://hdl.handle.net/10344/1182

A Friend in Need is a Friend Indeed

Michael English, Jim Buckley and Tony Cahill
CSIS Department, University of Limerick, Ireland.
{Michael.English, Jim.Buckley, Anthony.Cahill}@ul.ie

Abstract

Previous research has highlighted the extensive use of the C++ friend construct in both library-based and application-based systems. However, existing software metrics do not concentrate on measuring friendship accurately, a surprising omission given the debate friendship has caused in the Object-Oriented community.

In this paper, a number of software metrics, that measure the extent to which friend class relationships are actually used in systems, are defined. These metrics are based on the interactions for which the friend construct is necessary, as well as the direction of this association between classes.

Our results, in applying these metrics to the top 100 downloaded systems from sourceforge.net, indicate that up to 66% of friend class relationships in systems are redundant. Elsewhere, friend function declarations would have been more appropriate in many cases. In addition, it has been shown that friendship-based coupling contributes significantly to the high coupling of friend classes for only 25% of the systems studied.

1. Introduction

Many empirical studies of software systems, based on quality models, have been reported in the literature and are summarized in [2]. In general, these quality models attempt to predict external quality attributes from internal ones. Many of these models are based on coupling-related measurements. These include [13], [6] and [4] that have constructed models for predicting fault-proneness and [7] and [18] that focused on predicting effort from internal attributes. However, very few of the existing quality models account for the friend facility from the C++ language.

Ellis and Stroustrup, [14], define a friend of a class as a function “that is not a member of the class but is permitted to use the private and protected member names from the class”. However, a class B can also be defined as a friend of class A , allowing all the members of class B to access the private and protected members of class A . The friendship

relationship is not symmetric. In other words, B can be a friend of A without A being a friend of B .

The lack of empirical studies of the friend facility is surprising, given the debate in the literature about its appropriateness in software systems. For example, [9, 20, 17], claim that the encapsulation provided by the protected and private mechanisms is violated by friendship. In contrast Stroustrup, [21], defends the friendship mechanism, describing it as “one protection domain granting a read-write capability to another”. On the middle ground, Meyers, [19], and Booch, [1], suggest that friends should be chosen wisely.

In addition, the extensive use of the friend construct in software development in C++ has been recognized in [15, 10, 11]. In fact, Counsell *et al.*, [11], have ascertained that library-based systems “showed a distinct lack of any form of coupling (including inheritance) other than through the C++ friend facility”. Application-based software also uses the friend construct regularly, although to a lesser extent than library-based systems, [16].

Briand *et al.*, [3], studied the relationship between various design measures and the probability of fault detection during testing. They conclude that coupling based on friendship is likely to increase the likelihood of a fault. However, the friend-based coupling metrics used considered all interactions between classes where a friend class relationship existed, irrespective of whether an interaction was dependent on the friend relationship or not. Hence, it may be the case that the fault-proneness of these classes is not due to the existence of a friend relationship. In this paper, a set of friend-based coupling metrics are proposed. The metrics proposed measure the extent to which each friend relationship in a system is actually exploited. In other words, these metrics only measure the interactions between classes which are enabled by the friend construct and thus can be used to determine if linking fault-proneness to the use of the friend relationship is legitimate.

In the next section we review guidelines for metric refinements from the coupling arena, and current work in friend metrics. We combine these to identify a need for friend metric refinement which is discussed in section 3. The resultant metrics are defined in section 4. In section 5

our empirical study is described. Section 6 discusses our results in applying our friend-based metrics to the software systems analyzed and section 7 concludes the paper with a summary of results and discussion of future work.

2. Related Work

The friend construct enables specific coupling relationships to be forged between classes. However, existing coupling measures do not distinguish friend-enabled coupling from other forms of coupling. In this section, we consider guidelines for coupling measurement (which include measurements that consider friend relationships). These measurements form the basis of the friend-based coupling measures defined in section 4.

In the context of coupling metrics, Wilkie and Kitchenham, [23], have highlighted the different levels of granularity that can be associated with the coupling measure, i.e. some metrics count linkages at a class level, whereas others count connections between member functions and consider the types of the parameters of these functions. In [24], they propose a framework for the refinement of Coupling Between Objects (CBO), a coupling measure defined by Chidamber and Kemerer, [8]: “CBO for a class is a count of the number of other classes to which it is coupled”. They propose the following five refinement stages:

- Separate the CBO count based on the direction of the association
- Count the number of distinct message-passes using each association
- Consider the number and nature of the parameters in each distinct message-pass
- Count the frequency of occurrence of each type of message pass
- Dynamically measure the frequency of usage of each message-pass for given execution scenarios

These five stages have been used to define a number of coupling metrics. For example, the first stage of this process has been addressed by Wilkie and Hylands, [22], who introduced the idea of forward and backward coupling. Wilkie and Kitchenham, [24], addressed stage 1 of this refinement process by introducing the metrics CBO(forward) and CBO(back). The second refinement which counts the number of distinct message-passes using a particular linkage, in a particular direction was addressed by the metrics: Coupling Complexity in the Forward direction (CCF), and Coupling Complexity in the Backward direction (CCB), [24].

Briand *et al.*, [3], define coupling metrics which address the first stage of Wilkie and Kitchenham’s refinement

Metric	Relationship	Locus	Type
IFCAIC	Inverse Friend	Import	Class Attribute
IFCMIC	Inverse Friend	Import	Class Method
IFMMIC	Inverse Friend	Import	Method Method
FCAEC	Friend	Export	Class Attribute
FCMEC	Friend	Export	Class Method
FMMEC	Friend	Export	Method Method

Table 1. Briand *et al.*’s Friend Metrics

process through the notion of import and export coupling, which capture the *locus* of impact of changes on classes. In addition, they consider two other dimensions of coupling measurement: *relationship* and *type*. Friendship is a form of relationship within these dimensions as would be for example, inheritance and association. The type of relationship categorises how classes are connected, e.g., an object of one class might be an attribute of another class, giving a class-attribute type of connection. The friend-based metrics defined by Briand *et al.* and based on these dimensions are summarized in table 1. So, for example, FMMEC for a class containing friend class declarations counts all method-method interactions between classes declared friends of the class and the class itself.

Briand *et al.* and Wilkie and Kitchenham have examined the usefulness of their metrics as predictors of external quality attributes. Briand *et al.* showed that a number of their metrics were useful for identifying faulty classes. These included both IFMMIC (method-method interactions for classes declared as friends) and FMMEC (method-method interactions for classes declaring friends). Briand *et al.* concluded that import coupling from inverse friend classes and export coupling towards friend classes makes classes even more fault prone than import or export coupling towards or from other classes.

Thus, design constructs like friendship have been empirically shown to influence the fault-proneness of classes, [3]. However, more sophisticated metrics which associate interactions between classes with the design construct which enables the interaction (e.g., the friend construct is needed to access the private members of a class) can be used to refine the measurements which have been used in the past and the validate the results of previous studies. Some refined metrics for measuring friendship accurately are defined in this paper and will be used to assess any links between friendship and external attributes of systems in future work.

Other coupling metrics are too coarse-grained when considering friendship specifically. As mentioned above, the CCF(1) metric defined by Wilkie and Kitchenham measures all the distinct message passes from a class to all other classes that the class is connected to. Therefore, this metric does not consider the friend construct specifically either.

The CBO metric only measures coupling at a class level and does not consider either the type of connection or the interactions within a specific connection. Likewise, the Response Set for a Class (RFC), [8], defined as a count of all methods in the class, plus all methods that can potentially be executed in response to a message received by an object of the class, but does not consider the type of relationship needed to facilitate any method invocation.

In this paper more accurate metrics for friendship measurement are defined and are used to assess friendship usage in systems.

3. Extending Coupling Metrics for Friends

A number of refinements to existing metrics are proposed in this paper to improve the measurement of the friend construct. These metrics address stage 1 of Wilkie and Kitchenham's framework, in that they consider the direction associated with a particular relationship, which in turn distinguishes a class that contains friend class declarations from a class that is itself declared as a friend. However, we propose an intermediate step between stage 1 and stage 2 of this framework and a subsequent reformulation of stage 2. These refinements merge the approaches of Briand *et al*, [3], and Wilkie and Kitchenham, [24], discussed above. The enhancements we suggest are:

- For each identified class linkage determine all *types of inter-class relationship* using this linkage. These inter-class relationships include friendship, inheritance and other types of association.
- For each *type of relationship* using a specific class linkage, count the number of message-passes *which are enabled by this type of relationship* and use this linkage.

Therefore, for 2 classes involved in a friend relationship specifically, stage 2 suggests that we associate the specific linkages to the friendship relationship. The benefits of these enhancements are two-fold:

1: They allow the analysis of interactions which result from specific types of relationships between classes and thus allow for the separate analysis of the usage of such relationships across systems.

2: In relation to coupling measurement and in turn to the construction of predictive models of software quality, measuring the usage of each type of relationship across a class linkage separately, allows each measure to be included or excluded from a predictive model on an individual basis and also allows different weighting functions to be applied to the measures.

4. Definition of Friend-based Metrics

The five metrics defined below have been categorized on the basis of the direction associated with the coupling, i.e. forward or backward, thus acknowledging stage 1 in Wilkie and Kitchenham's framework, [24]. The metrics which are defined for classes that are declared as friends are considered forward-coupling metrics, since these classes have the potential to invoke hidden members of other classes. Similarly, the metrics defined for classes which declare friends are considered backward coupling metrics, since hidden members in these classes can be invoked by other classes.

A friend class declaration is exploited if the class specified in the declaration accesses 'hidden' members of the class containing the declaration. In this paper, members include both methods and attributes of a class and hidden members refer to those that cannot be called on an object of the class without the use of the friend construct. If the declaration is not exploited in this way, then it is considered redundant.

4.1. Forward Coupling Friend Metrics

In this section the metrics: Actual Friend Methods (AFM), Actual Friend Classes (AFC), Coupling Complexity in the Forward Direction for Friends (CCFF(1)), and Response set For Friend Class (RFFC(1)) are defined.

4.1.1 AFM and AFC

By definition, a friend class declaration is equivalent to granting friendship to all the methods in a class, [14]. Therefore, all the methods in a class declared as a friend could potentially exploit the friend class declaration. Currently however, there is no measure or statistical analysis available to express the number or proportion of methods in a class which exploit a friend class relationship.

As a result, the metric, Actual Friend Methods (AFM), counts the number of methods in a class that *access hidden members* of classes which declare the current class as a friend. AFM gives some insight into the extent to which a friend-class relationship is actually used. Small values of AFM would suggest that friend function declarations may be more appropriate than friend class declarations. By limiting friend declarations to friend function declarations, then the scope of possible functions which can access 'hidden' members of other classes is minimised. Given the considerable variation in opinions on the appropriateness of the friend construct outlined earlier, it seems that the best approach is to use the friend construct cautiously by minimising the number of member functions granted friend status.

Consider figure 1, which is based on the Buhr diagrams from Ada, [5]. The public interface methods of a class are

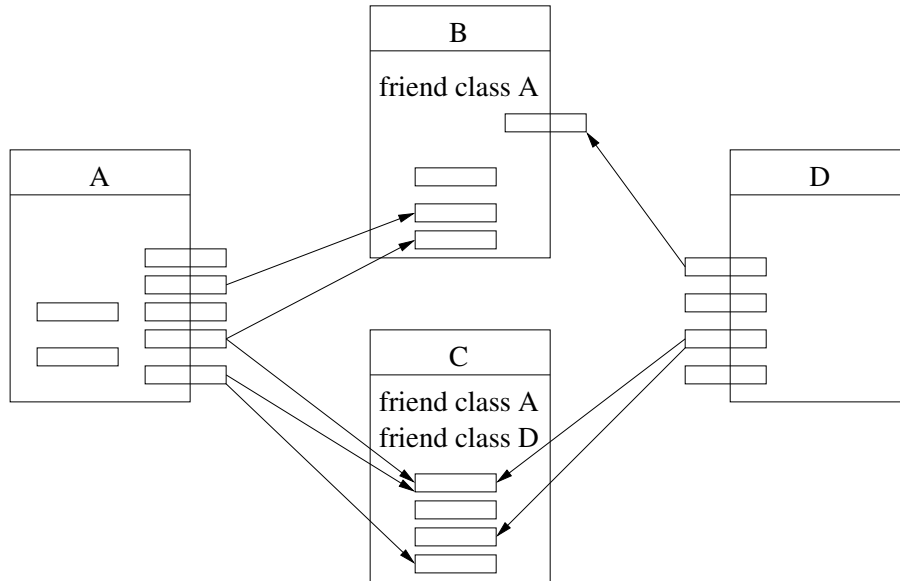


Figure 1. Illustration of the calculation of new metrics

shown as small rectangles on the perimeter of the larger rectangles, which represent the actual classes themselves. The hidden members are those drawn completely inside the class rectangles. Classes *A* and *D* are both declared as friends. The actual friend methods of *A*, $AFM(A)=3$ since three methods of *A* call hidden members of the classes in which *A* is declared a friend, i.e. *B* and *C*. *A* has a total of seven methods, i.e., $WMC(A)=7$ (for Weighted Methods per Class (WMC) a weight of 1 is applied to each method). For class *D*, $AFM(D)=1$ and $WMC(D)=4$. Given that $AFM(D)=1$, suggests that declaring a specific method of class *D* as a friend of class *C* would be more appropriate than declaring class *D* itself as a friend of class *C*.

At a system level, the number of friend classes, each of which exploits at least one friend-enabled relationship (where the class is declared as a friend), could also be maintained. This number of Actual Friend Classes (AFC), indicates the number of friend classes which are actually exploited through friendship. The proportion of actual friend classes, relative to the number of classes declared as friends gives an indication as to the usage, rather than the declaration of friendship in a system.

4.1.2 CCFF(1)

The metric $CCFF(1)$ is a refinement of the $CCF(1)$ metric of Wilkie and Kitchenham and measures the coupling complexity in the forward direction for classes declared as friends. This new measure, $CCFF(1)$, counts the number of distinct interactions with hidden methods and attributes in classes which declare this class as a friend. This metric dif-

fers from $CCF(1)$ on two counts:

- 1:** $CCFF(1)$ counts attributes which are accessed as well as methods which are called, in line with the Briand *et al.* metrics.
- 2:** Due to the nature of the friend construct, $CCFF(1)$ only counts accesses to hidden members of classes which declare the current class as a friend i.e., the accesses allowed by friendship.

From this metric, the proportion of *all* interactions, which correspond to *actual* friend interactions can be calculated. If this proportion is high, then many of the interactions between a class and classes declaring it as a friend involve accessing the hidden members in these classes. This could highlight a problem in the class design structure. However, it is expected that for many classes $CCFF(1)$ will be small.

In figure 1, $CCFF(1)(A)=5$, where each interaction is denoted by an edge from a method in *A* to a hidden member in *B* or *C*, since these are the two classes in which class *A* is declared a friend. $CCFF(1)(D)=2$, since there are two distinct interactions which access hidden members of class *C*, and class *C* is the only class which declares class *D* as a friend.

4.1.3 RFFC(1)

The $RFFC(1)$ metric for a class declared as a friend measures the *response set* for the subset of methods in the class which *access hidden members* of other classes. It counts the number of distinct hidden members which are accessed in classes which declare the class as a friend. This can be

Metric Name	Class/System	Forward/Backward	Description
AFM	class	forward	no. of methods in a class that utilize friendship
AFC	system	forward	no. of classes that utilize friendship
CCFF(1)	class	forward	no. of interactions dependent on friendship
RFFC(1)	class	forward	no. of hidden members accessed by class
CBOF(Back)	class	backward	no. of classes which access hidden members in class
AFCR	system	backward	no. of friend class declarations utilized
MAF	class	backward	no. of members accessed by friends

Table 2. Summary of Friend-based Metrics

seen as a refinement of Chidamber and Kemerer's RFC(1) metric. If RFFC(1) is small, then the use of the friend construct could be avoided by reclassifying the accessed hidden members of classes. Whether or not this is appropriate is, of course, a class design issue.

In figure 1, RFFC(1)(A)=4, since class *A* accesses 2 hidden members of class *B* and 2 hidden members of class *C*. These are the only 2 classes to which class *A* is declared a friend. In this case the proportion RFFC(1)/RFC(1), gives another indication of the extent to which accessing hidden members consumes the total response set for a class. In addition the ratio of CCFF(1) to RFFC(1) gives the average number of times each hidden member is called. A high value of this average would indicate that the classes declared as friends focus intensely on a relatively small number of hidden members.

4.2. Backward Coupling Friend Metrics

In this section the metrics Coupling Complexity in the Backward Direction for friends (CBOF(Back)), Actual Friend Class Relationships (AFCR), and Members Accessed by Friends (MAF), are defined.

4.2.1 CBOF(Back) and AFCR

In focusing on classes which declare friend classes, two additional metrics have been defined. Coupling Complexity in the Backward direction for Friends (CBOF(Back)), counts the number of declared friend classes which actually access hidden members of the class, i.e. that utilize the friend declaration. This metric is a refinement of the CBO(Back) metric defined by Wilkie and Kitchenham, [24]. This metric could be expressed as a proportion of the total number of friend class declarations in a class. If this proportion does not equal one then there are redundant declarations of friend classes. In figure 1, CBOF(Back)(B)=1 and CBOF(Back)(C)=2. In this example there are no redundant friend declarations.

Obviously this metric must evaluate to be less than or equal to the number of friend class declarations in a class.

Previous research has shown that the distribution of the number of friend class declarations of a class is strongly skewed towards 0, [16]. Therefore, it is to be expected that the values of CBOF(Back) will in general be very small.

The Actual Friend Class Relationships (AFCR) metric is a system level metric, like AFC. AFCR is the total sum of all the friend class relationships actually exploited in a system. Therefore, it is the sum of CBOF(Back) for all the classes in the system. This metric can be compared with the total number of friend class declarations in a system to establish the level of redundancy in friend class declarations.

4.2.2 MAF

The final metric, Members Accessed by Friends (MAF), counts the number of *hidden* members in a class *declaring* friends, which are accessed by classes which are declared friends of the class. MAF is a refinement of the CCB metric, [24]. However, it only counts those message passes for which the friend relationship is required. A small value for this metric might indicate the inappropriate application of an access specifier to the specific class members accessed. A large value might indicate the consistent use of the friend construct to facilitate coupling.

In figure 1, MAF(C)=3 and MAF(B)=2. An alternative indication of the extent to which classes declared as friends access hidden methods, could be provided by the proportion of the total number of hidden members in a class which are accessed by classes declared as friends.

4.3. Summary of Friend-based Metrics

Table 2 summarizes the forward coupling metrics and the backward coupling metrics defined in section 4.1 and section 4.2 respectively.

5 Empirical Study

In this section, the research questions being evaluated are outlined and the software systems used for this evaluation are discussed.

System Name	No. Classes	Fr. Classes	AFC	%AFC	%AFRC	Regression Analysis
Abiword	1336	86	60	69.7	50	yes
Audacity	190	22	18	81	73	yes
Bzflag	431	58	32	55	91.3	yes
dc++	205	12	7	58.3	85	yes
emule	381	53	31	58.4	63.8	yes
emulemorph	401	60	32	53.3	58.2	yes
emuleplus	227	26	15	57.6	52.9	yes
filezilla	132	7	6	85.7	87.5	yes
firebird	202	7	3	42.8	35.7	yes
gnucleas	35	4	3	75	75	yes
lame	21	5	3	60	60	yes
licq	342	28	24	85.7	78.4	yes
scummvm	403	27	20	74	63.2	yes
shareaza	401	76	68	89.4	84.3	yes
ultravnc	134	12	8	66.7	56.3	yes
Winscp	228	20	18	90	80	yes
Wxwidgets	1628	36	36	100	31.66	yes
7zip	434	6	6	100	85.7	no
Cdex	160	2	2	100	100	no
Celestia	234	1	1	100	50	no
CVS GUI	387	7	4	57.1	40	no
dscaler	69	2	0	0	0	no
eraser	97	3	3	100	100	no
mynapster	137	3	2	66.7	100	no
stepmania	309	2	1	50	33.3	no
VBA	113	2	1	50	33.3	no
Virtualdub	591	6	5	83.3	83.3	no
WDM	122	3	2	66.7	33.3	no

Table 3. Friend Usage in the Software Systems

5.1. Research Questions

In this empirical study we address a number of research questions which illustrate the utility of these refined metrics. The research questions address the level of redundancy in friend class usage in systems, i.e. discrepancies between declared and actual usage of friendship in systems, the appropriateness of the usage of friend classes as opposed to friend functions and the influence of friend-based coupling on the high coupling of classes declared as friends, [15]. The 4 research questions are:

1. Do classes, which are declared as friends in a system, exploit their friend-class relationships?
2. For classes which are declared as friends and which exploit the friend-class relationships, are friend function declarations more appropriate than friend class declarations, i.e., does the friendship need to be class-wide?
3. Is friend-based coupling a statistically significant con-

tributor to the high coupling of classes declared as friends?

4. Does the usage of friendship suggest that the level of protection granted to specific class members be re-evaluated?

5.2. Applications under study

The metrics defined in section 4 have been extracted from open-source software systems which have been obtained from sourceforge.net. Our purpose is to evaluate the research questions defined above using these metrics.

The applications studied in this analysis have been reported in [16]. Brief details are provided here for completeness. The analyzed systems were taken from the top 100 downloaded systems from sourceforge.net and included systems which were developed either partially or totally in C++. While download statistics do not directly reflect the usefulness of software, it does give some indication that

the products have reached a quality threshold acceptable to users.

Of the systems downloaded, 37 were at least partially developed in C++. Of these 37 systems, 33 systems contained some usage of the friend construct and 28 contained classes declared as friends. In this analysis we are focusing on systems which declare friend classes. Summary statistics in relation to system size, in terms of the number of classes and use of friendship is provided in table 3 for the 28 systems which declare friend classes.

A number of the 28 systems which declare friend classes, contain very few friend class declarations. Analyzing these systems statistically is not worthwhile since attempting to attain some statistical significance between two groups, one of which has a large number of values and the other has a very small number of values is difficult to achieve. Therefore, in the linear regressions reported below, we only report on the regression applied to 17 systems (see table 3). In these systems the number of friend class declarations was at least 5% of the number of classes in the system.

6. Results

In sections 6.1 to 6.4 each of the four research questions defined above are evaluated using appropriate friend-based metrics.

6.1. Do classes, which are declared as friends in a system, exploit their friend-class relationships?

The percentage of actual friend classes (AFC), in each system is provided in table 3. This corresponds to the percentage of classes declared as friends in each system for which $AFM > 0$. For the systems in this study, this percentage ranges from 42.8% to 100%. For about 60% of all systems the percentage of friend classes which exploit some of the friendship available to them is less than 75%. Therefore, for more than half of the systems in this study at least 25% of friend class declarations are redundant.

The CBOF(Back) metric counts the number of declared friend classes of a class, which access hidden members of the class containing the friend declarations. Therefore this metric provides a measure of redundancy at the class level. This metric returns mostly small values. There are just 2 systems with a median value greater than 1 for CBOF(Back). In addition, the maximum value of CBOF(Back) is less than 9 for all but one system. The maximum CBOF(Back) value for any system is 20. CBOF(Back) must be less than or equal to the number of declared friend classes of a class.

The AFCR metric, which highlights redundant friend class relationships at the system level is presented in table 3

as a percentage of the total number of friend class relationships in each system. This percentage ranges from 31.6% to 100%. The systems with a small number of friend class relationships tend to have less redundancy in terms of these relationships. This is to be expected since each friend class relationship in such a system is probably created for a specific purpose for which the friendship is necessary. For 19 of the 28 systems presented in table 3, the percentage of friend class relationships exploited is less than 80% compared with 17 of the 28 systems in which less than 80% of classes declared as friends were exploited. Therefore, approximately 66% of systems have a redundancy level greater than 20%.

6.2. For classes which are declared as friends and which exploit the friend-class relationships, are friend function declarations more appropriate than friend class declarations?

The AFM metric is a measure of the number of methods in a class which exploit friend class declarations involving the class. The median value for this metric is 1 for many systems. None of the 28 systems have a median value greater than 3. For some systems the maximum value is small, whereas for others the maximum value seems to be much larger than the 90th percentile, indicating that in a very small number of cases the friend relationship is exploited to a large extent. The large proportion of small values for this metric suggests that friend function declarations may have been more appropriate than friend class declarations. It may be the case that friend classes are declared instead of friend functions for convenience purposes or to facilitate the future evolution or maintenance of systems. In addition, it may be the case that while friend function declarations indicate directly the functions which access the hidden members, the amount of code that would need to be considered if friend classes were used might not be significantly bigger. In addition the use of friend function declarations might require the regular inclusion or exclusion of these function declarations thus eliminating any maintenance related benefits that could be gained by using these friend function declarations instead of friend class declarations.

The CCFF(1) metric also provides an indication of the extent to which friend-class declarations are exploited. For 12 of the 28 systems, the median value of CCFF(1) is ≤ 3 . This suggests that in almost 50% of systems, 50% of the classes declared as friends are involved in less than or equal to 3 interactions that are dependent on the friend construct. While more accurate measures of the amount of redundancy in friend class relationships in systems are defined below, this metric suggests that the redundancy level related to friendship is quite large.

6.3. Is friend-based coupling a statistically significant contributor to the high coupling of classes declared as friends?

Previous results have shown that classes declared as friends have higher coupling than classes which are not declared as friends, [15]. The coupling measurement on which this result was based on the class-level coupling measure CBO(forward), defined in [24]. The question arises as to whether this higher coupling is due solely to friend class couplings or whether other types of coupling relationships also influence the difference in coupling between classes declared as friends and classes which are not declared as friends.

In this section, we determine if previous results hold, which relate friendship and coupling. However, we use the more refined metrics CCF(1) and RFC(1) as the basis for coupling measurement in this analysis. The friend-based metrics defined above are also analyzed to determine the extent to which friend class declarations are exploited. Finally, by excluding the friend-based coupling measures from CCF(1) and RFC(1) we determine if classes declared as friends still have higher coupling than classes which are not declared as friends.

CCF(1) and RFC(1) have been extracted for all of the classes in each of the 37 systems. In doing so, these metrics have been extracted for the subset of classes which are declared as friends, since the friend-based metrics are only defined for this set of classes.

A linear regression analysis was performed to determine if classes which are declared as friends are significantly good predictors of classes with high coupling. The two measures of coupling used here were CCF(1) and RFC(1). The dependent variable in the analysis was either CCF(1) or RFC(1) and the only independent variable was the boolean variable which determined whether or not classes were declared as friends.

Our results indicate that 13 and 14 of the 17 systems showed a significant result for CCF(1) and RFC(1) respectively. Due to the nature of the distributions of these metrics we can conclude that classes declared as friends have significantly higher coupling than classes which are not declared as friends. This reinforces our previous results which used the coarser grained metric, CBO(forward), [15].

In addition, the size of each class, in terms of lines of code, was added as a second independent variable to the regression model for each of the dependent variables, since previous work by El Emam *et al.*, [12], suggests that some object-oriented metrics are confounded by size when predicting the fault-proneness of classes. In the regression model with CCF(1) as the dependent variable, 9 of the 13 systems which returned significant results without controlling for size, still returned significant results when control-

ling for size. In the regression model with RFC(1) as the dependent variable, 7 of the 14 systems returned a significant outcome when controlling for size. Therefore, classes declared as friends have significantly higher coupling than classes not declared as friends in 50% to 75% of systems, even when controlling for size. However, the question still remains as to whether or not this extra coupling in classes declared as friends is due to friend-based interactions between classes or other forms of interaction.

If friend-based coupling is the source of the higher coupling which exists in classes declared as friends, then a linear regression with the friend-based coupling removed should eliminate the relationship between classes declared as friends and high coupling. Therefore, the linear regression of the previous section has been reapplied here with an amendment: the coupling due to friendship has been removed from the dependent variable. The new dependent variable is CCF(1)-CCFF(1) for one regression and RFC(1)-RFFC(1) for the second regression. In using CCF(1)-CCFF(1) as the dependent variable 10 of the 13 systems which returned significant values with CCF(1) as the dependent variable still return a significant value. In addition, 10 of the 14 systems which returned a significant result with RFC(1) as the dependent variable still return a significant result with RFC(1)-RFFC(1) as the dependent variable. This suggests that in approximately 25% of systems, coupling due to friendship has a significant impact on the ability of classes declared as friends to predict classes with high coupling. Therefore, coupling due to friendship, contributes critically to the overall coupling of the classes concerned. For the remaining 75% of systems coupling due to friendship can be seen as only a contributing factor to the overall coupling of the classes declared as friends. However, there must be other forms of coupling present in the classes in these systems which ensure that their coupling is still significantly bigger than the coupling of classes declared as friends. Further research is required to determine the prevailing coupling mechanisms in these systems.

6.4. Does the usage of friendship suggest that the level of protection granted to specific class members be re-evaluated?

The MAF metric counts the number of 'hidden' members in a class which are accessed by classes declared as friends. If CBOF(Back) for a class is zero, then MAF for that class will also be zero. If CBOF(Back) is positive then MAF must be positive. However, a class might only declare one friend class, but this friend class may access many of the hidden members of the class declaring it as a friend, thus returning a large MAF value.

For the systems analyzed here a large proportion of classes declaring friend classes return a zero value for MAF,

again indicating that a considerable number of friend declarations are not exploited and as a result no hidden members inside classes which declare friends are accessed directly. In all but 2 systems the median value is less than or equal to 3. However, the distribution of this metric indicates that its maximum value is large in comparison to the maximum value for CBOF(Back). There are only 8 systems where the maximum value is less than or equal to 3. We can conclude that for all systems at least 50% of classes declaring friends only access a small number of hidden members, suggesting that the access specifier applied to these members might not be appropriate, i.e. that a hidden member of a class should not be hidden. However, this is a design decision and the MAF metric can only be used as a guideline in such decisions.

The RFFC(1) metric also provides some indication of the number of hidden members which are accessed in a class. However, if a class is declared as a friend of more than one other class, then this metric counts the number of hidden members accessed across all these classes.

The median value of RFFC(1) for 26 of the 28 systems was less than or equal to 3. Therefore, in almost all systems 50% of classes declared as friends require access to less than or equal to 3 hidden members in other classes. This metric provides an alternate indication of how much a friend class relationship is used. In many systems a large proportion of classes have an RFFC(1) value of zero, indicating that no use is made of any friend relationship, where this class is declared as a friend. In addition, small values of RFFC(1) suggest that, if appropriate, by changing the access specifier (public, protected or private), of a small number of members in the affected classes might have avoided the need to use the friend construct. However, given that the research question focuses on classes containing friend declarations, the MAF metric is the most appropriate measure to use to evaluate this question.

7. Threats to Validity

This section highlights some of the possible threats to the validity of the work carried out in this paper.

It is important to note that all the metrics have been extracted from the systems purely by static analysis. Thus connections created between classes dynamically are not considered.

Other possible threats to the validity of this study relate to the systems analysed. These have been highlighted in [16] but will be briefly outlined here. All the systems analysed are open-source software systems. Therefore, our results may not generalise to commercial software. The quality of the systems developed is unknown, however these systems were some of the most downloaded systems from sourceforge.net. In some systems multiple languages were

used in development whereas for others C++ was the only development language. There is also a large variation in the size and also in the level of maturity of the systems analysed.

8. Conclusion

Previous research has shown the widespread use of the friend construct in software systems. In this paper, the lack of metrics which measure friendship usage accurately has been highlighted, resulting in blunt measures of a phenomenon that has been widely debated in the literature. A number of metrics have been defined to address this limitation in friendship measurement. These metrics are based on an existing framework for the refinement of coupling measures, [24]. We suggest that the direction and the number of message passes which are facilitated by each type of relationship (e.g. friendship), should be counted separately. This refinement permits the definition of a set of metrics which focus on different types of relationships and thus has wider implications than for friend-based coupling measurement alone.

The 5 metrics defined here are useful in explaining the extent to which friend class declarations are exploited and give an indication, at the class level, of the amount of redundancy involved in friend class relationships.

Two additional metrics have been specified at a system level. They collate some of the metrics outlined above to identify the number of classes declared as friends and the number of friend class relationships which are utilized. These metrics provide information at a system level in relation to the amount of redundancy in friend class declarations.

Analyses are presented in this paper to illustrate the potential worth of these metrics. They suggest that there is up to 66% redundancy of friend class relationships in some systems, with a redundancy level between 30% and 40% common in the systems analysed. In addition to this, our results also suggest that friend class declarations are used, where friend function declarations might be more appropriate in many cases. The analysis also suggests that the level of protection assigned to some class members should be reconsidered, especially where friend classes only access a small number of hidden members in a class.

The results presented here confirm that classes declared as friends have higher coupling than other system classes, using CCF(1) and RFC(1) as the basis for coupling measurement. Furthermore, the results suggest that for about 25% of these systems, coupling due to the friend construct is critical in this relationship. For the remaining 75% of systems, coupling based on friendship is only a contributing factor, to ensuring that classes declared as friends have higher coupling than other classes in a system. Other forms

of coupling form the significant element of coupling relationships in these systems. These other forms of coupling need further investigation.

All of the metrics defined in this paper have the potential to be useful predictors of external quality attributes. Our future work will involve the construction of prediction models for external attributes of systems incorporating these metrics.

Our future work will also incorporate an analysis of why friend class declarations are used in situations where friend function declarations would suffice. The removal of redundant friend class declarations and the replacement of actual friend class declarations with friend function declarations would reduce the occurrence of friend class declarations enormously and have the effect of focusing any maintenance tasks involving friend declarations towards a much smaller code base, reducing the amount of maintenance effort involved.

Acknowledgments

We acknowledge the helpful comments from the anonymous reviewers which have been included in this paper and the financial support of the CSIS Department for funding this research.

References

- [1] G. Booch. *Object Oriented Design with Applications*. The Benjamin/Cummings Publishing Company Inc., 1991.
- [2] L. Briand and J. Wüst. Empirical Studies of Quality Models in Object-Oriented Systems. *Advances in Computers*, 59:97–166, 2002.
- [3] L. C. Briand, P. T. Devanbu, and W. L. Melo. An Investigation into Coupling Measures for C++. In *International Conference on Software Engineering*, pages 412–421, 1997.
- [4] L. C. Briand, J. Wüst, J. W. Daly, and D. V. Porter. Exploring the relationships between design measures and software quality in object-oriented systems. *The Journal of Systems and Software*, 51(3):245–273, 2000.
- [5] R. J. A. Buhr. *System design with Ada*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1984.
- [6] M. Cartwright and M. Shepperd. An Empirical Investigation of an Object-Oriented Software System. *IEEE Transactions on Software Engineering*, 26(8):786–796, August 2000.
- [7] S. Chidamber, D. Darcy, and C. Kemerer. Managerial Use of Metrics for Object-Oriented Software: An Explanatory Analysis. *IEEE Transactions on Software Engineering*, 24(8):629–639, 1998.
- [8] S. R. Chidamber and C. F. Kemerer. A Metrics Suite for Object Oriented Design. *IEEE Trans. Software Eng.*, 20(6):476–493, 1994.
- [9] J. O. Coplien. *Advanced C++ Programming Styles and Idioms*. Addison-Wesley Longman Publishing Co., Inc., 1992.
- [10] S. Counsell and P. Newson. Use of Friends in C++ Software: An Empirical Investigation. *Journal of Systems and Software*, 53(1):15–21, 2000.
- [11] S. Counsell, P. Newson, and E. Mendes. Design Level Hypothesis Testing Through Reverse Engineering of Object-Oriented Software. *International Journal of Software Engineering*, 14(2):207–220, 2004.
- [12] K. El Emam, S. Benlarbi, N. Goel, and S. N. Rai. The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics. *IEEE Trans. Softw. Eng.*, 27(7):630–650, 2001.
- [13] K. El Emam, W. Melo, and J. Machado. The Prediction of Faulty Classes Using Object-Oriented Design Metrics. *Journal of Systems and Software*, 56:63–75, 2001.
- [14] M. A. Ellis and B. Stroustrup. *The Annotated C++ Reference Manual*. Addison-Wesley Longman Publishing Co., Inc., 1990.
- [15] M. English, J. Buckley, T.Cahill, and K. Lynch. An Empirical Study of the Use of Friends in C++ Software. In *International Workshop on Program Comprehension*, pages 329–332, May 2005.
- [16] M. English, J. Buckley, T.Cahill, and K. Lynch. Measuring the Impact of Friends on the Internal Attributes of Software Systems. In *Fifth International Workshop on Source Code Analysis and Manipulation*, September 2005.
- [17] R. Harrison, S.Counsell, and R. Nithi. An Overview of Object-Oriented Design Metrics. In *International Conference on Software Technology and Engineering Practice (STEP)*, pages 230–234. IEEE Computer Society Press, July 1997.
- [18] W. Li and S. Henry. Object-Oriented Metrics that Predict Maintainability. *Journal of Systems and Software*, 23(2):111–122, 1993.
- [19] S. Meyers. *Effective C++*. Addison-Wesley, 1998.
- [20] M. Page-Jones. *What Every Programmer Should Know About Object-Oriented Design*. Dorset House Publishing, 1995.
- [21] B. Stroustrup. *The Design and Evolution of C++*. ACM Press/Addison-Wesley Publishing Co., 1994.
- [22] F. G. Wilkie and B. Hylands. Measuring Complexity in C++ Application Software. *Software - Practice and Experience*, 28(5):513–546, April 1998.
- [23] F. G. Wilkie and B. Kitchenham. Coupling Measures and Change Ripples in C++ Application Software. *The Journal of Systems and Software*, 52:157–164, 2000.
- [24] F. G. Wilkie and B. Kitchenham. An Investigation of Coupling, Reuse and Maintenance in a Commercial C++ Application. *Information & Software Technology*, 43(13):801–812, 2001.