Measuring the Impact of Friends on the Internal Attributes of Software Systems

Michael English,[1] Jim Buckley,[1] Tony Cahill,[1] and Kristian Lynch[2]
[1]Department of Computer Science and Information Systems, University of Limerick, Ireland.
[2]Dept of Clinical Sciences, Lund University, Sweden.
{Michael.English, Jim.Buckley, Anthony.Cahill}@ul.ie
Kristian.Lynch@med.lu.se

Abstract

Differing views have been expressed on the appropriateness of the friend construct in the design and implementation of object-oriented software in C++. However, little empirical analysis of its usage in the development of systems has taken place. This paper describes an empirical study of the top 100 most downloaded open-source software systems from sourceforge.net, to assess the design implications of the use of the friend construct.

On a larger cohort of systems our results provide further evidence that classes declared as friends are coupling hotspots. Our findings also indicate that class size does not seem to have the confounding effect on this relationship that has been suggested in previous research. In addition, there seems to be no indication that friendship is used as an alternative to inheritance although a small number of systems seem to use friendship as an alternative to multiple inheritance and vice versa.

1 Introduction

In [4], many of the empirical studies of software systems that have taken place are summarized. Most of these studies focus on predicting external quality attributes from internal object-oriented design measures. The external characteristics assessed are fault-proneness, effort, productivity and maintainability. The internal measures can be classified as variations of coupling measures, cohesion measures, inheritance measures and size measures. In this paper the friend construct of C++ is examined in depth by studying its effect on other internal design measures.

Briand *et al.*, [5], have pointed out that empirical studies of systems' structure and quality are needed to provide actual evidence of what constitutes good object-oriented design and several studies have been performed which apply a set of design measures to systems and investigate relationships between these measures, [1, 9, 15].

A friend of a class has been defined 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", [13]. A class C, can also be defined as a friend of another class, D. This makes all member functions of C friends of class D.

Little analysis exists of friend usage in systems or on its impact on internal or external quality attributes, (notable exceptions being [9, 15]). This seems surprising given the diametrically opposing views on its appropriateness which exist in the literature. For example, [8, 18, 16], claim that the encapsulation provided by the protected and private mechanisms is violated by friendship. In contrast Stroustrup, [19], defends the friendship mechanism, describing it as "one protection domain granting a read-write capability to another". On the middle ground, Meyers, [17], and Booch, [3], suggest that friends should be chosen wisely.

2 Related Studies

One study that did assess friendship with respect to other design measures was Counsell and Newson, [9]. They performed a case-study on the use of friends in four C++ software systems. From five initial hypotheses, they found that "classes containing friends that engage in inheritance have less descendants than other classes engaged in inheritance". This suggests that friends are more likely to be found deep in the inheritance hierarchy, thus obtaining access to extra functionality. They also found that "classes that do not engage in any inheritance have more friends than classes that do engage in inheritance", suggesting that classes use friends as an alternative to inheritance.

A subsequent study, [15], refined the study of Counsell and Newson by:

- Omitting all overloaded operators defined as friends, from the measurements taken, since such usage is enforced by the syntactic restrictions of C++.
- Refining the statement of some of the hypotheses;

System Name	Number of Classes	KLOC	Stand Alone Classes	Inh. Trees	Avg Classes per Tree	Max DIT	Mult Inh.	More than 1 Parent	Fr rels	Fr Cls.	Fr Fcns.	Fr Ops	Cls declared as Fr	Classes declaring Fr	%Fr	%FrCl
Abiword	1336	420	256	85	12.7	5	26	26	186	182	4	0	86	147	13.9	13.6
Audacity	190	166	58	12	11.0	3	5	3	30	26	4	0	22	23	15.8	13.7
Bzflag	431	106	171	28	9.3	3	1	1	59	58	1	1	36	50	13.7	13.5
Dc++	205	114	51	35	4.4	3	118	0	28	27	0	0	12	24	13.7	13.2
Emule	381	135	103	22	12.6	6	26	20	84	72	12	11	53	42	22.0	18.9
Emulemorph	401	180	121	15	18.7	6	18	18	98	79	19	18	60	45	24.4	19.7
Emuleplus	227	98	53	6	29.0	3	15	15	40	34	1	1	26	22	17.6	15.0
Licq	342	71	56	12	23.8	5	5	5	86	65	21	1	28	38	25.1	19.0
Scummvm	403	463	134	45	6.0	8	10	10	38	38	0	0	27	29	9.4	9.4
Shareaza	401	129	131	9	30.0	8	0	0	96	96	0	0	76	61	23.9	23.9
Ultravnc	134	95	91	7	6.1	2	0	0	17	16	1	0	12	11	12.7	11.9
Winscp	228	129	18	10	21.0	4	3	3	33	30	3	0	20	15	14.5	13.2
Wxwidgets	1628	280	349	68	18.8	6	18	18	328	120	206	203	75	90	20.1	7.4
Gnucleas	35	11	9	0	0	1	0	0	4	4	0	0	4	1	11.4	11.4
Lame	21	30	7	1	14.0	2	3	1	7	5	2	2	5	3	33.3	23.8
Filezilla	132	61	31	6	16.8	2	4	4	8	8	0	0	7	6	6.1	6.1
Firebird	202	1103	35	13	12.8	3	1	1	14	14	0	0	7	9	6.9	6.9
Celestia	234	136	91	18	7.9	2	5	5	15	2	13	12	1	2	6.4	0.9
Wdm	122	32	25	23	4.2	4	15	7	8	6	2	2	3	6	6.6	4.9
Cdex	160	91	24	4	34.0	4	1	1	2	2	0	0	2	1	1.2	1.2
Virtualdub	591	154	152	91	4.8	6	53	44	7	6	1	1	6	5	1.2	1.0
CVSGUI	387	69	75	22	14.2	5	13	12	12	10	1	0	7	9	3.1	2.6
7zip	434	77	186	31	8.0	4	215	115	7	7	0	0	6	7	1.6	1.6
AC3filter	32	19	13	6	3.2	2	5	4	0	0	0	0	0	0	0.0	0.0
Flaskmpeg	115	76	53	17	3.6	3	4	4	3	0	1	0	0	0	2.6	0.0
VBA	113	77	13	8	12.5	2	16	14	4	3	1	0	2	3	3.5	2.7
Dscaler	69	140	13	2	28.0	6	65	30	2	2	0	0	0	1	2.9	2.9
Bo2k	34	20	10	0	0	1	0	0	1	0	1	0	0	0	2.9	0.0
Mynapster	137	46	34	3	34.3	2	1	1	4	3	1	0	3	3	2.9	2.2
Tuxracer	4	32	4	0	0	0	0	0	0	0	0	0	0	0	0.0	0.0
Eraser	97	33	13	3	28.0	2	2	2	4	3	1	0	3	3	4.1	3.1
Stepmania	309	88	83	13	17.4	6	3	3	5	3	2	2	2	3	1.6	1.0
Dosbox	119	69	21	17	5.8	2	0	0	2	0	2	0	0	0	1.7	0.0
Bochs	91	113	13	5	15.6	4	0	0	0	0	0	0	0	0	0.0	0.0
Ypops	12	6	11	0	0	1	0	0	1	0	1	1	0	0	8.3	0.0
Genesis3d	48	186	10	0	0	2	0	0	1	0	1	0	0	0	2.1	0.0
Doomlegacy	3	110	3	0	0	0	0	0	0	0	0	0	0	0	0.0	0.0

Table 1. Summary Metrics for the 37 Software Systems

- Refining the measurements used in some hypotheses;
- Using a more rigorous approach to the statistical analysis;

This study found that:

- Classes declared as friends of other classes have more non-friend coupling than classes which are not declared as friends, implying that classes declared as friends are coupling hotspots in systems.
- The more declared friends of a class, the more private and protected members in that class. This is as expected since friends are only necessary to access the protected and private members of a class.
- Classes that do not engage in any inheritance have more friends than classes which do engage in inheritance, suggesting again that the friend mechanism is used as an alternative to inheritance.

Our results clearly supported the first two hypotheses. A variation of the second hypothesis indicated that a correlation didn't exist between the number of inherited protected members and the number of friends declared in a class. This seemed to indicate that friends are only used to access protected and private members in a class and not any inherited protected members. This outcome was also supported by the absence of a link between the Depth of Inheritance, (DIT), for classes engaged in inheritance which contain friends and those engaged in inheritance which do not contain friends. There was no evidence to suggest that classes declared as friends of other classes have less inheritance than other system classes but there was some support for the third hypothesis outlined above.

3 Empirical Study

In order to enlarge our previous study, [15], we chose to examine the top 100 most downloaded projects from www.sourceforge.net. 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 these systems, 37 were at least partially developed in C++. Summary statistics of all 37 systems is given in table 1.

Of the 37 systems, 33 systems contained some usage of the friend construct and 28 contained classes declared as friends. Previous analysis, [14], [15], indicated considerable usage of friendship to facilitate operator overloading, where it was reported that up to 50% of all friend relationships were used for this purpose. In this analysis, the friend construct is used to facilitate operator overloading in 12 of the 37 systems. However, there are just 4 systems where the mechanism is used more than twice. Only one system contains considerable usage of friendship to facilitate operator overloading, suggesting that this is not, in fact, a common practice in the systems studied.

Sourceforge.net allows system developers to supply information in relation to the topic and intended audience of each system hosted by Sourceforge. The 37 systems analyzed here have been categorized as word processors, multimedia tools, networking tools for file sharing, backup and instant messaging, games and mail transport agents.

In [15] a large quantity of friendship was found in a study of library-based systems. The proportion of friend relationships relative to the number of classes ranged from 40% to 200% for the four systems analysed. This is not surprising given that friends may be used to support interface design and for efficiency purposes in libraries. Counsell *et al.*, [11], have highlighted the large proportion of friend declarations in library-based systems and showed that library-based systems have a lack of other forms of coupling. Bieman and Zhao, [2], have suggested that developers put more effort into the design of libraries than into the design of application software, implying that such extensive usage of friendship is based on design rationale. Based on our analysis in this study of the end-user profile supplied by sourceforge, 34 systems could be described as application-based systems and 3 as library-based systems. In systems generally, friendship usage is lower than previously reported, ranging from 0% to 33%. Surprisingly in 2 of the 3 library-based systems friendship usage was low too. However, the final library-based system has 230 friend relationships, the largest number of friend relationships of any system in this study. This system is also one of the 4 systems mentioned previously which uses friendship consistently to facilitate operator overloading. In fact 45% of all friend relationships correspond to overloaded operators, (105 operator overloads, compared to 18 for the system with the next most frequent usage of operator overloading). For this library-based system it is clear that the friend construct is used extensively, especially to facilitate operator overloading, in a similar way to 3 of the library-based systems of the previous study, [15].

The properties detailed in table 1, may provide insight into the rationale underlying the decision to use the friend construct or may highlight trade-offs that had to be considered in its usage. For example, friendship has been suggested as an alternative to inheritance, [9, 10]. Thus, it might be expected that systems which declare friend classes may use less inheritance and hence may have a lower overall depth of inheritance, a smaller proportion of classes engaged in inheritance or a smaller average number of classes per hierarchy. It might also be the case that classes are declared as friends to avoid the use of multiple inheritance.

In addition, given the data in table 1, it seems worthwhile

to consider if the systems which contain classes declared as friends are bigger than the systems which do not declare friend classes.

3.1 Statement of System Hypotheses

We have formulated the following hypotheses to establish if any of the relationships described above can be established between the systems analyzed from the data presented in table 1.

- **System Hypothesis 1:** Systems with classes declared as friends are bigger than systems without declared friend classes.

  If the size difference between systems is found to be statistically significant then it should be considered in the interpretation of the analysis of the other system variables. For the purposes of this hypothesis the size of the systems is measured in terms of the number of classes in them.

- **System Hypothesis 2:** Systems with declared friend classes have less inheritance than systems without declared friend classes.

  To establish if there is any indication that friendship is used as a substitute for inheritance, we studied how the maximum depth of inheritance, the percentage of classes engaged in inheritance and the average number of classes per inheritance hierarchy, varied between the two groups of systems.

- **System Hypothesis 3:** Systems with classes declared as friends have less multiple inheritance than systems without classes declared as friends.

  Alternative measures of multiple inheritance are provided in the columns 'More than 1 Parent' and 'Mult Inh' in table 1. The 'More than 1 Parent' column returns the number of classes which have more than one parent in each system. The column entitled 'Mult. Inh.' is an alternative measure of multiple inheritance and counts the number of parents, greater than one which the classes in a system have, i.e. a class with n parents contributes $n - 1$ to this measure.

3.2 Statement of Class Hypotheses

Based on the work of Counsell and Newson, [9], and English *et al.*, [15], the effect of class friendship on coupling at a class level within systems needs to be investigated.

We are interested in studying systems with habitual use of the friend construct, the longer term goal being to examine the architectural trade-offs involved in its use. Thus, in examining the class hypotheses we will consider systems which contain at least 5% friend class relationships, avoiding systems with less friendship since such usage may be of a random nature. Some of the 37 systems developed in C++ had a very small number of references to the friend construct. Thus we will apply the class hypotheses outlined below to 17 systems.

In section 2 we discussed the 3 hypotheses statistically supported in the findings of [15]. These hypotheses were:

- **Class Hypothesis 4:** Classes declared as friends of other classes have more non-friend coupling than classes which are not declared as friends.

- **Class Hypothesis 5:** The more declared friends of a class, the more private and protected members in that class.

- **Class Hypothesis 6:** Classes that do not engage in any inheritance have more friends than classes which do engage in inheritance.

However, this evaluation was on a small pool of systems, of a predominantly library nature. In this study we repeat class hypotheses 4, 5 and 6 for a larger, more general cohort of systems. In addition to this we consider the size of classes as a confounding factor.

El Emam *et al.*, [12], have shown a confounding effect of class size on the validity of some object-oriented metrics. Therefore we apply hypothesis 7 outlined below to establish if classes which are declared as friends tend to be larger than other classes in software systems.

- **Class Hypothesis 7:** Classes declared as friends are bigger than other system classes.

This hypothesis has been evaluated based on two measures of size, Weighted Methods per Class, (WMC) and Lines of Code, (LOC).

In addition, to investigate the confounding effect of size, we have carried out a multivariate analysis to determine if CBO is significantly different for classes declared as friends than for classes not declared as friends, when controlling for size.

4 Results

4.1 Statistical Techniques

To evaluate the system hypotheses an analysis of the summary system variables is presented in table 2. In each case we applied the Mann-Whitney statistic to determine if any differences between the two groups of systems are statistically significant. If the p-value is statistically significant then the 25, 50 and 75 percentiles along with the minimum

Variable	p-value	Systems with Classes:	Percentiles				
Size			25	50	75	Min	Max
(No. of Classes)	.000	Not Declared as Friends	8	34	103	3	119
		Declared as Friends	132.5	227.5	401	21	1628
Max DIT	.002	Not Declared as Friends	0.5	2	2.5	0	4
		Declared as Friends	2	4	6	1	8
% Inheritance-based Classes	.076	Not Declared as Friends	2	61.5	81.25	0	86
		Declared as Friends	67.75	75	81	32	92
Avg. No. Classes per inh. tree	<.001	Not Declared as Friends	0	0	4.7	0	15.6
		Declared as Friends	7.9	13.4	23.1	0	34.3
Multiple Inheritance	.001	Not Declared as Friends	0	0	2	0	5
		Declared as Friends	1.25	5	18	0	215

Table 2. Comparison of System-level characteristics for systems with and without friend classes

and maximum values of the two groups for each variable are used to illustrate the nature of the differences.

The Pearson Chi-Square Test of Association was used to evaluate class hypotheses 4, 5, 6 and 7. The results are presented in table 3. Class hypothesis 2 was tested using the Spearman correlation coefficient and the results of this test are shown in table 4. Finally a multivariate regression was performed to assess the influence of class size, with the results presented in table 5.

4.2 Discussion of Results

In this section the results of the statistical analysis for each of the hypotheses are discussed. In addition the results of the multi-variate analysis, controlling for size, are also discussed.

4.2.1 Evaluation of System Hypotheses

System Hypothesis 1: Systems with classes declared as friends are bigger than systems without declared friend classes

The p-value presented in table 2 for this analysis is statistically significant, ($< .001$). In fact, each of the percentiles and the minimum and maximum values for systems with classes declared as friends are considerably larger. Therefore, we can conclude that systems which declare friend classes are significantly larger than other systems.

System Hypothesis 2: Systems with declared friend classes have less inheritance than systems without declared friend classes

The maximum DIT and average number of classes per hierarchy both returned significant p-values, (see table **??**), both indicating that systems with declared friend classes have a significantly higher maximum DIT and average number of classes per hierarchy. Given this result for either of these variables, the same result would be expected for the other. However, given that we have already shown that systems which contain classes declared as friends are bigger than other systems, the larger values for the maximum DIT and average number of classes per hierarchy may be because these systems are bigger than systems which do not contain classes declared as friends. The p-value for the percentage of inheritance-based classes in systems is not significant, indicating that any difference in the percentiles or minimum and maximum values between the two groups may be down to chance. Thus, we cannot confirm that the use of friendship is inversely related to the use of inheritance.

System Hypothesis 3: Systems with classes declared as friends have less multiple inheritance than systems without classes declared as friends

It is possible that friendship is used instead of multiple inheritance in systems. Table 2 illustrates that systems which declare friend classes use multiple inheritance significantly more than systems which do not declare friend classes, suggesting that friendship is not used as an alternative to multiple inheritance. However, it is worth noting that a small number of systems which have a lot of multiple inheritance have very little friendship. Table 1 illustrates that one system has 115 classes with more than one parent, approximately 26% of all classes in the system. This system has by far the most usage of multiple inheritance of all the systems irrespective of size but only 1.6% of its classes are declared as friends. Two other systems with little use of the friend construct also make considerable use of multiple inheritance. In addition, there are a number of systems which have a high number of friend classes and a small amount of multiple inheritance. For example, one class has 96 friend

classes and no multiple inheritance, another has 182 friend classes and 26 classes involved in multiple inheritance.

However, these systems are not in line with the statistical findings and illustrate an inconsistency in design considerations with respect to multiple inheritance and friends across systems. Most systems which use multiple inheritance often, also seem to use friendship relatively consistently. In most systems where multiple inheritance is used classes tend to have just two parents with about three exceptions in this analysis.

A Spearman correlation was also performed to investigate further how multiple inheritance and friendship are related. However, the Spearman statistic returned a significant positive correlation, indicating that in general as the use of multiple inheritance increases so does the use of friendship. The system size may have a confounding effect in this analysis since we have already seen that systems without friend classes are much smaller than other systems. In other words the small systems tend to have less multiple inheritance than larger systems.

The final consideration in relation to friendship and inheritance is whether or not a class is also a friend of its parent class. This type of usage could not impact multiple inheritance but may illustrate alternative uses of the friend construct. This scenario arises very seldom, with one system showing 3 occurrences and 3 systems with just one occurrence of this characteristic.

From this analysis we can conclude that while there seems to be some systems which use friendship instead of multiple inheritance or systems which have considerable multiple inheritance but do not use friendship, this association is not statistically significant for the systems analyzed in this study. There are also systems which use both multiple inheritance and friendship consistently.

If coupling to existing class hierarchies in a system is required, then system maintainers may consider it unwise to alter these existing hierarchical structures. Therefore, it may be the case that the friend construct is used to facilitate coupling with classes within these inheritance hierarchies. Further analysis is required to determine if this is actually the case.

4.2.2 Evaluation of Class Hypotheses

Class Hypothesis 4: Classes declared as friends have more non-friend coupling than classes not declared as friends

Of the 17 systems under study, this hypothesis was accepted for 13 of the systems, with $\alpha = .05$, (see table 3). This lends further support to our original findings that classes which are declared as friends have higher class couplings than other system classes (using CBO as our coupling metric). The possible reasons why this hypothesis

System	H4	H6	H7(a)	H7(b)
1	<.001	.505	<.001	<.001
2	<.001	.097	.003	.061
3	.004	.066	.002	.005
4	.002	.402	.227	.028
5	<.001	.005	<.001	<.001
6	<.001	.003	<.001	<.001
7	<.001	.709	<.001	<.001
8	<.001	<.001	<.001	<.001
9	.220	.405	.001	.190
10	.044	.004	.016	.005
11	.010	.463	.874	.082
12	.004	.423	.419	.002
13	.019	<.001	<.001	<.001
14	.024	.731	.170	.031
15	.271	.146	.003	.144
16	.490	.551	.376	.454
17	.292	.417	.852	.808

Table 3. P-values for the Chi-Square Statistic

	Hypothesis 5	
System	p-value	Spearman
1	.800	.007
2	<.001	.340
3	<.001	.234
4	<.001	.464
5	<.001	.218
6	<.001	.211
7	.028	.149
8	.064	.100
9	.041	.107
10	.967	.002
11	<.001	.306
12	.161	.094
13	.258	.029
14	.001	.280
15	.002	.222
16	.092	.289
17	.454	-.173

Table 4. Spearman Statistics for Hypothesis 5

fails for four of the systems are as follows:

- Two of these systems have only four and five classes declared as friends. These systems are very small, with 35 and 21 classes respectively. The next smallest system studied, in terms of the number of classes, has 132 classes. These two systems are outliers, (in terms of size), of the group of systems which declare friend classes. While the analysis in section 3 showed that systems with classes declared as friends are significantly bigger than systems without classes declared as friends, it is clear from table 2 that the size of these systems is much closer to the median size, (34), of systems which do not declare friend classes.

- It is difficult to achieve a statistically significant difference between two groups when one of the populations is very small, as is the case for both these systems, with 4 and 5 classes declared as friends. The chi-square statistic calculates the difference between the expected and observed frequency for each cell in the cross tabulation. If one row total is very small then differences calculated between observed and expected frequencies will also be small. Therefore it is more difficult to achieve a statistically significant outcome.

- These systems also contain very little inheritance, with just one inheritance hierarchy in one system and none in the other. Therefore it may be inappropriate to expect the same conclusions to hold for these small systems as the larger systems in the study. It may be the case that design considerations, such as information hiding, may not have received much consideration prior to coding and thus coupling in this system may specifically be facilitated by the use of the friend construct. The next system for which the hypothesis didn't hold has 202 classes but only 7 classes are declared as friends, corresponding to 3.5% of classes in the system, the smallest percentage of all systems studied.

- There is one remaining system which is of a reasonable size, and has a significant amount of declared friend classes for which this hypothesis does not hold. This system also contains a relatively small proportion of friend relationships. In addition, the number of classes, inheritance trees, average number of classes per tree and the maximum DIT give an idea of the shape of the inheritance trees in the system. There are 45 inheritance trees, with an average of six classes per tree, which is relatively small. The maximum DIT of 8 indicates that while the hierarchies may be deep they are quite narrow. These measures indicate that there is much potential for members to be inherited through each hierarchy. However, our coupling metric does not take inheritance into account in the evaluation of CBO, possibly resulting in a relatively lower CBO value for this system than for the remaining systems.

Class Hypothesis 5: The more declared friends of a class, the more private and protected members in that class.

The statistical results for this hypothesis are shown in table 4. A significant correlation exists between the declared friends of a class and the number of private and protected members in the class, for 10 of the 17 systems under study. While the Spearman statistic illustrates a positive correlation in all but one case, this correlation is weak, with only 1 of the 10 systems illustrating a correlation greater than 0.4.

To gain a better understanding of this result, it is interesting to study the distribution of the "declared friends of a class" variable, for each system. The distribution of this variable illustrates that the 25, 50 and 75 percentiles are zero, for all 17 systems. Having such a large proportion of classes with no declared friends is bound to influence the search for a linear correlation, by mapping the number of private and protected members of these classes to a zero value, corresponding to no declared friend classes. Furthermore, for all but one system more than 85% of classes in each system do not declare any friends. For the systems which did not show any correlation for this hypothesis, this percentage tended to be over 90% is most cases.

There are two variations of this hypothesis which may provide more meaningful results. The distribution considered above, suggests that it might be more appropriate if this hypothesis is only considered for classes which declare at least one friend, thus eliminating all zero values from the correlation. However, another consideration in this analysis must be the distribution of the variable, the number of declared friends of a class. This variable has between 1 and 5 distinct non-zero values across 16 of the 17 systems. The remaining system has 7 distinct non-zero values and this system rejected the original hypothesis. El Emam *et al.*, [12], have suggested that variables with less than six non-zero values should be excluded from further analysis, since as Briand *et al.*, [5], state, such measures with low variance do not differentiate classes very well. Therefore, with at most 5 distinct points for the number of declared friends of a class, attempting to identify a linear correlation may not be statistically worthwhile.

The other possible refinement of this hypothesis might try to establish if classes which declare friends have more private and protected members than classes which do not declare friends, thus checking to see if a difference exists between two groups as opposed to looking for a linear correlation.

Class Hypothesis 6: Classes that do not engage in any

inheritance have more friends than classes which do engage in inheritance.

This hypothesis is only supported by 5 of the 17 systems. Thus there is no evidence to support this hypothesis in general. In the original study, this hypothesis held for 3 of the 4 systems considered. It uses the same variable as hypothesis 5, (the number of declared friends of a class). The large number of zero values and the low variance of this variable have already been suggested as possible contributory factors in making this variable unsuitable for this type of analysis. A more appropriate approach might split this variable into just two groups, i.e. classes which declare friends and those that do not.

This approach would be closer to that of Counsell and Newson, who compared the proportion of stand-alone classes containing at least one friend with the proportion of inheritance-based classes containing at least one friend. However, when we compared the population of stand alone classes that had friends to those classes engaged in inheritance that had friends, there was only a significance difference in 5 of the 17 systems. Thus relaxing the hypothesis in this way did not result in any change in the outcome of the analysis.

Class Hypothesis 7: Classes declared as friends are bigger than other system classes

We have evaluated this hypothesis using 2 measures of size, Lines of Code, (LOC), and Weighted Methods per Class, (WMC), [7]. In this instance the weights assigned to the methods are all set to 1 and thus WMC counts the number of methods per class. The 2 sub-hypotheses which have been formed are outlined and their results are discussed below:

Class Hypothesis 7(a): Classes declared as friends have a higher WMC than other system classes

Class Hypothesis 7(b): Classes declared as friends have a higher LOC than other system classes

Hypotheses 7(a) and 7(b) are both supported by 11 of the 17 systems, with 8 systems supporting both hypotheses. We cannot state that classes declared a friends tend to be bigger than other system classes. However, at least one of these hypotheses holds for 14 of the 17 systems. Two of the remaining 3 systems are the small systems discussed in hypothesis 4.

The results of El Emam *et al.*, [12], may have an impact here. They found that the ability of existing object-oriented metrics to be associated with fault-proneness diminished when controlling for size. An association between classes declared as friends and high coupling has been shown in this paper and in previous work, [15]. We have also shown that for about 66% of systems, classes which are declared as friends are bigger than classes which are not declared as friends. El Emam *et al.*, have highlighted a number of studies which have shown a correlation between coupling and size, e.g. [5, 6].

Considering these findings the relationship between classes declared as friends and coupling may have been overestimated due to the effect of size. However, if classes declared as friends are shown to have high coupling after controlling for size then it may be the case that classes declared as friends would be better predictors of external quality attributes, than existing object-oriented metrics.

4.2.3 Multivariate Analysis Controlling for Size

To investigate if size does have a controlling effect, a multivariate analysis controlling for size has been undertaken. In particular we investigate if classes declared as friends have higher CBO values when we control for size.

Table 5 illustrates the results of this analysis. For completeness the results of a linear regression with one and two independent variables is provided. In this analysis the dependent variable corresponds to CBO. In the bivariate analysis, the independent variable is the boolean variable which determines whether or not a class is declared a friend. When controlling for size, two different size measures are added separately as independent variables to the regression model.

The results presented when controlling for size should be similar to those in table 3 for hypothesis 4. With $\alpha = .05$, 15 systems returned a significant p-value for CBO, (compared with 13 supporting hypothesis 4), indicating that CBO is significantly different for classes which are declared as friends compared with those classes which are not declared as friends. Hypothesis 4 did not hold for either of the two systems which do not show significance here.

When controlling for size, only the systems which show a significant p-value in the bivariate analysis need to be considered. In this case, 11 of the 15 systems which showed significance without controlling for size are still significant when controlling for size with both the WMC and LOC metrics.

The results presented here do not support the confounding effect of size to the same extent as reported in [12]. This suggests that while El Emam *et al.* have concluded that size has a confounding effect on other object-oriented metrics, in attempting to predict fault-prone classes, the results presented here suggest that classes which are declared as friends still tend to have high coupling even when size is taken into account. It may follow from this that classes which are declared as friends may be better predictors of fault-proneness than existing object-oriented metrics.

	Without Size Control	With Size Control	
		WMC	LOC
System	CBO	CBO	CBO
1	<.001	.003	.083
2	<.001	<.001	.005
3	.010	.014	.022
4	.019	.019	.105
5	<.001	<.001	<.001
6	<.001	<.001	<.001
7	<.001	.001	.011
8	<.001	.011	.089
9	.191	.487	.746
10	.040	.424	.862
11	.005	.003	.008
12	.002	.003	.033
13	.469	.269	.218
14	.707	.706	.712
15	.142	.206	.965
16	.285	.750	.763
17	.697	.693	.695

Table 5. Regression Model with and without controlling for size

5 Threats to Validity

This section reports on the threats to the validity of the study reported in this paper. The systems under analysis in this study are all open-source systems and were all downloaded from sourceforge.net. These systems were at least partially developed in C++ and are in the top 100 most downloaded systems from sourceforge.net, all of which were downloaded more than 1 million times. This download statistic is the only quality measure which we have used and can only be used as an indication of the quality of the systems concerned.

It is not possible to generalize the results presented here to commercial systems, since no evidence exists to suggest that commercial systems have similar characteristics to open-source systems. Another variable yet unquantified factor is that these systems may have been developed by many individuals located across a wide geographical area or may have been developed by a single individual. In addition, the experience of the developers involved is not known. These variables may have a confounding effect on the analysis performed here.

For many of the systems studied in this paper, multiple languages have been used in their development. The extent to which C++ is the development language used may impact the results presented here. In some cases C++ may be used for a complete project, whereas in other cases C++ might only be used to build certain components of the system. There is also a huge variation in system size which may impact the results. However, the size of the systems presented in table 1 only refer to the parts of the system developed in C++.

The projects analyzed in this paper may also vary with regard to their maturity level. For example, some projects may have gone through many iterations of changes whereas others may be relatively immature. The systems analyzed come from many and widely different application domains. Systems from different domains may also vary in terms of the different design alternatives applied in system development.

6 Concluding Remarks

In this paper we have performed an empirical study of a large cohort of open-source software systems to investigate the use of the friend construct in C++ software and to establish to what extent the friend relationship is exploited in the development of systems.

We have confirmed previous findings that classes which are declared as friends have higher coupling than other system classes. Classes declared as friends have also been shown to be bigger than classes not declared as friends. However, when controlling for the size variable, classes declared as friends still had higher coupling than other system classes. Thus the confounding effect of size on object-oriented metrics in predicting fault-proneness of classes does not hold in this instance.

We have found no statistically significant results which suggest that friendship is used as an alternative to inheritance or multiple inheritance. However, in a subset of the systems where multiple inheritance is popular, there seems to be little use of friendship and in other systems where friendship is used extensively there seems to be a relatively small amount of multiple inheritance. In general, this is not the case as the analysis has shown. If the friend construct is introduced into the system after the inheritance hierarchy has been constructed, then it may be the case that friendship is used instead of altering the inheritance structure, to facilitate coupling.

The systems analyzed in this study were mainly application based systems. Our analysis showed that operator overloading is used considerably less in these systems compared with its usage in the library-based systems of a previous study. The relationship between the number of private and protected members in a class and the number of friends declared in the class was supported to a much lesser extent in this analysis although overall it has been supported by 13 out of 21 systems.

Our results also show that friend classes seem to be used more frequently than friend functions. Given Stroustrup's

definition of friend classes, one might expect that their usage suggests that all methods in the class utilize the friend relationship. This is one example of a widespread phenomena in C++ whereby existing software metrics cannot establish the level of actual usage of the friend class mechanism.

In general the C++ language provides a huge number of structuring alternatives and simple measures do not highlight the interplay between design possibilities. Therefore more sophisticated metrics must be defined for exploring the tradeoffs between different structuring alternatives. Analysis of systems using these metrics should illustrate more accurately, ideally in an automated fashion, in particular the extent to which friendship is responsible for coupling between classes and in general for exploring the features of design decisions expressed in product structure. Our future work direction is to define metrics for exploring tradeoffs in the types of structuring alternatives used in software products.

References

[1] F. Abreu, M. G. ao, and R. Esteves. Towards the Design Quality Evaluation of Object-Oriented Software Systems. In *Fifth International Conference on Software Quality*, Austin, Texas, USA, October 1995.

[2] J. M. Bieman and J. X. Zhao. Reuse Through Inheritance: a Quantitative Study of C++ Software. In *SSR '95: Proceedings of the 1995 Symposium on Software reusability*, pages 47–52, New York, NY, USA, 1995. ACM Press.

[3] G. Booch. *Object Oriented Design with Applications*. The Benjamin/Cummings Publishing Company Inc., 1991.

[4] L. Briand and J. Wüst. Empirical Studies of Quality Models in Object-Oriented Systems. *Advances in Computers*, 59:97–166, 2002.

[5] 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.

[6] M. Cartwright and M. Shepperd. An Empirical Investigation of an Object-Oriented Software System. *IEEE Transactions on Software Engineering*, 2000.

[7] S. R. Chidamber and C. F. Kemerer. A Metrics Suite for Object Oriented Design. *IEEE Trans. Software Eng.*, 20(6):476–493, 1994.

[8] J. O. Coplien. *"Advanced C++ Programming Styles and Idioms*. Addison-Wesley Longman Publishing Co., Inc., 1992.

[9] S. Counsell and P. Newson. Use of Friends in C++ Software: An Empirical Investigation. *Journal of Systems and Software*, 53(1):15–21, 2000.

[10] S. Counsell, P. Newson, and E. Mendes. Architectural Level Hypothesis Testing through Reverse Engineering of Object-oriented Software. In *Proceedings of the 8th International Workshop on Program Comprehension (IWPC 2000)*, pages 60–66, Limerick, Ireland, 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] M. A. Ellis and B. Stroustrup. *The Annotated C++ Reference Manual*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[14] M. English, J. Buckley, and T. Cahill. Applying Meyer's Taxonomy to Object-Oriented Systems. In *Third IEEE International Workshop on Source Code Analysis and Manipulation*, pages 35–44, September 2003.

[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*, May 2005.

[16] 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.

[17] S. Meyers. *Effective C++*. Addision-Wesley, 1998.

[18] M. Page-Jones. *What Every Programmer Should Know About Object-Oriented Design*. Dorset House Publishing, 1995.

[19] B. Stroustrup. *The Design and Evolution of C++*. ACM Press/Addison-Wesley Publishing Co., 1994.