

ULRR

An empirical study of the use of friends in C++ software

Item Type	Meetings and Proceedings
Authors	English, Michael;Buckley, Jim;Cahill, Tony
Citation	Proceedings of the 13th international workshop on program comprehension (IWPC'05); pp 329-333
Publisher	IEEE Computer Society
Download date	2026-06-09 09:30:17
Item License	https://creativecommons.org/licenses/by-nc-sa/1.0/
Link to Item	https://hdl.handle.net/10344/1188

An Empirical Study of the Use of Friends in C++ Software

Michael English,¹ Jim Buckley,¹ Tony Cahill,¹ and Kristian Lynch²

¹Department of Computer Science and Information Systems, University of Limerick, Ireland.

²Dept of Endocrinology, Lund University, Sweden.

{Michael.English, Jim.Buckley, Anthony.Cahill}@ul.ie

Kristian.Lynch@endo.mas.lu.se

Abstract

A commonly held belief is that the friend construct in C++ is a violation of encapsulation. However, little empirical analysis of its use has taken place to provide evidence to support this claim.

This paper presents a study which assesses the design implications of including friendship in a system. A number of hypotheses are investigated based on previous work in this area by Counsell and Newson, [4]. Our initial findings suggest that classes declared as friends are coupling hotspots, that the more friends a class has the more protected and private members it will contain and that friendship is not used to access inherited protected members.

1 Introduction

Ellis and Stroustrup, [6], 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 declared 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 declared a friend of *A* without *A* being declared a friend of *B*.

Different perspectives on the appropriateness of the friend mechanism exist in the literature. Coplien, [3], claims that the encapsulation provided by the protected and private mechanisms is violated by friendship. Similar perspectives can be found in [11], [9] and [7].

In contrast Stroustrup, [12], defends the friendship mechanism, describing it as “one protection domain granting a read-write capability to another”. He argues that the declaration of a friend is part of the class declaration and rejects the argument that friendship violates encapsulation as “a combination of ignorance and confusion with non-C++

terminology”. Meyers, [10], also holds the view that friends can be considered part of the class’s interface.

Limited empirical evidence about the usage of the friend construct in the development of object-oriented software systems or its impact on other language features exists. One exception is the study undertaken by Counsell and Newson, [4], who examined a number of hypotheses in relation to the use of the friend mechanism with respect to other internal attributes and suggests that friendship is used as an alternative to inheritance. However, this study did not take a number of syntactic restrictions inherent in C++ into account in evaluating the hypotheses. In addition, Counsell and Newson disregarded the asymmetry of the friendship mechanism in their analysis of coupling. These were important factors in the formulation and evaluation of their hypotheses as explained at the start of section 2. The remainder of this paper discusses several refinements to the Counsell and Newson experiment.

2 Empirical Study

The hypotheses which are outlined below are based on the hypotheses presented by Counsell and Newson, [4]. The differences between our hypotheses and those previously presented, and the measures which we use in order to test the hypotheses are discussed here:

1: The use of the friend construct may be necessary in C++ to facilitate operator overloading. If commutativity of objects is required an operator function may have to be defined as a non-member function. If, in addition to this the function needs direct access to the private or protected members in a class, then the operator function must be made a friend of the class. Such friendship should be excluded when counting the declared friends of a class.

2: In addition we believe that due to the asymmetry of friendship, the use of the friendship mechanism will only impact the coupling for a class which is declared as a friend. Briand *et al.*, [1], have looked at coupling issues and specifically have defined metrics based on friendship, taking into

account the one-way nature of the friend mechanism, although there is some variation in the literature as to the definition of coupling, [2], [7].

The data required for this analysis has been extracted from four software systems with the aid of the source code analysis tool, Understand for C++, [8]. The definition of the measures used in evaluating the hypotheses are presented below.

2.1 Hypotheses Definition and Discussion

The following are the hypotheses under investigation:

H1: Classes declared as friends of other classes, have less non-friend coupling than classes which are not declared as friends.

Since friendship is considered a form of coupling, [4], [1], and since a class declared as a friend gains direct access to extra functionality, it might be expected that such a class needs other forms of coupling to a lesser extent and thus will have less non-friend coupling. Our focus is to compare the coupling of classes which are declared as friends and the coupling of classes which are not declared as friends, whereas Counsell and Newson argued that the more declared friends of a class, the less non-friend coupling of that class.

The CBO measure which is used here does not include coupling due to inheritance, nor does it include coupling due to friend classes or functions. This measure is also a one-way measure, i.e. if a class c uses methods or attributes of a class d , then this coupling counts towards the coupling of class c but not class d .

In contrast, Counsell and Newson, [4], base their measure of coupling on the Number of Associations, (NAS).

H2: The more declared friends of a class, the more private and protected members in that class.

This hypothesis differs from H2 of Counsell in that a count is taken of only the private and protected members and not all members of a class, since any class, already has access to the public members of another class. If the class is also a subclass in an inheritance hierarchy, then this class will also have direct access to all protected members inherited by A from its ancestors. These members are also considered in this hypothesis.

H3: Classes declared as friends of other classes, have less inheritance than classes which are not declared as friends.

This hypothesis is the same as hypothesis H3 of Counsell but we measure the amount of inheritance by the DIT metric. A class lower in the inheritance hierarchy is likely to have inherited more members and will also have used the inheritance mechanism more than a class close to the base of the hierarchy. If this hypothesis is accepted, then it may

indicate that friendship is used as an alternative to multiple inheritance for classes declared as friends and which are contained in inheritance hierarchies, but are not base classes.

H4: Classes which declare friends and which engage in inheritance appear lower in the inheritance hierarchy than other classes engaged in inheritance.

The lower a class is in the inheritance tree, the more functionality this class will have inherited and thus the more favourable it might be to declare a friend of the class, since the friend, in turn, will have access to more functionality.

Counsell and Newson formed a different hypothesis based on this rationale, stating that classes declaring friends that engage in inheritance have less descendants than other classes engaged in inheritance, (measured using the Number of Descendants, (NOD)).

We believe that the Depth of Inheritance Tree, (DIT), is a more accurate measure of both the amount of inheritance in a class and the extra functionality obtained through inheritance.

H5: Classes that do not engage in any inheritance have more friends than classes which do engage in inheritance.

This hypothesis is the same as that introduced by Counsell and Newson. The rationale behind the hypothesis is based on the fact that “classes not engaging in inheritance tend to use friends as an alternative to inheritance”. If this hypothesis is accepted, then this would indicate that friendship is used as an alternative to inheritance.

2.2 Overview of the Software Systems

Four software systems were analysed as part of this empirical study. These systems came from various application domains. Two of the systems under study, LEDA and Libg++, were also part of the analysis undertaken by Counsell and Newson, [4]. However, based on the summary statistics available, different versions of these systems were examined here. The two other systems were Darwin2k and Bayonne. Table 1 contains summary metrics for each of the four systems. The usage of the friend construct varies greatly among the four systems, (see table 1). The overall high usage of friendship to facilitate operator overloading in 3 of the systems illustrates the influence that this specific use of the friend mechanism could have on the evaluation of the hypotheses if it were included in the statistical analysis.

3 Statistical Analysis and Interpretation

The Spearman correlation coefficient was used for H2, assessing the linear association between two variables. For the other hypotheses we used the Pearson Chi-Square Test of Association to determine whether or not there was an association between two groups.

	Leda	Libg	Darw	Bayo
Version	4.1	2.6.2	0.90	1.2.13
Classes	521	150	319	174
Stand-alone Classes	198	51	74	12
Inheritance Trees	53	7	6	5
Avg No. Cl. per tree	6	14	41	32
Maximum DIT	5	5	6	3
Friends	863	296	129	138
Friend Classes	215	35	71	107
Friend Functions	648	261	58	31
Friend Op Overloads	298	146	27	0
Cl. Declared Friends	137	30	34	66
Cl. Declaring Friends	152	25	31	54

Table 1. Summary Metrics for the Four Systems

	H1	H3	H4	H5
LEDA	< .001	.002	< .001	.316
libg++	< .001	.033	.034	< .001
Darwin2k	.038	< .001	.001	< .001
Bayonne	< .001	.001	.477	.028

Table 2. P-values for the Chi-Square Statistic for Four Hypotheses

3.1 H1: Classes Declared as Friends have less Coupling

For three of the systems the p-value is < 0.001 for this hypothesis, (see column 1, table 2), and therefore we can conclude that the two populations under study are different for these three systems.

The Darwin2k system returned a p-value of .038. However, a pronounced difference between the distributions occurs in the lower and upper quartiles, with little difference in between. The trend seems the same as in the other three systems but not as pronounced. Given that the Chi-Square test does not report a significantly small p-value this needs further investigation, but it may be because the proportion of the classes declared as friends in the system is very small, (see table 1).

Our initial hypothesis was based on the premise that friendship might be used as an alternative to other forms of coupling and thus in classes which have a lower coupling value. However the opposite seems to be the case: classes which are declared friends have a high degree of non-friend coupling. This seems to indicate that classes which are declared friends are coupling hotspots in software systems.

		Leda	Libg	Darw	Bayo
H2	p	< .001	.197	.004	< .001
	Spearman	.191	.106	.161	.468
H2.1	p	< .001	.005	.252	.084
	Spearman	.297	-.283	-.073	-.136

Table 3. Pearson Statistics for H2 and H2.1

3.2 H2: The More Friends, the More Private and Protected Members

Table 3 shows the statistical results for hypothesis H2. We tried to determine if a linear correlation existed between the number of friends in a class and the number of private and protected members in the class. For three of the four systems a linear correlation was reported but this correlation was weak. Such a correlation was not uncovered by Counsell and Newson as we have only included those methods for which friendship is necessary to provide direct access.

3.3 H3: Classes declared as Friends have less Inheritance

The results for hypothesis H3, (table 2), indicate that the distributions of the DIT metric for classes which are declared as friends and for classes not declared as friends are significantly different for two of the four systems. One system supports the hypothesis, while the opposite seems to be the case for the other. The distributions or at least parts of them may also be different for the LEDA system.

We cannot determine any relationship between DIT and classes which are declared as friends. This may be because we are examining both inheritance based classes and stand-alone classes together and, given that all stand-alone classes have a DIT of zero this may be skewing the results. A refinement of this hypothesis might examine stand-alone and inheritance-based classes separately in relation to this hypothesis. It is also possible, in line with Counsell and Newson's findings, that the declaration of a class as a friend is independent of its placement in the inheritance hierarchy.

3.4 H4: Classes which declare friends have a higher DIT

The p-values presented in table 2 indicate that the distributions differ for the LEDA and Darwin2k systems. However, the opposite of the hypothesis seems true for these systems. Therefore, we reject this hypothesis.

To investigate the relationship between the usage of friendship and inheritance further we have looked at a variation of hypothesis H2, since we believe that it could cast further light on the results reported here. Therefore we have

carried out a variation on H2, (henceforth referred to as H2.1). Here, instead of measuring the number of private and protected members in total, we have only measured the inherited protected members. If a linear correlation does not exist between the number of friends in a class and the number of inherited protected members of the class, then this is further evidence that classes are not declared friends of other classes for the purpose of accessing this inherited functionality.

For H2.1, as table 3 indicates, the p-values suggest that a correlation exists for only two systems, but the Spearman value for these systems is positive for one and negative for the other. Therefore, we can conclude that there is no evidence to support a correlation between inherited protected members in a class and the number of friends that class has.

3.5 H5: Stand-alone classes have more friends than Inheritance-based classes

Table 1 shows the statistical results for hypothesis H5. There exists a difference between the distributions for stand-alone classes and inheritance based classes for Libg++, Darwin2k. An examination of the distributions provides support for the hypothesis above. There is no evidence to support this claim in LEDA. For the Bayonne system, the p-value indicates that the distributions may well be the same, or at least that any differences in them may be down to chance. However an examination of the distributions leads us to conclude that at least parts of the distributions are different, lending some support to the above hypothesis.

Given that there is a significant difference in the number of friends in stand-alone classes and inheritance based classes for 3 of the systems, the hypothesised relationship may well exist. It may also indicate that friends are used as an alternative to inheritance, or to multiple inheritance, if the classes declared as friends of the singletons already have a parent class. Further analysis is needed to investigate these issues. We cannot support this hypothesis conclusively since the hypothesis doesn't hold for LEDA.

4 Conclusion

In this paper an empirical study of four software systems was carried out to investigate the use of the friend mechanism in C++ software and to try to associate its usage with other program constructs. We have established that classes which are declared as friends have considerably higher coupling than classes not declared as friends. This relationship has heretofore not been established. This finding indicates that classes declared as friends could highlight coupling hotspots in systems.

We have also established a possible link between the number of friends in a class and the number of private and protected members in the class, illustrating a very practical usage of friendship, to access restricted methods and data, which otherwise would not be accessible directly. We have demonstrated these findings through a refined measurement process which ignores friend functions which are overloaded operators.

From this analysis it seems clear that a much more sophisticated set of metrics, than already exists is required for the analysis of software systems. These metrics need to be based on the rationality of expert programmers. Ejiogu, [5], has stated that effective software metrics should be "simple and computable" and "programming language independent". While such characteristics should be attributable to metrics from a practical perspective they are not applicable to sophisticated metrics which are required to analyse architectural tradeoffs in software development. These metrics must be program language dependent and be based on the rationale behind the use of constructs by appropriately skilled users.

A fuller version of this paper is available from:
www.csis.ul.ie/research/techrpts/ul-csis-05-1.ps

References

- [1] 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.
- [2] S. R. Chidamber and C. F. Kemerer. A Metrics Suite for Object Oriented Design. *IEEE Trans. Software Eng.*, 20(6):476–493, 1994.
- [3] J. O. Coplien. *Advanced C++ Programming Styles and Idioms*. Addison-Wesley Longman Publishing Co., Inc., 1992.
- [4] S. Counsell and P. Newson. Use of friends in C++ software: an empirical investigation. *Journal of Systems and Software*, 53(1):15–21, 2000.
- [5] L. Ejiogu. *Software Engineering with Formal Metrics*. QED Publishing, 1991.
- [6] M. A. Ellis and B. Stroustrup. *The Annotated C++ Reference Manual*. Addison-Wesley Longman Publishing Co., Inc., 1990.
- [7] 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.
- [8] S. T. Inc. Understand for C++. web: www.scitools.com.
- [9] M. Lorenz and J. Kidd. *Object-Oriented Software Metrics*. Prentice-Hall, Englewood Cliffs, N.J., 1994.
- [10] S. Meyers. *Effective C++*. Addison-Wesley, 1998.
- [11] M. Page-Jones. *What Every Programmer Should Know About Object-Oriented Design*. Dorset House Publishing, 1995.
- [12] B. Stroustrup. *The Design and Evolution of C++*. ACM Press/Addison-Wesley Publishing Co., 1994.