Object-Oriented Software Understandability: An Empirical Investigation

R. Harrison
Department of Electronics and Computer Science, Mountbatten Building, University of Southampton, Southampton, SO17 1BJ, U.K.
Tel. +44 (0) 1703 593249
email: rh@ecs.soton.ac.uk

S. Counsell
Department of Computer Science
Birkbeck College, University of London, Malet Street, London, WC1E 7HX. U.K.
Tel. +44 (0) 171 631 6723.
email: steve@dcs.bbk.ac.uk

Abstract

In this paper, the results of an empirical investigation into the relationship between software understandability and features of object-oriented software are described. Five C++ Systems of varying sizes were analysed and data relating to understandability, inheritance, coupling and other class features collected for each. A number of hypotheses were then investigated to identify relationships between understandability and properties of each system. The hypotheses asked whether the extent of class coupling in a system, inheritance in a system, or class size features (methods and attributes) affected the understandability of software.

Significant relationships between understandability and class coupling were found for three of the systems studied indicating that as coupling increases, understandability decreases. Only limited evidence of a relationship between understandability and inheritance was found for the five systems studied, suggesting that inheritance plays a minor role in the assessment of understandability. Significant positive relationships were found between understandability and class size measures, indicating that the number of methods and attributes in a class strongly influences understandability.

The results in this paper highlight the need for more empirical investigations in this area, particularly in the understanding of class cohesion, and the role of inheritance and coupling in the assessment of understandability.

1 Introduction

A commonly-held view in software engineering is that there is a link between complexity in software and the understandability of that software. The more coupling in a system, the more complex the system. One form of complexity is due to coupling between classes (where methods or attributes of one class access methods or attributes of another class or one class inherits from another). Too much coupling is indicative of a poorly thought out design and there is evidence to suggest that it can lead to more fault-prone software [6, 7, 14].

In this paper, the results of an empirical investigation into the relationship between
software understandability and features of object-oriented software are described. Five C++ Systems of varying sizes were analysed and data relating to understandability, inheritance, coupling and other class features collected for each. Five hypotheses were investigated to identify relationships between understandability and properties of each system. These hypotheses ask whether the extent of class coupling in a system, inheritance in a system or class size features (methods and attributes) affect the understandability of software.

Significant relationships between understandability and class coupling were found for three of the systems studied indicating that as coupling increases, understandability decreases. Coupling also seems to be overlooked when it occurs below a certain level (only classes with large coupling levels were deemed more difficult to understand). Only limited evidence of a relationship between understandability and class inheritance was found for any of the five systems studied, suggesting that inheritance plays a minor role in the assessment of understandability. Significant positive relationships were found between understandability and class size measures indicating that the number of methods and attributes in a class strongly influences understandability.

The paper is structured as follows: in Section 2, we provide details of the empirical evaluation, giving details of the metrics collected and each of the application domains. Data analysis, giving correlation analyses, is given in Section 3. Some conclusions and further work are described in Section 4.

2 Empirical Evaluation

Empirical evaluation can be used to investigate the association between proposed software metrics and other indicators of software quality such as understandability; thorough qualitative or quantitative analyses can be used to support these investigations [5, 2, 19, 1, 14, 12, 13]. This paper addresses the relationship between understandability and OO features through the investigation of a number of hypotheses.

Hypotheses

Five hypotheses related to understandability are investigated in this paper, and they have been structured into their respective areas. The class coupling hypothesis (H1) refers to any form of coupling (including inheritance), whilst the inheritance hypotheses refer to only coupling due to inheritance.

Class coupling

1. **H1**: A class with significantly more coupling than other classes is harder to understand.

Inheritance coupling

1. **H2**: A class with significantly more inherited methods than other classes is harder to understand. This hypothesis appeals to the belief that understanding the functionality of a class which inherits a large number of methods is more difficult than if it inherited few or no methods.

2. **H3**: A class with significantly more overridden methods than other classes is harder to understand. A class with a large number of overridden methods may indicate a poor design with unnecessary redefinition of methods. It may also be necessary for a developer or maintainer to understand the functionality of the superclass (overridden) methods before they can understand the methods...
being overridden; assessing the understandability of a class with a large number of overridden methods may therefore require reference to the superclass also.

Class Size

1. **H4**: A class with significantly more methods than other classes is harder to understand.

2. **H5**: A class with significantly more attributes than other classes is harder to understand.

In the next section, we describe the data collected.

Data Collection

1. **SU**: Software Understanding [3], which ranks software according to structure, application clarity and self-descriptiveness. Software understanding is rated on an ordinal scale of 1 to 5 (where 1 represents the simplest and easiest to understand class, and 5 the most complex and difficult to understand). The SU measure is based on ratings of individual classes, and was provided by the data collectors (two of the authors and a research assistant). Although subjective in nature, Boehm’s SU metric represents an easily-collectible, consistent and useful reflection of class complexity.

The following coupling metrics were collected:

1. **Coupling Between Objects (CBO)**. This is a count of the number of classes to which a class is coupled. Two classes are coupled when one uses methods or variables defined in the other. This includes coupling via inheritance [10].

2. The **Number of Associations per class (NAS)**. This metric is defined as the number of associations of each class, counted by the number of association lines emanating from a class on an Object Modelling Technique (OMT) [18] diagram. This measure also includes coupling via inheritance and aggregation.

We note that the CBO and NAS metrics are identical, except in the following case: if a class uses a method of another class more than once, then the CBO metric will count each usage as a separate occurrence of coupling. NAS, on the other hand counts repeated invocations as a single occurrence of coupling. Hence, we expect values of CBO for a class to be greater than the NAS for the same class.

The following inheritance metrics were collected:

1. **Number of Methods Inherited per Class (NMI)**. The total number of methods which can be potentially inherited by a class from all its superclasses.

2. **Number of Methods Overridden per Class (NMO)**. This is the number of virtual base class methods overridden by a class method. An overridden method has the same signature as the base class method but the semantics of the parent class method are changed.

The following size metrics were collected:
1. Number of methods per class (NMC). The number of public, private and protected methods declared in a class. This does not include inherited methods or friend functions declared in the class.

2. Number of attributes per class (NAC). The number of data values held by objects in each class.

In the next section, the application domains of the five systems investigated are described.

**Application Domains**

Our empirical analysis consisted of five systems. These were:

1. System One, the Gnu C++ Class Library, consisting of 53.5 KLOC (96 classes).

2. System Two, a Library of Efficient Data Algorithms (LEDA), written in C++ and consisting of about 123 KLOC (197 classes), designed and developed at the Max Planck Institute in Saarbruecken, Germany.

3. System Three, SEG1, consisting of eleven medium-sized traffic simulation systems written in C++. The average size of a system was 2.5 KNCSL, and a total of 113 classes were analysed for the ten systems. Each of the ten systems was developed in a MSDOS (Version 6.0) programming environment.

4. System Four, SEG2, consisting of twelve medium-sized traffic simulation systems written in C++. The average size of a system was 3.5 KNCSL, and a total of 172 classes were analysed. Each of the eleven systems was developed in an MSDOS (Version 6.0) programming environment.

5. System Five, SEG3, consisting of twenty-one medium-sized traffic simulation systems written in C++. The average size of a system was 4.5 KNCSL, and a total of 315 classes were analysed. Each of the eleven systems was developed in Windows'95 programming environment.

In the next section, the data analysis performed is described.

3. **Data Analysis**

**Summary of the systems**

Table 1 shows the summary data for each of the five systems investigated. All of the five systems contained inheritance. SEG 1 (System 3) contained relatively little inheritance (22% of classes were involved in inheritance) while SEG 2 (System 4) did contain inheritance, but inheritance hierarchies tended to be shallow (12% of classes engaged in inheritance). SEG3 contained only a limited amount of inheritance (3% of classes engaged in inheritance). Classes in SEG1 tended to perform a single function such as a display of a screen layout, or the change of state of a light bulb; this required no inter-class relationships.

Similarly, collections of independent classes were also found in the LEDA and Gnu systems. Key classes tended to contain a large number of methods and attributes, and this is reflected in the maximum values for NMC and NAC. All the systems investigated showed some evidence of coupling due to: class X using class Y as a parameter of one of its methods or class Y being a return data type for a method of class X. In the next section, we begin by analysing hypothesis H1.

**Hypothesis H1 (coupling)**

Table 2 shows a correlation analysis of SU against CBO and NAS. Significant relationships are shown for Systems 3, 4 and 5 (SEG
Table 1: Summary data for all five systems

<table>
<thead>
<tr>
<th>Metric</th>
<th>System One (Gnu)</th>
<th>System Two (LEDa)</th>
<th>System Three (SEG1)</th>
<th>System Four (SEG2)</th>
<th>System Five (SEG3)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CBO</td>
<td>Min</td>
<td>Max</td>
<td>Median</td>
<td>CBO</td>
</tr>
<tr>
<td>NAS</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>NMI</td>
<td>0</td>
<td>126</td>
<td>32</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>NMO</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>NMC</td>
<td>0</td>
<td>18</td>
<td>2</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>NAC</td>
<td>0</td>
<td>108</td>
<td>9</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

systems), but not for any of the other systems. Each of the SEG systems contained only limited amounts of inheritance, which implies that most of the class coupling in each of the three systems was non-inheritance based. The extent of coupling in these three systems, given by the NAS and CBO values can be seen in Table 1.

The significant relationships found for these three systems combined with a detailed investigation of the systems' architectures, suggests that classes with large coupling levels were deemed more difficult to understand by the assessors. Also, it seems there may be an optimum level of coupling; classes in each of the five systems were typically coupled to just one or two other classes (particularly in the case of the three SEG systems). Classes with higher coupling levels tended to be key classes found at the root of a wide inheritance hierarchy; this seemed to cause problems as far as understandability was concerned.
Hypothesis H1 is therefore supported, and we can conclude that the more coupling in a class the more difficult it is to understand that class.

Hypotheses H2 and H3 (inheritance)
Table 3 shows a correlation analysis of SU against NMI and NMO. Significant (negative) relationships are shown against NMI for two systems (SEG1 and SEG2), suggesting that the greater the number of methods inherited, the easier it is to understand a class. This can be explained by the tendency of subclasses to be significantly smaller than their base class in each of the five systems. A developer who understood the functionality of a base class then has less work to do in order to understand its subclasses. No significant relationships were found for NMO. The lack of use of inheritance in the five systems may also be a factor in the lack of correlations for NMI and NMO.

Just as there seemed to be for coupling (hypothesis H1), there may exist an optimum level of inheritance for a class expressed in terms of how many methods it inherits and how many overriding methods it contains. The inheritance hierarchies of the five systems tended to consist of forests of trees having between one and three levels of inheritance depth. An experiment was conducted by Daly et al. [11], in which a system with three levels of inheritance was shown to be easier to modify than a system with no inheritance. A system with five levels of inheritance was, however, shown to take longer to modify than the system without inheritance; this experiment therefore points to an optimum level of inheritance.

However, in Harrison et al. [15], results of an investigation into inheritance indicated that systems without inheritance were easier to modify than corresponding systems containing three or five levels of inheritance. Also, it was easier to understand a system without inheritance than a corresponding version containing three levels of inheritance. Larger systems are equally difficult to understand whether or not they contain inheritance.

Neither hypotheses H2 or H3 is supported, and so we cannot conclude that the more inheritance in a class, as measured by NMI and NMO, the more difficult it is to understand that class. In fact, other inheritance metrics such as the Depth of Inheritance Tree (DIT) [9] have shown that child classes are easiest to understand, as far as these systems are concerned [16].

Hypotheses H4 and H5 (class size metrics)
Table 4 shows a correlation analysis of SU against the class size metrics: NMC and NAC. Significant positive relationships are shown for four systems against NAC (LEDAN: System 2 (LEDA) 0.09 -0.06, System 3 (SEG1) 0.46* 0.52*, System 4 (SEG2) 0.61* 0.66*, System 5 (SEG3) 0.52* 0.63*.

* significant at the 1% level

Table 2: Spearman's rank correlation coefficient for all five Systems vs. SU
Table 3: Spearman’s rank correlation coefficient for all five Systems vs. SU

<table>
<thead>
<tr>
<th>System</th>
<th>NMI</th>
<th>NMO</th>
</tr>
</thead>
<tbody>
<tr>
<td>System 1 (Gnu)</td>
<td>0.13</td>
<td>-0.11</td>
</tr>
<tr>
<td>System 2 (LEDA)</td>
<td>-0.08</td>
<td>-0.02</td>
</tr>
<tr>
<td>System 3 (SEG1)</td>
<td>-0.30*</td>
<td>-0.02</td>
</tr>
<tr>
<td>System 4 (SEG2)</td>
<td>-0.27*</td>
<td>-0.10</td>
</tr>
<tr>
<td>System 5 (SEG3)</td>
<td>-0.13</td>
<td>-0.10</td>
</tr>
</tbody>
</table>

* significant at the 1% level
ni: no inheritance

SEG1, SEG2 and SEG3). Three positive significant relationships were found against NMC (SEG1, SEG2 and SEG3).

Table 4 shows a strong positive relationship between understandability and the number of attributes or methods in a class. As the number of attributes or methods increases, the class becomes more difficult to understand. One would expect, generally speaking, for a class with a large number of attributes or methods to be more difficult to understand than a class with few. Intuitively, understandability is influenced strongly by the number of attributes or methods, and also by the way the attributes are distributed amongst the methods of the class. Further investigation will focus on a more in depth analysis of class cohesion in all five systems.

Both Gnu and LEDA contained relatively large amounts of inheritance and made extensive use of the *friend* programming mechanism as a means of reuse. This may explain the lack of significance against NMC for these two systems.

Both hypotheses H4 and H5 are supported, and so we conclude that the more attributes or methods in a class, the more difficult it is to understand that class.

**Discussion of Hypotheses**
The lack of significant correlations for inheritance values (NMI and NMO) against SU is remarkable considering that three different assessors took part in the allocation of SU values. In addition to the suggestions already given, further explanations for the lack of correlation can be considered. Firstly, when assessing the understandability of a class, unless the class contains substantial amounts of coupling, the assessor seems to consider the class in isolation. Coupling with other classes is considered outside the boundary of what needs to be understood, and any form of coupling between classes seems to be overlooked. Additionally, without sufficient design documentation being available, for example, the system inheritance hierarchy, it would be difficult for the developer or maintainer to assess the level of overall class inheritance from the classes alone.

Our empirical investigations also lead us to believe that class cohesion is a significant factor within the context of understandability, and (in particular) that the distribution and use of attributes within a class is important in many cases. Unfortunately, there are few metrics available to measure cohesion effectively, and existing measures tend to be flawed. For example, the Lack of Cohesion of Methods (LCOM) of Chidamber and Kemerer [9] is intended to measure the lack of cohesion in the methods of a class. It is based on the principle that a variable occurring in
Table 4: Spearmans rank correlation coefficient for all five Systems vs. SU

<table>
<thead>
<tr>
<th>System</th>
<th>NMC</th>
<th>NAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>System 1 (Gnu)</td>
<td>-0.04</td>
<td>0.08</td>
</tr>
<tr>
<td>System 2 (LEDA)</td>
<td>0.08</td>
<td>0.26*</td>
</tr>
<tr>
<td>System 3 (SEG1)</td>
<td>0.74*</td>
<td>0.33*</td>
</tr>
<tr>
<td>System 4 (SEG2)</td>
<td>0.52*</td>
<td>0.40*</td>
</tr>
<tr>
<td>System 5 (SEG3)</td>
<td>0.65*</td>
<td>0.51*</td>
</tr>
</tbody>
</table>

* significant at the 1% level
†significant at the 5% level

many methods of a class causes that class to be less cohesive than one where the same variable is used in few methods of the class. However, apart from in small systems, it is too cumbersome to collect. There is an urgent need for more industrial-strength tools to aid in the collection and analysis of this kind of data.

**Threats to validity**

We consider here threats to the external and internal validity of this study [17]. *External validity* is the degree to which results from the study can be generalised to the population, i.e., other groups. The systems investigated in this paper represent a mix of projects of different sizes and application domains, reducing this threat to external validity. Note that Systems 3, 4 and 5 were developed by undergraduate students, not professional programmers. However, the conditions in which they worked on these group projects are intended to mimic those in the real-world as far as possible; the students (who have at least eighteen months programming experience) are expected to re-engineer medium-sized legacy systems, according to changing requirements. Boehn-Davis [4] and Brooks [8] support the use of students as subjects in certain situations.

*Internal validity* is the degree to which we can conclude that the dependent variable is accounted for by the independent variable. The SU values were provided by three different assessors, and the results of the SU correlations from the five different systems concurred. Finally, the nature of the application domain seems to have a strong bearing on the architecture of the system. A system with a graphical user interface may have different structural properties to that of a system containing library classes; this could be considered a threat to external validity.

**4 Conclusions and Future Research**

In this paper, we have analysed the relationship between understandability and characteristics of OO software. We hypothesised that understandability would be related to coupling and inheritance, as well as class features such as number of methods and number of attributes. Significant relationships between understandability and class coupling were found for three of the systems studied indicating that as coupling increases, understandability decreases. Only limited evidence of a relationship between understandability and inheritance was found for the five systems studied, suggesting that inheritance plays less of a role in the assessment of understandability. Significant positive relationships were found between understandability and class size measures indicating that the number of methods and attributes in a class strongly influences understandability.
There is a need for research to address issues arising from the results in this paper. For example, optimum levels of coupling and inheritance in systems may exist. Each of the systems studied in this paper had different levels of coupling. The main focus of future research will therefore be to carry out further empirical evaluations to establish whether such an optimum level does exist. Further research will also concentrate on an analysis of class cohesion and the interplay between it and coupling. This should provide a useful insight into the part cohesion plays in understanding software. The research in this paper has revealed how large a role the architecture of a system plays in determining coupling etc. Further research will also concentrate on identifying patterns of coupling and cohesion amongst applications of similar and different domains.

Acknowledgments
The authors wish to acknowledge the support of the UK EPSRC, which has funded the MOOPS project (GR/K83021), and also the University of Southampton for funding a Research Studentship to participate in this work.

REFERENCES


