Experimental Assessment of
the Effect of Inheritance on the Maintainability of
Object-Oriented Systems

R. Harrison, S. Counsell, R. Nithi
Department of Electronics and Computer Science,
Mountbatten Building,
University of Southampton, Southampton, SO17 1BJ, U.K.
Tel. +44 (0) 1703 593249, Fax. +44 (0) 1703 593045.
email: rh@ecs.soton.ac.uk

ABSTRACT
In this paper, we describe an empirical investigation into the modifiability and understandability of Object-Oriented (OO) software. A controlled experiment was conducted to establish the effects of varying levels of inheritance on understandability and modifiability.

The software used in this experiment consisted of a C++ system without any inheritance and a corresponding version containing three levels of inheritance, as well as a second larger C++ system without inheritance and a corresponding version with five levels of inheritance. For both of the systems, the application modelled a database for a University personnel system.

A number of statistical hypotheses were tested. Results indicated that the systems without inheritance were easier to modify than the corresponding systems containing three or five levels of inheritance. Also, it was easier to understand the system without inheritance than a corresponding version containing three levels of inheritance. Results also indicated that larger systems are equally difficult to understand whether or not they contain inheritance.

The results contained in this paper highlight the need for further empirical investigations in this area, particularly into the benefits of using inheritance.

Keywords
Empirical software engineering, object-oriented systems, maintainability, inheritance.

1 INTRODUCTION
A key feature of the object-oriented (OO) paradigm is that of inheritance [19, 6, 11, 18, 20]. In OO systems, use of inheritance is claimed to reduce the amount of software maintenance necessary and ease the burden of testing [1, 16, 7, 10]; the re-use of software through inheritance is claimed to produce more maintainable, understandable and reliable software [3, 2, 5].

In this paper, we discuss the results of an experiment which we carried out based on that carried out by Daly et al. [12], which investigated the modifiability of C++ programs with zero, three and five levels of inheritance and suggested that there was an optimum level of inheritance lying between three and five levels. The experiment described in this paper differed in several respects. Firstly, two C++ systems, neither containing inheritance, and which we will refer to as Systems A and B, were used in our experiment, together with their corresponding inheritance-based versions; one with three levels of inheritance and one with five. Secondly, the modifications to the inheritance hierarchy required of the experimental subjects in our experiment were different. Modifications to the two systems containing inheritance in our experiment were required for classes at the lowest (leaf) classes of the respective inheritance hierarchies. Lastly, we also expanded the scope of our experiment to incorporate assessment of the understandability of the system being examined. Hypotheses related to the effects on modifiability and understandability of the required tasks were proposed and then investigated. Results indicate that, firstly, the flat systems (without any inheritance) were easier to modify than the versions containing three or five levels of inheritance. Secondly, it was easier to understand the flat system A than its corresponding version containing three levels of inheritance. Lastly, the system B without inheritance was not found to be easier to understand than its corresponding version containing five levels of inheritance; the understandability of each was found to be the same. System B contained added functionality and therefore was considerably larger than System A. Results therefore indicate that larger systems are equally difficult to understand whether or not they contain inheritance.

Clearly, therefore, there is an urgent need for more empirical research to investigate when object-oriented techniques provide benefits over object-based techniques, and particularly into the effects of using inheritance.
In Section 2, related work in this area is described. The description of the experiment then follows (Section 3), including details of a pilot study conducted, hypotheses and systems investigated. Section 4 describes the results from our experiment, including a description of the threats to validity; Sections 5 and 6 present conclusions and further work respectively.

2 RELATED WORK

Very little formal experimentation has been performed to assess the impact of varying levels of inheritance on the maintainability of OO software.

This paper replicates an experiment conducted by Daly et al. [12], in which subjects were timed performing maintenance tasks on OO systems with different levels of inheritance. In their experiment, 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 seemed to imply that there is an optimum level of inheritance beyond which maintainability becomes problematic.

In Basili, Briand and Melo [3], the results of an empirical study of the Chidamber and Kemerer (C&K) [9, 10] metrics are presented. The metrics are used as predictors of fault-prone classes. Data from eight medium-sized management systems, developed in C++ was collected. Experimental hypotheses suggested that a class located deep in the inheritance hierarchy was more fault-prone than a class higher up in the hierarchy; this hypothesis was found to be supported with statistical significance. This clearly implies that far from aiding maintenance, use of inheritance had the opposite effect.

Other investigations seem to support a growing feeling in the OO community that inheritance should be used sparingly during development [13, 7, 3, 8].

Chidamber and Kemerer describe empirical analyses of systems at two sites, one of which used C++, the other Smalltalk. It was noted that the extent of inheritance at both sites was small (with median Depth of Inheritance Tree (DIT) values of 1 and 3 for the C++ and Smalltalk sites respectively). In Chidamber, Darcy and Kemerer [8], three commercial object-oriented system are empirically investigated, and, again, none showed significant use of inheritance.

Cartwright and Shepperd [7] describe an analysis of a large telecommunications system (133,000 lines of C++). They report relatively little use of inheritance in the system they analysed, and classes with the highest change densities were found to be low down in the inheritance hierarchy (i.e., away from its root). A major motivation for the use of empirical analysis is that is can be used to investigate the association between proposed software metrics and other indicators of software quality such as maintainability; thorough qualitative or quantitative analyses can be used to support these investigations [5, 3, 17]. In the next section, a description of the conducted experiment is given.

3 DESCRIPTION OF THE EXPERIMENT

The experiment described in this paper attempted to determine if the depth of inheritance has an effect on the modifiability and understandability of object-oriented software. A number of terms are used throughout this paper; these we now define.

1. Depth of inheritance (DIT) refers to the level of a class in the hierarchy where the base class is level zero. Any class of level n in the hierarchy has n superclasses. The level of the deepest leaf class is quoted as the depth of the hierarchy. DIT was originally defined by Chidamber and Kemerer (C&K) [10].

2. Maintainability. In this paper, maintainability is sub-divided into modifiability and understandability. Modifiability is defined as the ease with which a change or changes can be made to a program (n.b. a modification may require many changes to be made to the source code). Understandability is measured by Software Understanding (defined next).

3. Software Understanding (SU). ranks software according to structure, application clarity and self-descriptiveness [4]. 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). A system with strong modularity, well documented with a clear match between program and application world-views would be considered easily understandable and given a rating of 1. A system with low cohesion with no match between system and application world-views and containing obscure code would be rated at the other end of the scale (5).

In the next section, we describe the results of a pilot study, conducted to assess the possible effectiveness of a larger study.

Pilot Study

A pilot study using 12 students from the final-year BSc. (Hons.) Computer Science degree at the University of Southampton, UK, was conducted prior to the experiment described in this paper. The results of this experiment are contained in Table 1. For each of the twelve
subjects, the questionnaire required fourteen outputs to be identified, encompassing eleven classes in the inheritance hierarchy (these classes had to be identified by the subject), and required eight changes to be made to the code (as a result of the required modification to requirements); these values are contained in brackets in the table. To achieve hundred percent success, a subject had to successfully identify the outputs, the affected classes, and the changes needed. A value for SU was also provided by the subject for the understandability of the system as a whole. Subjects were given 45 minutes to complete the tasks.

Table 1 indicates that two subjects completed the experiment with full marks; they both rated the systems as easy to understand. Table 1 also indicates that correctly identifying changes to the source code proved most error-prone (only the same two subjects achieved 100% accuracy).

The pilot study provided useful information on the structure of the questionnaire, and eliminated mistakes which might have hampered a further study. In particular, on clarity of the questionnaire wording, and the layout and format of the experimental materials. As a result of the pilot study, a larger, more comprehensive experiment was then undertaken.

In the next section, the hypotheses for the experiment are described.

Hypotheses Investigated
Four hypotheses were investigated in this paper. We describe each of the null hypotheses together with the corresponding alternate hypotheses:

H01. The first null hypothesis is that the use of three levels of inheritance depth does not affect the modifiability of object-oriented systems.

H1. The first alternate hypothesis is that the use of three levels of inheritance depth does affect the modifiability of object-oriented systems.

H02. The second null hypothesis is that the use of five levels of inheritance depth does not affect the modifiability of object-oriented systems.

H2. The second alternate hypothesis is that the use of five levels of inheritance depth does affect the modifiability of object-oriented systems.

H03. The third null hypothesis is that use of three levels of inheritance does not affect the understandability of object-oriented systems.

H3. The third alternate hypothesis is that the use of of three levels of inheritance does affect the understandability of object-oriented systems.

H4. The fourth null hypothesis is that use of five levels of inheritance does not affect the understandability of object-oriented systems.

H4. The fourth alternate hypothesis is that the use of five levels of inheritance does affect the understandability of object-oriented systems.

Note that, in keeping with the experiment conducted by [12], no direction has been specified in the alternate hypotheses. It was not predicted whether the effect on maintainability or understandability would be positive or negative. There was one important difference in the nature of modifications between the experiment described in this paper and that of [12]: modifications to the inheritance structure herein were to classes at the leaf nodes of the inheritance tree; this was not the case in [12].

Systems Investigated
Two systems, each with two versions, were used as a basis of our experiment, defined as follows:

System A:
A system to store and manipulate personal details about lecturers, students and administrative staff in a university.

1. Version A0: a flat version of System A; A0 contained around 360 lines of code (including 25 comment lines).

2. Version A3: the inheritance-based version of System A, containing inheritance to a maximum depth of three. Version A3 contained around 390 line of code (including 35 comment lines).

System B:
An expanded version of System A containing details of additional university administrative staff, room details, student supervisors, etc.

1. Version B0: a second flat system, System B0 contained around 1200 lines of code (including 25 comment lines).

2. Version B5: the inheritance-based version of System B, containing inheritance with a maximum
depth of inheritance of five. Version B5 contained around 900 lines of code (including 35 comment lines).

Subjects
The participants of the experiment were second year Bsc. (Hons.) Computer Science students at the University of Southampton, UK, on a Software Engineering unit, the content of which included a group project implemented using C++. All participants had at least eighteen months experience of developing C++ software. Members of each group (consisting of four students) were initially allocated to groups (at the outset of the unit) to ensure that all groups contained students with different levels of ability and that the average ability levels of all the groups were similar. During the unit, subjects were asked if they would be interested in answering a questionnaire on inheritance within OO software. Altogether 48 students took part in the experiment, and were split into twelve groups of four students.

Experimental Materials
Subjects within a group were provided with paper copies of the questionnaire and source code for one of the four versions described in Section 3.3 (see the Appendix for details of each of the questionnaires). Hence, in every group, one student was given version A0 and a corresponding questionnaire, one given version A3 and corresponding questionnaire etc. for the four systems investigated. The allocation of system to student within a group was random. Each of the subjects was given forty-five minutes to complete their respective questionnaires. The questionnaires were completed by group members in their regular weekly meetings and each group of four was overseen by a project group leader.

### Experimental Design
To test the hypotheses, a $4 \times 12$ between subjects design in four blocks of twelve was employed. The independent variable was the type of system (zero, three or five levels of inheritance), the dependent variables being modifiability and SU. Since each of the students in each of the groups carried out only tasks on a single version, problems associated with learning effects were avoided. All data related to each of the four systems was collected. Four samples, henceforward known as: S1, S2, S3 and S4 can be identified, each containing twelve students. The allocation of students to systems was made on the following basis:

- Sample S1 were asked to complete a questionnaire on version A0.
- Sample S2 were asked to complete a questionnaire on version A3.
- Sample S3 were asked to complete a questionnaire on version B0.
- Sample S4 were asked to complete a questionnaire on version B5.

After completing the questionnaire, the twelve responses for version A0 were pooled, and, similarly, those for A3, B0 and B5. Since the allocation of system to student within the original groups was random, it can be assumed that the four samples of twelve allocated to versions A0 to B5 are similarly random, and represent a mixture of ability and experience.

### Analysis Procedure
Chi-square testing was performed to analyse the data collected. The data collected for version A0 was com-

<table>
<thead>
<tr>
<th>Subject</th>
<th>Outputs Identified</th>
<th>Classes Identified</th>
<th>Changes Identified</th>
<th>SU (1-5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>100</td>
<td>88</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>86</td>
<td>82</td>
<td>25</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>100</td>
<td>25</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>86</td>
<td>100</td>
<td>50</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>100</td>
<td>38</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>86</td>
<td>100</td>
<td>50</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td>100</td>
<td>63</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>100</td>
<td>75</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>100</td>
<td>100</td>
<td>13</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>100</td>
<td>100</td>
<td>50</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: Pilot Study results
pared with that for version A3. Similarly, data collected for version B0 was compared with that for version B5.

4 EXPERIMENTAL RESULTS

Tables 2 to 5 summarise the data collected from the experiment for versions A0 to B5 respectively.

For version A0, 10 outputs were expected from the experimental subject. Only three subjects failed to achieve 100% accuracy (see Table 2). For version A3, whilst the majority of subjects were able to identify the correct outputs and affected classes, making the required changes proved more difficult: only five subjects were able to achieve 100% accuracy (see Table 3).

In terms of SU, although both Tables 2 and 3 have identical medians, version A3 was consistently given a higher SU value by subjects assigned to version A0. This implies that subjects found it more difficult to understand A3 than A0. This is reflected in the mean SU values for the two versions (1.75 for A0 compared with 2.67 for A3).

For version B0, five of the subjects achieved 100% accuracy, with the remaining seven subjects achieving 100% accuracy in at least one of the questions (see Table 4). For version B5, whilst the majority of subjects were able to identify the affected classes, identifying the outputs and making the required changes proved more difficult: only four subjects were able to achieve 100% accuracy in the former and only two subjects in the latter (see Table 5).

In terms of SU, version B0 (Table 4) has a median value of two, version B5 (Table 5) has a median SU of three. Examination of the data confirms that version B5 was consistently assessed as being more difficult to understand than version B0.

Chi-square Analysis

We interpret the Chi-square analysis in terms of understandability and modifiability using two comparisons: A0 versus A3, and B0 versus B5. We then re-visit the hypotheses stated earlier.

Chi-square values for subjective understanding for A0 (flat version) versus A3 (three levels of inheritance) were found to be positively statistically significant at the 5% level. This indicates that version A0 was easier to understand than version A3. Similarly, chi-square values for modifications to version A0 versus version A3 were found to be positively significant at the 10% level. It was easier to modify version A0 than version A3.

Chi-square values for subjective understanding for version B0 (flat version) versus version B5 (five levels of inheritance) were not found to be positively statistically significant. Both versions B0 and B5 were therefore found to be equally difficult to understand.

Chi-square values for modifications to version B0 (flat version) versus version B5 (5 levels of inheritance) were found to be positively significant at the 10% level.

Overall, our results indicate that versions A0 and B0 were easier to modify than their inheritance-based counterparts (versions A3 and B5). In terms of understandability, results indicate that large systems with and without inheritance are equally difficult to understand. In the case of the experiment described here, the extra size can be attributed to added functionality.

In terms of the hypotheses stated earlier, we accept hypotheses H1, H2 and H3 (rejecting null hypotheses H01, H02 and H03). Since no evidence was found to support H4, we accept the null hypothesis H4 in that case (the use of five levels of inheritance does not affect the understandability of object-oriented systems). In the next section, we consider the threats to the validity of our experiment.

Threats To Validity

A major concern with an experiment of the type described in this paper are potential threats to the validity of the experiment. We consider both threats to internal and external validity. Two threats to the internal validity of our experiment can be identified.

1. Selection effects due to variations in subject performance. The possibility that more experienced and able students were assigned to one particular experiment. This threat was avoided by assigning on a random basis to each of the experimental pools.

2. Instrumentation effects due to differences in the experimental materials employed. The modification required for version A0 was equivalent to that for version A3, and the modification required for version B0 was equivalent to that for version B5, hence avoiding this threat.

Two external threats to our experiment can be identified:

1. The experimental subjects were students, not professional programmers. However, the conditions in which the experiment took place were 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 maintain and modify medium-sized systems, according to changing requirements, as well as being able to develop medium-sized software from specification.

2. The systems used in this experiment were not large. However, the levels of inheritance in the systems investigated are typical of those found in larger systems.
## 5 CONCLUSIONS

In this paper, the results of an experiment to assess the modifiability and understandability of OO systems have been described. The experiment was similar to one conducted by [12], in which it was suggested that an optimum level of inheritance existed.

Results from our experiment suggest that systems without inheritance are easier to modify than systems with either three or five levels of inheritance. This would seem to cast doubt on the use of inheritance.

Results contained in this paper also indicate that whilst modifying a flat system is easier than modifying one with five inheritance levels, this does not imply that the system without inheritance is easier to understand. Our analysis indicates that the size and functionality of a system has a greater impact on its understandability than the amount of inheritance used.

There is clearly, therefore, a need for research to address many urgent issues arising from the use of inheritance and its effect on the process of maintenance, taking system size and application domain into consideration. In the next section, we discuss some future work.

## 6 FUTURE RESEARCH

Results from this and other studies seem to indicate that the DIT metric provides only a limited view of the inheritance hierarchy of a system. As a result of this limitation, the MOOPS project [7, 13, 14, 15] introduced an alternative breadth-wise view of the inheritance hierarchy; in the next section we discuss this metric.

The Breadth of the Inheritance Tree (BIT) metric was introduced to give a system-wide view of an inheritance
Table 4: Version B0 (Sample 3): twelve subjects, no inheritance

<table>
<thead>
<tr>
<th>Subject</th>
<th>Outputs Identified</th>
<th>Classes Identified</th>
<th>Changes Identified</th>
<th>SU (1-5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
<td>100</td>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>100</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>100</td>
<td>0</td>
<td>70</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>90</td>
<td>100</td>
<td>80</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>100</td>
<td>80</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>100</td>
<td>0</td>
<td>70</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 5: Version B5 (Sample 4): twelve subjects, five levels of inheritance

<table>
<thead>
<tr>
<th>Subject</th>
<th>Outputs Identified</th>
<th>Classes Identified</th>
<th>Changes Identified</th>
<th>SU (1-5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>29</td>
<td>43</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>100</td>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>86</td>
<td>100</td>
<td>90</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>100</td>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>71</td>
<td>100</td>
<td>80</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>80</td>
<td>100</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>100</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>80</td>
<td>100</td>
<td>50</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>80</td>
<td>80</td>
<td>60</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>80</td>
<td>100</td>
<td>60</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>3</td>
</tr>
</tbody>
</table>

The BIT metric is a direct measure of the number of classes at a specific level in the inheritance tree. Adding an extra class to an inheritance tree always increases the breadth of the tree at a particular level by 1. The BIT is expressed in the form \(x_0, x_1, \ldots, x_{n-1}, x_n\) where \(n\) is the maximum depth of the inheritance tree, \(x_0\) is the number of classes at level 0 (the root), \(x_1\) is the number of classes at level 1, \(x_{n-1}\) is the number of classes at level \(n-1\), and \(x_n\) is the number of classes at level \(n\). Parentheses are used to represent systems and sub-systems in the inheritance hierarchy.

In terms of other future research, fundamental issues such as whether inheritance can make maintenance of OO systems easier, whether there is an optimum level of inheritance, and whether we should focus on alternative features as a means of reducing the maintenance burden all need be addressed.

Ideally, this empirical research should be carried out on as many industrial-sized systems as possible, (with subjects of varying experience), supported by well-designed hypotheses. Industrial-strength tools need to be provided to aid the speedy collection of data and dissemination of results. To encourage replication of the experiment contained in this paper by other researchers, experimental materials used are publicly available at:

http://www.ecs.soton.ac.uk/rvn95r/

7 ACKNOWLEDGEMENTS

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.

8 APPENDIX: QUESTIONNAIRES

Questionnaire 1

Questions

1. What is the Output from this program?
2. Draw the object diagram for this program.
3. Describe how you would change the program so that a course title for a lecturer could be printed. Please state clearly every change to the code. i.e. every line change, addition/removal of code.
4. On a scale of 1-5 how understandable do you think this program is?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Very</td>
<td>Moderately</td>
<td>Not very</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Understandable</td>
<td>Understandable</td>
<td>Understandable</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Questionnaire 2

Questions

1. What is the Output from this program?
2. Draw the inheritance hierarchy for this program.
3. Describe how you would change the program so that a supervisor’s room number could be printed. Please state clearly every change to the code. i.e. every line change, addition/removal of code.
4. On a scale of 1-5 how understandable do you think this program is?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Very</td>
<td>Moderately</td>
<td>Not very</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Understandable</td>
<td>Understandable</td>
<td>Understandable</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Questionnaire 3

Questions

1. What is the Output from this program?
2. Draw the object diagram for this program.
3. Describe how you would change the program so that a supervisor’s room number could be printed. Please state clearly every change to the code. i.e. every line change, addition/removal of code.
4. On a scale of 1-5 how understandable do you think this program is?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Very</td>
<td>Moderately</td>
<td>Not very</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Understandable</td>
<td>Understandable</td>
<td>Understandable</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Questionnaire 4

Questions

1. What is the Output from this program?
2. Draw the inheritance hierarchy for this program.
3. Describe how you would change the program so that a course title for a lecturer could be printed. Please state clearly every change to the code. i.e. every line change, removal/addition of code.
4. On a scale of 1-5 how understandable do you think this program is?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Very</td>
<td>Moderately</td>
<td>Not very</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Understandable</td>
<td>Understandable</td>
<td>Understandable</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

REFERENCES


