An Exploration of Novice Programming Errors in an Object-Oriented Environment

Matthew Barr[1], Sam Holden[1], Dave Phillips[1], Tony Greening[2]

[1] Basser Department of Computer Science, University of Sydney, NSW, 2006, Australia
{matt, sholden, davep}@cs.usyd.edu.au

[2] SITMS, University of Ballarat, Victoria, 3353, Australia
t.greening@ballarat.edu.au

Abstract

When studying a programming language for the first time, the majority of student errors fall into broad (and well-documented) categories [3]. This paper aims to investigate errors made by first year students in Blue: A new, object-oriented language specifically designed at the University of Sydney for teaching novice students [2].

These errors were investigated by a survey delivered over the World-Wide Web and consisting of multiple choice and free-form short-answer questions. The results of the survey suggest that a student who learns with Blue is no more likely to make errors that are commonly made by novice programmers, although is not necessarily better equipped to design and write code in an object-oriented paradigm. More research is indicated to make statements about the latter.

Introduction

Blue is a programming language and environment developed specifically for teaching object-oriented programming to first-year computer science students [1,2]. As such it has been designed to make teaching programming concepts easy by removing complexity from the language at the expense of performance.

Blue was used for the first time as the first year language at the Basser Department of Computer Science, the University of Sydney, in 1997. It was the first time that object orientation was taught to first-year students in the department. In addition, the first-year programme incorporated a radical shift in emphasis towards student-centred, group-based education [7,8].

The general trend from procedural to object-oriented languages in first programming courses corresponds to the industry change from smaller, individual coding projects to large-scale group work. Blue attempts to make the construction of larger projects less intimidating for the novice user by presenting a project in a graphical format, in addition to the traditional one. Recently, the Blue graphical environment has been adapted for use with Java.

Method

To gauge the level of understanding of the students an informal survey was taken. This consisted of a web-based form and was entirely voluntary for the students. We chose the survey as a means of quickly obtaining information from a number of students, and with little resource usage.

Of the 574 students who were enrolled in their second semester of Blue at Basser in 1997, 58 completed the survey. This is too small a response to make any strong conclusions, but large enough to get some useful results on which to base further research.

The issue of bias introduced by the type of student would take the time to fill out a voluntary survey has been raised. However, the students have been learning in a problem-based environment, so filling out a survey (and getting feedback on their answers) would be seen as a useful resource by both the better and poorer students. This was the way in which the survey was portrayed to students. Judging by the comments in the free-form answers (especially the last question which just asked for comments on problems with Blue) a wide range of abilities is represented in the survey.

Summary of the Survey

Syntax

The most basic aspect of learning to program is learning the syntax of a programming language; for our students, this is Blue. The syntax of Blue is designed to be easy to learn. Words have been given preference over symbols for language constructs (eg. *do...end* as opposed to *{...}*). Wherever possible, only one way to accomplish most basic tasks has been provided [1,2].

There were many questions in the survey that tested the understanding of Blue syntax. The students were asked to describe what some given code did, to choose the code that was correct, and to comment on errors in code.

The results were encouraging. The vast majority of students indicated that they knew the Blue syntax. The vast majority of the students surveyed did not make many of the errors mentioned in [3].

Use of Comments

One of the key aspects to designing and implementing a large project is writing maintainable code. Blue features a 'comment' tag as a required *syntactical* element for each function and class, and the computer science course attempts to present the importance of writing good comments.

To test the efficiency of this teaching method, we asked students to write what they felt were useful comments in the midst of code that calculated a simple exponent routine.

The results were disappointing. Some students wrote comments that mimicked, rather than explained the code. Others wrote comments that had little or nothing to do with the effect of the code. Only a small percentage of students wrote comments that were useful.

Object-Oriented Design
The crux of object-oriented programming is object-oriented design, so student understanding of the design stage of a programming project is an important indicator of their ability to deliver software solutions. The survey included a couple of multiple choice questions on class inheritance (which is simplified in Blue to support only single inheritance). The questions consisted of selecting the best inheritance tree for a set of classes. Blue's graphical environment was designed to assist in the understanding of such concepts.

The results were also disappointing, as many students did not choose the best option. They did, however, generally choose options that indicated an understanding of the 'is-a' relationship of inheritance. These results may also be partly explained by the fact that the students were not formally introduced to inheritance until second semester.

Code Reading
A very important skill for programmers to develop is an ability to read code written by other programmers. This is especially important when working in groups and on large projects, both of which were emphasised in the CS1 course at Sydney University.

A number of code fragments were presented and students were asked to give free-form comments on the good and bad points (in syntax, logic, and style) of the code. The vast majority of students pointed out errors in syntax, noticing the omission of a comma from a function definition for example. They also noticed problems of style - 'shocking indentation' was a remark from one student. However, the majority did not notice simple logic errors; for example, one question contained a tautological 'if' test that would have resulted in error messages in all cases, but it was not picked up by most of the students. This indicates a tendency of the students surveyed to read code in a superficial manner, which may reflect a similar trend in novice programmers in general.

Sample of Questions and Results from the Survey

This section presents some of the questions used in the survey, with a rationale for its inclusion, a summary of the student's answers to the question, and some thoughts based upon those answers. Note that a study of the Blue environment [6] is left for future research, but as the Blue environment is an integral part of Blue, we feel that this work would be an important complement for this study of the Blue language. To the extent that the environment was designed to support fundamental programming concepts, it plays an implicit role in the current work.

Some of the questions are modified versions of questions found in [5], and the concepts that our survey considered were derived from both [3] and [4].

Question 1:

```
What are the final values of a and b in the
following code fragment:
     var
        a, b : String
     do
        a := "black"
        b := "white"
        a := b
--------------------------------------------------------
a)  a contains "black", b contains "white"
b)  a contains "black", b contains "black"
c)  a contains "white", b contains "white"
d)  a contains "white", b contains "black"
```

Question 1 demonstrates the student's understanding of assignment; specifically, whether assignment is literally assignment, or just a swap operation (a possibility mentioned in [6]). The result was as expected - that students understood the nature of assignment.

Correct answer to Question 1: (c)
Responses:

a) 0%	b) 8%
c) 84%	d) 0%
None: 8%	

Question 6:

```
What is the output of the following code
section when an object is created:

class demo is
internal
     var
        howami : String
interface
     creation is
     ==
     do
        howami := "before"
        print (howami, " ")
        changeme
        print (howami)
     end creation
routines
     changeme is
     == this changes me from before to after...
     var
        howami : String
     do
        howami := "after"
     end changeme

end class
```

```
--------------------------------------------------------
 a)  before before
 b)  before after
 c)  after before
 d)  after after
```

Question 6 covers variable scope. In this case, a very popular answer was wrong. The main reason for an incorrect perception of scoping would appear to be the encouragement that students have had to give all variables in a class different names; their concept of scope is incomplete because they have yet to encounter problems in this area. This may represent an area in which teaching has intruded upon the learning process, by denying students (via sound advice!) the opportunity to experience problem areas associated with scoping and namespace.

Correct answer to Question 6: (b)
Responses:

a)	38%	b)	52%
c)	0%	d)	5%
None:	5%		

Question 9:

```
var
   a : NewClass
   b : NewClass
do
   a := create NewClass ("Learning", 40, "alpha")
   b := create NewClass ("Learning", 40, "alpha")
   if ( a = b ) then
      print ("equal!")
   else
      print ("not equal!")
   end if
--------------------------------------------------------
What will happen if this code is compiled and run?
a)  The code will not compile because the equality
     operator is not defined in NewClass
b)  a = b will give a runtime error because the
     equality operator is not defined in NewClass
c)  "equal!" will be printed
d)  "not equal!" will be printed
```

Question 9 tests an area of Blue that could be a source of confusion. Everything is presented as an object to the students. However, some built-in objects such as strings behave differently from user-defined objects. Comparing two string objects that have been initialised to the same string results in a *true*, while comparing to user-defined classes initialised with the same data results in a *false*.

The majority of the students answered this question correctly, however, a large number of students answered incorrectly. This suggests that the above mentioned inconsistency in Blue is a cause of problems amongst some students.

Correct answer to Question 9: (d)
Responses:

a)	5%	b)	14%
c)	19%	d)	57%
None:	5%		

Question 11:

```
List any errors, bad style, or good style in the
following blue code:

routine doInput( inputVal : Character, a: Integer,
     b: Integer, c: Integer
     d: Integer, name : String, address : String)

== comment for doInput
do
   if (inputVal <> 'a') or (inputVal <> 'b') or
      (inputVal <> 'c') or (inputVal <> 'd') then
      print "Invalid input!\n"
   end if

   if (inputVal = 'a') or (inputVal = 'b') then
      processInput(name,a,b,c)
   end

   if (inputVal = 'c') or (inputVal = 'd') then
      processInput(address,a,b,d)
   end
end doInput
--------------------------------------------------------
Most Popular Answers:
24: The routine lacks proper comments
23: "if" missing from ends
19: Bad variable names
14: Bad use of white space
13: Brackets needed around string to print
13: The keyword "is" is missing
12: There is no character type in Blue.
10: The ifs should be replaced by a case
 9: No apostrophes around a, b ,c and d
 7: Replace ifs with a if - else if sequence
 6: First if test is wrong (always true)
```

The students noticed the syntax errors in the code. They also noticed stylistic problems, mentioning anything that was slightly different than the way the Blue environment formats things (eg. they didn't like tab indents as they had used 2-space indents). However, only a small minority noticed that the first *if* test was incorrect, suggesting a superficial approach to reading code.

Question 13:

```
   exponent (a: Integer, b: Integer) -> (result: Integer)
is
   == (1) add your first comment
   var
      i : Integer
   do
      -- (2) add your second comment
      result := 1
      i := 0
      loop
         exit on (i = b)   -- (3) add your third comment
         result := result * a
         i := i + 1
      end loop
   end exponent
   ---------------------------------------------------------
   The above function takes two numbers, and returns
   the value of the first number raised to the power
   of the second. Write in what you think are
   appropriate comments at the numbered places.
```

The first routine comment was almost unanimously a rewording of the question. However, only a few students mentioned that *b* must be non-negative. For the second comment just under half the students said "initialise variables", or "set result to 1" or similar. The next common option (by 6 students) was along the lines of "compute function", or "execute routine". There was a range of answers for the third comments, many of which were incorrect, and most of which were useless. "Exit when finished" was popular, as was "multiply *a* by itself *b* times" (actually incorrect), and "multiply *result* by *a*, *b* times". Overall, the comments were superficial ones, reiterating what the code did at a low level. Commenting on the bounds of variables was rare, as was an explanation as to why the code worked.

Conclusion

The overall success or failure of Blue cannot be determined from a small study such as this. However, some interesting directions for further research have been identified. The students surveyed on the whole have a reasonable grasp of programming, at a level that would be expected after a semester and a half of programming. So, Blue does not appear to have had a negative impact on the learning of the students.

Students are, however, having more problems with the higher-level object-oriented areas as opposed to syntax and style. Since Blue is designed to enhance the teaching of programming in an object-oriented paradigm this may be a cause of concern. A shift away from the stereotyped novice obsession with low-level, syntactic issues is not evident.

It must be kept in mind that this research occurred in the first year that object-oriented programming was taught at Basser, and a significant shortage of resources hindered delivery of the program; this has been thoroughly addressed in the current course. Thus, the survey was conducted prior to a period of response and evolution of the course.

This has also been the first year of teaching using the Problem-Based Learning (PBL) approach, with the exception of a trial conducted in the previous year. It raises a question as to the source of problems relating to the development of high-level conceptualisations. A principal motivation of PBL is to address exactly such issues, something that was generally supported in its trial. There is some basis for the investigation of an *interaction effect* between the use of Blue and PBL in further research. Problem-Based Learning encourages student independence, and requires a degree of student ownership of the problem domain in order to be successful. Thus, students need a rich set of resources in order to explore the problem domain. While prescribed resources work well for narrowly defined, teacher-centred tasks, PBL feeds on diversity and may require a larger set of resource material. Java, for example, is stimulating a mushrooming set of resources readily accessible via the Internet. The trade-off is that it may possibly be criticised as less appropriate as a "teaching" language, an area in which Blue was designed to excel. This paper does not investigate nor reveal an interaction effect between Blue and PBL. It does, however, suggest the possibility of such. It also, therefore, raises the interesting prospect that two pedagogically sound approaches to education may be mutually diluted in their effectiveness in the presence of each other.

Further study is required to explore the impact of the interaction of these new components of the first year programme on student learning. The study has produced some potentially productive avenues for ongoing research.

Addendum

The first three authors completed this research as a very small part of their honours program in computer science. The last author supervised a number of such projects, all of which - although designated as coursework - clearly required students to engage in independent research. The reason that only some of the results are presented here (apart from page limitations) is simply to showcase the work of these students in sufficient detail to make a number of points. The first is that such experience is of benefit to the students involved. Educational research is typically of a radically different nature to the research undertaken during the honours year in computer science, complementing it and building a more diverse research skills profile. Additionally, it presents a stimulus to reflect on their own learning over the course of their degree; a number of students commented that this was the first time that had been encouraged to actively do so. The learning benefits of such reflection are well recognised. It also offers a sense of partnership with the department to students who have earned a special place in the undergraduate population. As well as benefits to the

Exploration *(continued from page 45)*

students, there are potential benefits to the department. While methodological issues may be raised about the work, it provides a great opportunity for a pilot study to identify areas in need of further research.

The ability of senior undergraduates to offer such contributions to improvements in teaching is a valuable resource that is characteristically under-utilised. Honours students taking this unit on CS education have been found to be enthusiastic, insightful, lateral-thinkers, and free of the inevitable biases that staff may develop towards education and their teaching. Some of the contributions of these students may stimulate new teaching ideas to the benefit of staff. In addition, as such students are often involved in casual teaching work within the department, the improvement in their awareness of teaching and learning issues may transfer to improvement of teaching within the department. Finally, as well as benefits to the honours students and to the school, there are potential benefits for computer science education as a whole, albeit somewhat more remote. The presence of good honours students in an advanced unit on CS education helps promote it as a valid sub-discipline of computer science, in terms of their own perceptions and possibly other members of staff. Ultimately, it may prove to be the first step in establishing computer science education as a future research direction within the department.

References

[1] Kölling, M., Koch, B., & Rosenberg, J. (1995). *Requirements for First Year Object-Oriented Teaching Language*, SIGCSE Bulletin, Vol 27, No. 1 Mar. 1995. pp. 173-177

[2] Kölling, M., & Rosenberg, J. (1996). *Blue - A Language for Teaching Object-Oriented Programming*, Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education, Mar. 1996, pp. 190-194

[3] Pane, J.F., & Myers, B.A. *Usability Issues in the Design of Novice Programming Systems*, School of Computer Science Technical Report, Carnegie Mellon Unversity, CMU-CS-96-132

[4] Basser Department of Computer Science (1997). *Computer Science 101 Workbook: 1997*, University of Sydney

[5] Basser Department of Computer Science (1997). *Computer Science 101 Supplementary Workbook: 1997*, University of Sydney

[6] Kölling, M., & Rosenberg, J. (1996). *An Object-Oriented Program Development Environment for the First Programming Course*, Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education, Mar. 1996, pp. 83-87

[7] Greening, T., Kay, J., Kingston, J.H., & Crawford, K. (1996). *Problem-Based Learning of First-Year Computer Science*, Proceedings of the First Australasian Conference on Computer Science Education, 1996, pp. 13-18.

[8] Greening, T., Kay, J., Kingston, J.H., & Crawford, K. (1997). *Results of a PBL Trial in First-Year Computer Science*, Proceedings of the Second Australasian Conference on Computer Science Education, 1997, pp. 201-206.