Novice Programmers in the First Three Semesters: Gaining a Better Understanding of the Problem

Malcolm Corney[1], Raymond Lister[2], James Hogan[1], Colin Fidge[1], Mike Roggenkamp[1] and Rachel Cardell-Oliver[3]

Intended Outcomes

The initial aim of this project was to carry out a longitudinal study across three semesters of students' performance on in-class test questions and on specific exam questions. The reason for carrying out such a longitudinal study was to see at what stage of their studies students' were able to demonstrate understanding of various concepts required for learning computer programming.

When the grant application was made, we were already in the process of collecting student performance data for specific test and exam questions for those students commencing their studies in Semester 1, 2011 for the first programming unit at QUT.

Specific outcomes from the project were intended to be:

1. An archive of in-class test and examination questions which could be disseminated to academics at other universities.
2. An anonymised repository of students' in-class test attempts and examination scripts for analysis of student progression. While the data was to be collected from students at QUT, UTS and UWA, the aim was for this data to be made available to other collaborators.
3. Performance data from students for a subset of the in-class tests and examinations.
4. It was also intended that the results of the project be disseminated via papers and workshops at computing education conferences both nationally and internationally, and to raise interest and broaden the project to include collaborators from other Australian universities, as part of an ALTC/DEEWR funded project.

Project Approach

Recent research on novice programmers has suggested that they pass through neo-Piagetian stages - sensorimotor, preoperational, and concrete operational - before eventually reaching programming competence at the formal operational stage (Lister 2011). The project began by creating some examination questions aimed at testing novice computer programmers for their concrete operational abilities to reason with quantities that are conserved, processes that are reversible, and properties that hold under transitive inference. These three questions are included in Appendix 1.

Another programming problem that has been used previously by other researchers is sometimes referred to as Soloway's Rainfall problem (Soloway 1986). A question based on this problem was devised and used in in-class tests and examinations in the first three computer programming units at QUT. This question is also included in Appendix 1.

A series of "Explain in Plain English" questions, testing program reading and comprehension skills was also developed. A subset of these questions have been published in Corney, Teague et al (2012).

[1] School of Electrical Engineering and Computer Science, Queensland University of Technology
[2] Faculty of Engineering and Information Technology, University of Technology, Sydney
[3] School of Computer Science and Software Engineering, The University of Western Australia

Some of these above questions were used in Semesters 1 and 2, 2011 across three units at QUT and in one unit at UTS. One of the original collaborators from UWA left the project as he took up an industry secondment. The remaining UWA collaborator was not involved in a suitable unit for these questions to be used in 2011.

Performance data was collected and analysed for these questions. Performance by students in their first and second units of the study of computer programming was poor on these selected questions.

This lead the project team to develop a series of incrementally more difficult questions for each of the three questions aimed at testing the concrete operational stage of neo-Piagetian development. The individual conceptual elements of these questions were pulled apart in an attempt to determine if individual concepts were causing difficulty for students or if the combined abstractions were the cause of the difficulties facing our students.

These questions were asked in a series of in-class tests at QUT, UTS and in on-line quizzes at UWA, and also in end of semester examinations at QUT and UTS. As examinations for Semester 1, 2012 have only just been finalised, performance data is still being collected, collated and analysed.

Project Outcomes

1. The questions in Appendix 1 below were used in in-class tests at QUT and UTS in Semesters 1 and 2, 2011. These questions were also published in conference papers (Corney et al. 2012; Lister 2011) for academics at other universities to use.

   The questions in our second iteration of student testing were used in in-class tests, on-line quizzes and end of semester examinations at QUT, UTS and UWA. These questions were also distributed to other academics who registered interest in our work from RMIT University, University of Newcastle and Charles Darwin University (Darwin).

   Questions that were part of our archive (Corney, Lister, and Teague 2011) prior to the commencement of the ALTA grant, have since been used in a replication study overseas (Murphy, McCauley, and Fitzgerald 2012).

2. In-class tests and end of semester examinations carried out during the project have been scanned and saved as PDF documents and collated to form a database of raw data. Software tools have been prepared for redaction of student identifiers in these PDF documents. While this data has been used and is being used by the project team, it has not yet been published for external use.

3. Performance data has been analysed from in-class tests and end of semester examinations from Semesters 1 and 2, 2011 from the first three programming units at QUT and from the first programing unit at UTS.

   Raw data has been collected from in-class tests, on-line quizzes and end of semester examinations from the first two programming units at QUT, and from the first programming units at UTS and UWA. The raw data is still in the process of being collated and analysed.

   Including the data collected prior to the commencement of the ALTA grant, we have longitudinal performance data for three cohorts of QUT students across their first two units of study in computer programming. We intend to continue collecting such performance data for future students.

4. After the ALTA grant was received, the QUT and UTS collaborators along with participants from four other Australian universities were involved in a successful ALTC funded grant bid (Lister et al. 2012). This ALTC grant covers work in three main areas, one of which was the work funded by this ALTA grant.

   The work conducted on this ALTA grant project has resulted in two computing education conference papers to date (Corney et al. 2012; Teague et al. 2012). We have also submitted papers to Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2012), International Computing Education Research (ICER 2012), and Australasian Association for Engineering Education Annual Conference (AAEE 2012). The first two of these submissions were unsuccessful, but the work from these papers will be submitted to the Australasian Computing Education Conference (ACE 2013). We are still to learn if the AAEE paper will be accepted.

   The project participants have also made the beginnings of collaborations with academics at Charles Darwin University (through a contact made at the ALTA Forum in Brisbane, 2012) and the University of Pretoria (South Africa).

Benefit to ICT Learning and Teaching Community

The main benefit of the work carried out to date in this project has been the refinement of test, quiz and exam questions aimed at determining if novice computer programming students have reached the concrete operational stage of reasoning in the neo-Piagetian scheme of development. While it is hoped that university education will result in students who have reached the formal operational level of reasoning about computer programs and programming, it is not expected that students undertaking our introductory level units will achieve this.

Analysis of our results is giving us a better understanding of the conceptual difficulties that novice programmers face, and allowing us to improve our teaching materials so that more of the students passing through our introductory level units reach the concrete operational stage of reasoning.

Even though our results analysis is incomplete, we will continue to test our students with the current question archive, and we will continue to develop new questions about concepts not covered by our current question archive. We will continue to publish in computing education conferences and journals and these publications will include some of the questions from our archive, thus disseminating our work to other academics both nationally and internationally.

Acknowledgements

The project participants would like to thank the ACDICT Learning and Teaching Academy for their generous support of this work.

Appendix 1

Reversing a Process

The purpose of the following code is to move all elements of the array `x` one place to the ***right***, with the ***rightmost*** element being moved to the ***leftmost*** position:

```
int temp = x[x.length - 1];
for (int i = x.length - 2; i >= 0; --i) {
      x[i + 1] = x[i];
}
x[0] = temp;
```

Write code that undoes the effect of the above code. That is, write code to move all elements of the array `x` one place to the ***left***, with the ***leftmost*** element being moved to the ***rightmost*** position.

Conservation

Below is incomplete code for a method which returns the smallest value in the array `x`. The code scans across the array, using the variable `minsofar` to remember the smallest value seen thus far. There are two ways to implement remembering the smallest value seen thus far: (1) remember the actual value, or (2) remember the value's position in the array. Each box below contains two lines of code, one for implementation (1), the other for implementation (2).

First, make a choice about which implementation you will use (it doesn't matter which). Then, for each box, draw a circle around the appropriate line of code so that the method will correctly return the smallest value in the array.

```
public int min(int x[] ){
   int minsofar = [ (a) 0
                    (b) x[0] ] ;
   for ( int i = 1; i < x.length; ++i ) {
      if ( x[i] < [ (c) minsofar
                    (b) x[minsofar] ] )
      minsofar = [ (e) i
                   (f) x[i] ] ;
   }
   return [ (g) minsofar
            (h) x[minsofar] ] ;
}
```

Transitive Inference

In plain English, explain what the following segment of Java code does:

```
bool bValid = true;
for (int i = 0; i < iMAX - 1; i++) {
   if (iNumbers[i] > iNumbers[i + 1]) {
      bValid = false;
   }
}
```

Soloway's Rainfall Problem

Write a Python program that repeatedly reads in numbers, until it reads the number 99999. Your program should ignore negative numbers. After seeing 99999, it should print out the average of the positive numbers. There is no need to define a function in your program.

To get numeric input from the user Python has a function named input. This function can be given a string as its parameter which acts as a prompt , e.g. "Enter a number: ". It then waits until the user enters a value and it returns the value that was input.

Bibliography

Corney, Malcolm W, Raymond Lister, and Donna M Teague. 2011. Early Relational Reasoning and the Novice Programmer: Swapping as the "Hello World" of Relational Reasoning. In *13th Australasian Computer Education Conference (ACE 2011)*, edited by John Hamer and Michael de Raadt. Perth, Australia: Australian Computer Society, Inc.

Corney, Malcolm W, Donna M Teague, Ahadi Alireza, and Raymond Lister. 2012. Some empirical results for neo-Piagetian reasoning in novice programmers and the relationship to code explanation questions. In *14th Australasian Computing Education Conference (ACE 2012)*, edited by Michael de Raadt and Angela Carbone. Melbourne, Australia: Conferences in Research and Practice in Information Technology.

Lister, Raymond. 2011. Concrete and Other Neo-Piagetian Forms of Reasoning in the Novice Programmer. In *13th Australasian Computer Education Conference (ACE 2011)*, edited by John Hamer and Michael de Raadt. Perth, Australia: Conferences in Research and Practice in Information Technology.

Lister, Raymond, Malcolm W Corney, James Curran, Darryl D'Souza, Colin J Fidge, Richard Gluga, Margaret Hamilton, James Harland, Jim Hogan, Judy Kay, Tara Murphy, Mike Roggenkamp, Judy Sheard, Simon, and Donna M Teague. 2012. Toward a shared understanding of competency in programming : an invitation to the BABELnot Project. In *14th Australasian Computing Education Conference (ACE2012)*, edited by Michael de Raadt and Angela Carbone. Melbourne, Australia: Australian Computer Society.

Murphy, Laurie, Renée McCauley, and Sue Fitzgerald. 2012. 'Explain in plain English' questions: implications for teaching. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education (SIGCSE 2012)*.

Soloway, Elliot. 1986. "Learning to program = learning to construct mechanisms and explanations." *Communications of the ACM* no. 29 (9):850-858.

Teague, Donna M, Malcolm W Corney, Ahadi Alireza, and Raymond Lister. 2012. Swapping as the "Hello World" of relational reasoning : replications, reflections and extensions. In *14th Australasian Computing Education Conference (ACE 2012)*, edited by Michael de Raadt and Angela Carbone. Melbourne, Australia: Conferences in Research and Practice in Information Technology (CRPIT).