CSEd Research Instrument Design: the Localisation Problem

Jacqueline L. Whalley
Computer and Information Sciences
Auckland University of Technology
New Zealand
jwhalley@aut.ac.nz

Abstract

Recently multi-institutional multi-national (MIMN) studies have increased in popularity in CSEd research. This paper addresses some of the issues surrounding this genre of research. In particular, does the localisation of study instruments to accommodate institutional differences in teaching paradigms and programming languages mean that conclusions drawn as a result of combined institutional data analysis are invalid? The results presented here show that the localisation of the BRACElet program comprehension question set did not produce a significant impact on the overall performance results.

Keywords: Computing education, novice programming, multi-institutional, programming language, problem localisation , research instrument design.

1 Introduction

A current trend in CSEd research is the adoption of the multi-institutional multi-national (MIMN) study. Such studies are becoming increasingly popular. This popularity is directly attributable to the clear advantages of using combined research data and experiment toolkits. The advantages of a MIMN approach to research include: increased statistical power, richness (address a broader range of issues), improved hypothesis generation and improved methodology (Fincher et. al 2005).

Bracelet is an MIMN study designed to investigate the reading and comprehension skills of novice programmers. It extends previous studies (Lister *et al*. 2004, McCracken et. al. 2001) by developing a question set within two key pedagogical frameworks: the revised Bloom (Anderson et.al 2001) and SOLO (Biggs and Collis 1982) taxonomies. The BRACElet team adopted the Leeds Working Group study design (Fincher *et al*. 2005) but the individual researchers co-designed the instruments used in the project.

Despite their obvious advantages, some non-trivial issues need to be addressed when MIMN studies are undertaken. Fincher et. al. (2005), noted the inherent difficulty of inter-rater reliability of data generated in different institutions. Another difficulty of MIMN studies is the adaptation (localisation) of a problem set to allow for individual institutional teaching approaches, programming language and paradigms without compromising the validity and reliability of the instrument. Indeed, in our experience, designing an instrument that covers multiple programming languages and their distinct idioms and yet presents the same problem set with the same difficulty level is not easy.

This paper reports on the localisation of the multi-choice questions in the BRACElet problem set.

2 The Problem Set

Two of the multiple-choice questions (MCQs) used in the BRACElet study (items 1 and 9) were taken directly from the Leeds working group instrument (5 and 2 respectively; Lister *et al*. 2004). These two questions were included to enable some direct comparative analysis. The remaining 7 MCQs were designed by the BRACElet working group, with the express purpose of devising a diverse set of questions to evaluate the program comprehension skills of novice programmers.

A multiple-choice format was used because this form of question can be administered consistently across institutions and marked reliably by different markers.

The main concern of the BRACElet team in developing a common set of programming comprehension questions for use across different institutions was the necessary localisation of the study instruments and how this may affect the validity of the conclusions drawn from the analysis of the combined institutional data.

The cognitive framework provided by the revised Bloom's taxonomy (Anderson et. al 2001) was used to try to ensure that the cognitive difficulty of each MCQ was not changed as a consequence of the localisation process. Once each question had been written it was analysed in terms of its perceived cognitive difficulty and allocated to a category within Bloom's taxonomy. After the questions were localised for each institution in the study, every question was again analysed and categorised. Application of Blooms taxonomy to the problem set was difficult at the subcategory level and we found that it was not possible to reliably assign a question to a subcategory. Therefore, for the purposes of the localisation of the problem set, we used the higher-level categories.

3 Localisation of the Problem Set

In MIMN studies of novice computer programmers localisation has usually been motivated by two distinct factors; the programming language used for teaching introductory programming and the way in which topics were taught at different institutions.

The institutions involved in the current iteration of the BRACElet project teach using object oriented programming languages C#, C++, Java, VB.NET and Delphi. Although all these languages follow the object-oriented paradigm there

This paper appeared at the *19th Annual Conference of the National Advisory Committee on Computing Qualifications* (NACCQ 2006), Wellington, New Zealand. Samuel Mann and Noel Bridgeman (Eds).

are differences in syntax and approaches to teaching programming.

One example of difference in teaching approach is the way in which algorithmic logic is taught and represented in the participating institutions. For example, two institutions use a mixture of flow diagrams and pseudo-code whereas others use structure diagrams.

A language specific consequence emerges when question stems are presented in Delphi. In some cases they require more lines of code than in C++ and/or additional explanation. In order to ensure that each question was presented on a single page the layout of the Delphi multi-choice answers varied from that of the standard C++/Java generic problem set in some cases. A complete set of problems in a number of programming languages can be found in the project's web site (Bracelet).

The original problem set was presented and developed by the research team in Java syntax. The localisation of the problem set was carried out by the teaching teams at each institution and then circulated amongst the entire BRACElet team for moderation.

3.1 Question 6 illustrating the main localisation processes

A detailed look at question 6 illustrates the main localisation processes undertaken in the translation of the BRACElet problem set.

3.1.1 Array Declaration and Initialisation

The Delphi students, who participated in this study, had not yet been taught dynamic arrays. It is only by using dynamic arrays that initialisation and creation can be carried out in one step in Delphi. On the other hand this is not an issue for the C++ version where the code in the question stem that creates and initialises an array can be declared in one step. The C++ question stem is provided in Figure 1 and the Delphi version is provided in Figure 2.

To present the question stem in Delphi in a simple and concise manner the stem was altered significantly. The students were instructed to assume that the array and fields had been declared in the appropriate section of the class and they were provided with a table that visualised the array, its indices and values.

In the following segment of code, **iNumbers** is an array of integers:

```
int iNumbers[iMAX] = {3,6,13,27,12,2,7};
int i = 0;
int iSum = 0;

while (i < iMAX)
{
        iSum = iSum + iNumbers[i];
        i++;
}
```

What is the value in the variable *iSum* after the code fragment is executed?

Figure 1: The C++ version of the question 6 stem

For this problem, assume that the following have been declared:

```
aiNumbers : array[1..7] of integer;
iLoop, iSum : integer;
```

The array has had the following values assigned to it:

Index	1	2	3	4	5	6	7
Value	3	6	13	27	12	2	7

The following code processes the array:

```
iLoop := 1;
iSum := 0;

while iLoop <= Length(aiNumbers) do
begin
   iSum := iSum + aiNumbers[iLoop];
   Inc(iLoop);
end;
```

What is the value in the variable *iSum* after the code fragment is executed?

Figure 2: The Delphi version of the question 6 stem.

3.1.2 Loop Termination Condition

The termination condition of the while loop for each of the two language versions is different. In C++ the loop terminating condition is *i < iMAX* and in the Delphi version it is *iLoop <= Length(aiNumbers)* which is equivalent to initialising *i* to one and using *i <= iMAX* in C++. This difference was necessary because the Delphi students had been taught to create and loop through Delphi arrays from an index of one to the length of the array. This is not possible in C++ where arrays always begin with an index of 0.

The reason that students had been taught to create arrays from an index of 1, rather from 0 which is a perfectly valid way of creating arrays in Delphi**,** was that the teaching team had found that students understood array-processing algorithms more readily if the array indices started from 1.

Figure 3 shows that there is little difference between the responses in Delphi and in C++. However, Distracter **D** appears to be stronger for the C++ students and it may be postulated that this is due to the difference between the maximum index value and the length of the array (6 and 7 respectively) that not unsurprisingly causes some confusion.

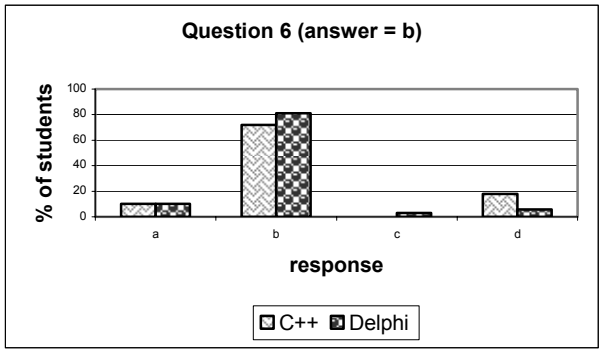

Figure 3: Responses to question 6 (N(C++) = 46 & N(Delphi) = 71).

The answer choices shown in Figure 4 are virtually the same in Delphi and C++ (for the Delphi version `iLoop` replaces `i` in answer **A**). Because the terminating condition is less than the length of the array some of the C++ students may have incorrectly assumed that the loop misses the last element in the array, forgetting that loop counter `i` has been initialised to 0 not 1. It may also be argued that the table provided to the Delphi students, suggestive of a position doodle (Lister *et al*. 2004, McCartney, R., Moström, J. E., Sanders, S. and Seppälä, O. 2004), may account for fewer Delphi students being fooled by distracter **D**.

A	sum of all values of `i`
B	sum of all elements in the array
C	sum of all elements in the array except for the first element
D	sum of all elements in the array except for the last element

Figure 4: The C++ answer choices.

4 Data collection

The data collected in the BRACElet study consisted of student answers to nine MCQs and two short answer questions. Responses from one hundred and seventeen students to the nine MCQs were used in the analyses reported in this paper. These students had either nearly completed or just completed the first semester of their first programming course. Of the one hundred and seventeen students 71 completed the problem set in Delphi and 46 completed it in C++.

Each institution was responsible for administering the problem set. Each researcher decided when students at their own institution would be capable of successfully completing the problems. In all cases the MCQs counted towards the student's final grade and were taken under examination conditions.

5 Data Analysis

A full analysis of the performance data for the MCQs was undertaken and reported by Whalley et. al. (2006). This initial analysis looked at the performance data independent of problem set localisation issues. Overall it was found that the most difficult question was question 7 and the easiest question was question 2 (Figure 5).

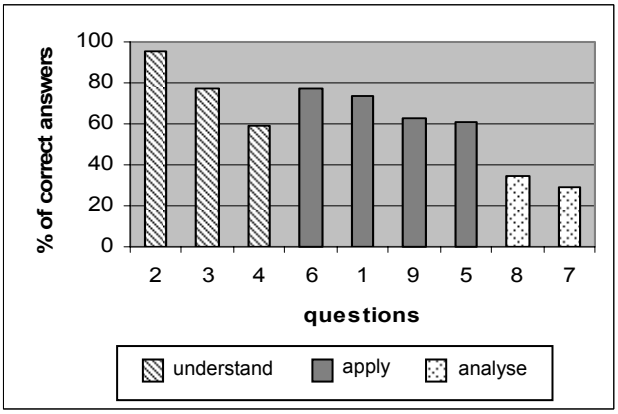

Figure 5: Performance by Bloom's category (N = 177).

In order to evaluate the possible effect of problem set localisation the analysis that follows compares the performance of students who completed the problem set in Delphi with those who completed it in C++ across two institutions. Unfortunately, there was insufficient data for comparative analysis of the C#, VB.NET and Java versions of the problem set.

No significant difference was found between the mean total score of students who completed the questions in Delphi and that of the students who completed the test in C++ ($t = 0.24$, $p = 0.811$, NS). This finding provides support to the notion that the localisation of the question set was achieved without distorting the difficulty level of the questions.

An analysis of differences on each of the nine MCQs was undertaken to see whether or not there were important differences in performance that were not revealed by the analysis of total scores. Some of the MCQ distracters attracted a very small number of responses so it was not possible to undertake a statistical analysis of the effect of language on each distracter separately. We therefore consolidated the data from the three distracters into a single category of incorrect response.

Significant differences were found in only two of the nine questions. In question 3 students who answered in C++ were more likely to answer correctly ($\chi^2=13.648$, $p<0.001$) whereas in question 4 students who answered in Delphi were more likely to answer correctly ($\chi^2=3.945$, $p<0.05$).

What follows is a discussion of a subset of the 9 MCQs. Firstly the two questions in which student performance was significantly different for the problem set. Then a look at the over pattern for the easiest and hardest questions in the problem set.

5.1 Question 3 – Is it really easier in C++

In question three the students were provided with the logic of an algorithm that copied the contents of an array in reverse order into a new array, represented as a structure diagram. The students were required to choose the code segment that correctly implemented the logic illustrated in the diagram.

All the students had used structure diagrams as a key component of their course and were expected to perform equally on this question but the C++ students actually performed significantly better. What made question 3 more difficult in Delphi than in C++?

Figure 6 shows that the distracters **A**, **B** and **C** were, in this instance, more powerful in Delphi than in C++? What makes them so? Is it that the answer code segments are more difficult to read in Delphi?

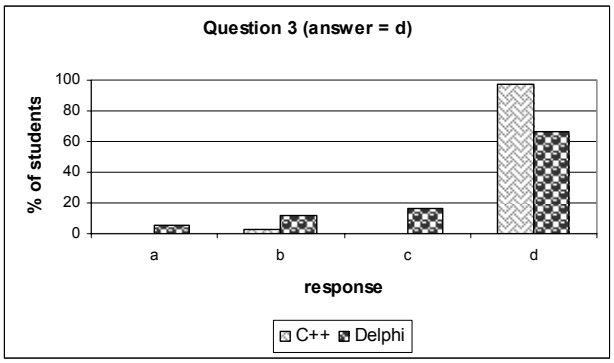

Figure 6: Student responses aggregated by language for question 3 (N(C++) = 46 & N(Delphi) = 71).

Figure 7 contains the C++ and Delphi versions of one potential answer; the other three choices exhibit similar features. There are differences such as loop termination and initial index values between the versions, but these exist in most of the problems in the BRACElet set.

If small syntactic differences are ignored the body of each of the while loops is the same. If the differences in language were important we would expect the Delphi students to have experienced the same difficulty with question 4 where the answer set was also provided as code segments. The localisation process does not seem to have produced differences in question difficulty that could account for difference in success found between the C++ and Delphi groups.

The students who used Delphi had been taught structure diagrams in the third week of their course. They had also completed an assessment that required them to represent the solution to a problem involving a series of interrelated structure diagrams as the first phase of a staged assessment. When they sat the test it had been a month since they had been required to use structure diagrams and during this period some of the students may have forgotten how to interpret them. However, this does not seem to be a particularly satisfactory explanation and further investigation of differences in the way in which Delphi and C++ students answer this question is needed before any firm conclusion can be reached.

Question 2, the easiest question of the problem set, was a similar type of problem to question 3 but no difference was found between the Delphi and C++ students on this question. Both questions involved a translation of logic from one form to another. In the case of question 2 the students were provided with a piece of code and asked to choose the flow diagram that represented the logic of the code. A discussion of the overall performance for questions 2 and 3 can be found in Whalley *et al*. (2006).

Neither the Delphi students nor the C++ students had been taught flow diagrams as a way of presenting algorithmic logic. Despite this they apparently found the interpretation of flow diagrams easier than structure diagrams.

C++

```
iIndex1 = 0;
iIndex2 = iMAX;
while (iIndex1 <= iMAX)
{
  iIndex2--;
  iArray2[iIndex2]=iArray1[iIndex1];
  iIndex1++;
}
```

DELPHI

```
iIndex1 := 1;
iIndex2 := MAX_SIZE + 1;
while iIndex1 < MAX_SIZE do
begin
  Dec(iIndex2);
  aiArray2[iIndex2] := aiArray1[iIndex1];
  Inc(iIndex1);
end;
```

Figure 7: Answer code segment for question 3.

5.2 Question 4 – Is it really easier in Delphi

Question 4 gives a segment of code that checks to see if a number is within a given range. The students are required to select from the answer set the segment of code that gives exactly the same result as the code provided in the question stem. In the answer the 'range logic' is reversed, as illustrated in figure 8 (C++ version).

stem code:

```
if((iNum < iMIN_NUM)||(iNum > iMAX_NUM))
{
  bValid = False;
}
else
{
  bValid = True;
}
```

(A)- the correct answer

```
if((iNum >= iMIN_NUM)&&(iNum <= iMAX_NUM))
{
  bValid = True;
}
else
{
  bValid = False;
}
```

Figure 8: Question 4 code stem and answer.

The percentage response for each distracter aggregated by programming language is given in figure 9.

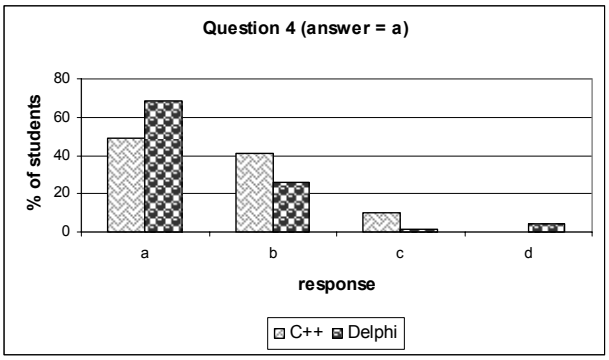

Figure 9: Responses to question 4.

There is no obvious reason why the students using Delphi found this question easier than those using C++.

One feature that was noted on the scripts of the Delphi students that was not present on the C++ student scripts was a doodle* or annotation that we call a 'range logic doodle'. An example from a student script is shown in Figure 10. Range annotation led to a correct answer in 90% of the cases where it was employed. Interestingly, this is a doodle that the tutors on the Delphi course had used to illustrate the subtleties of range logic and that, in turn, the student body had adopted.

It may be due to these range doodles that the students who were taught Delphi performed better on this question. A similar trend was observed by McCartney, Moström, Sanders and Seppälä (2004): who used a similar style of multiple-choice problem set and found that "Performance … improves when students annotate their tests…" (2004, p. 18).

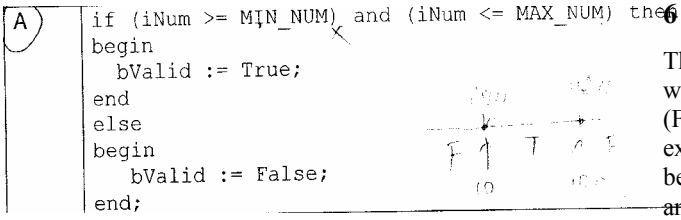

Figure 10: A students 'range logic doodle'.

5.3 The most difficult question (7) and the easiest question (2)

Independent of language, the group of students at each institution found Q7 to be the hardest question and Q2 to be the easiest question in the problem set.

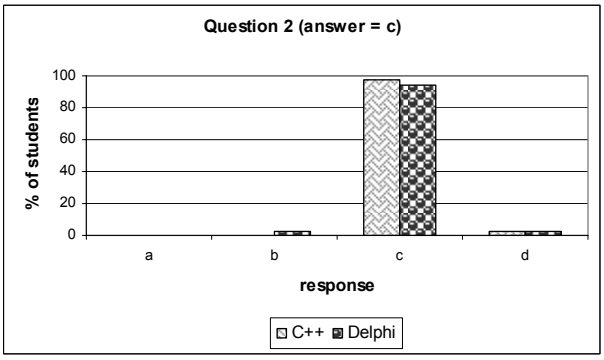

Figure 11: Responses to the easiest question.

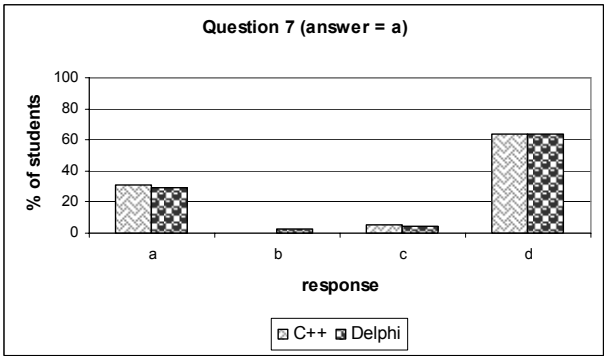

Figure 12: Responses to the hardest question.

Indeed even the effect of the distracters was almost identical, independent of language, as illustrated by Figures 11 and 12. This provides additional support for the view that the localisation process was achieved without distorting the difficulty level of the questions.

6 Conclusion

The use of Bloom's taxonomy as a cognitive framework within which questions were analysed for difficulty level (Figure 5) and within which the localisation of questions was examined for possible changes in difficulty appears to have been useful. Analysis of the actual responses of the Delphi and C++ students indicates that on most questions the taxonomy facilitated the localisation of the question set without causing changes in question difficulty.

More work needs to be done in refining Bloom's taxonomy to ensure that the categories, and especially the exemplars of categories, are useful for accurately identifying the difficulty of programming ideas and processes. However this study indicates that the taxonomy has the potential to be a positive force in guiding the localisation of question sets for MIMN studies.

The problems used in this study were similar in nature in that they required students to understand simple procedural algorithms. They did not require the students to understand or be aware of basic object oriented concepts such as classes and object interaction. Would the same result have been obtained using questions that target a higher level of thinking? The cognitive framework used has yet to be tested with questions representing a wider range of cognitive levels.

* A full analysis of the script annotations has been undertaken and will be reported in a future paper.

The findings of this study also add fuel to the debate about the programming language used in the delivery of introductory programming courses.

The choice of which instructional programming language to use for CS 1 has occupied a considerable amount of the time of computer science educators in recent years. "Some have even dubbed the debate as the 'language wars'." (Goldweber 1999, p.1). From the results of this initial study it appears that student performance is not hindered by language. This is corroborated to some extent by Grandell, L., Peltomäki, M., Back, R-J. and Salakoski T. (2006): in their discussion on the difficulties that students experience when learning programming.

Of course the choice of programming language is not only driven by pragmatic considerations such as providing a suitable learning and teaching development environment and pedagogy but also by students who demand instruction in a language that they perceive as popular, fun and in demand in the commercial sector (Smith and Rickman 1976, Clear and Sharp 1998, Guzdial and Soloway 2002, Grandell *et al*. 2006).

Acknowledgements

Data used in this study was contributed and collected by a number of dedicated programming educators in a number of institutions. Thanks to Gordon Grimsey, Christine Prasad, Ajith Kumar and Phil Robbins for their contribution to the data collection phase.

I also wish to acknowledge the contributions of:

- The entire BRACElet team for the development of the initial draft of the study instrument.
- Raymond Lister, Tony Clear and Errol Thompson for their helpful discussions and enthusiasm for the research being carried out on the BRACElet project.
- Tony Clear for the insightful conversations not only during the preparation of this paper but also during the course of the BRACElet project.

7 References

Anderson, L.W., Krathwohl, D.R, Airasian, P.W., Cruikshank, K.A., Mayer, R.E., Pintrich, P.R., Raths, R. and Wittrock, M.C. (Eds) (2001): *A Taxonomy for Learning, Teaching and Assessing. A Revision of Bloom's Taxonomy of Educational Objectives*. New York, Addison Wesley Longman, Inc.

Biggs, J. B. & Collis, K. F. (1982): *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*. New York, Academic Press.

Bracelet Project http://online.aut.ac.nz/Bracelet/repository.nsf/ (accessed April, 2006)

Clear, T. and Sharp, D., (1998) First Year Language Selection Criteria, Internal Report, Auckland University of Technology, Auckland, New Zealand.

Fincher, S., Lister, R., Clear, T., Robins, A., Tenenberg, J. and Petre M. (2005): Multi-institutional, multi-national studies in CSEd research: Some design considerations and trade-offs. *In Proceedings of the First International Computing Education Research Workshop, ICER'05, SIGCSE, ACM,* 111-121.

Goldweber, M. (1999): A Report on the Use of HyperTalk in CS1 Within a Liberal Arts Setting. *SIGCSE Bulletin,* **31**(2): 37-41.

Grandell, L., Peltomäki, M., Back, R-J. and Salakoski T. (2006): Why Complicate Things? Introducing Programming in High School Using Python. *IN Proc. of the 8th Australasian Computing Education Conference, Hobart, Australia*, *Conferences in Research in Practice in Information Technology*, **52**: 71-80.

Guzdial M. and Soloway, E. (2002): Teaching the Nintendo Generation to Program. *Communications of the ACM,* **45**(4): 17-21.

Lister, R., Adams E.S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J.E., Sanders, K., Seppällä, O., Simon, B. and Thomas, L. (2004): A Multi-National Study of Reading and Tracing Skills in Novice Programmers. *SIGCSE Bulletin*, **36**(4):119-150.

McCartney, R., Moström, J. E., Sanders, S. and Seppälä, O. (2004): Questions, Annotations and Institutions: observations from a study of novice programmers. *In Kolin Kolistelut - Koli Calling. Koli, Finland,* 11-19.

McCracken, M., V. Almstrum, D. Diaz, M. Guzdial, D. Hagen, Y. Kolikant, C. Laxer, L. Thomas, I. Utting, T. Wilusz, (2001): A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-year CS Students. *SIGCSE Bulletin*, **33**(4):125-140.

Smith, C. and Rickman, J. (1976): Selecting languages for pedagogical tools in the computer science curriculum. *SIGCSE Bulletin*, **8**(3): 39-47.

Whalley, J. L., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar A. P. K. and Prasad, C. (2006): An Australasian Study of Reading and Comprehension Skills in Novice Programmers, using the Bloom and SOLO Taxonomies. In *Proc. of the 8th Australasian Computing Education Conference, Hobart, Australia, Conferences in Research in Practice in Information Technology*, **52**: 243-252.