Full paper

Reliably Classifying Novice Programmer Exam Responses using the SOLO Taxonomy

Tony Clear
School of Computing and Mathematical Sciences, Auckland Univ. of Technology, New Zealand
tony.clear@aut.ac.nz

Jacqueline L. Whalley
School of Computing and Mathematical Sciences, Auckland Univ. of Technology, New Zealand
jacqueline.whalley@aut.ac.nz

Raymond Lister
Faculty of Engineering and Information Technology, Univ. of Technology, Sydney, Australia
raymond@it.uts.edu.au

Angela Carbone
Faculty of Information Technology, Monash University, Australia
Angela.Carbone@infotech.monash.edu.au

Minjie Hu
School of Business, Computing and Foundation, Tairawhiti Polytechnic, Gisborne, New Zealand
min@tairawhiti.ac.nz

Judy Sheard
Faculty of Information Technology, Monash University, Australia
Judy.Sheard@infotech.monash.edu.au

Beth Simon
Computer Science and Engineering Department, University of California, San Diego, USA
esimon@cs.ucsd.edu

Errol Thompson
2 Haven Grove, Lower Hutt, New Zealand
kiwiet@computer.org

Abstract

Past papers of the BRACElet project have described an approach to teaching and assessing students where the students are presented with short pieces of code, and are instructed to explain, in plain English, what the code does. The student responses to these types of questions can be analysed according to the SOLO taxonomy. Some students display an understanding of the code as a single, functional whole, while other students cannot "see the forest for the trees". However, classifying student responses into the taxonomy is not always straightforward. This paper analyses the reliability of the SOLO taxonomy as a means of categorising student responses. The paper derives an augmented set of SOLO categories for application to the programming domain, and proposes a set of guidelines for researchers to use.

Keywords: SOLO taxonomy, BRACElet Project, computing education research, novice programmers, assessment.

1 Introduction

A number of psychological studies have shown that novice programmers frequently understand the parts of a program but struggle to organize those parts into a coherent whole (McKeithen, Reitman, Rueter & Hirtle, 1981; Adelson, 1984; Wiedenbeck, Fix and Scholtz, 1993). Those psychological studies suggest that we need teaching techniques that help students to see "the forest", not just "the trees". That is, we need teaching techniques that help students to see the relationships between the parts of a program, and assessment techniques to test for that ability.

Academics participating in the multi-institutional BRACElet Project have experimented with a simple, pragmatic approach to developing and also assessing the capacity of novices to "see the forest" (Whalley, et al., 2006; Lister et al., 2006). In this approach, students are presented with short pieces of code, and are instructed to explain, in plain English, what the code does. The student responses are classified according to the first four levels of the SOLO taxonomy (Biggs & Collis, 1982). (As this type of question provides minimal opportunity to provide an 'extended abstract' response (the highest level in the SOLO taxonomy), this was excluded as an outcome.)

The SOLO taxonomy is not discipline-specific and, apart from Thompson (2004), we are not aware of any application of the SOLO taxonomy to novice programming prior to the BRACElet Project. Members of BRACElet have defined what the taxonomy means, in the context of teaching novice programmers, which is described in Table 1.

As a Multi Institutional project, the BRACElet project has had to deal with many pragmatic aspects of research coordination, including developing an agreed understanding of, and process for, classifying student responses according to the SOLO taxonomy. To develop a shared understanding of the SOLO taxonomy, participants at the 6th BRACElet workshop (held at Auckland University of Technology (AUT), on December 18, 2007) collectively rated a small sample

This quality assured paper appeared at the *21st Annual Conference of the National Advisory Committee on Computing Qualifications* (NACCQ 2008), Auckland, New Zealand. Samuel Mann and Mike Lopez (Eds).

of student responses to three examination questions, using data from AUT's introductory programming students. Subsequently, a sub group of three BRACElet participants worked on documenting and further developing the SOLO classification process, into a reliable method for teaching and assessment, and also into a reliable data analysis technique for further research in the BRACElet project. This paper now proceeds to examine the SOLO classifications of the student responses by the three BRACElet participants who form the SOLO subgroup.

Table 1: SOLO Categories

SOLO category	Description
Relational [R]	Provides a summary of what the code does in terms of the code's purpose. (The "forest")
Multistructural [M]	A line by line description is provided of all the code. (The individual "trees").
Unistructural [U]	Provides a description for one portion of the code.
Prestructural [P]	Substantially lacks knowledge of programming constructs or is unrelated to the question

2 The Data

The data used in this study are the responses of 14 students to three "explain in plain English" questions. The three questions were answered by the 14 students at Auckland University of Technology, as part of an exam attempted by them and all their class mates at the end of their first programming paper. Approximately 80 students sat the exam, but at the time of the workshop only these 14 students had given approval for their exam responses to be used in this project.

The three "explain in plain English" questions comprised three parts, "A", "B" and "C", of the tenth question in the exam paper. The three parts were preceded by the preamble shown in Figure 1. The code used in parts "A", "B" and "C" are shown in Figures 2, 3 and 4 respectively.

Students were provided with between 5 and 7 lines of space on the exam paper to answer each of the parts.

As part of their preparation for the exam, students had been shown "explain in plain English" questions, but this type of question was not central to their instruction throughout the semester.

3 Straightforward SOLO Classifications

This section reviews some straightforward SOLO classifications of the data, for parts "A", "B" and "C". In this classification exercise, the raters used an extended set of classifications from those in Table 1, with such logical extensions as "Relational Error" and "Multistructural Omission" being identified. The specifics of these codes will be expanded upon in a subsequent section. In addition, the raters provided multiple ratings for responses which they were uncertain how to correctly classify. This rather loose initial coding system produced 18 independently derived codes or code groupings, a considerable expansion on the set of 4 codes in Table 1.

3.1 Question 10A

Of the 13 non-blank student responses analysed, the three raters independently and unanimously agreed on 8 of the 13 (62%), all of which were classified as either "R" or "M".

3.1.1 Relational ("R") Responses

The three raters independently and unanimously agreed to assign "R" to 7 of the 14 student responses, including:

- *This method returns the sum of the numbers in the array.*
- *The purpose of this code is to get a list of numbers, then return a number, is to the total of the list of numbers.*
- *The purpose of this code is to add together all the numbers in the integer array list*
- *It is a method of a double data type and take an parameter from ArrayList. First, it sets the local variable as default store a value of 0. Then it uses a for loop statement which has the conditions if the count "iLoop" is less than the number of integers. The "iLoop" count will plus one after the variable "num" add the ArrayList number to its upon different "iLoop" index. At last, it returns the sum (totals) of the ArrayList numbers.*

While the second and third responses may contain details that are possibly errors ─ "list of numbers", "array list" ─ those responses demonstrate an overall grasp of what the code does. In the fourth response, it is the last sentence that is relational, and everything prior to that sentence is redundant. Without the final sentence, the fourth response would have been coded as multistructural.

3.1.2 Multistructural ("M") Responses

The three raters independently and unanimously agreed to assign "M" to 1 of the 14 student responses:

- *It sets the double num to zero and executes the loop as long as iLoop is less than the length of aNumbers. If it does execute the loop, it adds the value of num to aNumbers and re executes the loop by incrementing. Once the condition is no longer met it returns num.*

For each of these sections of code, explain in plain English what it does.

Note that more marks will be gained by correctly explaining the purpose of the code than by giving a description of what each line does.

Variable names are deliberately not very meaningful so you will have to work out what the code does.

Figure 1: The common preamble of Question 10, which preceded the parts "A", "B" and "C" pieces of code shown in Figures 2, 3, and 4.

```
public double method10A(double[] aNumbers)
{
    double num = 0;

    for(int iLoop = 0; iLoop < iNumbers.length; iLoop++)
    {
          num += aNumbers[iLoop];
    }
    return num;
}
```

Figure 2: The code from "Question 10A" analysed in this paper.

```
public void method10B(int iNum)
{
    for(int iX = 0; iX < iNum; iX++)
    {
          for(int iY = 0; iY < iNum; iY++)
          {
                System.out.print("*");
          }
          System.out.println();
    }
}
```

Figure 3: The code from "Question 10B" analysed in this paper.

3.2 Question 10B

The three raters independently and unanimously agreed to assign a newly derived code "Re" (Relational Error) to one of the 14 student responses.

- *This code prints out a vertical line of "*" Whatever amount or value you give the variable iNum is going to be the number of "*" printed. Example num =5* <student then drew 5 asterisks arranged vertically>

Here the student has correctly identified a portion of the purpose of the code, but has neglected the nested loop structure that prints multiple columns of stars.

There were some consistent patterns of categorisation between pairs of raters for the remaining responses, but the expanded classification scheme appeared to limit the scope for consistent responses across the group.

3.3 Question 10C

Of the 10 non-blank student responses analysed, the three raters did not independently and unanimously agree on any of the responses. As for question 10B there were frequent agreements by two raters, but the frequent choice of multiple codes by the third rater reduced the scope for unanimous agreement.

4 More Difficult SOLO Classifications

In addition to the issues arising from the rather loose coding scheme, disagreements between the three raters frequently arose from three sources:

- Differing opinions on the boundaries between categories ─ hypothetically, if we begin with a clear multistructural response, and progressively delete detail from that response, it is not clear when the modified response becomes unistructural.
- The communication skills of the students, particularly among students who write English as a second language. One rater may interpret a phrase in a student's response as being vacuous, while another rater may see something deeper in the response.

```
public boolean method10C(int[] aiNum, int iValue)
{
    int iX = 0;
    int iY = aiNum.length - 1;
    int iZ,
        iTemp;
    boolean bSwitch = false;

    while (!bSwitch && (iX <= iY))
    {
          iZ = (iX + iY) /2;
          iTemp = aiNum[iZ];

          if ( iValue == iTemp )
          {
                bSwitch = true;
          }
          else if ( iValue < iTemp )
          {
                iY = iZ - 1;
          }
          else
          {
                iX = iZ + 1;
          }
    }
    return bSwitch;
}
```

Figure 4: The code from "Question 10C" analysed in this paper.

• Prior to taking the exam used in this study, and again in the preamble to the exam question, students had been told that responses that summarised the purpose of the code were preferred over responses that gave a line by line description of what each line does. There was therefore the danger that students would make a guess at the purpose of the code, and raters would guess whether the students were guessing.

To try to clarify the definitions of the various categories, participants at the 6th BRACElet workshop augmented the SOLO taxonomy of Table 1 with the extra categories shown in Table 2. The remainder of this section describes the classification of student responses, using this augmented taxonomy, where the three raters disagreed.

Table 2: Augmented SOLO Categories

SOLO category	Description
Relational Error [RE]	Provides a summary of what the code does in terms of the code's purpose, but with some minor error.
Guess Relational [GR]	The response describes a summary of a piece of code, but the summary is so widely divergent from what the code actually does, it appears to be a guess.
Multistructural Omission [MO]	A line by line description is provided for most of the code, but with some detail omitted.
Multistructural Error [ME]	A line by line description is provided for most of the code, but with some minor errors.
Unable to Categorise [X]	The rater could not come to a firm decision.

4.1 Question 10A

In this section we reconsider student responses to Question 10A, using the augmented taxonomy.

4.1.1 Majority Relational Classification

In this subsection we consider student responses to 10A where at least two of the three raters assigned some form of relational category to the response.

Two of the raters categorised the following student response as R, while the third categorised it as RE:

- *Trying to add all the numbers stored in the arrayList that is less than the length of the arrayList. Go through each number from index 0 to the index just before then end of the arrayList*

Two of the three raters categorised the following student response as RE, and the third rater categorised it as being one of RE, GR, or U:

- *The purpose of this method is to count a double number*.

All three raters described the following student response as being possibly RE, but two of the raters added that it could also be GR:

- *It counts the number of index spaces used in the array*.

4.1.2 Majority Multistructural Classification

In this section we consider student responses to 10A where at least two of the three raters assigned some form of multistructural category to the response.

Two of the three raters categorised the following student response as M, and the third rater categorised it as being either U or P:

- *The method increments the loop and returns a number of type double as long as the loop is less than the length of aNumbers*.

Two of the three raters categorised the following student response as M, while the third rater chose MO:

- *The method loops through the aNumbers and adds the aNumber to num and gives the output of that equation num += aNumber[iLoop]*.

4.2 Question 10B

In this section we reconsider student responses to Question 10B, using the augmented taxonomy.

4.2.1 Majority Relational Classification

In this subsection we consider student responses to 10B where at least two of the three raters assigned some form of relational category to the response.

Two of the raters categorised the following student response as RE, while the third categorised it as M:

- *This code takes the number entered in as the parameter and prints out a star and a space the amount of times as the parameter. So for example, for 3 it would be* <the student then wrote 3 asterisks, arranged vertically>.

Two of the raters categorised the following student response as R, while the third categorised it as M:

- *This method takes a user declared int as a parameter it prints out * (astreix)* [sic] *on a single line. The amount of * it prints out is dependant on the int value entered. It always prints out whatever number you enter times itself number of *. For example if entered 3, it would print out 9 * (3×3).* <Remainder of response omitted>

Two of the three raters categorised the following student response as R, while the third rater chose RE:

- *The method prints out a amount of rows which contain the same amount of stars. Like int iNum = 3 two lines of stars that contain two stars*.

Two of the three raters categorised the following student response as GR, while the third rater chose M:

- *The purpose of this method is to print out the maximised number.*

4.2.2 Majority Multistructural Classification

In this section we consider student responses to 10B where at least two of the three raters assigned some form of multistructural category to the response.

Two of the three raters categorised the following student response as M, while the third rater chose MO:

- * *The method takes a parameter iNum*
  * *ix=0 and while ix is less than iNum, increment ix*
  * *iy=0 and while iy is less than iNum, increment iy*
  * *prints a star*

One of the raters categorised the following student response as M, another rater categorised it as ME, and the third rater categorised it as either M or MO:

- *As long as the value in each index is lower than the iNum, keep going through and print out the result.*

One of the raters categorised the following student response as M, another rater categorised it as ME, and the third rater categorised it as R:

- *This method prints a star for as long as the number passed in as a parameter is less than the value of iY and the value of iX.*

One of the raters categorised the following student response as M, another rater categorised it as either M or U, and the third rater categorised it as R:

- *prints * symbol followed by a empty line, for a specified amount of times*

One of the raters categorised the following student response as M, another rater categorised it as ME, and the third rater categorised it as GR:

- *This method loops around and checks for numbers which are less than iY and iX and prints the results with a "*".*

One of the raters categorised the following student response as M, another rater categorised it as ME, and the third rater categorised it as RE:

- *This method prints the line in the terminal window iX*iY with the iX and iY both substituted with their*

*individual values e.g. 1*1. But only when both iX and iY are both less than iNum*

Two of the three raters categorised the following response as ME, and the third rater categorised it as U:

- *The purpose of the code is to get a number, and if the number you gave match these* <indecipherable 1 or 2 words>, *it will print a star sign "*"* <indecipherable word>, if not will be stop

All three raters described the following response as M, with one rater adding that it could also be U:

- *This method prints a "*" if the value in iNum is greater than iX and iY. If the value in iNum is negative, the method prints out nothing.*

4.3 Question 10C

In this section we reconsider student responses to Question 10C, using the augmented taxonomy.

4.3.1 Majority Relational Classification

In this subsection we consider student responses to 10C where at least two of the three raters assigned some form of relational category to the response.

Two of the three raters categorised the following student response as R, while the third chose RE:

- *Checks if any input data in the array is the same as the number inputted in the second parameter*

Two of the three raters categorised the following student response as RE, and the third rater categorised it as being either RE or M:

- *It checks to see whether a specific value is contained at a specific index in the array.*

One rater classified the following response as R, another classified it as RE, and the third rater categorised it as being either RE or GR:

- *This code loops through until it matches value with it corresponding Arraylist index. It is basically a way of sorting.*

4.3.2 Majority Multistructural Classification

In this section we consider student responses to 10C where at least two of the the three raters assigned some form of multistructural category to the response.

All three raters described the following response as M, with one rater adding that it could also be MO:

- *This method checks whether iX is less than or equal to iY and then goes through the loop to check if iValue is equal to iTemp and outputs true and if not goes through the rest of the loop to check whether iValue < iTemp or not and returns the boolean result.*

Two of the three raters categorised the following student response as M, and the third rater categorised it as being one of M, U or P:

- *Returns true if iValue equals iTemp. It runs through the code and if iValue does not equal iTemp it checks to see if iValue is less than iTemp. If it does it minuses 1 from iZ to make it equal to iY*

Two of the three raters categorised the following student response as M, and the third rater categorised it as being one of P, U or X:

- *it assigns value to a number of local variables (i.e. int iX, int iY, int iZ, int iTemp, boolean bSwitch). Then skips over the while loop and returns the value of bSwitch which will always be false. This is because the condition for the loop is !bSwitch which means the code will only execute when bSwitch is true which will never be the case.*

4.3.3 Majority Unistructural Classification

All three raters described the following student response as U with one rater adding that it could also be GR:

- *trying to find Boolean result.*

5 A Formal Analysis of Reliability

In order to assess the level of reliability of the ratings, based on more than visual inspection of results, it was decided to perform a statistical analysis of the interrater reliability of the SOLO classifications made by the three raters.

The statistical test applied was "Kendall's coefficient of concordance (W)" which "is a measure of the agreement among several (p) judges who are assessing a given set of n objects" (Legendre, 2005). In this case the judges were the three 'SOLO raters' assigning categories to the 'objects' namely the individual 'student responses' to each question.

The null hypothesis of the statistical test was:

> H0: The p judges produced independent rankings of the objects (Legendre, 2005).

A statistical consultant recommended that the augmented SOLO classification system be simplified, both to assist in achieving consistent categorisations and to enable a stable statistical scale to be developed. Consequently, the categories from Tables 1 and 2 were converted to a numeric form for statistical analysis, as shown in Table 3. This numeric form preserves the characteristics of the original SOLO scale, as rank data which was inherently based on an ordinal scale of 'cognitive sophistication'. The alphabetic codes in the scale were converted to an arbitrary numeric scale which preserved the ordinality of the data. The category 'blank' was removed from the scale, as it required no judgement on the part of the rater, and thus would inflate any statistical comparison of rater judgements. Finally to remove equivocal classifications by a rater (e.g. M/U/P) the first choice of the rater was used (e.g. M/U/P was assigned to M, P/U/X to P, RE/GR to RE etc.).

A preliminary set of rating statistics for each question is given in Table 4. The degree of concordance among the judges varies from excellent (for part "A") to moderate, as does the degree of significance of the findings. The 'N' for both 'judges' and 'variables' is relatively small for this test, which is a variant on the classical χ^2 test, (cf. Legendre, 2005), but the results at least suggest some level of agreement among judges.

Table 3: Numeric SOLO Codes

Raw Codes	Converted Codes
R	10
RE	15
M	20
MO	23
ME	26
U	30
P	40

Table 4: Kendall's W figures for 3 Judges

Question	Kendall's W	Significance	N (students)
10a	0.962	.001	13
10b	0.553	.063	14
10c	0.605	.060	10

For the statistical analysis used here, three raters was a relatively low number, but not necessarily an issue for a non parametric test, which as our statistical consultant advised tends to be robust by nature. The impact of adding extra raters is illustrated in Table 5, in which a fourth judge was added. The additional judge has improved the significance level of the results in all cases, while slightly changing the level of interrater agreement (slightly down for Q10a & Q10b, and up for 10c). Thus while the size of the effect may have varied from the prior set of data, the presence of an effect appears more certain. Addition of the fourth rater thus served as validation for the initial findings. Encouragingly, the figures do suggest a moderate to strong level of agreement by judges upon their SOLO ratings. We surmise that a larger judging pool would give greater weight to these findings, but at the time of writing further ratings remain to be done.

Table 5: Kendall's W figures for 4 Judges

Question	Kendall's W	Significance	N (students)
10a	.768	.000	13
10b	.500	.017	14
10c	.669	.004	10

6 Conclusion

For the purposes of this paper, the results suggest that SOLO ratings can be applied with moderate levels of consistency. The results also make a good case for a better guidance mechanism to support researchers in applying SOLO ratings to BRACElet data.

Therefore we recommend adoption of the final SOLO categories of Table 6 by those intending to apply SOLO in analysing their programming assessments. This coding scheme omits blank codes, omits undecided categories (i.e. the judge must choose the best fit from this set of categories), and requires the rater to select only one code for each response. We believe that this tighter set of codes and coding protocol will improve the consistency of SOLO ratings undertaken by educators and researchers.

Table 6: The Final SOLO Categories

SOLO category	Description
Relational [R]	Provides a summary of what the code does in terms of the code's purpose. (The "forest")
Relational Error [RE]	Provides a summary of what the code does in terms of the code's purpose, but with some minor error.
Multistructural [M]	A line by line description is provided of all the code. (The individual "trees").
Multistructural Omission [MO]	A line by line description is provided for most of the code, but with some detail omitted.
Multistructural Error [ME]	A line by line description is provided for most of the code, but with some minor errors.
Unistructural [U]	Provides a description for one portion of the code.
Prestructural [P]	Substantially lacks knowledge of programming constructs or is unrelated to the question

The authors further hope that the examples provided here, together with these classification guidelines and the augmented SOLO coding scheme for the programming domain will serve as a practical guidance for computing researchers and educators in applying SOLO ratings to student responses.

We invite others to apply the SOLO coding scheme of Table 6 to their own programming assessments. We hope others will conduct their own evaluation of the results formally using the same statistical tests, so that the robustness of these techniques may be further validated. Should these techniques prove deficient in other contexts, we would welcome suggestions for

additional refinements arising from any issues identified in the field.

7 References

Adelson, B. (1984) When novices surpass experts: The difficulty of a task may increase with expertise. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **10**(3): 483-495.

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

Clear, T., Edwards, J., Lister, R., Simon, B., Thompson, E., & Whalley, J. (2008). The teaching of novice computer programmers: bringing the scholarly-research approach to Australia. In Simon & M. Hamilton (Eds.), *Conferences in Research and Practice in Information Technology* (Vol. 78, pp. 63-68). Wollongong, NSW, Australia: ACS.

Legendre, P. (2005). Species Associations: The Kendall Coefficient of Concordance Revisited. *Journal of Agricultural, Biological and Environmental Statistics, 10*(2), 226-245.

Lister, R., Simon, B., Thompson, E., Whalley, J. L., and Prasad, C. (2006). Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. In *Proceedings of the 11th Annual SIGCSE Conference on innovation and Technology in Computer Science Education.* (Bologna, Italy, June 26 - 28, 2006). ITICSE '06. ACM Press, New York, NY, 118-122.

McKeithen, K., Reitman, J.., Rueter, H., & Hirtle, S. (1981). Knowledge organization and skill differences in computer programmers. *Canadian J. of Psychology*, 13, 307-325.

Philpott, A., Robbins, P., and Whalley, J. (2007) Assessing the Steps on the Road to Relational Thinking. . In *Proceedings of the 20th Annual Conference of the National Advisory Committee on Computing Qualifications*, NACCQ, Nelson, New Zealand, July 8-11.

Thompson, E. (2004). Does the sum of the parts equal the whole? In S. Mann & T. Clear (Eds.), *Proceedings of the 17th annual conference of the National Advisory Committee on Computing Qualifications* (pp. 440-445). Christchurch, New Zealand: National Advisory Committee on Computing Qualifications.

Thompson, E., Whalley, J., Lister, R., Simon, B. (2006) Code Classification as a Learning and Assessment Exercise for Novice Programmers. In *Proceedings of the 19th Annual Conference of the National Advisory Committee on Computing Qualifications*, NACCQ, Wellington, New Zealand, July 7-10. pp. 291-298.

Whalley, J, Lister, R, Thompson, E, Clear, T, Robbins, P, Prasad, C (2006) An Australasian Study of Reading and Comprehension Skills in Novice Programmers, using the Bloom and SOLO Taxonomies. *Australian Computer Science Communications* 52: 243-252.

Whalley, J. (2006). CSEd Research Instrument Design: The Localisation Problem. In S. Mann & N. Bridgeman (Eds.), *Proceedings of The Nineteenth Annual NACCQ Conference* (pp. 307-312). Wellington: NACCQ.

Whalley, J., Clear, T., & Lister, R. (2007). The Many Ways of the BRACElet Project. *Bulletin of Applied Computing and IT. Retrieved June 3, 2007 from http://www.naccq.co.nz/bacit/0501/2007Whalley_BRACELET_Ways.htm, 5*(1).

Wiedenbeck, S., Fix, V. & Scholtz, J. (1993) Characteristics of the mental representations of novice and expert programmers: An empirical study. *International Journal of Man-Machine Studies*, 39, 793-812

8 Acknowledgements

The authors gratefully acknowledge funding via a Special Projects Grant from the ACM SIGCSE for the 6th BRACElet workshop (held on 18 Dec 2007) at Auckland University of Technology. This enabled the participants, whose work was integral to this paper, to meet and discuss the process of SOLO rating. We also thank Stuart Young from the School of Computing and Mathematical Sciences at Auckland University of Technology for statistical advice. The authors thank their other collaborators on the BRACElet project.