Not Seeing the Forest for the Trees: Novice Programmers and the SOLO Taxonomy

Raymond Lister
Faculty of Information Technology
University of Technology, Sydney
Broadway, NSW 2007, Australia
+61 (2) 9514 1850
raymond@it.uts.ac.au

Beth Simon
Computer Science and Engr. Dept., University of California,
San Diego, CA 92093, USA
+1 (858) 534-5419
esimon@cs.ucsd.edu

Errol Thompson
Dept of Information Systems
Massey University, P.O. Box 756,
Wellington, New Zealand
+64 (4) 8015799 x6531
E.L.Thompson@massey.ac.nz

Jacqueline L. Whalley
School of Computer and Information Sciences
Auckland University of Technology
Private Bag 92006, Auckland 1020, New Zealand
+64 (9) 921 9999 x5203
jacqueline.whalley@aut.ac.nz

Christine Prasad
School of Computing and Information Technology
Unitec
Private Bag 92025, Auckland, New Zealand
+64 (9) 815 4321 x6015
cprasad@unitec.ac.nz

ABSTRACT

This paper reports on the authors use of the SOLO taxonomy to describe differences in the way students and educators solve small code reading exercises. SOLO is a general educational taxonomy, and has not previously been applied to the study of how novice programmers manifest their understanding of code. Data was collected in the form of written and think-aloud responses from students (novices) and educators (experts), using exam questions. During analysis, the responses were mapped to the different levels of the SOLO taxonomy. From think-aloud responses, the authors found that educators tended to manifest a SOLO relational response on small reading problems, whereas students tended to manifest a multistructural response. These results are consistent with the literature on the psychology of programming, but the work in this paper extends on these findings by analyzing the design of exam questions.

Categories and Subject Descriptors

K.3 [**Computers & Education**]: Computer & Information Science Education - *Computer Science Education*.

General Terms

Measurement, Experimentation, Human Factors.

Keywords

Novice programmers, CS1, comprehension, SOLO taxonomy.

1. INTRODUCTION

CS1 is the beginning of a long and poorly understood process, where students begin their journey from novice to expert. Novices in every discipline make a similar journey, and there are many studies of the differences between novices and experts in various professional, scientific, and artistic disciplines [4,6]. Clearly, experts know more than novices, but research indicates that experts also organize that knowledge into more sophisticated and flexible forms. This is apparent in the classic studies of chess players [3]. When asked to memorize board positions of several chess pieces, novices tended to remember the position of each piece in isolation, whereas experts organized the information at a more abstract level, the attacking and defensive combinations. When recalling board positions that arise naturally in a chess game, experts outperformed novices, but when faced with unnatural arrangements of chess pieces, the performance of the experts decreased because the abstract patterns that the experts typically used were not present in the unnatural arrangements.

For programming, the differences between novices and experts have also been studied extensively, and tend to confirm the findings from other disciplines [5,10,11,12,13,15]. Expert programmers form abstract representations based upon the purpose of the code whereas novices form concrete representations based on how the code functions. In a study of programming that reflected the earlier chess studies, Adelson [1] showed that, when given typical tasks on well-written code, experts outperformed novices, but when faced with unnatural tasks, novices sometimes outperformed the experts.

1.1 The Leeds Working Group

The ITiCSE 2004 “Leeds” working group studied the reading and tracing skills of novice programmers [9]. Data was collected from 615 students, spread across 12 institutions in 7 countries. The students were asked to answer several multiple choice questions. In this paper we review the Leeds Group findings for one of those questions, Question 2, shown in Figure 1.

Someone answering Leeds Group Question 2 might (i) read through the code, (ii) infer that the code counts the common elements in two sorted arrays, (iii) count manually the number of common elements in the two arrays, which is 3, and (iv) conclude that the first option is correct. On a closer inspection of the code, however, it can be seen that the loop will terminate before the element at position 0 in either array is considered. Therefore, count will contain 2, the second option. In the Leeds study, 65% of students answered correctly, while 21% chose the first option.

Consider the following code fragment:

```
int[] x1 = {1, 2, 4, 7};
int[] x2 = {1, 2, 5, 7};
int i1 = x1.length-1;
int i2 = x2.length-1;
int count = 0;

while ((i1 > 0) && (i2 > 0)) {
     if (x1[i1] == x2[i2]) {
        ++count;
        --i1;
        --i2;
     }
     else {
        if (x1[i1] < x2[i2])
           --i2;
        else
           --i1;
     }
}
```

After the above while loop finishes, “count” contains what value?

a) 3
b) 2
c) 1
d) 0

Figure 1. The Java version of Leeds Group Question 2.

Given that expert programmers form abstract representations of code, programming teachers should not discourage students from forming abstract representations. However, some of the 21% of students who incorrectly chose the second option of Leeds Group Question 2 may have been misled precisely because they formed an abstract representation – that the code counted the number of common elements in the two arrays. Furthermore, the authors have found that when they show Question 2 to academics, they also frequently make that same mistake. The issue then arises as to whether questions like Leeds Group Question 2 are appropriate reading problems for testing students.

In the next section, we review a general taxonomy of learning outcomes, the “Structure of the Observed Learning Outcome” (SOLO) [2]. We relate SOLO to code reading problems. We then use the taxonomy to classify and evaluate the utility of questions like Leeds Group Question 2, and some alternative code reading problems.

2. CODE COMPREHENSION AND SOLO

The SOLO taxonomy [2] describes the responses a student may give to a task. SOLO is a general educational taxonomy, and has not previously been applied to the study of how novices manifest their understanding of small code reading problems. In this section, we introduce the taxonomy, and offer our interpretation of how the taxonomy applies to novices solving small code comprehension problems.

With multiple choice questions, it is not possible to assign a student’s response to a SOLO level when only provided with the option chosen by the student. Knowledge is required of how the student chose that option. The Leeds Group reported upon transcripts from 35 students who were asked to “think out loud” as they attempted the multiple choice questions [7, 9]. In this section, we will use those reports on the Leeds Group transcripts to illustrate the SOLO taxonomy.

The SOLO taxonomy describes five levels of student responses. These five levels are described in the following five subsections.

2.1 Prestructural SOLO Response

This is the least sophisticated type of response a student may give. In terms of reading and understanding a small piece of code, a student who gives a prestructural response is manifesting either a significant misconception of programming, or is using a preconception that is irrelevant to programming. For example, the Leeds group described a student who confused a position in an array and the contents of that position (i.e. a misconception). Novice misconceptions of programming constructs have been studied extensively [12, 13], and are not the focus of this paper.

2.2 Unistructural SOLO Response

This is a response where the student manifests a correct grasp of some but not all aspects of the problem. When a student has a partial understanding, the student makes what the Leeds Working group called an “educated guess”. In their transcripts, the Leeds Group found evidence for guessing in 10% of all student answers to questions, with approximately half of those being educated guesses. However, guessing rates varied widely across all the multiple choice questions, with the transcripts for one question indicating that 30% of answers were guesses. The Leeds group did not indicate what portion of that 30% were educated guesses. Unistructural responses are not the focus of this paper.

2.3 Multistructural SOLO Response

This is a response where the student manifests an understanding of all parts of the problem, but does not manifest an awareness of the relationships between these parts – the student fails to see the forest for the trees.

For example, for Leeds Group Question 2, a student may hand execute the code and arrive at a final value for variable “count”, but the student may not manifest an understanding of what that code does. The Leeds Group noted a strong tendency for students to hand execute the code of Question 2.

Note that a multistructural response may be either correct or incorrect. A student may make an error while hand executing the code, never-the-less the technique is multistructural.

The multistructural level, along with the next level, is at the focus of this paper, and will be discussed again in a later subsection.

2.4 Relational SOLO Response

This is a response where the student integrates the parts of the problem into a coherent structure, and uses that structure to solve the task – the student sees the forest. For example, after perusing the code in Leeds Working Group Question 2, a student may infer that the code counts the number of common elements in the two arrays, and calculate their answer without hand executing the code to completion. However, the Leeds Group found that few students manifested such an understanding of what the code computed

Note that a relational response may be either correct or incorrect. A student may not notice that the loop in Leeds Working Group Question 2 terminates before the first position in either array is inspected; never-the-less the approach is relational.

Someone answering Question 2 may begin with a multistructural response, and hand execute one or more iterations of the loop, then realize what the code is computing, and jump to the answer without completing the hand execution. Such a response is relational. In fact, we would expect most students who answer relationally to begin with such an approach.

In their analysis of their transcripts, the Leeds Group reported that very few students manifested (by our definition in this paper) a relational response, even among the top quartile students.

The relational and multistructural levels of the SOLO taxonomy are the focus of this paper.

2.5 Extended Abstract SOLO Response

In this highest SOLO level, the student response goes beyond the immediate problem to be solved, and links the problem to a broader context. For example, a possible extended abstract response to Question 2 may be a comment that the code will only work for arrays that are sorted. While interesting, extended abstract responses are not the focus of this paper.

2.6 Multistructural vs. Relational Responses

There is an extensive literature [5,10,11,12,13,15] indicating that expert programmers integrate the parts of a program into a coherent structure – the expert programmer sees the forest. If the aim of a teacher is to test novice programmers on their ability to form such a coherent structure, then the Leeds Group Question 2 is not a good question. The Leeds Group transcripts for this question almost exclusively manifest a multistructural response.

However, it does not follow from the above paragraph that all reading exercises need to elicit a relational response. If the aim of a teacher is to ascertain whether a student has a correct understanding of how while loops work (prestructural), or whether a student is disciplined enough to hand execute a piece of code (multistructural), then Question 2 is appropriate.

We are not advocating that students should be taught and tested in a chronological sequence reflecting the ascending levels of the SOLO taxonomy. The SOLO taxonomy is not a model of cognitive development. We merely advocate a mix of assessment exercises. At the very least, a teacher needs to be clear in their own mind as to the objective of any assessment exercise.

3. EXPERIMENT 1: EXPERT READING

The Leeds Group only studied novice programmers. We believe that experimental results for novice programmers are not best evaluated in isolation. Instead, results for novices should be compared to the results on the same task for more experienced programmers.

We asked eight computer science educators to "think out loud" while solving Leeds Group Question 2. Among these educators, a relational response was considerably more evident than it was among the students, but a relational response was not universal. Five of the eight educators manifested a relational response. Among that five, only three arrived at the correct answer, with the other two missing the premature end to the loop. These results for educators support our claim that Question 2 is not a good question when the aim is to elicit a correct, relational response.

3.1 Qualitative Analysis

It is interesting to examine the transcripts of the five educators who manifested a relational response. On their way to developing a single coherent structure for the code, these educators first articulated abstractions for parts of the code.

At the most basic level, the educators tended to articulate an abstraction of the loop structure:

> "going backward through these arrays."
>
> "we're starting from the high end."

In contrast, students, generally articulated nothing more than the presence of a loop, and sometimes also a literal statement about the terminating condition [7].

Several educators abstracted portions of the body of the loop, for example:

> "I'm always decrementing the index of the bigger one."

Often, utterances like the above occur after a detailed examination of the loop, or after hand executing one or two iterations.

There were also some extended abstract responses, for example:

> "It looks like the code is assuming the arrays are in sorted order from smallest to largest".

It is apparent that, even when initially hand executing the code, most educators are actively seeking to abstract beyond the concrete code. In contrast, most novices did not seek to abstract.

4. EXPERIMENT 2: CODE EXPLANATION

Given that the Leeds Group Question 2 is not suited to eliciting a relational response, then what sort of code reading question could be used for that purpose? There are probably many possibilities. In this section, we explore one possibility, the "explain in plain English" style of question. An example of such a question is given in Figure 2.

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;
    }
```

Figure 2. An "explain in plain English" question.

The "explain in plain English" question from Figure 2 was given to 108 students as part of their final exam in their first semester programming course. This question was given as Question 10 in the BRACElet study [14]. Students were required to provide a written answer. The students were from two institutions. Approximately 25% were non-native English speakers.

The student responses to this question were categorized according to the SOLO taxonomy by three of the authors. To be classified as a relational response, a student had to manifest an understanding that the code checks whether the array is sorted. One third of students provided such a relational response.

We also categorized responses from eight educators. In contrast to the majority of students, seven of the eight educators manifested a relational response.

Half of the students provided a multistructural response. In such a response, a student describes how the code works, frequently line-by-line, without indicating that the code checks whether the array is sorted.

The instruction "explain in plain English" is ambiguous – intentionally so, for reasons discussed in this paragraph – and so it might be argued that a student could have provided a relational response but instead elected to provide a multistructural response. If a student had the potential to provide both relational and multistructural responses, then why did the student not provide both? In fact, many students did supply both, and those students were categorized as having provided a relational response. If many of the students who only gave a multistructural response did so because they understood that to be the more appropriate response for the instruction "explain in plain English", then why did seven out of eight educators provide a relational response? We noted earlier that when answering Leeds Group Question 2, most of the educators actively sought to abstract beyond the concrete code, whereas students did not. For someone who actively seeks to abstract from concrete code, it is natural to provide a relational response to the instruction "explain in plain English". For someone who does not seek to abstract, the focus of attention is on the individual lines of code as separate entities, not on the relationships between those lines of code. We believe that if a student is to be adept at writing code, and debugging that code, then the relational response needs to become the natural and most obvious way of explaining what code does – just as the relational response appears to be the natural and obvious mode of explanation for seven of the eight educators.

4.1 SOLO Response by Quartile

As another part of their exam, the 108 students who answered the "explain in plain English" question from Figure 2 also answered nine multiple choice questions. These nine questions are similar to the Leeds Group questions (two of the nine are Leeds Group questions). A complete description of the nine questions is available elsewhere [14, 16].

Figure 3 shows the distribution of SOLO response for the "explain in plain English" question, broken into four quartiles, according to how well the students did on the nine multiple choice questions (i.e. there are 27 students in each quartile). The first quartile is the top quartile. Approximately one half of students in the top two quartiles manifested a relational response to the "explain in plain English" question, but multistructural responses dominated in the lower two quartiles.

Figure 3 may illustrate why the weaker students in many CS1 classes struggle to write code. If we assume student responses to the "explain in plain English" question are a reasonably consistent reflection of how the students reason about code, then it is apparent in Figure 3 that many of the weaker students do not naturally abstract from concrete code to ascertain the purpose of that code.

5. RELATIONAL DEBUGGING

Some readers may consider Leeds Question 2 to be a "trick" question, because the code does not compare the lowest two positions of the arrays. On the other hand, some may argue that teachers should test students' ability to recognize such "tricks" – to test students' ability to debug incorrectly functioning code. We can alter Question 2 so that it is a debugging question that requires a relational response, as illustrated in Figure 4.

On the other hand, part of good relational thinking is the identification of boundary conditions. In that context, Leeds Question 2 as it is currently expressed may reward abstract thinking in appropriately prepared students. Part of the preparation may be a warning to students that they need to identify and pay attention to boundary conditions in the exam questions.

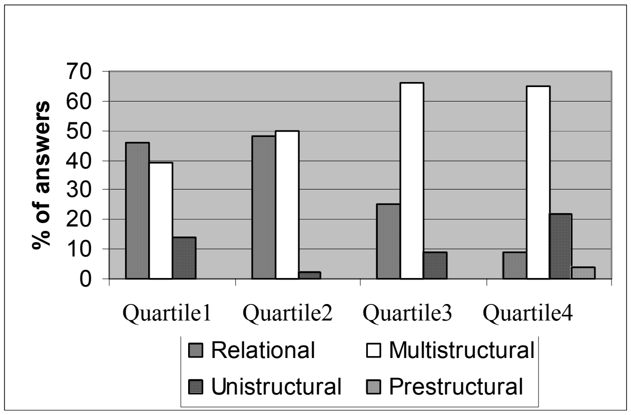

Figure 3. Performance on BRACElet Q10 by performance quartile, for the combined two institutions. (N=108) Quartile 1 is the top performing quartile.

Consider the following code fragment:

```
<code as given in Figure 1>
```

The above code is meant to count the number of equal numbers in the two arrays. There are three equal numbers in the two arrays, but when the above code finishes, the variable "count" contains 2. The bug is due to an error in one line or in two lines. Nominate the buggy line(s), explain what is wrong, and provide a corrected version of the lines(s).

Figure 4. Leeds Question 2, rewritten as a debugging question.

6. TEACHING ISSUES

This paper has focused upon testing students on their capacity to reason abstractly about code, and not teaching techniques for developing that thinking. Space limitations prevent a discussion of that vital issue, so we refer the reader to other sources [8, 10, 11, 12, 13]. We recommend the recent work on roles of variables [8] as a particularly promising approach.

7. CONCLUSION

This paper demonstrates that the SOLO taxonomy is a useful organizing framework for comparing work relevant to the testing of novice programmers via reading problems. Much of the work in the 1980s [12,13] focused on novice preconceptions and misconceptions of programming constructs, while the recent Leeds Working Group [9] focused upon the ability of students to reliably hand execute code. These are aspects of programming on which teachers should test their students. However, to focus solely upon those aspects of programming is to focus upon the three lower levels of the SOLO taxonomy; the prestructural, unistructural, and multistructural levels. Teachers also need to test their students in a way that is intended to elicit a relational response. In providing such a response, a student manifests an ability to read several lines of code and integrate them into a coherent structure – to see the forest, not just the trees. The literature on the psychology of programming [1, 5, 10, 11, 15] indicates that novices need to develop such a skill if they are to develop as programmers. We do not advocate the exclusive use of questions designed to elicit a relational response. Instead, we merely advocate that teachers use a suite of assessment strategies, and test students at all levels of the SOLO taxonomy.

In our view, students who cannot read a short piece of code and describe it in relational terms are not intellectually well equipped to write similar code. We are not advocating that students must first be taught to read code, and examined on their ability to manifest a relational response, before they ever write a line of code, but we do advocate a mix of reading and writing tasks.

The Leeds Group paper [9] ends with a proposition for a follow-on experiment – that students be given both reading tasks and writing tasks, to see if student performance on reading and writing correlate. We offer the following refinement to that experiment. The reading tasks should also group students on whether they tend to respond multistructurally or relationally. The reading performance of each of those groups should be correlated with the writing tasks. We suspect that the correlation for the students who tend to respond relationally will be higher than for the students who tend to respond multistructurally.

8. ACKNOWLEDGMENTS

The authors thank the members of the Leeds Working Group and BRACElet project. We also thank Gordon Grimsey for assistance in data collection, and Cornelia Box for assistance in writing.

9. REFERENCES

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

[2] 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.

[3] Chase, W. C., & Simon, H. A. Perception in chess. *Cognitive Psychology*, 4 (1973), 55-81.

[4] Chi, M. T. H., Glaser, R. & Farr, M. J. (Eds.) *The nature of expertise*. Hillsdale, NJ, Lawrence Erlbaum Associates, 1988.

[5] Corritore, C. & Wiedenbeck, S. What Do Novices Learn During Program Comprehension? *Int. J. of Human-Computer Interaction*, 3, 2 (1991), 199-222.

[6] Ericsson K, and Smith, J. (Eds) *Toward a General Theory of Expertise : Prospects and Limits.* Cambridge University Press,England, 1991.

[7] Fitzgerald, S., Simon, B., Thomas, L. Strategies that Students Use to Trace Code: An Analysis Based in Grounded Theory. In *Proceedings of the 1st International Workshop on Computing Education Research* (*ICER2005*) (Seattle WA, USA, October 1-2, 2005), 69-80.

[8] Kuittinen, M, and Sajaniemi, J. Teaching Roles of Variables in Elementary Programming Courses. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (*ITiCSE*'04) (Leeds, UK, June 28 - 30, 2004), 57–61.

[9] 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. A Multi-National Study of Reading and Tracing Skills in Novice Programmers. *SIGSCE Bulletin*, 36, 4 (2004), 119-150.

[10] Rist, R. Learning to Program: Schema Creation, Application, and Evaluation. In Fincher, S and Petre, M. (Eds) *Computer Science Education Research*. Routledge Falmer, 2004.

[11] Robins, A., Rountree, J. & Rountree, N. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13, 2 (2003), 137 - 172.

[12] Soloway, E. and Iyengar, S., Eds *Empirical Studies of Programmers*. Ablex, NJ, USA,1986.

[13] Soloway, E. and and Spohrer, J. (Eds) *Studying the Novice Programmer*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1989.

[14] 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. In *Proceedings of the Eighth Australasian Computing Education Conference* (*ACE2006*) (Hobart, Australia, January 16-19, 2006), 243-252. http://crpit.com/Vol52.html [April 2006]

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

[16] http://online.aut.ac.nz/Bracelet/shared.nsf