Further Evidence of a Relationship between Explaining, Tracing and Writing Skills in Introductory Programming

Raymond Lister
Faculty of Engineering and Information Technology
University of Technology, Sydney
NSW 2007, Australia
raymond@it.uts.edu.au

Colin Fidge and Donna Teague
Faculty of Science and Technology,
Queensland University of Technology,
Brisbane, Australia
{c.fidge, d.teague}@qut.edu.au

ABSTRACT
This paper reports on a replication of earlier studies into a possible hierarchy of programming skills. In this study, the students from whom data was collected were at a university that had not provided data for earlier studies. Also, the students were taught the programming language "Python", which had not been used in earlier studies. Thus this study serves as a test of whether the findings in the earlier studies were specific to certain institutions, student cohorts, and programming languages. Also, we used a non–parametric approach to the analysis, rather than the linear approach of earlier studies. Our results are consistent with the earlier studies. We found that students who cannot trace code usually cannot explain code, and also that students who tend to perform reasonably well at code writing tasks have also usually acquired the ability to both trace code and explain code.

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, tracing, comprehension, taxonomy.

1. INTRODUCTION
Some recent studies of novice programmers have focused upon the concept of a hierarchy of programming skills. Toward the bottom of the hierarchy is knowledge of basic programming constructs (e.g. what an "if" statement does). At the top of the hierarchy is the ability to write non–trivial, correct code using those programming constructs.

Whalley et al. (2006) investigated an intermediate skill in the hierarchy. They reported on a study in which students in an end–of–semester exam were given a question that began "*In plain English, explain what the following segment of Java code does*". It was found that some students responded with a correct, line–by–line description of the code while other students responded with a correct summary of the overall computation performed by the code (e.g. "*the code checks to see if the elements in the array are sorted*", which is a reasonable summary of what the code in that question did). Furthermore, the better a student performed on other programming–related tasks in that same exam, the more likely the student was to provide a correct summary of the overall computation performed by the code in the "explain in plain English" question. In a follow up to that study, Lister et al. (2006) found that when the same "explain in plain English" question was given to academics, they almost always offered a summary of the overall computation performed by the code, not a line–by–line description. The authors of that study concluded that the ability to provide such a summary of a piece of code ─ to "*see the forest and not just the trees*" ─ is an intermediate skill.

Philpott, Robbins and Whalley (2007) found that students who could only trace code correctly less than 50% of the time could not usually explain similar code. To express their result in terms of a hierarchy of programming skills, the ability to trace code is lower in the hierarchy than the abilty to explain code.

Sheard et al. (2008) found that the ability of students to explain code correlated positively with their ability to write code.

Lopez et al. (2008) analyzed student responses to an end–of–first–semester exam. A stepwise regression was used to construct a hierarchical path diagram. At the bottom of their hierarchy were exam questions that required basic knowledge. Highest in the intermediate levels of the path diagram were the ability to provide a summary for "explain in English" questions and the ability to trace iterative code. Figure 1 shows the higher portion of the Lopez et al. hierarchy. The points students earned on tracing iterative code accounted for only 15% of the variance in the points earned on the writing question (i.e. $R^2 = 0.15$). Also, the points students earned on "Explain" questions accounted for only 7% of the variance on points earned on the writing question. However, in combination, the tracing and "Explain" questions accounted for 46% of the variance in the writing question (as indicated in Figure 1 by $R^2 = 0.46$ within the box headed "Writing").

In this paper, we report on our own study of the relationships between tracing iterative code, explaining code, and writing code. The data we used was collected in an exam that students took at the end of a one–semester introductory course on programming. These students attended the university of the second and third authors. Students from this university had not participated in any of the earlier studies summarized above. Also, these students were taught the programming language "Python", a language not

ITiCSE'09, July 6–9, 2009, Paris, France.

used in any of the earlier studies. Thus this study serves as a test of whether the findings in the earlier studies were specific to certain institutions, student cohorts, and programming languages.

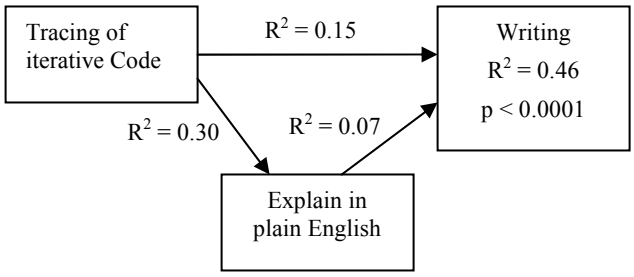

Figure 1: A portion of the stepwise regression model from Lopez et al. (2008).

2. STUDENT SCREENING

This study focuses upon the programming skills located high in the hierarchy of Lopez et al. (i.e. the skills in Figure 1). Therefore, we first need to screen the data, to eliminate students who manifest weaknesses in skills lower in the hierarchy of Lopez et al. In the exam that the students took, three questions tested the students at the lower levels of the hierarchy. Two of those questions required students to trace simple non–iterative code. Both questions contained "if" statements, one with nested "if" statements and a disjunctive "if" condition. The third question required students to describe an algorithm, in English, for finding the smallest and largest number, when the numbers are written on marbles in a bag. A detailed analysis of student responses to this third question has appeared elsewhere (Fidge and Teague, 2009).

There were 146 students who answered all three of these questions well. That assessment of these 146 students was done as a routine part of grading the whole exam, and was done prior to our analysis presented in this paper. Our analysis is restricted to these 146 students.

3. THE INSTRUMENT

The remainder of the exam paper included (1) three questions where students traced iterative code; (2) four "Explain in plain English" questions; and (3) a code writing question. (The exam also included a second code writing question that is not analyzed in this paper.) To ensure consistency of grading, each question on the exam was graded by one person. All of the exam questions analyzed in this paper were written by the second author of this paper. This section describes those exam questions.

3.1 Tracing Iterative Code

Each of these tracing questions began with the following instruction: *What is printed when the following Python code segment is executed?* The three pieces of code were as follows:

```
losses = [1, 25, 4, 9, 16]
net = 100
for loss in losses:
    net = net - loss
print net
```

... the correct answer, which is 45, was provided by 86% of the 146 students.

```
def exceeds(nums1, nums2): # parameters
 # are two equally long lists of numbers
   last = -1
   for index in range(len(nums1)):
       if nums1[index] < nums2[index]:
           last = index
   return last
print exceeds([1, 5, 2, 4, 2],
              [4, 2, 4, 7, 1])
```

... the correct answer, which is 3, was provided by 64% of the 146 students. The function returns the largest index where the number in list `nums2` exceeds that in `nums1` (or –1 if this is never the case).

```
def residual(num, den):
   if num < den:
       return num
   else:
       return residual(num - den, den)
print residual(8, 3)
```

... the correct answer, which is 2, was provided by 92% of the 146 students. The function returns the remainder of an integer division.

Table 1 provides the distribution of scores that the 146 students manifested on these three tracing questions.

Number of Tracing Tasks Correct	Number of Students	Percentage of Students
3	86	59%
2	38	26%
1	18	12%
0	4	3%

Table 1: Student scores on the tracing questions (n=146).

3.2 Explain in Plain English Questions

All four of these questions began with the following instruction: *Explain, in plain English, the purpose of the following Python function*. (*Do not say how the code works. Instead say what the function would be used for.*) The four functions given were:

```
def mystery(names): # names is assumed to
                    # be a list of strings
   return max(map(len, names))
```

A suitable answer, such as "*It returns the length of the longest name (string) in the list*", was provided by 61% of the 146 students.

```
def enigma(nums): # nums is assumed to
                  # be a list of numbers
   for index in range(len(nums) - 1):
       if nums[index] > nums[index + 1]:
           return False
   return True
```

A suitable answer, such as "*It checks to see if the list is sorted*", was provided by 71% of the 146 students. This is the same question (rewritten into Python) as used in some earlier studies (Whalley et al., 2006; Sheard et al., 2008).

-
```
def puzzle(nums): # nums is assumed to
                  # be a list of numbers
    answer = []
    for num in nums:
        if num >= 0:
            answer = answer + [num]
    return answer
```

A suitable answer, such as "*It returns the given list of numbers with all the negative numbers removed*", was provided by 75% of the 146 students.

-
```
def conundrum(text, bound): # parameters
# are a string and a natural number
    if len(text) >= bound:
        return text[0:bound]
    else:
        return text +
            (' ' * (bound - len(text)))
```

A suitable answer, such as "*It makes the given text 'bound' characters long, either by truncating it if it's too long, or by padding it on the right with spaces if it's too short*", was provided by 88% of the 146 students.

The assessment of whether students gave a suitable answer to any of these "Explain" questions was made by one person, as a routine part of grading the exam, prior to the analysis presented in this paper. Table 2 provides the distribution of scores that the 146 students manifested on these four "Explain" questions.

Number of Good Explanations	Number of Students	Percentage of Students
4	56	38%
3	47	32%
2	27	18%
1	11	8%
0	5	3%

Table 2: Student scores on the "Explain" questions (n=146).

3.3 The Code Writing Question

The code writing question required students to search for elements that occurred exactly twice in an array. If a student used certain pre–defined Python functions (which were pointed out in a "hint"), then a solution could be written using a single loop. Without these functions, two nested loops are required. The grading guidelines for this question specified that "*Any working implementation is acceptable — efficiency is not a* [grading] *criterion ... Ignore trivial syntax errors*." Just over half (53%) of the students provided a "good" answer, an answer which was either a correct answer or an answer containing only minor, easily corrected bugs (e.g. the omission of a variable initialization).

4. PAIR WISE RELATIONSHIPS

This section examines the pair wise relationships between student performance on tracing, explaining and writing.

4.1 Tracing and Writing

Table 3 shows the percentage of students who gave what was deemed to be a good answer to the writing task, broken down according to the number of tracing tasks answered correctly. For example, of the 86 students who answered all three tracing tasks correctly, 65% answered the writing task well. Most students who scored less than 50% on tracing did poorly on writing.

Tracing Tasks Correct	No. of Students	Percentage of Good Answers to Writing Question
3	86	65%
2	38	53%
1	18	11%
0	4	0%

Table 3: The percentage of students who gave a good answer to the writing task, for each score on the tracing tasks.

4.2 Explaining and Writing

Table 4 shows the percentage of students who provided a good answer to the writing task, broken down according to the number of good answers to the "Explain in Plain English" questions. Most students who scored 50% or less on the explanation tasks did poorly on the writing task.

Number of Good Explanations	Number of Students	Percentage of Good Answers to Writing Question
4	56	70%
3	47	55%
2	27	33%
1	11	27%
0	5	20%

Table 4: The percentage of students who gave a good answer to the writing task, for each score on the "Explain" tasks.

4.3 Tracing and Explaining

Table 5 shows the percentage of students who gave a good explanation for each of the "Explain" tasks, broken down according to their performance on the tracing tasks. For each "Explain" problem, a chi–square test compares the percentage of students who answered 2 or 3 of the tracing tasks correctly (i.e. the percentage above the respective χ^2 value) against the students who answered only 1 of the tracing tasks correctly (i.e. the percentage below the χ^2). A $\chi^2 \geq 4$ is significant at $p \leq 0.05$ (since the degree of freedom is 1), so only "puzzle" does not manifest a statistically significant relationship.

In Table 5, our decision to break the tracing scores into 2 "bins" was based upon the earlier work of Philpott, Robbins and Whalley (2007), who found that students who could only trace code correctly less than 50% of the time could not usually explain

similar code. For three of our four explanation tasks, our results are consistent with their results.

Tracing Tasks Correct	mystery	enigma	puzzle	conundrum
2 or 3	68%	78%	77%	93%
	$\chi2 = 10.7$	$\chi2 = 15.9$	$\chi2 = 2.0$	$\chi2 = 15.7$
1	28%	33%	61%	61%

Table 5: Percentage of students giving good explanations on the four "Explain" tasks, broken down according to the number of tracing tasks answered correctly (n=146).

Table 6 gives a breakdown of the number of students with various combinations of scores on the tracing and "Explain" tasks. In this table, the 9 students who scored zero on either the tracing tasks or the "Explain" tasks were omitted, hence n < 146.

Tracing Score	Number of Good Explanations on the Four Explain in Plain English Questions	
	1 or 2	3 or 4
2 or 3	24	98
1	10	5

Table 6: The number of students with various combinations of scores on the tracing and "Explain" tasks (n=137).

5. TRACING, EXPLAINING & WRITING

Table 7 summarizes the relationships between writing and the combination of tracing and explaining, as manifested by our students on the writing task. For example, the lower left cell of Table 7 (i.e. the cell containing "10%") shows the data for students who scored a tracing score of 1 and an explanation score of either 1 or 2. From the equivalent cell in Table 6, we see that 10 students recorded such a combination of scores. Table 7 shows that 1 of those 10 students (i.e. 10%) gave a good answer to the writing question.

If we move vertically from that lower left cell of Table 7, to the upper left cell, then the score on explanation remains as 1 or 2, but the tracing score increases to 2 or 3. From the equivalent cell in Table 6, we see that 24 students recorded such a combination of scores. Table 7 shows that 11 of those 24 students (i.e. 46%) scored well on the writing task. The $\chi2$ value of 4.0 between these upper and lower left cells of Table 7 indicates that the performance difference on writing between these two cells is statistically significant. (For all χ^2 values listed in Table 7, the degrees of freedom (df) = 1, so $\chi^2 \geq 4$ is significant at $p \leq 0.05$.)

For the two upper cells in Table 7, where the tracing score is 2 or 3, the associated χ^2 value of 3.8 approaches the traditional $p < 0.05$ threshold for statistical significance, and is well within the common $p < 0.1$ threshold, so we will regard the difference between these two cells as statistically significant.

In general, the χ^2 values between vertically or horizontally separated cells in Table 7 are not well above the 4.0 threshold for statistical significance, but that is consistent with the findings of Lopez et al. – skill in tracing alone is a factor in manifesting skill at code writing, but a stronger factor is combined skill in tracing and explaining.

Tracing Score	Number of Good Explanations on the Four Explain in Plain English Questions		
	1 or 2		3 or 4
2 or 3	46%	← $\chi2 = 3.8$ →	67%
	$\chi2 = 4.0$ ↕		$\chi2 = 9.4$ ↕
1	10%	← $\chi2 = 0.5$ →	0%

Table 7: Percentage of good answers to the writing task for combined scores on tracing and "Explain" tasks (n=137).

Some caution should be exercised in considering the three chi–square values in the middle and lowest rows of Table 7. A common statistical rule is that all cells in a 2 by 2 chi–square contingency table should contain a value (i.e. a frequency) greater than or equal to 5 (some argue that these frequencies should be at least 10). From an inspection of Tables 6 and 7, this is clearly not the case for these three chi–square values. This rule does hold for the chi–square value in the upper row, where the smallest frequency in the associated contingency table is 11.

Given the need to have reasonable values in the contingency table of a chi–square analysis, we could only place the scores on explanation into two "bins" (rather than the maximum possible four bins, in which each of the bins represented a specific total point score on the four explanation questions). Given two bins, the natural choice was a bin for ≤ 50% performance (i.e. 1 or 2 good explanations) and a bin for >50% performance (i.e. 3 or 4 good explanations).

6. DISCUSSION AND CONCLUSION

Our results are consistent with the findings of earlier studies. That is, we have also found that there are statistically significant relationships between tracing code, explaining code, and writing code. As the Lopez et al. study indicated, while skill in tracing alone is a factor in manifesting skill at code writing, it is the combination of tracing and explaining that is most closely associated with skill in writing

We used a non–parametric statistical analysis (i.e. the chi–square test) whereas Lopez et al. and others looked for linear relationships in the data. We feel that the relationships between tracing, explaining, and writing need not be linear, so a non–parametric approach is more appropriate.

Our data does not support the idea of a strict hierarchy; where the ability to trace iterative code would precede any ability to explain code, and where the development of both tracing and explaining would precede any ability to write code. Table 6 shows that five students did better on explaining code than tracing code (although we wonder whether these students made shrewd guesses), and Table 7 shows that one student scored well on writing despite performing poorly on both tracing and explaining. Rather than

arguing for a strict hierarchy, we merely argue that, for most students, some minimal competence at tracing code precedes some minimal competence at explaining code, and after a student has minimal competence in both of these skills, the two skills reinforce each other and develop in parallel.

Similarly, we merely argue that that some minimal competence at both tracing and explaining precedes some minimal competence at systematically writing code. After a student has minimal competence in all these skills, the skills then reinforce each other and develop in parallel. Having written a small piece of code, it is important that a student be able to pause, inspect the code, and see that the code actually does what the student intended. Also, when programmers struggle to see a bug by such an inspection, but the output from running the code shows that a bug is present, they may resort to tracing the code. Therefore, writing code provides many opportunities to improve tracing and explanation skills, which in turn help to improve writing skills.

It is our view that novices only begin to improve their code writing ability via extensive practice in code writing when their tracing and explaining skills are strong enough to support a systematic approach to code writing. Students who are weak at tracing and/or explaining cannot write systematically. Instead, in a desperate effort to pass their course, they will experiment haphazardly with their code − a behavior often reported by computing educators. Until students have acquired minimal competence in tracing and explaining, it may be counter productive to have them write a great deal of code. We do not advocate that students be forced to demonstrate minimal competence in tracing and explaining before they write any code. (Indeed, we suspect that such an approach would lead to motivational problems in students.) However, we do advocate that educators pay greater attention to tracing and explaining in the very early stages of introductory programming courses, and that educators discuss with their students how these skills are used to avoid a haphazard approach to writing code.

Computing educators today perpetuate the pedagogical practices from their own student days. Two of the authors of this paper (and many of the readers?) were novices when programming was done with punched cards and overnight batch runs. In such an environment, teachers did not need to explicitly encourage students to think carefully about their code before attempting their next compile–and–run. If students did otherwise, a careless error could waste a whole day. For today's novices, however, who are learning in an era where the next compile–and–run is only a mouse–click away, it is tantalizingly easy but ultimately futile for a novice to pursue a strategy of trying to get the computer to do the thinking for them. Today's educator needs to place greater explicit pedagogical emphasis on the importance of, and the practice of, the tracing and explaining skills that lead to systematic code writing.

While the recent work on a hierarchy of programming skills is a novel empirical approach to the study of novice programmers (particularly in its use of data collected in the "natural setting" of end–of–semester exams), a belief in the importance of tracing skills, and skills like explanation, are present in quite old literature on novice programmers. For example, Perkins et al. (1989) discussed the importance and role of tracing as a debugging skill, and Soloway (1986) advocated the explicit teaching of "mental simulation" to students.

It is easy to put "explain in plain English" questions into an exam and verify that students can see "the forest" and not just "the trees", but how does one teach a novice who does not yet see "the forest"? The authors of this paper believe that this is an area ripe for pedagogical innovation. We believe that one promising technique for helping students to see "the forest" is the concept of the roles of variables (Kuittinen and Sajaniemi, 2004).

ACKNOWLEDGEMENT

Raymond Lister's work on this project was funded by the Australian Learning and Teaching Council, under their fellowship program.

7. REFERENCES

[1] Fidge, C. and Teague, D. (2009) *Losing their Marbles: Syntax–Free Programming for Assessing Problem–Solving Skills.* 11th Australasian Computing Education Conference, Wellington, New Zealand, 75–82.

[2] Kuittinen, M, and Sajaniemi, J. (2004) *Teaching Roles of Variables in Elementary Programming Courses*. 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, Leeds, UK, 57–61.

[3] 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*. 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, Bologna, Italy, 118–122.

[4] Lopez, M., Whalley, J., Robbins, P., and Lister, R. 2008. *Relationships between reading, tracing and writing skills in introductory programming*. 4th International Workshop on Computing Education Research, Sydney, Australia, 101–112.

[5] Perkins, D. and Martin, F. (1986) Fragile Knowledge and Neglected Strategies in Novice Programmers. In Soloway, E. and and Spohrer, J, Eds (1989), Studying the Novice Programmer. Lawrence Erlbaum Associates, Hillsdale, NJ, 1989. pp. 213–229.

[6] Philpott, A, Robbins, P., and Whalley, J. (2007): *Accessing The Steps on the Road to Relational Thinking*. 20th Annual Conference of the National Advisory Committee on Computing Qualifications, Nelson, New Zealand, 286.

[7] Sheard, J., Carbone, A., Lister, R., Simon, B., Thompson, E., and Whalley, J. L. (2008). *Going SOLO to assess novice programmers*. 13th Annual Conference on Innovation and Technology in Computer Science Education, Madrid, Spain, 209–213.

[8] Soloway, E. (1986). Learning to program = Learning to construct mechanisms and explanations. Communications of the ACM, 29(9). pp. 850–858.

[9] Whalley, J. L., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar, P. K. A., & Prasad, C. (2006). *An Australasian Study of Reading and Comprehension Skills in Novice Programmers, using the Bloom and SOLO Taxonomies*. 8th Australasian Computing Education Conference, Hobart, Australia. 243–252.