A Closer Look at Tracing, Explaining and Code Writing Skills in the Novice Programmer

Anne Venables and Grace Tan
School of Engineering and Science,
Victoria University,
Melbourne, Australia
{Anne.Venables,Grace.Tan}@vu.edu.au

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

ABSTRACT
The way in which novice programmers learn to write code is of considerable interest to computing education researchers. One research approach to understanding how beginners acquire their programming abilities has been to look at student performance in exams. Lopez et al. (2008) analyzed student responses to an end-of-first-semester exam. They found two types of questions accounted for 46% of the variance on the code writing portion of the same exam. One of those types of question required students to trace iterative code, while the other type required students to explain what a piece of code did. In this paper, we investigate whether the results by Lopez et al. may be generally indicative of something about novice programmers, or whether their results are just an artifact of their particular exam. We studied student responses to our own exam and our results are broadly consistent with Lopez et al. However, we did find that some aspects of their model are sensitive to the particular exam questions used. Specifically, we found that student performance on explaining code was hard to characterize, and the strength of the relationship between explaining and code writing is particularly sensitive to the specific questions asked. Additionally, we found Lopez et al.'s use of a Rasch model to be unnecessary, which will make it far easier for others to conduct similar research.

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, hierarchy.

1. INTRODUCTION
Over the last five years, the BRACElet project has investigated a possible hierarchy of programming skills. At 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 code.

One of the earliest BRACElet papers (Whalley et al., 2006) reported on the performance of students in an end-of-semester exam. As part of that exam, the students were given a question that began "*In plain English, explain what the following segment of Java code does*". Whalley et al. 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*"). Furthermore, it was noted that the better a student performed on other programming–related tasks in that same exam, the more likely the student was to provide such a correct summary. In a follow up 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 in a hierarchy of programming skills.

In a subsequent BRACElet study, Philpott, Robbins and Whalley (2007) found that students who could trace code with less than 50% accuracy could not usually explain similar code, indicating that the ability to trace code is lower in the hierarchy than the abilty to explain code. Also, Sheard et al. (2008) found that the ability of students to explain code correlated positively with their ability to write code.

While the recent BRACElet 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 also skills similar to explanation, is present in quite old literature on novice programmers. Perkins et. al. (1989) discussed the importance and role of tracing as a debugging skill. Soloway (1986) claimed that skilled programmers carry out frequent "mental simulations" of their code, and he advocated the explicit teaching of mental simulations to students.

1.1 Lopez et al. (2008)
Of the BRACElet studies into a hierarchy of programming skills, the most statistically sophisticated was undertaken by Lopez et al. (2008). They analyzed student responses to an end-of-first-semester exam by placing their exam questions into several categories, which included:

- ● **Basics**, where students are required to recall knowledge of Java constructs, recognize the definition of common Java terms, and detect syntax errors.
- ● **Data**, where students are tested on their knowledge of data types, and operations on data types.
- ● **Parson's Problems**, where students are given a set of lines of code, in random order, and are required to place the lines into the correct order to solve a given task.
- ● **Tracing2**, which are tracing tasks that involve loops. Students have to nominate the values outputted, or the values in one or more variables when the code finishes.
- ● **Tracing1**, which are tracing tasks that do not involve loops.
- ● **Explain**, as discussed above.
- ● **Writing**, where students write code for a given problem.

In their analysis, Lopez et al. used stepwise regression to construct a hierarchical path diagram where the points each student earned on the exam's code writing tasks was the dependent variable (i.e. at the top of hierarchy). At or near the bottom of the resultant hierarchy were "basic" and "data" questions. Highest in the intermediate levels of the path diagram were "explain" and "tracing2" tasks. Figure 1 shows that higher portion of the Lopez et al. hierarchy. It can be seen that the points students earned on tracing iterative code accounted for only 15% of the variance of student points for the writing question (i.e. $R^2 = 0.15$) and the points students earned on "explain" questions accounted for only 7% of the variance in the writing question. However, in combination, the "tracing2" 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 a follow up study, Lister, Fidge and Teague (2009) performed a non-parametric analysis on similar data and found statistically significant relationships between these variables.

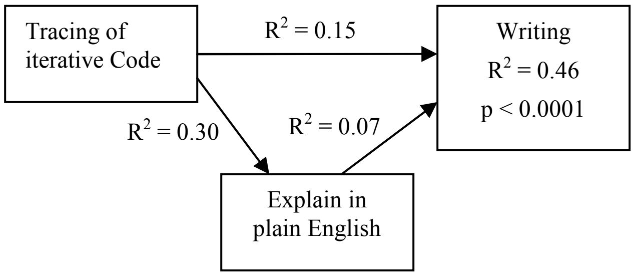


Figure 1: The upper portion of the stepwise regression model from Lopez et al. (2008).

1.2 Research Issues and Questions

In this paper, we report on our own study of the relationships between tracing iterative code, explaining code, and writing code. (Henceforth, when we refer to "tracing", we will be referring to "tracing2" problems, the tracing of iterative code.) Our study is motivated by a number of issues and questions raised by the Lopez et al. study, which we discuss in this section.

Are the relationships that Lopez et al. found a reflection of the intrinsic nature of the categories into which they broke their tasks (i.e. tracing, explain, etc), or are these relationships due to the specific exam questions they asked? For example, is their report of a weak ($R^2 = 0.15$) statistical relationship between tracing tasks and code writing due to the intrinsic nature of tracing tasks or is the weak relationship simply due to Lopez et al. using poor tracing questions? To express this more generally, did Lopez et al. find fundamental relationships between coding and non-coding tasks, or are their results simply an artifact of their specific exam paper?

An even more fundamental issue is whether code writing is something an end-of-first-semester novice can do reliably. That is, if we have a set of N programming problems that we (as teachers) regard as being similar, do novice programmers at an equal stage of their development tend to perform consistently across that set of problems? In other words, are N-1 problems in such a set of problems a good predictor of how novices will perform on the Nth problem? If not, then there is little point in looking for consistent, general statistical relationships between code writing and non-code writing tasks. The exam used by Lopez et al. contained two writing questions. They reported statistics indicating that students performed very differently on their two writing questions. For example, the average class grades on these two writing questions were 29% and 45%. Lopez et al. did not consider the reasons why student performance differed on these code writing questions, nor did they discuss any implication this inconsistency might have for the generality of their model. In this paper, we will study the performance reliability of end-of-first-semester novices on the questions in our exam.

1.2.1 *Rasch Model*

Another issue with the Lopez et al. study is the sophistication of the techniques used to analyse the data. They used a polytomous Rasch model to preprocess their data. Doing so addresses some of the non-linearities in grading. For example, in a question graded on a 10 point scale, the difference in quality between two student answers, one scoring 5 points and the other 6 points, is probably not the same as the difference in quality between two other student answers, one scoring 9 and the other 10 points. However, there are difficulties with using Rasch models, beyond the inherent complexity. One difficulty is that it is a technique originally intended for larger datasets than the sets typically collected from small college classes.

Another difficulty with using a Rasch model is the need to find a suitable parameter that describes the general ability of each student who took the exam. For a test containing many items (and ideally a test taken by a large number of people) the score of each person on the entire test is often used as the measure of general ability. However, for a test with a small number of questions, taken by a small number of people (as is the case for both Lopez et al. and us) it is not clear whether the overall score for the test can be used as the measure of general ability. In fact, and perhaps for this reason, Lopez et al. chose not to use the overall score. Instead, they set aside one exam question from their path analysis – an essay style question – and used student scores on that question as their measure of general ability. The question then arises as to whether the essay question used by Lopez et al. was a valid measure of general ability. More pragmatically, setting aside such an exam question for this purpose is not an option for us in our study, and likewise, it would not be an option for most computing education researchers attempting similar studies.

In this paper, we do not use a Rasch model. Instead, we work with relatively "raw" data points, to investigate whether the same relationships found by Lopez et al. on their Rasch-processed data can still be found in our data. It is our conjecture that it should still be possible to find these relationships in "raw" data, especially when the exam questions used display a relatively uniform distribution of grade points.

1.3 Study Sample

The data we used for our study 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 first and second authors. Students from this university had not participated in any of the earlier studies summarized above. These students were taught the Java programming language. Thirty two students took the final exam which was subsequently graded by one person.

2. DEPENDENT VARIABLE: WRITING

As teachers, our aim is to produce students who can write programs. Therefore, the analysis in this paper, like the analysis in the Lopez et al. paper, treats student performance on code writing as the dependent variable, and studies the relationship between student performance on code writing questions and non-code writing questions. In this section, we describe the three code writing questions from our exam, and investigate the performance reliability of our students on these three tasks.

As Traynor, Bergin, and Gibson (2006) have argued, the true indication of the difficulty of a code writing task is not the task itself, but how the task is graded. We therefore also describe, in detail, how each of our tasks was graded. In all three cases, the grading schemes were designed solely for the purposes of grading the students, prior to our analysis commencing. One person graded all writing tasks for all students.

2.1 Two Iterative Code Writing Questions

Two of the three code writing questions required the students to write iterative code.

2.1.1 Sum of N

For one of the two iterative code writing questions, the students were told to "*write a segment of Java code that will show the sum of 1 to n for every n from 1 to 20. That is, the program prints 1, 3 (the sum of 1 and 2), 6 (the sum of 1, 2 and 3), and so on. You may use either while or for loops and your program is required to produce screen output in two columns the same as the following* … <Sample output was then given, including a heading> … *You are not required to provide the entire program*". A suitable solution (with the heading omitted) may have looked like this:

```
int  sum = 0;
for ( int n=1 ; n<=20 ; n++ )
{
    sum = sum + n;
    System.out.println(n + "    " + sum);
}
```

The students' answers to this question were graded on a 7 point scale. One point was awarded for outputting a correct heading, and 1 point was awarded for a correct declaration and initialization of variable that would subsequently be used to accumulate the sum (i.e. "sum" in our sample solution). One point was awarded for a correct formulation of the "for" loop (or an equivalent "while"). Three points were awarded for correctly calculating the sum. In many cases, students wrote an inner loop to calculate the sum, and this was not penalized. The final point was awarded for a correct `println` statement. In subsequent analysis, we will regard students scoring 5 or higher as having manifested a grasp of the problem.

2.1.2 Average

The second iterative code writing question required students to provide code that "*continually asks the user to enter a list of positive numbers. When the user has completed their list, they type a negative number that stops the input process. Once input is complete, the program prints out the average of the numbers to the screen*". The question then continued, giving two sample sessions of the program. The second of these sample sessions illustrated the case where the first number inputted is negative, in which case the program was shown to output "No list to average". As students were not taught exception handling prior to this exam, they were not required to handle input exceptions in their answer to this question. A suitable solution may have looked like this:

```
int n = 0;
double sum = 0;
Scanner stdin = new Scanner(System.in);
double x = stdin.nextDouble();
while ( x > 0 )
{
    n++;
    sum = sum + x;
    x = stdin.nextDouble();
}
if ( n > 0 )
 System.out.println("Average: " + sum/n);
else
 System.out.println("No list to average");
```

This code writing exercise is similar to Soloway's well known "rainfall problem" (Soloway, Bonar, and Ehrlich, 1983; Soloway, 1986). As Soloway et al. demonstrated, this is not a problem that all novices solve easily.

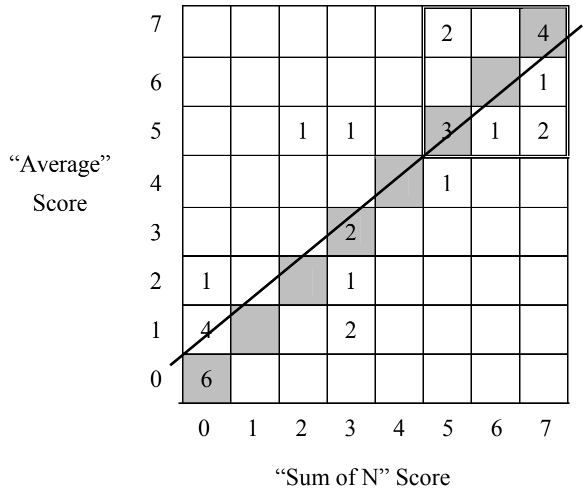
As with the "Sum of N" writing task, the students' answers to this task were graded on a 7 point scale. One point was awarded for declaring and initializing two variables analogous to "n" and "`sum`" in our sample solution. One point was awarded for a read prior to the loop and 1 point was for having a while loop that correctly tested for a positive input value. Within the loop, 1 point was awarded for an assignment statement to accumulate the sum, while 0.5 points were awarded for incrementing the variable that counted the number of values inputted, and another 0.5 points were for reading the next input value. The final 2 points were for the "`if`" statement after the loop. As with the other iterative writing problem, we will regard students scoring 5 or higher as having manifested a grasp of the problem.

2.1.3 Comparison of Grade Distributions

Figure 2 illustrates that most students scored similar points on these two iterative code writing tasks. Both problems were graded on a 7 point scale. The shaded squares indicate equal scores on both problems. Almost half the students (47%) scored exactly the same points on both problems. Almost three quarters of the students (72%) scored either the same points or points that differed by only 1 point. Only 1 student had a point difference greater than 2. The diagonal line though the figure is a regression line, with a relatively high R^2 value of 0.8. The 3 by 3 square at the top right of the Figure 2 (indicated by the double lines) contains the 13 students who are regarded as having manifested a grasp of both problems.

A student's performance on either of these two code writing problems is a good predictor of the student's performance on the other problem. Had that not been the case, there would be few grounds for believing that the non-code writing tasks that we analyze later in the paper might be good predictors for code writing. Furthermore, we can regard $R^2 = 0.8$ as an informal upper expectation of the extent of the relationship between code writing and non-code writing tasks.

Given this high degree of collinearity between these two iterative code writing tasks, there is little to be gained by building regression models for each and so, in further analysis, we will often consider the two problems combined. Figure 3 shows the distribution of student scores when their scores on "Sum of N" and "Average" are added together. There is some clumping at each end of the distribution, since several students did either very well or very poorly on both problems, but between the extremes the distribution is a relatively uniform.



"Average" \ "Sum of N"	0	1	2	3	4	5	6	7
7						2		4
6								1
5			1	1		3	1	2
4						1		
3				2				
2	1			1				
1	4			2				
0	6							

Figure 2: Student scores on "Sum of N" and "Average" code writing tasks, with a line of regression ($R^2 = 0.8$).

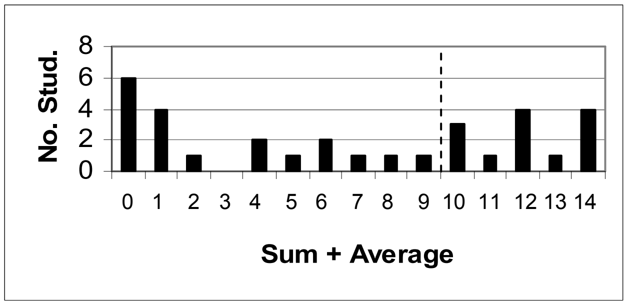


Figure 3: Distribution of student scores when their scores on "Sum of N" and "Average" are added together. The dashed line indicates the threshold at which we regard students as having demonstrated a grasp of both problems.

2.2 Non-Iterative Code Writing Question

The third code writing question required the students to write a nested set of "*if*" statements. The students were given the following instructions: *The speed limit on a freeway is 100 km/h. Drivers exceeding the speed limit will be issued a speeding ticket. The amount of the fine depends on whether this is their first traffic offence or not and how much over the speed limit they are driving if it is their first traffic offence. Using the table given below, write the missing code for the Java program on the following page so that the program will display one of the messages, depending on the input entered by the user. Demonstrate the use of nested if statements in your solution.*

Speed (km/h)	First Traffic Offence	Message to be displayed
<= 100	Not Applicable	Drive Safely!
101 – 120	True	Your ticket is $105
>121	True	Your ticket is $160
More than 100	False	Your ticket is $200

Students were provided with the following variable declarations:

```
  final int LIMIT = 100;
  int speed;
  String name;
  boolean fineBefore;
```

A suitable solution may have looked like this:

```
Scanner stdin = new Scanner (System.in);
System.out.print("Enter your name: ");
name = stdin.next( );
System.out.print("What was your speed: ");
speed = stdin.nextInt( );
System.out.print("Any previous fines: ");
fineBefore = stdin.nextBoolean( );

if ( speed <= LIMIT )
      System.out.println ("Drive Safely");
  else
  {
    System.out.print(name +
                       ", your ticket is ");
    if ( fineBefore )
        System.out.println("$200! ");
    else
    {
       if ( speed <= 120 )
          System.out.println("$105! ");
       else
          System.out.println("$160! ");
    }
   }
```

This question was graded out of 10 points. The grading scheme awarded a student up to 4 points for the code that (in our sample solution) precedes the first "`if`" condition, 1 point for an "`if`" handling "`speed <= LIMIT`", 1 point for an "`else`" associated with that first “`if`” (i.e. 6 points cumulative to this stage). Inside that “`else`”, 1 point was awarded for handling the input of whether the driver had been fined before, and the remaining 3 points were allocated for the remaining code commencing (in our sample solution) at “`if (fineBefore)`”. In subsequent analysis, we will regard students scoring 8 or higher as having manifested a grasp of the problem.

2.2.1 Grade Distribution

Figure 4 shows the distribution of student scores on this “Speeding” task. There are few students who did poorly on this task, and quite a few students who did well.

Figure 5 compares the performance of students on “Speeding” with their performance on the combined iterative problems. The dashed line in the figure indicates equal percentage points on “Speeding” and on the iterative tasks. Over 90% of the students did better on “Speeding”. The solid line is a regression, with a moderate $R^2 = 0.54$. Again, this R^2 of 0.54 is a guide to what we might expect of any relationship between code writing and non-code writing tasks.

It could be argued that the control logic in the “Speeding” task is more complex than the control logic of both iterative tasks. (One author of this paper took far longer to write a sample solution for “Speeding” than for the two iterative tasks – we suggest that readers place to one side the sample solutions given in this paper and see how long it takes them to write their own solutions.) That most students did better on “Speeding” may be due to iteration being harder to master than selection, but it also could be an artifact of the grading schemes for these exam questions.

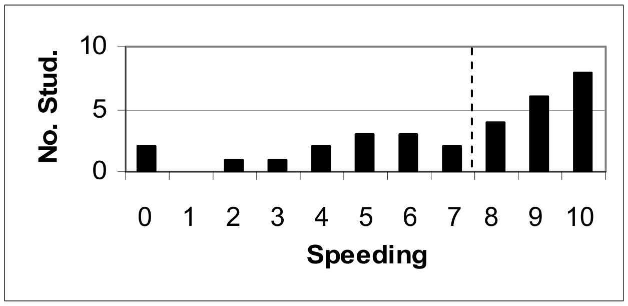


Figure 4: Distribution of student scores on the non-iterative “Speeding” code writing problem. The dashed line indicates the threshold at which students are regarded as having demonstrated a grasp of the problem.

A relatively uniform distribution of student scores for code writing results from adding together each student’s points on all three writing tasks, as is shown in Figure 6. In the remainder of this paper, we will refer to this combination of the three tasks simply as “writing”.

2.3 Aside: Predictors of Success

In Computer Science Education research, there has been a great deal of work on pre-enrollment “predictors of success” (e.g. Wilson, 2002), where “success” is often the grade students earned in their first programming course. Factors studied include SAT scores, mathematical background and gender. Some linear regression models have accounted for almost half the variation of student grades at the end of the first semester. Given the R^2 values we reported above, between our three code writing tasks that students completed in a single exam session, the performance of these “predictors of success” is impressive, and may be near the upper limit of what can reasonably be expected.

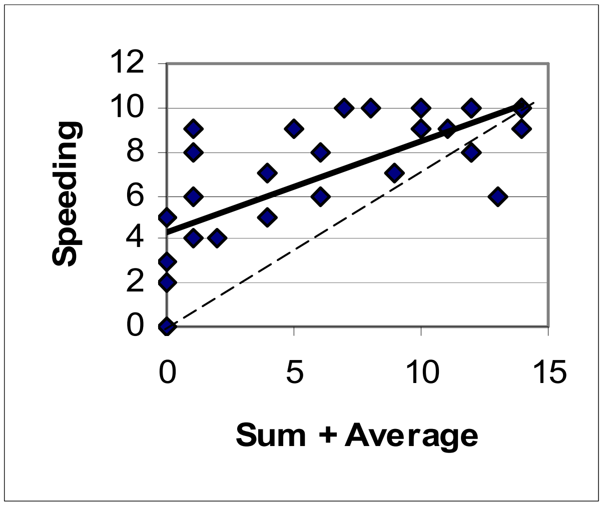


Figure 5: A comparison of student scores on the combined iterative tasks with the “Speeding” task. (R^2 = 0.54).

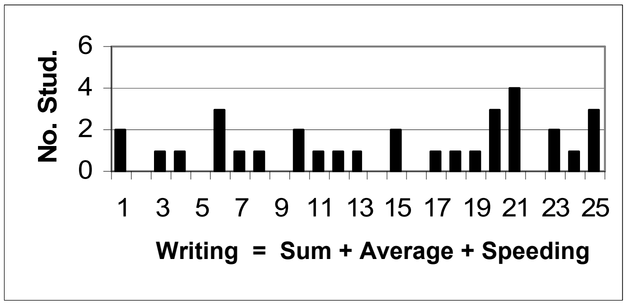


Figure 6: Distribution of student scores when their scores on “Sum of N”, “Average” and “Speeding” are combined.

2.4 Procedural vs. Object Oriented Questions

The exam also contained three other questions. Unlike all other questions described in this paper, those three questions tested students on aspects of object-oriented programming. Please note that those three questions are not included in this analysis, as our focus is upon the ability of students to understand and write procedural code.

3. INDEPENDENT VARIABLES

In this section, we describe the non-code writing questions in our exam, using the Lopez et al. classification scheme. We also provide examples of each type of question so that the reader can develop a stronger impression of what our exam questions are like. At this stage of the development of our field of research, we feel the only way to ensure that our work (and similar work by others) is reproducible is to include a representative sample of exam questions, verbatim. Eventually, Computer Science Education research may develop a comprehensive classification system for exam questions that allows authors to pithily describe the salient features of their exam to other researchers, but the classification scheme offered by Lopez et al. (2008) is a first and rudimentary step toward such a comprehensive exam question classification scheme.

The non-code writing exam questions analyzed in this paper consisted of three broad question types: (1) 30 multiple choice questions, (2) two short answer explanation questions; and (3) one Parson’s problem. The 30 multiple choice questions can be further broken down into four Lopez classifications – “basics”, “data”, “tracing1” and “tracing2”. Students had three hours to complete the entire exam, which was intended to be ample time, and anecdotal evidence suggests that it was ample time.

This paper focuses upon studying the higher levels of the Lopez et al. path diagram so we will not describe all our exam questions. Instead we will only describe our explanation and tracing questions.

3.1 The Two Explanation Questions

Both of our explanation questions begin with the following instruction: *Explain the purpose of the following segment of code. Assume that a, b, c are declared as integers and have been initialized*. The code presented in the first explanation question was:

```
c = b;
b = a;
a = c;
```

A good answer to this first explanation question would be “*It swaps the values in a and b*”. We took this question from the earlier study by Sheard et al. (2008). The code for the second explanation question was as follows:

```
if ( a < b)
   if ( b < c)
       System.out.println (c);
   else
       System.out.println (b);
else if ( a < c)
       System.out.println (c);
else
       System.out.println (a);
```

A good answer to this question would be “*It prints out the largest of the three values*”.

Each explain question was graded out of a total of 4 points where students were given 4 points if they correctly summarized the function performed by the code (as the above model answers do). Students were given 3 points if they correctly described the behaviour of every line of code, and fewer points if they showed some partial understanding of the lines of code. This grading was done as a routine part of grading the entire exam, and was done by one person, prior to the analysis presented in this paper.

Prior to the exam, students had seen only one example of an explanation question, but they were also told there would be at least one such question in the exam and answers that summarized the code would score more than line by line descriptions.

3.1.1 Reliability of Explain Questions

Table 1 provides the frequency of student scores on each of these two explanation problems. The table shows that the bulk of the students either did very well or very poorly on “Swap”, whereas relatively few students did very poorly on “Largest”. These results indicate that any statistical relationship between explanation and code writing (particularly linear relationships) may vary according to the exact explanation questions used. Given that “Swap” has fewer lines of code and fewer programming constructs than the “Largest” code, it would seem that the difficulty of explanation questions cannot be characterized by simple measures, such as the number of lines of code, or the programming constructs used.

		“Swap” Score				“Largest” Totals
		1	2	3	4	
“Largest” Score	1	2			1	3
	2	5			1	6
	3	3		2		5
	4	6			12	18
“Swap” Totals		16	0	2	14	

Table 1: The frequency of student scores on each of the two explanation questions.

3.1.2 Linear Prediction of Writing from Explaining

The statistical relationship between student points on the combined explanation questions and the combined writing questions is shown in Figure 7. The figure shows that the students who received maximum points on both explanation tasks (i.e. the rightmost vertical line of points) did better than most of their classmates on the writing tasks. A line of regression through all points in the figure (solid line) has $R^2 = 0.49$. For students who scored less than maximum points, however, the explanation questions are a poor linear predictor of writing performance. A line of regression through the points for just those students (dashed line) only has $R^2 = 0.06$. For their equivalent data on explanation and writing questions, Lopez et al. reported $R^2 = 0.07$ for a line of regression through all points. (We note, however, that our R^2 values were attained without using a Rasch model.)

Given the markedly different points distributions for “Swap” and “Largest”, and also between “Speeding” and the two iterative writing problems, we might expect to see markedly different R^2 values for various combinations of explanation and writing tasks. Table 2 shows that, while the R^2 values do vary, regularities are apparent. For example, the combination of the two explanation tasks always results in a higher R^2 than either of the explanation tasks alone. Also, the iterative code writing tasks, both individually and collectively, have a higher R^2 than the non-iterative “Speeding” task. Such regularities are an indication that, while the exact strength of the relationship may vary, there is a general relationship between explanation and writing.

It is interesting that the students who received maximum points on both explanation tasks (i.e. correctly summarized both pieces of code) did relatively well on the code writing questions, when neither of the explanation questions involved a loop, whereas two of the three writing tasks did require loops. We offer no firm explanation for this, but suggest that it may be that novices who can regularly “see the forest” in code, irrespective of what programming constructs are in that code, are better placed to exhibit the higher level skills required to envisage the solution to a writing task.

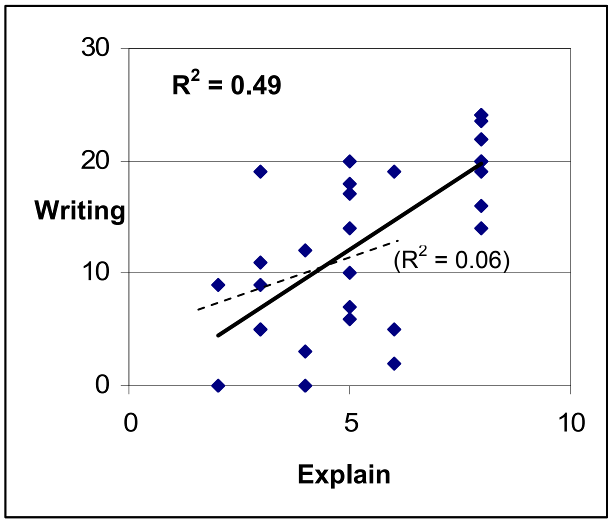


Figure 7: A comparison of student scores on explanation and writing tasks.

Writing Task(s)	Explanation Task(s)		
	Swap	Print Largest	Combined
Speeding	0.22	0.19	0.30
Sum of N	0.46	0.25	0.53
Average	0.35	0.28	0.46
Iterative Combined	0.43	0.28	0.52
All Writing	0.39	0.28	0.49

Table 2: R^2 values for various combinations of explanation and writing tasks.

3.1.3 Non-Parametric Analysis

Figure 7 shows that, for students who scored less than the maximum number of points on the explanation questions, there is no clear relationship between their performance on those explanation questions and their performance on the writing questions. In contrast, none of the students who scored the maximum number of points on explanation questions did poorly on the code writing questions. This suggests that, while there may be a relationship between being able to explain code and being able to write code, that relationship is not linear. In this section, we explore the possibility of such a non-linear relationship, by carrying out a simple non-parametric analysis of the data.

Earlier in the paper, we defined a score of 8 or higher on the non-iterative "Speeding" task, and a score of 5 or higher on either iterative task, as an indication that a student had demonstrated a grasp of these writing tasks. These threshold scores, for each problem, divide the students into two groups. Similarly, a student who receives the maximum score (i.e. 4) on an explanation question has (by definition of the grading scheme) demonstrated a grasp of the overall computation performed by that particular piece of code. Again, students can be divided into two groups on each explanation problem, according to whether or not they demonstrated sufficient grasp of the explanation question to receive a perfect score. For any one writing task, in combination with any one explanation task, we can divide the students into four groups, according to whether or not they achieved the threshold score on each task. With four such groups, the non-parametric chi-square test can be used to test whether there is a relationship between the two tasks.

Table 3 shows the results of chi-square analysis on combinations of explanation and code writing tasks. In our analysis, with four groups in each chi square calculation, the degrees of freedom (df) = 1, so any $\chi^2 \geq 4$ is significant at $p \leq 0.05$. All the relationships in Table 3 are significant at that level, except for the relationship between the "largest" explanation task and the combination of all three writing tasks (where $\chi^2 = 2.3$). Therefore, with that one exception, our data indicates that there is a non-linear but statistically significant relationship between the ability to "see the forest" in a given piece of code and being able to write code.

Writing Task(s)	Explanation Task(s)		
	Swap	Print Largest	Both Explains
Speeding ≥ 8 points	5.0	7.7	5.7
Sum of N ≥ 5 points	12.3	8.8	12.3
Average ≥ 5 points	10.0	6.5	10.2
Both Iterative ≥ 5	9.8	7.2	14.6
Speeding ≥ 8 and both Iterative ≥ 5	4.3	2.3	7.6

Table 3: The χ^2 values for various combinations of explanation and writing tasks.

3.2 Tracing Questions

Lopez et al. (2008) define "tracing2" questions as tracing tasks involving loops. As discussed earlier, we will henceforth refer to tracing2 questions simply as "tracing" questions. The following multiple choice tracing question was the easiest for our 32 students, with 81% of them correctly selecting option "c":

What is printed to the screen by the following code?

```
for (int count=0; count<4; count--)
   System.out.print(count);
```

```
a) 0 + 0 + 0 + 0 + 0 + 0 + …continuously
b) 0 + 1 + 2 + 3 + 4 + 5 + …continuously
c) 0-1-2-3-4 …continuously
d) no output to the screen
```

The lowest percentage of correct responses for a tracing question was 56%, for this question, where the correct solution is "b":

What is the output to the screen by the following code?

```
int n = 4;
for (int row=1; row<=n; row++) {
  for (int column=1; column<=n; column++)
     if (row==1 || row==n || column==1
                          || column==n)
        System.out.print ("* ");
     else
        Sytem.out.print ("  ");
     System.out.println( );
}
```

```
a) *           b) * * * *
   * *            *     *
   * * *          *     *
   * * * *        * * * *

c) * * * *     d) * * * *
   * * * *          * * *
   * * * *            * *
   * * * *              *
```

(Note: In the exam, which was formatted as a single column document, the "`if`" in the above code was not broken across two lines).

No other tracing question involved nested loops. However, two other tracing questions were like the above tracing question in that they contained a conditional inside a loop**.** Henceforth, we will refer to these three problems collectively as the "Complex" tracing problems. One of the other complex tracing questions was the second hardest tracing question for the students (59% answered it correctly), while the remaining complex question ranked as one of the more easily answered questions of all nine tracing questions (69% answered it correctly).

Six of the "tracing" problems involved a single loop without a conditional inside the loop. Henceforth, we will refer to these six problems collectively as the "Simple" tracing problems. Below is the simple tracing question that received the median percentage of correct responses (i.e. 63% for option "b") of all 9 tracing questions:

What is printed to the screen by the following code?

```
int number = 3;
while (number == 3)
  { System.out.print(number + " + " );
    number++;
  }
```

```
a) 3
b) 3 +
c) 3 + 4
d) 3 + 4 + 5 + 6 + …continuously…..
e) 3 + 3 + 3 + 3 + …continuously…..
f) nothing will be printed to the screen
```

Figure 8 shows the distribution of scores for all "tracing" questions. Nineteen of the students (59%) scored 7 or higher on these 9 questions. Given the variable number of options used in these multiple choice questions, the expected value for students answering by guessing is 1.8 (20%).

3.2.1 Simple vs. Complex Tracing Questions

On any pair of the three tracing questions given in full above (i.e. any pairing of the three tracing questions that our students found easiest, hardest and of median difficulty) the percentage of students who answered both questions in the pair correctly, or both questions in the pair incorrectly, varied in the small range of 63-69%. Those percentages are an informal but intuitive description of the reliability of the nine tracing questions.

A more formal measure of reliability is the classic Cronbach's alpha. For the 9 tracing questions answered by our 32 students, Cronbach's alpha is 0.87. A common rule of thumb is that an alpha higher than 0.7 (or 0.8 for some people) is considered an indication of a reliable set of multiple choice questions.

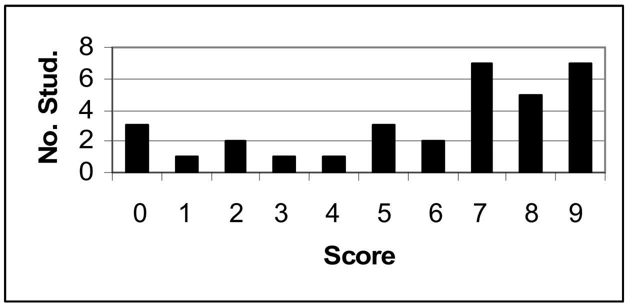


Figure 8: The distribution of student scores on the nine tracing questions.

For the complex tracing questions, Cronbach's alpha was only 0.66, but a lower alpha is to be expected for only three questions. For the six simple tracing questions, Cronbach's alpha was 0.85.

The performance relationship between simple and complex tracing questions is non-linear. Of the 8 students who scored less than 50% on the simple problems, all but one scored 0 or 1 on the 3 complex tracing problems. Given that these complex tracing problems are multiple choice questions, that is a performance level that is probably most easily explained by chance. For the 24 students who scored 50% or higher on the simple tracing questions, the average student score on the 3 complex tracing problems was 2.2. However, a line of regression through the data points for these 24 students is almost horizontal, and has an R^2 value of only 0.03. In mathematical parlance, ≥50% performance on the simple tracing problems is a necessary condition for being able to answer the complex tracing problems, but it is not a sufficient condition. We surmise that the complex tracing problems require a systematic approach to tracing (perhaps involving pen and paper) that is not required for the simple tracing problems.

3.2.2 Linear Prediction of Writing from Tracing

The linear statistical relationship between student score on the nine tracing questions and the combined writing questions is shown in Figure 9. The solid line in the figure is a line of regression through all data points, with an associated $R^2 = 0.50$ (shown in bold). This is an R^2 considerably higher than the $R^2 = 0.15$ reported by Lopez et al. These differing R^2 values indicate (as we similarly concluded for explanation questions) that the predictive power of tracing questions is sensitive to the exact nature of the questions. Furthermore, we note (as for explanation questions) that our higher R^2 value was attained, unlike Lopez et al., without using a Rasch model.

The dashed line in Figure 9 is a line of regression for the subset of tracing scores ≥4. That line only has an associated $R^2 = 0.23$. That is, Figure 9 illustrates that students who score poorly on the tracing questions rarely score well on the code writing tasks, but there is no clear relationship with code writing for students who scored well on tracing questions. This suggests a causal relationship, where a minimal level of skill at tracing is necessary for code writing, but that minimal skill at tracing is not sufficient by itself to enable code writing.

Figure 10 shows the relationship between student scores on the six simple tracing questions and the combined writing questions.

The solid line in the figure is a line of regression through all data points, with an associated $R^2 = 0.51$ (shown in bold). However, the dashed line in Figure 10 is a line of regression for that subset of simple tracing scores ≥3, and it only has an associated $R^2 = 0.06$. Again, this suggests a causal relationship between tracing and code writing, where a minimal level of skill at tracing is necessary for code writing, but that minimal skill at tracing is not sufficient by itself to enable code writing.

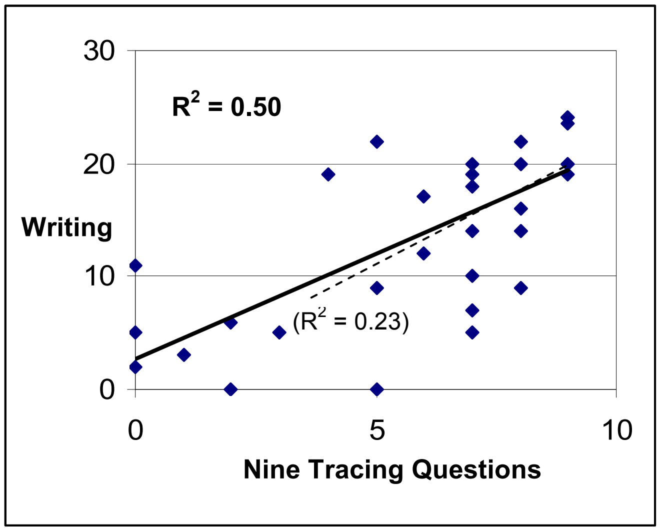


Figure 9: A comparison of student scores on all nine tracing questions and the combined writing questions.

Figure 11 shows the relationship between student scores on the three complex tracing questions and the combined writing questions. Many students who scored a perfect 3 on these complex tracing questions also scored well on the writing tasks. Consequently, each data point in the upper right of Figure 11 represents multiple students. For example, of the rightmost data points, the 3 highest represent 9 students, which makes less obvious in Figure 11 the relationship between the code writing tasks and the three complex tracing tasks. Never-the-less, the line of regression through all of the data points in Figure 11 only has associated $R^2 = 0.27$, so there is not an obvious linear relationship between the complex tracing problems and code writing.

Table 4 shows R^2 values for various combinations of tracing and writing tasks. On all combinations of writing tasks, the six simple tracing tasks have a markedly higher R^2 value than the three complex tracing tasks.

Our intuition was that tracing is an easier skill than writing, so the complex tracing tasks would relate better to performance on writing than the simple tracing tasks. We have no firm explanation for why this proved not to be the case. One possibility is that complex tracing is an error prone activity, and thus best avoided, so part of the skill in code writing is verifying code without doing complex tracing.

3.2.3 Non-Parametric Analysis

Figures 9 and 10 show that, for students who scored above certain threshold values on tracing tasks, there is no clear linear relationship between tracing tasks and code writing tasks. In this section, we explore the non-linear relationship between tracing tasks and code writing tasks, via a chi-square analysis.

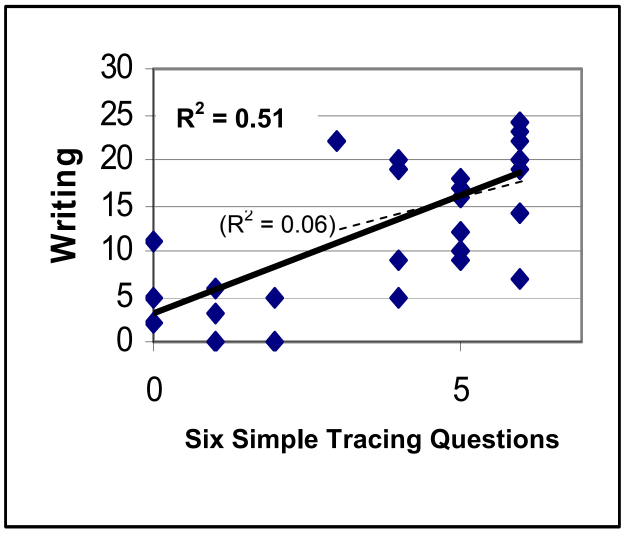


Figure 10: A comparison of student scores on the six simple tracing questions and the combined writing questions.

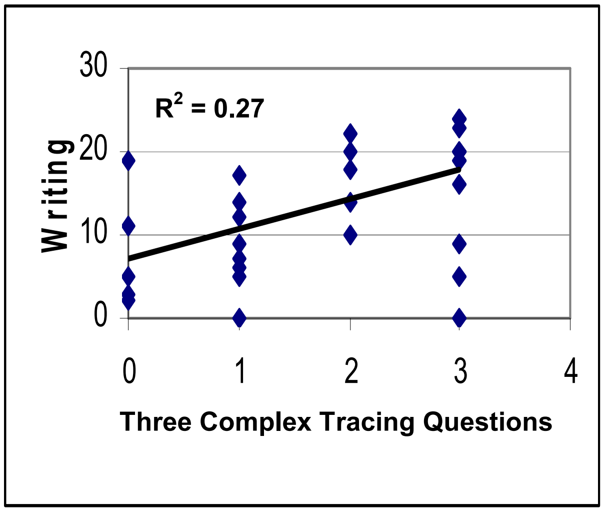


Figure 11: A comparison of student scores on the three complex tracing questions and the combined writing questions.

Writing Task(s)	Tracing Task(s)		
	Simple	Complex	Both
Speeding	0.49	0.15	0.42
Sum of N	0.40	0.27	0.42
Average	0.40	0.29	0.43
Iterative Combined	0.42	0.30	0.45
All Writing	0.51	0.27	0.50

Table 4: R^2 values for various combinations of tracing and writing tasks.

Table 5 shows the results of our chi-square analysis on various combinations of tracing and writing tasks. As was the case with our earlier chi square analysis of the explanation questions, any $\chi^2 \geq 4$ is significant at $p \leq 0.05$. The chi square analysis confirms two relationships which already appeared to be the case from the above linear analysis – that (1) students who score three or higher on the six simple tracing problems tend to do better on writing tasks than students who scored less than 3, but (2) a score of five or higher on the six simple tracing problems is not necessarily an advantage over a score of 3 or 4. The chi square analysis also confirms two relationships not apparent in the linear analysis – that (3) students who performed at or above a given threshold on the tracing problems (either simple, complex or both) tend to do better on the non-iterative "Speeding" problem than students who performed below the given threshold, and (4) there is a significant non-linear statistical relationship between the three complex tracing tasks and code writing.

Writing Task(s)	Score on Tracing Task(s)			
	Simple		Complex	Tracing
	≥3	≥5	≥2	≥ 7
Speeding ≥ 8	13.7	7.8	9.8	9.8
Sum of N ≥ 5	8.3	2.3	11.6	7.2
Average ≥ 5	9.4	3.4	8.7	5.0
Iterative ≥ 5	7.3	1.5	9.8	5.8
Speeding ≥ 8 && Iterative ≥ 5	6.4	0.8	8.3	4.6

Table 5 The χ^2 values for various combinations of tracing and writing tasks.

4. TRACING AND EXPLAINING

In the introduction, we mentioned that Philpott, Robbins and Whalley (2007) found that students who traced code with less than 50% accuracy could not usually explain similar code, indicating that the ability to trace code is lower in the hierarchy than the abilty to explain code. In support of that finding, Lopez et al. found a linear relationship between tracing and explaining ($R^2 = 0.30$, see Figure 1). In this section, we investigate whether we also find a similar relationship in our data.

Between our nine tracing tasks and our two explanation tasks, we find a linear relationship of comparable strength to that found by Lopez et al. (our $R^2 = 0.26$). Table 6 shows R^2 values for various combinations of tracing and explanation tasks. Most of the R^2 values indicate that the linear relationships between tracing and explanation are weak.

Table 7 presents the results of a chi-square analysis, into the non-linear relationship between tracing and explaining. For example, the analysis that led to the top left chi value of 4.2 compared student performance on the "Swap" question (i.e. whether or not the students provided a correct summary) with their performance on the six simple tracing tasks (i.e. whether or not the students answered at least 50% of those questions correctly). As with earlier analysis, $\chi^2 \geq 4$ is significant at $p \leq 0.05$. For all combinations of tracing and explanation tasks, and for all threshold values on the tracing tasks, there is a statistically significant relationship between tracing and explaining.

Explain Task(s)	Score on Tracing Task(s)		
	Simple	Complex	Both
Swap	0.13	0.10	0.14
Largest	0.27	0.09	0.24
Both	0.27	0.13	0.26

Table 6: R^2 values for various combinations of tracing and explanation tasks.

Explain Task(s)	Score on Tracing Task(s)				
	Simple		Complex	Tracing	
	≥3	≥5	≥2	≥ 5	≥ 7
Swap	4.2	5.0	7.2	4.2	7.2
Largest	8.3	12.3	5.8	8.3	9.8
Both	6.4	5.7	8.3	6.4	8.3

Table 7: The χ^2 values for various combinations of tracing and explanation tasks.

5. TRACING, EXPLAINING & WRITING

Until this point of the paper, we have investigated pair wise relationships between any two of tracing, explaining and writing. In this section, we analyze the combined effect of tracing and explaining upon writing.

Figure 12 illustrates the results of a multiple regression, with score on the code writing tasks as the dependent variable. The independent variables are the scores on the nine tracing tasks and the scores on the two explanation tasks. The line of regression in the figure has an associated $R^2 = 0.66$. Our R^2 is higher than the $R^2 = 0.46$ reported by Lopez et al., indicating (as earlier regressions also showed) that the predictive power of these models is sensitive to the exact nature of the exam questions used. Furthermore, we note (as we have for earlier regressions) that our higher R^2 value was attained, unlike Lopez et al., without using a Rasch model.

Earlier in the paper, in section 2.1.3, and illustrated in Figure 2, we reported that a plot of the two iterative tasks (i.e. "Sum of N" and "Average") yielded $R^2 = 0.8$. At that point in the paper, and given the similarity in those two iterative code writing tasks, we wrote that $R^2 = 0.8$ could be regarded as an informal upper expectation of the extent of the relationship between code writing and non-code writing tasks. Given that expectation, the $R^2 = 0.66$ reported in Figure 12 (i.e. 83% of our informal upper expectation) is an excellent outcome, especially with only two explanation tasks available for inclusion in our model. Furthermore, we remind the reader that the R^2 value between the non-iterative "Speeding" task and the combination of the two iterative tasks was only $R^2 = 0.54$, which is lower than what we have achieved with the regression model in Figure 12.

In all of our earlier figures, there were data points that fell well away from the associated line of regression. A visually striking feature of Figure 12 is the absence of data points that are well removed from the line of regression – there are no points toward the upper left or the lower right of Figure 12. One interpretation of this relatively tight distribution around the line of regression is that the two independent variables describe most of the factors leading to performance at code writing.

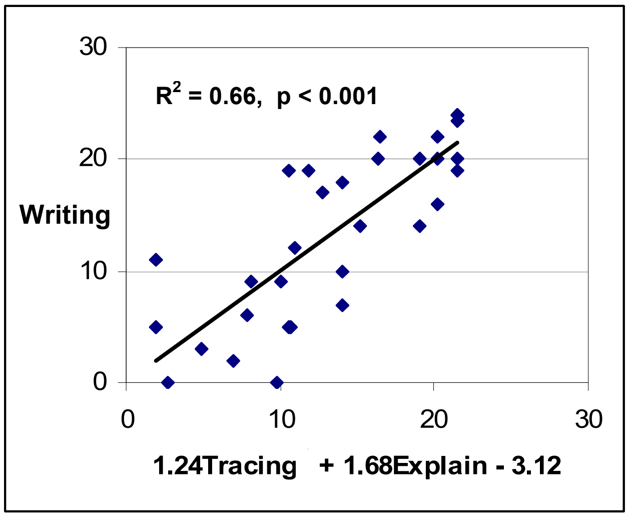


Figure 12: A multiple regression, with score on code writing as the dependent variable, and the combination of scores on tracing and explaining as the dependent variables.

As the label of the x-axis in Figure 12 shows, the co-efficient of the "Tracing" variable is 1.24 and the co-efficient of the "Explain" variable is 1.68. This difference can largely be attributed to the differing number of points awarded to the tracing questions (9 points) and explanation questions (8 points). When allowance is made for that point difference, the weight of tracing questions and the explanation questions in the model is roughly equal. However, while that equality may be due to the tracing and explanation being of equal importance, the difference in how the tracing and explanation tasks were framed and graded suggests that the equality may be a coincidence – the tracing questions were framed as multiple choice questions with 0/1 grading, while the explanation questions were framed as free response questions with a more sophisticated grading strategy.

Table 8 shows R^2 values for various subsets of the three writing tasks. All of these multiple regressions are statistically significant ($p < 0.001$). The R^2 value for the non-iterative "Speeding" task is less than the other R^2 values.

Speeding	Sum of N	Average	Iterative	Writing
0.48	0.63	0.59	0.64	0.66

Table 8: R^2 values for the writing tasks, on multiple regressions of scores on tracing and explain questions.

5.1.1 *Non-Parametric Analysis*

A non-parametric analysis further underlines the strength of the relationship between the code writing and the combination of skill in tracing and explaining. Table 9 shows the percentage of students who provided good answers to the iterative writing tasks (i.e. a score ≥10) for several combinations of their scores on tracing and explain tasks. Of the 8 students who did relatively poorly on tracing and explaining (i.e. the bottom left cell of Table 9), only 1 student out of 8 students (13%) scored 10 or higher on the two iterative writing tasks. Moving horizontally from that lower left cell, we see that there are no students who score less than 50% on tracing and answer both explanation tasks correctly (further illustrating, as Philpott, Robbins and Whalley first observed, that ≥50% accuracy in tracing precedes skill in explaining). If instead we move vertically from the lower left cell (thus maintaining the number of explanation tasks answered correctly at less than 2), we see that only 2 students out of 12 students (17%) scored 10 or higher on the two iterative writing tasks. That percentage difference between these two cells is not statistically significant ($\chi^2 = 0.07$). However, moving from the upper left cell to the upper right cell, we see that 10 students out of 12 (83%) scored 10 or higher on the two iterative writing tasks. The percentage difference between these two upper cells is statistically significant ($\chi^2 = 10.7$). Thus, Table 9 demonstrates that it is the combination of tracing and explaining, more so than each skill independently, that leads to skill in writing.

	Number of correct explanations	
Tracing Tasks Correct	< 2	2
>50%	17% of 12	83% of 12
<50%	13% of 8	Zero students

Table 9: Percentage of good answers to the iterative writing tasks (i.e. score ≥10) for combined scores on tracing and explanation tasks.

6. DISCUSSION

Our results are consistent with the earlier findings of Lopez et al. and the studies upon which they in turn had built. That is, we also found statistically significant relationships between tracing code, explaining code, and writing code. Unlike those earlier studies, we also used non-parametric statistical tests to establish non-linear relationships in our data, and unlike Lopez et al., we found all our reported relationships without resorting to a Rasch model.

We are surprised at the strength of the statistically significant relationships that we found, given our limited amount of data – our exam only contained two explanation questions, and three writing tasks, and was administered to only 32 students. Despite our limited data, our multiple linear regression (i.e. Figure 12) yielded a relatively tight distribution of data points around the line of regression, with an $R^2 = 0.66$ which is 83% of our informal upper expectation. Also, our non-parametric analysis of tracing and explaining in combination (i.e. Table 9) demonstrated that it is the combination of tracing and explaining, more so than each skill independently, that leads to skill in writing.

A high fit between writing and the combination of tracing and explaining may only be observed when the questions within each task type (i.e. tracing, explaining and writing) span a comparable range of difficulty and when these tasks are a good match to the range of abilities found among the students who take the exam.

Our results show that the strength of the relationships between tracing, explaining and writing tasks do vary considerably according to the exact nature of the tasks. Furthermore, the strength of those relationships is not simply a function of obvious aspects of the code, such as the number of lines of code in the tasks, nor is it a function of the degree of congruence between the programming constructs used in the explanation/tracing/writing tasks. More work is required to characterize the critical features of these tasks that explain the variation in the strength of the relationships. Such work will require larger sets of explanation, tracing and writing problems. These larger sets may be too large to administer as part of a conventional end-of-semester exam, and this type of empirical work may need to move to more conventional experimental settings, using student volunteers.

This paper is a study of novice programmers at a very early stage of their development. It is possible that the relationships we report in this paper between tracing, explaining and writing may not hold later in the development of the novice programmer. By analogy, just as a child begins to learn to read by "sounding out" words, so may a novice programmer begin by tracing code, but as both the child reader and the novice programmer develop, they may move to more sophisticated strategies. An obvious and interesting direction for future research would be a cross sectional or longitudinal study of tracing, explaining and writing skills across the entire undergraduate degree.

7. CONCLUSION

From this BRACElet study, and the earlier BRACElet studies upon which it builds, a picture is emerging of the early development of the novice the programmer. First, the novice acquires the ability to trace code. As the capacity to trace becomes reliable, the ability to explain code develops. When students are reasonably capable of both tracing and explaining, the ability to systematically write code emerges.

Most of the results in this paper are correlations, and correlation does not prove causality – perhaps the harder a student studies, the better the student gets at tracing, explaining, and writing? On the basis of the evidence presented in this paper, we cannot dismiss such an argument. However, there are three reasons why we argue for a hierarchy of skills. First, Figures 9 and 10 do not show a strong linear correlation, but instead show a threshold effect, where writing ability is poor below a (roughly) 50% tracing score, and writing ability is only weakly correlated with tracing above that 50% threshold. Second, a hierarchy of tracing, explaining and writing is consistent with general results in cognitive science. Third, the Rasch model used in the earlier work of Lopez et al. allows for underlying group invariance.

While arguing for a hierarchical development of programming skills, we do not support the idea of a strict hierarchy; where the ability to trace iterative code, and explain code, precedes any ability to write code. We believe that all three skills reinforce each other and develop in parallel. Having written a small piece of code, a novice programmer needs to be able to inspect that code, and verify that it actually does what the novice intended – novices need to be able to "explain" their own code to themselves. Also, when writing code, a novice will sometimes need to trace the code. Thus, writing code provides many opportunities to improve tracing and explanation skills, which in turn helps to improve writing skills. In arguing for a hierarchy of programming skills, we merely argue that that some minimal competence at tracing and explaining precedes some minimal competence at systematically writing code. Any novice who cannot trace and/or explain code can only thrash around, making desperate and ill-considered changes to their code − a student behavior many computing educators have reported observing.

ACKNOWLEDGEMENT

Raymond Lister's contribution was part of a Fellowship awarded and funded by the Australian Teaching and Learning Council.

8. REFERENCES

[1] 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.

[2] Lister, R., Fidge C. and Teague, D. (2009) *Further Evidence of a Relationship between Explaining, Tracing and Writing Skills in Introductory Programming*. 14th Annual Conference on Innovation and Technology in Computer Science Education, Paris, France.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] Soloway, E., Bonar, J., and Ehrlich, K. (1983). Cognitive strategies and looping constructs: an empirical study. *Commun. ACM* 26, 11 (Nov. 1983), 853-860.

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

[9] Traynor, D., Bergin, S. and Gibson, J.P. (2006). *Automated Assessment in CS1*. In Proc. Eighth Australasian Computing Education Conference (ACE2006), Hobart, Australia. CRPIT, **52**. Tolhurst, D. and Mann, S., Eds. ACS. 223-228.

[10] 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.

[11] Wilson, B. (2002) A Study of Factors Promoting Success in Computer Science Including Gender Differences. *Computer Science Education*. Vol. 12, No. 1-2, pp. 141-164.