Relationships between reading, tracing and writing skills in introductory programming

Mike Lopez
Manukau Institute of Technology
Private Bag 94006
South Auckland Mail Centre
Manukau 2240
New Zealand
+64 9 968 8000
mike.lopez@manukau.ac.nz

Jacqueline Whalley,
Phil Robbins
Auckland University of Technology
Private Bag 92006
Auckland 1020
New Zealand
+64 9 921 9999
jacqueline.whalley@aut.ac.nz
phil.robbins@aut.ac.nz

Raymond Lister
University of Technology, Sydney
P.O. Box 123
Broadway, NSW 2007
Australia
+61 2 9514 2000
raymond@it.uts.edu.au

ABSTRACT
This study analyzed student responses to an examination, after the students had completed one semester of instruction in programming. The performance of students on code tracing tasks correlated with their performance on code writing tasks. A correlation was also found between performance on "explain in plain English" tasks and code writing. A stepwise regression, with performance on code writing as the dependent variable, was used to construct a path diagram. The diagram suggests the possibility of a hierarchy of programming related tasks. Knowledge of programming constructs forms the bottom of the hierarchy, with "explain in English", Parson's puzzles, and the tracing of iterative code forming one or more intermediate levels in the hierarchy.

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
The nineteen eighties was a period of extensive study of novice programmers. To name just two of the many studies from that time, Soloway et al. (1983) found that that only 38% of early computer programming students could write a program to calculate the average of a set of numbers, while Perkins and Martin (1989) reported that students had fragile knowledge of basic programming concepts and a "shortfall in elementary problem-solving strategies". At the end of that decade, an entire volume of papers, "*Studying the Novice Programmer*" documented many of the difficulties of learning to program (Soloway and Spohrer, 1989).

At the turn of the millennium, an ITiCSE 2001 working group, the "McCracken Group", assessed the programming ability of a large set of students, from four universities across two countries (McCracken et al., 2001). Each student was required to write a program from a related set of program-writing tasks. Almost all students performed poorly on their task and many students did not even get close to finishing the task. The nature of the experiment did not allow the McCracken group to make firm conclusions as to why the students struggled, but they speculated that it was due to the students having a weak capacity to problem-solve. That is, as the McCracken group defined problem-solving, the students were weak at an iterative five step process: (1) Abstract the problem from its description, (2) Generate sub-problems, (3) Transform sub-problems into sub-solutions, (4) Re-compose, and (5) Evaluate and iterate.

At the ITiCSE 2004 conference held in Leeds, the only working group conducted that year (hence its name, the "Leeds Group") conducted an experiment designed to challenge the speculation that the results from the McCracken Group were due to students being weak in problem-solving (Lister et al., 2004). The Leeds group studied student performance on programming-related tasks that did not require problem-solving. The students were required to answer several multiple choice questions. The questions were of two types, "fixed code" questions and "skeleton code" questions. In fixed code questions, students were given a piece of code and were required to identify the value in a variable after the given code had finished executing. Answering such a question requires a student to understand all the constructs in the code and also requires that they be able to systematically hand execute ("trace") through code. In skeleton-code questions, students were given a piece of code with one or two lines missing, they were also told what the code should do, and they were then required to identify the missing lines of code (as the question was multiple choice, the students did not have to write the lines, but merely identify the correct code among four options). While many students performed well on these questions, approximately 25%

of the students performed at a level consistent with guessing. In the concluding remarks of their working group report (Lister et al., 2004), the Leeds Group wrote:

> "*We accept that a student who scores well on the type of tests used in this study, but who cannot write novel code of similar complexity, is most likely suffering from a weakness in problem solving. This working group merely makes the observation that any research project that aims to study problem-solving skills in novice programmers must include a mechanism to screen for subjects weak in precursor, reading-related skills*."

In this concluding remark, the Leeds group position the ability to read code (of a given complexity) as a precursor skill to the ability to write code (of a similar complexity). In positing the existence of such a precursor skill, the Leeds Group opened the possibility of there being a multi-level hierarchy of programming skills. The Leeds Group data already indicates two possible levels below code writing, since students were less successful at the skeleton-code questions in that study than the fixed code questions.

The BRACElet project has since built upon the results of the Leeds Group, by probing for intermediate levels in a hierarchy of programming-related skills (Whalley, et al., 2006). The initial research instrument designed by BRACElet members duplicated some of the Leeds Group fixed-code questions. The instrument also contained some other questions that were designed by a more systematic, theoretically grounded, approach than the ad hoc methods used by the Leeds Group. The revised Bloom's taxonomy (Anderson et al., 2001) and the SOLO taxonomy (Biggs and Collis, 1982) were used by BRACElet participants as a cognitive framework for assigning levels of difficulty to the tasks in the instrument (Whalley, Clear and Lister 2007; Whalley and Robbins 2007; Thompson et al. 2008). From data collected in this initial BRACElet study, it was established that students did find the tasks from higher levels of the framework harder than those tasks assigned lower levels (Whalley, et al., 2006).

In this first BRACElet study, one of the tasks required students to "*In plain English, explain what the following segment of Java code does*". Student responses were analysed in terms of the SOLO taxonomy. It was found that some students responded with a correct, line-by-line description of the code (which is, in terms of SOLO, a multi-structural response) while other students responded with a correct summary of the overall computation performed by the code (which is, in terms of SOLO, a relational response). Furthermore, the better a student performed on other BRACElet tasks, the more likely the student was to give a relational response to the "explain in plain English" question. Also, when the same "explain in plain English" question was given to academics, they almost always offered a relational response. The BRACElet group (Lister et al. 2006) concluded that the ability to read a piece of code and explain it in relation terms ─ that is, to see the forest and not just the trees ─ is an intermediate skill on the hierarchy of programming-related skills. Their conclusion is consistent with earlier literature on the psychology of programming (Adelson 1984, Corritore and Weidenbeck 1991, Wiedenbeck, Fix and Scholtz 1993).

Philpott, Robbins and Whalley (2007) presented further data suggesting that code tracing is a precursor skill to relational thinking in "explain in plain English" questions:

> "*The green light for relational thinking would seem to be a complete mastery of the code tracing task. A better than 50% performance on the tracing task could be viewed as an orange light. However if tracing ability is at a lower level than 50% then … the light is definitely red when it comes to relational thinking*."

1.1 Our Study

The ultimate aim of research investigating novice programmers is to improve the practice of teaching novice programmers. But the relationship between research and practice need not be one way, from research to practice. With suitable preparation, practical teaching activities can also be data collection activities for research. While the Leeds Group data collection was mostly carried out independently of teaching activities, the multiple choice questions used in that study were all taken from exam papers previously used by one of the project participants. Also, most of the data collected in the BRACElet studies has come from questions incorporated into end-of-semester exams.

In this paper, we analyze an end-of-first-semester examination given to students at a single institution. Some of the questions in this exam were designed to build upon the results from the Leeds group and BRACElet studies. From a research perspective, the exam was intended to further investigate the notion of a hierarchy of programming-related skills. The specific research questions addressed in this study are:

- Is tracing skill associated with writing ability?
- Is reading skill (i.e. "explain in plain English") associated with writing ability?
- Is the student performance on this exam consistent with a hierarchy of programming-related skills?

2. THE SAMPLE

The exam was undertaken by 78 students at the end of a first semester of programming in Java. Thirty eight of those students gave the institutionally required approval for their exam work to be used as research data. This subset of the students provided a reasonable grade spread which reflected the class distribution as a whole. The average mark of the subset was slightly higher than that of the class as a whole (62% versus 58%). Students were given two hours to complete the 28 page exam.

The students attended common lecture sessions but were divided into practical programming laboratory classes taught by four different tutors. The exam paper was set mainly by two of those tutors, with the others providing feedback and proof reading. The exam was strip marked by all four tutors ─ each question, for all papers, was marked by one of the tutors and subsequently moderated by the rest of the teaching team.

3. THE INSTRUMENT

The exam described here is a result of 3 years of refinement and research informed assessment design. In this section of the paper, we present the questions that comprised the exam, and also provide simple statistics (e.g. average, quartiles) describing the mark distribution for each question. Table 1 presents the complete set of simple descriptive statistics. The purpose of this section is to familiarize the reader with the exam questions before the more sophisticated statistical analysis, including the path analysis, which is described later in the paper.

Table 1: Descriptive statistics for each exam question and the statistical variable to which each question is assigned

Exam Question No.	Statistical Variable	Possible Mark	Average Mark (absolute)	Average Mark (percentage)	Third Quartile	Median	First Quartile
1	Basics	7	5.2	75%	6.8	6.0	4.0
2	Basics	7	6.4	91%	7.0	7.0	6.3
3	Basics	4	3.7	93%	4.0	4.0	3.6
4	Basics	8	5.3	66%	6.9	5.5	4.1
5	Sequence	5	4.4	88%	5.0	4.5	4.5
6	Sequence	6	4.9	83%	6.0	6.0	4.0
7, A-Ei	Tracing	15	10.1	67%	12.8	10.0	9.0
7, Eii	Exceptions	2	0.9	47%	1.4	1.0	0.5
8	Data	10	7.8	78%	9.0	8.0	7.0
9	Writing	6	1.8	29%	3.6	0.8	0.0
10	Explain	8	3.2	40%	4.8	3.0	2.0
11	Writing	6	2.7	45%	6.0	2.0	0.0
12	Exceptions	6	1.5	25%	3.0	0.5	0.0
13	General	10	4.7	47%	6.0	5.0	4.0
Total		100	62.7	63%	69.9	62.8	54.6

Table 2: Descriptive statistics for types of questions (i.e. variables)

Average Mark Rank	Type (i.e. variable)	Possible Mark	Average Mark (absolute)	Average Mark (percentage)	Third Quartile	Median	First Quartile
2	Basics	26	20.6	79%	24.4	21.3	18.6
1 (easiest)	Sequence	11	9.3	85%	11.0	10.5	8.5
4	Tracing1 (non-iterative)	10	7.7	77%	10.0	8.0	6.0
5	Tracing2 (iteration)	5	2.4	48%	3.8	2.0	1.0
9 (hardest)	Exceptions	8	2.4	30%	4.0	1.8	0.6
3	Data	10	7.8	78%	9.0	8.0	7.0
8	Writing	12	4.4	37%	7.4	3.5	1.0
7	Explain	8	3.2	40%	4.8	3.0	2.0
6	General	10	4.7	47%	6.0	5.0	4.0

Prior to any analysis, the authors and colleagues placed the 13 exam questions into 8 categories, based upon their teaching experience and the earlier work of the BRACElet project. Those 8 categories are *Basics, Sequence, Tracing, Exceptions, Data, Writing, Explain* and *General*. During analysis, it became apparent that tracing should be broken into two categories, *Tracing1 (non-iterative)* and *Tracing2 (iteration)*.

3.1 Basics (Questions 1-4)

These questions required students to identify Java constructs, to recognize the definition of common Java terms and to detect syntax errors. These questions emphasized recall of knowledge.

3.1.1 Question 1: Matching Terms to Definitions

The first question presented the student with 7 terms: *assignment, compiler, constructor, debugger, method, overloading,* and *variable*. Also, 7 definitions were presented, 2 of those being "*Translates source code into object code*" and "*Code called when an object is created*". Students were required to match the terms to definitions.

Each correct match was worth 1 mark. The general performance of the students was good. The average mark was 5.2 out of 7 (75%).

3.1.2 Question 2: Matching Terms to Code

This question was another exercise in matching. Students were presented with a page of code, a complete class definition, with a single constructor, a single accessor method, a single mutator method, and a "print" method that used System.out.println to output some private data members. Ten of the lines of code were annotated with an alphabetic character, "A" to "J". Students were also presented with 7 definitions/descriptions of code, including "*the name of an accessor method*" and "*signature of constructor*". Students were required to match the definitions/descriptions to the lines of code annotated "A" to "J".

Each correct match was worth 1 mark. In general, students did well on this question. The average mark was 6.1 out of 7 (91%).

3.1.3 Question 3: Method Headers

This question presented students with 4 method headers (i.e. Java code). Students were required to provide the number of parameters and the return type of each header. Each header was worth 1 mark. The students performed well on this task, with an average mark of 3.7 out of 4 (93%).

3.1.4 Question 4: Syntax Errors

In this question, students were required to find 8 syntax errors in a page of code, which was a complete class definition. Students were told to find 8 syntax errors, and there were 11 syntax errors in the code. The class contained 5 private data members, of type String, double and ArrayList. It also contained a single constructor, 5 simple accessor/mutator methods, and another method that deleted a specific element from an ArrayList.

Students were awarded 1 mark for each correctly identified syntax error. Student performance was fair on this task, especially as there were 11 syntax errors available, with an average mark of 5.3 out of 8 (66%).

3.1.5 Overall Performance on Basics

As a whole, the 38 students demonstrated mastery of these "Basic" tasks , with an average mark of 21 out of 26 (79%). Three quarters of the students scored 18 or higher (69%). In some other questions presented below, the class as a whole does much worse, and these first four "Basics" questions establish that poor overall performance on subsequent questions is not due to a poor overall grasp of these basics.

3.2 Sequence (Questions 5-6)

3.2.1 Question 5: Missing Lines

This question begins with a preamble, "*The code below is from one of the BlueJ projects you have used this semester. You will note that it uses an ArrayList. Some of the code has been removed*." The subsequent code is a class definition, taking up a page and a half, containing two private data members (one of which is an ArrayList), a single constructor, and 3 methods, which add to, delete from, and print the contents of the ArrayList. Below the code, the students are set the following task: "*The table below shows the missing lines of code, but not necessarily in the correct order. It also has one extra line of code that is not needed. Identify which line of code should go where ...*"

This question is like the skeleton code questions from the Leeds group (Lister et al. 2004), but in the Leeds study the task was simpler, as there was usually a single missing line in the code and four alternative options were provide for that missing line. On the other hand, as the preamble indicated, the students had seen this code before, which would make the task easier.

Students were awarded 1 mark for each line of code correctly placed in the 5 available positions. They did very well on this task, with an average mark of 4.4 out of 5 (88%).

3.2.2 Question 6: Parsons Puzzle

A Parsons Puzzle (Parsons and Hayden, 2006) is the extreme case of the previous type of question. It requires students to take a set of lines of code, presented in random order, and place those lines into the correct order to perform a given function. The puzzle we used in this exam is given in Figure 1.

These puzzles require students to apply their knowledge of the common patterns in basic algorithms, apply some some heuristics (e.g. initialize a variable before using it) and perhaps also, to a degree, manifest some design skill. Parson and Hayden claimed that these puzzles require skills between code reading and code writing. No empirical study, before now, has been undertaken to investigate their claim.

We had not used a Parsons Puzzle in any previous exam, nor had we shown students any examples of these puzzles prior to the exam. The code used in this question was adapted from an "explain in plain English" question from a previous exam, but we strongly suspect that few students would have seen that previous exam prior to taking this exam.

We were surprised at how well the students performed on this question, with an average mark of 4.9 out of 6 (83%). Three quarters of the students scored at least 4, and a mark of 4 implies that the answer would be perfect if two lines swapped places.

In retrospect, given how well the students performed on this question, students may have been able to correctly place many of the lines of code by applying shallow heuristics. For example, the heuristic "always place a return statement at the bottom" would work in this case. (In the Leeds group study, students had difficulty with a question where a "return" did not occur at the bottom.) Also, by providing the placement of the braces, we may have given away too much information, as only lines A, B and E could reasonably precede the three opening braces. Perhaps the same question would have been harder had we not provided braces and instead told the students to add braces where required.

3.3 Tracing1 – Non-iterative (Quest. 7, A-C)

Parts A, B and C of Question 7 were very simple tracing tasks. Part A (3 marks) required students to nominate the values in three variables after they were initialized and then altered by three assignment statements `a += 2;` `b -= 4;` `c = b * a;` Part B (4 marks) required students to calculate the value of the expression `num1 % num2` for two different sets of values of the two variables. Part C (3 marks) required students to calculate the value of the variable "`bValid`" for the following code:

```
boolean bValid = false;

if (iValue >= FIRST_VAL && iValue < SECOND_VAL)
{
        bValid = true;
}
```

for three different sets of values of the variables, `iValue`, `FIRST_VAL` and `SECOND_VAL`.

Here are some snippets of code that, when used in the correct order, would make up a method to count the occurrences of a letter in a word (e.g. how many times does the letter 'm' appear in the word Programming?).

A. if(sWord.charAt(i) == toCount)
B. for(int i = 0; i < sWord.length(); i++)
C. return count;
D. int count = 0;
E. public int countLetter(String sWord, char toCount)
F. count++;

Each box below represents a placeholder for one of the lines of code above. Each line of code must be placed in only 1 of the boxes. Indicate which line of code goes in which box by writing its letter (A to F) in the appropriate box.

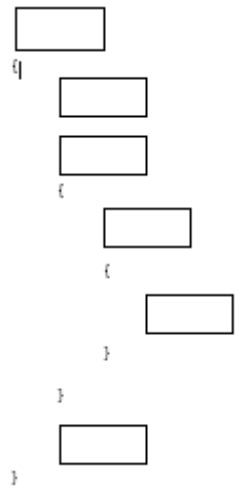

Figure 1: Question 6, the Parsons Puzzle

The average combined mark on these three simple tracing tasks was 7.7 out of 10 (77%), with three quarters of the students scoring 6 or higher. Overall performance was lowest on part B, perhaps because students did not know the "%" operator. The average mark on parts A and C combined was 5.3 out of 6 (88%), which is a higher average mark percentage than other question type in Table 2.

3.4 Tracing2 – Iteration (Questions 7, D-E(i))

3.4.1 Question 7D: Tracing a while loop

Students were presented with the code shown in Figure 2, and were asked to provide the value returned by the method, for three different values of the parameter "`iLimit`", -1, 3 and 0. The collective performance of the students was mediocre, with an average mark of 1.8 out of 3 (60%). This result is consistent with the findings of the Leeds Group finding that many students struggle with tracing loops.

3.4.2 Question 7E(i): For loop containing an "if"

Students were presented with the code shown in Figure 3, and were asked to provide the value returned by the method, for two different values of the parameter "aNumbers", {1, 2, 3, 4, 5} and {20, -10, 6, -2, 0}. The collective performance of the students was poor, with an average mark of 0.6 out of 2 (32%).

As shown in Table 2, the performance of the students on these two loop tracing tasks combined was mediocre, with an average mark of 2.4 out of 5 (48%)

```
public int q7D(int iLimit)
{
        int iIndex = 0;
        int iResult = 0;

        while (iIndex <= iLimit)
        {
                iResult += iIndex;
                iIndex++;
        }
        return iResult;
}
```

Figure 2: Question 7D, a while loop

```
public int q7E(int[] aNumbers)
{
    int iResult = 0;

    for(int iIndex=0; iIndex<aNumbers.length; iIndex++)
    {
        if(aNumbers[iIndex] > iResult)
        {
            iResult = aNumbers[iIndex];
        }
    }
    return iResult;
}
```

Figure 3: Question 7E(i), a for loop containing an "if"

3.5 Exceptions (Question 7 E(ii) and 12)

Two questions required students to manifest some understanding of the concept of an exception. The first of these, question E(ii), used the code given in Figure 3, and asked the students what would happen if the method was called thus: `q7E(null);` The collective performance of the students was mediocre, with an average mark of 0.9 out of 2 (47%).

The second question on exceptions, question 12, comprised two parts of equal marks. The first provided students with a half-page of code, which manipulated a simple data structure. The students were then told: "*The code contains an error - it compiles and runs, but sometimes stops with an* Index OutOfBoundsException. *Explain why this problem occurs*." A suitable answer would be that the "if … else if … else …" guards failed when the data structure was full and an attempt was made to access the highest element in the structure. The second part of question 12 was analogous to the first part, but involved object references and a NullPointerException. Most students gave poor answers to both parts, with an average mark of 1.5 out of 6 (25%).

3.6 Data (Question 8)

This question tested the students on their knowledge of data types, and consisted of four parts. In the first part, students were presented with 5 data types: *ArrayList, Boolean, double, int* and *String*. The students were also given 5 descriptions of data, including "*The name of a student*" and "*Whether a person is married or not*". Students were required to choose the "most appropriate" data type for each description.

In the second part of question 8, students were asked to underline the data type in the code: `public Server host;`

In the third part, students were asked to write "*the declaration for a variable that is to be visible only within the current class, whose name is* `representative` *and which is of type* `Student`."

In the fourth part, students were tested on their understanding of scope. Students were provided with a piece of code from their text book, a complete class about half a page long, containing some private data members, a constructor and an accessor, called "showPrice". They were also told that a small error had been introduced into the code, and that "*The code compiles, but does not work as expected. Whatever value is passed to the constructor, when showPrice, is called it displays:* `The price of a ticket is 0.0 cents`. *What is the problem*?" The error was that, inside the constructor, an assignment statement intended to update the private data member "`price`" was incorrectly written as `double price = ticketCost`; thus the local variable "`price`" was updated, not the private data "`price`".

The 38 students showed a solid grasp of this material, with an average mark of 7.8 out of 10 (78%). Three quarters of the students scored 7 or higher.

3.7 Writing (Questions 9 and 11)

3.7.1 Question 9: Write your Hip Hop name

Students were asked the following: "*Write a java method that generates your hip hop name. Your hip hop name starts with DJ. This is followed by your pet's name, followed by the first 3 characters of your first name and the first 2 characters of your mother's maiden name*." The student's were also provided with the following information about methods they could use to produce their Hip Hop Name

- `char charAt(int index)` Returns the char value at the specified index.
- `int length()` Returns the length of this string.
- `String[] split(String regex)` Splits this string around matches of the given regular expression.
- `String substring(int beginIndex)` Returns a new string that is a substring of this string.
- `String substring(int beginIndex, int endIndex)` Returns a new string that is a substring of this string.

The students performed poorly on this task, with an average mark of 1.8 out of 6 (29%).

3.7.2 Question 11: Zombie Task Force Test Driver

Students were given documentation in, JavaDoc format, for methods that managed a "Task Force" of instances of the class "Zombie", including these methods:

- `void addZombie(Zombie oNewMember)` Allows a Zombie to join the task force
- `int countZombies()` Counts the number of Zombies in the task force
- `void deleteZombie(Zombie oMember)` Allows a Zombie to be removed from the task force
- `boolean findZombie(java.lang.String sName)` Search for an Zombie by name

The students were then asked to write a single line of code to perform each of the following tasks:

- Test that there are initially no Zombies in the group.
- Add the Zombie named "Scrog"
- Check that there is only one Zombie in the group.
- Check that "Scrog" can be found in the group.
- Check that "Blob" cannot be found in the group.
- Remove "Scrog"
- Check that there are now no Zombies in the group.

The collective performance of the students was mediocre, with an average mark of 2.7 out of 6 (45%).

3.7.3 Marking and Prior Exposure

When the tutors marked these two writing questions, they were instructed to ignore small syntactic errors. Java documentation was supplied for any Java library methods that might be useful to the students. Question 9 was very similar to an exercise given to the students in the month before the exam. Unit testing, as in Question 11, had been featured throughout the semester. Despite such favorable circumstances, the average aggregate mark on these two writing questions was only 4.4 out of 12 (37%). One quarter of the students scored 1 mark or less.

3.8 Explain (Question 10)

This question comprised 3 parts, worth 2, 3 and 3 marks respectively. In each part, students were presented with code and told to "*explain in plain English what it does*". The code in each of the first two parts is shown in Figures 4 and 5 respectively. The code in the third part was longer, and implemented binary search on an array of integers.

Unlike earlier studies using this type of question (Whalley, et al., 2006; Lister, et al., 2006), these students had seen this type of question before the exam. Similar questions had been presented during lectures and on previous exam papers, with model answers. However, the three pieces of code used in this exam had not been presented to students prior to the exam, so answers could not have been memorized. Students knew, that a line-by-line description of the code (i.e. a SOLO multi-structural response) was a low scoring answer while a high scoring answer was a summary of the overall computation performed by the code (i.e. a SOLO relational response). The preamble to question 10 reminded students: "*Note that more marks will be gained by correctly*

explaining the purpose of the code than by giving a description of what each line does".

Despite their prior experience, the student performance on these questions was mediocre, with the average aggregate mark on all three parts being only 3.2 out of 8 (40%).

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

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

Figure 4: the code for Question 10A, a reading question.

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

Figure 5: the code for Question 10B, a reading question.

3.9 General (Question 13)

The final exam question instructed students to "*Select ONE of the following and write an answer in clear English. 4 to 5 paragraphs will be expected*". The four topics from which students could choose were:

- *"My code works so it must have been well written". With reference to the concepts of coupling and cohesion, discuss how to design programs which not only work, but are easy to modify.*
- *Arrays and ArrayLists are both data structures that you have studied in Programming 1. Describe the main differences and similarities between these data structures. Credit will be given for examples of where you have used them in any of the code you have written this semester.*
- *Why do we need to test code? Describe how you tested the code you wrote for your assignment this semester. How did your testing help you to write correct, working code?*
- *Describe the various standards and formatting techniques recommended in Programming 1, and explain how each of these improves the quality of the code and makes a programmer more productive when they are used.*

This question clearly tests knowledge and skills quite different to those skills tested in other questions.

4. METHOD

In the previous section, the questions in the exam / research instrument were described in detail. The description included simple descriptive statistics indicating the distribution of students marks on each question and on each type of question. In this section, we present a more sophisticated statistical analysis, which examines the relationships between the exam questions, and which provides evidence (for or against) a hierarchy of programming-related skills.

4.1 Analysis Approach

When marks or scores are allocated to questions, the marker is generally confident in the ordinal properties of the marks (i.e. 4 is better than 3, 3 is better than 2, etc.). There is less confidence in the interval properties of the marks. For example, does the difference between 3 and 4 represent the same difference in knowledge or skills as that between 2 and 3? However, interval level measurement is required for valid arithmetic on variables.

To address this, a polytomous Rasch model (Rasch, 1960; Andrich, 1978) was used to create interval level variables from the marks; this is a stochastic model that identifies the maximum likelihood estimates of person and item threshold locations by simultaneous modeling of location estimates and the uncertainty in their location. Figure 6 shows an illustrative probability density map (for question 6) showing the probability of marks 1, 2, 3, 4 and 6 being awarded for any given ability with the ability scale standardized to the range 0 to 10 and centred at 5.

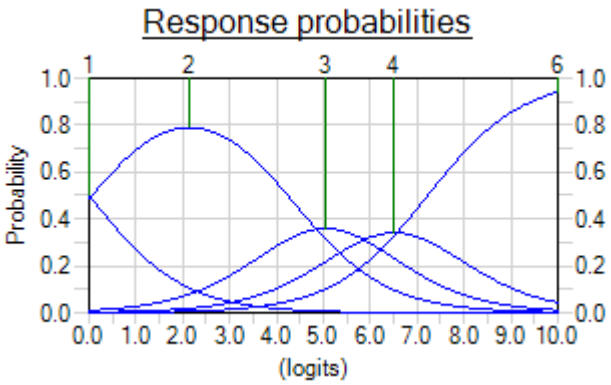

Figure 6: Sample response probabilities for Question 6, the Parsons Puzzle

From this diagram, we can see that for abilities between 0 and 4.7, the most likely mark is 2, from 4.7 to 5.8, the most likely mark is 3, from 5.8 to 6.6 it is 4, and above 6.6 it is 6. This diagram illustrates both the varying intervals associated with the marks, and the clear ordinal pattern. We can also see the uncertainty in the marks (e.g. for imputed ability 1, the most likely mark is 2, but there is about 30% chance of 1 being awarded).

A key assumption of the model is that the construct being measured is one-dimensional; this was verified with a conventional principal components analysis.

There are many reasons for variability in exam marks; in order to examine the potential structural relationships between the various constructs, our first step was to attempt to partition the variability in the dataset into that associated with the identified programming constructs and that associated with more general personal traits and attributes. The general variable (i.e. Question 13) was used as an estimate of the latter, establishing a base level against which other estimates could be compared. We chose this question because the ability to answer it does not seem to be directly related to the key constructs examined in this study. It should be noted however, that the use of a single question rather than a scale is unable to capture the full range of personal factors.

We partitioned the variability by carrying out a set of bivariate regressions between the general variable and all others and then using the coefficients from these regressions to subtract the general effect from each remaining variable. Using the general variable as a covariate in this way also improves generality by controlling for the bias introduced by limiting the sample to those giving consent. This process resulted in seven variables, each representing the unique contribution made by that variable over and above the base level. The first two research questions (i.e. "*Is tracing skill associated with writing ability*?" and "*Is reading skill (i.e. explain in plain English) associated with writing ability*?" were then addressed by a hypothesis of positive correlation between the appropriate variables.

A technique of step-wise multiple regression was used to address the third research question (i.e. *Can we elaborate the relationships between these and other constructs*?). With this technique, writing was used as the criterion variable and all other variables were initially introduced into the model as potential predictors. At each step, the variable with the smallest unique contribution was removed until we reached the three stopping conditions of (1) maximum adjusted R^2, (2) a significant (at .01) overall regression, and (3) a significant (at .05) contribution from each variable.

Having identified the significant predictors of writing variance, the process was then repeated taking each of these predictors in turn as the criterion variable and searching for significant predictors among the remaining variables. This process was repeated for each identified predictor to build a path diagram of potential chains of association.

Regression residuals were tested for normality, homoscedasticity and serial independence with a Jarque-Bera test (Bera and Jarque, 1980).

5. RESULTS

In this section, we begin by setting out the results of the data screening process and then address the results of the research questions. The changes made to the procedures *a posteriori* are addressed under the appropriate research questions.

5.1 Data screening

All constructs were uni-dimensional with the exception of tracing which appears to have two underlying factors. Question 7, parts A, B and C loaded strongly on the first factor, and parts D and E(i) on the second factor. The main difference between these parts appears to be that the first three parts involved tracing a single sequential pass through code and the last two involved a repetition structure. The factor loadings, with separation optimized by a varimax rotation, are summarized in Table 3.

Table 3: Tracing factor loadings

Part	Characteristic	F1	F2
A	sequence	**0.5335**	0.0579
B	sequence	**0.6901**	-0.0320
C	non-iterative	**0.7617**	0.2836
D	repetition (while)	-0.0272	**0.9030**
E(i)	repetition (for)	0.1236	**0.8691**

Because of the two factors, two additional variables were created: Tracing1 (non-iterative) and Tracing2 (iteration) and these two were used for the elaboration of the third research question.

The "General" variable had a mean of 48% and a standard deviation of 29%. Only two students received a zero mark suggesting that, although it was the last question in the exam, time pressure was not a major barrier to students attempting the question.

All of the derived variables showed positive inter-correlations with the exception of "Data" vs. "Explain" which had a small (but not significant) negative correlation. The mean inter-correlation between all variables was 0.3864.

5.2 Tracing skill

To address the proposition that program writing skill is associated with tracing skill, we tested the hypothesis that the correlation between tracing and writing is positive. A Pearson correlation analysis produced a correlation of 0.5621 between (Tracing) and (Writing). This was significant at the 0.01 level ($r_{(36)}$=0.5621; p=0.0001), and positive. The relationship accounts for 32% of the variability; adjusted R^2 was 30%. The regression equation is "**Writing = 0.9728 • Tracing - 3.1803**"; the confidence interval of the coefficient is: $CI_{.99}$ = (0.3269 ≤ Tracing ≤ 1.6187). A Jarque-Bera test of normality indicates (p=0.9794) that the distribution of the residual from the regression is acceptably close to a normal distribution, which suggests that a parametric approach is appropriate.

Because two underlying factors were associated with the tracing variable, separate tests were then made for each of these. No significant correlation ($r_{(36)}$=0.3028; p=0.0308) was found between Tracing1 (non-iterative) and Writing. There was a significant positive correlation between Tracing2 (iteration) and Writing ($r_{(36)}$=0.6267; p<.0001). The relationship accounts for 39% of the variability; adjusted R^2 was 38%. A Jarque-Bera test of normality indicates (p=0.3947) that the distribution of the residual from the regression is acceptably close to a normal distribution, which suggests that a parametric approach is appropriate.

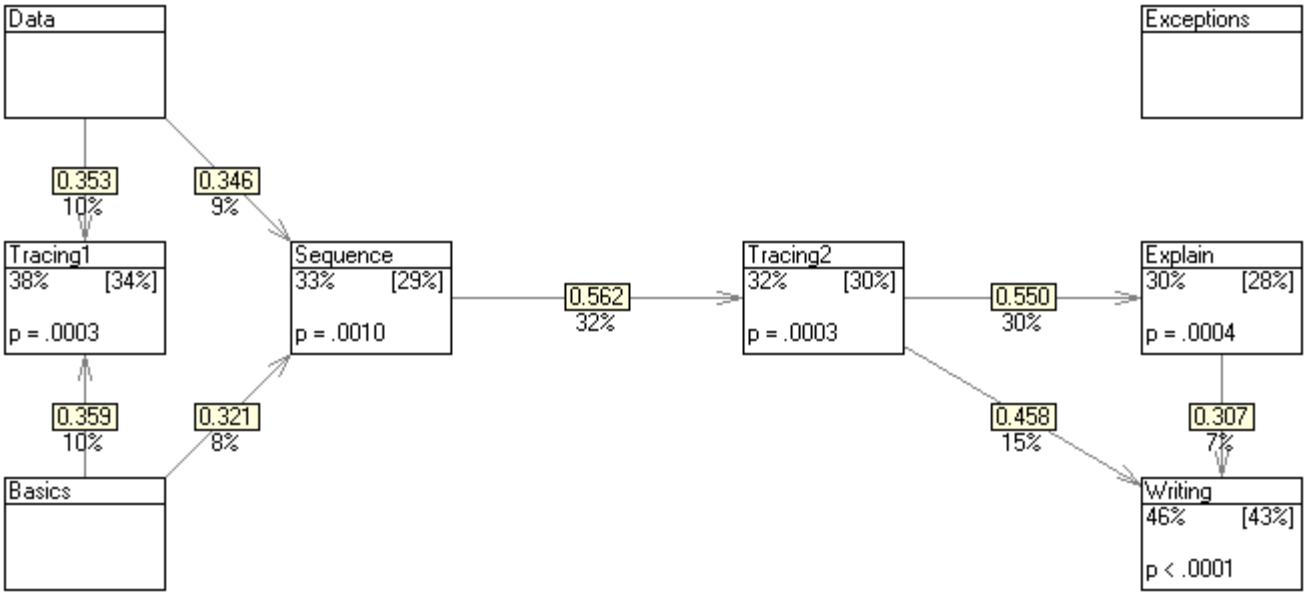

Figure 7: The Path Diagram

5.3 Reading ("Explain") skill

To address the proposition that program writing skill is associated with program reading skill, we tested the hypothesis that the correlation between the ability to explain code and writing was positive. A Pearson correlation analysis produced a correlation of 0.5586 between (Explain) and (Writing). This was significant at the 0.01 level ($r_{(36)}$=0.5586; p=0.0002), and positive. The relationship accounts for 31% of the variability; adjusted R^2 was 29%. The regression equation is: "**Writing = 0.6100 • Explain + 1.1772**"; the confidence interval of the coefficient is: $CI_{.99}$ = (0.2013 ≤ Explain ≤ 1.0187). A Jarque-Bera test of normality indicates (p=0.3661) that the distribution of the residual from the regression is acceptably close to a normal distribution, which suggests that a parametric approach is appropriate.

5.4 Elaboration

The stepwise regression procedure identified the relationships shown in Figure 7. In this path diagram, variables are shown in a titled box containing the variance explained, adjusted R^2 in square brackets and the significance of the overall regression. The boxes on the paths show the beta weights of the paths with the semi-partial correlation squared shown underneath. This last represents the unique contribution made to the explanation of the criterion variable over and above that shared with other predictor variables. The direction of the arrows should not be interpreted as evidence of causation (although that is of course plausible).

A degree of multicollinearity is to be expected in any valid assessment instrument, but has the consequence that estimates of the relative contribution of each predictor to a criterion have an increased confidence interval, even though the overall regression remains stable. In this dataset, the variance inflation factor attributable to multicollinearity in the regression ranged from 1.30 to 1.43. Although this is moderate, it should be noted that at this sample size, the relative contribution of predictors is subject to considerable uncertainty.

6. DISCUSSION

With regard to the first two research questions addressed in this study:

- We found strong support for an association between code tracing and code writing skills, particularly when the tracing involved loops (i.e. Tracing2). The correlation between Tracing2 and Writing was 0.6267 (p<.01), accounting for 39% of the variability, with an adjusted R^2 of 38%.
- We found support for an association between code reading (i.e. "Explain in plain English") and code writing skills, with a correlation 0.5586 (p<.01) accounting for 31% of the variability, and an adjusted R^2 of 29%.

6.1 Evidence for a hierarchy?

This subsection explores the third research question addressed in this study, by exploring evidence in this exam for a hierarchy of programming-related skills.

As with any statistical analysis, the co-variances identified in this study are not, alone, evidence of causality. However, if we begin by positing a causal model, then a subsequent statistical analysis can increase our confidence in that model by manifesting co-variances consistent with the model. Prior to undertaking the statistical analysis (indeed, prior to setting the exam), we had our own intuitions as to how the hierarchy of programming-related skills was structured. This subsection explores whether our intuitions, and past literature, about such a hierarchy are consistent with the path diagram that emerged from the statistical analysis.

6.1.1 Basics and Data

We begin at the bottom of our hypothetical hierarchy. Given the influence upon us of the revised Bloom's Taxonomy (Anderson et al., 2001) we believe knowledge of basic programming constructs forms the bottom of the hierarchy. This is consistent with "Basics" and "Data" appearing at the bottom of the path diagram. Taken together, these two variables are strong predictors of both elementary tracing skills (Tracing1) and the Sequence tasks.

6.1.2 Tracing1 and Sequence

We were surprised that there isn't a statistically significant relationship between Tracing 1 and Tracing 2. In retrospect, perhaps the Tracing1 tasks in his exam were too simple to show a relationship with Tracing2. If that is so, then characterizing the difficulty of a task simply by its nature alone may be insufficient. The difficulty of a task may also be a function of its size, and perhaps also the programming constructs involved. Such a multidimensional view of difficulty may be consistent, in principal, with the two-dimensional revised Bloom's Taxonomy (Anderson et al., 2001).

Prior to the statistical analysis, we suspected that the Sequence tasks were intermediate, and the path diagram is consistent with that suspicion. However, we were surprised that the Sequence variable appeared lower in the path diagram than Tracing2, as our intuition is that the Sequence tasks ─ both completing skeleton code and Parsons Puzzles ─ are a higher skill than tracing. However, as we discussed earlier, perhaps the particular Parsons puzzle we used was too easy. As discussed in the previous paragraph, with regard to Tracing1 and Tracing2, perhaps there are other characteristics of a task, apart from its nature, that determine its level of difficulty.

6.1.3 Tracing2 and Explain

The relationship between Tracing2 and Explain in the path diagram is consistent with Philpott, Robbins and Whalley's (2007) "traffic light" conjecture ─ the green light for relational thinking is mastery of code tracing.

In combination, Tracing2 and Explain account for 46% of the variance in Writing (a percentage considered good by many in the social sciences, given a preponderance of other factors in the messy lives of people). This is consistent with our prior intuition, and also consistent with prior literature, that Tracing and Explain are intermediate skills. However, we were surprised that Explain alone accounts for a smaller amount of the variance in Writing than Tracing2 alone. Again, perhaps there are other characteristics of a task, apart from its nature, that determine its level of difficulty. Also, perhaps programming-related skills do not form a strict hierarchy, but instead skills from more than one lower level may influence the performance of a skill at some higher level of the hierarchy. And perhaps, more prosaically, the reading tasks in our exam may not have been very effective assessment tasks.

6.1.4 Exceptions and other issues

In retrospect, we are not surprised that Exceptions is unconnected in the path diagram. The particular writing tasks in this exam did not require a grasp of exceptions.

This exam is a reflection of our hybrid times, with its mixture of object-oriented concepts and classic 3GL control structures. The writing tasks emphasized messages to objects (or procedure calls, in 3GL terms), but both the Tracing2 and Explain tasks emphasized an understanding of the code that occurs within methods, particularly control structures. It may therefore be surprising that Tracing2 and Explain account for 46% of the variance in Writing. Perhaps the strong combined connection of the Tracing2 and Explain tasks to the Writing tasks is less a reflection of specific programming knowledge, and more a reflection of generic reasoning skills required in these tasks.

7. CONCLUSION

For most academics, an exam paper is an instrument for assigning grades to students. From a computing education research perspective, an exam paper is also a research instrument, an opportunity to study the knowledge, skills and learning of novices. In this paper we have used an exam paper as a research instrument to explore the concept of a hierarchy of programming-related skills. We found statistical evidence which is consistent with our prior intuition, and the prior literature, on the structure of such a hierarchy.

There were also unanticipated results in our analysis, such as indications that solving Parsons Puzzles might be a lower skill than tracing iterative code. However, there may be other characteristics of a task, apart from its nature, that determine its level of difficulty. As we (and we hope others) design and analyse future exams, such issues and anomalies will be resolved.

Are there limitations to using exams as a research instrument? For example, to better understand the relationship between "explain in plain English" tasks and code writing, perhaps we need to have each student complete 20 explanation tasks, and write 20 short pieces of code. That is not possible in an exam, given the short time length of an exam, and the need to test students on wide spectrum of knowledge and skills. But that raises an interesting conundrum at the nexus of teaching and research ─ if an exam is too short to be a valid instrument for researching students, can it then still be a valid instrument for grading students?

8. ACKNOWLEDGMENTS

The authors thank Tony Clear and their other colleagues in the BRACElet project. Raymond Lister is an Associate Fellow of the Australian Learning and Teaching Council (ALTC). However, the views expressed in this paper are not necessarily the views of the ALTC.

9. REFERENCES

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.

Anderson, L. W., Krathwohl, D. R., Airasian, P. W., Cruikshank, K. A., Mayer, R. E., Pintrich, P. R., Raths, J., & Wittrock, M. C. (eds). (2001). *A taxonomy for learning and teaching and assessing: A revision of Bloom's taxonomy of educational objectives*. New York: Addision Wesley Longman Inc.

Andrich, D. (1978). A rating formulation for ordered response

categories. *Psychometrika*, 43, 561-73.

Bera, A., Jarque, C. (1980), "Efficient tests for normality, homoscedasticity and serial independence of regression residuals". *Economics Letters* 6 (3): 255–259. doi:10.1016/0165-1765(80)90024-5.

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

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

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.

Lister, R. On blooming first year programming and its blooming assessment. *Proceedings of the Australasian Conference on Computer Science Education*. ACM Press, 2000, 158-162.

Lister, R., Simon, B., Thompson, E., Whalley, J. and Prasad C. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. *ACM SIGCSE Bulletin,*38(3), 2006, 118 - 122

Parsons, D. and Haden, P. (2006). *Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses*. In Proc. Eighth Australasian Computing Education Conference (ACE2006), Hobart, Australia. CRPIT, **52**. Tolhurst, D. and Mann, S., Eds., ACS. 157-163.

Philpott, A, Robbins, P., and Whalley, J. Accessing The Steps on the Road to Relational Thinking. *Proceedings of the 20th Annual NACCQ*. Mann, S and Bridgeman, N. (eds), Nelson, NZ, July 8-11, 2007, 286.

Rasch, G. (1960/1980). *Probabilistic models for some intelligence and attainment tests.* (Copenhagen, Danish Institute for Educational Research), expanded edition (1980) with foreword and afterword by B.D. Wright. Chicago: The University of Chicago Press.

Soloway, E. and Ehrlich, K, Bonar, J., and Greenspan, J. (1983) *What do novices know about programming*? In B. Shneiderman and A. Badre, editors, Directions in Human-Computer Interactions, 27–53. Ablex, Inc., Norwood, NJ.

Spinellis, D. Reading, writing and code. *ACM Queue*, 1(7), 2003, 84 - 89.

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

Thompson, E., Luxton-Reilly, A., Whalley, J., Hu, M., & Robbins, P. (2008). *Bloom's taxonomy for CS assessment.* Proceedings of the Tenth Australasian Computing Education Conference (ACE2008), Wollongong, Australia, January 2008.

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. *Proceedings. of the Eighth Australasian Computing Education Conference (ACE2006)* (pp. 243-252). Hobart, Australia :CRIPT.

Whalley, J, Clear, T, and Lister, R. (2007), The Many Ways of the BRACElet Project. *Bulletin of Applied Computing and Information Technology* (BACIT) Vol. 5, Issue 1. ISSN 1176-4120. http://www.naccq.co.nz/bacit/0501/2007Whalley_BRACELET_Ways.htm

Whalley, J and Robbins, P (2007) Report on the Fourth BRACElet Workshop. *Bulletin of Applied Computing and Information Technology* (BACIT) Vol. 5, Issue 1. ISSN 1176-4120. http://www.naccq.ac.nz/bacit/0501/2007Whalley_BRACELET_Workshop.htm

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.