Ability to 'Explain in Plain English' Linked to Proficiency in Computer-based Programming

Laurie Murphy  
Pacific Lutheran University  
Tacoma, WA USA  
lmurphy@plu.edu

Raymond Lister  
University of Technology  
Sydney AUSTRALIA  
Raymond.Lister@uts.edu.au

Sue Fitzgerald  
Metropolitan State University  
St. Paul, MN USA  
sue.fitzgerald@metrostate.edu

Renée McCauley  
College of Charleston  
Charleston, SC USA  
mccauleyr@cofc.edu

ABSTRACT
This study investigates the relationship between novice programmers' ability to explain code segments and their ability to write code. Results show a strong correlation between ability to correctly answer 'explain in plain English' (EiPE) questions and ability to write code indicating that there are aspects of reasoning about code that are common to both writing code and explaining code. Student explanations were categorized using the Structure of the Observed Learning Outcome (SOLO) taxonomy. The better programmers were more likely to articulate relational aspects of the algorithms. While earlier work also found such a link, the code writing in those earlier studies was done on paper. This is the first such result where the writing component was done with 'hands on' a computer. Our results add further evidence for the existence of an aspect of reasoning about code that is common to both explaining code and writing code, which in turn suggests that a judicious mix of teaching both code skills and code explaining skills may lead to a more effective process by which novices learn to reason about code.

Categories and Subject Descriptors  
K.3.2 [Computing Milieux]: Computers and Education - Computer and Information Science Education

General Terms
Human Factors

Keywords
Computer science education research, qualitative research methods, mixed methods, explain in plain English, SOLO, novice programmers

1. INTRODUCTION
Several recent studies have explored the relationship between the ability of novice programmers to read and explain code and their ability to write code. Whalley et al. (2006) [10] reported on a study in which students in an end-of-semester exam were given a question that began “In plain English, explain what the following segment of Java code does”. They found that the better a student performed on other programming-related tasks in that same exam, the more likely the student was to provide a correct summary of the overall computation performed by the code in that explain in plain English (EiPE) question. As a result, they hypothesized that “a vital step toward being able to write programs is the capacity to read a piece of code and describe [its function].”

This result is interesting because it suggests that there may be other ways to teach programming, beyond writing lots of code. It is clear that some students pick up on programming quickly, while others struggle. In addition to writing code, is there another way to assist students in developing programming skills and perhaps reach students who struggle when taught only through coding?

Whalley et al. (2006) [10] analyzed student responses to EiPE questions using the Structure of the Observed Learning Outcome (SOLO) taxonomy [1]. There are five SOLO reasoning categories, roughly based on Piaget’s stages of development, which describe how students understand problems in increasingly abstract ways. Four of the five SOLO reasoning categories are described in Table 1 (adapted from [10]). The fifth level, extended abstract, is not relevant to their work or our work, and so it is not discussed here. The highest level of reasoning shown in Table 1, relational reasoning, occurs when a student can describe the purpose of a code segment, with minimal or no reference to specific details of the code. Table 1 provides examples of each of these SOLO categories. The examples are excerpts from actual student descriptions of a code segment which sums all the positive numbers stored in an array (see problem iv in Section 2.3).

Lopez et al. (2008) [5] built upon the work of Whalley et al., using stepwise regression to analyze student responses to an end-of-semester exam. They found that, when combined with student performance on code tracing questions, the ability to provide a correct relational response to an EiPE question accounted for almost half the variance in a code writing question in that same paper-based exam. Venables, Tan and Lister (2009) [9] also found...
a strong relationship between student performance on tracing questions, EiPE questions, and code writing questions in a paper-based exam. Lister, Fidge and Teague (2009) [4] performed a non-parametric analysis of similar data, collected in a paper-based exam, and reported similar findings.

The above studies have clearly demonstrated a relationship between EiPE questions and code writing in paper-based exams. However, the question remains as to whether asking students to write code on paper is a valid indication of their code writing ability. Prior to taking these paper-based end-of-semester exams, the novices involved would have written most of their code ‘hands on’ (i.e., on a computer). While some computing educators might argue that having novices write code on paper has its virtues (as Edsger Dijkstra did), it remains a fact that doing so was probably a relatively unfamiliar process for these novices. Could that unfamiliarity have affected their performance?

Apart from familiarity, writing code ‘hands on’ allows the novice to run and test their code. By doing so, a novice might detect logical errors that would otherwise go undetected on paper. Could these novices have lost points on the paper-based test that they would not have lost had the test been computer-based?

Like the earlier studies cited above, this paper reports on a study of the relationship between introductory students’ ability to explain code and to write code at the end of their first programming course. However, unlike the earlier studies, this paper reports on a final exam regime in which the students first completed the EiPE questions on paper and then immediately afterwards completed the computer-based code writing questions.

We believe that there is an underlying skill set common to both reading and writing code. Thus this study investigates whether the relationship between EiPE questions and code writing found in earlier studies was merely an artifact of having students write code on paper. The hypothesis for this study is that the relationship between EiPE questions and code writing also exists when the code writing is done on a computer.

This paper is organized as follows: Section 2 describes the study methodology. The results are presented in Section 3 and discussed in Section 4. Finally, Section 5 presents ideas for future work.

2. METHODOLOGY

2.1 Participants

Participants were 107 undergraduate students (34 female, 73 male) enrolled in six sections of a traditional “objects later” introductory Java programming course, offered over four semesters (Spring 2010 through Fall 2011). All sections of the class were taught by one instructor using very similar course materials. Students who took the class in Spring and Fall 2010 were not exposed to EiPE questions prior to the final exam, while those in Spring and Fall 2011 were given several EiPE questions on quizzes and tests throughout the term. The instructor also reviewed and emphasized the EiPE responses with the 2011 students when the graded tests were returned. (See [6] for details.)

### Table 1. SOLO Categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
<th>Examples for a method that sums the positive elements in an array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational</td>
<td>Provides a summary of what the code does in terms of the code’s purpose</td>
<td>“adds up all values in the array that are positive and returns them”</td>
</tr>
<tr>
<td>Multistructural</td>
<td>A line-by-line description is provided of most of the code</td>
<td>“For integer i = 0 and i less than the amount of numbers in the array [numbers], add 1 to i and if the array numbers is greater than zero add i to the num, return the num at the end.”</td>
</tr>
<tr>
<td>Unistructural</td>
<td>Provides a description for only a small portion of the code</td>
<td>“Returns the value of num”</td>
</tr>
<tr>
<td>Prestructural</td>
<td>The answer demonstrates no relevant knowledge or is unrelated to the question</td>
<td>“This will return how many letters there are in statement”</td>
</tr>
</tbody>
</table>

2.2 Data Collection

All data were collected during a two-part final exam. Both the written part (worth 55 points) and programming part (worth 45 points, plus 5 bonus points) were administered in a computer classroom during a one hour and 50 minute final exam period. Students were encouraged to complete the written exam during the first hour to allow enough time to complete the programming part. Final exams are not returned to the students, so identical or nearly identical questions were used every semester.

2.3 Written Test Instrument

The written exams were closed book and closed notes. The EiPE questions were worth three points each (12 points total) out of 55 points for the written exam. The students were required to provide an explanation for four separate pieces of code (see i-iv below), which were preceded by the following instruction:

For each of these sections of code, explain in plain English what it does. For example, “It displays the sum of the two integers” or “It returns the last value in the array.” Don’t describe the code line-by-line; state what it does overall.

i. (Nested ifs) Assume a, b and c are all int variables that have been declared and initialized.

```java
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);
```

Note: The code above displays the largest of the three values stored in variables a, b and c. Some students in our data sample were presented with a version that printed the smallest value. For either version, student responses were graded appropriately.
ii. (Longest String) Consider the following method that is passed an array of strings:

```java
public String methodC(String[] word)
{
    String strOne = word[0];
    for (int i = 1; i < word.length; i++)
    {
        if (word[i].length() > strOne.length())
            strOne = word[i];
    }
    return strOne;
}
```

iii. (Linear Search) Consider the following method that searches an array:

```java
public int methodB(int data[], int x)
{
    int found = -1;
    for (int i = 0; i < data.length; i++)
    {
        if (data[i] == x)
            found = i;
    }
    return found;
}
```

iv. (Sum of Positives) Consider the following method that is passed an array of doubles:

```java
public double methodA(double[] numbers)
{
    double num = 0;
    for (int i = 0; i < numbers.length; i++)
    {
        if (numbers[i] > 0)
            num += numbers[i];
    }
    return num;
}
```

2.4 Computer-based Instrument

Once students finished the written final, they were given a programming exam that required them to complete a set of short Java programs by adding basic constructs and simple object-oriented code. Students were provided with three syntax reference sheets during the programming exam, but were not allowed to use other references. To prevent access to lab and homework solutions, students were assigned temporary accounts that only included the exam source files. Their machines were also monitored using classroom management software. The programming exam counted for 45 of the 100 exam points with an opportunity to earn 5 extra credit points. Table 2 describes the programming questions.

### Table 2: Computer-based Programming Exam Questions

<table>
<thead>
<tr>
<th>Problem Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

2.5 Data Coding

For grading purposes, the EiPE questions were marked along with all other questions on the final exam and they received partial credit in some cases. Points awarded during the marking were used for analysis purposes. These marks were then used in the analysis to examine correlations between performance on the explanation questions and the programming exam. These results are discussed in section 4.

To facilitate analysis, students were placed in “buckets” according to their test scores on the computer-based programming exam. The students were divided roughly into four quartiles. The quartiles were 0–30 points (28 students), 31–42 points (28 students), 43–49 points (27 students) and 50 points (24 students).

We also conducted analyses of the answers to the explanation questions using the SOLO taxonomy (see Table 1). Responses were categorized independently by two researchers who then argued to consensus on the appropriate categorization of 97.2% of the responses.

3. RESULTS

In this section we present an analysis of our results. Section 3.1 describes overall student performance. Sections 3.2-3.5 focus on each of the EiPE questions. The EiPE questions are presented in order from easiest to most difficult.

While our research hypothesis is that there exists a relationship between EiPE questions and code writing, we make no hypothesis about the exact nature of that relationship (e.g. a linear...
relationship). Consequently, we took a non-parametric approach to the analysis. As noted in section 2.5, the students were divided into quartiles based on their performance on the hands on code writing exam. In sections 3.2-3.5, the quartiles are compared via a close examination of the students' responses to the EiPE questions using a χ² test. A statistically significant difference in performance between any two quartiles reveals a relationship between ability to describe algorithms relationally and ability to program. This analytic approach was also used in previous work by Lister, Fidge and Teague [4].

3.1 Overall Statistics

With four semesters of data involving 107 participants, we considered scores on the EiPE questions, and results on the programming portion of the final. The EiPE question scores, out of a total of 12 points, had a range of 0-12, a mean of 8.65 and a standard deviation of 3.27. The final programming exam scores, out of a total of 50 points, had a range of 0-50, a mean of 36.83, and a standard deviation of 14.2. Our data was not normally-distributed, so a Spearman's correlation was used to show that the points on the EiPE questions and the computer-based programming instrument are highly correlated; Spearman's rho is .687, with p < .001. This is highly significant and indicates that it is extremely unlikely that this relationship occurred by chance.

We compared the EiPE question scores for students in Spring and Fall 2010 to those of students in Spring and Fall 2011. Students in Spring and Fall 2011 received extra instruction in how to explain code segments. We found no significant differences between the scores on EiPE questions between the two groups; the 2011 students' prior experience with EiPE questions did not appear to affect ability to answer explaining questions correctly.

An examination of the SOLO categorization of student explanations showed that students were more likely to express themselves relationally than any other way (see Table 3). For the most difficult question (summing positive numbers, question iv), approximately 40% of the students either identified only one element of the problem (unistructural) or answered meaninglessly (prestructural) but for the simplest problem (display the largest element of the problem) or answered meaninglessly themselves relationally than any other way (see Table 3). For the other three explanation questions. While 89% of the third quartile students answered the question correctly, that percentage is not statistically significantly different from the 71% percentage for the second quartile (χ² test, p=0.11, as shown in column (i) row 5 of Table 4). Also, the 96% performance of the top quartile is not significantly different from the 89% performance of the third quartile (χ² test, p=0.36, as shown in row 7 of Table 4).

Common sense suggests that we should not expect much difference between two students whose respective scores on the computer-based programming test differ by only 2 points out of 50. It follows that we should not expect much difference between students who scored 1 mark below a quartile boundary and students who scored 1 mark above that same quartile boundary. Thus, our use of the traditional statistical p=0.05 criterion is a tough standard to apply to adjacent quartile buckets. Table 5 summarizes χ² tests on the non-adjacent second and top quartiles.

In summary, the nested if problem was an easy explanation question for this sample of 107 students, as many students who performed in the bottom quartile of the computer-based programming test could answer this question. The most useful aspect of this question is that it serves to demonstrate that at least 79% of the entire sample of 107 students had a good understanding of what was required of them when answering this explanation question. It follows that those 79% of students also had a good understanding of what was expected when answering the other three explanation questions.

3.3 Longest String

The performance of the students on the longest string explanation question is summarized in column (ii) of Table 4. Of the entire sample of 107 students, 70% provided a SOLO relational response, indicating that the code printed the largest/smallest value, depending upon which version of the code the student was given. This 79% is the highest percentage of relational answers for the four explanation questions, which is perhaps not surprising, given that this code was the only code that did not contain a loop.

Almost two-thirds (64%) of the bottom quartile provided a suitable answer to this question, as did 71% of the second quartile. A χ² test failed to establish a statistically significant difference in the percentages of these two quartiles (p = 0.57, as shown in Table 4, in column (i), row 3, the row between the 64% and 71% performance figures of these two quartiles). While 89% of the third quartile students answered the question correctly, that percentage is not statistically significantly different from the 71% percentage for the second quartile (χ² test, p=0.11, as shown in column (i) row 5 of Table 4). Also, the 96% performance of the top quartile is not significantly different from the 89% performance of the third quartile (χ² test, p=0.36, as shown in row 7 of Table 4).

The following sections focus upon how the performance on EiPE questions differs between buckets.

3.2 The Nested ifs

Student performance on the nested if EiPE question is summarized in column (i) of Table 4. Of the entire sample of 107 students, 79% provided a suitable answer (i.e., a SOLO relational response, indicating that the code printed the largest/smallest value, depending upon which version of the code the student was given). This 79% is the highest percentage of relational answers for the four explanation questions, which is perhaps not surprising, given that this code was the only code that did not contain a loop.

Almost two-thirds (64%) of the bottom quartile provided a suitable answer to this question, as did 71% of the second quartile. A χ² test failed to establish a statistically significant difference in the percentages of these two quartiles (p = 0.57, as shown in Table 4, in column (i), row 3, the row between the 64% and 71% performance figures of these two quartiles). While 89% of the third quartile students answered the question correctly, that percentage is not statistically significantly different from the 71% percentage for the second quartile (χ² test, p=0.11, as shown in column (i) row 5 of Table 4). Also, the 96% performance of the top quartile is not significantly different from the 89% performance of the third quartile (χ² test, p=0.36, as shown in row 7 of Table 4).

Common sense suggests that we should not expect much difference between two students whose respective scores on the computer-based programming test differ by only 2 points out of 50. It follows that we should not expect much difference between students who scored 1 mark below a quartile boundary and students who scored 1 mark above that same quartile boundary. Thus, our use of the traditional statistical p=0.05 criterion is a tough standard to apply to adjacent quartile buckets. Table 5 summarizes χ² tests on the non-adjacent second and top quartiles.

In summary, the nested if problem was an easy explanation question for this sample of 107 students, as many students who performed in the bottom quartile of the computer-based programming test could answer this question. The most useful aspect of this question is that it serves to demonstrate that at least 79% of the entire sample of 107 students had a good understanding of what was required of them when answering this explanation question. It follows that those 79% of students also had a good understanding of what was expected when answering the other three explanation questions.

3.3 Longest String

The performance of the students on the longest string explanation question is summarized in column (ii) of Table 4. Of the entire sample of 107 students, 70% provided a SOLO relational response.

Less than one third (29%) of the bottom quartile provided a suitable answer to this question, compared with more than two thirds (68%) of the second quartile. A χ² test established a statistically significant difference in these percentages (p = 0.003). This cell and some other cells in Table 4 are shaded to highlight statistically significant differences between quartiles. The “ES = 4” in that cell shows the effect size, as measured by the odds ratio. The 4 means that the odds a student in the second quartile gave a relational response are four times higher than for a student in the bottom quartile. Rosenthal (1996) [7] classified effect sizes measured by odds ratios as follows: an odds ratio in the range

<table>
<thead>
<tr>
<th>Question</th>
<th>(i) Nested ifs</th>
<th>(ii) Longest String</th>
<th>(iii) Linear Search</th>
<th>(iv) Sum of Positives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational</td>
<td>79%</td>
<td>70%</td>
<td>54%</td>
<td>49%</td>
</tr>
<tr>
<td>Multi-structural</td>
<td>9%</td>
<td>5%</td>
<td>24%</td>
<td>11%</td>
</tr>
<tr>
<td>Unistructural</td>
<td>7%</td>
<td>9%</td>
<td>13%</td>
<td>27%</td>
</tr>
<tr>
<td>Prestructural</td>
<td>4%</td>
<td>16%</td>
<td>8%</td>
<td>13%</td>
</tr>
</tbody>
</table>

* Results may not sum to 100% due to round off error

The following sections focus upon how the performance on EiPE questions differs between buckets.
1–1.5 is small, 1.5–2.5 is medium, around 4 is large, and around 10 is "very large". Thus the effect size “ES = 4” in that cell is large.

While there is not a statistically significant difference between the second and third quartiles, or the third and top quartiles, Table 5 shows that there is a statistically significant difference between the second and top quartiles (as is the case with all four explanation questions). Furthermore, the effect size between the second and top quartiles is 11 — the odds that a student in the top quartile gave a relational response are eleven times higher than for a student in the second quartile. Such an effect size is very large.

### 3.4 Linear Search

The performance of the students on the linear search explanation question is summarized in column (iii) of Table 4. Of the entire sample of 107 students, just over half (54%) provided a response in which they mentioned that the code performed a search and returned a position in the array.

While there is not a statistically significant difference between the second and third quartiles, or the third and top quartiles, Table 5 shows that there is a statistically significant difference between the second and top quartiles with a large effect size of 8.

As discussed in the subsection on the nested if explanation question, the traditional statistical p=0.05 criterion is a tough standard to apply to adjacent quartile buckets. Thus, the p = 0.07 value for the difference between the second and third quartiles is weakly suggestive evidence that there is a performance difference, particularly as p=0.07 can be interpreted as indicating that there is only a 1 in 14 chance of the difference being a sampling fluke, compared to the 1 in 20 chance of the traditional p=0.05. The cell containing this p=0.07 value is more lightly shaded than other shaded cells, to indicate that it is weakly suggestive but not statistically significant.

Table 6 provides more detailed statistics on this Linear Search explanation question. The column headed “Notion of Searching” records the percentage of students, both as a whole sample and in each quartile, who indicated that the code was performing some form of search. Only 79% of the entire sample did so, which we regard as surprising since searching is mentioned in the preamble to the code (“Consider the following method that searches an array”). A mere 43% of the bottom quartile indicated that the code was performing some form of search, which is significantly lower than the 82% of the second quartile. Table 7 shows that there is not a statistically significant difference between the 82% of second and the 96% of the top quartile.

Only 31% of the entire sample mentioned that -1 is returned if the search value is not found. In fact, only 58% of the top quartile mentioned this aspect of the code, but the data in Table 7 indicates that the top quartile performed significantly better than the second quartile, and the data in Table 6 is weakly suggestive of a performance difference between the third and top quartiles.

The final column in Table 6 presents data on a subtle aspect of this piece of code – what happens when the search value occurs more than once in the array? Among the 56 students in the lowest two quartiles, only one student raised this issue in their answer. The percentage of students in the highest two quartiles who raised this issue is not high, but it is significantly different than the performance of the lowest two quartiles, with a large effect size.

In summary, the better performers on the computer-based programming test (students in the higher quartiles) were more likely to articulate aspects of the algorithm beyond simply that it performs a search. Note that we are not claiming that a student who did not mention aspects of the algorithm, such as for example returning -1 when the search value is not found, is incapable of understanding that aspect of the code. All we claim is that students who performed better on the computer-based programming test were more likely to make note of such details in their answers. We speculate that good programming students are also more likely to spontaneously think about contingencies when writing code.

### 3.5 Sum of Positive Numbers

The students performed surprisingly poorly on explaining the code that sums the positive elements of an array. As indicated in column (iv) of Table 4, only 49% of the entire sample of 107 students provided a suitable response. The performance of the bottom quartile was particularly poor (11%), and significantly worse than the performance of the second quartile, even though only 39% of the second quartile students answered appropriately.
Even in the third quartile, only two-thirds of the students answered appropriately.

An examination of student answers reveals that a common mistake was to respond that the code summed all the elements of the array. In making that mistake, students ignored the “if” statement within the loop – to do so is an egregious error. Table 8 summarizes statistics on the students who made this error. The percentages are calculated from the subset of 72 students who either articulated that the code summed the positive elements, or that it summed all elements. Only 8 students in the bottom quartile gave either type of answer, but 5 of those 8 (63%) ignored the “if” statement and wrote that the code summed all elements of the array. The performance of students improves with each higher quartile, with only 1 of the 21 students (5%) in the top quartile making this mistake of ignoring the “if” statement. While none of the quartiles meet the strict $p = 0.05$ criteria, a 45% of the second quartile and the 5% of the top quartile ($p = 0.003$), with a very large effect size of 16.

This error of ignoring the “if” statement is of a different class than, for example, not mentioning that the linear search returns -1 when the search value was not found. The “if” statement within the loop checking for positive values is written there in plain sight on the page in front of the student – how can a student possibly ignore it? The theory of cognitive load [8] provides one possible explanation. When the working memory of any person is overwhelmed, important information may be ignored. In the specific case of this code explanation question, a student who is struggling to cope with the details of this code may note the occurrence of a “for” loop controlled by a variable “i”, and then focus upon the line “num += numbers[i]”. Such an explanation is also consistent with the behavior of pre-operational novices as defined within the Neo-Piagetian view of the novice programmer [3].

### Table 5. Comparing Performance of Second and Top Quartiles (* indicates $p \leq 0.05$ and ** indicates $p \leq 0.01$)

<table>
<thead>
<tr>
<th>Row</th>
<th>Description</th>
<th>n</th>
<th>(i) Nested ifs</th>
<th>(ii) Longest String</th>
<th>(iii) Linear Search</th>
<th>(iv) Sum of Positives</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Second Quartile, 31−42 points</td>
<td>28</td>
<td>64%</td>
<td>68%</td>
<td>46%</td>
<td>39%</td>
</tr>
<tr>
<td>5</td>
<td>$\chi^2$ test</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Top Quartile , 50 points</td>
<td>24</td>
<td>96%</td>
<td>96%</td>
<td>88%</td>
<td>83%</td>
</tr>
</tbody>
</table>

### Table 6. Comparing Performance on Linear Search by Quartile (* indicates $p \leq 0.05$ and ** indicates $p \leq 0.01$)

<table>
<thead>
<tr>
<th>Row</th>
<th>Description</th>
<th>N</th>
<th>Notion of Searching</th>
<th>Returns position</th>
<th>Returns -1 when not found</th>
<th>Returns position of last found</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Whole sample</td>
<td>107</td>
<td>79%</td>
<td>57%</td>
<td>31%</td>
<td>15%</td>
</tr>
<tr>
<td>2</td>
<td>Bottom Quartile , 0−30 points</td>
<td>28</td>
<td>43%</td>
<td>18%</td>
<td>11%</td>
<td>0%</td>
</tr>
<tr>
<td>3</td>
<td>$\chi^2$ test</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Second Quartile, 31−42 points</td>
<td>28</td>
<td>82%</td>
<td>57%</td>
<td>25%</td>
<td>4%</td>
</tr>
<tr>
<td>5</td>
<td>$\chi^2$ test</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Third Quartile, 43–49 points</td>
<td>27</td>
<td>96%</td>
<td>70%</td>
<td>33%</td>
<td>22%</td>
</tr>
<tr>
<td>7</td>
<td>$\chi^2$ test</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Top Quartile , 50 points</td>
<td>24</td>
<td>96%</td>
<td>88%</td>
<td>58%</td>
<td>38%</td>
</tr>
</tbody>
</table>

### Table 7. Comparing Performance of Second and Top Quartiles for Linear Search Question (* for $p \leq 0.05$, ** for $p \leq 0.01$)

<table>
<thead>
<tr>
<th>Row</th>
<th>Description</th>
<th>N</th>
<th>Notion of Searching</th>
<th>Returns position</th>
<th>Returns -1 when not found</th>
<th>Returns position of last found</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Second Quartile, 31−42 points</td>
<td>28</td>
<td>82%</td>
<td>57%</td>
<td>25%</td>
<td>4%</td>
</tr>
<tr>
<td>2</td>
<td>$\chi^2$ test</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Top Quartile , 50 points</td>
<td>24</td>
<td>96%</td>
<td>88%</td>
<td>58%</td>
<td>38%</td>
</tr>
</tbody>
</table>
The crucial issue is whether this correlation is that better programming students do better on every aspect of evidence for that [2]. While we would not go so far as to say, as there is some accurate assertion would be "correlate well with grades in general" [2]. Perhaps a more known that grades for introductory programming courses do not variance across quartiles that we see in our data, since it is unlikely that general ability would account for most of the tasks, as Lopez et al. did in their study [5]. However, it is interesting to include a non-programming task, and attempt to factor out general ability from the analysis of the programming abilities versus acquired abilities, and also near transfer versus far transfer in learning. These assumptions have implications for the interpretation of our results, which we discuss in the remainder of this section.

In future experiments, building on these results, it would be interesting to include a non-programming task, and attempt to factor out general ability from the analysis of the programming tasks, as Lopez et al. did in their study [5]. However, it is unlikely that general ability would account for most of the variance across quartiles that we see in our data, since it is known that grades for introductory programming courses do not correlate well with grades in general [2]. Perhaps a more accurate assertion would be "better programming students are better at every aspect of programming", as there is some evidence for that [2]. While we would not go so far as to say that better programming students do better on every aspect of programming, our results show that writing code and explaining code correlate. The crucial issue is whether this correlation is because (1) the ability to explain code is dependent on the ability to write code, or (2) the opposite causal relationship (i.e. the ability to write code is dependent on the ability to explain code), or (3) the ability to write code and read code are both dependent on a common set of underlying skills. While our experimental results cannot choose among these three possibilities, our intuition is that the latter is the most likely possibility.

A further question is whether the ability that is common to both writing and explaining is innate or is acquired through learning. While there may be a component to programming ability that is innate, our intuition is that programming ability is largely acquired, at least at the CS1 level, but our results neither prove nor disprove our intuition.

4. DISCUSSION
Our results (and earlier results by others) indicate that there are aspects of reasoning about code that are common to both writing code and explaining code. Our use of a computer-based test of code writing ability establishes that the results of earlier studies were not an artifact of having students write code on paper. There are probably reasoning abilities specific to writing code and abilities specific to explaining code. However, our results suggest the possibility that the reasoning abilities common to both writing code and explaining code are non-trivial, given the clear statistical relationships we have found between performance on our code explanation questions and our computer-based code writing questions. This result suggests that there may be other ways to teach and learn programming, beyond writing lots of code -- ways that focus on the elements common to reading, writing and explaining code.

It could be asserted that our results are unsurprising, as "better students tend to be better at everything". However, this assertion contains a mix of implicit assumptions about innate abilities versus acquired abilities, and also near transfer versus far transfer in learning. These assumptions have implications for the interpretation of our results, which we discuss in the remainder of this section.

In future experiments, building on these results, it would be interesting to include a non-programming task, and attempt to factor out general ability from the analysis of the programming tasks, as Lopez et al. did in their study [5]. However, it is unlikely that general ability would account for most of the variance across quartiles that we see in our data, since it is known that grades for introductory programming courses do not correlate well with grades in general [2]. Perhaps a more accurate assertion would be "better programming students are better at every aspect of programming", as there is some evidence for that [2]. While we would not go so far as to say that better programming students do better on every aspect of programming, our results show that writing code and explaining code correlate. The crucial issue is whether this correlation is because (1) the ability to explain code is dependent on the ability to write code, or (2) the opposite causal relationship (i.e. the ability to write code is dependent on the ability to explain code), or (3) the ability to write code and read code are both dependent on a common set of underlying skills. While our experimental results cannot choose among these three possibilities, our intuition is that the latter is the most likely possibility.

A further question is whether the ability that is common to both writing and explaining is innate or is acquired through learning. While there may be a component to programming ability that is innate, our intuition is that programming ability is largely acquired, at least at the CS1 level, but our results neither prove nor disprove our intuition.

5. FUTURE DIRECTIONS
Our results add further evidence for the existence of an aspect of reasoning about code that is common to both explaining code and writing code which in turn suggests that a near-complete pedagogical emphasis on writing code is too great, as writing code is a slow, tedious process, especially for novices. A more judicious mix of tracing code, reading code (e.g. by explaining it) and writing code may lead to a more effective and efficient process by which many novices would learn to reason about code.

Although proficiency at code explaining and code writing are linked, we can report that mere exposure to code explanation tasks does not appear to improve code writing ability. Across the four semesters in which we collected data, the students in the two early semesters were not shown any code explanation questions prior to the final exam, whereas the students in the two later semesters were given code explanation questions as part of their learning throughout the semester. We saw no statistical difference in code explaining performance of students in the first two semesters from those in the later semesters. The lack of such a difference is not evidence against placing a greater pedagogical emphasis on code explaining. While the students in the later two semesters were given some code explanation questions throughout semester, and the answers for these questions were discussed by the teacher in the classroom, the addition of these questions fell well short of being a systematic pedagogical intervention. Just as some students need a great deal of help to learn how to write code, some students will need a well developed pedagogical approach if they are to learn how to explain code. The development of pedagogical approaches that mix tracing, describing, and writing code is an area requiring substantial future work.

6. ACKNOWLEDGEMENTS
We thank Brian Hanks for his invaluable assistance with our statistical analysis.

Table 8. The Percentage of Students Responding that the Sum of Positives Code Returned the Sum of All Elements

<table>
<thead>
<tr>
<th>Description</th>
<th>N</th>
<th>Percentage of Sum of All Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whole sample</td>
<td>72</td>
<td>28%</td>
</tr>
<tr>
<td>Bottom Quartile</td>
<td>8</td>
<td>63%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\chi^2$ test, $p = 0.4$</td>
</tr>
<tr>
<td>Second Quartile</td>
<td>20</td>
<td>45%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\chi^2$ test, $p = 0.10$ (1 in 10)</td>
</tr>
<tr>
<td>Third Quartile</td>
<td>23</td>
<td>22%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\chi^2$ test, $p = 0.10$ (1 in 10)</td>
</tr>
<tr>
<td>Top Quartile</td>
<td>21</td>
<td>5%</td>
</tr>
</tbody>
</table>
7. REFERENCES


