ABSTRACT
Recent studies have linked the ability of novice (CS1) programmers to read and explain code with their ability to write code. This study extends earlier work by asking CS2 students to explain object-oriented data structures problems that involve recursion. Results show a strong correlation between ability to explain code at an abstract level and performance on code writing and code reading test problems for these object-oriented data structures problems. The authors postulate that there is a common set of skills concerned with reasoning about programs that explains the correlation between writing code and explaining code. The authors suggest that an overly exclusive emphasis on code writing may be detrimental to learning to program. Non-code writing learning activities (e.g., reading and explaining code) are likely to improve student ability to reason about code and, by extension, improve student ability to write code. A judicious mix of code-writing and code-reading activities is recommended.

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

General Terms
Human Factors

Keywords
Explain in plain English, CS2, computer science education research, data structures, qualitative research methods, mixed methods, SOLO taxonomy, neo-Piaget.

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 [3, 7, 9, 10, 16, 17]. Whalley et al. (2006) [17] reported on a study in which first-semester 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 analyzed student responses to these “explain in plain English” (EiPE) problems using the Structure of the Observed Learning Outcome (SOLO) taxonomy [2]. The five SOLO reasoning categories describe how students understand problems in increasingly abstract ways. Table 1, adapted from Whalley et al., and copied from Murphy et al. [9], provides definitions for the first four SOLO categories for describing code. 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.

The fifth and highest SOLO level, extended abstract, is the stage at which an individual generalizes and applies his/her abstractions to new domains. Neither subjects in Whalley’s study nor in this study were given problems which explored their ability to extend knowledge and therefore the extended abstract stage is not discussed further.

Table 1. SOLO Categories for Student Responses to “Explain in Plain English” (EiPE) problems

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</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>
</tr>
<tr>
<td>Multistructural</td>
<td>A line-by-line description is provided of all the code.</td>
</tr>
<tr>
<td>Unistructural</td>
<td>Provides a description for only one portion of the code.</td>
</tr>
<tr>
<td>Prestructural</td>
<td>Substantially lacks knowledge of constructs or is unrelated to the question.</td>
</tr>
</tbody>
</table>

Whalley et al. found that the better a student performed on other programming-related tasks in that same exam, including code writing tasks, the more likely the student was to provide a relational response to an EiPE question. As a result, they...
Lopez et al. [7] built upon the work of Whalley et al., using stepwise regression to analyze first-semester student responses to an end-of-semester exam. They found that, when combined with student performance on code tracing problems, the ability to provide a correct relational response to an EiPE question accounted for almost half the variance in student performance on a code writing question in that same paper-based exam. That is, although student code writing performance varied considerably, on average code writing and code explaining combined were a good predictor of student ability to write code.

Venables, Tan and Lister [16] also found a strong correlation between first-semester student performance on tracing problems, EiPE problems, and code writing problems in a paper-based exam. Murphy et al. [9] also studied 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 in which students wrote code on paper, Murphy et al. [9] required students to write and implement code on a computer. Murphy et al. found results similar to those in the paper-based coding studies.

The success of some newer pedagogies in CS 1 [11] may be partly attributable to their support for the development of code explaining skills. For example, partners request and provide explanations of their code when pair programming [6] and students taught using peer instruction gain a deeper understanding of course material by explaining concepts to each other and by hearing others’ explanations [14].

While previous studies have demonstrated a relationship between writing and explaining code in CS0 and CS1 courses, there have been no studies using “explain in plain English” problems beyond those introductory courses. While there appears to be benefit for students in introductory courses in explaining code to themselves and each other, the benefits of doing so as students progress have not previously been shown. This study investigates the relationship between students’ ability to explain code and to write code in a data structures course. Our null hypothesis is:

$H_0$: There is no correlation between code writing and explaining code for students in a course on data structures.

This paper is organized as follows: Section 2 describes the study methodology. The results are presented in Section 3 and discussed in Section 4. Section 5 presents pedagogical implications and Section 6 gives ideas for future work.

2. METHODOLOGY

2.1 Participants

Participants were 120 out of 137 undergraduate students (14 female, 123 male) enrolled in their third programming related course titled “Data Structures and Algorithms”, which was taught at a large publicly funded university. The semester-long course was taught using C++. In prior courses students programmed in Python and C#; and some participants were also concurrently enrolled in a software development course that used Java. Most participants were traditional college students (around 20 years of age) and most spoke English as their first language.

2.2 Data Collection

All data were collected during a written final exam that was scored out of 50 points and made up 50% of the final grade for the course. Problems on the exam required students to trace code, compare implementations and write code involving C++ pointers and standard template library classes, and compare and identify the runtimes of common data structure algorithms. The exam was preceded by two programming assignments. Assignment 1 was about poker hand evaluation (worth 20% of the final grade). Assignment 2 was an implementation of an algorithm for finding approximate solutions to the travelling salesman problem, using a minimum spanning tree and depth first search (worth 30% of the final grade).

2.3 Test Instrument

The problems of interest for this study were worth one point each (4 points total) out of 50 points for the written final exam. They were part of a larger question, which was worth 10 points. The problems asked students to trace and explain recursive methods that operated on a linked list. The tracing problems (3a and 3c below) were multiple choice and required students to trace code for a specific list made up of only two nodes. The explanation problems (3b and 3d below) required students to reason about the code generally and describe what it would do for any linked list.

Prolog to problems 3a & 3b

The questions that follow refer to the class Node. Below is the code that commences the definition of this class. This class is used to form a linked list. Each instance of class Node contains an integer data.

```cpp
#include <cstdlib>
using namespace std;

class Node {
private:
    int data;
    Node *next;
public:
    Node(int num, Node *nextNode) {
        data = num;
        next = nextNode;
    }
    int GetData() { return data; }  
    // rest of class Node not shown here
};
```

The following code builds a linked list of two nodes, where head points to the first Node in the linked list:

```cpp
Node *x = new Node(1, NULL);
Node *head = new Node(2, x);
```

That linked list is represented by the following diagram:

![Linked List Diagram]

The function Puzzle is shown below and is a public function in the Node class.

```cpp
bool Puzzle() {
    if (next == NULL)
        return true;
    else
        if (data < next->GetData())
            return false;
        else
            return next->Puzzle();
}
```
3a. If the function Puzzle was executed on the linked list shown [above] by executing the expression

head->Puzzle() which one of the following statements is correct:

(i) The Puzzle functions in both objects would execute and both would return false.
(ii) The Puzzle functions in both objects would execute and both would return true.
(iii) Only the Puzzle function in the object containing data = 2 would execute and it would return false.
(iv) Only the Puzzle function in the object containing data = 2 would execute and it would return true.
(v) The Puzzle function in the object containing data = 1 would return true and the Puzzle function in the object containing data = 2 would return false.
(vi) The Puzzle function in the object containing data = 1 would return false and the Puzzle function in the object containing data = 2 would return true.

3b. Complete the following sentence: If the function call head->Puzzle() is invoked on any linked list of two or more nodes, then head->Puzzle() will return true if and only if _____________.

3c. The functions Conundrum and Teaser are shown below and are public functions in the Node class:

Node* Conundrum() {
    if (next == NULL) return this;
    else return next->Teaser(this);
}

Node* Teaser(Node *p) {
    Node *temp;
    if (data < p->GetData()) temp = this;
    else temp = p;
    if (next == NULL) return temp;
    else return next->Teaser(temp);
}

3d. Complete the following sentence: If the function call head->Conundrum() was executed on any linked list of two or more objects of class Node, that expression would return a pointer to _______________.

2.4 Data Coding

Five of the six authors (henceforth the five “analysts”) independently categorized the students’ responses to the explanation problems (3b and 3d above) according to the SOLO taxonomy, as shown in Table 1. The analysts did not teach the course in which the data was collected (it was taught by the first author) although they were involved in the design of the test problems. All are experienced teachers and researchers. The researchers comprise a multi-national group.

Initially, each analyst coded independently. Then analysts met to discuss ratings, and tried to come to consensus on responses where there were differences. Details of these discussions are covered in section 3.1. Inter-rater agreement, from a statistical view, is discussed in section 3.2.

3. RESULTS

The code in 3a & b verifies that a list is in descending order. The correct answer for 3a is ii. The code in 3c & d returns a pointer to the node with the lowest data value. The correct answer for 3c is i.

Table 2 shows examples of typical responses to problems 3b and 3d which fall into each SOLO category.

<table>
<thead>
<tr>
<th>Category</th>
<th>Example Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational (R)</td>
<td>data in each node is in descending order (3b) pointer to the node with the lowest value (3d)</td>
</tr>
<tr>
<td>Multistructural (M)</td>
<td>following items in the list had a greater data value and second last element pointed to a list item with a greater data value and null for the next item (3b) first node that is smaller than its previous node or last node in list if all nodes are larger than previous (3d)</td>
</tr>
<tr>
<td>Unistructural (U)</td>
<td>next is not equal to null (3b) the last value in the list (3d)</td>
</tr>
<tr>
<td>Prestructural (P)</td>
<td>all values = null (3b) pointer to first object contains data = 1 and next = null (3d)</td>
</tr>
</tbody>
</table>

Through discussion among analysts, it became apparent that there were several responses that some analysts considered relational while others considered them to be multistructural. Similarly, there were a number of responses that received a mix of prestructural and unistructural ratings. After some consideration, it became clear the responses were largely divisible into two groups: those where the reasoning was at least multistructural, and those where the reasoning was unistructural at most. Given the nuances of these understandings, relational and multistructural answers were combined and unistructural and prestructural answers were combined for the purposes of statistical analysis.

3.1 Reliability of Data Coding

Analysts agreed on the ratings for a majority of participant responses. When answers were clean, concise and correct as shown in the first row in Table 2, all raters categorized them as relational. Answers which erred only in direction (“nodes sorted by data value in ascending order” rather than the correct “descending order”) were also marked as relational. Similarly,
nonsensical answers such as those in the last row of Table 2 were rated as prestructural by all analysts.

However, they disagreed on a fair number of responses and their discussions led to further consideration of how to apply the SOLO taxonomy to programming tasks. In the interests of explaining the use of the SOLO taxonomy as a tool for evaluating the quality of student explanations, a detailed discussion of the analysts’ interpretations is given below.

Analysts disagreed on answers such as “all linked values are smaller than previous.” Three of five analysts considered this correct answer to be relational because it demonstrated that the participant understood that the code verified that the nodes were in descending order. These analysts opined that a student can have a correct answer even if that answer is not expressed in the clearest possible way. The other two analysts rated this correct answer as multistructural since it provided a detailed explanation, rather than the succinct clarity indicating a deep understanding of the problem. These analysts argued that a correct answer is not necessarily relational; any answer explained in a detailed, line-by-line way is multistructural.

Other considerations were debated. During discussion, it became clear that some analysts took a strict interpretation of the SOLO categorization as described in Table 1. They focused on the student’s description of the code. However, others considered a node by node discussion of the data structure to be multistructural. That is, they did not interpret multistructural as referring only to code but intuitively extended the definition to include descriptions of the data structure. This suggests a different interpretation than that used by Whalley et al. [17] in earlier studies of array-based problems for CS 1 students.

There were also some unresolved differences between ratings of unistructural and prestructural. This was perhaps a reflection of the individual analyst’s willingness to read meaning into an answer. All analysts agreed that for many problems, there was little difference between incorrect unistructural and nonsensical prestructural answers. Unistructural and prestructural answers were combined for the purposes of statistical analysis.

A few answers triggered a wider range of classifications and sparked considerable discussion, for example, this answer to question 3b:

“deeper values are less than predecessor” (1U, 2M, 2R)

Some analysts interpreted “deeper” as “nodes further along in the list”, a relational answer, while other raters were not willing to add meaning to the written answer and judged the answer as multistructural or even unistructural.

Similarly, there was no consensus on the SOLO ranking for this answer to question 3b:

“If next node returns true, the only way for the second node to return true is if the third node returns true as well” (1P, 1U, 3M)

Three raters noted that the student mentioned multiple nodes and interpreted this answer as multistructural. Others were unable to find meaning and interpreted it as prestructural or unistructural.

This response to question 3d also generated a mixed rating:

“first node that has a smaller data value than the previous node” (3U, 2M)

Clearly this answer is not correct; the correct answer is “the code returns a reference to the smallest node”. However, the subject refers to more than one node, leading to a multistructural categorization by 2 analysts. Yet, only two nodes are referenced which does not imply that the condition holds for the entire list. As a consequence three analysts categorized the answer as unistructural.

### 3.2 Quantitative Results

As mentioned previously, the final exam was worth 50 points, and the two EiPE problems studied were part of question 3, which was worth a total of 10 points. For the quantitative analysis in this section, the scores on question 3 were excluded from the final exam score, as the goal was to see how student performance on the EiPE problems was related to scores on other types of problems. Including question 3 would have artificially increased the correlation between the SOLO ratings and student performance. Thus, the final exam, for purposes of this study, was worth 40 points.

The SOLO ratings of P, U, M, and R were converted to integer equivalents (P -> 1, U -> 2, M -> 3, and R -> 4) for quantitative analysis. An average SOLO rating was computed for each participant’s response on each problem. Indecisive ratings (e.g., a rating of “R/M”) by individual analysts were scaled down (in this case, to “M”). Prior to the consensus meeting, there were 16 such indecisive ratings for responses to problem 3b and 20 for problem 3c (out of 600 ratings per problem); after the meeting, there were no indecisive ratings on either problem.

Participants were divided into two groups for each problem, based on average SOLO rating. One group (labeled the “R/M” group) had an average SOLO ranking of at least 3.0 (which means that their response was at least multi-structural). The other group (labeled “U/P”) had an average SOLO ranking of at most 2.0, indicating that their response was unistructural at best.

Because the exam score and SOLO ratings were not uniformly distributed, nonparametric statistical tests were used for analysis. Specifically, the R statistical package [12], with a p value of .05 for significance was used.

#### Inter-rater Agreement

Inter-rater agreement was computed using Spearman’s rank correlation coefficient ρ, which is appropriate when there are more than two raters and the data is on an ordered scale. First, ρ was computed for each pair of raters, and then the average of these values was computed. The level of inter-rater agreement before and after the meeting to discuss ranks and attempt to come to consensus was also computed.

On problem 3b Spearman’s ρ ranged from 0.669 to 0.938 with an average of 0.817 before the raters compared their results. After negotiation, ρ ranged from 0.875 to 0.967, with an average of 0.921. On problem 3d, inter-rater agreement ranged from 0.827 to 0.930 with an average of 0.885 before rater negotiation. After rater negotiation, the correlation values ranged from 0.911 to 0.971, with an average of 0.936. All correlation values were significant at the p < .001 level. These are all high levels of agreement, and indicate that the analysts were consistent in their interpretation of the SOLO scale.

#### Problem 3b

There was a high correlation (Spearman’s ρ = 0.508, p < .001) between the average SOLO ranking on problem 3b and the students’ total exam scores (with problem 3 excluded).

On this problem, there were 74 students with responses in the R/M group (that is, the response was seen as at least multi-
structural), and 43 students in the U/P group (that is, these responses were regarded as unistructural or preoperational). Only 3 responses had average rankings between 2 and 3.

Looking at students’ exam scores by SOLO category (summarized in Table 3), the average exam score for the students in the R/M group was 23.2 (out of a possible 40), while the average score for the students in the U/P group was 15.1. This difference was highly significant (Wilcoxon rank sum test W=13689, p < .001).

### Table 3. SOLO Categorizations for Problem 3b

<table>
<thead>
<tr>
<th>Category</th>
<th>Combined categories - # (%)</th>
<th>Average Exam Score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational</td>
<td>74 (61.7)</td>
<td>23.2 (58.0)</td>
</tr>
<tr>
<td>Multistructural</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unistructural</td>
<td>43 (35.8)</td>
<td>15.1 (38.8)</td>
</tr>
<tr>
<td>Prestructural</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Problem 3d**

There was also a highly significant correlation between the average SOLO ranking on problem 3d and a student’s performance on the exam, with Spearman’s rho = 0.451, p < .001.

On problem 3d (summarized in Table 4 below), 55 students were in the R/M group, and 64 students were in the U/P group. There was only one response that received an average SOLO rank between 2 and 3. The mean final exam score for the R/M group was 24.5; for the U/P group it was 16.8. Again, this difference was highly significant (W = 14161, p < .001).

### Table 4. SOLO Categorizations for Problem 3d

<table>
<thead>
<tr>
<th>Category</th>
<th>Combined categories - # (%)</th>
<th>Average Exam Score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational</td>
<td>55 (45.8)</td>
<td>24.5 (61.2)</td>
</tr>
<tr>
<td>Multistructural</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unistructural</td>
<td>64 (53.3)</td>
<td>16.8 (42.0)</td>
</tr>
<tr>
<td>Prestructural</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For both problems 3b and 3d, the students in the “R/M” group received an approximately 50% higher score on the remainder of their exam than the students in the “U/P” group.

At the university where this exam was conducted, a 50% score is the pass/fail boundary. For both “explain in plain English” problems used in the exam, students in the “R/M” group earned more than 50% of the possible points on the remainder of the exam. The average exam score for students in the problem 3b “R/M” group was 58%, and the problem 3d “R/M” group students had an average test score of 61%. Students in the “U/P” group received less than 50% on the remainder of the exam, with an average of 39% for the problem 3b group and 42% for the 3d group. Thus, students who were able to explain what code does (at least at a multistructural level) were more likely to pass the course.

### 4. DISCUSSION

On the basis of these results, the null hypothesis is rejected – there is a correlation between code writing and explaining code for students in (at least) this particular course on data structures. The results for this CS2 class are therefore consistent with earlier studies of CS1 students.

However, no claim is made that the ability to write code is dependent upon the ability to explain code, or vice versa. To do so would be to make a well-known fallacy of statistical reasoning commonly identified as “correlation does not imply causation”. We believe that the more likely explanation for a correlation between writing code and explaining code is that both writing and explaining depend upon a common set of skills concerned with reasoning about programs. Research on the psychology of programming has demonstrated that, as expertise develops, a programmer’s knowledge is organized into more abstract, flexible forms, which would benefit both code writing and code explaining [1, 4, 5, 8, 13, 15].

#### 5. IMPLICATIONS FOR TEACHING

If reasoning about code is the underlying skill that is common to both code writing and code explaining, then the crucial pedagogical question is how to most efficiently develop that underlying skill.

Clearly, there are lessons about code writing that can only be learned by writing code. However, traditional pedagogical practices emphasize code writing to such an extent that almost all the active learning exercises (i.e., laboratory exercises and assignments) require novices to write code. For example, prior to the final exam analyzed in this paper, the students had completed two program writing assignments, worth half of their final grade. Many students who submitted those assignments, especially the students around the pass/fail boundary, probably expended significant amounts of time in relatively unproductive activities, such as a “scatter gun” approach to debugging, in which students essentially guess at what might be the cause of an error, because those students lack the reasoning skills to deduce the error. Such students might write code more productively if they first improved their ability to reason about code, through non-writing learning activities such as reading and explaining code. We believe that a judicious mix of code-writing and code-reading is the most effective way for students to ultimately develop good programming skills.

A pragmatic difficulty with using EiPE problems as summative assessment (e.g., in final exams) is that there are only a relatively small number of algorithms covered in an introductory data structures course. A student might rote learn appropriate EiPE answers for all code studied during the semester, and then in the final exam choose the appropriate answer to give based upon superficial features of a piece of code. The value of EiPE problems may therefore be more as formative assessment rather than summative assessment. For example, rather than have the instructor explain a piece of code, these empirical results and the recent successes of peer instruction [14] suggest that students gain a deeper understanding of course material by explaining the code to each other and by hearing the explanations of other students. The format of the problems used in this study, those that require students to first trace code for a specific case and then explain the code in the general case, may be useful for peer instruction during programming courses. Since peer instruction typically utilizes multiple choice “clicker” questions [14], students’ incorrect open-ended responses on written assessments could serve as reasonable multiple-choice distractors during peer instruction.

The ability to explain code clearly and concisely is a valued and necessary skill in industry and one CS educators strive to develop in their students, particularly at the upper-level undergraduate
level. For example, software engineering students are required to explain their code when they write detailed documentation, give technical presentations, or participate in peer code reviews. EiPE problems give novice and intermediate level students opportunities to practice these valuable skills on a smaller scale. Other such opportunities might include asking students to explicitly explain code to their partners in plain English during pair programming, or to instruct them to use EiPE as a guide when documenting their code.

In some educational institutions, there is an increasing emphasis on establishing that graduates, regardless of their degree, have acquired certain generic attributes, such as communication skills. If it is necessary to develop communication skills as part of a course on data structures, then code explanation tasks provide one possible avenue of doing so.

Educators should also keep in mind that many of the students in this study had difficulty articulating a clear and concise explanation of the code. It is also likely some students may be similarly confused when shown code during lecture and that instructors may overestimate how quickly or how well they understand it. Asking students to explain code they have just seen may be a useful means of self-assessment and prompt them to ask questions to further their understanding.

6. FURTHER WORK

While it was revealed in this study that many students could not explain code, no effort was made to determine why those students struggled. To establish this will require more direct means of studying students. The next phase of this work will involve one-on-one interaction with students, having them “think out loud” as they attempt to explain the purpose of pieces of code in an attempt to determine how they reason as they read code. If “why students struggle” can be precisely identified, we will be better positioned to develop educational materials and approaches designed to help those students.

7. REFERENCES


