Some Empirical Results for Neo-Piagetian Reasoning in Novice Programmers and the Relationship to Code Explanation Questions

Malcolm Corney and Donna Teague
Faculty of Science and Technology
Queensland University of Technology, Brisbane, QLD, Australia
{m.corney,d.teague}@qut.edu.au

Alireza Ahadi and Raymond Lister
Faculty of Engineering and Information Technology,
University of Technology, Sydney, Sydney, NSW, Australia
Raymond.Lister@uts.edu.au

Abstract
Recent research on novice programmers has suggested that they pass through neo-Piagetian stages: sensorimotor, preoperational, and concrete operational stages, before eventually reaching programming competence at the formal operational stage. This paper presents empirical results in support of this neo-Piagetian perspective. The major novel contributions of this paper are empirical results for some exam questions aimed at testing novices for the concrete operational abilities to reason with quantities that are conserved, processes that are reversible, and properties that hold under transitive inference. While the questions we used had been proposed earlier by Lister, he did not present any data for how students performed on these questions. Our empirical results demonstrate that many students struggle to answer these problems, despite the apparent simplicity of these problems. We then compare student performance on these questions with their performance on six explain in plain English questions.

Keywords: Novice programmer, CS1, neo-Piagetian.

1 Introduction
It is well documented within the research literature that many CS1 students around the world struggle to learn to program. For example, McCracken’s (2001) multinational ITiCSE working group collected data from over 200 CS1 students. The students were required to write code to evaluate arithmetic expressions. The average student score was only 21%. Most tellingly, many of the students did not write any code, as they spent their allotted 90 minutes trying to come up with a design for the program. Inspired by the McCracken working group, the ITiCSE 2004 “Leeds” Working Group (Lister et al., 2004) tested the reading and tracing skills of over 500 end-of-CS1 students, from twelve universities in seven countries. The average score for the students was 60%, with a quarter of the students performing at a level consistent with choosing options at random.

The literature on the novice programmer also abounds with reports on puzzling behaviours exhibited by novice programmers. For example, Thomas, Ratcliffe, and

Thomasson (2004) wrote about their frustrations at getting novices to use diagrams:

... when they might appropriately use [diagrams] themselves, weaker students fail to do so. They are often impatient when the instructor resorts to drawing a diagram, then amazed that the approach works. ... [also] providing [students] with what we considered to be helpful diagrams did not significantly appear to improve their understanding, ... This was completely unexpected. We thought that we were ‘practically doing the question for them’...

Perkins et al. (1986) described the behaviours of students who are “movers” or “stoppers”:

Many students disengage from the task whenever trouble occurs, neglect to track closely what their programs do by reading back the code as they write it, try to repair buggy programs by haphazardly tinkering with the code, or have difficulty breaking problems down into parts suitable for separate chunks of code.

Ginat (2007) described a similar approach to that of the tinkerer, but in more advanced students:

A hasty design may be based on some simplistic application of a familiar design pattern ... based ... on some premature association that seems relevant. Errors are not always discovered, as the test cases on which the program is tested are very limited. The devised program is batched, and “seems correct”. Then, an outside source (e.g., a teacher) points out a falsifying input. A patch is offered. Sometimes the patch is sufficient for yielding correctness, but more often than not, the patch is insufficient. An additional patch is offered; and the cycle of batch-&-patch continues.

1.1 Overview
Results such as those cited above recently led Lister (2011) to describe a neo-Piagetian perspective of the novice programmer. In the next section of this paper, we review his neo-Piagetian perspective. In section 3 we report upon empirical results for some neo-Piagetian inspired exam questions suggested by Lister. Section 4 then examines the relationship of those questions to code explanation questions.
2 Background: Neo-Piagetian Development

Lister (2011) proposed four stages of cognitive development of the novice programmer, based on neo-Piagetian theory. In the following subsections, we outline those neo-Piagetian stages. For a more detailed description, the reader should see Lister (2011).

2.1 Sensorimotor Stage

The first neo-Piagetian stage is the sensorimotor stage. Based upon the empirical results from Philpott, Robbins and Whalley (2007), Lister proposed that novices who trace code with less than 50% accuracy are at the sensorimotor stage. The afore-mentioned Leeds Working Group (Lister et al., 2004) demonstrated that there certainly exist students who cannot trace code with 50% reliability at the end of their first semester of learning to program.

Without the ability to reliably produce consistent results via tracing, novices at the sensorimotor stage see code as somewhat magical. That is, they do not experience an executing program as a deterministic machine.

2.2 Preoperational Stage

At the next stage of development, the preoperational stage, novice programmers can reliably trace code, but they do not routinely abstract from the code to see a meaningful computation performed by that code. Again, the Leeds Working Group (Lister et al., 2004) described students who were able to trace code reliably, but...

“... While working out their answer, none of these students volunteered any realization of the intent of the code ...” (p. 138).

Novice programmers at this stage are the novices that Thomas et al. (2004) wrote about:

“... providing [students] with what we considered to be helpful diagrams did not significantly appear to improve their understanding.”

For the preoperational novice, the lines in a piece of code are only weakly related. This stage in the development of the novice programmer is like the stage that Piaget identified in a child’s understanding of machines, such as bicycles, where the various parts are known to be necessary, but how the parts work together is not understood (Piaget, 1930, pp. 205–210). In an interview extract given in Traynor, Bergin, and Gibson (2006) a student described his approach to answering coding questions in an exam:

“... you usually get the marks by making the answer look correct. Like, if it’s a searching problem, you put down a loop and you have an array and an if statement. That usually gets you the marks ... not all of them, but definitely a pass”.

That student quoted by Traynor et al. was perhaps being cynical, but in the context of this paper, that student is describing all that a preoperational novice can do when they are required to write code – put down the elements that they recognise must be there, but not be able to fit those elements together in a way that produces correct code.

Without being able to see how the lines in a piece of code relate, a novice at the preoperational stage is likely to struggle with describing the purpose of a piece of code (“explaining”).

2.3 Concrete Operational Stage

Unlike students at the preoperational stage, students at the concrete operational stage can reason about abstractions of their code. They can, for instance, relate code to diagrams. They can also see how the individual lines in a piece of code work together to perform some overall computation. However, a defining characteristic of concrete thinking is that the abstract thinking is restricted to familiar situations (hence “concrete”).

The three archetypal manifestations of concrete thinking are the abilities to reason (1) about processes that are reversible, (2) with quantities that are conserved and (3) properties that hold under transitive inference. In the next three subsections, we review three exam questions that Lister (2011) identified as requiring these three types of reasoning.

2.3.1 Reversing

Figure 1 contains a question that Lister (2011) nominated as requiring the novice programmer to reason about reversing.

```
int temp = x[x.length-1];
for (int i=x.length-2; i>=0; --i)
  x[i+1] = x[i];
x[0] = temp;
```

Figure 1: A question that requires the concrete operational ability to reason about reversing (from Lister, 2011).

```
The purpose of the following code is to move all elements of the array x one place to the right, with the rightmost element being moved to the leftmost position:
```
```csharp
The purpose of the following code is to move all elements of the array x one place to the left, with the leftmost element being moved to the rightmost position.
```

2.3.2 Conservation

Lister (2011) identified one type of conservation in programming, which is the preservation of a specification across variation in the implementation. Figure 2 contains a question he nominated as requiring the novice programmer to reason about conservation. In that question, either of the options in each box could be right, depending upon what choices the novice has made in the other boxes. Thus, the novice needs to be able to see how the lines of code are related.
In plain English, explain what the following segment of Java code does:

```java
bool bValid = true;
for (int i = 0; i < iMAX-1; i++)
{
    if (iNumbers[i] > iNumbers[i+1])
    {
        bValid = false;
    }
}
```

This paper focuses on the types of reasoning that precede formal operational reasoning, so in this paper it is only necessary to further sharpen the reader’s understanding of concrete operational reasoning by describing how people who reason at the formal operational level differ from people reasoning at the concrete operational reasoning:

- They can reason about unfamiliar situations.
- They tend to begin with the abstract and move to the concrete.
- They reason with abstractions routinely, logically, consistently and systematically.
- They have a reflective capacity — an ability to think about their own thinking.
- They can perform hypothetico-deductive reasoning.

In the context of programming, hypothetico-deductive reasoning is nicely illustrated by an extract from Edwards (2004), in a paper where he argued that novice programmers needed:

"... practice in hypothesizing about the behavior of their programs and then experimentally verifying (or invalidating) their hypotheses. ... These activities are at the heart of software testing." (p. 27)

- They reliably manifest problems solving skills on unfamiliar problems. McCracken et al. (2001) defined problem solving as a five step process: (1) abstract the problem from its description; (2) generate subproblems; (3) transform subproblems into subsolutions; (4) recompose; and (5) evaluate and iterate.
3 Results for Reversing and Conserving
We placed into our end of semester exam the questions in Figures 1 and 2 which Lister (2011) had proposed as requiring concrete operational reasoning. This section discusses the results for those two questions.

3.1 Screening
Before performing the analysis below, we screened students, using two tracing questions from that same end of semester exam. The purpose in the screening was to eliminate from further study any students who were at the sensorimotor stage. Students who answered either of the two tracing questions incorrectly were eliminated from further study.

One of the two screening questions required students to determine the final values in five variables after a series of ten assignment statements. The first five assignment statements initialised each variable. The remaining five statements assigned values between these variables, and were designed to detect students who had any of the well known misconceptions about variables and assignment statements (du Boulay, 1988).

The other screening question required students to reason about two nested if statements. The conditions in the if statements involved comparisons among three integer variables, a, b and c. Each then and else part of the if statements results in the output of one of those variables. The question was framed as a multiple choice question, where students had to reason backward, from output to input. Specifically, students were asked “Which of the following values for the variables will cause the value in variable b to be printed?”

After this screening, 93 students remained in the sample for further analysis. These 93 students were considered to be reasoning at a level no lower than preoperational.

3.2 Reversing
When writing the solution to the problem in Figure 1, the student must recognise that the assignment \( x[i+1] = x[i] \) in the loop body needs to be replaced with either \( x[i] = x[i+1] \) or \( x[i-1] = x[i] \). We feel that such a change is the simplest of all the changes required, and is even a change within the grasp of any exam-savvy student reasoning at the preoperational level. Rather than indicating a low level of neo-Piagetian reasoning, an error on that line of code might simply be due to a student misunderstanding the question, perhaps because of poor English language reading skills. Therefore, we eliminated from the analysis of this question any student who did not make a correct change to that assignment statement in the loop body, which left us with 70 students in our sample. All of these students provided a four-line solution that resembled the code provided in the question.

Of the 70 students, only 45 (64%) provided a correct first line, in which they saved the leftmost element of the array to the temporary variable, and 38 students (54%) provided a correct final line, in which they assigned the temporary value to the rightmost position in the array. Only 37 students (53%) provided both a correct first line and a correct last line. We classify the 33 students (47%) who did not provide correct versions of both lines as clearly exhibiting preoperational reasoning. (Recall that these 33 students did provide a correct assignment in the loop body, and thus showed some understanding of the problem.)

Of the 37 students who provided a correct first, third and fourth line, 15 (41%) provided a correct version of the second line, the for loop. We classify those 15 students as clearly exhibiting concrete operational reasoning. We consider the remaining 22 of these 37 students to be exhibiting some degree of concrete operational reasoning. Among these 22 students, the most common errors were off-by-one errors. Often, the values specified in line 2 through which the control loop variable \( i \) would iterate were appropriate, in isolation, and so was the assignment statement on line 3, in the body of the loop. However, those two lines, in combination, were often not compatible. Perhaps with a little more careful checking, at least some of those students might have provided a correct solution.

In summary, for this question we see evidence for preoperational and concrete operational reasoning among our sample of students, who had passed a screen for sensorimotor reasoning.

3.3 Conservation
Table 1 shows student performance in the final exam on the concrete operational “choose from each box” task shown in Figure 2. The 40% of students who provided a correct solution (i.e. either ADEH or BCFG) are clearly exhibiting concrete operational reasoning. The 32% of students who provided ACFG show some signs of concrete operational reasoning, by virtue of choosing CFG. The remaining 28% of students are clearly exhibiting preoperational reasoning.

A possible threat to the validity of this question is the lengthy English instructions prior to the code. A student who reads English as a second language may be disadvantaged.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ADEH (correct)</td>
<td>8%</td>
</tr>
<tr>
<td>BCFG (correct)</td>
<td>32%</td>
</tr>
<tr>
<td>ACFG (close)</td>
<td>32%</td>
</tr>
<tr>
<td>Others (all wrong)</td>
<td>28%</td>
</tr>
</tbody>
</table>

Table 1: Student performance in the final exam on the concrete operational “choose from each box” task shown in Figure 2. (n=93)

3.4 Comparing Reversing and Conservation
Table 2 shows the relationship between student performance on these two questions, as a contingency table. Given that few students answered the code reversal problem with complete accuracy, we elected to just use the data on that question for how many students handled the end element correctly (i.e. lines 1 and 4). A \( \chi^2 \) test yielded \( p=0.08 \), which is higher than the traditional \( p=0.05 \) threshold for statistical significance.

With the given data, it is unclear whether the absence of a statistical relationship between the two questions is due to competence differences (i.e. the two questions
require different sorts of reasoning skills) or performance differences (i.e. the framing of the two questions test skills other than the ability to reason about code; see Chomsky, 1965). One obvious potential performance difference is the greater demands placed upon a student’s English language reading ability by the “select from the boxes” task. A more detailed study of this issue is warranted. An essential element of such a study is the use of two or more questions of each type, to assess the consistency of student performance within each type of question, before assessing the significance of consistency of student performance between these two types of questions.

<table>
<thead>
<tr>
<th>Select code from boxes (see Figure 2)</th>
<th>Write the reverse of a given shift; correct treatment of end element (i.e. correct line 1 and line 4, analogous to Figure 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>wrong</td>
<td>wrong: 36, right: 20, total: 56</td>
</tr>
<tr>
<td>right</td>
<td>wrong: 17, right: 20, total: 37</td>
</tr>
<tr>
<td></td>
<td>wrong: 53, right: 40, total: 93</td>
</tr>
</tbody>
</table>

Table 2: The contingency table for student performance on the two concrete operational questions. ($\chi^2 = 3.06, p = 0.08$)

4 Code Explanation and Concrete Reasoning

This section explores the relationships between the two concrete operational questions studied in the previous section and explain in plain English questions.

Explain in plain English questions were used extensively in the BRACElet project (Whalley, et al., 2006; Lister et al., 2006). However there has been some controversy as to whether these questions are really testing the ability of students to read and understand code (i.e. competence) or the ability of the students to express themselves in English (i.e. performance; see Simon, et al., 2009; Simon, 2009; Simon and Snowdon, 2011). If we see in our data a direct relationship between how well our students answer the concrete operational questions in Figures 1 and 2 and how well they answer explain in plain English questions, then that would suggest that the explain in plain English question requires, at the minimum, concrete operational reasoning skills.

4.1 The Six Explanation Questions

Our end of semester exam contained six explain in plain English questions. These questions, labelled (a) to (f) are shown in Figures 4 and 5, along with handwritten answers from a student who did the exam:

(a) This question was intended to be a benchmark of each student’s ability to express themself in English. Since the code does not contain either loops or arrays, it is simpler than the remaining explanation questions.

(b) This question was intended to test the student’s ability to explain code operating on arrays, using an `if` statement within a loop.

(c) In addition to the skills required to explain the previous question, this question required the student to reason about the effect of a `return` statement within a loop. The purpose in including this question was that it requires the same skills as question (f), with the exception of transitive inference.

(d) We regard this as our simplest question on loops and arrays.

(e) The basic purpose of this code is relatively simple — to find the position of a target value in a list. However, there are two details required in a completely correct answer. The first detail concerns the behaviour of the code if the target value is not found. The second detail concerns the behaviour of the code when the target value occurs more than once in the list.

(f) This question is similar to the question in Figure 3, that Lister asserted required transitive inference. However, our code in the exam was different from Figure 3 in two ways. The first difference is that the code we used in our exam was in Python. The second difference is our Python code does not set a flag, but instead (as in question (c)) breaks out of the loop with a `return` statement. A comparison of student performance on questions (c) and (f) is a test of Lister’s assertion that this question requires transitive inference.

4.2 Reversing and Explanation Questions

Row 1 of Table 3 shows the performance of the 93 students who passed our sensorimotor screening test, which was described earlier. For example, the column headed “(a)” shows that 89% of the whole sample answered the explanation question “(a)” correctly. The two hardest explanation questions (by far), were explanation questions (c) and (f), with only 36% and 31% of our whole sample answering those questions correctly. (Note that students’ answers to question (f) were marked as correct even if they failed to mention the indexing error that may be generated.) The remaining rows of Table 3 show the percentage of students who answered correctly each explanation question, given their performance on aspects of the concrete operational “shift left” question in Figure 1.

Row 2 of Table 3 shows the performance of the 33 students who (in addition to passing the sensorimotor screening test) provided a correct assignment statement within the body of the loop, but failed to provide a correct handling of the end element of the array (i.e. code like lines 1 and 4 in Figure 1). Only a quarter of these students could answer explanation questions (c) and (f) correctly. Less than half (45%) of these students answered explanation question (e) correctly.

Earlier in the paper, we surmised that writing the correct assignment in the `for` loop lay within the grasp of exam-savvy students reasoning at the preoperational level. The statistical data for Row 2 is consistent with that claim. Over half the students in Row 2 are able to correctly answer the two explanation questions (b) and (d). To do so, we believe a student need not understand...
QUESTION 18

This question contains a series of functions. For each function, describe the purpose of the function in one sentence that you should write in the corresponding box below that function. Do NOT give a line-by-line description of what the code does. Instead, tell us the purpose of the code:

(a) (1 mark)

```python
def function_a(x, y):
    if x > y:
        return x
    else:
        return y
```

Returns the bigger value of x or y, or y if they are equal.

(b) (1 mark)

```python
# x and y are lists of equal length
def function_b(x, y):
    c = 0
    for index in range(len(x)):
        if x[index] > y[index]:
            c = c + 1
    return c
```

Compares the list values at the same positions and returns the amount of numbers where the value in list x is bigger.

(c) (1 mark)

```python
# x and y are lists of equal length
def function_c(x, y):
    for index in range(len(x)):
        if x[index] > y[index]:
            return True
    return False
```

Goes through the list and checks if there is once a bigger value in x or a value in y (at the same index).

Question 18 continued overleaf...

Figure 4: The first three of the six explanation questions used in the exam paper.
Figure 5: The final three of the six explanation questions used in the exam paper.
**Table 3**: The percentages of students who answered each explain in plain English question correctly, given their performance on aspects of the concrete operational “shift left” question in Figure 1. The shaded cells indicate statistically significant differences in the percentages shown in the cells above and below the shaded cell.

<table>
<thead>
<tr>
<th>Row</th>
<th>Description</th>
<th>n</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
<th>(e)</th>
<th>(f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Whole sample</td>
<td>93</td>
<td>89%</td>
<td>76%</td>
<td>36%</td>
<td>82%</td>
<td>60%</td>
<td>31%</td>
</tr>
<tr>
<td>2</td>
<td>Correct assignment in for loop</td>
<td>33</td>
<td>82%</td>
<td>58%</td>
<td>24%</td>
<td>67%</td>
<td>45%</td>
<td>24%</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>p &lt; 0.04*</td>
<td>p &lt; 0.01*</td>
<td>p = 0.8</td>
<td>p = 0.01*</td>
<td>p = 0.01*</td>
<td>p = 0.3</td>
</tr>
<tr>
<td>4</td>
<td>End element handled correctly</td>
<td>22</td>
<td>100%</td>
<td>95%</td>
<td>27%</td>
<td>95%</td>
<td>68%</td>
<td>23%</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>p = 0.22</td>
<td>p = 0.78</td>
<td>p &lt; 0.01*</td>
<td>p = 0.78</td>
<td>p = 0.20</td>
<td>p &lt; 0.03*</td>
</tr>
<tr>
<td>6</td>
<td>Code entirely correct</td>
<td>15</td>
<td>93%</td>
<td>93%</td>
<td>73%</td>
<td>93%</td>
<td>87%</td>
<td>60%</td>
</tr>
</tbody>
</table>

**Table 4**: Complete contingency tables used to calculate the $\chi^2$ test probabilities in row 3 of Table 3. The column heading “w” (wrong) indicates data for students who answered that particular explanation question incorrectly.

<table>
<thead>
<tr>
<th>Explain in Plain English Questions (a) to (f)</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) right w</td>
<td>(b) right w</td>
</tr>
<tr>
<td>6 27 (82%)</td>
<td>14 19 (58%)</td>
</tr>
<tr>
<td>0 22(100%)</td>
<td>1 21 (95%)</td>
</tr>
<tr>
<td>6 49 (89%)</td>
<td>15 40 (73%)</td>
</tr>
<tr>
<td>p &lt; 0.04*</td>
<td>p &lt; 0.01*</td>
</tr>
</tbody>
</table>

**Table 5**: Complete contingency tables used to calculate the $\chi^2$ test probabilities in row 5 of Table 3.

<table>
<thead>
<tr>
<th>Explain in Plain English Questions (a) to (f)</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) right w</td>
<td>(b) right w</td>
</tr>
<tr>
<td>0 22(100%)</td>
<td>1 21 (95%)</td>
</tr>
<tr>
<td>1 14 (93%)</td>
<td>1 14 (93%)</td>
</tr>
<tr>
<td>1 36 (97%)</td>
<td>2 35 (95%)</td>
</tr>
<tr>
<td>p &lt;0.22</td>
<td>p = 0.78</td>
</tr>
</tbody>
</table>

**Table 6**: Contingency table comparing the performance on explanation questions (c) and (f) of the n=15 students who answered entirely correctly the concrete operational “shift left” question in Figure 1 ($\chi^2$ test, $p = 0.1$).

<table>
<thead>
<tr>
<th>Explain in plain English question (c)</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) wrong</td>
<td>(b) right</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>

**Table 7**: The middle row shows the percentages of students who answered each explain in plain English question correctly, given correct performance on the “select from the boxes” question in Figure 2. The rows beginning “4” and “6” are the same rows as in Table 3, for comparison.

<table>
<thead>
<tr>
<th>Explain in Plain English Questions (a) to (f)</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(b)</td>
</tr>
<tr>
<td>4 Handled end element</td>
<td>22</td>
</tr>
<tr>
<td>5 Select boxes correct</td>
<td>37</td>
</tr>
<tr>
<td>6 Entire shift correct</td>
<td>15</td>
</tr>
</tbody>
</table>
every aspect of that code. Like the student we quoted earlier from the paper by Traynor et al., who knew how to get half marks on a code writing task, without understanding the code he put down, a student doesn’t need to understand every aspect of the `for` statements in (b) or (d) to guess that the code will run across all elements of the list. After making such an assumption, a student can then answer the question by focussing solely upon the `if` within the loop, and its associated assignment statement.

4.2.1 Rows 2 & 4: Preoperational to Concrete
Row 4 of Table 3 shows the performance of the 22 students who succeeded at all the same tasks on the “shift” problem that the students in Row 2 were able to do, and who were also able to correctly handle the end element of the list (i.e. lines like 1 and 4 in Figure 1). However, these 22 students did not provide a suitable `for` statement (i.e. a line like line 2 in Figure 1). As discussed in section 3.2, these 22 students comprising row 4 exhibit some degree of concrete operational reasoning, whereas the students in row 2 exhibit preoperational reasoning.

A chi-square test was performed on each of the data forming rows 2 and 4 of the table. That is, for each explanation question, the raw data from which the two percentages in rows 2 and 4 were derived were used to perform a chi-square test. The resultant probability values for each column are shown in Row 3. (To assist others who may attempt to replicate our findings, the full contingency tables from which the probabilities were calculated are provided in Table 4.)

The shaded cells in Row 3 indicate the explanation questions for which the percentage in Row 2 is significantly different (i.e. p < 0.05) to the percentage in Row 4. All four of the easier explanation questions show a statistically significant improvement from Row 2 to Row 4. However, the two harder explanation questions (i.e. c and f) do not show a statistically significant improvement. Also, although the percentage of explanation question (e) rises between Rows 2 and 4 (from 45% to 68%), even with that increase almost one third of the students who could correctly handle the end element in the “shift” problem could not answer this explanation question. While explanation questions (b) and (d) are too difficult for quite a large percentage of the students in Row 2, these two questions were answered correctly by almost every student in Row 4.

To summarize this subsection: some degree of concrete operational reasoning tends to be both necessary and sufficient for answering correctly explanation questions (b), (d) and (e).

4.2.2 Rows 4 & 6: Growing Concrete Skills
Row 6 of Table 3 shows the performance of the 15 students who were able to provide a completely correct solution to the “shift left” problem. These 15 students comprising row 6 exhibit solid concrete operational reasoning, perhaps even formal operational reasoning.

A chi-square test was performed on each of the data forming rows 4 and 6 of the table. (As before, to assist others who may attempt to replicate our findings, full contingency tables for calculating these probabilities are provided in Table 5.) The resultant probability values for each column are shown in Row 5 of Table 3. The shaded cells in Row 5 indicate a statistically significant improvement on the two harder explanation questions (i.e. c and f) from Row 4 to Row 6.

A substantial minority of Row 6 students cannot answer these explanation questions. From our reading of incorrect student responses to (c), we conclude that many of these students did not understand that executing a `return` statement within a loop will immediately terminate the loop. Such a weak grasp of the `return` statement is consistent with Lister et al. (2004), who reported that misconceptions about `return` statements were the only misconceptions observed in that study.

4.2.3 Transitive Inference and Explanation
Recall that the purpose of explanation question (c) was that it required the same reasoning skills as question (f), with the exception of transitive inference. In this subsection, we compare student performance on those two explanation questions.

Student performance on both (c) and (f) is so poor that it is difficult to make any comparisons. In Row 2 of Table 3, only 24% of the students answered each of (c) and (f) correctly. In Row 4 of Table 3, only 27% and 23% of the students answered (c) and (f) correctly. These two explanation questions are too hard for most students represented in those two rows.

Row 6 of Table 3 is the only row where any comparison of (c) and (f) is at all viable. Here, the respective percentages are 73% and 60%, but that is for a tiny sample of only 15 students. Table 6 is a contingency table for that data. The resultant probability value is p = 0.1, which is above the standard p = 0.05 threshold. However, p = 0.1 does mean that the chance that the difference in the percentages is a statistical fluke is only 1 in 10. Given that, and the small sample size, our data does not fundamentally contradict Lister’s assertions about transitive inference, but it is at best only very weakly supportive. Further work is warranted.

4.3 Conservation and Explanation
Row 5 in Table 7 shows the percentages of students who answered each explanation question correctly, given correct performance on the "select from the boxes" question in Figure 2. The rows beginning “4” and “6” are the same rows as in Table 3, from the “shift” problem, for comparison.

Of particular interest in Table 7 is the data for the two harder explanation questions, (c) and (f). The percentages for “select from the boxes” on questions (c) and (f) lie between the percentages for the rows beginning “4” and “6”, from the “shift” problem — which may indicate that the difficulty of this particular conservation problem is beyond some concrete operational students who can handle the end element in the “shift” problem, but lies within the grasp of most concrete operational students who can correctly solve the entire “shift” problem. That, in turn suggests (but does
not prove) that the “select from the boxes” task used in this paper requires concrete operational reasoning.

5 Conclusion
Our empirical results support the claims made by Lister (2011). We see students who manifest preoperational reasoning skills by their poor performance on a reversal task (“shift left”) and a conservation task (“fill in the boxes”). However, while our data does not fundamentally contradict Lister’s assertions about transitive inference, our limited data is at best only very weakly supportive. For transitive inference, a larger study will be required.

While there may be some controversy as to whether the nature of the problem that students face with explanation questions is competence-related or performance-related, there is less doubt about students who, when explicitly supplied with code that moves all elements of a list (or array) one place to the right, cannot alter that supplied code to move all the elements one place to the left. Thomas et al. (2004) described the diagrams they gave their students as “practically doing the question for them” — how much more so for the “shift left” problem we gave our students? Such a question clearly establishes that there are students in our class who, at the end of their first semester of programming, are at the preoperational level of reasoning about code. It would be very interesting to see if the “shift left” problem we gave our students? Such a study will be required.

Preoperational students are woefully under-prepared for the rigours of traditional programming assignments. On such assignments, preoperational students can only flail about, exhibiting the behaviours described by Perkins et al. (1986), Thomas et al. (2004), and Ginat (2007) — behaviours which have puzzled and exasperated many, many CS1 teachers around the world. A neo-Piagetian perspective on the novice programmer actually positions these behaviours as normal behaviours to be expected in the long and torturous cognitive development of the novice programmer.

References
http://doi.acm.org/10.1145/1041624.1041673
http://crpitt.com/confpapers/CRPITV52Traynor.pdf