# Falling Behind Early and Staying Behind When Learning to Program

Alireza Ahadi & Raymond Lister

*University of Technology, Sydney*
*Australia*
*raymond.lister@uts.edu.au*

Donna Teague

*Queensland University of Technology*
*Australia*
*d.teague@qut.edu.au*

*Keywords*: *POP-I.B. barriers to programming, POP-II.A. novices, POP-V.A. Neo-Piagetian, POP-VI.E. computer science education research*

## Abstract

We have performed a study of novice programmers, using students at two different institutions, who were learning different programming languages. Influenced by the work of Dehnadi and Bornat, we gave our students a simple test, of our own devising, in their first three weeks of formal instruction in programming. That test only required knowledge of assignment statements. We found a wide performance difference among our two student cohorts. Furthermore, our test was a good indication of how students performed about 10 weeks later, in their final programming exam. We interpret our results in terms of our neo-Piagetian theory of how novices learn to program.

## 1. Introduction

Many computing educators have conjectured that success in learning to program requires a special talent or mental inclination. Some recent work in that area by Dehnadi and Bornat (Dehnadi and Bornat, 2006; Dehnadi, 2006) has generated much interest. They devised a test, consisting of nothing but assignment statements, which they gave to students who (it was believed) had no prior instruction in programming. They found that the test was a reasonable predictor of success in a first programming course. However, attempts at replicating those results with students from other institutions have met with mixed results (Caspersen, Larsen and Bennedsen, 2007; Bornat, Dehnadi and Simon, 2008; Lung, Aranda, Easterbrook and Wilson, 2008). Those mixed results have led to some clarifications and refinements on the original work (Bornat, Dehnadi, and Barton, 2012).

A key concept in Dehnadi and Bornat's work is consistency in the application of an algorithmic model of program execution. They believe that, prior to formal instruction in programming, a student's model of program execution need not be a correct model. Instead, what matters is whether a student applies a model consistently. For example, when hand executing assignment statements, prior to receiving formal instruction on programming, a student might mistakenly assign the value from left to right rather than right to left, but as long as the student applied that model consistently, prior to receiving formal instruction on programming, Dehnadi and Bornat found that the student stood a better chance of learning to program from subsequent formal instruction.

While our work was influenced by Dehnadi and Bornat, our pedagogical interest is in the early identification of students (as early as weeks 2-3 of formal instruction) who are in danger of failing their introductory programming course. Since, after 2-3 weeks, our students have been taught the correct model for assignment statements, our interest is in whether the students have learnt that correct model. We have therefore designed our own test, which we describe in the next section.

## 2. The Test

The specific test questions shown below are from the Python version of the test, used at the Queensland University of Technology (QUT). Given that this test is only concerned with assignment statements on integer variables, the Java version of the test used at the University of Technology, Sydney (UTS) is almost the same. The most common change in the Java version is that each line of code ends with a semicolon. The UTS Java version has been presented in full elsewhere (Ahadi and Lister, 2013).

The design of our test was influenced by our prior work on applying neo-Piagetian theory to programming (Lister, 2011; Teague and Lister, 2014). In that work, we proposed a three stage model of the early stages of learning to program, which are (from least mature to most mature):

- **Sensorimotor:** The novice programmer has an incorrect model of program execution.
- **Preoperational:** The novice can reliably manually execute ("trace") multiple lines of code. These novices often make inductive guesses about what a piece of codes does, by performing one or more traces, and examining the relationship between input and output.
- **Concrete operational:** The novice programmer reasons about code deductively, by reading the code itself, rather than using the preoperational inductive approach. This stage is the first stage where students begin to show a purposeful approach to writing code.

## 2.1 Semantics of Assignment Statements — Questions 1(a) and 1(b)

Questions 1(a) and 1(b) tested whether a student understood the semantics of assignment statements—that is, the value on the right of the assignment is copied to the left, overwriting the previous value. The specific questions were as follows:

Q1 (a). In the boxes, write the values in the variables after the following code has been executed:

```
a = 1
b = 2
a = 3
```

The value in `a` is $\boxed{\phantom{0}}$ and the value in `b` is $\boxed{\phantom{0}}$

Q1 (b). In the boxes, write the values in the variables after the following code has been executed:

```
r = 2
s = 4
r = s
```

The value in `r` is $\boxed{\phantom{0}}$ and the value in `s` is $\boxed{\phantom{0}}$

Dehnadi and Bornat used similar questions. Du Boulay (1989) summarised a number of problems that students have with variables and assignment statements. One of the problems he described was the analogy of a variable as a "box", leading to the misconception that a variable may hold more than one value, and thus the novice does not realize that the old value in a variable is overwritten by a new value. The above two questions are intended to detect students who have misconceptions like these. According to our neo-Piagetian model, students struggling with the above two questions are at the sensorimotor stage of learning to program.

## 2.2 Effect of a Sequence of Statements — Question 1(c)

Question 1(c) tested whether a student understood the effect of a sequence of assignment statements; that the statements were executed one at a time. The specific question was as follows:

Q1 (c). In the boxes, write the values in the variables after the following code has been executed:

```
p = 1
q = 8

q = p
p = q
```

The value in `p` is $\boxed{\phantom{0}}$ and the value in `q` is $\boxed{\phantom{0}}$

Dehnadi and Bornat used a similar question. Some students mistakenly interpret this code as swapping the values in variables `p` and `q`, which Pea (1986) called an "intentionality bug", where novices believe "there is a hidden mind somewhere in the programming language that has intelligent interpretive powers". (While sequence also matters in Q1(a) and Q1(b), the intentionality bug in Q1(c) exposes novices who apply sequence weakly.) A student who can do Q1(a) and 1(b), but who struggles with this question, is working at the late sensorimotor / early preoperational stage.

### 2.3 Tracking Intermediate Variable Values — Questions 1(d) and 1(e)

Even when students understand both assignment statements and the effect of a sequence of statements, they can still make frequent errors because they cannot reliably track the changing values in variables through a series of assignment statements. In many cases this is because the students try to retain the variable values in their mind (i.e. working memory), rather than write down those values. Another source of error is that students use arbitrary and error prone ways of recording the variable values on paper (Teague and Lister, 2014). Questions 1(d) and 1(e) were designed to test whether a student could track the changing values of three variables in a sequence of three assignment statements (i.e. the three statements following the initialization of the three variables):

Q1 (d). In the boxes, write the values in the variables after the following code has been executed:

```
x = 7
y = 5
z = 3

x = y
z = x
y = z
```

The value in `x` is ☐ `y` is ☐ and `z` is ☐

Q1 (e). In the boxes, write the values in the variables after the following code has been executed:

```
x = 7
y = 5
z = 0

z = x
x = y
y = z
```

The value in `x` is ☐ `y` is ☐ and `z` is ☐

Dehnadi and Bornat used similar questions. According to our neo-Piagetian model, a student who can do these questions is at least in the middle range of the preoperational stage.

### 2.4 Inductive Reasoning — Question 1(f)

A novice who answers Question 1(f) correctly but who answers incorrectly Question 2 (see next section) is probably reasoning about code inductively and thus is working at the preoperational stage:

Q1 (f). In part (e) above, what do you observe about the final values in `x` and `y`? Write your observation (in one sentence) in the box below.

Sample answer: *the original value in x is now in y, and vice versa.*

### 2.5 Deductive Reasoning and Code Writing — Questions 2 and 3

Questions 2 and 3 were aimed at the novice who reasons in a concrete operational fashion:

Q2. The purpose of the following three lines of code is to swap the values in variables `a` and `b`, for any set of possible values stored in those variables.

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

In one sentence that you should write in the box below, describe the purpose of the following three lines of code, for any set of possible initial integer values stored in those variables. Assume that variables `i`, `j` and `k` have been declared and initialized.

```
j = i
 i = k
 k = j
```

Sample answer: *it swaps the values in variables i and k.*

Q3. Assume the variables `first` and `second` have been initialized. Write code to swap the values stored in `first` and `second`.

Sample answer:
```
  temp = first
 first = second
second = temp
```

Du Boulay (1989, p. 290) described some of the problems novices might have when attempting Q3: *Many students get these assignment statements in the wrong order and express individual assignments back to front. Difficulty in expressing the overall order of the assignments may be due to a lack of regard for the sequential nature of the three* [lines of code, that] *look a lot like three equations which are simultaneous statements about the properties of* [the three variables] *rather than a recipe for achieving a certain internal state ...*" According to our neo-Piagetian model, students who struggle with Q2 and Q3 in these (and other) ways are students at the preoperational stage (or lower). Students who correctly answer both Q2 and Q3 are probably at the concrete operational stage.

## 3. The Conduct and Grading of the Test — including some threats to validity

The introductory programming courses at both institutions comprised a 13 week semester where classes each week comprised a two hour lecture and, commencing in week 2, 2-3 hours of tutorial and/or lab classes. Students completed our test at the start of their lecture in either week 2 (at QUT) or week 3 (at UTS). The test was presented to the students on a single piece of paper, printed on both sides. At QUT, we eliminated from our data the small number of students who scored zero on the test, as those students had probably not attended the week 1 lecture. For consistency, at UTS we also eliminated data from students who scored zero. The UTS test contained an extra question that, for consistency, we have subsequently ignored. Some results for the UTS test with that extra question have been published earlier (Ahadi and Lister, 2013). As the test did not contribute to a student's final grade, the students may have had little motivation to perform well on the test, but equally they had little motivation to cheat. We stopped the test after around 15 minutes. Very few if any students were still working on the test when we called a stop.

Questions 1(a) to 1(f) were all worth 1 point, as were Q2 and Q3, for a total of 8 points. No fractional points were awarded — answers were treated as being either right or wrong, but English language issues in Q2, and syntactic errors in Q3, were ignored as long as a student's intention was clear.

## 4. Results: Falling Behind Early...

There are many differences in what and how the UTS and QUT cohorts are taught. For example, the UTS cohort was taught Java while the QUT cohort was taught Python. Our primary interest is in finding patterns in our results that are common to both institutions, as those patterns are more likely to generalise to other institutions.

Figures 1 and 2 show the distribution of student scores on the test, at UTS and QUT respectively. While the respective distributions have a different shape, a common feature of both distributions is the wide variation in test scores, spanning the entire range of possible marks.

As is always the case when grading students, it is one thing to assign a score to a student, but it is another thing entirely to know what that score means — for example, are the students who scored 4 on this test qualitatively different, as a general rule, from students who scored 6? Tables 1 (for UTS) and 2 (for QUT) address exactly that sort of question. These tables show the percentage of students who answered correctly each part of the test, broken down by total test score. In the remainder of this section, we describe the results in those two tables. (Sections 4.1 and 4.2 below go to some pains to introduce and explain the information displayed in those tables.)

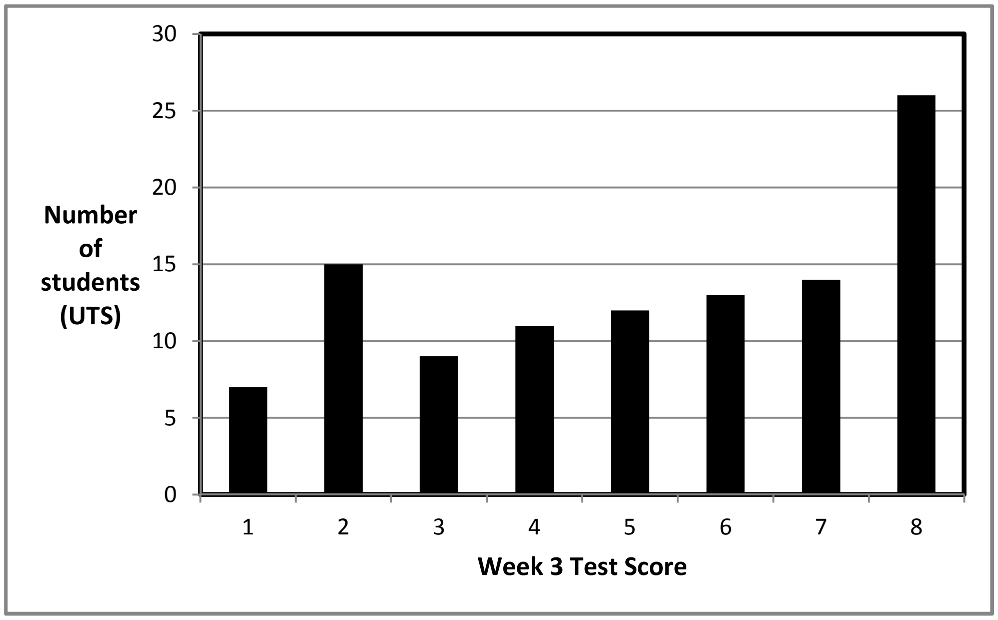

*Figure 1 — Distribution of student total scores on the test at UTS* (*N = 107*)

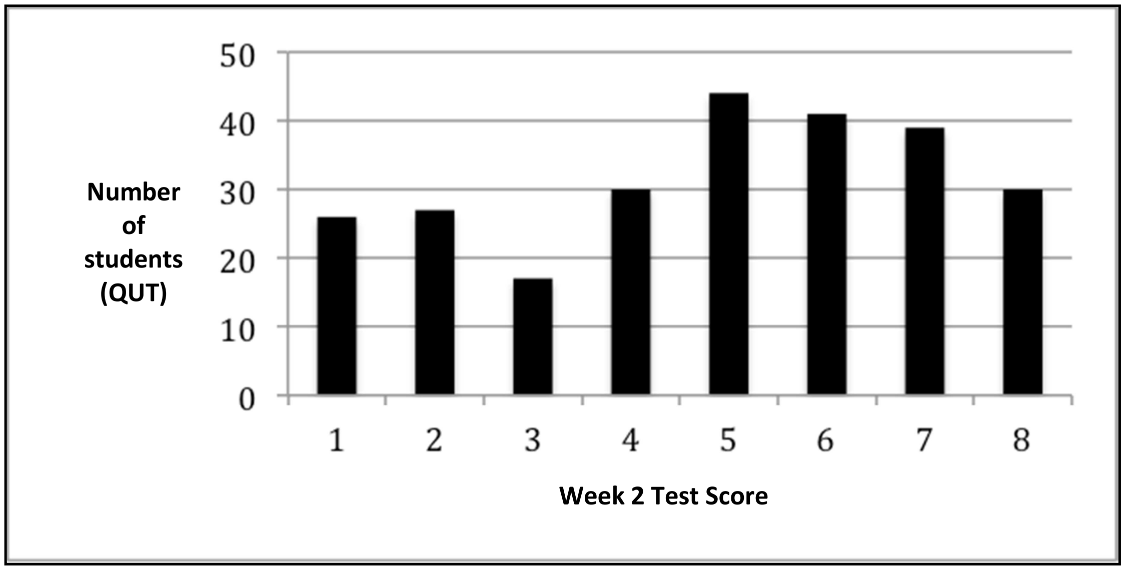

*Figure 2 — Distribution of student total scores on the test at QUT* (*N=254*)

## 4.1 Semantics of Assignment Statements — Questions 1(a) and 1(b)

At UTS (see Table 1), among the 15 students who scored 2 out of the possible 8 on the test, 93% answered Q1 (a) correctly (i.e. only one student answered incorrectly). All 15 students answered Q1 (b) correctly. However, for these 15 students, a performance difference of 93% and 100% on Q1 (a) and (b) is not statistically significant.

Of the 11 UTS students who scored 4 on the test, all answered Q1 (a) correctly and all but one student answered Q1 (b) correctly (i.e. 91%). In general, irrespective of their total score on the test, UTS students did very well on questions 1(a) and 1(b).

| Week 3 Test Score | UTS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | N | Sensorimotor | | | Preoperational | | | Concrete Operational | |
| | | assignment | | sequence & tracking values | | | induction | deduction | |
| | | Q1(a) | Q1(b) | Q1(c) | Q1(d) | Q1(e) | Q1(f) | Q2 | Q3 |
| 2 | 15 | 93% | 100% | ▌7% | 0% | 0% | 0% | 0% | 0% |
| $\chi^2$ | | | | * | *** | *** | *** | *** | |
| 4 | 11 | 100% | 91% | ▌45% | 55% | 27% | 27% | 36% | 18% |
| $\chi^2$ | | | | ** | ** | ** | | | |
| 6 | 13 | 92% | 92% | 92% | 100% | 85% | ▌54% | 31% | 54% |
| $\chi^2$ | | | | | | | *** | *** | *** |
| 8 | 26 | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% |
| 1 to 8 | 107 | 95% | 89% | 64% | 64% | 56% | 50% | 44% | 53% |

*Table 1 — The percentage of UTS students who answered correctly each part of the test, broken down by total score. Cells containing asterisks indicate a statistically significant difference in the two percentages above and below the asterisk(s) ($\chi^2$ test, * $p \leq 0.05$, ** $p \leq 0.01$ and *** $p \leq 0.001$). A thick vertical bar indicates a statistically significant difference in the two percentages to the left and right of the bar ($\chi^2$ test, but only at the $p \leq 0.1$ level). All $\chi^2$ tests were performed on the raw numbers from which the percentages were calculated.*

| Week 2 Test Score | QUT | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | N | Sensorimotor | | | Preoperational | | | Concrete Operational | |
| | | assignment | | sequence & tracking values | | | induction | deduction | |
| | | Q1(a) | Q1(b) | Q1(c) | Q1(d) | Q1(e) | Q1(f) | Q2 | Q3 |
| 2 | 27 | 59% | 70% | ▌26% | 26% | 15% | 4% | 0% | 0% |
| $\chi^2$ | | * | | *** | ** | ** | | * | |
| 4 | 30 | 87% | 83% | 80% | 63% | 53% | ▌13% | 13% | 7% |
| $\chi^2$ | | | * | * | *** | *** | *** | * | * |
| 6 | 41 | 88% | 98% | 98% | 98% | 88% | ▌66% | ▌34% | 32% |
| $\chi^2$ | | * | | | | * | *** | *** | *** |
| 8 | 30 | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% |
| 1 to 8 | 254 | 78% | 83% | 76% | 72% | 65% | 42% | 33% | 31% |

*Table 2 — The percentage of QUT students who answered correctly each part of the test, broken down by total score. The cells containing asterisks, and also the thick vertical bars between some cells, indicate the same types of statistically significant differences as in Table 1.*

The QUT students represented by Table 2 also did fairly well on questions 1(a) and 1(b). The only exception is the performance on 1(a) of students who scored 2 on the test. Only 59% of those 27 students answered that question correctly. Of the 30 QUT students who scored 4 on the test, 87%

answered Q1 (a) correctly. In Table 2, between those two percentages for Q1(a) (i.e. 59% and 87%), there is a grey cell containing an asterisk, which indicates that the difference in these two percentages is statistically significant ($\chi^2$, $p \leq 0.05$). In the two columns for questions 1(a) and 1(b), there are two other grey cells containing asterisks; thus, while the QUT students represented by Table 2 did fairly well on questions 1(a) and 1(b), those QUT students with higher overall scores on the entire test did statistically better on those questions.

In summary, on inspection of both Table 1 and Table 2, among the students at both institutions who scored 2 or higher on the test, most had a good grasp of the semantics of assignment statements.

## 4.2 Effect of a Sequence of Statements — Question 1(c)

At UTS (see Table 1), among the 15 students who scored 2 out of the possible 8 on the test, only 7% (i.e. 1 student) answered Q1 (c) correctly. Among the students who scored 4 on the test, 45% answered Q1 (c) correctly. As indicated by the grey cell between those two percentages, which contains an asterisk, the difference in these percentages is statistically significant. Below that 45% in Table 1, another grey cell, containing two asterisks, indicates that there is a statistically significant difference ($p \leq 0.01$) between the students who scored 4 and students who scored 6 (i.e. 45% vs. 92%).

While the percentages for Q1 (c) at QUT are different (see Table 2), the test for statistical significance shows the same pattern at both institutions — students who scored 2 did poorly on Q1(c), while students who scored 4 did significantly better, but not as well as students who scored 6 or 8.

In both Table 1 and 2, in the row for the students who scored 2, there is a thick vertical bar between the cells representing Q1 (b) and (c). This vertical bar, and the other vertical bars like it throughout both tables, indicate a statistically significant difference between the two horizontally adjoining cells ($\chi^2$, but $p \leq 0.1$). There is another thick vertical bar in Table 1 between cells in the columns for Q1 (b) and Q1 (c), in the row for students who scored 4, but there is no corresponding vertical bar in Table 2.

In summary, on inspection of both Table 1 and Table 2, most students who scored 2 had a poor grasp of sequence. Most of the students with higher scores on the test had a better grasp of sequence.

## 4.3 Tracking Intermediate Variable Values — Questions 1(d) and (e)

As described in section 2, questions 1(d) and 1(e) were designed to test whether a student could track the changing values of three variables in a sequence of three assignment statements. In summary, on inspection of both Table 1 and Table 2, only the students who scored 6 or higher could reliably track the values in variables. Thus most students scoring 6 or higher were preoperational or higher.

## 4.4 Inductive Reasoning — Question 1(f)

As described in section 2, question 1(f) was designed to identify students who can make reasonable inductive guesses about the function of a piece of code based upon the input/output behaviour. Since a student could not be expected to answer Q1 (f) correctly if that student had answered Q1 (e) incorrectly, it is the difference in percentages between Q1 (f) and Q1 (e) that is of interest, especially statistically significant differences (i.e. the thick vertical bars between those two table columns).

At both institutions, most students who scored 2 on the test performed poorly on both Q1 (e) and Q1(f). At QUT, among students who scored 4, there is a statistically significant difference between performance on Q1 (e) and Q1 (f), but not at UTS. There is, however, a statistically difference at both institutions among students who scored 6.

At both institutions, when looking down the table column for Q1 (f), it is apparent that most students who scored 2 or 4 did very poorly on this question, while the students who scored 6 exhibited mixed performance. Only the group of students who scored 8 on the test did very well on this question.

For this question, the only clear result that applies across both institutions is that students who scored 6 on the test tended to do well on the Q1 (e) tracing question but did significantly worse on the Q1 (f) inductive reasoning question.

### 4.5 Deductive Reasoning and Code Writing — Questions 2 and 3

On none of the overall test scores, at either institution, was there a statistically significant difference in the performance on Q2 and Q3. In both Tables 1 and 2, the only group of students who did well on both Q2 and Q3 were the students who scored a perfect 8 on the test.

We did not survey our students to establish any prior knowledge in programming, since self reporting is notoriously unreliable, but the results for Q2 and Q3 suggest that most students who scored 6 or less on this test are unlikely to have had any useful prior experience of programming.

### 4.6 A Neo-Piagetian Summary of Tables 1 and 2

On inspection of both Table 1 and Table 2, students with a total score of:

- 2 tended to have a grasp of the semantics of individual assignment statements but a poor grasp of sequence, and were thus working at the late sensorimotor / early preoperational stages.
- 4 were showing some ability to track values but many struggled with inductive reasoning, so we characterise this group of students as being early to mid-range preoperational.
- 6 were usually successfully tracking values and a majority could perform inductive reasoning, so we characterise this group of students as being late preoperational.
- 8 were the only group of students who performed consistently well on Q2 and Q3, so we characterise this group of students as being concrete operational.

## 5. Results: ... and Staying Behind

This section examines the relationship between performance on the test held early in semester and performance on the final exam at the end of the 13 week semester.

### 5.1 UTS Multiple Choice Exam

At UTS, the exam was entirely multiple choice. Figure 3 shows the probability that a UTS student would finish in the top half of the class, as a function of their performance on the week 3 test. The size of each black disc indicates the number of students who received that week 3 test score (i.e. the size of the discs is proportional to the size of the bars in Figure 1). The linear regression calculation represented by the dashed line was weighted according to the size of the discs. This was done by

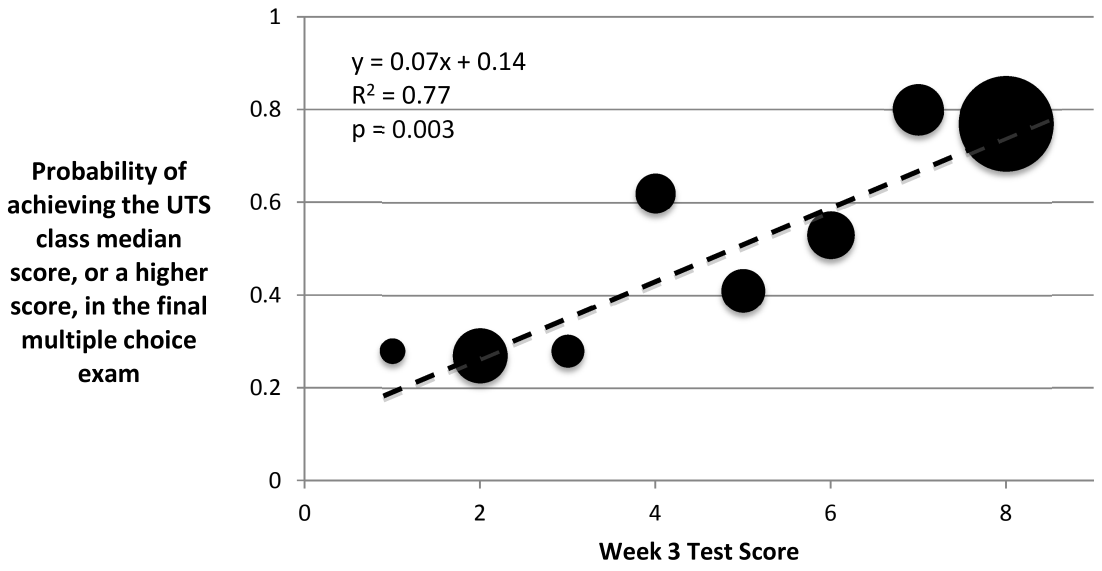

*Figure 3 — UTS student scores on the test versus performance on the final exam (N=107).*

performing the regression with 26 duplicate data points for test score = 8, 13 duplicate data points for test score = 6 and so on, for all test scores. All regression lines in subsequent figures were calculated this same way.

The use of the median in Figure 3 facilitates comparisons across institutions, since student ability and exam difficulty varies across institutions. Of course, approximately half of all students must perform above the median, and half below. However, that condition would still be satisfied if the regression line in Figure 3 was horizontal. In fact, from a pedagogical point of view, it would be best if that line of regression was horizontal, since the end-of-semester fate of a student should not be strongly attributable to their performance as early as the third week of a 13 week semester — but on the contrary, Figure 3 shows that many students did not recover from their slow start to the semester.

By the end of a 13 week semester, students had of course covered many more programming topics than the assignment statement tested in week 3. At UTS, approximately half the final exam covered basic object-oriented concepts, while the other half emphasized common 3GL searching algorithms and quadratic sorting algorithms. The following question indicates the general level of difficulty:

> This question refers to the Linear Search algorithm, studied in lectures, for an array “s” where the elements are stored in ascending order, and the final position in the array is stored in a variable “last”. The search should terminate as soon as either the value in variable "e" is found in the array, or it is established that the value is not in the array. Using a variable “pos” to scan along the array, the correct loop is:

```
(a) while ( (pos<=last)          &&    (pos < e) )      ++pos;
(b) while ( (pos<=last)          &&    (s[pos] < e) )   ++pos;
(c) while ( (s[pos]<=s[last])    &&    (pos < e) )      ++pos;
(d) while ( (s[pos]<=s[last])    &&    (s[pos] < e) )   ++pos;
```

## 5.2 QUT Multiple Choice Questions

The final exam at QUT comprised two parts: a set of multiple choice questions and a set of questions that required students to write Python code. Figure 4 shows the probability that a student would finish in the top half of the class for the multiple choice part of the exam, as a function of their performance on the week 2 test. As was also the case for UTS, Figure 4 shows that QUT students who performed poorly on the test — held in week 2! — were unlikely to overcome their poor early start to the semester and finish in the top half of the class.

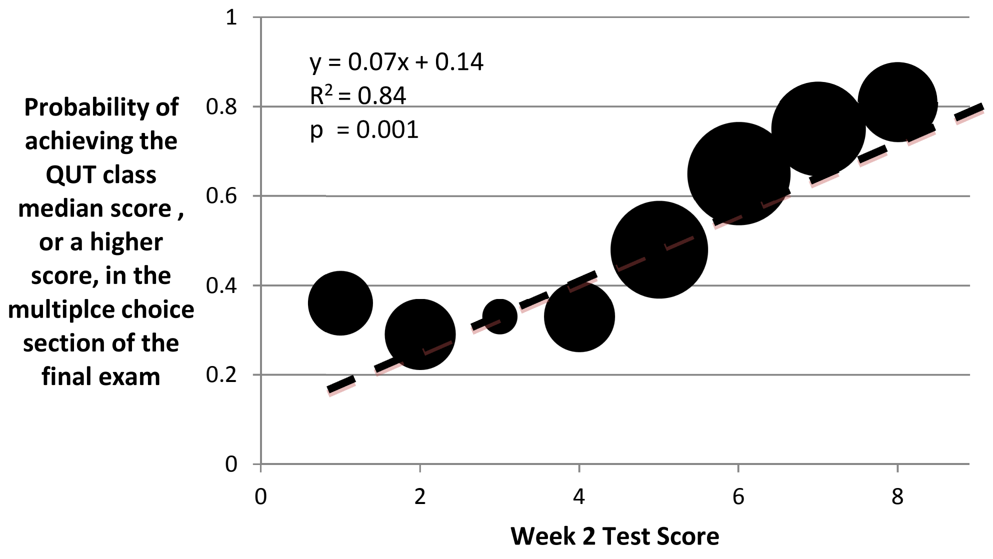

*Figure 4 — QUT student scores on the test versus performance on the multiple choice component of the final exam (N=254).*

## 5.3 QUT Short Answer Questions

While Figures 3 and 4 both describe a clear relationship between the test near the start of the semester and performance on the end-of-semester multiple choice questions, perhaps that relationship can be attributed to the relatively simple nature of multiple choice questions? For example, multiple choice questions do not require a student to write code. Figure 5 tests that idea. It shows the probability that a QUT student would finish in the top half of the class, as function of their performance on the week 2 test, for the short answer part of the QUT exam. This figure is similar to Figures 3 and 4 — the resemblance between Figure 5 and Figure 4 is uncanny.

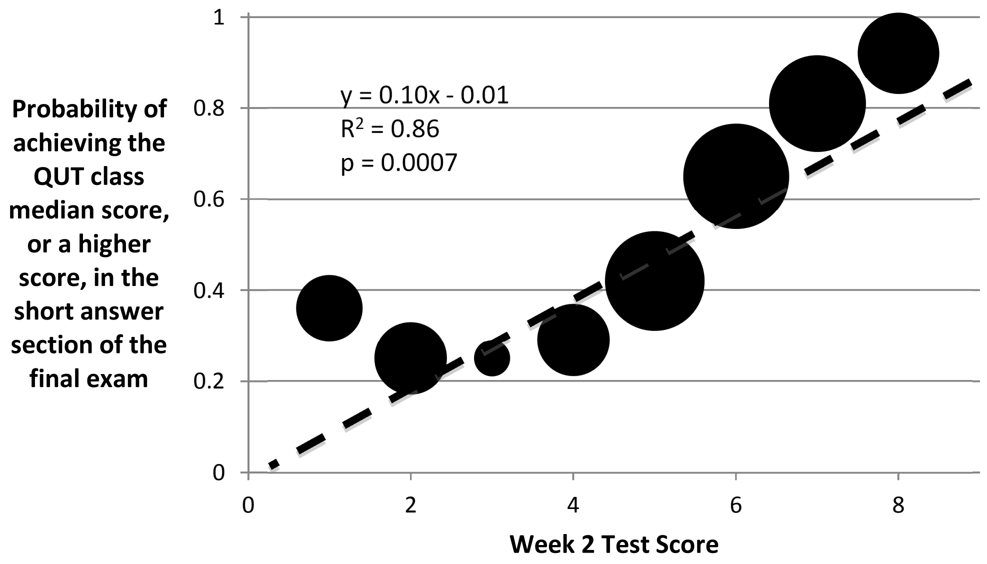

*Figure 5 — QUT student scores on the test versus performance on the short answer section of the final exam (N=254).*

One of the short answer questions in the exam is illustrated by the code shown below. The students were given the code on the left, which "rotates" the values in the list `items` one place to the left, with the leftmost value moving to the rightmost position. The students were required to write code to do the opposite transformation; that is, write code to rotate the values in array `items` one place to the *right*, with the rightmost item moving to the leftmost position. The solution is shown below:

*"Rotate Left" Code Given to the Students*

```
temp = items[0]
for index in range(len(items)-1):
     items[index] = items[index + 1]
items[len(items) - 1] = temp
```

*"Rotate Right" Code Required from Students*

```
temp = items[len(items) - 1]
for index in range(len(items)-1,0,-1):
     items[index] = items[index - 1]
items[0] = temp
```

As part of the instructions for this question, students were effectively given the **`for`** loop header required in their answer, so the question was marked out of 3, with one point for each of the remaining three lines of code. Figure 6 shows the probability that a student scored 2 or 3 for this question, as a function of their performance on the week 2 test. This graph is similar to the three earlier graphs for performance on final exams — our results are robust, across the two institutions, and also across multiple choice and short answer questions.

In neo-Piagetian terms, this "Rotate Right" short answer question requires the student to manifest concrete operational reasoning (Lister, 2011). In section 4.6, we provided a neo-Piagetian summary for Tables 1 and 2. Building on that earlier summary, we now provide a neo-Piagetian summary where we contrast the performance of students in Table 2 (i.e. at week 2 of semester) and their end of

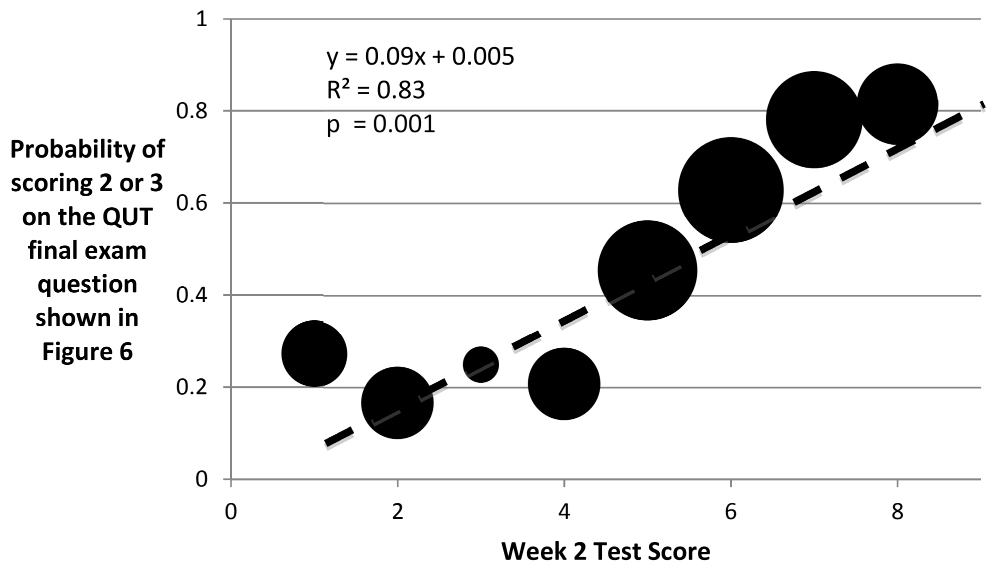

*Figure 6 — QUT student scores on "Rotate Right" short answer question (N=254).*

semester performance at QUT on the "Rotate Right" problem (as shown in Figure 6). Students with a total score in Table 2 of:

- 2 or 4 were characterised as being spread from sensorimotor to mid-preoperational. By the end of semester, around 20% of those students manifested concrete operational reasoning on the "Rotate Right" problem.
- 6 were characterised as late preoperational. By the end of semester, around 60% of these students manifested concrete operational reasoning on the "Rotate Right" problem.
- 8 were characterised as concrete operational. By the end of semester, around 80% of these students manifested concrete operational reasoning on the "Rotate Right" problem. (N.B. the remaining 20% did not "go backwards", as the week 2 test was on assignment statements only, whereas the "Rotate Right" problem tested more demanding concepts and skills.)

## 6. Conclusion

In this paper, we have described a test on novice programmers, in weeks 2 and 3 of semester, with students from two different institutions, where (among other pedagogical differences) the students are taught two different programming languages. At both institutions we found a wide performance difference among each student cohort on that test. Furthermore, that early test is a good indication of how students performed about 10 weeks later, in their final exam. In terms of neo-Piagetian theory, students who exhibit lower neo-Piagetian stages in the early test are unlikely to manifest the higher concrete operational stage of reasoning in the final exam.

People who believe that programming requires an innate talent may feel justified by our results. While our results do not disprove the existence of an innate talent for programming, we do not subscribe to that view. As we have summarised in this paper, in earlier work we have developed a neo-Piagetian theory of how novices learn to program. Neo-Piagetian theory is based upon the constructivist principle that cognitive skills are primarily learnt, not innate. Our neo-Piagetian perspective leads us to view the curriculum for programming as comprising two dimensions. On one dimension are the nuts and bolts of how programming languages work. That dimension is emphasised in today's classroom. The other and more neglected dimension comprises the skills for reasoning about programs, sometimes referred to as the notional machine (du Boulay, 1989), but which we think of in neo-Piagetian terms. This dimension is often not explicitly taught, especially in the first few weeks of learning to program. We believe that, with every increment along the "nuts and bolts" dimension (i.e. with every new programming construct taught), all the neo-Piagetian stages of reasoning need to be explicitly reprised. In the test we used in weeks 2 and 3 of semester, questions 1(f), 2, and 3 represent the types of learning exercises that students need when they are introduced to assignment statements. As a further example, the Bubblesort algorithm could be introduced well before loops are explicitly taught, using implicit "uncompressed" loops (Milner, 2008).

We close by speculating, from a neo-Piagetian perspective, on Dehnadi and Bornat's claim that people who apply a consistent model of program execution are more likely to learn to program, even when their model is wrong. Perhaps those people enjoy an early affective advantage, not a cognitive advantage. That is, people who show an early preference for consistency may be especially well motivated to perform the deep learning required to push through the earlier neo-Piagetian stages and gain the consistency of reasoning that only begins at the concrete operational stage.

## 7. Acknowledgements

The grant for this work was from the Office of Learning & Teaching, of the Australian Government.

## 8. References

Ahadi, A. and Lister, R. (2013). *Geek genes, prior knowledge, stumbling points and learning edge momentum: parts of the one elephant*? Ninth International Computing Education Research Workshop (ICER '13). ACM, USA, pp. 123-128. http://doi.acm.org/10.1145/2493394.2493416

Bornat, R., Dehnadi, S., and Simon (2008) *Mental models, consistency and programming aptitude*. Tenth conference on Australasian Computing Education (ACE '08). pp. 53-61. http://crpit.com/confpapers/CRPITV78Bornat.pdf

Bornat, R., Dehnadi, S., and Barton, D. (2012) *Observing Mental Models in Novice Programmers*. 24th Annual Workshop of the Psychology of Programming Interest Group, London. http://www.ppig.org/papers/24/8.Observing_mental_models-Richard%20Bornat.pdf

Caspersen , M., Larsen , K., Bennedsen, J. (2007) *Mental models and programming aptitude*. Innovation and Technology in computer science education (ITiCSE '07), Scotland, pp. 206-210. http://doi.acm.org/10.1145/1269900.1268845

Dehnadi, S., and Bornat, R. (2006) *The camel has two humps* (*working title*) https://www.cs.kent.ac.uk/dept_info/seminars/2005_06/paper1.pdf

Dehnadi, S. (2006) *Testing programming Aptitude*. 18th Annual Workshop of the Psychology of Programming Interest Group, Brighton, pp. 22-37. http://www.ppig.org/papers/18th-dehnadi.pdf

Du Boulay, B. (1989). *Some difficulties of learning to program*. In E. Soloway and J. C. Sphorer (eds), Studying the novice programmer, New Jersey: Lawrence Erlbaum. pp. 283-300.

Lister, R. (2011) *Concrete and Other Neo-Piagetian Forms of Reasoning in the Novice Programmer*. Thirteenth Australasian Computing Education Conference (ACE '11), pp. 9-18. http://crpit.com/confpapers/CRPITV114Lister.pdf

Lung, J., Aranda, J., Easterbrook, S., and Wilson, G. (2008) *On the difficulty of replicating human subjects studies in software engineering*. 30th international conference on Software engineering (ICSE '08), pp. 191-200. http://doi.acm.org/10.1145/1368088.1368115

Milner, W. (2008) *A Loop is a Compression*. 20th Annual Workshop of the Psychology of Programming Interest Group, Lancaster. http://www.ppig.org/papers/20th-milner.pdf

Pea, R. (1986) *Language-Independent Conceptual "Bugs" in Novice Programming*. Journal of Educational Computing Research, Vol. 2(1), pp. 25-36.

Teague, D. and Lister, R. (2014) *Longitudinal Think Aloud Study of a Novice Programmer*. Sixteenth Australasian Computing Education Conference (ACE '14), pp. 41-50. http://www.crpit.com/Vol148.html