Combining Unsupervised and Invigilated Assessment of Introductory Programming

Donna Teague
Queensland University of Technology (QUT)
Brisbane, Queensland
d.teague@qut.edu.au

Colin Fidge
Queensland University of Technology (QUT)
Brisbane, Queensland
c.fidge@qut.edu.au

Yue Xu
Queensland University of Technology (QUT)
Brisbane, Queensland
yue.xu@qut.edu.au

ABSTRACT

We compared student performance on large-scale take-home assignments and small-scale invigilated tests that require competency with exactly the same programming concepts. The purpose of the tests, which were carried out soon after the take home assignments were submitted, was to validate the students' assignments as individual work.

We found widespread discrepancies between the marks achieved by students between the two types of tasks. Many students were able to achieve a much higher grade on the take-home assignments than the invigilated tests. We conclude that these paired assessments are an effective way to quickly identify students who are still struggling with programming concepts that we might otherwise assume they understand, given their ability to complete similar, yet more complicated, tasks in their own time. We classify these students as not yet being at the neo-Piagetian stage of concrete operational reasoning.

Categories and Subject Descriptors

• Social and professional topics~CS1 • Social and professional topics~Information technology education • Social and professional topics~Student assessment

Keywords

Introductory programming; novice programmers; assessment; neo-Piagetian theory.

1. INTRODUCTION

With a global legacy of high failure rates in introductory programming units (Bennedsen & Caspersen, 2007; Corney, Teague, & Thomas, 2010; Watson & Li, 2014), there has been a concerted effort to address the issues associated with learning to program (Gomes & Mendes, 2010; Lang, McKay, & Lewis, 2007; Teague, 2011; Woszczynski, Haddad, & Zgambo, 2005).

But in the process, have expectations of students simply been lowered (Utting et al., 2013)? Academics are certainly under pressure to retain students (especially full fee-paying) and maintain minimum pass rates. As our cohort sizes continue to increase, teaching and computing resources are stretched even further.

ACSW '16 Multiconference, February 02-05, 2016, Canberra, Australia
 ISBN 978-1-4503-4042-7/16/02…$15.00
DOI: http://dx.doi.org/10.1145/2843043.2843064

The culture of collaboration among students is commendable from a first-year-experience perspective (Kift, 2008; Nelson, Kift, & Clarke, 2008). However, this culture makes assessment of individuals difficult in anything but the strictest invigilated examination. Students are collaboratively building or sharing solutions amongst themselves, finding solutions (in part or whole) online (e.g., blogs and other sites like www.stackoverflow.com and www.reddit.com), or paying ghost-writers to provide custom solutions (Besser & Cronau, 2015). According to a whitepaper on plagiarism and the web, by far the most popular student source for plagiarised assignments was social and content-sharing sites (Turnitin, 2011).

The digital age continually provides new and improved opportunities for plagiarism at university in all disciplines. It also makes plagiarism detection easier. However, the former seems to be always one step ahead of the latter, and possibly on a much grander scale. For example, ghost writers can be engaged to provide cheap and timely custom-written solutions for any type of assignment, for minimal cost. It is difficult to detect plagiarism via ghost-writing unless further investigation, for example oral examination, is undertaken. Although used successfully in the past (Teague & Corney, 2011), unfortunately oral examinations are not logistically possible with large cohorts of students. The plethora of wearable digital devices available today also makes conventional exams more difficult to invigilate (Adams, 2011).

Some institutions have adopted a "softly-softly" attitude to first year assessment by enforcing a low weighting on final exams, or prohibiting exams entirely. In any event, written exams may not be authentic assessment of programming skill either (Teague et al., 2012). The pedagogical dilemma in introductory programming units, therefore, is about being able to make a genuine assessment of each individual student's ability without requiring significantly more resources for teaching, marking and plagiarism detection. Unfortunately, a tempting compromise may well be to turn a blind eye to plagiarism and potentially have a larger number of graduates with fewer genuine skills.

Our introductory programming unit is probably representative of many others at other institutions. We have seen a growth in student numbers from approximately 200 in 2010 to 600 in 2015, with no change to staffing levels apart from an increased budget for tutors. The need for valid and authentic assessment is increasing, but the ability to provide it is becoming even more difficult. The hard part is providing interesting assessment items that test what we want to examine, but which leave plenty of scope for students who want to be creative and challenged. The hardest part is satisfying ourselves that the students do the work for themselves.

Research into verification of student work has included methods that combine peer assessment or written summaries with programming assignment submissions (Assiter, 2010; Hafner & Ellis, 2005). However, no literature was found that reported on the combination of multiple assessment items to validate the authorship of programming tasks.

In this paper, we report on an approach to assessment of programming skills which validates the work a student submits as an individual assignment with the use of a simple invigilated exercise testing the same programming skills. We have included samples of student work from the invigilated test and offer an analysis of the type of reasoning those students may have used in production of their solutions. Recent literature on novice programming makes use of the neo-Piagetian framework to describe student behaviour and reasoning (Lister, 2011; Teague, Corney, Ahadi, & Lister, 2013; Teague, Lister, & Ahadi, 2015), and our analysis uses this framework as summarised in the following section. This analysis of student work helps us to understand why discrepancies in marks are likely for many students between take-home assignments and invigilated tests.

2. NEO-PIAGETIAN FRAMEWORK

Lister first proposed that we may be able to describe novice programmer behaviour using neo-Piagetian theory (Lister, 2011). This theory describes the development of reasoning through sequential and cumulative stages of cognition. Teague et al. (2013; 2014; 2015) provide empirical evidence to support Lister's conjecture of a connection between neo-Piagetian theory and learning to program.

The first, and least mature, neo-Piagetian stage is sensorimotor. At this stage, the novice's understanding of the notional machine (du Boulay, O'Shea, & Monk, 1981; Sorva, 2013) is fragile and misconceptions about programming concepts are evident in their reasoning. Sensorimotor novice programmers are unable to trace code accurately and are similarly unable to form syntactically correct code.

When most of a novice's programming misconceptions have been resolved and they have a more solid understanding of the notional machine, they are likely reasoning at the pre-operational stage. However, at this stage, even though the novice has the skill to trace code with some accuracy, they are still unable to reason about that code's purpose. Preoperational novices have difficulty seeing the relationship between parts of the code and abstracting beyond the code itself. Other characteristics of preoperational reasoning include the inability to apply newly acquired knowledge to unfamiliar situations, and the tendency to apply what skill they have developed to inappropriate situations. Preoperational novice programmers also tend to develop code in a quasi-random style, using programming elements they recognise might be required, without being able to form a workable solution from those elements.

It is not until the next development stage, concrete operational, that a novice programmer can identify relationships between parts of the code and reason about the code's overall purpose. It is also at this stage that a novice programmer can more easily write code to perform familiar actions.

Rather than conceiving the neo-Piagetian framework as a one-way staircase model, we prefer to adopt an Overlapping Waves Model (Boom, 2004; Feldman, 2004; Siegler, 1996) to describe the transition between stages. According to this model, behavioural characteristics of an earlier neo-Piagetian stage dominate initially, but over time there is an increase in reasoning at the next more mature level and a decrease in the less mature. According to this model therefore, it is not unusual for a novice programmer to show evidence of simultaneously reasoning at two neo-Piagetian stages.

In analysing our own students' work, we use this neo-Piagetian framework to assess their likely reasoning ability with programming tasks.

3. METHOD

IT students at our university are initially introduced to programming in a unit which also exposes them to a variety of other technologies including pattern matching, databases and web interfaces. The assessment for this unit includes weekly on-line quizzes (25%), two major portfolios of programming tasks (50%), and an end of semester MCQ exam (25%).

This paper focuses on the assessment of programming language skills, using Python, with data collected over two teaching semesters.

3.1 Portfolio Assessment Pairs

Each portfolio is a pair of assessment items consisting of a Take-Home Task (THT), followed by a short In-Class Test (ICT). Here we describe the first portfolio pair, undertaken by students in Week 7 of the 13 week semester. (A similar portfolio pair is undertaken by students in Week 12 but with an emphasis on user interfaces and analysis of Web pages, rather than basic programming concepts.)

3.1.1 Take Home Task

The first THT required students to produce a drawing of some sort using Python's Turtle graphics module, in order to demonstrate basic programming skills (e.g., iteration, selection and list processing). The THT was specified in terms which gave each student the opportunity for creativity and individuality in design. The students were supplied with a list of data values on which they needed to perform arithmetic operations and then display information in a visual form using Turtle graphics.

The THT in one semester, for example, gave students several lists of data, each of which consisted of quadruples representing the style of a visual icon, the *x-y* co-ordinates for placement of the icon on a graph, and the size of the icon. For example, one such data set is shown in Figure 1.

```
data_set_09 = [['Icon 0', -265, -80, 50],
               ['Icon 2', 100, -146, 78],
               ['Icon 3', -50, 130, 69],
               ['Icon 1', 210, 100, 96],
               ['Icon 4', 200, 300, 45]]
```

Figure 1: Example THT Supplied Data Set

The key challenge was to be able to define reusable functions that could draw recognisable icons at any position and size. This was motivated by the familiar notion of a 'bubble chart', i.e., a chart which displays three-dimensional data on an *x-y* coordinate graph, with the bubble's size denoting the *z* value. In this case the 'bubbles' were not just circles, but were instead 'icons' (symbols or logos) intended to represent the data value of interest visually. Students were given the freedom to design and draw their own icons using Turtle graphics, but were required to place them on the supplied graph in bubble chart style exactly as stipulated by the data

set provided as a list. Their solution was required to work for any data set in an appropriate format.

Figure 2 and Figure 3 show output for two different students' submissions for this THT using one of the given data sets, in this case one that displays all five icons prominently at the same size. (Other data sets had the icons scattered about the graph at various locations and various sizes.) Given that these images were entirely drawn using Turtle graphics primitives, a large amount of Python code was required to produce results like these. Nonetheless, many students went to the trouble of making their icons realistic, even though they did not receive additional marks for doing so.

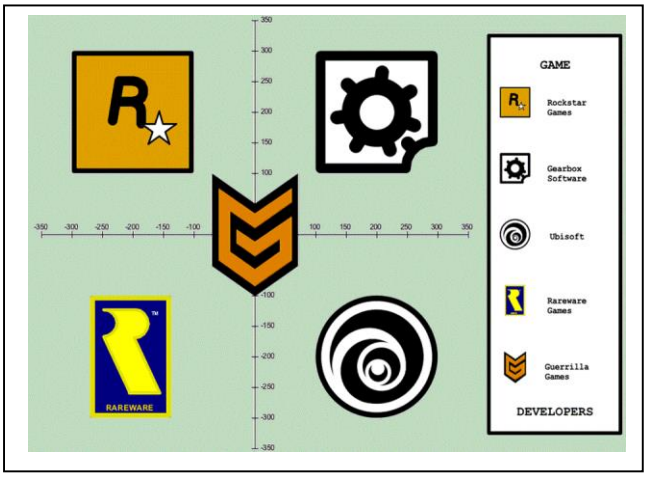

Figure 2: Example 1 - THT

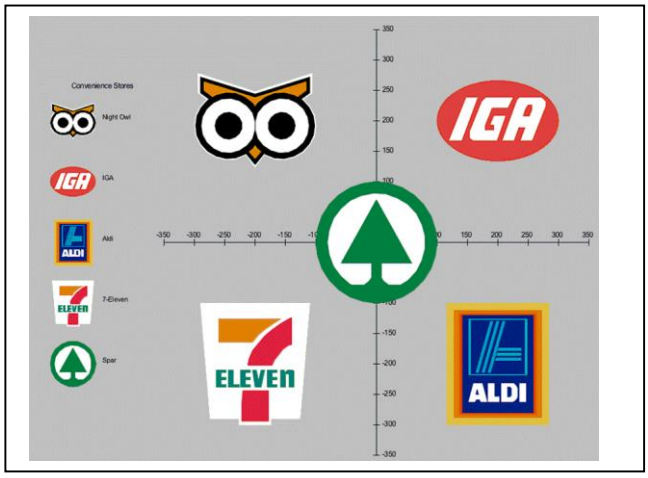

Figure 3: Example 2 - THT

3.1.2 In-Class Test

The first ICT, scheduled for just after the submission date of the THT, tested a subset of the very same concepts the THT required (selection, iteration, list processing and simple graphics) under invigilated exam conditions with a strict time limit. Each student was randomly assigned one task from a pool of many questions on a similar theme. This means that it was unlikely that two students seated next to each other in the ICT were assigned exactly the same question. It also reduced the chance of copying between the multiple test sessions that were required to process several hundred students via the university's small computer laboratories. The tests were managed using Blackboard and password controlled so that only students in the classroom at the time could take the test.

A template Python file was supplied for each ICT which included the task specifications and a stub method or two. Each ICT was "open book" (including the student's own THT solution) and students had access to the Internet. We stipulated only that they not confer or collaborate with others during the exam, and this included a ban on the use of mobile devices and of accessing social media sites and blogs, etc. Each ICT was invigilated by at least two staff members who enforced these restrictions.

An example ICT consisted of a supplied Python template file containing a predefined list of 169 pairs representing *x-y* coordinates, part of which is shown in Figure 4.

```
coords_list = [[-210, 210],
               [-175, 210],
               [-140, 210],  ... ]
```

Figure 4: Example ICT Supplied Data Set

The template file also contained the task description shown in Figure 5.

```
#-----Task Description-------------------------------------------------#
# You are required to write a program to draw dots using the data
# given in the variable coords_list below. The variable coords_list
# contains a list of (x, y) coordinate pairs. Each (x, y) pair
# specifies a position on the screen.
#
# For each (x, y) in coords_list, if x is smaller than y, draw a red
# dot at (x, y), otherwise draw nothing. All the dots have the same
# size, which is specified by the variable dot_size below.
#
# As a result, all dots are drawn in the top-left side of the screen
# and form a right angled triangle. All dots must be red.
```

Figure 5: Example ICT Task Description

Students were also supplied a screen shot of sample output for the task which was designed to reinforce their understanding of the specifications, and could be used by the student as visual confirmation that their code either was, or was not, working correctly when executed.

Figure 6 shows the expected output from this particular ICT, one of about 25 in the randomly-allocated pool.

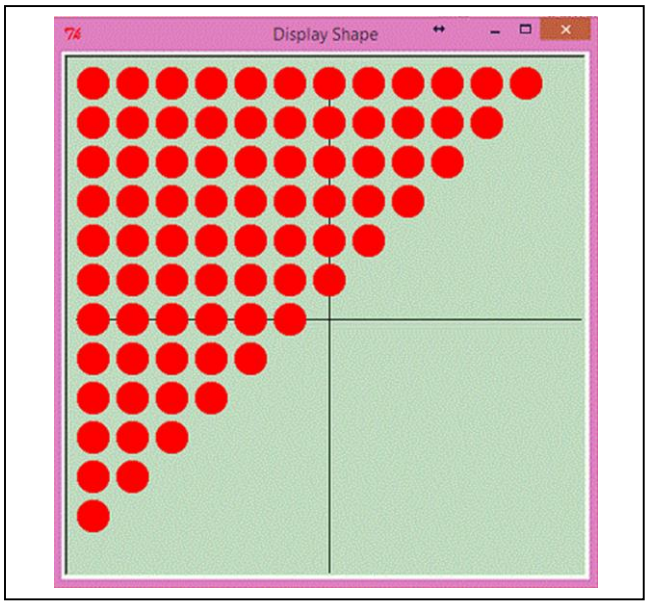

Figure 6: Example ICT Expected Output

A sample solution for this task is shown in Figure 7, omitting the "template" code supplied to the student to create the Turtle graphics drawing canvas.

```
color("red")
for pair in coords_list:
    if pair[0] < pair[1]:
        penup()
        goto(pair)
        pendown()
        dot(dot_size)
```

Figure 7: Example ICT Solution

The supplied data set was large enough so that a student could not feasibly (within the time limit) create the required output by brute force (i.e., 169 sequential hard-coded statements to draw a dot at a certain location).

There were many other variations of this ICT task created. For example, another task was to produce dots in a mirror image of that shown in Figure 6 (i.e., at locations where the *x* co-ordinate was larger than the y co-ordinate). Another was to draw dots on the diagonals of the grid (i.e, where the absolute values of the *x* and *y* co-ordinates were the same). The common theme with all the ICT tasks was to iterate through a given dataset, perform a simple mathematical calculation and conditionally produce visual output on a supplied grid. All of these programming tasks were also necessary to complete the THT successfully (see Section 3.1.1).

Each ICT was designed to be completed within around 20 minutes. The ICTs were run as timed Blackboard tests with a time limit of 45 minutes. Our design assumption was that any student who had successfully completed the THT by themselves would have little difficulty completing the ICT.

A simple grading system (a mark between 0 and 4) was used for both the THT and ICT assessment items. Marks for the THT were based on both functionality and presentation criteria, while for the ICT only functionality was assessed.

3.2 Complementary Assessment

Apart from the portfolio assessment pairs, throughout semester we also conducted ten weekly quizzes on Blackboard designed to test the technologies covered in lectures and workshops in the preceding week. Two of the quizzes were coding questions and the remainder were each a random selection of five questions from a large bank of questions on that week's topic which were automatically marked by the Blackboard system.

The main objectives for these weekly quizzes were to get "hands on keyboards" since most questions are answerable by testing some program code, and for consistent and timely opportunities for self-reflection. We stipulated that the quizzes were for individual assessment, and that by completing a quiz a student implicitly agreed that it represented their own work and that they were bound by the university's rules of academic integrity and code of conduct. However, since these quizzes were completed outside of class, we cannot be certain that every submission was 100% that student's own work.

The final item of assessment for the unit was an invigilated end of semester open book exam. The exam consisted of 25 multiple-choice questions answered on a mark-sense sheet. Each question had five alternatives, so by the law of averages, guessing was unlikely to result in an exam pass grade.

This large number of assessment items for the subject, fifteen individual parts in total, may sound like a lot of work, which risks being unpopular with the students. On the contrary, however, end-of-semester surveys consistently tell us not only that the students appreciate the weekly reminders to do some (small amount of) work on the subject, but that they wish other subjects used the same approach to assessment! We attribute this to the fact that none of the assessment items is worth a large amount, many were only 2% of their final grade, which avoided the stress associated with submitting high-value assignments.

4. RESULTS

In this section we focus on the results of implementing the first portfolio assessment pair described in the previous section, i.e., the first Take-Home Task immediately followed by the corresponding In-Class Test. In our quantitative analysis, we compare each student's mark awarded for the THT and ICT. We also include qualitative data in the form of observations by invigilators during the ICT and feedback from students via on-line university feedback mechanisms and student interviews.

4.1 Quantitative Analysis

Our quantitative analysis focuses on two consecutive semesters where the portfolio assessment pairs were used. In these semesters, each of the cohorts consisted predominantly of first year Bachelor of IT students.

The THT and ICT pairs were designed to test the same programming concepts with the THT submission date followed closely in the same week with the invigilated ICT. Rather than simply reporting on the outcome of each of the assessment items, there is thus more value in analysing the marks achieved for the *pair* of assessments for each student.

Figure 8 and Figure 9 show the differences between THT and ICT marks for two semesters for students who completed both the THT and ICT. Each histogram shows the frequency (measured as a percentage of students who completed both items of assessment) of a difference in grade for the pair of assessment items (THT minus ICT). Each item was marked out of 4, so the range of differences is between −4 and +4. The range of marks forms the *x* axis of the histograms.

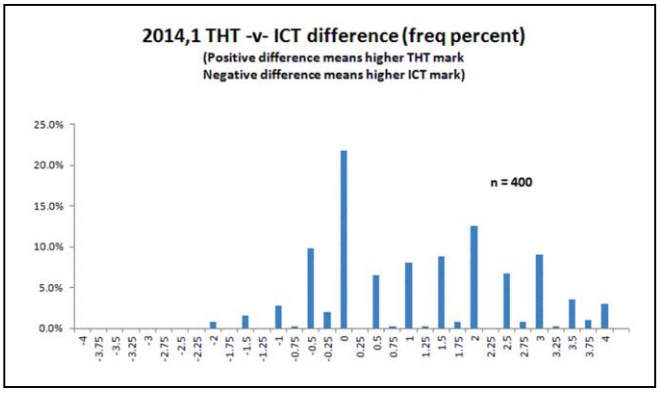

Figure 8: THT and ICT mark differences (Semester 1, 2014)

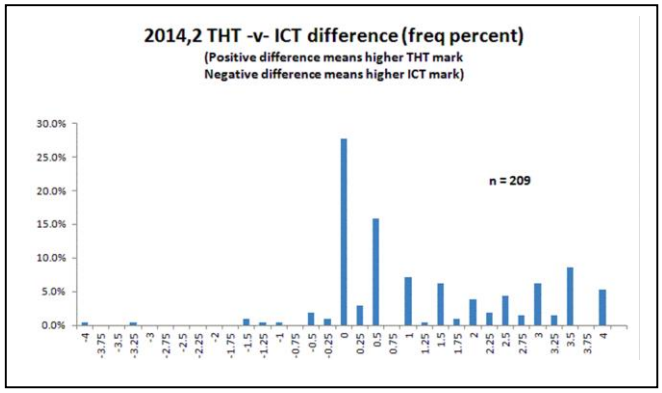

Figure 9: THT and ICT mark differences (Semester 2, 2014)

Any differences on the right hand side of the histograms (i.e., positive differences, shown to the right of 0 on the *x* axis) indicate a higher THT mark than ICT mark. Conversely, any differences on the left hand side of the histograms (i.e., negative differences, shown to the left of 0 on the *x* axis) indicate a higher ICT mark than THT mark.

4.1.1 High THT, Low ICT (positive difference)

Both sets of results show that the greatest percentage of students achieved the same mark for the THT as they did for the ICT (that is, a difference of 0). However, with a heavier distribution of differences on the right-hand side of the histograms, there is an obvious and consistent trend for a higher THT mark than ICT mark.

One explanation for these positive differences could be that with a great deal more time to complete the task, around three weeks as compared to 45 minutes in this case, students found take-home assessments easier. Another is that students received more help with the THT and simply did not have the same level of skills as their THT submissions reflect.

4.1.2 Low THT, High ICT (negative difference)

In both semesters there were few students who actually achieved a higher mark for the ICT than the THT. (This data would be shown in the histograms to the left of 0 on the *x* axis.)

We are not surprised by this result. Even though the ICT is a much simpler task, it is completed under time pressure in a laboratory environment. For the THT, students had a number of weeks to complete the task at their own pace, were encouraged to talk to their peers about their problem solving strategies, and had ample opportunity to seek help from teaching staff. They had time to completely abandon one, or a number of, strategies and start afresh if their initial attempts failed. This was not the case for the ICTs.

We believe that it is likely that the small number of students who received a much higher mark for their ICT were those students who failed to adhere to the specifications of the THT, possibly because they did not read or understand them in their entirety. The instructions for the THTs tend to be between 6 and 8 pages long, and a surprising percentage of students tackle the task without bothering to read them. The low mark for their THT would therefore not be due to a lack of the programming skills needed to successfully complete the assignment.

4.1.3 Mark Discrepancies

It is not completely unexpected that students achieve different grades for two different assessment items. We could however consider a difference in grades of more than 2 marks (out of 4) to be a major discrepancy in this instance. For example, THT-ICT marks of 4 and 1.5, or 0 and 2.5.

Looking at the positive differences in marks, using a difference of > 2 as the benchmark for mark "discrepancy", we can quantify the occurrence of discrepancies for the two semesters as shown in Figure 10.

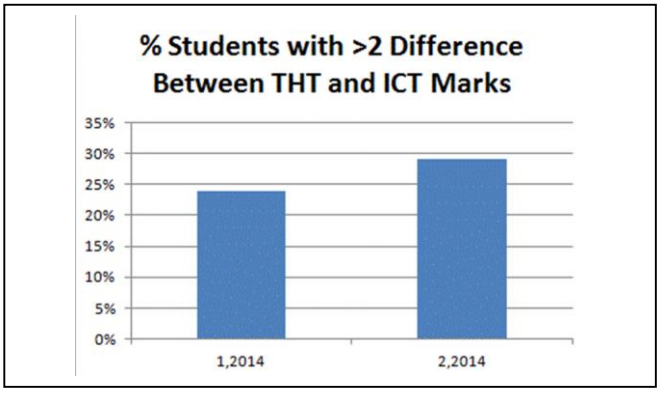

Figure 10: Mark Discrepancies

We further broke down the differences between THT and ICT marks for the two semesters studied for each performance quartile, based on final exam mark. (Performance quartiles were calculated resulting in the following range of exam marks: 1st: 0–5; 2nd 6–12; 3rd: 13–19; 4th: 20–25.)

Figure 11 includes students who completed both the THT and ICT as well as the final exam. Each bar in Figure 11 represents the

percentage of students in that quartile with a mark discrepancy for the THT and ICT.

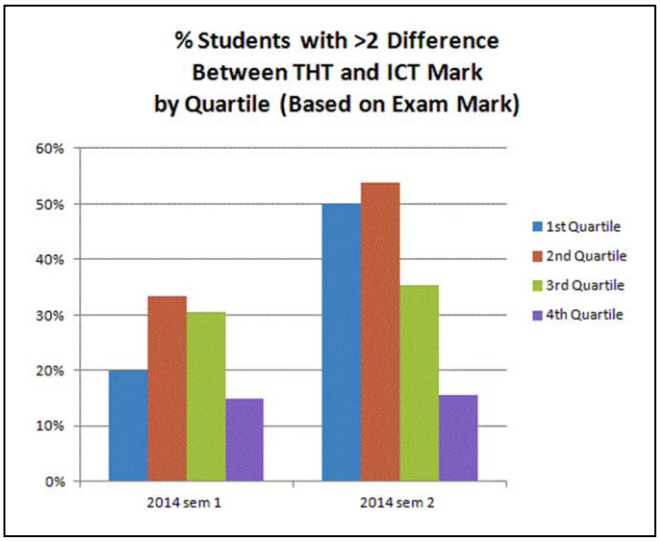

Figure 11: Mark Discrepancies per Quartile

It is worth noting the number of students from these cohorts who fall into each quartile (see Table 1). For example, there were 5 students in the first performance quartile for semester 1, 2014, and 20% of them (one student) had a mark discrepancy. Similarly, in semester 2, 2014 there were only 2 students in the first performance quartile, and one student had a mark discrepancy.

Semester	1st	2nd	3rd	4th	Students completing exam	Students *not* completing exam
2014/1	5	39	183	140	367	33
2014/2	2	26	85	84	197	12

Table 1: Number of Students in Each Performance Quartile

An alternative analysis might be to use performance quartiles based on overall grade. However, as we are seeing discrepancies between invigilated and take-home assessments, we consider it appropriate to use the final, written exam as a more likely representation of the students' ability than an overall grade which also includes their unsupervised weekly tests.

It is obvious from Figure 11 that students who perform well on the final exam are less likely to get a much higher mark for their THT than ICT. It is students from the lower three quartiles, in particular more consistently in the 2nd and 3rd quartile, who seem more competent with a take-home assessment item rather than one which enforces individual work.

Our plagiarism checks on student THT submissions do not reveal widespread major issues with code sharing for the first THT. (We believe that the free choice of what to draw discourages file copying because it would be so obvious to the markers.) We might therefore assume that students are either colluding with people other than fellow students (e.g., having someone else write or help them with their solutions) or that exam stress (e.g., time pressure, mental blocks etc.) has a significant detrimental effect on a great number of students.

However, there is another likely explanation. Students reasoning at the preoperational stage, according to neo-Piagetian theory, are those who can neither reason about code nor have any concept of the relationships between sections of code that combine to form the whole program. Preoperational novices have an adequate command of the language syntax and semantics and can mechanically trace code to determine its outcome. They may be capable of writing, for example, a `for` loop and a conditional statement. However, a preoperational novice would struggle to combine both of these constructs to achieve a conditional iteration of a list. It is not unreasonable to expect that such a student might obtain the necessary assistance from teaching staff who quite rightly see their THT attempt as well on the way to being correct. Even left to their own devices, given enough time to experiment using the quasi-random process of code writing that is characteristic of preoperational novice programmers, they may, with much effort, happen upon a working solution. (The part of the THT program that gives weaker students the most trouble is the function that iterates over the values in the list and calls other functions to draw the icons. The teaching staff receive many requests for help with this iterative function.) The issue is that preoperational novices are not yet at the stage where they can make that connection between the code parts easily themselves. They are therefore unlikely to be able to do so again in an unfamiliar, albeit logically identical, task under time pressure (e.g., the ICT).

What we believe we are seeing in our quantitative data is that the first three quartiles of our cohorts are dominated by students who are not yet at the concrete operational stage. That is, they are likely either operating at the sensorimotor or preoperational stage.

4.2 Qualitative Analysis

The remarkable thing about the ICT assessment was the ability of the invigilators to quickly identify which students were likely to complete the task successfully, and those who were struggling. It was simply a matter of looking over students' shoulders while invigilating the ICT. Thanks to the striking visual nature of the required output it is even possible to watch students making incremental progress towards a solution from across the room.

The nature of the ICT (i.e., to reproduce a Turtle image from given data) meant that students tended to execute their code frequently for immediate visual feedback. In fact, most students would run the supplied template without any changes before writing any code at all.

The able students very quickly produced correct, or near correct output. The majority of these students' time was used to check and recheck their output against the supplied image, to format or document their solution, and then, in some cases, simply to admire their handiwork for a period of time!

Struggling students were also identifiable, by the state of their visual output. For example, one student, S1, who was unable to produce anything visually (i.e., no more than the supplied grid), submitted code showing little understanding of either list processing or Turtle drawing, as shown in Figure 12.

```
pencolor("red")
for each in (coords_list):
    if [0] > [1]:
        print dot(dot_size)[0,1]
```

Figure 12: S1's Attempt - Code

S1 received full marks (4 out of 4) for the THT and only 1 out of 4 for the ICT. (S1's ICT code does not run, but a mark was awarded for at least recognising that a loop and nested conditional statement were involved in the solution.) This clearly falls within our definition of a "discrepancy" between the pair of assessments for this student.

Without knowing how S1 arrived at this solution, from the code we see evidence of both sensorimotor and preoperational stage reasoning according to neo-Piagetian theory.

S1's endeavour to use a nested conditional statement inside a loop gives us reason to believe he was struggling, as is characteristic of preoperational novices, to apply newly acquired knowledge (i.e., programming concepts used in the THT) in a new context. His code uses a mix of semantically ill-formed expressions. S1's submission also provides evidence that he has not yet mastered the language's syntax, which is characteristic of sensorimotor behaviour. The code contains malformed expressions and shows a fragile understanding of list processing, iteration and Turtle drawing. According to the Overlapping Waves Model mentioned in Section 2, it is not unusual for novice programmers to simultaneously reason at two neo-Piagetian stages. S1 may be a novice who is in the process of transitioning from sensorimotor to the preoperational stage.

Another student, S2, produced output that only partially matched the expected image as shown in Figure 13.

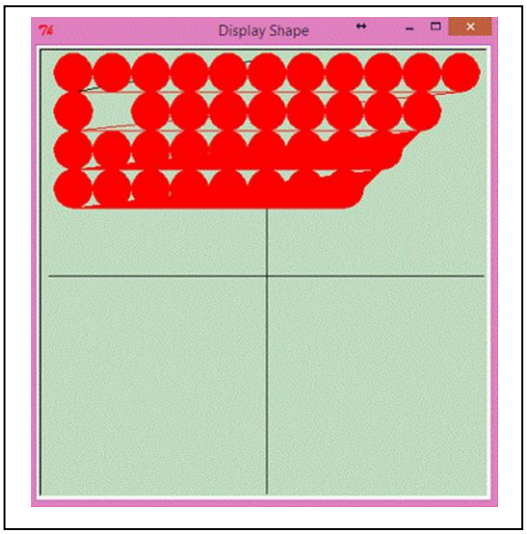

Figure 13: S2's Attempt - Output

Student S2's approach was total loop avoidance, with an attempt to draw each dot at hard-coded co-ordinates, a portion of which is shown in Figure 14.

```
pendown()
goto(-200,190)
begin_fill()
color('red')
circle(20)

goto(-160,190)
begin_fill()
color('red')
circle(20)

goto(-120,190)
begin_fill()
color('red')
circle(20)
...
```

Figure 14: S2's Attempt - Code

S2's marks for both the THT and ICT were 0.5 out of 4. Having not been able to successfully complete the THT, it is little wonder that he could not complete the ICT which tested the same skills. It is unclear from this submission whether S2 was familiar with loops or not. We can therefore make no definitive assessment as to which neo-Piagetian stage this student may have been reasoning. However, it is unlikely this student was reasoning at a concrete operational level if he was unable to recognise that list processing and a conditional loop were appropriate. His sequential code however is correct, and given enough time, he may well have been able to produce the correct output.

Another student, S3, failed to achieve much in terms of output (see Figure 15), yet his code reflected at least a high level understanding of the algorithm required to solve the problem.

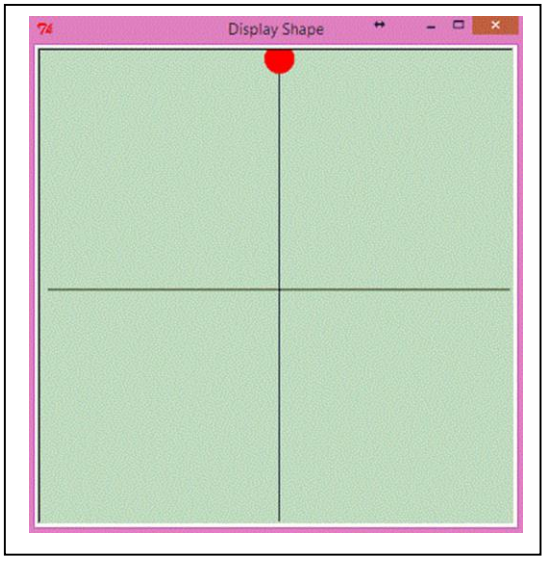

Figure 15: S3's Attempt - Output

However, S3 hard-coded the co-ordinates (incorrectly), failed to use the elements of the dataset at all, and made no attempt to reposition the Turtle-graphics cursor. S3's code is shown in Figure 16.

```
x = [0,0]
y = [0,1]
def dot_scale(x,y):
    for each in coords_list:
        if x<=y:
            dot(30)
            color('red')
print dot_scale(x,y)
```

Figure 16: S3's Attempt - Code

S3 received a mark of 3.5 out of 4 for the THT and 2 out of 4 for the ICT. This difference of 1.5 falls just under our definition of mark discrepancy benchmark of > 2.

We believe S3's solution is characteristic of early preoperational reasoning. There are parts of the solution that are syntactically correct (e.g., the construction of the loop; conditional statement; call to Turtle functions, etc.). Yet, as a whole, there is little of semantic value in this code. S3 struggled, as is typical of preoperational novices, to form logical relationships between necessary parts of the code (e.g., between the iterator and the conditional statement).

Student S4 also produced a drawing which only partially matched the expected output as shown in Figure 17.

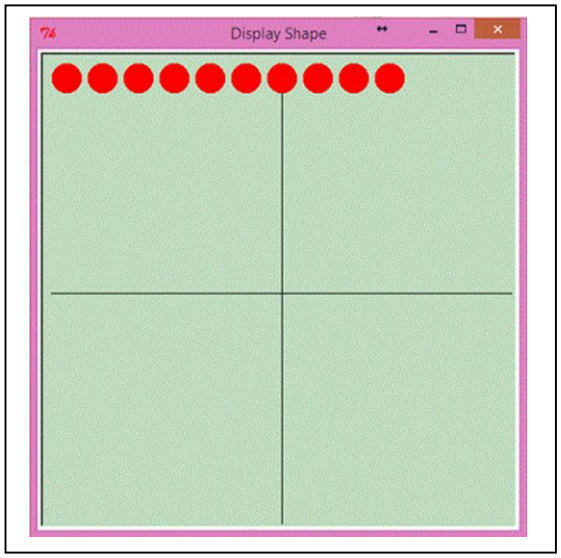

Figure 17: S4's Attempt - Output

S4 demonstrated a working understanding of functions and parameter passing, by encapsulating the code to draw a dot at a set of co-ordinates in a function. He even constructed a loop to iterate over the dataset. However, he struggled to write a suitable conditional statement. His attempt is shown in Figure 18.

```
def reddot(x,y):
    goto(x,y)
    setheading(0)
    pendown()
    dot(30, 'red')
    penup()

def drawreddot(coords_list):
    for each in coords_list:
        x = each[0]
        y = each[1]
        if each[0] == 'icon 0':
            reddot(x,y)
        elif each[1] == coords_list:
            reddot(x,y)
drawreddot(coords_list)

# Running out of time and panicing
penup()
reddot(-210, 210)
reddot(-175, 210)
…
```

Figure 18: S4's Attempt - Code

Rather than comparing the values of the *x* and *y* co-ordinates in the dataset, the conditional statement used by S4 in Figure 18 makes little sense. It appears to be heavily influenced by the THT implementation, by referring to "icon 0", a feature of the THT but not the ICT. The `elif` branch makes no sense at all (i.e., comparing the second element of the sublist with the entire dataset). The attempt, although still not working, is already far longer than the expected solution.

S4 received a mark of 2.75 out of 4 for the THT and only 0.5 out of 4 for the ICT, well within our definition of mark discrepancy.

In terms of neo-Piagetian classification, S4 would appear to be reasoning much like a preoperational novice. That is, his code is syntactically correct, but he is struggling to apply familiar constructs (i.e., those used in the THT) in an unfamiliar context. His frustration is evident with his comment that he was "… running out of time and panicing [sic]". S4's solution included (what looks like as an after-thought) a hard-coded drawing of the first 10 dots in the dataset.

The time taken to finish the ICT varied, with some students completing in less than ten minutes. Realistically, the majority of students take at least 30 minutes. Some of these students are obviously checking and rechecking their work after having a working solution within a shorter period of time. A few students produce working solutions just before the 45 minute deadline, but this is rare (and often accompanied by whoops of joy!). We might expect concrete operational students who take longer than 10 or 15 minutes to be those who have the ability to re-think their strategy or apply alternative constructs on a failed initial attempt.

4.3 Student Feedback

Polls were conducted to elicit feedback from students about the ICT. The responses from students were varied, with nearly 40% of them saying they had run out of time. There was not much agreement between students about the test's level of difficulty, with a fairly even split between it being too hard, too easy, and just the right level of difficulty (see Table 2).

Response	Average % Respondents
Ran out of time	38.5%
Too hard	11%
Too easy	15.5%
Just the right level of difficulty	15.5%

Table 2: Students' Perception of ICT

We also interviewed a number of volunteer students in an attempt to find out why someone who could complete the THT successfully would have trouble completing an invigilated test of the same set of programming skills (i.e., the ICT).

Unfortunately, but not surprisingly, the five students who responded to the call for volunteers had achieved high marks for both of the paired assessments. However, their responses may explain some of the discrepancies seen in marks across the cohort.

In response to the question:

> *Do you believe the ICT tested for the skills required to complete the THT*?

all five responded in the affirmative, although two of them indicated that they had not thought of the relationship between the THT and ICT assessments in that way before. This is surprising, as in our view the THT and ICT tasks are very closely linked (and this is stated several times in the lectures). That is, they both required iterating through a given dataset, performing a simple mathematical calculation and conditionally producing output in Turtle graphics on a supplied grid.

The interviewees were asked what skills each of the THT and ICT were testing for. The theme that dominated their responses for THT skills was Turtle drawing. For example:

> *How well you can use Turtle using the least moves possible to make a shape.*
>
> *Well we did a lot of [Turtle] stuff*
>
> *That was like reproduce a picture*
>
> *…your ability to try and be … like Photoshop.*

However, Turtle drawing (i.e., producing images) featured less in their responses about the skills being tested in the ICT. For example:

> *Ability to process data given.*
>
> *To reiterate the knowledge of iterating over a list.*
>
> *Lists … using the `for` loop with an array; conditionals; time based pressure.*

This is an interesting phenomenon, and counters the argument that the use of an ICT that closely follows a THT may simply be testing a student's memory. If a student was able to recognise that list processing and a conditional loop were required in the ICT, and was able to implement them, we believe that is a great outcome for a student seven weeks into an introductory programming unit. Preoperational novices are not likely to be able to make such a connection.

All five respondents believed that anyone who had completed the THT successfully should have been able to also complete the ICT. However, the underlying theme to these respondent's explanations for mark discrepancies was that of "time pressure" and "going blank" under exam conditions. One student said:

> *…[It] feels like the inability to ask the person next to you is the greatest challenge.*

5. CONCLUSIONS

Satisfying ourselves that students' unsupervised assessment submissions are their own work is nearing an impossibility. Plagiarism tools used in our subject like MoSS (Measure Of Software Similarity, https://theory.stanford.edu/~aiken/moss/) help us determine if students have likely shared code, but are not much use in alerting us to situations where a student has sought external help (e.g., ghost writing or assignment writing services).

More important than detecting plagiarism though, is our need as educators to ensure that students are developing programming skills and not just finding ways to pass assessment.

In response to our concerns about plagiarism, we developed the system of paired assessments described in this paper. This involved a take-home task to be completed over a number of weeks, followed by a short in-class test which assesses working knowledge of the same programming concepts required to complete the take-home task. For the first of these paired assessments, discussed herein, the in-class tests are designed to produce a simple visual output from a given dataset of values. The output can be compared to a supplied image of the expected result. We found this a very simple way of identifying students who are struggling with the programming concepts being tested. In fact, rather than having students submit their solutions and distributing them for marking, it may be viable to have the invigilator grade some of the students on the spot.

We conclude that, discounting any negative effect of exam stress and time constraints, invigilated tests seem to be a more accurate method of ongoing assessment of novice programmers' abilities. They can be used to check the likelihood of extensive collusion and/or collaboration resulting in a discrepancy between take-home assignments and in-class test marks. More importantly, the invigilated test following a take-home task is an effective way to identify students who are still struggling to use the programming concepts with which we might otherwise assume they are competent. We believe most of these students fall into the neo-Piagetian category of 'preoperational'. They have mastered the syntax of the programming language, can write isolated sections of semantically correct code using familiar constructs, but struggle to reuse those constructs in an inter-related manner and in an unfamiliar context. However, we cannot confidently categorise students' neo-Piagetian reasoning without observing them in the process of completing these types of tasks, which we hope to be able to do in future semesters.

We aim is to continue developing paired assessments which test for identical programming skills, and gathering quantitative and qualitative data for further analysis of student performance with invigilated and take-home assessment. However, the confounding issue is that students often complete tasks in ways we had not anticipated, avoiding use of the programming constructs and concepts for which we had hoped to test. Our challenge is to write specifications for assessment tasks that allow for creativity and challenge, yet confine implementation to a well-defined set of skills.

6. REFERENCES

[1] Adams, A. (2011). Student Assessment in the Ubiquitously Connected World. *Computers and Society, 41*(1), 6-17.

[2] Assiter, K. (2010). *Work In Progress - Programming Assignment Summary for Analysis, Qualitative Assessment and Continuous Improvement in CS1 - CS3.* Paper presented at the ASE/IEEE Frontiers in Education Conference, Washington, DC.

[3] Bennedsen, J., & Caspersen, M. E. (2007). Failure Rates in Introductory Programming. *Inroads - The SIGCSE Bulletin, 39*(2), 5.

[4] Besser, L., & Cronau, P. (Producer). (2015, April 20). Degrees of Deception. *Four Corners.* [Television Program] Retrieved from http://www.abc.net.au/4corners/stories/2015/04/20/4217741.htm

[5] Boom, J. (2004). Commentary on: Piaget's stages: the unfinished symphony of cognitive development. *New Ideas in Psychology, 22*, 239-247.

[6] Corney, M., Teague, D., & Thomas, R. (2010). *Engaging Students in Programming*. Paper presented at the 12th Australasian Computing Education Conference (ACE2010), Brisbane, Australia. http://elena.aut.ac.nz/homepages/ace2010/

[7] du Boulay, B., O'Shea, T., & Monk, J. (1981). The Black Box Inside the Glass Box: Presenting Computing Concepts to Novices. *International Journal of Man-Machine Studies, 14*(3), 237-249.

[8] Feldman, D. H. (2004). Piaget's stages: the unfinished symphony of cognitive development. *New Ideas in Psychology, 22*, 175-231.

[9] Gomes, A., & Mendes, A. J. (2010). *A Study on Student Performance in First Year CS Courses*. Paper presented at the ITiCSE 2010, Bilkent, Ankara, Turkey.

[10] Hafner, W., & Ellis, T. J. (2005). *Work in Progress - Authenticating Authorship of Student Work: Beyond Plagiarism Detection.* Paper presented at the ASEE/IEEE Frontiers in Education Conference, Indianapolis, IN.

[11] Kift, S. (2008). *The next, great first year challenge: Sustaining, coordinating and embedding coherent institution–wide approaches to enact the FYE as "everybody's business".* Paper presented at the 11th International Pacific Rim First Year in Higher Education Conference, An Apple for the Learner: Celebrating the First Year Experience, Hobart, Australia.

[12] Lang, C., McKay, J., & Lewis, S. (2007). Seven Factors that Influence ICT Student Achievement. *ACM SIGCSE Bulletin, Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education ITiCSE '07, 39*(3).

[13] Lister, R. (2011). *Concrete and Other Neo-Piagetian Forms of Reasoning in the Novice Programmer*. Paper presented at the 13th Australasian Computer Education Conference (ACE 2011), Perth, WA. http://crpit.com/confpapers/CRPITV114Lister.pdf

[14] Nelson, K., Kift, S., & Clarke, J. (2008). *Expectations and realities for first year students at an Australian University.* Paper presented at the 11th Pacific Rim First Year in Higher Education Conference, Hobart, Australia.

[15] Siegler, R. S. (1996). *Emerging Minds.* Oxford: Oxford University Press.

[16] Sorva, J. (2013). Notional Machines and Introductory Programming Education. *ACM Transactions on Computing Education, 13*(2). doi: DOI: http://dx.doi.org/10.1145/2483710.2483713

[17] Teague, D. (2011). *Pedagogy of Introductory Computer Programming: A People-FIrst Approach.* (Master of Information Technology (Research) Monograph), Queensland University of Technology, Brisbane.

[18] Teague, D., & Corney, M. (2011). *Is anybody there? Bootstrapping attendance with engagement*. Paper presented at the ASCILITE 2011 Changing Demands, Changing Directions, Hobart. http://eprints.qut.edu.au/48134/

[19] Teague, D., Corney, M., Ahadi, A., & Lister, R. (2013). *A Qualitative Think Aloud Study of the Early Neo-Piagetian Stages of Reasoning in Novice Programmers*. Paper presented at the 15th Australasian Computing Education Conference (ACE 2013), Adelaide, Australia. http://crpit.com/confpapers/CRPITV136Teague.pdf

[20] Teague, D., Corney, M., Fidge, C., Roggenkamp, M., Ahadi, A., & Lister, R. (2012). *Using Neo-Piagetian Theory, Formative In-Class Tests and Think Alouds to Better Understand Student Thinking: A Preliminary Report on Computer Programming.* Paper presented at the Australasian Association for Engineering Education Conference (AAEE 2012), Melbourne. http://eprints.qut.edu.au/55828/

[21] Teague, D., & Lister, R. (2014). *Longitudinal Think Aloud Study of a Novice Programmer*. Paper presented at the 16th Australasian Computing Education Conference (ACE 2014), Auckland, New Zealand. http://crpit.com/confpapers/CRPITV148Teague.pdf

[22] Teague, D., Lister, R., & Ahadi, A. (2015). *Mired in the Web: Vignettes from Charlotte and other Novice Programmers*. Paper presented at the 17th Australasian Computing Education Conference (ACE 2015), Sydney, Australia. http://crpit.com/confpapers/CRPITV160Teague.pdf

[23] Turnitin. (2011). Plagiarism and the Web: Myths and Realities. Retrieved September 10, 2015, from https://turnitin.com/static/resources/documentation/turnitin/company/Turnitin_Whitepaper_Plagiarism_Web.pdf

[24] Utting, I., Bouvier, D., Caspersen, M., Elliott Tew, A., Frye, R., Kolikant, Y. B.-D., . . . Wilusz, T. (2013). *A Fresh Look at Novice Programmers' Performance and Their Teachers' Expectations.* Paper presented at the ITiCSE-WGR'13, Canterbury, England UK.

[25] Watson, C., & Li, F. W. B. (2014). *Failure Rates in Introductory Programming Revisited*. Paper presented at the Innovation and Technology in Computer Science Education (ITiCSE 2014), Uppsala, Sweden. http://dx.doi.org/10.1145/2591708.2591749

[26] Woszczynski, A. B., Haddad, H. M., & Zgambo, A. F. (2005). *Towards a Model of Student Success in Programming Courses*. Paper presented at the 43rd ACM Southeast Conference, Kennesaw, GA, USA.