

Assessing the Success of an Introductory Programming Course

Author

Ford, Marilyn, Venema, Sven

Published

2010

Journal Title

Journal of Information Technology Education

Rights statement

© 2010 Ford et al; licensee Informing Science Institute. This article is distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by-nc/3.0/us/>), which permits sharing and adapting, provided the original work is properly cited.

Downloaded from

<http://hdl.handle.net/10072/37498>

Link to published version

<http://www.jite.org/documents/Vol9/JITEv9p133-145Ford810.pdf>

Griffith Research Online

<https://research-repository.griffith.edu.au>

Assessing the Success of an Introductory Programming Course

Marilyn Ford and Sven Venema
*Griffith University, School of Information and Communication
Technology, Meadowbrook, Queensland, Australia*

m.ford@griffith.edu.au; s.venema@griffith.edu.au

Executive Summary

With universities having difficulty attracting students to study information technology (IT), the scores needed for entry into IT degrees have dropped markedly. IT schools are thus having to cope by adjusting their introductory courses to ensure that students will still learn what is expected but without negatively impacting on pass rates. This paper considers short objective tests, designed by other researchers, to examine whether students who have passed an introductory course have achieved an understanding of fundamental concepts in programming. The Dehnadi test, which was originally designed to be taken before a programming course to predict who would be successful, proved to be useful in showing that many students who had passed an introductory programming course had little or no understanding of fundamental concepts. The test was useful if the number of correct responses was considered. Implications of students passing an introductory course but being unable to respond correctly on a multiple choice test of the most fundamental concepts of programming are considered.

Keywords: programming students, assessment, grade inflation, teaching programming

Introduction

In 2001, the McCracken group reported on a multi-national, multi-institutional assessment of the programming skills of first year computer science students (McCracken et al., 2001). There were 10 educators in the working group, from 8 institutions, from the USA, Australia, Israel, Wales, England, and Poland. This group had come together because of a concern that their students were failing to learn the fundamental skills of programming. Their aim was to investigate the skills of students who had completed one or two courses in computer science. Their investigation of 216 students from four universities showed that most students performed more poorly than anticipated. On a test that was scored out of 110, the average score was 22.89. Many students wrote programs that did not compile and, according to the group, a significant number of the “solutions” showed that the students were “clueless.” This was the case even though the group be-

Material published as part of this publication, either on-line or in print, is copyrighted by the Informing Science Institute. Permission to make digital or paper copy of part or all of these works for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage AND that copies 1) bear this notice in full and 2) give the full citation on the first page. It is permissible to abstract these works so long as credit is given. To copy in all other cases or to republish or to post on a server or to redistribute to lists requires specific permission and payment of a fee. Contact 0HPublisher@InformingScience.org to request redistribution permission.

lieved the problems were considered to be such that “students in any type of Computer Science programme should be able to solve them” (McCracken et al., 2001, p 128).

The McCracken group focused on problem solving in programming and suggested a 5-step process that students should learn: 1. abstract the problem from its description, 2. generate sub-

problems, 3. transform sub-problems into sub-solutions, 4. re-compose the sub-solutions into a working program, and 5. evaluate and iterate. The assessment they used reflected their emphasis on this 5-step process. It involved writing arithmetic evaluation programs that a student who had finished one year of study would be expected to complete in about one and a half hours. Another group of concerned educators suggested that the students' difficulties may be more basic than a difficulty with problem solving (Lister et al., 2004). They suggested that students might lack skills that are a precursor to problem-solving. The Lister group consisted of 12 educators, from 12 institutions, from Australia, England, the USA, Finland, New Zealand, Sweden, Denmark, and Wales. They believed that students might have only a "fragile grasp" of computer skills, perhaps being able to articulate items of knowledge, but when asked to apply the knowledge might show they only "sort of know" something. They suggested that students might have trouble even *reading* simple code.

The Lister group assessed 556 students from 6 institutions, most of whom had completed, or nearly completed, a semester of studying programming. They used 12 multiple choice questions written in Java or C++, depending on the institution, that required students to demonstrate understanding of existing code. The example in (1) is the question with the highest success rate, 73%.

(1) Consider the following code fragment:

```
int[] x = {0, 1, 2, 3};
int temp;
int i = 0;
int j = x.length-1;

while (i < j)
{
    temp = x[i];
    x[i] = x[j];
    x[j] = 2*temp;
    i++;
    j--;
}
```

After this code is executed, array "x" contains the values:

- a) {3, 2, 2, 0} *correct answer*
- b) {0, 1, 2, 3}
- c) {3, 2, 1, 0}
- d) {0, 2, 4, 6}
- e) {6, 4, 2, 0}

As the authors expected, the distribution of performance was better than that shown by the McCracken group; after all, it was a multiple choice assessment and required understanding only small pieces of existing code. The authors did conclude, however, that many of the students showed a fragile ability to systematically analyze a short piece of code. The items differed in success rates, ranging from 35% to 73%. The easiest questions were those that asked for the outcome of the code, while the more difficult questions tended to be those that required students to choose the piece of code missing in a certain place from a program. It was found that the top quartile of students scored from 10 – 12, the next quartile from 8 – 9, the third from 5 – 7 and the lowest quartile from 0 – 4. The authors suggest that the students in the lowest quartile probably have difficulties more fundamental than an inability to problem solve. They also suggest that if students in the top quartile have difficulty writing code it is probably because they have a weakness in problem solving. The group suggested that readers should decide what score they would

consider acceptable, but they did say that they believed a student with a score of 5 would not have a level of skill needed for a subsequent course. They suggested that if institutions consider a failure rate of 25% as the upper limit acceptable, then some students would be passing who should not.

The dismay felt by the McCracken and Lister groups is also felt strongly today by educators faced with the task of teaching programming. Ma, Ferguson, Roper, and Wood (2007) have suggested that students' poor performance in programming is most likely the main reason computer science courses have such high attrition rates, around 30-50% according to Denning & McGettrick (2005; also see De Raadt et al., 2005). Not only are departments that teach programming faced with high attrition rates and pressure to limit failure rates, but since the dot com bust in early 2000 there has been a marked decrease in students enrolling in computer science programs (De Raadt, Watson, & Toleman, 2004; Denning & McGettrick, 2005). To survive, many departments must be admitting students that would not have been accepted in the past. Certainly in Australia, scores needed for admittance to information technology degrees have decreased over the last few years.

One response to the problem of many students having difficulty with programming is to develop new teaching approaches (Bruce, Buckingham, Hynd, McMahon, Roggenkamp, & Stoodley, 2004; Garner, 2009; Ma, Ferguson, Roper, Ross, & Wood, 2008). However, no panacea has yet been found. Another approach is to see if predictors for success can be found (De Raadt et al., 2005; McGettrick, Boyle, Ibbett, Lloyd, Lovegrove, & Mander, 2005; Simon et al., 2006; Tolhurst et al., 2006). Finding predictors may help decrease failure rates, though, to remain viable, some departments would need to take students who did not fare well on a predictor test.

Dehnadi (2006) suggested that one of the early hurdles to learning programming is an understanding of assignment. An example of a Java statement with single assignment is given in (2).

```
(2) int a = 10;
    int b = 20;
```

```
    a = b;
```

Given (2), $a = b$ so now $a = 20$, and b would remain with the value 20. A more complicated example is given in (3).

```
(3) int a = 5;
    int b = 3;
    int c = 7;
```

```
    a = c;
```

```
    b = a;
```

```
    c = b;
```

Here, a student must understand that sequencing matters; $a = c$, so now $a = 7$; $b = a$, so now $b = 7$; finally, $c = b$, so now $c = 7$. Thus, at the conclusion of the statements, all variables have the value of 7. Bornat, Dehnadi, and Simon (2008) tested the idea that a multiple choice test of assignment problems, consisting of three like (2) and nine like (3), could be given before a student has started a programming course to predict success in the course. Bornat et al. at first thought they had evidence that students who responded with a consistent strategy (even if it was an inappropriate strategy) obtained better results in their end of semester classes than those who responded inconsistently. However, as students in other institutions were tested, they felt that the predictive effect of the test failed to live up to its early promise. Unfortunately, interpretation of

the data is hampered by several factors. One problem is that the reported data was simple frequency data, such as the number who passed or failed the course and the number who were classified as mainly correct, mainly consistent but not correct, and not consistent on the predictive test. This meant that only weak statistical tests could be used. Moreover, although the authors claim to have done analyses of variance, they report an R^2 value as a result, which is a correlational statistic and which could not be used on their simple frequency data. It is certainly true, however, that the data from their experiment 3 with participants who had not done programming before did not give any support for a predictive effect of consistency on the test.

Possibly the most interesting aspect of the paper, although it is not commented upon by the authors, is the stark differences in results for different institutions. Thus, if one does the calculations, it can be seen that at one institution (Middlesex University, Experiment 2) only 11.27% (27/71) of the students were classified as giving correct answers on the predictive test, while for another institution (University of York) 86.54% (90/104) were classified as giving correct responses. Clearly, the institutions are admitting quite different populations to their programming courses. It would, of course, be interesting to know how these two populations of students would fare after the same university programming course, but that is unknown.

While consistency *per se* on the Dehnadi test may not be predictive of performance in a programming course, the test does provide a simple means of checking whether students understand assignment and sequencing. Most University of York students are entering with that understanding, while most Middlesex students are entering without such understanding. Given how fundamental assignment and sequencing are to programming, educators would hope that by the end of a semester long course, students would do well on the Dehnadi test. Thus, while the written test of the McCracken group or even the reading test of the Lister group might be too hard, at least the skills tested by the more fundamental test of Dehnadi should be mastered.

Ma et al. (2007) tested 90 volunteer students on the Bornat et al. (2008) test at the end of an introductory programming course. They acknowledged that these 90 students came from a possible pool of 124 students and that it seemed that weaker students did not volunteer. Ma et al. (2007) found that 63% of the students gave consistently correct responses at the end of the course (also see Ma et al., 2008). Ma et al. (2007) expressed concern that approximately one-third of the students did not have an understanding of assignment and sequencing after an introductory programming course. In fact, the figure of one third is probably an underestimate given that weaker students did not volunteer to do the test.

Ma et al. (2007) tested differences in student performance in the programming course, depending on whether students gave correct, consistent but not correct, or inconsistent answers on the Dehnadi test. Unfortunately, Ma et al. (2007) fell into the trap of making multiple pairwise comparisons of their groups. This is not legitimate unless special tests are done because the tests are not independent and they can thus result in possibly spurious significant or conflicting results (Gliner and Morgan, 2009). However, it seemed that the Correct group performed better in the programming course than the other two groups. Ma et al. obtained conflicting results regarding whether the Consistent/Incorrect group performed better or worse than the Inconsistent group, depending on whether they were looking at the final exam or in-course assessment. Lung, Aranda, Easterbrook, and Wilson (2008) carried out a study very similar to that of Ma et al. (2007). Unfortunately, they fell into the same trap as Ma et al. (2007) of making multiple pairwise non-independent comparisons, leading them to make the following inconsistent claims within their paper: in a programming course, the Correct group scored significantly better than the Consistent/Incorrect group, the Correct group did not score significantly differently from the Inconsistent group, the Consistent/Incorrect group did not score significantly differently from the Inconsistent group. Again, though, the Correct group appears to show better performance in programming classes compared to at least some other students.

It seems that being correct on the Dehnadi test, at least after having taken a programming course, may be related to course performance. As Ma et al. (2007) note, it is disturbing that a large percentage of their students did not understand such fundamentals as assignment and sequencing after an introductory programming course. Given the simplicity of the Dehnadi test and the fact that it assesses skills that are fundamental to programming (considerably more fundamental than what is tested by the McCracken or Lister group tests), it seems that it could be used to assess how well students are progressing in programming and could be used to compare students at different stages and at different institutions. That is, instead of thinking of it as a possibly predictive test, it could be used as a test of some of the most basic skills needed to be learned in a programming course. All institutions of course hold their own tests. No doubt, these differ in difficulty. But the Dehnadi test could be a useful barometer of how students are doing at the most basic skill level. If students are passing introductory programming courses, yet cannot give correct answers on the Dehnadi test, then probably something is wrong.

The Griffith University Programming Course

As with many universities, Griffith University in Australia has had a decrease in applications for entry into their IT degrees over the last few years and the score required for admittance has thus dropped markedly. It became clear, very quickly, that the compulsory programming courses, Programming 1 (P1) and Programming 2 (P2), needed to be modified. In 2003, P1 was changed in a number of ways. First, as well as using a traditional PC based programming environment, students now use LEGO[®] MINDSTORMS[®] NXT robots to solve physical real-world problems. It was hoped that the use of physical robots would encourage engagement in the course. Second, a smaller subset of Java is now introduced to the students. The subset includes everything except user defined classes and objects and can be compiled using a special compiler used in the course. Third, assessment was modified. The traditional 40% final examination was eliminated. Instead, there are now ten weekly 5% laboratory exercises, five in-lecture 5% multiple choice quizzes, and a final 25% project intended to tie together all the concepts covered within the course.

Topics covered in P1 are: 1. programming concepts, 2. problem solving and design, 3. implementation in a subset of the Java programming language, and 4. use of programming tools and environments. Lectures and labs both cover the topic of assignment and sequencing. Examples are discussed and worked through in lectures and students also apply the concepts to problems in the labs. An example of a program worked through in the lectures (in week 4 of the 13 week course) is given in (4).

```
(4)    import console;

        print("How much is the loan for ($) ? ");
        double amount = readDouble();
        print("What is the annual interest rate (%) ? ");
        double annualRate = readDouble();
        print("how many months ? ");
        int months = readInt();
        double rate = annualRate / (12 * 100);
        double repayment = amount * rate
            / (1.0 - pow((1.0 + rate), -months));
        double totalInterest = months * repayment - amount;
        println("Monthly repayment = $" + repayment);
        println("Total interest = $" + totalInterest);
```

It should be clear that students who pass the course would be expected to understand assignment and sequencing. As well as doing P1, in their first semester students also take three other courses,

one of which introduces computer architecture and Boolean logic. This course gives students further exposure to assignment and sequencing in programming.

The new approach to P1 has seen a dramatic reduction in the failure rate. Thus, in the last offering of P1 the failure rate was 18.2%, whereas before the changes failure rates were in the 30 – 40% range. Attendance rates for laboratory sessions are above 90%. Attendance rates at lectures vary, with very high attendances in the weeks with the assessed quizzes. Student satisfaction, measured in the form of a course evaluation survey is generally very high, usually in the range of 5-7 on a scale where 7 is the best value.

The fact that failure rates have decreased, even in the face of students entering with lower entry scores than in the past, could suggest that all is going very well. However, the fear is that assessment strategies or decreased expectations could be influencing results. This is why an independent, objectively scored test covering the most fundamental topic of assignment and sequencing could be valuable. Also, given that there has been a lot of interest in the Dehnadi test, we wanted to explore results obtained with it in more detail and with more sophisticated statistical testing than previously done. We decided to give the test to students at the beginning of P2. These students had done P1 and passed. Apart from giving the students the Dehnadi test, we wanted to give them more difficult questions to see how they fared and to see whether the results on these tests related to how they had performed in P1. We chose a problem given by Reges (2008) as possibly being a question that discriminates well between students who have, or do not have, an aptitude for programming. The question was translated into Java as the students had been exposed to this language. We also chose four problems of a type considered by Ma et al. (2007) to be harder than items on the Dehnadi test. We will, however, focus on the results from the Dehnadi test.

Methodology

Participants

Students enrolled in P2 at two of the three Griffith university campuses where IT is taught were asked to do the tests in one of their classes. They were told that it was not compulsory, but most students stayed to do the tests. There were 111 students consisting of 98 males and 13 females and all had passed P1. Of these students, 103 were from one campus, and the remaining 8 from another, small campus. Students had approximately 25 minutes to do the tests.

Procedure

The students were given a consent sheet and an information sheet. It was explained that participation was voluntary and all students were given a test booklet and an answer sheet. The tests were in the order presented here. The first author, who does not teach or convene P1 at any campus, but who convenes the IT degree at the small campus, administered the tests at the large participating campus. The second author, who teaches and convenes P1 at the small campus, administered the tests at the small campus.

The Tests

The Dehnadi test

The Dehnadi test consisted of three questions similar to (2) and nine similar to (3).

For the questions like (2), students were given 10 choices plus a space to give another answer if they thought it was appropriate. For the questions like (3), students were given between 16 and

19 choices plus a space for a different answer if they thought it was required. The correct answer was always one of the choices given. The choices were designed by Dehnadi to distinguish between different strategies for answering the questions. The twelve questions, the choices given for each question, and the strategy each choice indicated can be found on Dehnadi's site:

www.cs.mdx.ac.uk/research/PhDArea/saeed/

The Reges question

The Reges question, given in Java, was:

If b is a Boolean variable, then the statement
 $b = (b == \text{false})$; has what effect?

The students were given 5 choices plus a space for another answer if they thought it was required. The correct answer, *It always changes the value of b* , was one of the choices given.

The Ma et al. (2007) reference assignment test

The Ma et al. reference test consisted of four questions, given here:

```
Person a, b;
a = new Person ("Jack");
b = a;
```

```
Person a, b;
a = new Person ("Jack");
b = new Person ("Tom");
b = a;
```

```
Person a, b;
a = new Person ("Jack");
b = new Person ("Tom");
b = a;
a = b;
```

```
Person a, b, c;
a = new Person ("Jack");
b = new Person ("Tom");
c = new Person ("Jim");
b = a;
a = c;
c = b;
```

Each question was followed by diagrams representing different states of memory. Students were asked to choose the diagram that would represent the memory state after the statements were executed. Nine to ten choices were given for each question and a space was provided for another answer if the student thought it was appropriate.

Results

The Dehnadi test

Students were classified as responding correctly, consistently but incorrectly, or inconsistently. The criteria for classifying students are given here:

Correct: 10/12 correct

Consistent/Incorrect: 10/12 answered with the same inappropriate strategy

Inconsistent: less than 10 questions answered with the same strategy

The criterion for consistency is in line with Dehnadi’s suggestion and was also used by Ma et al. (2007). The criterion for being classified as correct was also used by Ma et al. A less stringent criterion was used by Bornat et al. (2008), that is, 9/12. The more stringent classification was chosen because, like the participants in Ma et al.’s study, the participants here were doing the test after having completed an introductory programming course.

The percentage of students classified in each category is given in Table 1.

Table 1. Percentage of students classified as Correct, Consistent/Incorrect, or Inconsistent

Correct	Consistent/Incorrect	Inconsistent
49.55%	23.42%	27.03%
n = 55	n = 26	n = 30

The average score out of 100 for each of these groups for their completed P1 course is given in Table 2.

Table 2. Average score on the completed P1 course for each group

Correct	Consistent/Incorrect	Inconsistent
80.34	72.02	67.63
n = 55	n = 26	n = 30

One way of determining whether the results of the three groups differ is to do an analysis of variance. This test takes in the P1 score for each student and indicates whether the groups differ. The analysis of variance yielded $F = 13.40$, $p = 0.000006$. The result is thus highly significant. To determine where the difference lies, a Tukey HSD (Honestly Significant Difference) test was performed. The Tukey HSD test allows comparisons between each mean, but corrects for the fact that multiple comparisons are being made. The Tukey HSD test showed that the P1 performance of the Correct group was significantly better than that of the Consistent/Incorrect group, $p = 0.007102$ and also better than that of the Inconsistent group, $p = 0.000008$. The test also showed that there was no significant difference between the Consistent/Incorrect group and the Inconsistent group, $p = 0.320201$.

Notice that by simply grouping students into one of three categories a lot of information is lost because the number of correct responses for each student on the Dehnadi test is not used. All that is used is their classification as Correct, Consistent/Incorrect, and Inconsistent, and their P1 score. Also, the first three items of the Dehnadi test are simpler than the remaining nine, but information about scores on the two types of questions is lost. To overcome this problem, a multiple regression was carried out using the number of correct responses for each student on both the three easy questions and the nine harder questions. Other information known about the participants was also used: namely, age, gender, whether they were a domestic or international student, whether they said they spoke a language better than English, and whether they said they spoke a language as well as English. The multiple regression was used to find out which of these factors related to P1 performance as shown by the marks received for the course. The results showed that there were two significant factors. First, the greater the number of correct responses on the 9 harder Dehnadi

questions, the better a student's P1 score, $p = 0.00000053$. Second, students who reported that they speak another language better than they speak English scored worse in P1 than other students, $p = 0.0195$. (A multiple regression using the number of consistent responses per student rather than the number of correct responses per student yielded no significant effects).

As a way of investigating the data further, the average performance on the three easy Dehnadi questions and the nine harder questions as a function of students' P1 performance was calculated, with students grouped according to the decile of their P1 score. The average scores are given in Tables 3 and 4.

Table 3. Average score on the nine harder Dehnadi questions as a function of P1 performance (indicated by decile groupings)

50s	60s	70s	80s	90s
3.00	3.94	3.69	5.65	8.29
n = 19	n = 18	n = 29	n = 31	n = 14

Table 4. Average score on the three easy Dehnadi questions as a function of P1 performance (indicated by decile groupings)

50s	60s	70s	80s	90s
2.00	1.94	2.00	2.61	3.00
n = 19	n = 18	n = 29	n = 31	n = 14

The Reges Question

Only 13.5% (15/111) students chose the correct answer for the Reges question. Given that there was a 20% chance of giving the correct answer by choosing one of the given responses, this question turned out not to be very useful.

The Ma et al. (2007) Reference Assignment Test

Poor performance on these questions meant that the test was not very useful. The percentage of students scoring 0/4 was 66.67%. However, it was found that there were four students who scored 4/4, and these four students were also among the small percentage of students who answered the Reges question correctly. The P1 scores for these students were 97.5, 96.0, 85.0, and 69.5.

Discussion

Consistency, Correctness and the Dehnadi test

It is apparent that, at least for performance after completing a programming course, it is the number of correct results on the Dehnadi test that matters, not consistency *per se*. It is possible that conflicting results regarding differences between students classified as Consistent/Inaccurate and Inconsistent in the past literature (see Bornat et al, 2008; Lung et al. 2008; and Ma et al. 2007) may have been caused by such classification being ill-advised

It also seems that the most discriminatory part of the Dehnadi test may be the nine harder questions. It may be that the Reges and the Ma et al. (2007) questions are discriminatory, with only very good students able to do them all successfully.

More studies are probably needed to determine whether the number of correct responses on the Dehnadi test, taken before an introductory programming course, can predict success in such a course. Part of the problem is that it is sometimes not clear whether students in an introductory programming course at university have actually done any programming before. Also, it is apparent from the Bornat et al. (2008) studies that students at different institutions perform quite differently on the Dehnadi test before they have taken their introductory university course. If all of these students could be given the same university course and compared, then a relationship to the number correct responses on the Dehnadi test may well be found.

For the moment, it does appear that the number of correct responses, especially on the nine harder Dehnadi items, is related to how well a student has performed on a preceding programming course. It is highly likely that students who do well on the test before having done programming will do very well in programming. We suggest that if many students do poorly on the Dehnadi test after a semester long introductory course on programming at university, then there is probably cause for alarm. The test has helped our school see clearly that there is a problem and it has done so in a way that could be shown easily to others. The real value of the Dehnadi test may be in this post-course setting, as an objective test of the most basic of programming concepts. It has certainly led to suggested changes in P1 at Griffith University. On the other hand, at least for participants similar to those in the present study, the Reges and the Ma et al. (2007) questions may not be very useful.

The Need for Change

Ma et al. (2007) were disappointed that only 63% of their students were classified as being in the Correct group in their study. Their participants included students who failed the programming course. The fact that only 49.55% of the students in our study were classified as being in the Correct group, even though they had all passed P1, is very disturbing. It is also disturbing that the other students, on average, received reasonable marks out of 100 in P1, with the average for the Consistent/Incorrect group being 72.02 and the average for the Inconsistent group being 67.63.

While the decrease in failure rate in P1 since the modifications were implemented might seem a positive feature to some, there is reason for concern. The Dehnadi test consists of examples that should be easy for anyone who has completed and passed a 13 week introduction to programming course at university.

It could be thought that perhaps the results on the tests were poor because the students did not try. This may be true of some students. However, most students seemed to be working on the problems and to use the allotted time. We believe that the results need to be taken seriously and that changes need to be made to the introductory course. We suggest that other educators might also wish to reflect on whether their pass rates for introductory programming courses are reflecting something positive or not. There seem to be two primary factors to consider. First, what improvements can be made to how programming skills and concepts are taught? Second, are marks given too easily in the programming course?

Changes to Teaching

Now that the weakness of students in understanding assignment and sequencing has been highlighted, the P1 lecture notes will be revised, with examples modified and added at strategic points in the lecture notes to draw attention to the concepts. Perhaps this will also help students who reported that they speak another language better than they speak English, which of course means they speak English worse than some other language. Griffith University is also making an English course compulsory in the first semester of study for students who do not have an overall 7 on

the IELTS (International English Language Testing System) test. Perhaps this will help these students.

However, more major changes are probably required. After seeing the results of the study, the lecturer and convenor of P1 at the large campus, who was not involved with the study, wrote an additional feature for the compiler used by the students. This new feature allows students to step through their program one line at a time to check what the program is doing. Using color as an emphasis, it visually registers the introduction and removal of variables from the current context and any change of value. It thus gives the students an opportunity to “look inside” the machine as it is operating and to observe the impact of specific statements on the values of variables as they come into scope and leave it. The students will be able to adjust the speed at which the program steps through the program and will be able to pause it. The visualizer will be introduced into P1, hopefully in semester 1, 2010. It will be used in specific laboratory exercises and students will be asked to use it so that they can better understand the behavior of their own programs.

The use of a visualizer is in accord with a suggestion by Ben-Ari (2001) that program visualization aids could create a good learning environment. Ma et al. (2007) obtained some evidence that using program visualization in the way suggested here helped one of their introductory programming classes. We are hopeful that the new program visualizer will help our students. However, more changes are probably required. A concern is that students are receiving high marks too easily and thus not even realizing that they have developed only weak, or even wrong, concepts about programming.

Changes to Assessment

Given that the average marks for P1 were quite reasonable, yet performance on the Dehhadi test was so bad, one needs to consider whether grade inflation has occurred and what might have caused it. Let’s consider the three types of assessment used in P1.

There were five in-class 5% multiple choice quizzes, thus amounting to 25% of the marks. These need to be examined to determine whether the incorrect answers, that is, the distractors, are so obviously wrong that anyone could guess the correct answer. The questions should be fair, but hard to guess. Race (2001) suggests that lecturers check whether there are any distractors that no students ever choose and suggests that such distractors are probably serving no useful purpose. Race also suggests testing multiple choice questions with other students, to check whether they are choosing the correct answer for the right reason. Students can obtain marks by guessing on multiple choice questions, but the aim should be to keep this to a minimum. It is certainly possible that many students are scoring several extra marks by guessing due to poor distractors. Any such questions and distractors will need to be modified.

There were ten weekly 5% laboratory exercises, thus amounting to 50% of the marks. Students worked on the exercises and could, during the lab, keep submitting them until they were correct. They could also get help from the instructor during the lab if, on submitting their answer, they found that it was wrong. Such a process is possibly quite good. However, it could be better not to mark that effort, but the next week to give a similar problem but with no help permitted. This way, students would be encouraged to learn from their mistakes and not simply rely on someone being there to help them fix mistakes.

There was a final 25% project. Unfortunately, there is no assurance that the student who submits the project has actually done the project. Such a 25% project is probably acceptable, if the remaining assessment is really testing the students’ skills and understanding. However, this may not be the situation. The justification for eliminating the traditional 40% exam needs to be re-examined and the question of whether the course needs to re-introduce such an exam needs to be considered seriously.

Moving on to a Second Programming Course

If students pass an introductory programming course without a firm grasp of introductory programming concepts and skills, they are likely to struggle with subsequent courses that attempt to build on these introductory concepts. We have in fact seen that students struggle with P2. Hopefully, the efforts we are making will lead to greater learning in P1 and then greater success in P2.

Conclusion

Today, with universities finding it harder to attract IT students and with the scores needed to obtain entry into IT degrees decreasing over the last few years, many IT schools are trying different strategies to cope. The Dehnadi test may be a way for some IT schools to check whether their efforts are being successful or whether further adjustments need to be made. Certainly for the IT school considered here, the test was useful in revealing potential weaknesses in course content and assessment procedures, leading to suggestions for change to help address these weaknesses. Finally, we believe that schools using the test should focus on the number of answers correct and not consistency *per se*.

References

- Ben-Ari, M. (2001). Program visualisation in theory and practice. *InformatikInformatique*, 2, 8-11.
- Bornat, R., Dehnadi, S., & Simon. (2008). Mental models, consistency and programming aptitude. *Proceedings 10th Australasian Computing Education Conference (ACE2008)*, Wollongong, Australia, 53-62.
- Bruce, C., Buckingham, L., Hynd, J., McMahon, C., Roggenkamp, M., & Stoodley, I. (2004). Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university. *Journal of Information Technology Education*, 3, 143-160. Retrieved from <http://www.jite.org/documents/Vol3/v3p143-160-121.pdf>
- De Raadt, M., Hamilton, M., Lister, R., Tutty, J., Baker, B., Box, I., et al. (2005). Approaches to learning in computer programming students and their effect on success. *Proceedings of the Annual International Conference of the Higher Education Research and Development Society of Australasia (HERDSA)*, Sydney, Australia, 407-414.
- De Raadt, M., Watson, R., & Toleman, M. (2004). Introductory programming: What's happening today and will there be any students to teach tomorrow? *Sixth Australasian Computing Education Conference*, Dunedin, New Zealand.
- Dehnadi, S. (2006). Testing programming aptitude. *Proceedings of the 18th Annual Workshop of the Psychology of Programming Interest Group*, Brighton, England, 22-37.
- Denning, P. J., & McGettrick, A. (2005). Recentering computer science. *Communications of the ACM*, 48 (11), 15-19.
- Garner, S. (2009). A quantitative study of a software tool that supports a part-complete solution method on learning outcomes. *Journal of Information Technology Education*, 8, 285-310. Retrieved from <http://www.jite.org/documents/Vol8/JITEv8p285-310Garner398.pdf>
- Gliner, J. A., & Morgan, G. A. (2009). *Research methods in applied settings: An integrated approach to design and analysis*. New Jersey: Lawrence Erlbaum Associates, New Jersey.
- Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., et al. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, 36(4), 119-150.
- Lung, J., Aranda, J., Easterbrook, S., & Wilson, G. (2008). On the difficulty of replicating human subjects studies in software engineering. *Proceedings of the 30th International Conference on Software Engineering*, New York, 191-200.

- Ma, L., Ferguson, J., Roper, M., Ross, I., & Wood, M. (2008). Using cognitive conflict and visualisation to improve mental models held by novice programmers. *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, Portland, 342-346.
- Ma, L., Ferguson, J., Roper, M., & Wood, M. (2007). Investigating the viability of mental models held by novice programmers. *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, Covington, 499-503.
- McCracken, M., Kolikant, Y., Almstrum, V., Laxer, C., Diaz, D., Thomas, L., et al. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, 33(4), 125-140.
- McGettrick, A., Boyle, R., Ibbett, R., Lloyd, J., Lovegrove, L., & Mander, K. (2005). Grand challenges in computing education – A summary. *The Computer Journal*, 48(1), 42-48.
- Race, P. (2001). *The lecturer's toolkit: A resource for developing learning, teaching, & assessment*. London: RoutledgeFalmer.
- Reges, S. (2008). The Mystery of "b := (b = false)". *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, Portland, 21-25.
- Simon, Fincher, S., Robins, A., Baker, B., Box, I., Cutts, Q., et al. (2006). Predictors of success in a first programming course. *Proceedings of the 8th Australasian Computing Education Conference*, Hobart, 189-196.
- Tolhurst, D., Baker, B., Hamer, J., Box, I., Lister, R., Cutts, Q., et al. (2006). Do map drawing styles of novice programmers predict success in programming? A multi-national, multi-institutional study. *Proceedings of the 8th Australian Computing Education Conference*, Hobart, 213-222.

Biographies



Dr. Marilyn Ford is an associate professor in the School of Information and Communication Technology at Griffith University. She has publications in the areas of linguistics, sentence perception, sentence production, reasoning, and education.



Dr Sven Venema is a lecturer in the School of Information and Communication Technology at Griffith University. He has publications in the field of computer science and specializes in the design and analysis of computer algorithms.