Automated Assessment in CS1

Des Traynor, Susan Bergin, J. Paul Gibson

National University of Ireland, Maynooth
Co. Kildare, Ireland,
Email: {dtraynor,sbergin,pgibson}@cs.nuim.ie

Abstract

A system has been developed for providing automated assessment in CS1. During the academic year 2004-2005 this system was evaluated empirically by examining a sample group of students using both the traditional assessment methods and also the automated techniques, four times during the year. A significant correlation was found between the performance in both tests, however the correlation was only strong for students who performed well during the year.

To further this study, students were interviewed and asked their opinion on the generated questions. The students offered reasons for the variation in their performance and provided an insight into where the discrepancies lie. We discovered that weaker students were employing rote-learning and using it to score marks in the class exams.

As this survey was conducted on paper, a large amount of student roughwork ("doodles") was collected, the analysis of this roughwork is also discussed.

Keywords: Assessment, Program Comprehension, First Year Programming

1 Introduction

This paper discusses a study assessing the feasibility of automated assessment performed during the academic year 2004-2005 at the NUI Maynooth. Previously a system had been developed for generating code based comprehension questions(Traynor & Gibson 2005). This study aimed to assess the performance of the tool in a rigorous manner, examining students at regular intervals throughout the academic year and finally interviewing them at the end of the study. The goal of this research is to determine the extent to which assessment in CS1 can be automated whilst maintaining a consistent standard.

The failings of assessment in computer science have been highlighted by group studies such as the McCracken group (McCracken, Almstrum, Diaz, Guzdial, Hagan, Kolikant, Laxer, Thomas, Utting & Wilusz 2001). The conclusion of this study was that the majority of students in computer science courses do not possess the ability to perform basic programming tasks. If the assessment procedure in Computer Science was more accurate, this report would not have had such a strong impact. Accurate assessment is the key to learning(Sprinthall, Sprinthall & Oja 1998); this multi-national failure of assessment must be alleviated before computer science education can move forward.

1.1 Motivation

Computer science education sees many successful and unsuccessful attempts to replace educators with automated counterparts. It is not the goal of this tool to replace educators. Teaching has always been a social process, relying heavily on the dynamics of the teacher-student relationship. This tool aims to provide a "resource light" option to teachers for assessing the ability of large class sizes. The intention is to alleviate the difficulties of assessment detailed by Carter*et al.*(Carter, Ala-Mutka, Fuller, Dick, English, Fone & Sheard 2003) who noted that 74% of computer science educators use practicals for assessment, and almost all grading is done manually.

There are currently several systems available for automated assessment of different aspects of student programming ranging from input-output testing (RoboProf(Daly 1999), Ceilidh CourseMarker(Higgins, Symeonidis & Tsintsifas 2002)), graphical interface testing (English 2004) and code 'style' testing(Kirsti Ala-Mutka & Jrvinen 2004).
Ala-Mutka discussed a great number of these tools in a survey detailing the strengths and weaknesses of automated assessment (Ala-Mutka 2005). All of the afmorementioned tools require the lecturer to supply questions however, so whilst they are considered automated assessment, a more accurate name would be automated grading.

The tool under evaluation in this study produces questions that require students to trace the execution of a generated piece of code, and select what they believe the output of the program to be from a generated list of options. The generated questions are very similar in style to those proposed by Lister and Leaney (2003). The system receives inputs that determine the length, difficulty, and topic for each question and will generate a piece of code which is presented to the students. The students must then trace the execution path of the program and select its behaviour from a set of of possible answers. Further details of the system can be found in Traynor & Gibson (2003).

1.2 Overview of the paper

The next section of this paper discusses the methodology of the study, and how we constructed a fair evaluation of the tool. The third section shows results achieved from the study, and discusses both student performance and correlations with doodle data[1] gathered during the study. The fourth section discusses

[1]Doodle data is a phrase coined by Lister et al. for described the written thoughts of students whilst solving problems on paper

the information gathered from the interviews, and the final section details our findings and future work of this project.

2 Methodology

The assessment data was collected approximately every six weeks. This interval was chosen as it mirrored the frequency of the class exams. The question sets were distributed in small booklets to the students. To participate in the survey students signed consent forms. Students participating in the interviews were also required to sign a separate consent form.

Typically the surveys were performed the week after an exam. Each question set consisted of eight generated questions and one "blankpaper" question where the student was required to write code to solve a problem. The surveys were given to the students on paper, and students were asked to write only on the paper provided, or to hand up any additional notes taken.

The test was originally designed to be delivered through a web browser, but it was decided that this medium would provide less information, and also the code based questions could easily be compiled therefore rendering the results meaningless.

2.1 Analysis of Sample

The first year CS100 (aka CS1) module in Maynooth provides an "Introduction to programming" for students from a variety of disciplines. In 2004-2005 there were 118 students undertaking the module. 85% of the students enrolled are taking either a 4 year course in "Computer Science & Software Engineering", or are taking a degree in Science with Computer Science being one of their elective modules. There are however students from Arts and Biotechnology also enrolled for the module.

The study began 2 weeks into term, with 58 students($\doteq$ 50% of class) volunteering to participate. After the students had taken 2 class exams, a t-test was performed along with an equality of variance test(Levine test) which verified that there was no significant difference in the mean performance of the sample group and the remainder of the class. In addition, assumptions of normality were tested(Kolmogorov-Smirnov) which showed the sample to be normally distributed. All statistical analysis was performed using the SPSS sample analysis software.

2.2 Interviews

At the completion of the final question set, students were asked to participate in interviews regarding their experiences of assessment in CS100 and how it compares with the question sets that had been used in the surveys. 19 students volunteered to participate in the interviews which were performed in the final week of the second semester.

2.3 Other Information

The programming module in XYZ is taught through Problem Based Learning(Kelly, Bergin, Mooney, Ghent, Gaughran & Dunne 2004) and the module teaches Java with an "Objects Last" pedagogy. Students have three lecture hours per week, two hours of PBL workshops and two hours of laboratory/applied work. The lab examinations referred to in this paper were 1 hour tests during which the students had to write programs to demonstrate their knowledge of topics recently covered during lectures.

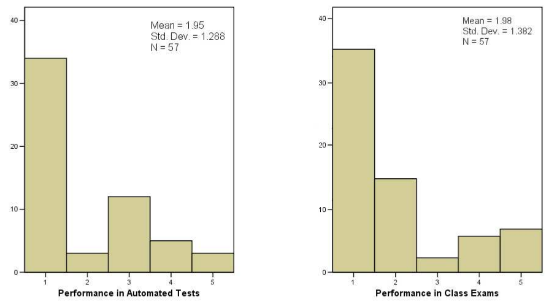


Figure 1: Histograms of Students' Performance in Generated Tests and Lab Exams

3 Analysis of Data

The total data gathered consisted of four sets of questions, rough work paper and 19 interview recordings. The data analysis consisted of four stages, firstly checking for correlation between student performance in the generated tests and the class exams, secondly examining the generated questions that caused most difficulty, thirdly analyzing "doodle" data, and finally, analyzing the difference between the students opinions of paper based tests and their actual performance.

3.1 Test Correlations

A Pearson correlation of .63 ($p<0.05$) was found when analyzing students' mean performance in the generated test compared against their mean performance in the corresponding class exams (i.e. 63% of the variation in programming scores for the class could be accounted for using generated tests). The correlation was found to be higher (.74, $p<0.01$) for students in the top quartile of the class, however students who performed poorly in class exams tended to perform extremely badly in the generated tests. This relation was further examined during the interviews with the students.

Figure 1 shows a histogram comparing the distribution of grades in both tests. In the figure, 1-5 represents the different grades 1 being a score from 70 to 100 and 5 being a score of 15 or below. It can be seen in the histogram that 14 students score a grade 2 (57-69) in the class exam, whilst only 3 students score likewise in the generated tests. It is also clear that more students fail badly in the class exam(grade 5=20% or lower) than in the generated exam. We believe this is due to the nature of the questions. The class exams were not solely multiple choice questions, and as a result scores of 0-20 were possible when students possessed no knowledge of the concepts being examined. However, across 32 MCQs, even random guessing would usually yield a score of 20-25% since we were not using negative marking(Eisner 1998).

The initial findings from the survey performed this year would indicate that automated assessment is possible for students (and in particular high performing students), however the variation in student performance is an area that requires further study. The interview data discussed in section four offers some potential explanations.

3.2 Question Analysis

The tool used to generate each question also generates feasible distractors to appeal to the weaker students. The majority of the questions generated had one reasonable distractor that weak students chose,

```
String s = "Life, if you will is
full of funny surprises";
int x  = s.indexOf('f');
switch (x) {
case -1:
  System.out.println("Well Hi!");
   break;
case 2:
   System.out.println("Pleased to meet you");
   x=21;
case 7:
   System.out.println("Good day to you sir");
   break;
case 21:
   System.out.println("Who on earth are you?");
   break;
default:
System.out.println("Bonjour");
 x--;
}
```

Q: What is the output of this program?
a) Who on earth are you
b) Pleased to meet you followed by Who on earth are you?
c) Bonjour
d) Pleased to meet you, followed by Good day to you sir
e) Pleased to meet you, followed by Good day to you sir, followed by Who on earth are you?

Figure 2: A Generated Question regarding the Switch Statement

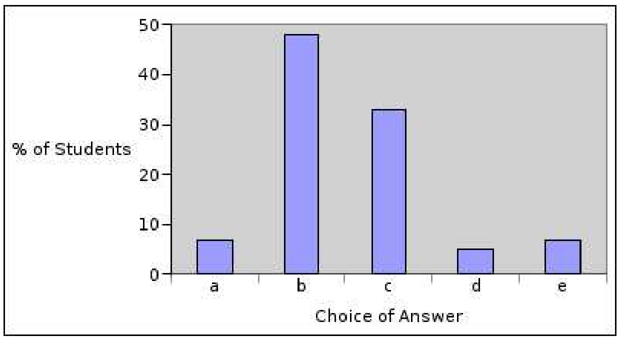


Figure 3: Answers chosen for the Question in Fig.2

and several highly infeasible distractors which combined would account for 10% of the answers. The methodology for developing feasible distractors is still in an early stage of development, and as such its performance is not yet highly effective.

Some of the generated distractors however were capable of misleading almost the entire class. Figure 2 demonstrates a generated question to test students' knowledge of the switch statement, assuming the students had a prior knowledge of Strings. The question tests whether the students understand the internal workings of a Java switch statement and in particular the `break` keyword. As Figure 3 shows, the majority of students went for the generated distractor 'b'. Only 4 students in 59 answered the question correctly.

Most questions exhibited the desirable traits discussed by Lister et al.(Lister, Adams, Fitzgerald, Fone, Hamer, Lindholm, McCartney, Moström, Sanders, Seppälä, Simon & Thomas 2004), in that the correct answer should be must popular amongst the good students, and the lower groups show no bias toward any particular answer. An example of a well formed question on for-loops can be seen in Figure 4 and its corresponding analysis of the answer trends

```
int sum = 0;
for(int i=0;i<10;i+=3)
{
 i--;
 sum+=i;
}
System.out.println("Sum = "  + sum);
```

Q: What is the output of this program?
a) sum=30
b) sum=15
c) sum=12
d) sum=17
e) sum=14

Figure 4: Generated Question on For-loops

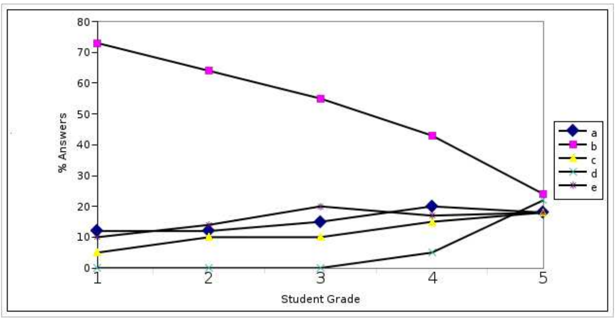


Figure 5: Analysis of Answer Trends for Question in Fig.4

for this question can be see in Figure 5.

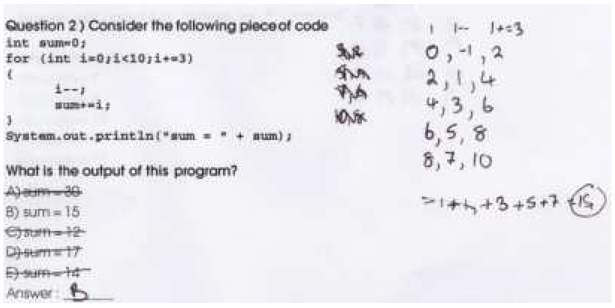
3.3 Doodle Data

When performing this survey we wanted to ensure that we collected as much data as possible, this included any notes that students may take whilst solving the problems. Analysis of student "doodling" was performed by Lister et al(Lister et al. 2004) on the multi-national study of students' ability to trace the execution of programs. The doodle data that we gathered was largely similar in categories to the twelve found in the multi-national study. The most frequent occurring we found to be *synchronized trace* and *rule out*(see Figure 6) and *position trace* (see Figure 7). On the rough work sheets provided many of the doodles appear to be a combination of a *computation* and *number trace.* Figure 8 shows a good example of this.

The significance of the doodles themselves was also analyzed, yielding some interesting results. The survey contained approximately 980 questions that were attempted (we define attempted as being some marks made on the question sheet, or an answer given). Due to time restrictions we analysed only one question from each set of questions for doodles, and collected doodle data from 113 questions. Of these, 92 were answered correctly(81%). The distribution of doodles across the class is not arbitrary, typically the students who perform best doodle the most. Table 1 shows the amount of doodles performed by each section of the class. It is not clear whether this is a cause or effect however. It could be argued that only good students doodle, or that students perform well because of their doodling.This was revisited during the interviews, where students were asked if they believed they could score equally well without doodling. It was also noted that the majority(69%) of the doodle data collected was from females.

Grade of Student	Number of Doodles
1	68
2	11
3	14
4	12
5	8

Table 1: Doodling is habit of good students



Question 2) Consider the following piece of code

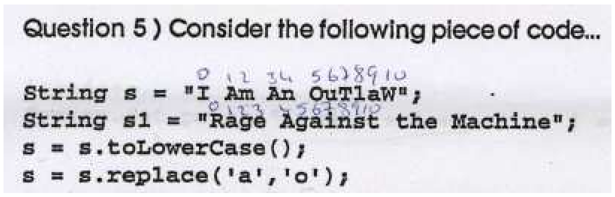
```
int sum=0;
for (int i=0;i<10;i+=3)
{
      i--;
      sum+=i;
}
System.out.println("sum = " + sum);
```

What is the output of this program?

~~A) sum = 30~~
B) sum = 15
~~C) sum = 12~~
~~D) sum = 17~~
~~E) sum = 14~~
Answer : B

i i-- i+=3
0 , -1 , 2
2 , 1 , 4
4 , 3 , 6
6 , 5 , 8
8 , 7 , 10
=1+4+3+5+7 = 15

Figure 6: A synchronized trace and “rule out”

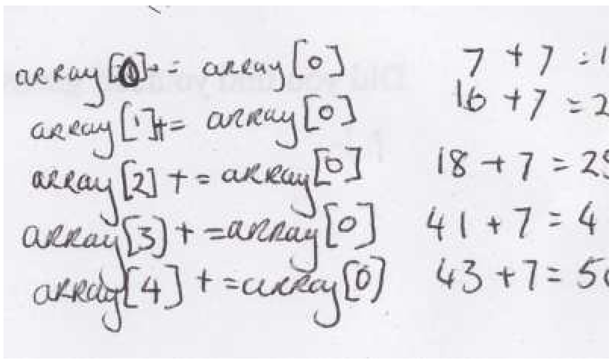


Question 5) Consider the following piece of code...

```
String s = "I Am An OuTlaW";
String s1 = "Rage Against the Machine";
s = s.toLowerCase();
s = s.replace('a','o');
```

0 1 2 3 4 5 6 7 8 9 10
0 1 2 3 4 5 6 7 8 9 10

Figure 7: Position Trace for indexing



array[0] += array[0]
array[1] += array[0]
array[2] += array[0]
array[3] += array[0]
array[4] += array[0]
7 + 7 = 1
16 + 7 = 2
18 + 7 = 2
41 + 7 = 4
43 + 7 = 5

Figure 8: Computation and Number Trace

4 Interviews

Nineteen students were interviewed during the final week of term. This subgroup was not perfectly representative of our sample group, it had a slight bias toward students with high grades (8 students from the top grade were included). We believe this to be unavoidable as we were requesting students to volunteer for interviews on programming. It seems inevitable that good students are most willing to discuss their experiences of programming in CS1.

The average duration of an interview was 25 minutes. The interviews were carried out by postgraduates with whom the student had no prior interaction (either as lab demonstrator or invigilator) to ensure they would speak freely about their experiences in CS100. This paper will only discuss the questions asked that are relevant to the work described here, namely the student's opinions of the question style, the student's performance in the survey questions compared to class tests, and finally questions regarding the rough work they performed whilst solving the questions.

4.1 On Generated Code Questions

Whilst the top students regarded the questions as “easy and nice, because they were so short”, the weaker students complained that the feasibility of the program made it difficult to comprehend...

> **Student:**“Its hard to say what the program does, the variable names don't mean anything, and the program itself doesn't do anything. Its just a load of numbers going up and down.”

The feasibility of generated code questions seems to be a persistent problem; naming schemes are planned for the next generation of the system that aim to add some semantics to variable names. Whilst weak students struggle to comprehend code when it is void of semantics, the high performing students appear to experience no additional difficulty.

4.2 On Differences in Performance

As previously mentioned in Section 3.1, the high performing students performed equally well in both the generated exams and the class exams, as a result these questions were mainly intended for students who experienced a drop in performance when answering generated questions.

The weaker students complained about the rigidity of the marking in multiple choice questions. Many students complained that programming knowledge is not “black and white”. The opinion is best expressed by one student who scored 55% on a class exam on iteration, but 10% on the generated test for the same topic.

> **Interviewer:**“Why do you think you scored so low on the for loops survey?”
> **Subject:**“I know for-loops pretty well, I can write them for exam questions, I have a good idea how they work. But I get zero on these tests if I don't know the exact answer. When I am writing code in class exams I know I'll get *some* marks for getting the loop right, but here I get none. Thats what the difference is.”

This seems similar to what Lister describes as “fragile knowledge”. The students can, when requested, articulate a description of a particular piece of knowledge, but only when explicitly asked to do so. It would appear that this fragile knowledge is earning them marks in class tests, but certainly not in multiple choice questions. It is also possible that this is due the inherent differences between holistic assessment as employed in the class exams and the analytical assessment which is used in the automated assessment. It has previously been noted that holistic and analytical marking schemes can yield different results(Olson 1988), and in particular how a student's code can score reasonably high holistically whilst failing in simple analytical categories e.g. compilation/functionality.

Some students confessed to rote learning code, which is certainly troublesome. Rote learning assumes that the student has no comprehension (as per Bloom(Bloom 1956)) of the code, nor are they capable of applying the code to a given problem. According to the students it is the marking scheme which encourages rote-learning...

> **Interviewer:** "When you say you 'learn off' the code, what do you mean?"
> **Subject:**"Well, most of the questions are looking for the same thing, and you usually get the marks for making the answer *look* correct. Like if its a searching problem, you put down a loop, and you have an array and an if statement. That usually gets you the marks. "
> **Interviewer:**"What do you mean by 'the marks'?"
> **Subject:**"Not all of them, but definitely a pass"

This habit of rote-learning is consistent with Mayer's contrast of rote learning versus meaningful learning where he says novice programmers will rote-learn if they lack 'appropriate anchoring ideas'(Mayer 1981).

Ausubel(Ausubel, Novak & Hanesian 1978) explains that in order for comprehension (and hence meaningful learning) to occur, the new information in the short term memory must link to information in the long term memory and establish itself in the learner's knowledge network. If this does not happen the learner is reduced to memorizing each piece of information as if they were simply items on a list.

It is possible that this is what is happening with the weaker students; they are lacking the knowledge anchors for new concepts, and proceed to rote-learn as it is their only option.

4.3 On Doodling

As we had previously noted that doodling tended to correlate to a correct answer, so during interview it was decided to discuss this habit with students. Out of the 19 students interviewed 11 had previously provided some doodle data. The question asked was simply "Could you have done as well without using rough work?". Seven of the students answered yes, explaining that they were only using the rough work to make sure their answer was correct, the remainder said that without doodling they most likely would have made mistakes at some point. All students explained that they knew exactly how to solve the question when they began doodling, they just found it is easier to write numbers down rather than remember them.

Due to the high scoring of questions with doodle data, it would appear that students doodle because they are methodical and careful. It appears to be a good indicator of students with a good ethic. They may find the problems difficult, but they are meticulous enough to ensure they do not make any simple errors when attempting to solve them.

5 Conclusions

Having conducted an empirical study of the 59 students throughout the year, we have learned much about the nature of assessment in computer science. The automated assessment tool was evaluated thoroughly and proved to be successful for a large percentage of the class. It was successful in that these students did not comment in a negative manner about the tool, and their performance was consistent across both the tool and the class exams. However, this is true only for top students in the class(students with a grade 2 or higher). The students in the lower grades scored far less in the generated questions than they did in their corresponding class exams.

Based on data gathered during the interviews MCQs seem to provide a more accurate indication of a student's programming ability. The class tests where students were required to write programs to solve problems award marks for effort, which seems to promote rote-learning and as a result, fragile knowledge. If programming must be examined on paper, then it is best done through MCQs as devising a marking scheme for programs that does not have these failings has proven to be very difficult.

Our findings on the doodle data are largely in line with those proposed by Lister et al., in that 80% of the doodles resulted in the correct answer being chosen. Whilst again, not surprising, it is important information. During interviews students who doodled confessed to being 'perfectionists' and 'thorough' when approaching problems. These are traits that are expected of good programmers, and should be encouraged by educators.

5.1 Future Work

The future work for this project involves a modification of the system itself and secondly scaling the project up to cover more universities in an attempt to replicate and generalize the work. The system will be modified to choose reasonable or at least familiar variable names in the code in the hope of alleviating the main complaint from students thus far. In addition we would also like to perform experiments to assess the validity of this complaint, by measuring the difference in students' performance when semantically accurate variable names are provided.

6 Acknowledgments

The work presented in this paper is part of ongoing research in the National University of Ireland, Maynooth funded by the Irish Research Council for Science Engineering and Technology (IRCSET).

References

Ala-Mutka, K. M. (2005), 'A survey of automated assessment approaches for programming assignments', *Computer Science Education* **15**(2), 83–102.

Ausubel, D., Novak, J. & Hanesian, H. (1978), *Educational Psychology: A Cognitive View*, Rinehart & Winston.

Bloom, B. S. (1956), *Taxonomy of Educational Objectives Handbook 1: Cognitive Domain.*, McKay & Co.

Carter, J., Ala-Mutka, K., Fuller, U., Dick, M., English, J., Fone, W. & Sheard, J. (2003), How shall we assess this?, *in* 'ITiCSE-WGR '03: Working group reports from ITiCSE on Innovation and technology in computer science education', ACM Press, New York, NY, USA, pp. 107–123.

Daly, C. (1999), Roboprof and an introductory computer programming course, *in* 'ITiCSE '99: Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education', ACM Press, New York, NY, USA, pp. 155–158.

Eisner, M. P. (1998), 'The probability of passing a multiple-choice test', *The College Mathematics Journal* **29**, 421–247.

English, J. (2004), Automated assessment of GUI programs using JEWL, *in* 'ITiCSE '04: Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education', ACM Press, New York, NY, USA, pp. 137–141.

Higgins, C., Symeonidis, P. & Tsintsifas, A. (2002), The marking system for coursemaster, *in* 'ITiCSE '02: Proceedings of the 7th annual conference on Innovation and technology in computer science education', ACM Press, New York, NY, USA, pp. 46–50.

Kelly, J. O., Bergin, S., Mooney, A., Ghent, J., Gaughran, P. & Dunne, S. (2004), An overview of the integration of problem based learning into an existing computer science module, *in* 'Proceedings of the International Conference on Problem Based Learning'.

Kirsti Ala-Mutka, T. U. & Jrvinen, H.-M. (2004), 'Supporting students in C++ programming courses with automatic program style assessment', *Journal of Information Technology Education* **3**, 245–262.

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B. & Thomas, L. (2004), 'A multi-national study of reading and tracing skills in novice programmers', *SIGCSE Bull.* **36**(4), 119–150.

Lister, R. & Leaney, J. (2003), Introductory programming, criterion-referencing, and Bloom, *in* 'Proceedings of the 34th SIGCSE technical symposium on Computer science education', ACM Press, pp. 143–147.

Mayer, R. E. (1981), 'The psychology of how novices learn computer programming', *ACM Comput. Surv.* **13**(1), 121–141.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I. & Wilusz, T. (2001), 'A multi-national, multi-institutional study of assessment of programming skills of first-year CS students', *SIGCSE Bull.* **33**(4), 125–180.

Olson, D. M. (1988), The reliability of analytic and holistic methods in rating students' computer programs, *in* 'SIGCSE '88: Proceedings of the nineteenth SIGCSE technical symposium on Computer science education', ACM Press, New York, NY, USA, pp. 293–298.

Sprinthall, R. C., Sprinthall, N. A. & Oja, S. N. (1998), *Educational Psycholgy*, 7th edn, McGraw-Hill Education.

Traynor, D. & Gibson, J. P. (2005), Synthesis and analysis of automatic assessment methods in CS1: generating intelligent MCQs, *in* 'SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education', ACM Press, New York, NY, USA, pp. 495–499.