

Observations of Student Competency in a CS1 Course

Janet Rountree

Nathan Rountree

Anthony Robins

Robert Hannah[†]

Department of Computer Science

[†]Department of Classics

University of Otago

P.O. Box 56, Dunedin, New Zealand

Email: rountree@cs.otago.ac.nz

Abstract

Two issues of related interest are investigated in this paper. The first issue is associated with the statement that “Learning to program is a key objective in most introductory computing courses, yet many computing educators have voiced concern over whether their students are learning the necessary programming skills in those courses” (McCracken et al. 2001). The second issue considers which task CS1 students find more difficult: code generation or code comprehension. To investigate this, we analysed our CS1 course results in terms of laboratory exercises, comprehension, generation, factual/conceptual, and multiple-choice exam questions. Contrary to our initial expectations, the code comprehension and generation skills of our students appear to be tracking each other.

Keywords: CS1 assessment generation comprehension.

1 Introduction

Like many CS educators, particularly those with large first year programming classes, we have wondered precisely what programming skills our CS1 students have attained during their course. Students receive a grade for their work, but this result tells us little about the nature of their knowledge. We wished to explore how the competencies displayed by students differ not just in degree (i.e. final mark), but in the kind of skills they have developed (e.g. the ability to read code versus write code). It is a matter of general debate as to whether code comprehension is more or less difficult than code generation.

Code comprehension is an essential skill for programmers to develop, particularly since much of their time is spent on code maintenance. Spinellis (2003) relates a theory to explain why code can be easy to write and hard to read: there are many different ways to achieve a program specification but gradually, as the program progresses, choices narrow to a specific route. Conversely, when reading code, “. . . each different way we interpret a statement or structure opens many new interpretations for the rest of the code, yet only one path along this tree is the correct interpretation. We thus write code toward a decision tree root, but, unfortunately, read toward the tree’s branches.”

Copyright ©2005, Australian Computer Society, Inc. This paper appeared at the Australasian Computing Education Conference 2005, Newcastle, Australia. Conferences in Research and Practice in Information Technology, Vol. 42. Alison Young and Denise Tolhurst, Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

(Spinellis 2003, pp. 85–86). On the other hand, Lister (2000) makes the point that the ability of students to read computer code naturally outstrips their ability to write it, in the same manner that children develop language skills. Of course Spinellis is writing about his perspective on professional programmers, while Lister is involved with teaching CS1. However, there is an impression that the skills of comprehension and generation are not of equal difficulty, and there is some disagreement as to which is easier to develop. Further, in the context of learning to debug programs after having written them, Winslow (1996) notes, “Studies have shown that there is very little correspondence between the ability to write a program and the ability to read one. Both need to be taught along with some basic test and debugging strategies.”

It is generally taken as a necessary emphasis in a CS1 course that students will develop the ability to write simple programs that will compile and run. Aspects of CS1 education such as debugging and studying code fragments presented in lectures and textbooks develop (we hope) the ability to read code. We anticipated, however, that writing code is a skill that novice programmers find more difficult than reading code and that this would probably be reflected in marks received on different types of exam questions. To investigate the kinds of skills students at Otago have developed, we analysed the marks achieved on the different types of assessment undergone by our students in the 2003 academic year. These involved two laboratory assignments per week, a mid-semester test, and an end-of-semester exam that contained multiple-choice, factual/conceptual, code comprehension, and code generation questions.

2 Background

When students begin the introductory programming course at the University of Otago they are only expected to have a basic computing knowledge (e.g. be able to save files). By the end of the 13 week course they should know how to write a simple application; understand the purpose of objects, methods and data fields; and solve simple problems using loops and arrays. The language used is Java and, given the short nature of the course, data structures such as stacks, queues and trees are addressed later in a second year course.

The final mark awarded to each student is made up of a mid-semester test (15%), the successful completion of laboratory programming assignments (25%), and a final examination (60%). The aim of the laboratory assignments is to provide experience and practice of writing code, and to develop each student’s

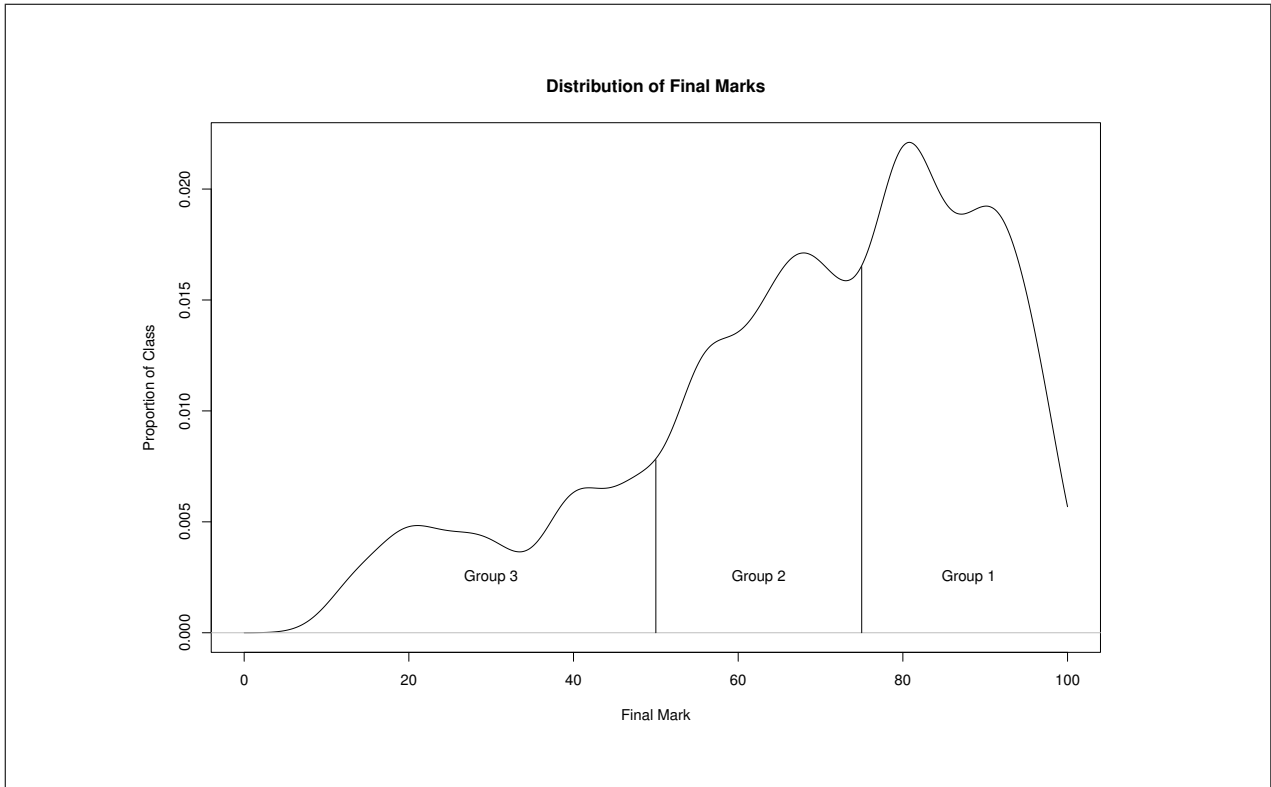


Figure 1: Group boundaries and distribution of final marks.

ability to solve programming problems. The final examination consists of a multiple-choice section and a short answer section. The type of questions asked in the short answer section can be divided into three categories:

1. Factual or conceptual questions: e.g. “What package of classes is automatically imported into every Java program?”
2. Comprehension questions (ability to read code): e.g. “Given the classes below, assume that `b` refers to an object of type `B`. What six lines of output are created when we call `b.test()`?” (16 lines of Java code are provided.)
3. Generation questions (ability to write code): e.g. students are asked to write a simple class with specified data fields, methods and constructors, and a second class which extends the first class. About 12 lines of code are required to complete this task.

The multiple-choice section includes questions from the first two categories with an emphasis on comprehension, while the short answer section is made up of all three categories with an emphasis on questions regarding generation. In this paper, for the purpose of analysis, there are six categories representing different kinds of skills or knowledge. These are (1) comprehension questions, (2) generation questions, (3) factual/conceptual questions, (4) multiple-choice questions (considered separately due to their prompting the student for the correct answer), (5) laboratory programming assignments, and (6) mid-semester examination (considered separately because it is taken very early in the course, and designed as a “wake-up call”).

3 Data Analysis

Our first task is to try to separate the students into sensible groups, in order to identify the mid-range students. This is necessary because students who get high grades tend to be good at everything: they seem to have understood and mastered all the examined tasks. The exact opposite is true for students who do poorly. The mid-range students are interesting due to their having understood only *some* parts of the course material and not others. Our interest lies in whether they have understood all parts of the course to about the same degree, or whether they have done particularly well in some of the six categories of assessment and poorly in others.

In 2003, 251 students sat the final examination. In order to separate these into three groups that did “well”, “modestly”, and “poorly”, k-means clustering (MacQueen 1967) was applied to the raw assessment data with $k = 3$. Each student record consisted of a mark for comprehension, generation, factual/conceptual, multiple-choice, laboratories, and mid-semester exam. Since the final mark (a linear combination of the six components) was not available to the k-means calculation, there was no particular expectation that the resulting groups would have a strong connection with final mark. However, the resulting clusters can essentially be characterised as those students who got above 75% (B+ grade or better), those who failed, (got below 50%), and those who got between 50 and 75%. This simple thresholding gets 240 out of the 251 students into the same group as that determined by k-means, so we take these as our definitions of the three groups: Group 1 (students who got over 75%), Group 2 (students who got between 50% and 75%), and Group 3 (students who got less than 50%).

Figure 1 shows the distribution of final marks and

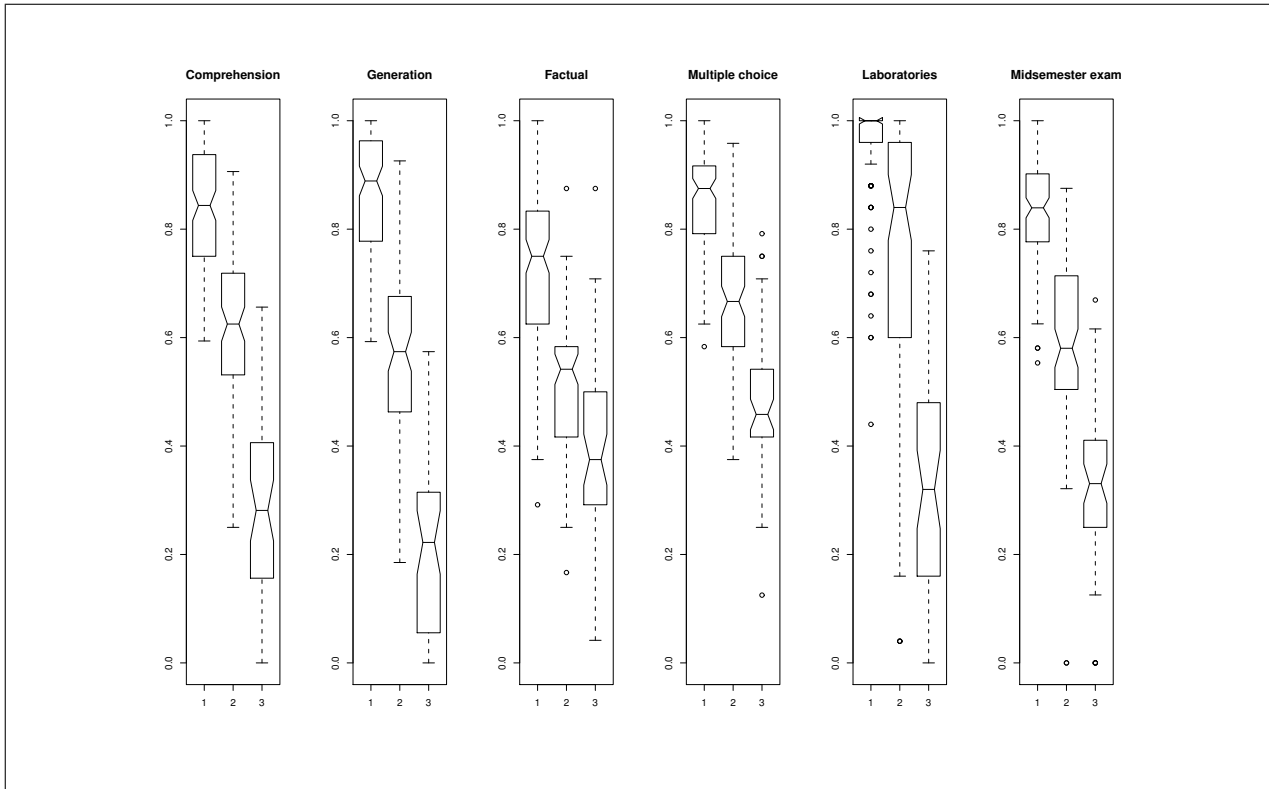


Figure 2: Boxplots for each of the six forms of assessment.

the cut-off points for the three groups. There are 114 students in Group 1, 88 students in Group 2, and 49 students in Group 3. From the plot of the distribution, it would appear that Group 3 really consists of two groups: those who fail and those who fail very badly.

It is possible to visualise the relative performance of these three groups by creating boxplots of their performance on each of the six assessment tasks. Figure 2 shows box-and-whisker plots for each of the groups, organised by type of assessment. The centre-stroke in each box represents the median, the top and bottom of the box represent the upper and lower quartiles, and the whiskers represent no more than 1.5 times the inter-quartile range. Data values beyond the whiskers are considered outliers. A useful addition to these boxplots is a notch cut either side of the median so as to approximate the median’s range; non-overlapping notches suggest a difference in medians with approximately 95% confidence. Medians and inter-quartile ranges are used in boxplots rather than means so that one can see where the central 50% of the data lie, and to provide a measure of central tendency that is insensitive to distribution or outliers.

These plots show that each element of our CS1 assessment is producing the same result in terms of separating students out into Groups 1, 2, and 3. Only the plots for “factual” questions in the exam have any serious overlap, and then only between the top of Group 3 and the bottom of Group 1. Clearly, Group 1 is strongly characterised by students who get full (or nearly full) marks in laboratories. Almost as strongly noticeable is that Group 3 students are cherry-picking the easier multiple-choice questions: but not enough to pass! Also obvious is that the one thing that Group 3 students really cannot do is generate code.

We note that there is no evidence of a difference in medians between comprehension and generation

questions. If skills in generating code were lagging behind skills in comprehension, then one of two situations would be expected: either the generation median of at least one of the three groups would differ from its comprehension median, or the generation and comprehension results would appear to be drawn from two different distributions. The boxplot shows that the comprehension and generation medians are not different at the 95% level of confidence, since the notches for Group 1’s comprehension results overlap with Group 1’s generation results, and so on. To test whether comprehension and generation results come from different distributions, a two-sided two-sample Kolmogorov-Smirnov test was done, producing a D-value of 0.0956 ($p = 0.2014$). There is therefore no evidence that comprehension and generation results do not come from the same distribution.

The results above suggest that the ability of our students to generate code is roughly tracking their ability to generate it—assuming our generation questions are not easier than our comprehension questions. (It is certainly true that generation questions are “easier” in the sense that we require students to write code that is less complicated than the code that they must comprehend. However, they are “as difficult” in that they represent precisely what we expect a student to be able to do under exam conditions and without a compiler.)

This result does not support the expectation that students’ ability (after just one semester of study) to read code outstrips their ability to write code. Why, then, are our students finding generation questions neither harder nor easier than comprehension questions? The answer may lie in the results for laboratories. The students are required to attend two two-hour laboratories per week, each of which is worth one percent if completed successfully. Successful completion requires a small program that runs successfully

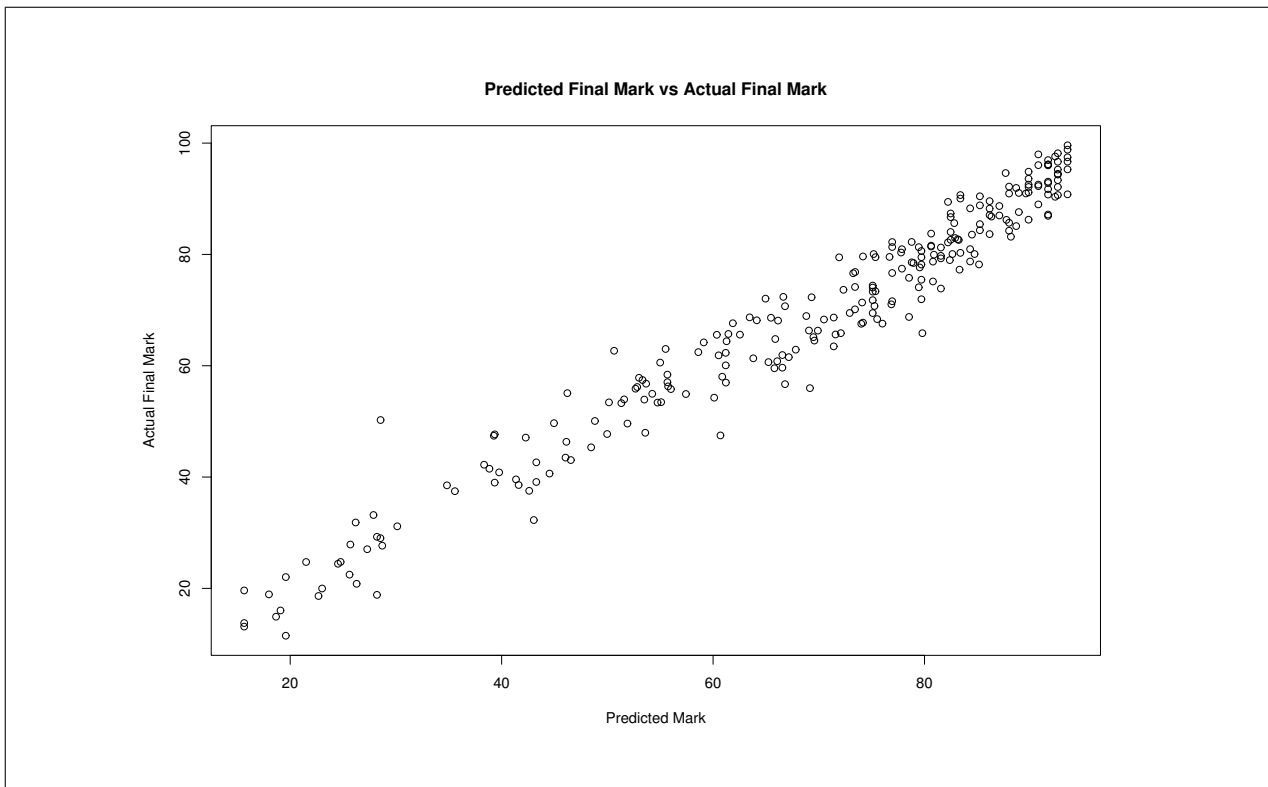


Figure 3: Correlation of predicted mark vs. final mark.

on some test data, and code judged by the demonstrator to be satisfactorily tidy, clearly laid out, and appropriately commented. There are 25 labs in total, requiring considerable commitment from students. We observe that only the 9 outliers in Group 1 missed more than 2 of the 25 labs; that most of Group 2 completed between 15 and 24 of the labs; and that only one outlier in Group 3 completed more than 15.

We believe that this pattern of lab completion can account for the relationship between the comprehension and generation marks. The lab medians of Groups 1 and 2 are higher than the corresponding medians for comprehension and generation, which might lead us to suppose that the lab marks were easier to get than the exam marks, were it not for the fact that there is no evidence of a difference in medians for Group 3's results in labs, comprehension, and generation. We therefore suspect that the generation results for Groups 1 and 2 may have been lower if they had had less commitment to completing labs—a proposition that is rather difficult to test ethically.

One further indication of the link between the laboratory and generation results is what happens when they are combined to form a linear model to predict final mark. The final mark predicted has a 98% correlation with actual final mark as shown in Figure 3, a strength of correlation that is not achieved with any other pair of the six results. With generation and lab results converted to proportions, the prediction formula calculated by least-squares regression is:

$$\text{predicted} = 14.48 + 49.76(\text{generation}) + 29.29(\text{labs})$$

This result accords very strongly with our day-to-day observation that the most at-risk students are those who are not completing labs regularly and on time.

4 Related Work

Studies of code comprehension are quite plentiful, both of expert programmers (e.g. Koenemann & Robertson 1991, Corritore & Wiedenbeck 2000) and of novices (e.g. Corritore & Wiedenbeck 1991, Ramalingam & Wiedenbeck 1997). Most are concerned with trying to determine the reader's conceptual model of the program, whereas we are trying to ascertain whether certain educational goals have been met. By contrast, studies of code generation are not plentiful: McCracken et al. (2001) is notable for its focus on determining whether the programs that students *produce* actually do the jobs they are supposed to do.

The extent to which assessments reflect educational goals has been discussed by Lister (2001) and a criterion-referenced grading scheme based on Bloom's taxonomy suggested by the same author (Lister 2000). While our CS1 assessments are not criterion referenced, they do represent tasks that we expect a competent and conscientious student to be able to perform at the end of one semester's instruction. Crawford & Fekete (1997) investigate the extent to which exam questions test particular skills using factor analysis; we intend in future work to use a similar idea to determine when different questions in the same exam appear to be testing the same thing. This approach may help to identify different types of comprehension/generation questions, and which types (if any) are harder than others.

5 Conclusion

It is necessary that CS1 students develop skills to both read and write code. There is, however, an impression of a lack of correspondence between the skills of code generation and comprehension. It is not clear

as to which of comprehension or generation is easier, although there is a reasonable expectation that the skill of learning to read code precedes the ability to write it. At the conclusion of our one-semester CS1 paper we analysed the marks achieved by students on different aspects of the course.

First, we used cluster analysis to determine which students were most similar to our concept of “good”, “poor”, and “in-between”. Comparing the results of these groups suggested no within-group difference in their ability to read code compared to their ability to write it. Furthermore, there is no evidence that the comprehension results and the generation results are not drawn from the same distribution. We conclude that—if our comprehension and generation questions are indeed representative of the level we expect CS1 students to achieve—our students’ ability to read simple classes and methods matches their ability to write them.

This result seems to be linked to each student’s commitment to completing successfully the 25 2-hour laboratory sessions. Since it is possible to predict each student’s final mark with 98% correlation using only the lab marks and responses to code generation questions, we admit that our CS1 course is biased toward code-writing. However, if learning to write code is in fact more difficult than learning to read it, our course emphasis may provide the means by which students achieve a balance in the two skills. While in principle these may be two different skills, in practice if both are taught explicitly then perhaps performance in each can be equivalent.

Acknowledgements

We would like to thank Dr Richard O’Keefe for his interest and suggestions and to acknowledge the financial support provided by the University of Otago Research Committee, by means of the University of Otago Bridging Grant.

References

- Corritore, C. L. & Wiedenbeck, S. (1991), ‘What do novices learn during program comprehension?’, *International Journal of Human-Computer Interaction* **3**(2), 199–222.
- Corritore, C. L. & Wiedenbeck, S. (2000), Direction and scope of comprehension by procedural and object-oriented programmers: an empirical study, in ‘IEEE 8th International Workshop on Program Comprehension (IWPC 2000)’, pp. 139–148.
- Crawford, K. & Fekete, A. (1997), What do exam results really measure?, in ‘Proceedings of the 2nd Australasian Conference on Computer Science Education’, pp. 185–190.
- Koenemann, J. & Robertson, S. P. (1991), Expert problem solving strategies for program comprehension, in ‘Proceedings of the SIGCHI conference on Human factors in computing systems’, ACM Press, pp. 125–130.
- Lister, R. (2000), On blooming first year programming, and its blooming assessment, in ‘Proceedings of the Australasian Conference on Computing Education’, ACM Press, pp. 158–162.
- Lister, R. (2001), Objectives and objective assessment in cs1, in ‘Proceedings of the 32nd SIGCSE Technical Symposium’, pp. 292–296.
- MacQueen, J. (1967), Some methods for classification and analysis of multivariate observations, in L. L. Cam & J. Neyman, eds, ‘Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability’, Vol. 1, pp. 281–297.
- 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 Bulletin* **33**(4), 125–180.
- Ramalingam, V. & Wiedenbeck, S. (1997), An empirical study of novice program comprehension in the imperative and object-oriented styles, in ‘Papers Presented at the Seventh Workshop on Empirical Studies of Programmers’, ACM Press, pp. 124–139.
- Spinellis, D. (2003), ‘Reading, writing, and code’, *ACM Queue* **1**(7), 84–89.
- Winslow, L. (1996), ‘Programming pedagogy—a psychological overview’, *SIGCSE Bulletin* **28**(3), 17–22.