Testing the perception of time, state and causality to predict programming aptitude

José Paulo Leal
DCC/FCUP & CRACS/INESC-TEC
University of Porto, Portugal
Email: zp@dcc.fc.up.pt

Abstract—**The aim of the research presented in this paper is the development of a novel approach to predict programming aptitude. The existing programming aptitude tests rely on the past academic performance of students, on their psychological features or on a combination of both. The novelty of the proposed approach is that it attempts to measure student capabilities to manipulate abstract concepts that are related with programming, namely time, state and causality. These concepts were captured in OhBalls - a physical simulation of the path taken by a sequence of balls through an apparatus of conveyor belts and levers. An engine for this kind of simulation was implemented and deployed as a web application, creating a self-contained test that was applied to a cohort of first-year undergraduate students to validate the proposed approach. This paper describes the proposed type of programming aptitude test, a software engine implementing it, a validation experiment, discusses the results obtained so far and points out future research.**

I. Introduction

PROGRAMMING is hard to learn for most people [9], [12]. Although some students seem to learn it without apparent difficulty, this is not the case of most of them, especially those majoring in subjects other than computer science or software engineering. Even students in those areas are not immune to these problems, as many educators feel that they do not acquire the necessary programming skills in introductory courses [10]. Probably some students are not cut out to be programmers and knowing it in advance would a great advantage [7].

Predicting which students are likely to succeed in learning programming is not easy, some even say it is unfeasible [9]. Nevertheless, this is precisely the goal of the research described in this paper: to create a programming aptitude test, and especially to explore new ways to predict programming aptitude.

Several studies suggest that the high school performance in mathematics is the best indicator of a programming aptitude in college [3], [6], although the correlation between both is usually small. The standard explanation is that both mathematics and programing require abstract reasoning, thus a student with a good performance in mathematics during high school is bound to succeed also in college programming courses.

It is indisputable that a connection exists between programming and mathematics, at least in a broad sense. Although it is much more difficult to define mathematics than programming, if we accept that the realm of mathematics is patterns of thought them clearly computer science in general, and computer programming in particular, are deeply connected to this discipline. However, the mathematical knowledge of a high school student is essentially algebra, calculus and basic logic, and these fields lack a number of abstract concepts that are essential in programming, namely time, state and causality.

The concept of *time* is central to computing. Computers have an internal clock that regulates how instructions are executed. Programs are sequences of instructions that are executed in a flow over a period of time. All the elements of a computer, or of a program, are in a certain *state* that evolves over time. The execution of instructions *causes* changes to these states, that are influenced by their previous states.

This type of reasoning, however abstract, is in general absent from mathematics, especially from the branches of mathematics taught in high school. Take calculus for instance. Although the variable of a function may be interpreted as time, the function itself exists beyond time; it always existed and never changes. A derivative may be seen as the amount of change of the function's value but it has no discernible causes or consequences. Or take Boolean logic as another instance, where A implies B that does not mean that A precedes B or caused by B. If A is true then it has always been true and will never change. This timelessness exists also in algebra where "variables" are actually "unknowns", values that can and eventually will be determined by a computation, not values that actually change over time.

It can be argued that some programming languages, namely declarative languages, are closer to mathematics and above these concepts of time, state and causality. Although this is true, these languages are not widely used and certainly not used as much as Java, C/C++ and Python in introductory programming courses [11]. Moreover, although the denotational semantics of these languages may be independent from time and state, their operational semantics depends on them and the programmer must understand them, if not for anything else, to be able to debug programs.

An algorithm is probably the mathematical concept learned before college that most closely resembles to a computer program. However, students learn specific algorithms that they execute, for instance the division algorithm, rather than study algorithms as a topic, without actually creating new algorithms. These differences between mathematics and programming are possibly the reason why a student may reveal aptitude for maths but none for programming and vice versa, and why math grades are insufficient to predict proficiency in programming.

The motivation for this research comes from the intuition that there is a kind of reasoning that is specific to programming, that is different from the reasoning required in mathematics. Based on this insight, the goal of this research is to develop a new kind of test based on the concepts of time, state and causality. The test should be self-contained, in the sense that it should not require a person to administer it and should not require any previous knowledge of programming concepts.

The remainder of this paper is organized as follows. Section II reviews the related work on predicting programming aptitude. Section III presents the proposed type of aptitude test and describes a JavaScript engine implementing it. Section IV reports on an experiment to evaluate the proposed type of test. The final section summarizes the research conducted so far and identifies paths for future research.

II. Related work

Predicting programming aptitude is still a challenging task, although this topic is being studied for more than 40 years [1] and its relevance has been well establish for almost a quarter of a century [7].

Most of the recent research in the literature attempts to correlate programming aptitude with factors that are unrelated with programming, either the student academic record or psychological features. As part of the academic performance the most relevant factor is previous math grades [2], [3], [8] although science grades and even average grades have also been investigated. Other plausible factors were also investigated, such as creativity, problem solving aptitude, attitude toward computers, with even lower correlations [6]. None of these factors has yet provided a good predictor of programming aptitude.

The test recently proposed by Dehnadi [5] differs from the previous since it is actually related to programming; it is a sequence of questions on Java assignment statements. Dehnadi claims that *consistency* in the interpretation rather than its correctness is the main indicator of programming aptitude. The purpose of the test is not to discriminate students who answered correctly, which would assume prior knowledge of programming. The purpose is actually to determine those who answered consistently according to a single mental model. This would reveal the ability to create a meaningful rule to interpreter the assignment command, from which the aptitude to learn a programming language could be inferred. Unfortunately, this experiment was later repeated by Caspersen [4] and also by Wray [12] and none was able to reproduce the results of Dehnadi.

Wray [12] explored the known link between autism and descendants of mathematicians and scientists. He proposed an alternative method for predicting programming aptitude based on mild autistic-spectrum related questionnaires from the Autism Research Center. These questionnaires tests two facets of autism: the level of understanding of systems of objects (SQ) and the level of understanding of other people emotions (EQ). Individually these tests are moderately correlated, but combined they provide a good correlation (r=.67) with programming aptitude. However, the test was applied only to 17 students after they have completed an introductory programming course, and no subsequent results were published on the use of this method to predict programming aptitude.

III. Testing programming aptitude

The goal of this research is to develop a new kind of test to estimate programming aptitude. This kind of test intends to estimate the perception of time, state and causality, under the assumption that these concepts are present in programming reasoning and should reveal an aptitude to program. Also, the test must not require any programming knowledge and be self-contained, in the sense that anyone should be able to take it alone, without or with minimal supervision.

The test that is being developed is based on a set of physical simulations with a common scenario called *OhBalls*. This name comes both from the blue ball than moves on a screen, and the interjection that participants pronounce when it doest not behave exactly as expected. Since it is based on a simulation the test must taken on a computer, which nowadays is hardly a difficulty. In its current implementation the OhBalls test is deployed on the web hence it can be taken virtually anywhere.

A test with the OhBalls scenario is composed by a sequence of panels. Each panel presents a physical simulation set in a room where balls are dropped from a pipe on the ceiling, move through a system of conveyor belts and eventually fall in one of several buckets on the floor. The participant must predict the number of balls that will land on each bucket when the simulation is executed. Figure 1 depicts a example of those panels.

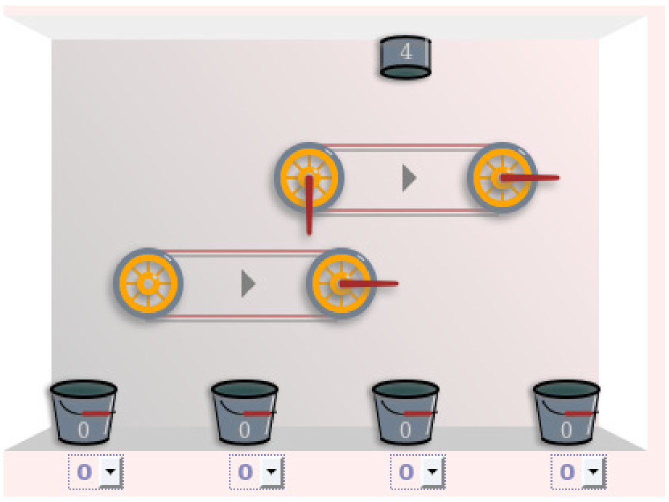

Fig. 1. Example of an OhBalls panel

Balls falling from the pipe land on a conveyor belt that carries them either to the left or to the right. The direction in which the upper side of the belt is moving and the ball would be carried is shown by the arrow head in the middle of the conveyor belt. This direction can be reversed by one of the levers connected to the wheels of the conveyor belt when a ball pushes it. For instance, the first ball falling from the pipe in Figure 1 will be carried to the right by the top conveyor belt and fall in the rightmost bucket. When falling to the bucket it will activate the right lever of the top conveyor belt, reversing its direction, and thus the second ball will go to the left.

The levers have always the same effect – reverse the conveyor belt to which they are connected – but are activated in different ways. For instance, the right lever on the top belt is activated when the ball falls out of the belt, while the left lever on the same belt is activated when the ball is carried to the right by the bottom belt. Hence, the second ball will fall on the second bucket from the right, and will reverses the direction of both belts.

The levers change the state of the system from one ball to the other. In the panel of Figure 1, when the third ball falls from the pipe the top belt will be moving again to the right, while the bottom one will be moving to the left. Hence, this ball will eventually land on the rightmost bucket, raising its count to 2, while reversing the top belt. At this moment both belts will be moving to the left. Thus, the fourth and last ball will be carried to the left, and is dropped on the leftmost bucket. Any subsequent ball would also end in the leftmost bucket but this simulation stops after 4 balls.

The number initially on the pipe indicates the number of balls that will fall during the simulation, and each bucket counts the number of balls that reached it. Under each bucket there is a selector where the participant predicts the number of balls that will reach it when the simulation is completed. The simulation is started by pressing a button (not shown in Figure 1) that is activated when the sum of these selectors equals the number in the pipe.

During the simulation balls fall from the pipe on the ceiling one after another. When a ball is dropped the counter in the pipe is decreased. When the ball reaches a bucket its counter is increased. Then a new ball is dropped from the pipe if this counter has not yet reached 0. When all balls reach a bucket the simulation stops and the number in each bucket is compared with the number in the selector beneath it. The participant is considered to have successfully predicted the outcome of the simulation if these figures match for all buckets. The participant may replay each simulation several times but will not be able to change the answer made before executing the simulation for the first time.

It should be noted that levers do not stop balls by themselves. They are activated by balls and change the direction of belts that carry them through the apparatus. For instance, the left lever on the top belt will be activated when a ball is moving to the right on the lower belt. The lever will not stop the ball in this belt but will reverse the direction of the top belt, affecting only the following balls. If this particular lever (in the left on the top belt) was pointing up instead of down then it would affect balls coming to the left in the top belt. It would not stop them but it would reverse the belt before they reach the point where they would fall.

In summary, the apparatus is composed of one or more conveyor belts that carry a ball falling from the pipe to the buckets on the floor. The motion of each belt is given by a pair of wheels. Levers are always bound to a belt, more precisely to one of its wheels, and can be in 4 possible directions (left, right, up and down).

It is obvious that many panels of this kind can be created with a different number of belt and lever settings. For instance, with a single belt there are 16 different combinations. On each side there are 4 possibilities for placing a lever: no lever at all, pointing up, pointing down, left or right, depending on the position of the wheel[1]. More than one lever per wheel would be possible but would also be too confusing. In a panel with 2 belts these must be arranged so that falling balls go either to another belt or to a bucket.

The easiest way to achieve this is to use a *grid* to place the center of the belts, the buckets and the pipe. The distance between the wheels of a belt must be set in a way that balls are dropped in alignment with the center of other belts and buckets positioned bellow them. Also, the distance between consecutive rows must take in consideration that a certain gap between belts, large enough for the ball to move between them, and small enough for the ball carried by the lower belt to activate a lever in upper belt. The pipe should be at the center and have a belt aligned bellow.

With this approach a setup with 2 belts in two different ways can be created, with the lower belt either to left or to the right, with a total of 512 possibilities. It would not make sense to place belts exactly over each other, or any relative position where balls would not go from one to the other.

An engine to execute this kind of simulation was implemented in JavaScript using the HTML 5 canvas element with a 2D context. This engine runs on a recent version of all major web browser. It can be parametrized with any number of panels following the approach described above. Currently the panels are limited to a grid of 5 rows by 7 columns, which is large enough to place 4 belts, a pipe and bottom row of buckets in each column of the grid.

The current version of engine supports only the prediction of the simulation outcome. In a future version it should allow the participant to place levers in order to achieve a certain configuration. Obviously, this is much closer to the reasoning involved in programming than the current implementation, which is comparable to tracing a program for debugging it. Implementing this feature is not difficult. The main reason for not having it in the first version was lack of knowledge on how the participants would react to this kind of test and the possibility that they would find it too complex as it is.

[1]a lever pointing inwards would be senseless

IV. Experiment

An experiment was designed to investigate how potential participants perceive the OhBalls type of test and its effectiveness in predicting programming aptitude. For this experiment a number of students enrolled in an introductory programming course took an OhBalls test and the outcome was compared with the grades of their middle term exam. This section presents the web application developed as the main instrument for this experiment, analyses the data collected with it and discusses the obtained results.

The web application developed for the experiment is based in the simulation engine described in section III. It allows a considerable number of participants to take the test simultaneously, while it collects data for later processing.

The interaction of each participant with the web application proceeds in four stages: identification, questionnaire, tutorial and test. The total time of each participation is about 20 minutes. To start the participation each student introduces his or her ID that is checked against a list previously loaded into the application, ensuring that each student participates only once. After being identified, the participant completes a small questionnaire with demographic data, mathematics and average grades from high-school, and former experience with computer and programming.

The OhBalls test is preceded by a tutorial that explains how it works. The tutorial runs on the same type of interface and highlights each important part while a text in a message box provides the necessary details. It explains how the balls are carried by belts in the simulation, how they activate levers and these change the direction of belts, how they reach buckets and new balls repeat the simulation until a predefined number of balls is processed trough the simulation. This tutorial explains also that the participant must predict the number of balls ending in each bucket before running the simulation, and how to activate it and proceed to the next panel. This tutorial runs in a loop until the participant decides to start the test. During the test the participant may rerun this tutorial, if needed.

The OhBalls type of test configured for this instrument consists of a sequence of 30 panels. Each panels is accompanied by a small text that emphasizes a particular point that was not present in the previous ones, such as “note that levers may by activated while balls are falling”.

The first set of panels has a single conveyor belt, and each panel is increasingly more complex than the previous ones. The following set of panels has two belts also with increasing complexity. Nevertheless, the first panels with two belts are less complex than the last ones with a single belt, since they have no levers or just a single lever. The following two sets, with three and four belts, are ordered in the same fashion: the first panels are fairly easy and the last are more difficult.

After the simulation is run the participant is informed if she succeeded in predicting the outcome of the simulation, the time she took to complete it (the number of seconds the panel was shown before pressing the button to start the simulation) and the percentage of correct answers.

When the participant proceeds to the next panel the application sends the data it collected to the server. The data collected for each panel includes the time the student took to complete it, the number of balls the student expected in each bucket and a Boolean indicating if the outcome of the simulation was predicted with success or not.

The participants were students enrolled for the first time in an introductory programming course. This course is common to the computer science and computer engineering programs offered by the computer science department of the faculty of sciences at the university of Porto. The course syllabus is problem solving oriented and uses C as programming language.

The experiment took place in September of 2012 during their first practical class and the participation was optional. Although no student refused to participate in the experiment, those that were unable to complete the test due to timetable constraints were excluded. The students received a brief explanation on the purpose of the experiment and were assured that their participation would not have any impact on their course grades.

The number of participants in the OhBalls test was 153 of which 115 where considered valid. Of these students a considerable number decided they were not ready to take the middle term exam. Only the data referring to the 57 students that took also the middle term exam was used in this experiment. Of these 57 students considered in the experiment the number of females was 10 (17.5%) and 18.16 was the average age.

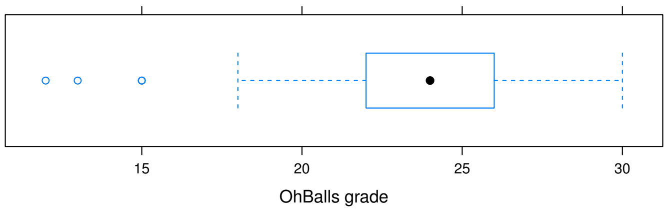

Fig. 2. Box plot of experiment grades

The time taken in each panel by the participants varied from 1 to 558 seconds, with a mean of 33.25 seconds. A possible inverse correlation between time spent analysing the panel and the a correct prediction was investigated, but it was not very high ($c = -0.24$).

To measure the outcome of each participant’s test a grade was computed by assigning 1 point to to each panel correctly answered and 0 otherwise. Thus, each participant had a grade with range 0 to 30 (the number of panels) assigned to her. Considering the series of test grades, the minimum grade was 12, the mean 23.3, the median 24 and the maximum 30. A 5 value summary of the ObBalls grades in the experiment is shown in Figure 2.

There was a good number of very simple panels, to make sure the participants understood the test, but most likely this difficulty was overestimated. In fact, taking 1 for a panel correctly answered and 0 otherwise, the overall median of was 1 and the mean 0.74; by panel, in 23 out 30 the median was 1. The data suggests that the OhBalls test used in the experiment is too simple and more complex panels are needed.

The correlation of the OhBalls grade with the middle term exam was not high ($c = 0.31$) and inferior to the correlation between the high school math grade reported by the students ($c = 0.39$). Nevertheless it was possible to identify a subset of 8 panels for which the correlation is comparatively high ($c = 0.54$). Figure IV is a plot of the the grade for this selected set of panels (as percentage) and the grade in the middle term exam (also as percentage).

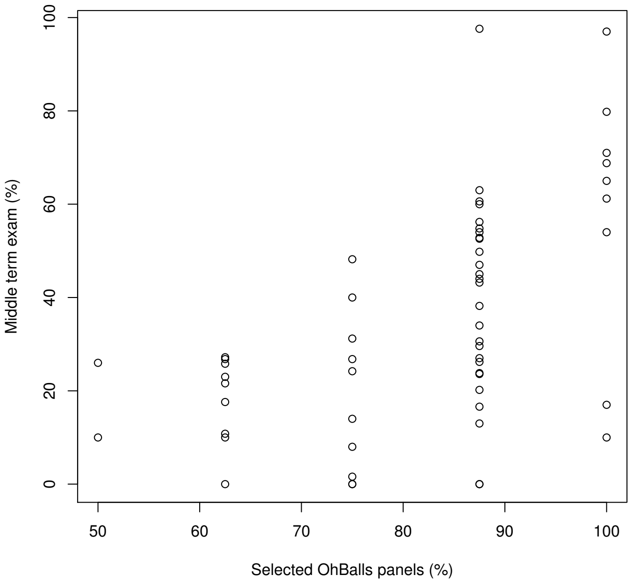

Fig. 3. Scatter Plot of selected panels and mid-term grades

Figure IV shows that the selected OhBalls panels predict the maximum grade the student will obtain, i.e. the student grade is almost always lower than the grade obtained in the selected panel of OhBalls.

The grade in the selected panels was also used in complement with other factors that are known to have influence in programming aptitude, namely math and other high school grades. The initial questionnaire collected the average grade used by Portuguese state universities to rank students applying to their programs. This average includes in equal parts the high school average grade and the national exams grades in particular subjects. In this case these subjects can be either math alone or math, physics and chemistry. A sum of equal parts of these 2 grades (selected OhBalls panels and average grade) reached a higher correlation (c=.64). By comparison, the average grade alone obtained a smaller correlation (c=0.57). The scattered plot of these combined grades with the mid-term grades, with the regression line in red, is presented in Figure 4.

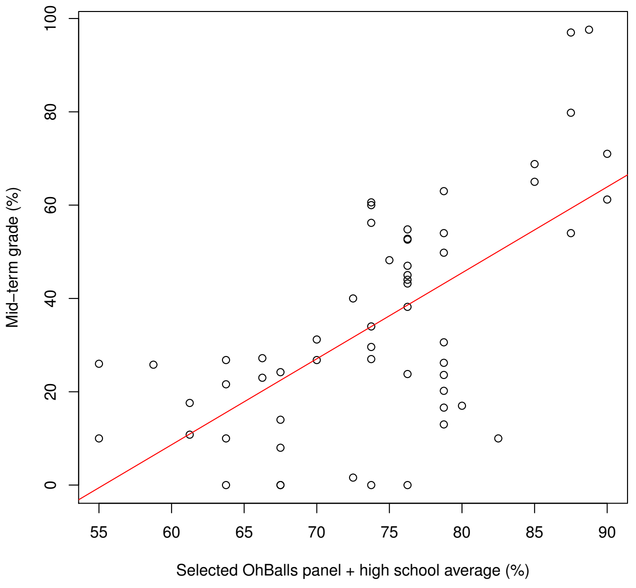

Fig. 4. Scatter plot of selected panels + average and mid-term grades

In any event, these are just preliminary results suggesting that there is room for improvement. The OhBalls panels used in this test were clearly too simple and more complex ones should be used. The capacity for the participants to understand the test seems to have been underestimated. With the current implementation of the OhBalls engine it is easy to create more complex panels. However, new types of panels where the participant must place levers in order to achieve a certain outcome (number of balls in each bucket) should also be used.

The experiment showed also that the OhBalls type of test is able to engage students. They were much more quiet and focused when they were taking the test than they were in the rest of the class. This kind of test has a game-like quality that motivates students to complete it, which is a requirement if students have to take the test on their own without supervision.

V. Conclusion and future work

This paper presents a novel approach to estimate programming aptitude based on the understanding of the concepts of time, state and causality. The proposed type of test is named OhBalls and does not require any prior knowledge of programming since it is based on physical simulations displayed on a a web application. The object of the simulation is the path of a sequence of balls trough an apparatus of conveyor belts and levers, until they reach a bucket, which is simple to understand by any undergraduate student. The test is self-applicable, in the sense that it does not require the presence of a person supervising its application.

An OhBalls test was applied to a cohort of computer science and software engineering undergraduate students. The initial results are promising but reveal that more work is still needed to fine tune the test. The average results are comparatively high, suggesting that larger number of panels, and more difficult panels ones, are necessary. Still, a subset of the panels from the current version has a reasonably high correlation with the student intermediary grades.

The main conclusion is that the OhBalls tests must have more panels and more difficult ones, in order to discriminate better the students with higher understanding of time, state and causality. Moreover a new class of panels will be added with a different type of challenge. Instead of simply predicting the number of balls reaching each bucket, considering the influence of the levers, the participant will have to position levers bound to the belts to achieve a certain configuration of balls in the buckets. The kind of reasoning involved will be closer to programming, since currently it can be considered closer to debugging.

In the continuation of this research the methodology of the experiments will have also to be changed. In the experiment presented in this paper only the students that took the middle term test were considered and the students that dropped out where ignored. This "negative" information will be taken in consideration when comparing with the final results.

The current results obtained with the OhBalls test used in the experiment need to be checked against not only the final course grades but also with other programming courses that these students are going to take in the following semesters. To prove the effectiveness of OhBalls test the kind of experiment presented in this paper must be repeated in computer science programs with different pedagogical approaches, in different universities and countries.

ACKNOWLEDGMENT

The author wishes to thank to the students that voluntarily participated in the test reported in this paper, as well as to the lecturer and teaching assistants of the introductory programming course where it took place. This work is in part funded by the ERDF/COMPETE Programme and by FCT within the FCOMP-01-0124-FEDER-022701 project.

REFERENCES

[1] Carol Ann Alspaugh. Identification of some components of computer programming aptitude. *Journal for Research in Mathematics Education*, 3(2):pp. 89–98, 1972.

[2] Susan Bergin and Ronan Reilly. Programming: factors that influence success. *SIGCSE Bull.*, 37(1):411–415, February 2005.

[3] Pat Byrne and Gerry Lyons. The effect of student attributes on success in programming. In *Proceedings of the 6th annual conference on Innovation and technology in computer science education*, ITiCSE '01, pages 49–52, New York, NY, USA, 2001. ACM.

[4] Michael E. Caspersen, Kasper Dalgaard Larsen, and Jens Bennedsen. Mental models and programming aptitude. In *Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, ITiCSE '07, pages 206–210, New York, NY, USA, 2007. ACM.

[5] S. Dehnadi. Testing programming aptitude. In *Proceedings of the 18th Annual Workshop of the Psychology of Programming Interest Group*, pages 22–37, Brighton, UK, 2006.

[6] Yavuz Erdogan, Emin Aydin, and Tolga Kabaca. Exploring the psychological predictors of programming achievement. *Journal of Instructional Psychology*, 35(3):264–270, September 2008.

[7] Gerald E. Evans and Mark G. Simkin. What best predicts computer proficiency? *Commun. ACM*, 32(11):1322–1327, November 1989.

[8] Annagret Goold and Russell Rimmer. Factors affecting performance in first-year computing. *SIGCSE Bull.*, 32(2):39–43, June 2000.

[9] Tony Jenkins. On the Difficulty of Learning to Program. In *3rd annual Conference of LTSN-ICS,*, Loughbourgh, 2002.

[10] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. *SIGCSE Bull.*, 33(4):125–180, December 2001.

[11] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. A survey of literature on the teaching of introductory programming. In *Working group reports on ITiCSE on Innovation and technology in computer science education*, ITiCSE-WGR '07, pages 204–223, New York, NY, USA, 2007. ACM.

[12] Stuart Wray. Sq minus eq can predict programming aptitude. In *PPIG 19th Annual Workshop*, University of Joensuu, Finland, July 2007.