Mental models, Consistency and Programming Aptitude

Richard Bornat[1] **Saeed Dehnadi**[1] **Simon**[2]

[1] School of Computing Science, Middlesex University, LONDON NW4 4BT, UK
Email: {R.Bornat,S.Dehnadi}@mdx.ac.uk

[2] School of Design, Communication, and Information Technology, University of Newcastle, Australia
Email: Simon@newcastle.edu.au

Abstract

Learning to program is notoriously difficult. Substantial failure rates plague introductory programming courses the world over, and have increased rather than decreased over the years. Despite a great deal of research into teaching methods and student responses, there have been to date no strong predictors of success in learning to program.

Two years ago we appeared to have discovered an exciting and enigmatic new predictor of success in a first programming course. We now report that after six experiments, involving more than 500 students at six institutions in three countries, the predictive effect of our test has failed to live up to that early promise. We discuss the strength of the effects that have been observed and the reasons for some apparent failures of prediction.

Keywords: Programming aptitude

1 Introduction

Despite numerous counterexamples, it appears normal to believe that with careful teaching, adequate motivation and sufficient time, anybody can learn anything. Some teachers would make an exception for mathematics, others for music or art, while many programming teachers would make an exception for programming. The ability to read and write programs is widely held to be essential for study in computer science, but a significant proportion of those attempting a first university course in programming fail to learn these skills.

Part of the problem is that the subject is not widely taught at school, so undergraduates arrive without having being streamed into those who can do well and those who can't. A reasonable predictor of success in learning to program would be invaluable here – just as, before computer science graduates became widely available, a reasonable predictor of programming ability would have been invaluable when recruiting workers for the computer industry. The need has been obvious throughout the entire half century or so that the industry has existed and the four decades or so that the subject has been taught, yet, despite considerable research, no adequate predictors have yet emerged.

The problem is that there is not a simple distribution of ability, some being merely better than others. It really seems that a significant proportion never learn to program during their university course, as reported by McCracken et al. (2001), for example. A strongly bimodal distribution of marks in the first programming course is frequently reported anecdotally, and corresponds to our experience in several different academic institutions over a considerable period of time.

Attempts to predict programming success often begin with the intuitions of individual researchers, among whom Soloway was a pioneer. Adelson & Soloway (1985) reported that familiarity with the problem domain helps novices to understand the programming problems that they are asked to solve. Perhaps less obviously, Soloway & Spohrer (1989) concluded that skill in natural language seemed to have a great deal of impact on students' conceptions and misconceptions of programming. In a result that some would find counter-intuitive, Bonar & Soloway (1985) put forward evidence to support the startling theory that prior knowledge of one programming language has a *negative* impact on novices' attempts to program in a second language. Other researchers (van Someren 1990, Mayer 1992, Canas et al. 1994) concluded that people who know how their programs work are more successful than those who do not.

Hand in hand with the search for predictors of success was the more pragmatic approach of developing techniques or tools to make programming easier for all novices. Some, for example Bornat (1986), chose to teach programming by way of formal reasoning. If expert programmers could justify their programs, perhaps novices could be taught to program by first learning to formalise their reasoning. Others devised integrated development environments (IDEs) in the hope that a point-and-click environment would free novices from the minutiae and leave them to concentrate on the algorithms (Kölling et al. 2003, Vaughan-Nichols 2003). Yet others devised ways of measuring the relative difficulty of different programming languages (Green 1997), or designed whole new programming languages, for example LOGO (Papert 1980) and Miranda (Turner 1985), intended to simplify the task of programming. Despite massive efforts in all of these directions, there is no evidence that these initiatives have had any significant impact on the success rate among novices.

Another approach involved studying the problems encountered by novice programmers, either categorising the problems explicitly and discovering that they fall into a handful of types (Bonar & Soloway 1983) or, at a more abstract level, categorising the deeper domains in which the mistakes lie (du Boulay 1986).

A final approach explored the way in which novices and/or experts go about solving problems. Putnam et al. (1986) found that novices' misconceptions about the capabilities of computers could have a massive negative impact on their success at programming. Pennington (1987) discovered that, despite the ad-

monition of the computer science establishment to construct programs top down, experts build them bottom-up. Perkins et al. (1989) described different problem-solving strategies used by novices. Murnane (1993) related programming to psychological theories, specifically Chomsky's notion of natural language acquisition and Piaget's theories of education.

Johnson-Laird (1975) contributed the idea of *mental models* to the study of people's competence in deductive reasoning, proposing that individuals reason by carrying out three fundamental steps:

1. They imagine a state of affairs in which the premises are true – i.e. they construct a mental model of them.
2. They formulate, if possible, an informative conclusion that is true in the model.
3. They check for an alternative model of the premises in which the putative conclusion is false. If there is no such model, then the conclusion is a valid inference from the premises.

Johnson-Laird & Steedman (1978) implemented the theory in a computer program that made deductions from singly-quantified assertions, and its predictions about the relative difficulty of such problems were strikingly confirmed: the greater the number of models that have to be constructed in order to draw the correct conclusion, the harder the task. Johnson-Laird concluded that comprehension is a process of constructing a mental model (Johnson-Laird 1981), and set out his theory in an influential book (Johnson-Laird 1983). Since then he has applied the idea to reasoning about Boolean circuitry (Bauer & Johnson-Laird 1993) and to reasoning in modal logic (Johnson-Laird & Bell 1997).

This paper does not seek to explain the difficulties faced by novices, neither does it propose a remedy. Instead we investigate a possible predictor of success in a first programming course – which, as pointed out by Simon et al. (2006), is the only effective measure of programming aptitude that is available in large-scale educational experiments.

Our predictor is based on a study of mental models of simple program operations. We find some regularity in the way that these models are used, even in students not previously exposed to programming or to programming languages. Our first results showed a surprising correlation between this regularity and scores in programming examinations.

In what follows we describe the test and its administration, and explore the results of six experiments.

2 The test

From experience it appears that there are two major hurdles which trip up novices in a first course on imperative programming.[1] In the order they are introduced in a conventional imperative-programming course they are:

1. assignment and sequence;
2. recursion / iteration;

There are other hurdles between and beyond these two, but it is at these that novices most obviously stumble. The second hurdle surprises nobody: recursion is conceptually difficult, and the proper treatment of iteration is mathematically complicated. Assignment and sequence, on the other hand, hardly look as if they should be hurdles at all: storage of/remembering information and doing one thing after another are part of everyday patterns of life and thought, and you might have expected (as at first do most teachers) that students' experience could be analogised into some kind of programming expertise. Not so: it is a real hurdle, which many fail to surmount, and it comes at the very beginning of most programming courses. We decided to investigate it.

Our first intention was to observe the mental models that students used when thinking about assignment instructions and short sequences of assignments. We did not use Johnson-Laird's theories of deductive reasoning directly, but we assumed that our subjects, in answering questions about a computer program, would necessarily use a mental model of the activity described by the program. We hoped to be able to find out what those models are, and to track changes over time.

We expected that our novices would initially display a wide range of mental models, but that as time went by the ones who were successfully learning to program would converge on a model that corresponds to the way that a program actually works in the language that they are being taught. So we planned to administer a test just after the students had begun to be taught about assignment and sequence, near the beginning of their course, then a second time to the same subjects after the topic had been dealt with, and then a third time just before the examination. We expected to correlate the results of these three administrations with each other and also with the subjects' marks in the official end-of-course examination.

Dehnadi devised a test, with questions as illustrated in figure 1. Each question gives a sample Java program, declaring two or three variables and executing up to three variable-to-variable assignment instructions. We had some prior notion, from our teaching experience, of the ways that a novice might understand the programs, and Dehnadi devised mental models accordingly (see table 1).

When a question includes more than one assignment, as in figure 2, things are more complicated. In addition to a model of assignment, subjects must use a model of statement juxtaposition. We have encountered only three, shown in table 2. Model S1 is the normal model of imperative programming. Models S2 and S3 are widely used by novices, though.

To accommodate the combination of assignment and juxtaposition models, there are many plausible answers for questions with multiple assignments. The answer sheet for such a question, shown in figure 3, is correspondingly complicated. Note that S1, the default model, is represented by a blank (so, for example, M1 in the first row means assignment model M1 with juxtaposition model S1). There are a number of multi-tick answers, because for example a respondent diligently using model M10 (equality) would tick the seventh box ($a = b = c = 5$) and would also write in additional outcomes of $a = b = c = 3$ and $a = b = c = 7$. In addition, some of the single-tick answers correspond ambiguously to several possible models.

In order to resolve that ambiguity in interpreting answers, we use the mark sheet shown in figure 4. This summarises all the information about a subject. Notionally we mark ambiguous replies in pencil, later inking the marks that maximise membership of a single column. We also mark the use of juxtaposition models. Once again, S1 is unmarked.

All the current test instruments are available from Dehnadi (2006). Students are allowed as long as they want to complete the test. We have found that they generally finish within 10 or 15 minutes.

[1] Functional and logical programming novices face different hurdles. We focus on imperative programming because it is the majority experience at present.

1. Read the following statements and tick the box next to the correct answer in the next column.	**The new values of a and b are:**	**Use this column for your rough notes please**

`int  a = 10;`
`int  b = 20;`
 	
`a = b;`
	☐ `a = 10` `b = 10`
	☐ `a = 30` `b = 20`
	☐ `a = 0` `b = 10`
	☐ `a = 20` `b = 20`
	☐ `a = 0` `b = 30`
	☐ `a = 10` `b = 20`
	☐ `a = 20` `b = 10`
	☐ `a = 20` `b = 0`
	☐ `a = 10` `b = 30`
	☐ `a = 30` `b = 0`
	
	Any other values for a and b:
	
	a = **b** =
	a = **b** =
	a = **b** =	

Figure 1: A test question with a single assignment

Table 1: Anticipated mental models of assignment `a=b`

M1. Value moves from right to left (a←b and b←0 – eighth line in figure 1).

M2. Value copied from right to left (a←b – fourth line of figure 1, and the 'correct' answer in Java).

M3. Value moves from left to right (b←a and a←0 – third line of figure 1).

M4. Value copied from left to right (b←a – first line of figure 1, a reversed version of the 'correct' answer).

M5. Right-hand value added to left (a←a+b – second line of figure 1).

M6. Right-hand value extracted and added to left (a←a+b and b←0 – tenth line of figure 1).

M7. Left-hand value added to right (b←a+b – ninth line of figure 1).

M8. Left-hand value extracted and added to right (b←a+b and a←0 – fifth line of figure 1).

M9. Nothing happens (sixth line of figure 1).

M10. A test of equality (first and fourth lines of figure 1).

M11. Variables swap values (seventh line in figure 1).

Table 2: Anticipated mental models of statement juxtaposition

S1. [sequence] The first assignment has its effect with initial values, then the second with the values produced by the first. (One effect is reported; the corresponding box is ticked.)

S2. [simultaneous, multiple] Each assignment takes effect using the initial values of variables. (All effects are reported; the boxes corresponding to each effect are ticked.)

S3. [simultaneous, single] Each assignment takes effect using the initial values of variables, but only the effects on the destination side are reported. (One overall effect is reported; the corresponding box is ticked.)

2.1 Assessment of consistency

The object of the test is to assess *consistency* in the use of assignment models. For that reason we attempt to maximise the number of marks in a single column by helpful interpretation of ambiguous answers. Then, in the row labelled C0, we total the marks in each column. A subject who has more than eight marks in a single column is assessed consistent at level C0. In an attempt to accommodate lower levels of consistency, we allow some variation across what we judge are cognitively similar models. Thus models M1 and M2, which differ only in whether the source variable becomes zero, are combined at level C1, and similarly other pairs of models. C2 combines four related models, and finally C3 distinguishes between the eight assignment-related models and the three others.

A subject who scores eight or more in one column of Cn, but not at Cn-1, is assessed consistent at Cn. We note that it is possible for a respondent to be classified as C3 by giving eight answers that comprise eight different models, so this is clearly an extremely weak measure of consistency. However, although in discussing our experiments we do mention levels C1-C3, the results that we present in this paper distinguish only students consistent at level C0.

Our current analysis pays little attention to which model of sequence is used by a consistent student, except to distinguish the group CM2, which has the correct Java models of assignment and sequence (see section 3.2 for a discussion of this group).

2.2 A test of invention

Notwithstanding our original intention of mapping the change in subjects' models, after seeing our initial results we decided to administer the test to subjects who have, ideally, no prior exposure to programming or to programming languages. We do expect, however, that they have encountered secondary-school algebra and are familiar with algebraic formulae.

We do not seek with this test to judge respondents according to the model they use. Rather, we test how they respond to questions which are about a mathematical system – a programming language – that they have not encountered before. No explanation is given about the meaning of the questions. The decision of the Java designers to use the equality sign '=' to indicate assignment,[2] as in column 1 of the questions, means that apart from the word '`int`' and the various semicolons, the formulae employed will be reasonably familiar to anybody who has encountered secondary-school algebra. That is, our respondents will have some notion of what `x=y` might mean, and will use that knowledge in guessing what box to tick in column 2. Incidentally, in the middle column the equality sign is used algebraically rather than Javishly.

Viewed strictly as algebra the programs are pretty nonsensical. But the question asks what the 'new values' of variables might be, and many subjects seem to respond by regarding the program as something that produces a change. Thus we are measuring the ability (and the inclination) to guess an answer. The test is unusual, we believe, in that to the complete novice the questions are strictly meaningless yet each has only a limited number of systematic answers, using the models listed in the tables.

[2] As computer scientists we normally excoriate this appalling design decision, but here we are able to benefit from it.

3 Experiments

3.1 Experiment 1: Barnet College and Middlesex University

The test was first administered by Dehnadi to about 30 students on a further-education programming course at Barnet College. As the initial intention was to track changes in subjects' mental models, Dehnadi decided to first investigate these models *before the students had received any programming teaching whatsoever* – that is, in week 0 of the course. In the early version of the test that was used for this experiment, two models – M7 and M8 – were accidentally omitted from the questions and the analysis; assessment of consistency was subjective; previous programming experience, education, age and sex weren't recorded; and the overall assessment was into three categories (consistent, inconsistent and blank). Dehnadi interviewed half of the students before admission, and taught them all. It was his impression (though in this experiment the questionnaire did not record the necessary details) that few if any had any previous contact with programming, and that all had enough school mathematics to make the equality sign familiar.

The same test was then administered to about 30 students in the first-year programming course at Middlesex University, once again before they had received any programming teaching. They were mostly male, aged about 18-20, from the middle range of educational attainment in the UK. Dehnadi tutored them but did not teach the course. It was his impression (again without any recorded evidence) that they had had little or no contact with programming but had received school mathematics teaching. The data from the second Middlesex experiment (experiment 3, section 3.3) suggests that this may be a mistaken assumption.

This experiment is reported in Dehnadi (2006), so here we will briefly summarise its findings. Because the test asked subjects about the effects of simple programs, at a stage when they hadn't encountered any programs at all, we did not expect that they would be using the models of assignment and juxtaposition that they would later be taught. We discovered instead a subdivision into three groups in the first administration of the test.

- 27 subjects (44%) used the same models for all, or almost all, of the questions. We call this the ***consistent*** group (C).
- 24 subjects (39%) used different models for different questions. We call this the the ***inconsistent*** group (I).
- The remaining five subjects (8%) failed to answer all or almost all of the questions. We call this the ***blank*** group (B).

We did not interview our subjects to determine anything about their group membership, so we do not know whether students chose consciously or unconsciously to follow one strategy or another, nor how conscious choices (if any) were motivated, nor what any particular choice meant to a subject who made it.

For reasons addressed in Dehnadi (2006), we decided to investigate the correlation between respondents' categorisation (C/I/B) in the consistency test and their marks in the second in-course exam.

The pass/fail statistics for this experiment are shown in table 3. Overall the failure rate was 37%, but in the consistent group it was only 11%, in the inconsistent group it was 65%, and in the blank group

7. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 5;`
`int b = 3;`
`int c = 7;`
 	
`a = c;`
`b = a;`
`c = b;`
	The new values of a and b and c are:
	
	☐ `a = 3 b = 5 c = 5`
	☐ `a = 3 b = 3 c = 3`
	☐ `a = 12 b = 14 c = 22`
	☐ `a = 8 b = 15 c = 12`
	☐ `a = 7 b = 7 c = 7`
	☐ `a = 5 b = 3 c = 7`
	☐ `a = 5 b = 5 c = 5`
	☐ `a = 7 b = 5 c = 3`
	☐ `a = 3 b = 7 c = 5`
	☐ `a = 12 b = 8 c = 10`
	☐ `a = 10 b = 8 c = 12`
	☐ `a = 0 b = 0 c = 7`
	☐ `a = 0 b = 0 c = 15`
	☐ `a = 3 b = 12 c = 0`
	☐ `a = 3 b = 5 c = 7`
	
	Any other values for a and b and c :
	
	`a = b = c =`
	`a = b = c =`
	`a = b = c =`	

Figure 2: A test question with three assignments

Question	Answer(s)	Model(s)

7.
 	
`int a = 5;`
`int b = 3;`
`int c = 7;`
 	
`a = c;`
`b = a;`
`c = b;`
	`a = 0 b = 0 c = 7`	M1

	`a = 7 b = 5 c = 3`	M1 S3 / M2 S3 / M11 S3
	`a = 7 b = 7 c = 7`	M2
	`a = 3 b = 5 c = 5`	M3, M4
	`a = 3 b = 7 c = 5`	M3 S3 / M4 S3
	`a = 12 b = 14 c = 22`	M5
	`a = 12 b = 8 c = 10`	M5 S3 / M6 S3
	`a = 0 b = 0 c = 15`	M6
	`a = 8 b = 15 c = 12`	M7
	`a = 10 b = 8 c = 12`	M7 S3 / M8 S3
	`a = 3 b = 12 c = 0`	M8
	`a = 5 b = 3 c = 7`	M9
	`a = 3 b = 5 c = 7`	M11

	`a = 5 b = 5 c = 5`
	`a = 3 b = 3 c = 3`
	`a = 7 b = 7 c = 7`
		M10

	`a = 7 b = 3 c = 0`
	`a = 0 b = 5 c = 7`
	`a = 5 b = 3 c = 3`
		M1 S2

	`a = 7 b = 3 c = 7`
	`a = 5 b = 5 c = 7`
	`a = 5 b = 3 c = 3`
		M2 S2

	`a = 0 b = 3 c = 5`
	`a = 3 b = 0 c = 7`
	`a = 5 b = 7 c = 0`
		M3 S2

	`a = 5 b = 3 c = 5`
	`a = 3 b = 3 c = 7`
	`a = 5 b = 7 c = 7`
		M4 S2

	`a = 12 b = 3 c = 7`
	`a = 5 b = 8 c = 7`
	`b = 3 c = 10`
		M5 S2

	`a = 12 b = 3 c = 0`
	`a = 0 b = 8 c = 7`
	`a = 5 b = 0 c = 10`
		M6 S2

	`a = 5 b = 3 c = 12`
	`a = 8 b = 3 c = 7`
	`a = 5 b = 10 c = 7`
		M7 S2

	`a = 0 b = 3 c = 12`
	`a = 8 b = 0 c = 7`
	`a = 5 b = 10 c = 0`
		M8 S2

	`a = 7 b = 3 c = 5`
	`a = 3 b = 5 c = 7`
	`a = 5 b = 7 c = 3`
		M11 S2

Figure 3: Answer sheet, with ambiguous assessments

Questions	Assignment								No effect	Equal sign	Swap values	Remarks (including participants' working notes)
	Assign-to-left		Assign-to-right		Add-Assign-to-left		Add-Assign-to-right		Values don't change (M9) /S2	Assign means equal (M10) /S2	Swap values (M11) /S2 /S3	
	Lose-value (M1) /S2 /S3	Keep-value (M2) /S2 /S3	Lose-value (M3) /S2 /S3	Keep-value (M4) /S2 /S3	Keep-value (M5) /S2 /S3	Lose-value (M6) /S2 /S3	Keep-value (M7) /S2 /S3	Lose-value (M8) /S2 /S3				
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
C0												
C1												
C2												
C3												

Figure 4: Mark sheet allowing for judgement of level of consistency

Table 3: Experiment 1: Consistency against pass/fail results in second in-course exam

	Pass	Fail	
Consistent	23	3	26
Inconsistent	8	15	23
Blank	6	4	10
	37	22	59

40%. The correlation is significant by the χ^2 test[3] ($p \leq 0.001$).

Pass/fail statistics can be distorted relatively easily by moving the mark boundary. An analysis of variance[4] is fairer: it shows that the difference between the groups is significant ($p < 0.001$), with $R^2 = 0.255$, that the C group is significantly different from the I group ($p < 0.001$) and from the B group ($p \leq 0.016$), but that the I and B groups aren't significantly different ($p \leq 0.445$).

We might consider making an argument that an R^2 value of 0.255 is not to be sneezed at in an educational context, where success and failure have so many causes, but it does not trumpet success, and in the experiments below we restrict our discussion to the simpler χ^2 test of significance. We note, however, that the ANOVA significance measurements do not in any instance contradict those which we report.

[3]The χ^2 test is a standard statistical tool applied to data counted into categories, as we have counted students into consistent/inconsistent/blank and into pass/fail. Where the numbers in the categories appear to be dependent, for example membership of the consistent group appears related to membership of the pass group, the test helps to decide whether this apparent dependence is likely to be reflected in the whole population. The figure generally reported is p, a measure of the likelihood that the apparent dependence is coincidental. The accepted standard to which we are working is that a result is significant if $p < 0.05$; in loose terms, there is less than 5% chance that the observed effect is coincidental.

[4]An analysis of variance, ANOVA, is a standard statistical tool that determines whether there is a significant difference between the means of two or more groups in a population. The figures generally reported are R^2, a measure of the strength of the correlation between group membership and the differing scores, and p, a measure of the significance of the difference. The accepted standard to which we are working is that a difference is significant if $p < 0.05$; in loose terms, there is less than 5% chance that the observed difference is coincidental. However, even a strong significance does not tell us much if the correlation is weak.

Table 4: Expt 1: Pass/fail statistics for test 1 grouping and test 2 correctness

Test 1 (wk 0)	Test 2 (wk 3)	Pass	Fail	
consistent	correct	20	0	20
blank	correct	3	0	3
inconsistent	correct	3	2	5
consistent	incorrect	3	3	6
blank	incorrect	3	4	7
inconsistent	incorrect	5	13	18
		37	22	59

3.1.1 Week 3 test

In an aside to experiment 1, the same subjects were given the same test again in week 3 of the course, after assignment and sequence had been taught. There were no blank returns; all but two of the consistent group remained consistent; half of the other groups became consistent. In this test we recorded not only consistency but also correctness (use of M2 and S1 models). Table 4 shows that 93% of those who scored correctly in this test passed the second in-course exam, against only 35% of those who scored incorrectly. An analysis of variance shows significance as before $p < 0.001$) and the correct/incorrect classification explains more of the variance ($R^2 = 0.442$).

3.2 Experiment 2: The University of Newcastle

The second experiment was carried out by Simon at the University of Newcastle, Australia. During this experiment the test instruments were refined and improved, and questions about previous programming experience, previous programming courses, age and sex were added to the questionnaire.

The test was administered once, before the course began. There were 90 subjects, of whom 19 withdrew or failed to attend the examination, leaving 71 for analysis.

3.2.1 Test results

43 subjects were assessed as C0 (eight questions or more with a single model), four C1 (eight questions or more with two related models), one C2 (four models),

Table 5: Experiment 2: Consistency against pass/fail results

	Pass	Fail	
consistent (CM2)	24	3	27
consistent (C0)	10	6	16
other	13	15	28
	47	24	71

Table 6: Experiment 2: Consistency against pass/fail results, split by previous programming experience

Programmed before = yes

	Pass	Fail	
consistent (CM2)	20	2	22
consistent (C0)	6	5	11
other	3	10	13
	29	17	46

Programmed before = no

	Pass	Fail	
consistent (CM2)	2	1	3
consistent (C0)	2	0	2
other	9	5	14
	13	6	19

five C3 (more than four models M1-M8), 17 inconsistent and one blank. As in experiment 1, C0 was by far the largest group. The most striking feature of the data, however, is that in the C0 group 33 subjects out of 43 used the correct models (M2, S1) of assignment and juxtaposition before the course started. We shall henceforth refer to this group as CM2. All but four of the CM2 group indicated that they had had previous programming experience.

3.2.2 Assessment of programming skill

We had access to all the course marks of the subjects: two practical tests, two assignments, an examination and a mark combining all of those. In our analysis we used the combined mark. As well as a numerical mark, the students were given grades of FF (fail), P (pass), C (credit), D (distinction) and HD (high distinction).

3.2.3 Data analysis

Correlating grades with fine consistency grading gives a table too sparse to analyse: combining columns P, C, D and HD into 'Pass' and rendering FF as 'Fail', and separating out those who appear to have the correct models of assignment and sequence before starting the course, gives table 5. There is χ^2 significance overall ($p \leq 0.001$), but this disappears if the CM2 row is left out of the calculation ($p \leq 0.30$).

But simply to ignore the CM2 group is not enough: they are the successful fraction of those who have attempted to learn to program before. What about the unsuccessful fraction? If the result of the week 3 test in the first experiment (section 3.1.1) is indicative, those who have failed to learn the correct models for assignment and sequence might be expected to fail at this second attempt.

If we take into account subjects' response to the question about prior programming experience, we derive the two tables in table 6 (67 subjects, because four failed to report either way). The first table is significant ($p \leq 0.001$), and that shows the same result as in table 4: those who have tried to learn to program but failed to learn the models of assignment and sequence aren't likely to do well. The second table cannot be subjected to χ^2 analysis, because numbers are too small.

Table 7: Experiment 3: Consistency against pass/fail results, split by previous programming experience

Programmed before = yes

	Pass	Fail	
consistent (CM2)	8	0	8
consistent (C0)	6	2	8
other	11	7	18
	25	9	34

Programmed before = no

	Pass	Fail	
consistent (C0)	8	4	12
other	18	7	25
	26	11	37

We get similar results if we further refine the data by looking at the answer to the question about prior instruction in programming: out of the four subgroups, the only one to show significance is those who have programmed and have taken a prior course (31 subjects, $p \leq 0.044$), but the other groups are too small for χ^2 analysis. Intriguingly, but insignificantly because of the size of the sample, the group who report prior experience in programming but no prior instruction split perfectly: all four inconsistent fail, all nine consistent pass (including seven CM2).

At first sight the discovery that two-thirds of the subjects have prior programming experience, leaving too few inexperienced subjects for proper analysis, might suggest that there is nothing to learn from this experiment. Certainly we can't use it to support our hypothesis that consistency in our test predicts success in the course. But we can see, in measuring those with prior experience, that mental models of assignment and sequence do seem to have something to do with the matter.

3.3 Experiment 3: Middlesex University

Dehnadi administered the refined test instrument to 118 subjects at Middlesex University in September 2006. The test was administered once, before the course began.

3.3.1 Test results

41 subjects (35%) were rated C0; three (2.5%) were C1; ten (8.5%) were C2; eighteen (15%) were C3; 28 (24%) were inconsistent and 18 (15%) blank. Eleven of the C0 group (27% of the group, 9% of the total) already had the correct models of assignment and juxtaposition (CM2).

As in the first experiment we have analysed the results of the second, more technical, in-course examination. This examination was taken by 75 students, the remainder having withdrawn from the course. Table 7 sets out the results in the same form as the second experiment.

These results seem a blow against our hypothesis that consistency in the test predicts novices' success in the course. There is no significance in the results of those without experience (indeed, the failure rates are almost identical for the two groups). The other group does show significance in ANOVA ($p < 0.0005$, $R^2 = 0.496$), which disappears when the CM2 group is removed, once again supporting the hypothesis that mastery of assignment and sequence is predictive of success.

Table 8: Experiment 4: Consistency against pass/fail results, split by previous programming experience

Programmed before = yes

	Pass	Fail	
consistent (CM2)	8	0	8
consistent (C0)	7	1	8
other	0	1	1
	15	2	17

Programmed before = no

	Pass	Fail	
consistent (CM2)	2	2	4
consistent (C0)	22	1	23
other	7	7	14
	31	10	41

3.4 Experiment 4: University of Sheffield

The refined test was administered to 58 subjects by Peter Rockett (p.rockett@sheffield.ac.uk). Twelve students were CM2, 31 C0, 2 C3, 12 inconsistent and 1 blank. Unlike other experiments, which were all in computer science departments, this test was carried out in an electrical engineering department.

Pass/fail data is shown in table 8. One notable feature is the low failure rate overall (21%). Because of the low failure rates we cannot rely on χ^2 significance ($p < 0.014$ for the yes group, $p < 0.003$ for the no group): an ANOVA analysis shows significance ($p < 0.055$ and $p < 0.007$ respectively), but with low explanation of variance ($R^2 = 0.339$ and 0.233 respectively).

3.5 Anomalous experiments

The first experimental result, without statistical analysis, was publicised on the web by Bornat and Dehnadi, and in that publication they made strong claims for its importance. In response to this and to Dehnadi's subsequent paper (Dehnadi 2006), several people volunteered to try the test in their own institutions.

At the University of York, UK, Berbatov (2007) tested 104 subjects, of whom all but 14 used the correct models before his course. No significant differences were found between consistency groups.

At the University of Aarhus, Caspersen et al. (2007) tested 142 subjects, of whom 124 rated C0, almost all of them CM2, and only six failed the examination. No significant differences were found between consistency groups.

At the Royal School of Signals, UK, Wray (2007) tested 19 subjects, five months after the end of their programming course. All but one scored perfectly CM2 (the one got one question wrong).

Each of these experiments essentially investigates subjects who have experience of programming and can answer the questions in our test with knowledge of the programming language involved. These are not the circumstances for which the test was devised, in which only a minority have prior instruction or experience. We do not, therefore, feel that it is helpful to examine their data in more detail.

4 Summary and discussion of significant results

In experiment 1, involving 60 subjects at Barnet College and Middlesex University, we found significant differences over the three groups C, B, and I, over the two groups C and B, and over the two groups C and I. We also found that our test, administered after assignment and sequence had been taught, showed a strongly significant difference between those who used the correct models and those who did not.

In experiment 2, involving 71 subjects at the University of Newcastle, we discovered that most had previous programming experience. There was no significant effect of consistency in the 19 who reported no prior experience; there was in the 46 who did claim experience, essentially due to the CM2 subjects who had the correct models before the course began.

In experiment 3, involving 75 students at Middlesex University, we found no significant differences.

In experiment 4, involving 58 students at the University of Sheffield, we found a significant effect of consistency in those with no programming experience.

In experiments 5 and 6 all or almost all of the participants had the correct models before the course commenced, and hence our test was irrelevant.

Experiment 7 tested subjects after the course had finished, revealing a commendable efficiency in programming instruction but also making prediction impossible.

In experiments 1 and 3 we explored correlations between consistency and students' performance on an assessment item which was a plausible surrogate for a final course examination, while the other experiments used final results in the course.

Apart from experiment 1, for which the information is not available, we have used the same groupings (programming experience yes/no, consistency CM2/C0/other) in each of the analyses.

5 Conclusions and further work

When we began this work we had high hopes that we had found a test that could be used as an admissions filter to reduce the regrettable waste of human effort and enthusiasm caused by high failure rates in universities' first programming courses. We can see from the experiments reported above that our test doesn't work if the intake is already experienced, and in experiment 3 didn't work at all. We cannot claim to be separating the programming goats from the non-programming sheep: experiment 3 demolishes the notion that consistent subjects will for the most part learn well, and others for the most part won't. And even in the most encouraging of our results, we find a 50% success rate in those who don't score C0 or CM2.

None the less, some of our results indicate that there may be something going on with consistency. There is a case for continuing our investigations.

In the meantime we present this work as a case study in good science. Having some preliminary results that appeared extremely promising, we and colleagues have refined the test instrument, conducted further experiments, and applied the appropriate analysis. It is unfortunate that the outcome does not live up to the initial promise, but it has not quite closed the door on our explorations.

As well as continuing to investigate the phenomenon of consistency, we intend in the future to interview subjects, asking them why they answer the questions as they do. In addition, we shall investigate whether there is any significant difference between the way male and female subjects answer the questions.

Acknowledgements

We are grateful to Middlesex University, Barnet College, the University of Newcastle and the University of Sheffield for giving us permission to test their students, to them and to the subjects themselves for al-

lowing us access to their examination results, and to our collaborators for administering the questionnaire.

We have benefited enormously from advice from the statistical consultancy group at Middlesex University: David Jarrett, Jeff Evans, Gary Hearne and Anne Humbert; and from Maureen Townley-Jones at the University of Newcastle. Without their advice we would not have discovered what our data contained nor been able to give any coherent account of our results.

References

Adelson, B. & Soloway, E. (1985), 'The role of domain experience in software design', *IEEE Transactions on Software Engineering* **11**(November), 1351–1360.

Bauer, M. & Johnson-Laird, P. (1993), How diagrams can improve reasoning: Mental models and the difficult cases of disjunction and negation, *in* 'Proceedings of the Annual Conference of Cognitive Science', Lawrence Erlbaum Associates, Boulder, Colorado.

Berbatov, D. (2007). private communication.

Bonar, J. & Soloway, E. (1983), Uncovering principles of novice programming., *in* '10th ACM POPL', pp. 10–13.

Bonar, J. & Soloway, E. (1985), 'Pre-Programming Knowledge: A Major Source of Misconceptions in Novice Programmers', *Human-Computer Interaction* **1**(2), 133–161.

Bornat, R. (1986), *Programming from First Principles*, Prentice/Hall International.

Canas, J. J., Bajo, M. T. & Gonzalvo, P. (1994), 'Mental models and computer programming', *Journal of Human-Computer Studies* **40**(5), 795–811.

Caspersen, M. E., Bennedsen, J. & Larsen, K. D. (2007), Mental Models and Programming Aptitude, *in* 'Proceedings of ITICSE 2007'.

Dehnadi, S. (2006), Testing Programming Aptitude, *in* P. Romero, J. Good, E. A. Chaparro & S. Bryant, eds, 'Proceedings of the PPIG 18th Annual Workshop', pp. 22–37.
URL: *http://www.ppig.org/papers/18th-dehnadi.pdf*

du Boulay, J. B. H. (1986), 'Some difficulties of learning to program', *Journal of Educational Computing Research* **2**(1), 57–73.

Green, T. (1997), Cognitive Approach to Software Comprehension: Results, Gaps and Introduction, *in* 'Workshop on Experimental Psychology in Software Comprehension Studies', University of Limerick.

Johnson-Laird, P. (1975), Models of deduction, *in* R. Falmagne, ed., 'Reasoning: Representation and Process', Erlbaum, Springdale, NJ.

Johnson-Laird, P. (1981), 'Comprehension as the construction of mental models', *Philosophical Transactions of the Royal Society, Series B* **295**, 353–374.

Johnson-Laird, P. (1983), *Mental Models*, Cambridge University Press, Cambridge.

Johnson-Laird, P. & Bell, V. (1997), A model theory of modal reasoning, *in* 'Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society', pp. 349–353.

Johnson-Laird, P. & Steedman, M. (1978), 'The psychology of syllogisms', *Cognitive Psychology* **10**, 64–99.

Kölling, M., Quig, B., Patterson, A. & Rosenberg, J. (2003), 'The BlueJ system and its pedagogy', *Journal of Computer Science Education* **13**(4), 249–268.
URL: *http://www.bluej.org/papers/2003-12-CSEd-bluej.pdf*

Mayer, R. E. (1992), *Thinking, Problem Solving, Cognition*, 2 edn, W. H. Freeman and Company Second Edition ISBN 0716722151, New York.

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, *in* 'Working group reports from ITiCSE on Innovation and technology in computer science education', ACM Press, Canterbury, UK.

Murnane, J. (1993), 'The Psychology of Computer Languages For Introductory Programming Courses', *New Ideas in Psychology* **11**(2), 213–228.

Papert, S. (1980), *Mindstorms: Children, Computers, and Powerful Ideas*, Basic Books, New York, NY, USA.

Pennington, N. (1987), 'Stimulus structures and mental representations in expert comprehension of computer programs', *Cognitive psychology* **19**, 295–341.

Perkins, D., Hancock, C., Hobbs, R., Martin, F. & Simmons, R. (1989), Conditions of Learning in Novice Programmers, *in* E. S. Spohrer & J. C., eds, 'Studying the Novice Programmer ', Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 261–279.

Putnam, R. T., Sleeman, D., Baxter, J. A. & Kuspa, L. K. (1986), 'A Summary of Misconceptions of High School Basic Programmers', *Journal of Educational Computing Research* **2**(4).

Simon, Fincher, S., Robins, A., Baker, B., Box, I., Cutts, Q., de Raadt, M., Haden, P., Hamer, J., Hamilton, M., Lister, R., Petre, M., Sutton, K., Tolhurst, D. & Tutty, J. (2006), Predictors of success in a first programming course, *in* 'Proc. Eighth Australasian Computing Education Conference (ACE2006)', ACS, pp. 189–196.

Soloway, E. & Spohrer, J. C. (1989), Some Difficulties of Learning to Program , *in* E. Soloway & J. C. Spohrer, eds, 'Studying the Novice Programmer', Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 283–299.

Turner, D. A. (1985), Miranda: a non-strict functional language with polymorphic types, *in* 'Proc. of a conference on Functional programming languages and computer architecture', Springer-Verlag New York, Inc., New York, NY, USA, pp. 1–16.

van Someren, M. W. (1990), 'What's wrong? Understanding beginners' problems with Prolog', *Instructional Science* **19**(4/5), 256–282.

Vaughan-Nichols, S. J. (2003), 'The Battle over the Universal Java IDE', *Computer* **36**(4), 21–23.

Wray, S. (2007), SQ Minus EQ can Predict Programming Aptitude, *in* 'Proceedings of the PPIG 19th Annual Workshop'.
URL: *http://www.ppig.org/papers/19th-Wray.pdf*