Meta-analysis of the effect of consistency on success in early learning of programming

Saeed Dehnadi, Richard Bornat, Ray Adams

S.Dehnadi@mdx.ac.uk, R.Bornat@mdx.ac.uk, R.G.Adams@mdx.ac.uk
School of Engineering and Information Sciences, Middlesex University

Abstract. A test was designed that apparently examined a student's knowledge of assignment and sequence before a first course in programming but in fact was designed to capture their reasoning strategies. An experiment found two distinct populations of students: one could build and consistently apply a mental model of program execution; the other appeared either unable to build a model or to apply one consistently. The first group performed very much better in their end-of-course examination than the second in terms of success or failure. The test does not very accurately predict levels of performance, but by combining the result of six replications of the experiment, five in UK and one in Australia. we show that consistency does have a strong effect on success in early learning of programming. Background programming experience, on the other hand, has little or no effect.

1 Introduction

Programming is hard to learn. The search for predictors of programming ability has produced no significant results. The problem is international and longstanding.

It is a commonplace that some students find programming extremely easy to learn whilst others find it almost impossible. Dehnadi observed that some novices confronted by simple programming exercises gave rational but incorrect answers. Their answers were mechanically plausible: for example, in Java assigning the value of a variable from left to right rather than right to left, or moving a value in an assignment rather than copying.

This suggested to us that some novices may have been equipped with some abilities before the course started. In an experiment Dehnadi (2006) administered a test made up of questions about assignment and sequence programs. The test was administered before the first week of an introductory programming course, without giving any explanation of what the questions were about. Almost all participants gave a full response. About half used a rational model which they applied consistently to answer most or all of the questions. The other half did not seem to use a recognisable model or appeared to use several models. The consistent subgroup had an 85% pass rate in the course examination, and the rest a 36% pass rate.

There were some deficiencies in that experiment, and much of the research community was sceptical of his result, particularly when two experiments (Caspersen et al., 2007; Wray, 2007) appeared to refute it, and a review of several others (Bornat et al., 2008) concluded that the test does not predict very much of the variance in levels of performance in the course examination. The first 'refutation' was an experiment conducted under what seem to be unusual conditions, and the second showed only that the test was not psychometric, in that it is ineffective in a population of programmers. The review did not distinguish clearly between evidence for the presence of an effect and measurements of its strength.

This paper provides evidence for the claim that consistency affects performance, by a meta-analysis of six replications of an improved version of the experiment. It shows that consistency is not simply the result of background programming experience, and that by contrast such experience has little or no effect on success.

2 Previous Work

The search for predictors of success in learning to program has turned up very little. Cross (1970), Mayer and Stalnaker (1968) and Wolfe (1971) tried to use occupational aptitude tests

to predict successful candidates for employment in the software industry. McCoy and Burton (1988) said good mathematical ability was a success factor in beginners' programming. Wilson and Shrock (2001) found three predictive factors: comfort level, mathematical skill, and attribution to luck for success or failure. Tukiainen and Mönkkönen (2002) evaluated a programming aptitude test devised by Huoman (1986), and found that an initial correlation between test scores and success disappeared when programming background was taken into account. Beise et al. (2003) found that neither sex nor age is a good predictor of success in the first programming class. Rountree et al. (2004) reveal that the students most likely to succeed are those who are expecting to get an 'A' grade and are willing to say so. In a multi-national project Lister et al. (2004), Fincher et al. (2005) and Raadt et al. (2005) opined that incapability of students in entry-level programming is due to lack of general problem-solving ability. Simon et al. (2006a), Tolhurst et al. (2006) and Simon et al. (2006b) followed up the project and came to similar conclusions. Bennedsen and Caspersen (2006) found no correlation between cognitive development and results in a model-based introductory programming course; and they found, in a three-year longitudinal study (Bennedssen and Caspersen, 2008), that general abstraction ability was not a predictor for success in learning computer science.

Johnson-Laird's notion of 'mental model' (Wason and Johnson-Laird, 1968) lies behind this study, and much other research on programming learning. Kessler and Anderson (1986) and Mayer (1981) all stressed the significance of mental models. du Boulay (1986) catalogued the difficulties that novices experienced and classified them according to mental models. Fix et al. (1993) differentiated mental models of novices and experts. Perkins and Simmons (1988) described novice learners' according to their problem-solving strategies as "stoppers", and "movers". Mayer (1992) described existing knowledge as a "cognitive framework" and how new information is connected to existing knowledge. Dyck and Mayer (1985) emphasised the necessity for a clear understanding of the underlying virtual machine in novice's learning process. Putnam et al. (1986) studied the impact of novices' misconceptions of the capabilities of computers. van Someren (1990) found that mechanical understanding of the way the language implementations is the key to success.

Our research is based on the observation of mental models of rational misconceptions. Spohrer and Soloway (1986) used novices' mistakes and misconceptions to hypothesise what they were thinking as they programmed. They found that just a few types of bug cover almost all those that occur in novices' programs. They hypothesised that a "programming goals/subgoals/plans" theory is used widely by novices to break down the complexities of program applications. In (Soloway and Spohrer, 1989) they studied novices' errors and explained how their background knowledge interfered with their learning process and caused many of their misconceptions.

Shneidermann (1985) investigated different uses of variables and addressed issues such as assignment statements, the difference between a variable and the value stored in it, printing, using, changing, and comparing the value stored in a variable. He distinguished counting from summing from other uses of a variable.

Samurcay (1985) looked at different ways that variables are assigned values: assignment of a constant value (`a=3`); assignment of a calculated value (`a=3*b`); duplication (`a=b`); and accumulation (`x=x+1`). He described how each of these techniques can be used within two different contexts: external - where variables are inputs to or outputs of the program, under control of the program user; and internal - where variables are necessary only for the solution of the problem and are controlled by the programmer. He claimed that internal variables are harder for novices to process by illustrating three uses of variables in loops: update; test; and initialisation. Novices found more difficulties with initialisation than updating or testing.

du Boulay (1986) identified misconceptions about variables. He illustrated that the analogy of a box or drawer with a label on it may lead learners to build a mental model that a variable can hold more than one value at a time; when a variable is used to hold a running total, the

fact that learners forget to initialise the total to zero comes from the fact that a box which, even when empty, contains something. Comparing a variable to a box triggers learners' minds to their existing mental model of boxes, rather than building a correct model of variable. He also noted students' misconceptions when assigning one variable to another: for example `x=y` may be viewed as linking the second variable to the first, and therefore a change in `x` results in a change in `y`. Novices may not understand that a value stays in a variable until it is explicitly changed, the contents of memory are erased or the machine is switched off. He concluded that novices with a lack of mechanical understanding build a inappropriate mental model.

Perkins et al. (1986) explained that novices have misconceptions about the names of variables, even though they know in principle that the choice of variable names is theirs. Having a weak mental model of the virtual machine, students can have the notion that a computer program knows that the highest input values should go into variables with names like `largest`. They noted that people commonly fail to notice inconsistencies in their mental models, and often when inconsistencies are brought to their attention cannot notice their importance.

Kessler and Anderson (1986) studied students' errors in learning recursion and iteration. They observed that students who had poor mental models of the mechanical processes of recursion and iteration in programming adapted poor learning strategies. They emphasised that novices' appropriate mental models of such techniques should be developed prior to engaging them in any implementation task.

Mayer (1981) believed that experts are able to think semantically and demonstrates four areas of differences between them and novices in computer programming: semantic knowledge, syntactic knowledge, ability in task management; and having an effective mental model of the virtual machine. Novices are at the beginning of their mental model development; syntactic knowledge is hard for them because it's difficult to catch grammatical mistakes; they are inexperienced in problem decomposition; and lack of strategic knowledge forces them to lean on low-level plans during problem solving. Mayer describes existing knowledge as a "cognitive framework", and "meaningful learning" as a process by which new information is connected to existing knowledge. In (Mayer, 1992) he stated that people who know how their programs work do better than those who do not, and mental models are crucial to learning and understanding programming.

3 Initial experiment

Dehnadi's experience of teaching led him to believe that novices bring patterns of reasoning to the study of programming: some of them appear to use rational mechanisms, distinct from those taught in the course, to explain program behaviour. He designed a test (Dehnadi, 2006) to see what would happen when students were confronted with programming problems before they had been given any explanation of the mechanisms actually involved in program execution. His questions each gave a program fragment in Java, declaring two or three variables and executing

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

Fig. 1. The first question in the test, a single assignment

one, two or three variable-to-variable assignment instructions, as illustrated in figure 1. The student was asked to predict the effect of the program on its variables and to choose their answer/s from a multiple-choice list of the alternative answers. There was no explanation of the meaning of the questions or the equality "=" sign that Java uses to indicate assignment. Except for the word "int" and the semicolons in the first column, the formulae employed would have looked like school algebra, but when the question asked about the "new values" of variables it hinted that the program produces a change.

Dehnadi had a prior notion of the ways that a novice might understand the programs, and prepared a list of recognised mental models. The models of assignment that he expected subjects to use were M1-M8 and M11 from table 2. Questions with more than one assignment required a model of composition of assignments as well as a model of single assignment. The three recognised models of sequential composition are shown in table 3.

The test was administered to 30 students on a further-education programming course at Barnet College and 31 students in the first-year programming course at Middlesex University. Two dropped out before the examination. No information was recorded about earlier education, programming experience, age or sex. It was believed that few had any previous contact with programming and that all had enough school mathematics to make the equality sign familiar.

Despite the lack of explanation of the questions, most students gave a more or less full response. About half (the consistent group C) gave answers which corresponded to a single mental model of assignment in most or all questions; about as many (the inconsistent group I) gave answers which corresponded to different models in different questions or didn't use recognisable models; and a small number (the blank group B) answered few or no questions.

Table 1. Consistency (week 0) and second quiz pass/fail

	Pass	**Fail**	**Total**
C	22	4	26
I	8	15	23
B	4	6	10
Total	34	25	59
$\chi^2 = 13.944,\ df = 2,\ p < 0.001$ **highly significant**			

When the result of the test was correlated with the result of in-course examinations it was found that in the later examination which examined technical skill more thoroughly, the consistent (C) group had an 85% pass rate, but the others (I and B together) only 36% (table 1). This was a large effect, compared to earlier research on predictors, and a χ^2 test showed that it was significant. The overall pass rate was 58% so the test might be seen as a predictor, but the false negative rate – the fraction of the I and B groups who managed to pass the examination – was 36%, which was less convincing. Indeed when the data was re-analysed in (Bornat et al., 2008), it was found that the test explained only about 26% of the variance in examination results.

This experiment was over-hyped by Dehnadi and Bornat (2006), and perhaps as a result attracted a good deal of critical attention.

3.1 Two apparent refutations and an analysis

An experiment at Aarhus University, Denmark by Caspersen et al. (2007) used Dehnadi's test but found no effect of consistency on success. Wray (2007) at the Royal School of Signals, UK used Dehnadi's test and found no effect of consistency on success but found a strong effect of Baron-Cohen's SQ and EQ measures (Baron-Cohen et al., 2001, 2003).

In the Aarhus experiment 124 subjects (87%) were assessed as C, 18 (13%) as notC, and the average failure rate in the course was 4%. The high proportion of consistent subjects and the low failure rate makes this experiment very different from all but one of the experiments included in our meta-analysis below, and the very low failure rate may have constrained statistical analysis. The researchers were unable to correlate the test with success/failure statistics, and their decision to look for correlations with graded data would in any case have weakened any signs of the effect of consistency. Despite the difficulties of dealing with such a population, our meta-analysis includes an experiment at the University of York which on the face of it has an intake similar to that in the Aarhus experiment.

Wray's experiment refutes the regrettable claim by Dehnadi and Bornat (2006) that Dehnadi's test divides novices into programming sheep and non-programming goats. The test should not be considered psychometric since a normal programming course attempts to train novices to pass it. Wray used the test five months *after* the course ended. It could not be expected to deliver a meaningful result in those circumstances.

An analysis by Bornat et al. (2008) looked individually at the initial experiment and some of the experiments discussed below and concluded that the test is not a good predictor of performance, explaining only about a quarter of the variance of examination results: that is, consistent subjects don't all do better than all inconsistent subjects, and the overlap is considerable. The false negative rate in the experiments, which might have been caused by faulty positioning of a pass mark, is instead a sign of a deficiency in the test. Once again the claim about sheep and goats is resoundingly rejected. This analysis shows that the effect is not overwhelming, and indeed not strong enough to be used as an admissions test, but it does not disprove its existence.

4 Improved experimental protocol

In response to criticism of the initial experiment some improvements were made to the test. Most importantly, the judgement of consistency was made explicit and repeatable. The number of models of assignment recognised was expanded to the eleven shown in table 2. Answers to single-assignment questions still correspond to a single tick in all cases except M10, which requires a student to tick all answers which make all variables equal – the fourth and fifth answer in figure 1, for example. The models of composition were made explicit as shown in table 3: note that model S2 generates multiple ticks in all multiple-assignment questions. Answer sheets were produced, illustrated in figure 4, which identified the mental models apparently used in particular answers or patterns of answers; in particular this exposed ambiguity in some answers to multiple-assignment questions.

Questions about background were added to the questionnaire. We asked about age and sex, about previous programming experience (yes/no and if yes, what languages used) and previous programming course attendance (yes/no). A mark sheet was produced, shown in figure 3, which exposed the assessment of consistency. A marking protocol was developed to resolve ambiguities in multiple-assignment responses: essentially, we mark ambiguous responses across each row and look for the column which maximises the number of marks. We note that this makes consistency easier to achieve and, if anything, may dilute its effect on success.

These improved materials were used in all of the experiments analysed below.

5 New experiments

To evaluate more reliably the claim that consistency has a noticeable effect on success in learning to program, the improved experiment was repeated several times by collaborating experimenters: at the University of Newcastle, Australia; twice at Middlesex University, UK; at the University of Sheffield, UK; at the University of York, UK; at the University of Westminster, UK; at Banff and Buchan college, UK and at OSZ TIEM Berlin, Germany. The data for the first six of these experiments has been provided to us and is analysed here.

Question	Answers/s		Model/s
5. int $a = 10$; int $b = 20$; $a = b$; $b = a$;	$a = 10$	$b = 0$	M1+S1
	$a = 20$	$b = 10$	(M1+S3)/(M2+S3)/(M3+S3)/(M4+S3)
	$a = 10$	$b = 10$	M2+S1
	$a = 0$	$b = 20$	M3+S1
	$a = 20$	$b = 20$	M4+S1
	$a = 40$	$b = 30$	M5+S1
	$a = 30$	$b = 30$	(M5+S3)/(M6+S3)/(M7+S3)/(M8+S3)
	$a = 30$	$b = 0$	M6+S1
	$a = 30$	$b = 50$	M7+S1
	$a = 0$	$b = 30$	M8+S1
	$a = 10$	$b = 20$	(M9+S1)/(M11+S1)/
			(M11+S3)
	$a = 20$ $a = 10$	$b = 20$ $b = 10$	(M10+S1)/(M2+S2)/(M4+S2)
	$a = 0$ $a = 20$	$b = 10$ $b = 0$	(M1+S2)/M3+S2)
	$a = 30$ $a = 10$	$b = 20$ $b = 30$	(M5+S2)/(M7+S2)
	$a = 0$ $a = 30$	$b = 30$ $b = 0$	(M6+S2)/(M8+S2)
	$a = 10$ $a = 10$	$b = 20$ $b = 20$	(M11+S2)

Fig. 2. Sample answer sheet

Participant code	Age	Sex	Time to do test	Prior programming	A-Level/s	Prior programming courses	Course result

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) / S	Assign means equal (M10) / S	Swap values (M11) /Ss / I	
	Lose-value (M1) /Ss / I	Keep-value (M2) /Ss / I	Lose-value (M3) /Ss / I	Keep-value (M4) /Ss / I	Keep-value (M5) /Ss / I	Lose-value (M6) /Ss / I	Keep-value (M7) /Ss / I	Lose-value (M8) /Ss / I				
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
C0												
C1												
C2												
C3												

Additional notes:

Fig. 3. The marksheet

The pass rates in the experiments and overall are shown in table 4. The pass rate in Middlesex 2 is exceptionally low, in Sheffield high and in York exceptionally high. There is a gradient in UK universities in the prior achievement levels of admitted students: Middlesex and Westminster are towards the lower end, Sheffield and York towards the higher. Differences in pass rates may reflect this. Note that Middlesex 1 and Middlesex 2 were successive in-course examinations of the same cohort.

We first divide the population according to whether they reported prior programming experience, and alternatively according to reported prior programming course attendance. In addition, based on reports of programming languages used, we classify programming experience as relevant to the test or not (essentially, whether or not subjects had been exposed to mechanisms of assignment and sequence similar to those used in the course they were about to take). Table 5 shows the effects of these background factors on success (the small number who did not reply in each case are ignored). There were few strong differences in the figures. Some of these results were more significant than others: we don't comment on that at this point, but rely on meta-analysis. Effects of age and sex are not analysed here: there were very small numbers of women in the experiments, too small to analyse with the tools we have available; and the very small age spread in the populations, typically two or three years, showed almost no differences in the experiments in which we analysed it.

To begin to look at the effect of consistency measured in the test on success in the course examination, we looked at the success rates of consistent subjects against the rest. As part of the improved experimental protocol, we were able to recognise levels of consistency: subjects who use a single model throughout are consistent; those who use two related models are also consistent but less so. By combining neighbouring columns in the marksheet of figure 3 we identified consistency levels C0 (one model), C1 (two related models), C2 (four related models) and C3 (any assignment model). In practice C0 was always large and the others, except in Middlesex 2, very small. Nevertheless we analyse the effect of consistency in two ways: C0/notC0 and C0-C3/notC. Table 6 gives the results.

In each experiment we found the same thing: consistent subjects did better than the rest. The effect was weakest in the Middlesex 1, Westminster and York experiments. At York, as at Aarhus, most subjects scored consistently in the test (99 out of 105 in York, 124 out of 142 in Aarhus). There were so few non-consistent subjects at York that we could put little weight on that particular result, but we can, as we should, include it in the meta-analysis. At Middlesex, where we have access to the examination materials, we know that Middlesex 1 was a non-technical first in-course quiz, largely bookwork, whereas Middlesex 2 was a more technical second quiz. Middlesex 2 separated students more radically and showed a stronger effect of consistency, with far fewer passes in the non-consistent groups. It may be that the effect of consistency is generally weaker in non-technical examinations. We believe that this matter is

Table 2. Anticipated mental models of `a=b`

Model	Description	Effect
M1	right to left move	a←b ; b←0
M2	right to left copy	a←b
M3	left to right move	a→b ; 0→a
M4	left to right copy	a→b
M5	right to left move and add	a←a+b ; b←0
M6	right to left copy and add	a←a+b
M7	left to right move and add	a+b→b ; 0→a
M8	left to right copy and add	a+b→b
M9	no change	
M10	equality	a=b
M11	swap	a⇆b

Table 3. Anticipated mental models of `a=b; b=a`

Model	Description
S1	`a=b; b=a` Conventional sequential execution
S2	`a=b` || `b=a` Independent assignments, independently reported
S3	`a,b=b,a` Simultaneous multiple assignment, ignoring effect upon source

Table 4. Success rates

	NewC	Mdx1	Mdx2	Shef	West	York	Overall
Population	71	92	72	58	110	105	508
Success	66%	63%	47%	79%	70%	90%	70%

Table 5. Effect of programming background on success in separate experiments

		NewC		**Mdx1**		**Mdx2**		**Shef**		**West**		**York**		**Overall**	
		pop	succ	pop	succ	pop	succ	pop	succ	pop	succ	pop	succ	pop	succ
Prior experience	yes	46	63%	45	56%	33	52%	14	93%	65	75%	91	92%	294	74%
	no	19	68%	44	70%	36	44%	44	75%	36	61%	14	71%	193	65%
Relevant experience	yes	33	61%	29	76%	21	62%	7	86%	21	71%	58	95%	169	78%
	no	32	69%	50	68%	48	42%	51	78%	80	69%	47	83%	308	68%
Prior course	yes	30	63%	64	64%	51	51%	14	93%	61	74%	34	88%	254	69%
	no	32	69%	23	65%	17	47%	47	81%	30	70%	71	90%	220	76%

Table 6. Effect of consistency on success in separate experiments

	NewC		**Mdx1**		**Mdx2**		**Shef**		**West**		**York**		**Overall**	
	pop	succ	pop	succ	pop	succ	pop	succ	pop	succ	pop	succ	pop	succ
C0	44	80%	35	77%	28	79%	43	91%	62	77%	99	92%	311	84%
notC0	27	44%	57	54%	44	27%	15	47%	48	60%	6	50%	197	48%
C0-C3	52	79%	56	70%	42	64%	45	91%	80	75%	103	90%	378	80%
notC	19	32%	36	53%	30	23%	13	38%	30	57%	2	50%	130	42%

Table 7. Effect of consistency on success in subgroups in separate experiments

		NewC		Mdx1		Mdx2		Shef		West		York		Overall	
		pop	succ	pop	succ	pop	succ	pop	succ	pop	succ	pop	succ	pop	succ
Correct model	CM2	23	87%	11	100%	9	89%	11	73%	8	75%	87	92%	149	89%
	notCM2	48	56%	81	58%	63	41%	47	81%	102	70%	18	78%	359	62%
Incorrect model	C0	21	71%	24	67%	19	74%	32	97%	54	78%	12	92%	162	80%
	notC0	27	44%	57	54%	44	27%	15	47%	48	60%	6	50%	197	48%
With prior experience	C0	33	79%	12	75%	16	88%	14	93%	40	85%	88	92%	203	87%
	notC0	13	23%	33	48%	17	18%	0	_	25	60%	3	100%	91	44%
No prior experience	C0	13	85%	23	78%	12	66%	29	90%	16	62%	11	91%	104	80%
	notC0	6	33%	21	62%	24	33%	15	47%	20	60%	3	0%	89	47%
With relevant experience	C0	24	79%	16	94%	10	100%	7	86%	15	80%	57	95%	129	90%
	notC0	9	11%	13	54%	11	27%	0	_	7	50%	1	100%	40	38%
No relevant experience	C0	15	80%	19	63%	18	67%	36	92%	41	76%	42	88%	171	80%
	notC0	17	59%	40	55%	30	27%	15	47%	39	63%	5	40%	146	50%
With prior course	C0	21	76%	24	79%	22	77%	10	80%	35	77%	65	94%	177	84%
	notC0	9	33%	40	55%	29	31%	1	0%	26	69%	6	50%	111	50%
No prior course	C0	15	93%	11	73%	7	86%	33	94%	14	93%	34	88%	114	89%
	notC0	17	47%	12	58%	10	20%	14	50%	16	50%	0	_	69	46%

Table 8. Overall effect of programming background on success

		pop	succ	χ^2	*df*	p	**Significance**
Prior experience	yes	294	74%	21.02	12	$0.05 < p < 0.10$	weak
	no	193	65%				
Relevant experience	yes	169	78%	18.26	12	$0.10 < p < 0.20$	very weak
	no	308	68%				
Prior course	yes	254	69%	4.67	12	$0.95 < p < 0.98$	none
	no	220	76%				

worth further investigation, and we return to it in our conclusion. In Newcastle, Middlesex 2 and Sheffield we find that consistent subjects do about twice as well as the rest.

Because the mark sheet identified not only consistency but also the particular model(s) used, we can recognise those subjects who apparently have already learned, before the course begins, the mechanisms of assignment and sequence that are going to be taught. In Java these models are M2 for assignment and S1 for sequence, and we called the group that consistently used these models in the test the CM2 group. Almost all of them (131 out of 149) reported prior programming experience. It would seem that most of the York intake could already program: 87 out of 105 were CM2. To see whether the effect of consistency was simply the effect of prior learning of programming, we looked at the population divided by CM2/notCM2 (correct model), and we looked again at subjects outside the CM2 group (incorrect model) divided by C0/notC0. Then we looked at each of the subgroups defined by the background questions analysed in table 5. This gives eight population divisions, shown in table 7.

In the first row of this table the CM2 group does generally better than the rest in every experiment but one – the exception is Sheffield, in which 5 out of 11 had no reported programming background, making this the only experiment in which a substantial proportion of the CM2 group appeared to guess the correct models. In the rest of the table in *every single case* the consistent group does better than the others, usually by a considerable margin.

Table 9. Overall effect of consistency on success

	pop	succ	χ^2	*df*	*p*	**Significance**
C0	311	84%	49.84	12	$p < 0.001$	very high
notC0	197	48%				
C0-3	378	80%	44.51	10	$p < 0.001$	very high
notC	130	42%				

Table 10. Overall effect of consistency on success in filtered subgroups

		pop	succ	χ^2	*df*	*p*	**Significance**
Correct model	CM2	149	89%	34.75	12	$p < 0.001$	very high
	notCM2	359	62%				
Incorrect model	C0	162	80%	40.27	12	$p < 0.001$	very high
	notC0	197	48%				
With prior experience	C0	203	87%	31.65	10	$p < 0.001$	very high
	notC0	91	44%				
No prior experience	C0	104	80%	35.45	12	$p < 0.001$	very high
	notC0	89	47%				
With relevant experience	C0	129	90%	31.60	10	$p < 0.001$	very high
	notC0	40	38%				
No relevant experience	C0	171	80%	35.82	12	$p < 0.001$	very high
	notC0	146	50%				
With prior course	C0	177	84%	35.98	12	$p < 0.001$	very high
	notC0	111	50%				
No prior course	C0	114	89%	38.35	10	$p < 0.001$	very high
	notC0	69	46%				

6 Meta-analysis

The Winer procedure of meta-analysis (Winer et al., 1971) was used to examine the overall effect of consistency and/or programming background on success. The procedure combines p values from χ^2 analysis of separate experiments – the probability of obtaining the effect by accident – to give an overall p value. Because this is a meta-analysis of several experiments, our threshold significance value is set at a conservative 0.01 (1%).

Table 8 summarises the overall effects of programming background on success, showing the size of the effect, the χ^2 value and significance. None of the programming background factors had a large or a significant effect. The weak effect of prior programming experience and prior relevant experience was driven by and overall 61% of candidates with prior programming experience, a third of which came from the York experiment. Attending a programming course was shown to have no significant effect on success, both overall and in each individual experiment.

On the other hand, meta-analysis shows in table 9 a large and highly significant effect of consistency on success in both the C0/notC0 and C/notC group arrangements. Despite the weak effect in the first quiz at Middlesex and the Westminster experiment, especially in C/notC, none of the experiments is driving the result: if we eliminate any one of them there is still strong significance.

Table 10 summarises the overall effect of consistency on success in the eight population divisions characterised by programming background factors. The overall result confirms the result of the initial experiment by demonstrating a highly significant effect of consistency on success in *every* slice. None of the experiments is driving the overall result: if we eliminate any one we still find a large significant effect.

This analysis shows that consistency is not simply the effect of learning to program. The CM2 group does do better than any other, as might be expected. But there are slightly more

individuals who are C0-consistent but not CM2, their success rate is almost as good, and they are almost twice as likely to pass as those who are not consistent. We note that the CM2/notCM2 division (89%/62%) is a more effective predictor of success than prior programming experience (74%/64%), even if we take the weakly significant effect of prior experience at face value, but both give many more false negatives than either measure of consistency (C0/notC0 84%/48%, C/notC 80%/42%). We can see these effects more starkly if we look at failure rates: the CM2 group, which has learnt one of the basics of imperative programming, has an 11% failure rate against 38% for the others; programming background gives 26%/36%; consistency gives either 16%/52% or 20%/58%.

The size of the effect varies according to the population division but it is significant everywhere and it is never small. Programming background, or its absence, doesn't eliminate the effect of consistency.

7 Conclusion and future work

The test characterises two populations in introductory programming courses which perform significantly differently. More than half of novices spontaneously build and consistently apply a mental model of program execution; the rest are either unable to build a model or to apply one consistently. The first group perform very much better in their end-of-course examination than the second. Despite the tendency of institutions to rely on students' prior programming experience as a predictor of success, programming background has only a weak effect on novices' success, and though effective prior learning of assignment and sequence has a stronger effect, it is not as strong as consistency.

Our test is by no means perfect. We ask subjects (outside the CM2 group, almost all of whom know the right answer) to both invent and consistently use a mental model of a mechanism they have never encountered before: there may be an effect to be discovered in those who can use but not invent such a model; there may be an effect to be discovered using non-programming problems. The large proportion of false negatives – students judged inconsistent who nevertheless succeed in the examination – is the greatest deficiency of our test instrument at present.

We haven't interviewed our subjects to find out why they answered as they did in the test. This is a deficiency in our research. It would be interesting to discover why inconsistent subjects, in particular, made the choices that they did.

Our research has uncovered an effect, but has not begun to explain it. Consistent subjects build a mental model, something that follows rules like a mechanical construct, and this is what more or less what Baron-Cohen's systematizers do. We speculate that we might be measuring a similar trait by different instruments, and Wray's results tend to support this. But there may be different subgroups in the consistent group, with different reasons for being consistent, and the number of false negatives in the inconsistent group suggests strongly that there could be subgroups there too.

We would like to investigate the phenomenon of consistency and its effects in a longitudinal study. More than half of university entrants to introductory programming courses display consistency, but how early in life can we detect it? And how does it affect computer science study in other introductory subjects, or later study?

Simon et al. (2006b) stated "We do not pretend that there is a linear relationship between programming aptitude and mark in a first programming course, or that different first programming courses are assessed comparably; but we have succumbed to the need for an easily measured quantity." We have followed in that tradition, using institutions' own examinations to assess programming skill. But the levels of achievement they measure vary considerably, and we have seen signs that initial, necessarily non-technical, assessments of skill may be ineffective. We are also aware that institutional pressures, for example to produce a particular pre-determined success rate, may distort assessment. If we are to research reliably the connection between mea-

sures of programming aptitude and success in learning to program, it seems that we should consider how to measure programming skill(s) in an institution-independent way.

8 Acknowledgements

The initial experiment, the improved experimental materials and the analysis reported here are the work of Dehnadi, who was supported and criticised by collaborators/supervisors Bornat and Adams. Simon, of Newcastle University, contributed questions about programming background, sex and age, conducted one of the experiments analysed here, and helped develop the data analysis.

We would like to thank Mr. Ed Currie, Dr. Peter Rockett, Mr. Christopher Thorpe and Dr. Dimitar Kazakov, the collaborators who conducted the other experiments and made the data available to us, and our peers in PPIG (Psychology of Programming Interest Group) for their valuable criticism and advice.

References

Baron-Cohen, S., Richler, J., Bisarya, D., Gurunathan, N., and Wheelwright, S. (2003). The systemising quotient (SQ): An investigation of adults with Asperger's syndrome or high functioning autism and normal sex differences. *Philosophical Transactions of the Royal Society, Series B, Special issue on "Autism : Mind and Brain"*, 358:361–374.

Baron-Cohen, S., Wheelwright, S., Skinner, R., Martin, J., and Clubley (2001). The autism spectrum quotient (AQ): Evidence from Asperger's syndrome/high functioning autism, males and females, scientists and mathematicians. *Journal of Autism and Developmental Disorders*, 31:5–17.

Beise, C., Myers, M., VanBrackle, L., and Chevli-Saroq, L. (2003). An examination of age, race, and sex as predictors of success in the first programming course. *Journal of Informatics Education and Research*, 5(1):51–64.

Bennedsen, J. and Caspersen, M. E. (2006). Abstraction ability as an indicator of success for learning object-oriented programming? *SIGCSE Bull.*, 38(2):39–43. 1138430.

Bennedssen, J. and Caspersen, M. E. (2008). Abstraction ability as an indicator of success for learning computing science? In *ICER '08: Proceedings of the fourth international workshop on Computing education research*, pages 15–26, New York, NY, USA. ACM.

Bornat, R., Dehnadi, S., and Simon (2008). Mental models, consistency and programming aptitude. In *ACE '08: Proceedings of the tenth conference on Australasian Computing Education Conference*, pages 53–61, Darlinghurst, Australia. Australian Computer Society, Inc.

Caspersen, M. E., Larsen, K. D., and Bennedsen, J. (2007). Mental models and programming aptitude. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 206–210, New York, NY, USA. ACM.

Cross, E. (1970). The behavioral styles of computer programmers. In *Proc 8th Annual SIGCPR Conference*, pages 69–91, Maryland, WA, USA.

Dehnadi, S. (2006). Testing programming aptitude. In *Proceedings of the PPIG 18th Annual Workshop*, England.

Dehnadi, S. and Bornat, R. (2006). The camel has two humps. In *Little PPIG*, Coventry, UK.

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

Dyck, J. L. and Mayer, R. E. (1985). BASIC versus natural language: is there one underlying comprehension process? In *CHI '85: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 221–223, New York, NY, USA. ACM.

Fincher, S., Lister, R., Clear, T., Robins, A., Tenenberg, J., and Petre, M. (2005). Multi-institutional, multi-national studies in CSEd research: some design considerations and trade-offs. In *ICER '05: Proceedings of the first international workshop on Computing education research*, pages 111–121, New York, NY, USA. ACM.

Fix, V., Wiedenbeck, S., and Scholtz, J. (1993). Mental representations of programs by novices and experts. In *CHI '93: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*, pages 74–79, New York, NY, USA. ACM.

Huoman, J. (1986). Predicting programming aptitude. Master's thesis, Department of Computer Science, University of Joensuu, Finland.

Kessler, C. M. and Anderson, J. R. (1986). Learning flow of control: Recursive and iterative procedures. *Human-Computer Interaction*, 2:135–166.

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B., and Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. In *ITiCSE-WGR '04: Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 119–150, New York, NY, USA. ACM.

Mayer, D. and Stalnaker, A. (1968). Selection and evaluation of computer personnel: the research history of SIG/CPR. In *Proc 1968 23rd ACM National Conference*, pages 657–670, Las Vegas, NV, USA.

Mayer, R. (1992). *Thinking, Problem Solving, Cognition*. W. H. Freeman and Company, New York, 2nd edition.

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

McCoy, L. P. and Burton, J. K. (1988). The relationship of computer programming and mathematics in secondary students. *Computers in the Schools*, 4(3/4):159–166.

Perkins, D., Hancock, C., Hobbs, R., Martin, F., and Simmons, R. (1986). Conditions of learning in novice programmers. *Journal of Educational Computing Research*, 2(1):37–55.

Perkins, D. N. and Simmons, R. (1988). Patterns for misunderstanding: An integrative model for science, math, and programming. *Review of Educational Research*, 58(3):303–326.

Putnam, R. T., Sleeman, D., Baxter, J. A., and Kuspa, L. K. (1986). A summary of misconceptions of high school BASIC programmers. *Journal of Educational Computing Research*, 2(4):459–472.

Raadt, M., Hamilton, M., Lister, R., Tutty, J., Baker, B., Box, I., Cutts, Q., Fincher, S., Hamer, J., Haden, P., Petre, M., Robins, A., Simon, Sutton, K., and Tolhurst, D. (2005). Approaches to learning in computer programming students and their effect on success. *Higher Education in a changing world: Research and Development in Higher Education*, 28:407–414.

Rountree, N., Rountree, J., Robins, A., and Hannah, R. (2004). Interacting factors that predict success and failure in a CS1 course. *SIGCSE Bull.*, 36(4):101–104.

Samurcay, R. (1985). The concept of variable in programming: Its meaning and use in problem-solving by novice programmers. *Education Studies in Mathematics*, 16(2):143–161.

Shneidermann, B. (1985). When children learn programming: Antecedents, concepts and outcomes. *The Computing Teacher*, 12(5):14–17.

Simon, Cutts, Q., Fincher, S., Haden, P., Robins, A., Sutton, K., Baker, B., Box, I., de Raadt, M., Hamer, J., Hamilton, M., Lister, R., Petre, M., Tolhurst, D., and Tutty, J. (2006a). The ability to articulate strategy as a predictor of programming skill. In *ACE '06: Proceedings of the 8th Australasian Computing Education Conference*, pages 181–188, Darlinghurst, Australia. Australian Computer Society, Inc.

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., and Tutty, J. (2006b). Predictors of success in a first programming course. In *ACE '06: Proceedings of the 8th Australasian Computing Education Conference*, pages 189–196, Darlinghurst, Australia. Australian Computer Society, Inc.

Soloway, E. and Spohrer, J. C. (1989). Some difficulties of learning to program. In *Studying the Novice Programmer*, pages 283–299. Lawrence Erlbaum Associates.

Spohrer, J. C. and Soloway, E. (1986). Novice mistakes: are the folk wisdoms correct? *Commun. ACM*, 29(7):624–632.

Tolhurst, D., Baker, B., Hamer, J., Box, I., Lister, R., Cutts, Q., Petre, M., de Raadt, M., Robins, A., Fincher, S., Simon, Haden, P., Sutton, K., Hamilton, M., and Tutty, J. (2006). Do map drawing styles of novice programmers predict success in programming?: a multi-national, multi-institutional study. In *ACE '06: Proceedings of the 8th Australasian Computing Education Conference*, pages 213–222, Darlinghurst, Australia. Australian Computer Society, Inc.

Tukiainen, M. and Mönkkönen, E. (2002). Programming aptitude testing as a prediction of learning to program. In *PPIG '02: Proceedings of the 18th Annual Workshop of the Psychology of Programming Interest Group*, pages 45–57, London, England.

van Someren, M. W. (1990). What's wrong? understanding beginners' problems with Prolog. *Instructional Science*, 19(4/5):257–282.

Wason, P. and Johnson-Laird, P. (1968). *Thinking and Reasoning*. Harmondsworth: Penguin.

Wilson, B. C. and Shrock, S. (2001). Contributing to success in an introductory computer science course: a study of twelve factors. *SIGCSE Bull.*, 33(1):184–188.

Winer, B., Brown, D. R., and Michels, K. M. (1971). *Statistical principles in experimental design*. Mc.Graw-Hill, Series in Psychology.

Wolfe, J. (1971). Perspectives on testing for programming aptitude. In *Proc 1971 26th ACM National Conference*, pages 268–277, Chicago, IL, USA.

Wray, S. (2007). SQ minus EQ can predict programming aptitude. In *Proceedings of the PPIG 19th Annual Workshop*, Finland.