Mental models and the learning of programming (case for support)

Richard Bornat$^1$, Benedict du Boulay$^2$ and Saeed Dehnadi$^1$

$^1$School of Engineering and Information Sciences, Middlesex University, UK
$^2$Department of Informatics, University of Sussex

1 Previous Track Record

**Professor Richard Bornat** has a track record across a wide range of computer science, from compiling [4] through programming languages [1] and interfaces [6] to separation logic [7]. His interest in the problems of learning programming stem from decades searching fruitlessly for effective ways of teaching the subject [5]. He supervised Dehnadi’s thesis which provided strong evidence for the rôle of mental models. During his work on the Jape logic calculator he collaborated with the Open University in evaluating Jape’s effect and learnt the value of close observation of novices in evaluating educational interventions [2, 3]. He has held several EPSRC grants; two related to Jape (The Calculator Project, Joint ESRC, MRC, SERC Cognitive Science/HCI initiative, 9019558; Visualisation in the software development process, GR/M06581/01) are relevant to this application. His expertise in the formal background of programming, his record of software development and his long experience as a teacher are all relevant to this work.

**Professor Benedict du Boulay** is a founding member of the Human-Centred Technology (HCT) Research group at the University of Sussex. His initial interest in the psychology of programming showed the influence of novices’ mental models in early learning of programming [13]. He also identified some of the specific problems novices faced with concepts such as assignment and loops [12, 11]. Various pieces of work explored novices understanding of Prolog [8, 14], and in particular, how the notation of the tracing mechanism assisted or hindered their understanding of the underlying (otherwise hidden) mechanisms such as backtracking [15]. More recent work was concerned with understanding how different factors affect the way novice users co-ordinate representations when performing programming tasks such as debugging [16]. In terms of the current application he will deploy his expertise in designing and running experiments to elucidate novice programmer behaviour and understanding.

He has been PI or CI on 6 EPSRC grants. Grants in the area of this application are: Co-ordination of multiple external representations in learning programming (GR/N64199/01, see above); Authoring as acting: exploring embodied interaction in game authoring environments for children (ES/F031106/01, which developed ways to teach children elementary programming through embodied interaction with a system).

**Dr. Saeed Dehnadi** has experience as a mathematics teacher and analyst/programmer. His thesis research [9, 10] inspired this application. He has shown an ability to devise novel tests of novice performance, skill in administering experiments, has organised work with several collaborators, has experience of analysis of results and software development. His experience of interacting with students in generating a test for the use of mental models is particularly relevant.

There is considerable experience at Sussex of research into the psychology of learning programming. Middlesex has an extensive programme of student profiling, with which this project will cooperate.

Selected papers


2 Background

UK industry is currently bewailing a shortage of competent programmers. Much of the focus of discussion has been on schools [47, 17, 36, 24]. But it is universities that train programmers, and they find it difficult to convert a high enough proportion of computer science undergraduates into competent practitioners. This application seeks to address that issue.

Teaching undergraduates to program presents an unexpected challenge. A high proportion of students on a typical introductory course fail to reach an acceptable standard, while others succeed with relative ease, so that examination results show a ‘double hump’ rather than a normal curve [18, 46]. This is a long-standing international problem [39, 35, 26], persisting as long as the subject has been taught, which occurs even though students are apparently well-qualified and strongly motivated. Attempts to predict success in early programming courses by educational attainment, by other measures of background, and by psychometric tests have all disappointed [45, 38, 31, 25, 51, 20, 19, 46]. The problem persists despite considerable and continuing innovation in programming languages, often motivated by educational concerns, and a range of innovatory teaching methods.

This project focusses on this intriguing problem, and will develop new tools to help educators address it. Even though we do not explicitly focus on schools, our tools will be applicable there.
1. Read the following statements and tick the box next to the correct answer in the next column.

<table>
<thead>
<tr>
<th></th>
<th>The new values of (a) and (b):</th>
<th>Use this column for your rough notes please</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(a = 10) (b = 10)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(a = 30) (b = 20)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(a = 0) (b = 10)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(a = 20) (b = 20)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(a = 20) (b = 20)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(a = 0) (b = 30)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(a = 10) (b = 20)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(a = 20) (b = 10)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(a = 20) (b = 0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(a = 10) (b = 30)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(a = 30) (b = 0)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: A sample test question

3 Previous work

The double-hump effect is widely reported. Bennedsen and Caspersen [18] researched the question, with most of their evidence coming from the USA. Two large-scale international projects [39, 35, 26] showed widespread occurrence of low achievement, and the latter two studies, using their own tests, highlighted low levels of programming skill after an introductory course.

Predictors of success are weak. Simon et al. [48], following those multi-national studies, looked at various cognitive activities and found no strong predictors (the best, though weak, had to do with drawing maps). Robins [46] gives a survey of predictive attempts. Evans and Simkin [25] and Mazlack [38] describe typical experiments with aptitude tests; IBM, which invented and used the PAT test, no longer circulates or promotes it; Huoman [31] found no correlation between a modern aptitude test and success in a programming course, once programming background was allowed for. A three-year longitudinal study by Bennedssen and Caspersen [19] found that general abstraction ability was not a predictor for success in learning computer science. Background education, often used as an admissions criterion, has little predictive value [45, 51]. Neither do age or sex [20].

3.1 Dehnadi’s test

Programming is complex. Programmers need to be able to predict the execution of a program; they must reason backwards from execution failure to a cause in the program; they need to reason inductively about loops; they must be able to design a program as a solution to a problem. But a considerable thread in the literature on the psychology of programming, far too voluminous to reference fully but see [23, 29, 32, 49, 33, 37, 21, 44, 41] holds that a novice must form a mechanical mental model of a program’s execution in order to understand programming and to be able to invent programs of their own.

Dehnadi [9,10] discovered that novices divide reliably into two groups, one apparently capable of spontaneously forming and using an algorithmic mental model of simple program execution, the other apparently not ready to do so. The failure rate differs significantly between the two groups over a large test population and a range of institutions. Analysis shows that the difference is not determined by prior educational experience (the test does not, for example, simply measure prior programming education or experience). Although there is a considerable proportion of ‘false negatives’ who pass the course despite not forming a model initially, it does provide novel evidence for the rôle of mental models.

The test confronts subjects, ideally programming innocents, with symbolic programming problems. The problems are expressed using partially familiar notation (equality symbol,¹ integer values, single-letter variable names) with some novel elements (the word ‘int’, the semicolon, multiple lines) and ask about ‘new values’, implicitly suggesting that something more than algebraic calculation is going on. No further explanation is given.

The questions involve assignment and sequence, the simplest mechanisms taught in a typical introduc-

¹Use of ‘=’ for assignment is educationally problematic: equality and assignment are very different notions. But in this particular context it seems to be an advantage that a familiar symbol has been used.
Problem programs are written in Java, the most common introductory language. The simplest questions, like the one in figure 1, involve a single assignment. Eight of the tickboxes in the second column correspond to a particular execution mechanism of the assignment statement: combinations of left-to-right or right-to-left action, copying or moving of the source, and replacement of or addition to the destination. In addition there are swap, algebraic equality (variables are made equal) and non-execution (nothing happens).

Other questions involve two or three assignments in sequence. Here the tickboxes are ambiguously associated with mechanisms of assignment and of sequence (three are recognised: conventional sequential action and two versions of simultaneous action).

Overall it is possible to see if a subject appears to be using throughout the test a mental model of assignment which corresponds to a single recognisable execution mechanism, or uses several alternately, or none. It is remarkable that, in six experiments in a range of five universities in the UK and one in Australia, almost all candidates were prepared to attempt the test despite the intentional lack of explanation, and that a high proportion – between 50% and 70% – seemed to form and use a single rational mental model of execution throughout the test. About 80% of those who did so passed the introductory course, against only about 45% of those who did not. The effect persists even if those subjects are eliminated who apparently know something about programming, because they use the correct Java models in their answer. It is equally present in those who report prior programming education and/or experience as in those who do not.

The test is not accurate enough to predict programming aptitude – 45% of those with a negative prediction succeed at the end of the year, some few of those succeed very well – but it is replicable and its results are highly significant: \( p < 0.001 \) in the meta-analysis of all Dehnadi’s experiments [9,10].

3.2 Impact of Dehnadi’s result

Dehnadi’s result sparked a good deal of interest, including a considerable amount of scepticism. First responses were along the lines that all that was being tested was prior experience [30, 34]. Caspersen et al. [22] and Wray [52] ran different experiments which didn’t find a result and claimed a refutation. Scepticism persists [43]. But his result reawakened a dormant research area and raised a considerable number of issues.

On the face of it, Dehnadi’s result suggests that there could be an explanation for the double-hump phenomenon in terms of innate psychology. Robins [46], however, argues that there has been so much research into predictors of success in learning to program that if there were such an explanation (the ‘geek gene’) it would already have been found, and presents a persuasive alternative. Initial programming courses are made up of a sequence of topics, each underpinning the next. Robins’ simulations of performance under such conditions produce bimodal distributions like those observed. The mechanical nature of execution is a fundamental initial concept, and students who stumble over it can make only flawed progress. There are a succession of further similar obstacles along the way.

Dehnadi’s test was designed to investigate novices, but can be turned to measure the progress of students throughout the course. Ford and Venema [27] used the test to examine the achievement of students at the end of their introductory course. Remarkably, amongst those with passing grades, only 50% appeared to understand assignment and sequence, and 23% used no recognisable model.

The picture overall is of disappointing, even perplexing, student achievement in introductory programming courses. Robins’ simulation holds out the hope that, if obstacles can be identified and ways charted around them, something might be done. Dehnadi’s test has identified one such obstacle and it seems also that it can be used, post-course, to measure whether that obstacle has in fact been overcome. This project proposes further investigation of that obstacle and student experience of it, the development of test(s) to identify students who have problems with it, and diagnostic instruments independent of conventional examinations to measure the effects of different teaching methods.

2In functional/declarative programming the simplest mechanisms are function application and argument substitution, which have their own difficulties.
4 Academic impact

The main academic impact of this project will be the production of a large body of empirical data about novices and their experience in learning to program.

1. It will enable a large number of investigators to experiment with a refined version of Dehnadi’s test and will collect and collate the data.
2. It will provide data about novices’ expressed reasons for answering test questions as they do.
3. It will provide tests of another identified obstacle and will collect and collate the results.
4. It will track undergraduates over the course of their degree, testing and interviewing, and will provide data about their development.
5. It will provide tests which educators can use to assess the effectiveness of their teaching and will collect and collate the data.

By investigating the intellectual trajectory of novices confronting the conundrum of formal (meaning-free) program execution, it will resolve or support the scepticism of those who doubt the raw results of testing. By widening the area of investigation to other cognitive areas it will provide larger and more varied data, permitting new hypotheses and new investigations. By providing tests of competence in differing programming areas, it will permit differentiation of subpopulations and more focussed psychological modelling.

5 Research Hypothesis and Objectives

Our hypothesis is that a more detailed understanding of novices’ difficulties in learning to program will considerably improve the effectiveness of computer science education. Our objectives are

1. To produce online easily usable tests, integrated with existing learning environments.
2. To interview test subjects in order to discover more about the mental models they use (or not).
3. To develop new tests which measure the development of mental models of a more complicated programming construct (probably loops).
4. To use our tests to measure, post-course, the effectiveness of different courses.
5. To conduct longitudinal studies over the span of the project measuring the evolution of mental models of programming concepts and comparing with academic progress.

6 Programme and methodology

Dehnadi’s test identifies the first major obstacle to the learning of imperative programming. We shall map that obstacle in order that ways may be found round it. Other, later, obstacles exist (see §6.4) but lack corresponding diagnostic tests. We shall provide tests to identify one such obstacle at least and map it to see if ways can be found round it. We shall investigate the effectiveness of teaching techniques, and provide diagnostic tools for teachers to use.

6.1 Webification

We propose a large programme of testing, data collection and collation, with a range of collaborators in different institutions. Generation of Dehnadi’s test from a list of execution models is already automated, and we shall develop online delivery and assessment. We believe that this can be achieved with the widely-used open-source VLE Moodle, thus exploiting class registration lists and the like, as well as providing a guarantee of confidentiality. It will also enable us to experiment with rapid feedback to test subjects. Unfortunately the existing interface to Moodle’s testing and marking mechanism won’t support our test, so this is not a completely straightforward task. The widely-used commercial VLE Blackboard is unfortunately unsuitable, and some stand-alone solution will have to be provided – again not completely straightforward. The outcome will be a test which educators can use easily to test their students.

6.2 A refined version of Dehnadi’s test

The problem we are investigating is failure, which is not simply the flip side of success. Dehnadi’s test reliably identifies two groups, one expected to succeed and the other to fail. There are some false posi-
tives in the first group, but a very large proportion of false negatives in the second. Refinement will concentrate on false negatives.

Some part of the test’s inaccuracy is surely due to its peculiarity (novel notation, no explanation). We shall experiment with alternative presentations, with more or less explanation and preamble. As a result of interviews (§6.3) we may recognise some novel models which could be added to the test, or some pattern of answers not currently identified. The outcome will be a test with a reduced proportion of false negatives, which will be made available to educators.

6.3 Mapping the obstacle

Dehnadi’s test focusses on mental models of symbolic execution. Symbolic execution is based on formal rules, and such rigid formality is far from everyday experience. By using tests which investigate mental models of physical and mechanical systems, we shall investigate whether the problem is with mental modelling per se, or with the particular kinds of mental models required for programming. By interviewing test subjects we shall learn a great deal more about their use of models (or non use). For this purpose we shall be able to access students at Middlesex and Sussex; we expect also to gain access at Queen Mary (University of London), and possibly at Cambridge – across a range of institutions and the academic spectrum. We shall experiment to see if with some minor assistance those who don’t initially form a model can be encouraged to do so, as in Vigotsky’s ZPD [50]. The outcome will be a clearer delineation of the obstacle as seen by the novice, which we hope will be useful in remediation; the data will be made available to other researchers.

6.4 Further obstacles

Sequential execution is only the first of the modelling problems which arise when learning programming: procedure call is a mechanism which also appears to require mental modelling; pointers (even implicit pointers as in Java) raise difficult modelling questions; conditionals demand ‘logical arithmetic’ reasoning; and loops (or recursion) are a very large and complex obstacle [40, 42].

We cannot deal with all of these problems within a single project, and we intend to focus on loops. Our method will be to observe students who are grappling with exercises, to talk to them and listen to their rationalisations of their answers – some of which will be mistaken but illuminating of their understanding. This was the method used by Fung et al. [28] in their work on novices’ understanding of Prolog, and by Dehnadi [9,10] in developing his test. The outcome will be a diagnostic test to measure the development of novices’ understanding of loops, and which will be made available to educators.

6.5 Cohort testing

The online test developed in §6.1, the refined test of §6.2 and the tests developed in §6.4 will all be applied to year cohorts at Middlesex and Sussex, probably at Queen Mary, and at other institutions where we are able to recruit collaborating researchers. As in Dehnadi’s original research, the results will be correlated with course examination results; in addition (see §6.6) we shall correlate them with the results of tests run after the end of the course. The outcome will be that this data (suitably anonymised) will be collated and made available to researchers.

6.6 Investigating teaching methods

Claims of the effectiveness of teaching methods depend largely on course examination results, and are notoriously difficult to assess. Using method-independent tests of understanding will enable us to make comparisons between different methods. We shall not, of course, be able to rank teaching methods since courses may have different objectives, but we will be able to compare them in important aspects. Following Ford and Venema [27] we shall run post-course assessments, using the tests we develop, to see what proportion of ‘successful’ examination candidates have grasped the concepts which our tests cover. The outcome will be a procedure which educators can use to assess their own effectiveness.

6.7 Longitudinal studies

By interview and repeated testing of a few individuals we shall discover their learning trajectories; following Robins [46], we would expect to see some of those who don’t initially form an algorithmic model
to do so later, later still to use the correct models, and then to achieve overall success. We also intend to track the progress and understanding of a few individuals over the three years of their degree study, tracking the development of their understanding of fundamental programming notions and how that understanding affects their choices and their levels of success. The outcome will be a narrative that captures the experience of our subjects; the data will be made available to other researchers.

6.8 Division of labour

There is a single full-time researcher (Dehnadi), one half-time (unnamed), plus two part-time: Bornat will spend two days per week, du Boulay one day per week. Timescales and dependencies of tasks are shown in the workplan.

Dehnadi will be involved in all activities. The unnamed researcher will concentrate on data collection and collation. Bornat will participate in webification. Bornat and du Boulay will participate in refining the test, mapping the obstacle, further obstacles and longitudinal studies – i.e. all interviewing tasks.

Project management will be by meeting of all participants, bi-monthly on average. Between meetings there will be communication by email and by video call (e.g. Skype). The team has extensive experience of inter-institutional collaborative research.

7 Importance

This project aims to diminish the waste of talent and enthusiasm in the annual obstacle race which is the university introductory programming course, enabling a greater proportion of each year’s intake to become competent and employable programmers. Students benefit, industry benefits, and university educators waste less of their time and effort dealing with unnecessary failure. School educators may benefit (though this is not our focus) from a clearer identification of the problems they face.

If successful we expect that the focus of programming-teaching innovation will shift away from choosing a 'best' programming language and an 'inspiring' application area to techniques which concentrate on individual student’s immediate difficulties.

References


[28] Pat Fung, Mike Brayshaw, Benedict Boulay, and Mark Elsom-Cook. Towards a taxonomy of novices’ mis-


[36] Ian Livingston and Alex Hope. Next Gen.: Transforming the UK into the world’s leading talent hub for the video games and visual effects industries. Technical report, 2011.


