The evaluation of TED, a techniques editor for Prolog programming

Tom Ormerod
Department of Psychology
Lancaster University

Linden Ball
School of Health and Community Studies
Derby University

Submitted to PPIG 7, Edinburgh January 1995.

This note forms part of a larger paper to be submitted for publication by the full TED project team whose additional memebers are Dave Robertson, Andy Bowles and Helen Pain from Edinburgh University, Pual Brna from Lancaster University, Mike Brashaw and Hank Kahney from The Open Univeristy. We report only the work carried out by the authors as part of the overall project when they were both at Loughborough University.

Aims

The aim was to evaluate both the utility of a techniques approach to teaching Prolog, and also to evaluate the extent to which the TED editor facilitates the application of a techniques approach. In addition, the evaluation enabled research to be undertaken into the nature of Prolog expertise, in particular the extent to which Prolog expertise is dictated by internalised knowledge about Prolog program structures, by strategic knowledge about program design, and by external features of the programming environment.

Method

The Prolog course

Three groups of students were taught a course in Prolog programming over consecutive years. The course involved ten weeks of lectures and practicals, in which students were required to complete weekly programming exercises under tutors' guidance. In order that the teaching process could be equated across the different student groups, a standard 'CCP' (Cases, Control & Processes) method was used throughout for the development and explanation of all programs. CCP describes the order in which students were encouraged to consider the design of Prolog programs.

Subjects

Subjects were 32 students studying various Information Technology courses. All had a small amount of prior programming experience with a variety of languages, though none were more than novice level in terms of experience or

proficiency at any other language, as assessed by a student questionnaire. None had any previous experience of Prolog programming.

Design

The first group of students (N=10) received a standard course in Prolog without any techniques instruction and using the MacProlog environment. The second group (N=12) were taught essentially the same course except that all instruction and programming involving recursion was based around a set of seven techniques developed at Edinburgh, again using the MacProlog environment. The third group (N=10) received exactly the same course as the second group, except that the TED editor was used as the sole programming environment.

Tasks, materials and procedure

Data were collected at the end of the course using three main paradigms. The first consisted of a program recall task, in which students were required to reconstruct a complex recursive Prolog program presented to them for a short duration over repeated trials. The recall paradigm can indicate the extent to which students internalise techniques knowledge as they acquire Prolog skills. The second paradigm consisted of problem and program categorisation tasks, in which students were required to sort sets of cards showing either problem statements or Prolog programs for simple recursive procedures into categories of the students' own choosing. The categorisation paradigm can indicate whether students recognise techniques within programs and whether they can utilise techniques knowledge in understanding problem statements. The third paradigm consisted of six program writing tasks, in which students were required to produce coded solutions to problem statements requiring simple recursive procedures. The writing paradigm can indicate whether students are able to apply techniques in the design and coding of Prolog solutions.

Six program writing tasks of increasing complexity were presented to subjects. All programs required the development of two-arity procedures containing list_head, same, arithmetic_after and arithmetic_before techniques. For example, the problem statement and possible solution for question 2 was as follows:

Problem: " Write a procedure which finds the number of items in a list. For example, the number of items in the list [e,e,s] is 3."

Solution:
```
count([], 0).
count([H | T], N):-
 count(T, N1),
N is N1 + 1.
```

As well as the collection of quantitative measures of student performance, verbal and keystroke protocols were also recorded from students performing each of the three tasks to provide qualitative data, giving a total in excess of 200 hours of recorded protocols. In addition to data from these three tasks, a

written log was kept throughout each course to detail specific issues that influenced students' performance, in particular any ergonomic problems affecting the usability of the TED editor. These were used in part to inform the design at Edinburgh of the final TED interface, but also provide an ergonomic evaluation of the TED interface itself.

Results

The results reported here focus on the program writing tasks, since they indicate whether a techniques approach to writing Prolog programs is beneficial and whether the TED editor facilitates this process. A complete description of the results from all three paradigms is presented in Ormerod & Ball (1994, in preparation). The keystroke logs were codedto identify the major edits made along with edit times and errors (error data are from on questions 1-4 only: only one error of each type was scored for each subject on each question, and errors were recorded only if present when the program was tested). For each problem, a number of measures were examined, from which we focus on three: solution times, technique-related errors and other errors. Additional measures, such as the order in which techniques were added to programs and the time taken to reach a successful addition of a technique are reported by Ormerod & Ball (op. cit).

1. Time to solution: Figure 1 shows mean solution times for each group on each question. Both the TED and Techs-only groups reached solutions faster than the No techs group for questions 1 and 2. There was no clear difference between the three groups' solution times for questions 3 and 4. The Techs-only group reached solutions faster than the TED and No techs groups for questions 5 and 6.

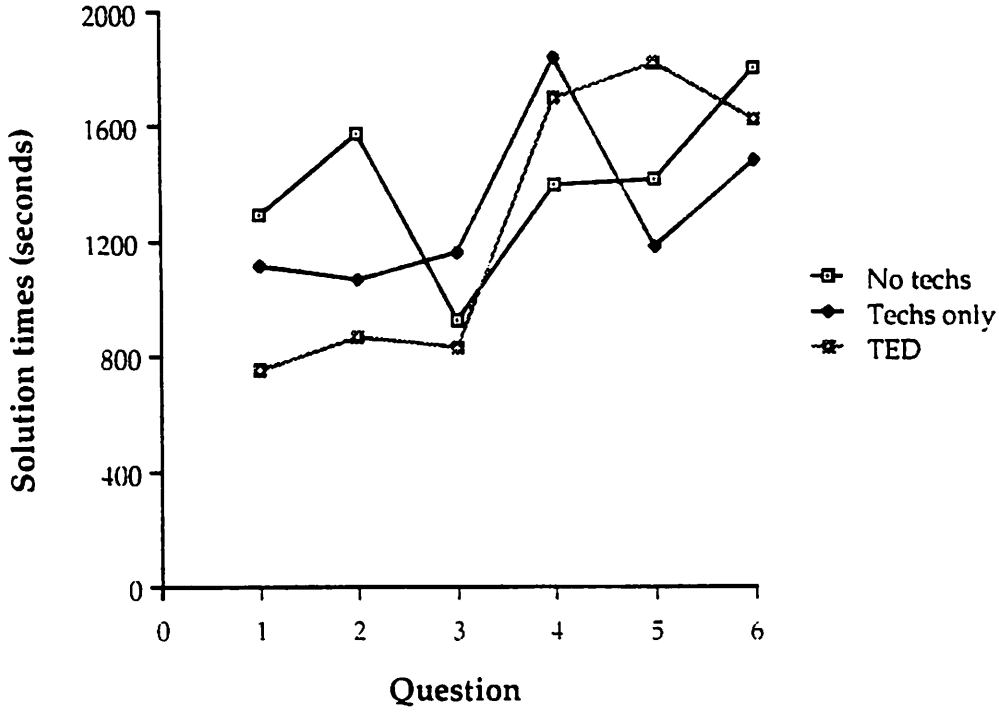

Figure 1. Mean solution times for each question by each training group.

For each problem the following 2-tailed tests were carried out using SPSS: (1) a one-way ANOVA; (2) a post hoc Scheffé at $p < 0.05$; (3) a post-hoc Scheffé at $p < 0.1$; and (4) a post-hoc parametric trend test of linearity in the ordering of means for the three between-subjects groups (linear direction being TED < Techs-only < No techs). Only two of the ANOVAs were statistically significant at $P<0.05$: for questions 2 ($F(2, 28) = 6.7976$) and question 5 ($F(2, 19) = 3.7647$). Post-hoc Scheffés tests reveal that both the TED and Techs-only groups were significantly faster at producing solutions to question 2 than the No techs group. Also, the Techs-only group produced solutions significantly faster than the TED group for question 5. There were significant trends of linearity on question 1 ($F = 3.940$; $p = 0.057$) and question 2 ($F = 12.6065$; $p = 0.0014$). Interestingly, there were significant trends of non-linearity on question 3 ($F = 3.0861$; $p = 0.0912$) and question 5 ($F = 5.6180$; $p = 0.0285$).

2. Technique selection errors: These were explicit in the TED group's protocols and were inferred from the no-techs and techs-only groups' protocols. Table 1 shows the total number of selection errors for each group, ($\chi 2 = 9.35$, $p<0.05$), and the four most common selection errors (frequency of all other errors <4).

Table 1. Total number of technique selection errors for each group

	No techs	Techs only	TED
Same for List_head	9	9	5
Arith_before for Arith_after	8	6	0
List_head for Same	2	8	3
General_after for List_head	0	6	0
Others	5	10	4
Total	24	39	12

3. Other errors: Table 2 shows the total number of errors that could not be counted as technique selection errors for each group, ($\chi 2 = 28.19$, $p<0.01$). A total of 32 discrete error types were identified. These were re-classified according to six broad categories, whose frequencies are also shown in Table 2, along with the most common discrete errors in each category that together account for 49% of all other errors.

Table 2. Number of other errors in each category (most common discrete error shown in brackets: components consist of base case, recursive head or body, head or body arguments, processes/subgoals, and whole clauses)

	No techs	Techs only	TED
Incorrect component	30	35	12
(Incorrect argument in base case)	23	28	9
Unnecessary component	21	23	13
(Unnecessary process)	12	16	8
Typographical	19	19	8
(Mis-spellings)	4	8	3
Recursive syntax	17	21	0
(List notation in recursive body arguments)	7	7	0
Arithmetic syntax	13	16	8
('X is X+1' errors)	8	8	0
Missing component	12	14	6
(Missing disjunctive clause)	6	6	5
Total	112	128	47

Discussion

The error data suggest that students in the TED group made fewer errors, both technique-related and other kinds, than students in the other groups. Second, the efficacy of TED and the techniques approach seems to depend on the type of question that is being undertaken. The strongest evidence for an advantage for TED and the techniques approach can be found on questions 1 (to return a list of squared numbers) and 2 (to calculate the length of a list). These were relatively simple programming problems, in which much of the difficulty seems to be in coding rather than in the conceptual design of a program solution.

The advantage for the TED group is particularly strong with question 2, which requires a non-tail recursive solution. This particular problem is interesting because understanding why the recursion is necessary before the process generally causes students a lot of difficulty (see also Solowaty et al, 1982, for similar findings regarding read-process versus process-read loops

with novice Pascal programmers). The provision of arithmetic_after and arithmetic_before techniques within TED seemed to alleviate this problem, perhaps by making the contrast between them explicit to the students. Although the techniques-only group received equal tuition in the techniques, they did not seem to gain as much advantage as the TED group. This may be because they lacked the explicit reminder of the alternative techniques and therefore resorted to the more conceptually 'natural' arithmetic_before form. The error data seem to support this in that no TED students made an arithmetic_ before for arithmetic_after error whereas half the techs-only group and most of the no techs group made this error (see Table 1).

The absence of a difference in solution times between groups on question 3 ('prefix', another relatively simple problem to code) probably reflects the fact that a related example ('suffix') was given in the course notes that students had available during the session. The difficulties faced by the No techs group in coding their conceptual designs to this problem appear to have been overcome by use of the example. Interestingly, the errors made in this question seem to reflect an over-reliance on the example. In particular, all the errors in which a Same technique (as found in the second argument of the example) was used instead of the required List_head technique occured on this question. On question 4 (to return a list containing the squares of even numbers only from an input list) we begin to see the coding advantages of the TED group being outweighed by other factors. Problems 4-6 present students with tasks that are more taxing in terms of problem understanding and conceptual design. For example, problem 4 necessitates the identification of disjunctive cases (the need for a case to deal with even numbers and another case to deal with odd numbers). TED students made as many 'missing disjunctive clause' errors as the other groups.

Analysis of the error data suggests that use of the TED editor generally reduced the number of errors made by students. This was particularly the case with syntactic classes of error, as arithmetic, recursive argument syntax and typographical errors. There was also evidence of a reduction in errors associated with semantic features of code, such as the presence and correctness of clause components. For example, an incorrect base case arguemnt (e.g. putting [] instead of 0) was the most common error made by subjects, but the incidence of this error with the TED group was significantly lower than with the other groups. There were a few aspects of TED use, however, that created novel problems for students. For example, a common error made by TED group students but not by any other group was an 'X is Y' process statement, casued by the selection of arithmetic techniques without alteration of the edit dialogue boxes.

A number of ergonomic issues emerged during the evaluation of TED. A sample of these is as follows:

- insisting that code is 'undone' rather than 'edited' means that much work is required if a mistake is made at an early stage in program construction. Whilst this feature may encourage students to think

harder about their program before committing it to code, it tends to discourage exploration of possible program solutions.

- when adding arithmetic techniques, subgoals are always added directly after the recursive call. This means that if a series of techniques are to be added in order to carry out a calculation in a particular sequence, they have to be added in reverse order.
- the use of the term 'head variable' in the arithmetic techniques is confusing. It appears that it refers to the variable in the head of the clause, as opposed to the head of any lists. This occasionally led to confusion.
- there is no scrolling on the subgoal field when adding arithmetic techniques (or any of the fields?). This causes a problem when trying to construct programs using sensible variable names.
- when displaying subgoals which include the mod/2 operator, the editor removes the leading space. Whilst this is a very minor problem it did, on a few occasions, cause some confusion.

Of these points the restrictive Undo feature was the source of most problems and irritation. In removing all later edits, it requires the programmer to reconstruct all the edit sequences that contributed usefully to the program. A possible solution to this problem s to restrict the application of program construction histories and restrictive UNDO to technique edits only. In this way, the addition of sub-goals and extra arguments would not be affected.

Conclusions

There is evidence that TED can facilitate the coding of programs, especially where the design requirements for program solutions are relatively simple. In its current form the editor has a few features that cause students unnecessary difficulty and repetitive action, notably the restricted 'undo' editing mode. Nevertheless, students generally coped very well with the TED interface, and found no difficulty in moving round the program window or using the menus.

A useful avenue of further research would be to examine how TED might be extended to support the conceptual design of programs, so that the coding advantages offered by the editor can be maximised. Furthermore, the error analysis has revealed systematic differences in the errors made by each group. The error analysis enables us to to generate a set of re-design rules for future versions of TED. For example, the most common error, that of incorrect base case arguments, might be limited further by linking techniques and base case additions, either as a single edit or as a series of linked edits in which obvious errors (e.g. associating a list in the base case with an arithmetic technique) are trapped.

The project to evaluate TED has been successful, not only in assessing the relative merits of TED and a techniques approach to Prolog programming, but also in showing us the importance of program *design* activities. Much of this has come from our related work with Prolog experts, where we have examined the nature of expert design strategies (Ormerod & Ball, 1993).

Acknowledgements

Th TED project was supported by the Joint Councils' initiative in Cognitive Science/HCI (evaluation at Loughborough University under MRC grant number G9030402).

References

Ormerod, T.C. & Ball, L.J. (1993) Does Prolog programming knowledge or design strategy determine shifts of focus in Prolog programming? In C.R. Cook, J.C. Scholtz & J.C. Spohrer (Eds.), Empirical Studies of Programmers: Fifth workshop. Norwood, NJ: Ablex.

Ormerod & Ball (1994, in preparation). Evaluating TED, a techniques editor for Prolog programming novices. Unpublished manuscript, Department of Psychology, Lancaster University.

Soloway, E., Erhlich, K., Bonar, J. & Greenspan, J. (1982). What do novices know about programming? A.Badre and B.Schneiderman (Eds.), Directions in HCI. Norwood, NJ: Ablex.