COMPUTER SCIENCE

# Developing a Self-Paced Interactive Package for Introductory Computer Programming

V.J. Hobbs and T.J. McGill
*val@cs.murdoch.edu.au*
*mcgill@csuvax1.murdoch.edu.au*

Technical Report CS/94/07

# Developing a Self-Paced Interactive Package for Introductory Computer Programming

V.J. Hobbs and T.J.McGill[*]
*val@cs.murdoch.edu.au, mcgill@csuvax1.murdoch.edu.au*

*Teaching introductory programming can be a challenging task. Students can become too concerned with learning syntax at the expense of more general conceptual understanding. Distance education students in particular often have problems as the difficulty of the course content is compounded by the problems of isolation from other students and their tutor. Although instructional strategies for emphasising conceptual knowlege are well known for face-to-face instruction, there has been little attempt to apply them to the external situation. This paper describes the development of a supplementary package for external students designed to address some of these problems.*

## 1. INTRODUCTION

Distance education in Australia provides opportunities for people who are unable to physically attend a university to pursue their education off campus but make use of the educational resources of the university. Murdoch University has a large proportion of distance education students who take courses in 'external' mode. These students may enrol externally due to their physical distance from the university, or because their work or family commitments make it difficult for them to attend regular classes. External students are provided with specially written guides for self-paced study, sometimes supplemented with other materials such as laboratory kits, audio-visual materials, or dial-up access to university computers.

The teaching of external students in introductory programming courses presents a special challenge. The lack of opportunities for verbal and visual support makes the teaching and learning of programming concepts and of programming-specific problem-solving skills difficult. A project currently being undertaken at Murdoch University proposes a practical solution to the problem. This paper describes the development of an interactive self-paced package for introductory programming to be used by external students.

### 1.1. INSTRUCTIONAL STRATEGIES FOR TEACHING PROGRAMMING

Instruction in first year computer programming courses has often failed to equip students with a sound understanding of programming concepts. Schwartz et al (1989) and Kurland et al (1986) found that after months and sometimes even years of programming instruction, many students still display conceptual misunderstandings and exhibit poor programming skills.

Linn & Clancy (1992) have argued that many introductory programming courses foster the development of syntactic knowledge and do not put enough emphasis on the development of conceptual knowledge and program design skills which are often left to unguided discovery.

[*] Information Systems Programme, Murdoch University

The goal of encouraging students to plan their problem solutions is supported by evidence that expert programmers spend much of their programming time designing and planning problem solutions before coding the program into a specific language, while novices tend to start writing computer code immediately (Petre, 1990).

Linn and colleagues (Linn, Sloane, & Clancy, 1987; Dalbey & Linn, 1985) argue that explicit instruction in program design strategies has a beneficial effect on students' learning, and that the best results are obtained when teachers model their own design processes at a level accessible to students.

This technique has been used at Murdoch University in face to face instruction. An instructional strategy emphasizing expert demonstration and interactive guided practice has been successfully piloted (Volet, 1991). The retention rates of students taught using this instructional strategy were significantly better, especially among the sub-group of students who started the course without any background in computing and among the sub-group of students enrolled in majors other than computing. Students developed a better understanding of programming principles and demonstrated a greater ability to apply their programming knowledge to novel problems.

There is also evidence that expert programmers organize their knowledge of programming in larger conceptual structures than do novices. Expert programmers have been shown to organize their conceptual knowledge using templates or plans which consist of sequences of stereotypic actions which can be adapted as necessary for reuse (Soloway & Ehrlich, 1984). Students appear to have difficulty constructing templates yet studies suggest that representing programming knowledge in the form of short programming templates can help students to write programs (Anderson & Reiser, 1985; Linn & Dalbey, 1985).

## 1.2. PROBLEMS IN COMPUTING DISTANCE EDUCATION

Although advances have been made in face to face instruction of programming little has been done to improve the teaching and learning of programming for external students.

An analysis of the retention rates of students enrolled in the introductory course in computer programming offered at Murdoch University revealed that external students were twice as likely to withdraw (14.9% in 1990-1992) as internal students enrolled in the same course (7.9%). We believe that the majority of students who withdraw have little or no background in computing, as this was found to be the case in an earlier study of internal students (S.E. Volet, pers. comm., 1994).

A survey conducted at Murdoch University (Aveling, Smith & Wilson, 1992) documented the typical problems experienced by external students. In particular, students cited as major handicaps lack of interaction with tutors and other students, lack of immediate feedback, and reliance on written material only.

While the syntax of a computer language can be learnt from books and sample programs, design skills and conceptual knowledge are more difficult to acquire from written materials. Whereas in lectures and tutorials it is possible to give oral demonstrations and to show students how to how to tackle a novel problem, the written format typically used for teaching external students is less suitable for communicating this essential type of knowledge. At the moment, the best that can be offered is individual instruction on the phone, which is not only cumbersome, but expensive. Even explaining the nature of a programming problem to a tutor can be difficult over the phone.

## 2. THE INSTRUCTIONAL PACKAGE

The aim of this project is to apply some of the advances in programming instruction described above to the problems experienced by external students studying introductory programming. Linn & Clancy (1992) state that 'competent programmers need both well-organised knowledge of a programming language and problem-solving skill'. The project is designed to provide instruction and practice in both these aspects.

The instructional package developed for this project supplements and is integrated with the existing course material provided to external students in the introductory programming course. The materials consist of two components. A video and workbook set, called 'From Problem to Program: A Self Paced Tutorial for Introductory Programming', provides an interactive tutorial for program planning and design. A manual, 'Building Blocks for Programming: A Template Reference Book', introduces students to the concept of templates and provides them with a useful reference.

Video was chosen because of its potential to allow students to see the design process modelled and to enable students to participate in that process in an interactive way. Explicit instruction in program design strategies has a beneficial effect on students' learning (Linn, Sloane, & Clancy, 1987; Dalbey & Linn, 1985). However, it is difficult to provide explicit instruction of this type to external students where communication is via the mail or telephone and students are unable to participate in the design process with tutors or other students. In addition, video players are readily accessible and have been identified by students themselves as their preferred option for supplementing traditional study material (Aveling, Smith & Wilson, 1992).

Print-based materials and video were chosen, rather than computer-based materials such as hypertext or multimedia, as external students would generally not have a hardware configuration suitable for running multimedia applications. This would have severely limited the utility of the package. In addition, students would have been required to learn a new software package at the same as learning to use the Turbo Pascal® environment. Linn (1992) discusses some of the problems encountered with using hypermedia tools to teach programming to on-campus students. We anticipated that these problems would be magnified for external students.

A project reference group was formed to provide feedback on the package during its development. Members of the reference group included past students in the course, the lecturer of the course, and an adviser from the external studies unit. Feedback was also obtained from tutors, experts in instructional design and educational psychologists, and incorporated into the final package.

### 2.1. VIDEO AND WORKBOOK

'From Problem to Program' consists of a video and workbook, designed to enable external students to participate in the program design process in a guided way. It focuses on how to design problem solutions, rather than on programming language concepts.

The video contains a demonstration of the program design methodology used successfully at Murdoch University (Volet, 1991; summarised in Figure 1), and several interactive sessions that guide students step-by-step through the design of a number of different programs. The problems are introduced and solutions modelled by a 'tutor', and graphics are used to illustrate the problem definition, algorithms, and structure charts. The entire video lasts for approximately 40 minutes. The workbook contains the same problems as the video, and some additional problems for further practice. The workbook has ample blank space for students to develop their own solutions and also provides sample solutions.

**PROGRAM DESIGN METHODOLOGY**

1. **Problem Specification**

2. **Algorithm Development**
   - **2.1 Top Level Algorithm**
   - **2.2 Structure Chart**
   - **2.3 Final Algorithm**

3. **Coding**

4. **Testing and Debugging**

**Figure 1.** Program design methodology used in the video and workbook

In an interactive session the problem is introduced and the student is then prompted to apply the program design methodology, stage by stage. At each stage the student pauses the video where directed, and works out that part of the problem in the workbook, before returning to the video. The video then provides a possible solution, and continues to the next stage of the design process. The complete solution is also included in the workbook for the student to refer to in detail.

The workbook and video are each divided into three main sections, of increasing difficulty. Students attempt the sections at different stages in their programming course, returning to the video once the necessary syntactical constructs have been learnt.

It is important to encourage students to use the program design methodology from the beginning of their course, to ensure that they possess the necessary program design skills when they encounter more difficult problems. Thus, the first section in 'From Problem to Program' is very simple and can be attempted as soon as students have learnt the structure of a Pascal program and can handle simple arithmetic and input and output. The second section requires understanding of selection and looping. The third section can be completed after students have learned to use text files and arrays.

## 2.2 TEMPLATE REFERENCE BOOK

The aim of 'Building Blocks for Programming' is to provide a manual that encourages the student to develop expertise in finding, using and adapting suitable templates, and which provides a resource that they can build on and refer to throughout their programming courses.

The template reference book is organised into five main template types: selection, fixed looping, variable looping, arrays, and files. Each of these types is described in general terms, illustrating its main features and showing which parts of the template may be varied. Further specific templates illustrate typical variations within each main type; for example, 'Variable Looping' includes templates for 'Validating Input' and 'Processing Until a Sentinel Value is Entered'. There are 24 specific templates (see Figure 2).

**SPECIFIC TEMPLATES**

**Recognising a Value**
**Selectively Updating Variables**
**Error Checking Data**
**Directing Flow of Control Using Nested IF**
**Directing Flow of Control Using CASE**
**Generating a Sequence of Numbers**
**Obtaining a Sequence of Input Values from the User**
**Creating Tables of Values**
**Generating Patterns Based on Counts**
**Processing Until a Sentinel Value is Entered**
**Validating Input**
**Processing Until Some Value is Calculated**
**Text Processing**
**Finding Components in a File**
**Adding Components to a File**
**Updating a File**
**Updating Files from a Transaction File**
**Initialising a One-Dimensional Array**
**Processing All Array Elements**
**Searching Arrays for Particular Values**
**Counting and Grouping**
**Shifting the Location of Array Elements**
**Inserting into an Ordered Array**
**Sorting Items in an Array**

**Figure 2.** Specific templates in the template reference book

Each specific template description contains the sections listed in Figure 3.

**TEMPLATE COMPONENTS**

**Pseudocode**
**Example Problem**
**Sample Output**
**Explanation**
**Pascal Code**
**C Code**
**Desk Checking and Testing**
**Debugging Hints**
**Self Check Exercises**

**Figure 3.** Components of each template description

Each template has a name and a brief description of its purpose. A typical problem that would require the template, and possible output that would be produced from the template code, are also given. These assist students in recognising similar situations where the template might be used.

Each template is represented in pseudocode and the pseudocode is explained further in simple English. The pseudocode layout and the explanation show clearly which parts of the pseudocode are responsible for the template action, and thus how it might be modified to produce a slightly different result.  Shank & Linn (1993) found that multiple representations of a template helped students to understand the purpose and action of a template, and that a variety of representations meant that students were more likely to find a representation that was more consistent with their thinking.

Both Pascal and C code are provided for the template so that the student can apply the concepts of each template (first encountered in their initial programming course in Pascal) to later courses. To ensure students gain a more robust understanding of the template, we also included typical problems that the student might encounter (Debugging Hints) and suggestions for desk checking.

The self-check exercises involve modifying the template slightly to solve a different problem. This reinforces the student's understanding of how the parts of a template can be adapted.

In addition to the self-check exercises for each of the specific templates, general exercises are included for each of the main template types so that the student gains experience in recognising and adapting the most suitable template to the problem at hand.

The sequence in the book progresses from simple templates to more complex ones, (eg. processing all elements in an array using FOR loops) demonstrating how simple templates can be used to build up more complex templates. By the time the student reaches the more complex templates for arrays and files, they should be familiar with using the earlier selection and looping templates.

The student can access a template either by its purpose or by the main Pascal construct used. Early feedback from students indicated that as textbooks are typically organized by syntax, they initially preferred to search this way.

## 2.3. INTEGRATION OF TEMPLATE REFERENCE BOOK AND VIDEO/WORKBOOK

Templates from 'Building Blocks for Programming' that can be utilised in developing the low level algorithm for a problem in 'From Problem to Program' are referenced in the appropriate sections in the video workbook. By creating explicit links between the two books we hoped to help students gain a more robust understanding of the programs they develop while watching the video, and to provide them with an additional 'way in' to the material in the template reference book.

### 2.4. INTEGRATION WITH EXISTING COURSE MATERIAL

The materials are intended to supplement rather than replace the existing course materials. Thus neither the video and workbook nor the template reference book contain details of syntax or programming constructs that would normally be taught as part of the course.

External students are required to have the recommended textbook and Turbo Pascal® compiler, and are also supplied with a study guide that summarises the main topics and objectives of the course on a week-by-week basis. In semesters when the supplementary package is used, the students are also mailed the package at the start of their course. They are provided with a summary that lists the topics covered in the course, week by week, alongside the parts of the video and template reference book that are relevant at each stage.

## 3. EVALUATION

The package has been included as supplementary material for all external students in the semester 2 1994 intake in the target course, and will also be provided for the semester 1 1995 course.

The impact of this instructional package on external students' learning will be evaluated by:

- comparing the retention rates, performance and personal accounts of their learning with those of students taught by the more traditional method during the previous 3 semesters

- eliciting students' own evaluation of the usefulness of the new instructional materials for their learning in the course, in terms of whether and how they used the materials, how useful those materials were to help them progress in the course, and how the materials could be improved.

The package will also be made available to internal students in semester 1 1995 and feedback obtained on its usefulness.

## 4. CONCLUSIONS

Based on the impact of the instructional strategy on internal students, it is expected that its adaptation for distance education will lead to an increase in the retention rates of external students and a significant improvement in the quality of their learning outcomes, in particular their ability to design effective programs to solve novel problems. There is also strong evidence to suggest that it will have a positive impact on students' satisfaction with their learning, and in turn on their motivation for pursuing studies in computer science (Volet, 1991).

Further practical outcomes involve the use of the instructional package by internal students who have difficulties in coping with their study in computer programming. The audio-visual materials could also form a resource for training inexperienced computer science tutors to develop effective teaching strategies.

**ACKNOWLEDGMENT**

This project was funded by a National Teaching Development Grant from the Committee for the Advancement of University Teaching.

## REFERENCES

Anderson, J. & Reiser, B. (1985) The Lisp tutor. *Byte*, 10, 159-175.

Aveling, N., Smith, S. & Wilson, C. (1992) Meeting the needs of isolated students: Is a technological fix the answer? In Parer, Ed. *Academia Under Pressure: Theory and Practice for the 21st Century, Research and Development in Higher Education*, 15, Churchill Vic: HERDSA.

Dalbey, J. & Linn, M.C. (1985) The demands and requirements of computer programming. A review of the literature. *Journal of Educational Computing Research*, 1, 253-274.

Kurland, D.M., Pea, R.D., Clement, C. & Mawby, R. (1986) A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research*, 2(4), 429-458.

Linn, M.C. (1992) How can hypermedia tools help teach programming? *Learning and Instruction*, 2, 119-139.

Linn, M.C. & Clancy, M.J. (1992) Can experts' explanations help students develop program design skills? *International Journal of Man-Machine Studies*, 36, 511-551.

Linn, M.C. & Dalbey, J. (1985) Cognitive consequences of programming instruction: access, and ability. *Educational Psychologist*, 20, 191-206.

Linn, M.C., Sloane, K. & Clancy, M.J. (1987) Ideal and actual outcomes from precollege Pascal instruction. *Journal of Research in Science Teaching*, 24, 467-490.

Petre, M. (1990) Expert programmers and programming languages. In J.M. Hoc, T.R.G. Green, Samurcay, R. & D.J. Gilmore, Ed. *Psychology of Programming*. London: Academic Press.

Schwartz, S., Perkins, D.N., Estey, G., Kruidenier, J. & Simmons, R. (1989) A "metacourse" for BASIC: Assessing a new model for enhancing instruction. *Journal of Educational Computing Research*, 5(3), 263-297.

Shank, P.K. & Linn, M.C. (1993) Supporting Pascal programming with an on-line template library and case studies. *International Journal of Man-Machine Studies*, 38, 1031-1048.

Soloway, E. & Ehrlich, K. (1984) Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*, 10, 595-609.

Volet, S.E. (1991) Modelling and coaching of relevant metacognitive strategies for enhancing university students' learning. *Learning and Instruction*, 1, 319-336.

®Turbo Pascal is a registered trademark of Borland International Inc.