Designing Programming Tasks to Elicit Self-management Metacognitive Behaviour

Angela Carbone, Ian Mitchell, Dick Gunstone, John Hurst
Monash University, Australia

ABSTRACT

In most introductory programming courses tasks are given to students to complete as a crucial part of their study. In this paper, a set of guiding principles is presented for designing programming tasks aimed to elicit metacognitive behaviours in first year students. The main source of data gathered for this study is provided by first year programming students that studied in two different Information Technology degrees.

1. Introduction

This research was aimed at tackling perceived problems in the teaching of first year programming. The main concerns were high failure rates, a low flow of students into higher degrees, and variation of teaching skills. The project team, known as Edproj, gathered data via student interviews, tutor and lecturer discussions, and classroom observations. Comments from a number of tutors and lecturers suggested that they felt metacognitive actions are qualities needed in a university environment. Yet the data gathered indicated that most students were rarely metacognitive in that they did not reflect on what they did and did not understand, and they lacked strategies for improving their learning. The nature of the programming tasks seemed to accentuate this, for example a number of tasks were reported by tutors, to have been solved and completed by copying slabs of code directly from the text book.

The investigation revealed that most lecturers and tutors, preferred students to be metacognitive and use metacognitive actions to learn in a university environment. This research raises the following questions that are of interest and discussed in this paper: *What is metacognition and why is it important? Are there features of programming tasks that influence students' metacognitive behaviours? Can a set of principles for designing programming tasks be devised to induce metacognition in students?*

2. Literature Review

The ability to monitor one's knowledge and learning processes is no trivial matter as far as education is concerned. Space does not allow us to explore the literature that is available, but the following ideas are salient. Students can enhance their performance by becoming aware of their own thinking as they read, write and solve problems. Academics can promote this awareness directly by designing activities to enhance metacognition and learning. The definition of metacognition adopted for this paper is that is metacognition "*captures two essential features- self-appraisal and self-management of cognition*" (Paris and Winograd). Self-appraisals are people's personal reflections about their knowledge states and abilities, and their affective states concerning their knowledge, abilities, motivation, and characteristics as learners. Such reflections answer questions about "*what you know, how you think, and when and why to apply knowledge or strategies*". Self-management refers to "*metacognition in action*," that is, mental processes that help to *"orchestrate aspects of problem solving"*.

Metacognition appears to be a central feature of learning; those who are better managers of their thinking are judged to be better learners, more often metacognitive processes play a major role in learners' successes in acquiring new knowledge. Metacognitive research includes studies that have examined ways in which metacognitive theory can be applied to education. Broadly defined, these studies have focused on a fundamental question - Can instruction of metacognitive processes facilitate learning? The researchers who have contributed to the present volume of literature have responded to this question with a resounding *yes*.

3. Research Methodology

A variety of data collection techniques were used including; semi-structured student interviews, recordings and feedback from tutors at the tutor meetings and written descriptions of students' engagements in the programming tasks. This broad based data was complemented by more intensive data. The two groups of students that provided data studied first year C programming language and Visual Basic. The first student group attended a short course titled "Learning how to Learn". The workshop covered: deep and surface approaches to learning, poor learning tendencies and good learning behaviours. Small numbers allowed for authentic discussions and also

Proceedings of the International Conference on Computers in Education (ICCE'02)
0-7695-1509-6/02 $17.00 © 2002 IEEE

allowed time for each participant to present information about their own learning and engagement in various tasks. The workshops were designed to provide students with frames to analyse and describe their own learning so that rich data could be gathered on complex factors and issues.

No attempt was made to quantify the data, rather an attempt was made to gain a deep understanding of its meaning. The approach taken in this study was to formulate common themes emerging from the students' written stories using the constant comparative method. Reports on codes were then examined and codes merged and revised. The final coding system reflects those themes observed across students' stories and those issues salient to participants. This paper reports on the theme that relate to how tasks can influence students' self-management metacognitive action.

4. Results

The results presented below are organized under the self-management aspect of metacognition described above, reflecting the focus of this paper.

Planning and Linking Some students attempt to understand what the problem is about before hastily attempting a solution. This may also involve planning how they should proceed or taking on a particular course of action. Students indicated that this promoted positive self-perception, and motivated their learning.

A feature of the domain of programming is that programming is extremely cumulative, in that there is no opportunity to "shelve" a tough topic. Aspects of one lesson are usually required in the next. This implies that for students who have understood earlier work, planning a solution becomes easier. However, for students that remain confused or stuck on a concept, planning for the following task almost becomes impossible.

Awareness of a time pressured environment In most programming units students have difficulty managing their time. Despite efforts to manage time, many students saw the demands of the tasks and pressures to handle events outside the capabilities chew away their time. This usually meant that not enough time was left for students to build an understanding of the concepts. With many programming tasks, it is common for students to spend much of their time debugging their code. Some make conscience decisions not to devote large amounts of time to this aspect of programming. Yet others are unable to continue unless their code is bug free. Often, pass grades are only given to those that demonstrate a working piece of code; otherwise, they are at risk of failing. This pressure to demonstrate working code within a limited amount of time available often causes students to miss valuable lessons spent in the process of debugging, to produce working solutions.

A significant dimension of programming that needs considerable attention is the dependence on the technology that often results in time-consuming problems that cannot be managed by the programmer. An inevitable part of working with new and ground breaking technology is that often the equipment may be faulty, the software is not behaving as it should, or unpredictable network failures occur. Again students see this as a source of frustration, not beneficial to their learning experience, and something that is beyond their ability to manage and control.

What seems particularly interesting about programming is that no matter how well students plan to manage their time and effort when working on a programming task, they are often confronted by unexpected technical difficulties, which they have little or no control over. It is common for academics teaching programming to feel the need to set tasks that require coding and the use of a computer, in fact not all activities should require the use of a computer to master concepts. Many students spend time debugging programs that have not been clearly thought through because of their haste to *'get on the computer'*. Given that this may be the case, academics need to think about appropriate strategies that encourage students to reflect what they have learnt from their experiences rather than superficially completing the task at hand.

Monitoring and Reflection Often novice programmers who find planning difficult continue to work on a problem even if they are unsure how to proceed. This makes monitoring or keeping track of how well they are going difficult, because they do not really know where they are heading. However, when they arrive at the right outcome they need to reflect on their actions.

5. Conclusion: Guiding principles to develop metacognitive skills

This research explored factors of programming tasks that influence the process of metacognition: reflection, exploration and linking. This paper focuses on those issues surrounding characteristics of tasks that influence certain metacognitive actions and learning behaviours in students. The results presented in this paper are largely derived from the thematic analysis of the written descriptions provided by students of their engagement in the programming tasks. These results provide a useful starting point, for producing a set of principles for designing programming tasks to encourage the use of metacognitive skills in students.

Overall, the impression one obtains is that the tasks can influence the metacognitive behaviours of our students. These insights along with an understanding of the student's learning situation are useful in devising guidelines for designing programming tasks that academics can apply.