An exploration of internal factors influencing student learning of programming

John Hurst

An Exploration of Internal Factors Influencing Student Learning of Programming

Angela Carbone[1], John Hurst[4]
Faculty of Information Technology
Monash University, Australia
61 3 9903 1911
`Angela.Carbone@infotech.monash.edu.au`
`John.Hurst@infotech.monash.edu.au`

Ian Mitchell[2], Dick Gunstone[3]
Faculty of Education
Monash University, Australia
61 3 9905 2857
`Ian.Mitchell@education.monash.edu.au`
`Dick.Gunstone@education.monash.edu.au`

Abstract
This paper explores internal factors influencing student learning of programming. This is based on literature relating to student learning and learning of programming. Two dimensions: *motivation* and *capability* are used as a framework to explore the data gathered from a study of first year undergraduate IT programming students at an Australian University. The authors propose that the exploration conducted in this study is useful in assisting academics with developing tasks to facilitate student motivation and skills to learn programming.

1 Introduction

Literature relating to learning suggests that there are many factors that influence cognitive engagement and the learning process. This includes the individual, the learning environment and the tasks set. The problem of researching learning is that each of these factors is regarded as important, with all having an immediate impact. Consequently, studying any one of them in isolation from the others carries obvious risks. On the other hand, it is difficult, if not impossible to discuss all of them at once. Our research does explore all these factors, but this paper concentrates on one of the major strands, that is, individual factors that influence student learning of programming. This paper is part of a broader study that attempts to pull together information in this field in a comprehensive manner that is integrated into the broader literature on learning.

The assertion made in this paper, is that there exists an individual domain which encapsulates personal factors that influence student learning of programming, and that these factors can be described by a student's motivation towards engaging in the learning activities and his/her capability to complete these activities.

The body of this paper will tease out some of the issues relating to student capability and motivation required to learn to program. Of course, other factors such as the learning environment and the task contribute to the students learning, and once these are fully understood, one can examine the interplay between the set of internal and external influences on student learning of programming. However, the external influences, although explored in the broader study are beyond the scope of this paper.

2 Literature Review

There are many factors that influence cognitive engagement and the learning process. Biggs (1987) groups these factors into student based factors, teaching based factors and the system as a whole. Helme and Clarke (2001) frame these factors affecting cognitive engagement as; the individual, the learning environment and the learning tasks.

The focus of this paper is on developing a framework outlining factors related to the individual, which can be used to promote programming capability. With regard to the individual, Helme and Clarke (2001) state that

> "students need to have both the will (motivation) and the skill (capability) to be successful learners. It is the experience of teachers that students who are motivated to learn and who think carefully about what they are learning develop deeper understanding of the material being covered." (p136).

They also state that "the individual brings to the learning situation numerous characteristics that influence their cognitive engagement. These include: skills, knowledge, dispositions, aspirations, expectations, perceptions, needs, values and goals" (p138) (Helme & Clarke, 2001).

Both studies suggest that personal or individual factors can be described and grouped according to student motivation towards engaging in the learning activities and student capability in completing these activities unspecific to any context. The studies reviewed next therefore consider the types of skill and motivation needed to learn programming. Although these issues are the primary focus of this section, most studies investigated the relationship between academic performance and the students' personal attributes or predisposition factors (Katz et al, 2003; Wiedenbeck, 2005).

One multi-national, multi-institutional study that investigated the programming competency of first year tertiary students found that "many students do not know how to program at the conclusion of their introductory courses" (McCracken et al., 2001, p. 125). Other studies have attributed programming success to factors such as the student's background in maths and science (Byrne & Lyons, 2001; Wilson 2002); (Bergin & Reilly, 2005); learning styles and problem-solving skills (Beaubouef, Lucas, & Howatt, 2001); (Goold & Rimmer, 2000); (Haden, 2006); prior academic experience; self-perception; and specific cognitive skills (Bergin & Reilly, 2005). Other predisposition factors that have been reported in the literature include spatial visualisation skills and map drawing styles, which both had a significant correlation with marks (Simon et al., 2006; Simon et al., 2006; Tolhurst et al., 2006). However, the most frequently mentioned factor for success in programming is previous programming experience (Bunderson & Christensen, 1995; Byrne & Lyons, 2001; Diane Hagan & Markham, 2000; Ramalingam, LaBelle, & Wiedenbeck, 2004; Taylor & Mounfield, 1994; Wilson & Shrock, 2001). These factors are listed to indicate the broad range of issues that researchers have found to influence learning.

In terms of specific skills needed by students to learn programming, fewer studies have been conducted. One comprehensive study by Caspersen (2007) suggests that students need a number of capabilities to become competent programmers. This is because there are a number of steps involved in writing a program. These include: reaching an understanding of the problem to be solved and its solution space; translating the solution into computer terms; testing and debugging then analysing and reflecting on the result (Winslow, 1996). As a consequence, students need many capabilities.

One such capability includes the ability to close track code (Perkins, Hancock, Hobbs, Martin & Simons 1986), yet studies have show that many students are weak at tracing code (Lister et al., 2004) or do not have the willingness to do so (Denny, Luxton-Reilly & Simon, 2008). Others include: the ability to tinker with code effectively, break down a programming task into sub-problems (Perkins, Hancock, Hobbs, Martin and Simons 1986); problem solving skills (de Laet, Slattery, Kuffer, & Sweedy, 2005; de Raadt, Watson, & Toleman, 2006); and the ability to work in a team to solve problems and write programs in collaboration with others (Daigle, Doran, & Pardue, 1996; Hagan, 2004).

Motivation has been found to affect students' learning progress (Helme & Clarke, 2001; Jenkins, 2001). Motivation is an abstract concept and there are a number of theories about it. These include: need and drive theory, which suggests people are motivated by their needs to develop and achieve to their fullest potential and capacities (Maslow, 1962); trait theory, which suggests that motivation is a personality trait of the individual (Kassin, 2003); and other cognitive theories that seek to integrate individual characteristics, job characteristics and organisational characteristics. These theories of motivation are not developed or investigated in this paper. Instead this study determines the type of motivation that students display when working on programming tasks using the concepts intrinsic, extrinsic and achieving motivation (Entwistle, 1998). Intrinsic motivation is derived from a personal interest in the subject. Deci (1975) states that intrinsic motivation is evident when an activity is performed for its own sake and out of interest and curiosity. Extrinsic motivation refers to the desire to complete the course in order to attain some expected reward. Achievement motivation is based on doing well and sometimes performing better than peers (Ryan & Deci, 2000)

The broader study investigates those characteristics of programming tasks that might cause a student's state of motivation to change.

3 Data Collection

This paper reports on a set of individual factors that influence first year undergraduate ICT students learning of programming. The factors are derived from a much larger study that investigated characteristics of programming tasks that influence student learning. These factors are used to construct a framework describing the individual domain that influences student learning of programming. This section reports on the context and design of the study and the participants.

3.1 Context of Study

Data was collected from undergraduate ICT students studying first year programming at Monash University. Two groups of students participated in this study:

- students enrolled in a Bachelor of Computer Science degree studying CSC1011 Introductory Programming (semester 1), and CSC1030 Data Structures and Algorithms, and Computer Systems (semester 2), and
- students enrolled in a Bachelor of Information Management and System (SIMS) degree studying IMS1000 First Year Studio (full year).

CSC1011 provided students with a general introduction to computers and covered basic programming concepts in the C programming language. CSC1030 comprised two components: developing solutions to more complex problems using sophisticated algorithms and data structures; and computer systems, which involves programming using a lower level machine language called MIPS.

IMS1000 was a full-year unit in which students studied Visual Basic programming for six weeks of the 26-week programme. The focus of IMS1000 was on *IT Tools and Technology*. This covered introductory programming concepts, using the Visual Basic programming language for implementation. Other topics were also covered: information management, system analysis, knowledge management and computer systems.

3.2 Project design

Data were gathered from students using the following methods:

i) Semi-structured interviews;

ii) Written descriptions of students' engagement in a task;

Semi-structured interviews with students

The first data collection method used a sample of eight students (four male and four female) from a cohort of 315 Computer Science students who volunteered to participate in the study. The students were guaranteed anonymity, to increase the candidness of interviews and quality of conclusions (Miles & Huberman, 1994), p. 277). All participants were under the age of 20, with programming results ranging from Pass to High Distinction.

Semi-structured *conversational* interviews (Patton, 1990) were used to determine how students perceived the tasks. The eight students were interviewed on a weekly basis for approximately 30 minutes. Students were approached two hours into the laboratory session. The timing of the interview was crucial as current issues and problems that students faced would be at the forefront of their minds and so could easily be captured. The interview was conducted using a conversational style or open approach to questioning. The students were asked to answer some specific questions as part of the interview, for example, "So you are studying programming at university; tell me, what was your first impression of this programming exercise?" Follow-up telephone calls and emails were used to clarify details where necessary. Narrative-analytical accounts (Bassey, 1999) of each student's engagement in a task were transcribed and analysed from the interviews. Once compiled, accounts were returned to the students to be approved and corrected as part of the consent process and to verify that the account was an accurate reflection of what had happened.

The sense during interviews was that participants felt pressure to return to the laboratory to finish the task, which limited the time they had to think and respond thoughtfully. Although some inferences could be made about the sorts of issues affecting students' learning of programming by scrutinising the qualitative aspects of student responses, students did not really elaborate on their learning. This was a problem with the interviews. Students did not elaborate because of time restrictions or because they lacked the vocabulary to describe their engagement in the tasks. Consequently an alternative data collection method was sought to stimulate their thinking about their learning.

Written descriptions of students' engagement in a task;

The second data collection method, which then became the main data collection method, sought data from students via cases in the form of self-reports. Seven CSC1011/CSC1030 students volunteered and all seventy IMS1000 students submitted two written descriptions about their engagement in a task as part of the unit requirement.

Self-reports are a primary source of data in the social sciences. Researchers rely on such reports to learn about individuals' thoughts, feelings and behaviours and to monitor societal trends (Schwarz, 1999). Self-reports are also used to infer cognition, as it is not readily observable (Fredricks, Blumenfeld, & Paris, 2004). Shulman (1997) pioneered practitioner-written casebooks as a professional development tool for teacher educators and staff developers. She describes cases as detailed scenarios written about the real-life experiences of teachers or administrators. Cases reflect reality: they help teachers learn to connect theories and concepts to the complex, idiosyncratic world of practice. The use of cases in this study was to provide students with an opportunity to reflect upon, inquire and analyse their behaviours. The students' stories provided a rich data set.

To help students write self-reports they were invited to attend a workshop titled *Learning how to learn.* The workshop had three broad aims: (1) to make students more aware of their learning; (2) to provide students with a conceptual framework to help them analyse and describe their own learning; and (3) to develop a rapport between the students and the researcher (myself), which would encourage a more thoughtful, reflective exchange of questions and answers. These aims were directed to helping students express and write about their learning, and engagement in tasks. In particular, students were asked to write about their engagement in tasks that left them with powerful impressions of either successful or unsuccessful learning. Their stories also provided insights into their perceptions of other issues that affected their learning of programming.

3.3 Data analysis and reporting

Data was analysed according to the three broad based themes as described in the literature: the individual (internal domain), the learning environment (external domain) and the programming tasks. The internal domain encompasses personal factors unique to the student, the external domain encompasses factors outside the student's control and programming tasks focuses on the characteristics of tasks that influence a students learning of programming. The programming tasks and external domain are not discussed in this paper.

Student quotes were coded into categories that represented the Poor Learning Tendencies and Good

Learning Behaviours (Baird & Mitchell, 1991; Baird & White, 1982b; Baird & Northfield, 1995). Each category was then investigated to determine what other factors, other than the task, might have influenced student learning. These were coded according to *individual* factors and the *learning environment*. Both of these broad categories (*individual* and *learning environment*) were then analysed and recoded into sub-categories. This categorisation is pictorially represented in Figure 1.

The first-level sub-categories for *individual* factors that influence student learning were divided into *student motivation* and *student capability.* Second-level sub-categories were generated from common themes that emerged from the data itself. As these sub-categories were generated or "grounded" in the data provided by the participants, this part of the analysis used a grounded-theory approach. These sub-categories were expanded and modified as the data was carefully compared against them. The *constant comparative method*[1] (Glaser & Strauss, 1967) was used as the analytical approach to capture common themes across the data.

In the next section when data is reported a specific method of notation is employed. For example, to denote a student's written descriptions of their engagement in tasks the notation *CS.1a* stands for *written description (a) by student 1 in the first year computer science degree studying C programming or MIPS*. If students have written multiple descriptions, each description is labelled *a, b, c* etc consecutively. Any information that may have identified participants has been replaced by a suitable description.

4 Research Findings

The data were analysed to identify

i) causes of shifts in students' motivation, and
ii) a set of skills needed by first year students.

The following sections explore motivation and skills, both technical and generic, which emerged from the data.

4.1 Motivation

This section examines the motivation students displayed when working on programming tasks, using the concepts of intrinsic, extrinsic and achieving motivation.

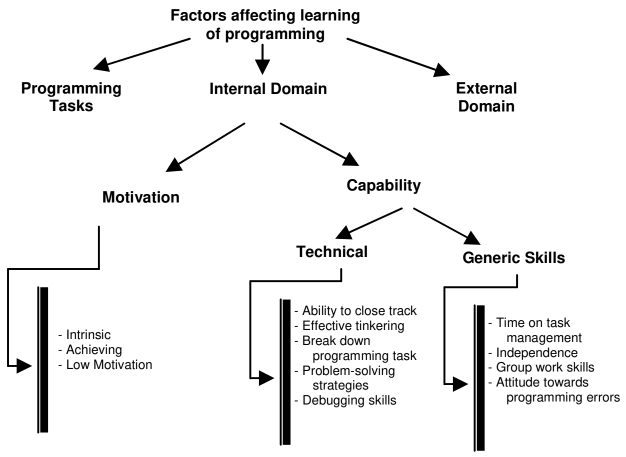

Figure 1 Representation of Internal Domain

4.1.1 Intrinsic motivation

In the programming units studied there were students who exhibited intrinsic motivation. These students usually undertook to learn programming in their own time, sometimes prior to the course commencing, working hard at developing their skills. They usually had prior programming experience, providing them with a foundation that enabled them to grasp another programming language easily; they possessed a repertoire of programming skills; and demonstrated a mastery of general programming techniques. Their familiarity with fundamental programming concepts allowed them to concentrate on learning new programming paradigms and language syntax, rather than having to master those basic concepts.

Students displaying intrinsic motivation generally displayed higher capability. These students could apply information presented in lectures to practical problems, and to new situations. They engaged in programming meaningfully and would apply what they had learnt to real life problems, suggesting that a deep learning strategy had been adopted. These students demonstrated a persistence to play around with the code and used alternative resources, not just lecture notes. They played around with the code by making a series of small changes and tests and approached problems methodically and reflectively as compared to others, who approached their work in a more trial-and-error or impulsive fashion. They persevered with compilation and run-time errors, and showed an obvious desire to experiment beyond the requirements of the task. They were also able to overcome difficult challenges.

These students generally worked harder and invested significant amounts of time to complete the task. They often explored different resources and persisted to learn something from the process. Students who adopted such strategies to maximise their understanding repeatedly spoke about a sense of personal achievement and satisfaction when they managed to get their program working.

[1] Constant comparative method is a strategy used to analyse interviews. Four distinct stages are used by Glaser and Strauss (1967). These include: (i) comparing incidents applicable to each category, (ii) integrating categories and their properties, (iii) delimiting the theory, and (iv) writing the theory.

4.1.2 Achieving motivation

This section reports on students driven by *achieving* motives. Achievement motivation is based on doing well and adopting whatever strategy is needed to secure the best results in the form of the highest marks.

Some students referred to achievement as their main motivator in their written descriptions, either to achieve a pass, or to do as well as they could. The prominence of this type of motivation could have resulted from the nature of the data collected, that is, being based on short-term outcomes, for example, how the task influenced a student's immediate learning. Excerpt CS.4c provides an example of a student driven by the possibility of gaining a high grade and excerpt CS.3b provides an illustrative example of a student wanting to pass.

> The thought of a HD on my end of year report spurred me on. Unfortunately, what one wants and what one gets do not always coincide. Several more attempts at implementing the parts of the questions which I was stuck on resulted in little. Time was running short and I had many other things to do … [Excerpt CS.4c]

> I don't feel like I learnt anything of value, except that MIPS is incredibly frustrating. I knew how to do the prac — I could see it ticking over in my head, but I at least needed to get it working to get a pass grade. That was also frustrating — all of my energies went toward "marks" instead of "learning". It would have been an invaluable prac, but as it was, I walked away with naught but a headache. [Excerpt CS.3b]

These examples illustrate how high levels of motivation are not necessarily associated with high levels of cognitive engagement. This finding is supported by similar observations made by Blumenfeld et al. (1992).

4.1.3 Low motivation

Few students indicated that they had low motivation. The low number could have resulted from the method of data collection. The computer science students volunteered to participate in the project, and none of the participants reported or showed signs of lacking motivation. The students studying Visual Basic were required, as a compulsory part of the unit, to write about their engagement in tasks and it was in these descriptions that the students with poor motivation were revealed. Excerpts VB.16 and VB.63 provide two examples of students explicitly stating that they lacked motivation to engage in a programming task, which was purely based on a lack of interest in the unit.

> As I've foretold earlier, I don't have a great interest in VB that didn't make me strive harder to learn it 100%. [Excerpt VB.16]

> I was too distracted and lazy to concentrate on the learning task at hand. Independent learning requires us, as students to want to learn and not be forced by others to learn. Maybe I still thought I was in high school, and expected the teacher to come around, instructing, "Get to work! Where are you up to? What? You should be up to here by now … " [Excerpt VB.63]

4.1.4 Changes in motivation

An interesting insight that emerged from this study was that students could start off intrinsically motivated to learn, but then, while working on a task, experience a change in their state of motivation. Students could shift from being intrinsically motivated to achieving or low motivation, or move from a state of low motivation to one that was intrinsic or achieving in nature.

In excerpt CS.2a the student originally put extra effort into the task, but since he wasn't rewarded for the extra effort, he adopted a purely achieving learning approach in which his sole aim was to acquire the marks.

> It was the first prac I had for CSC1030 and I put extra effort into it only to find that that wasn't required ... since that time prac, I have just done the pracs with "marks" on my mind instead of trying to make it more efficient and user friendly. [Excerpt CS.2a]

In excerpt CS.5b, the student finds dealing with a bug in the simulator too much of a challenge, and her state of motivation changes. Unfortunately, by the end of the laboratory session, the student had lost the motivation to spend any further time on the task, even after the lecturer had granted her an extension.

> It really burns when someone tells you, you should have done something and you didn't, and you know you should have done it. A bad mark is even worse. And that's what I got for this MIPS prac I had gone to the lecturer just a little upset. Yes, she said, I had encountered bugs in the simulator, and she offered me some more time to complete the prac, but I don't think I could have finished it anyway. All bugs and time wasted aside, once I had walked out of that prac room, I didn't want to know about it. [Excerpt CS.5b]

A more serious concern than motivation changing from an intrinsic state to an achieving one is when motivation shifts from high to low motivation. This is illustrated in excerpts VB.63 and CS.1b. Students can lose interest in what they are doing quickly, and this is usually because they have encountered difficulties in the task, or have no way of proceeding. They can move into state where they no longer wish to continue working on the task, and at times this can lead them to disrupt others, as in the case of excerpt VB.63.

> The learning process was hindered in this instance because once I realised that I was having difficulty completing the task; I decided to spend the studio time talking. This was a big mistake, because the three hours in which I could have attempted to learn something about VB applications turned into something of a gossip session. Reflecting back on the time, I now realise that this is a powerful

> impression of unsuccessful learning as I was also disrupting other students who may have wanted to seriously complete their studio activity. [Excerpt VB.63]

In excerpt CS.1b, the student encounters a problem, and because he is unable to fix it, ends up abandoning the task rather than persisting in order to fix it.

> I had no idea how to fix this problem, and as we only had fifteen minutes left in the prac it didn't look likely that I would be able to solve the problem. So I left. [Excerpt CS.1b]

Just as students can move into a state of low motivation, they can also move into a motivated state. The data suggests that once students acquire the necessary programming skills and develop a familiarity with the programming language, they can become motivated to learn more. So their motivation shifts and feeds on their success. Excerpts VB.37 and VB.39 illustrate this point.

> However, once I got the hang of coding it became very addictive. I could not go to sleep if I didn't get the program running the way it should. Hence I gradually developed an interest that I never thought I would. I also became very interested in coding every exercise in the studios. [Excerpt VB.37]

> The entire impression of successful learning only really took full effect after I had completed the CADAL Quiz and reviewed my results. With persistence and consistent hard work, I had managed to score a perfect mark on the test. Although I did refer back to the notes of the seminar, I was able to remember and consequently learn a lot of the material covered, which I put into practice during the quiz. Even … as delighted and happy as I was it is only now after being presented with such a question, do I seriously realise how that simple activities coordinated and directed my positive impressions and motivation to learn Visual Basic. [Excerpt VB.39]

4.2 Capability

4.2.1 Technical Skills

The technical skills that students lacked are discussed below.

Inability to close track

Students didn't sufficiently track their code to localise problems accurately.

> I started to do it, designed the interface for the VB application, and wrote the codes for the simple ones... When I wanted to use the Do … Until to write the coding, I met the problem, I notice that when I running the program, was not be looped as I wished. So I knew the coding that I wrote maybe wrong, then I rewrote the code more than fve times, but it still got the same problem.
>
> I had no ideas by myself, time gone so quickly, and just left one hour to finish. So I found the tutor and asked him how to solve this problem, and the tutor so kindly and tell me what was wrong for my coding, but he did not help me to rewrite the coding, and gone away. [Excerpt VB.19]

As in the case above, the student attempted to correct the code over five times but each repair was incorrect so the student was unable to isolate the problem and complete the program on his own. As a result, he sought assistance from the tutor who was able to track the problem and advise him on how to proceed. Although students may demonstrate an eagerness to tinker with the code, they fail to track errors, and rely on tutors to do that for them.

Ineffective tinkering

> I had no idea where my program was wrong; all I could do to fix the program was try as many different things I could. [Excerpt CS.1a]

Students often lack the ability to tinker effectively with their code, producing error after error, and adding more and more errors every time they make a change to the code. Instead of rethinking how they are solving the problem and question their understanding they continue to make minor changes to correct the code. These students tinker without sufficient tracking and therefore have little grasp of why the program is behaving like it is.

Inability to breakdown a programming task

Students in this study did not to recognise the need to break a programming problem into parts. Data shows that students only considered breaking down the problem and testing it in parts when the idea is suggested by the demonstrator.

> Help was required. I asked my demonstrator to check over the code, see if they could find any obvious errors, but they just scoffed when they saw the mess in front of them. "Why didn't you break it up into sections?" they asked. "You start with a small piece, test it, see if it works and then move on. There could be a million things wrong here". .My eyes began to burn at the thought of restarting. Instead, I attempted yet again to debug. Hours later, which turned into days later the problem had not shown itself. All I knew was there was a problem with one of the loops, but which one and where? [Excerpt CS.4b]

When students did have to break down problems in nontrivial ways, they often faltered. Perkins et al. (1986) claims that this strategy may be feasible for expert programmers because they have at their disposal a well-developed repertoire of programming plans for different chunks of the programming task. However, for the novice, with a scanty repertoire of programming plans, this often leads to an unworkable breakdown of the problem. (p51)

Lack of problem solving skills

Many students would dive straight into the coding part of programming without a clear understanding on the problem or a systematic strategy to solve it. Students commonly would work out a solution to a problem by trial and error, without a plan, trying to solve their problem haphazardly, and relying totally on feedback from the compiler to correct their errors.

> We usually would work out a problem by trial and error. That would involve responding to the many error messages we would come across and trying to find the problem by looking back at the lecture notes and if possible previous VB exercises. [Excerpt VB.1]

In most cases, the compilation process produces a series of error messages highlighting the incorrect use of the syntax of the language. The compilation errors can be eliminated by an iterative process of modifying the code and recompiling, yet students found this process overwhelming especially when they are confronted with many errors. Once the compilation is successful the student runs their program to see if it executes. Sometimes further errors are encountered during the execution of the program; this may be the result of a logical problem (ie. divide by zero error or the actual logic of the flow of control is incorrect) or runtime problem.

Limited Skills in Debugging

The common practice for students was to type their whole solution, without any testing along the way. In one case, the student's program produced a series of syntax errors during compilation consisting of multiple "bad address" error messages during execution.

> I loaded the whole program to firstly weed out the obvious syntax errors. I then, optimistically, tried to run it. Who doesn't check to see if its going to work first go?
>
> "Bad address at …"
>
> "Bad address at …"
>
> just kept scrolling down the screen. I couldn't see what was causing this and neither could the demonstrator. I spent ages by myself stepping through trying to find this bad address, but I couldn't see what was wrong. [Excerpt CS.5b]

This students, was a typical case, lacking basic debugging strategies and skills in using the debugger to help complete the task.

4.2.2 Personal Skills

The personal skills that students lacked are discussed next.

Poor Time Management

Despite efforts to manage time, many students failed to produce a complete working system within the time constraints. They spent all of their time on one aspect of the problem and many were unable to progress until they achieved a fully working, bug free, solution. Initially students wanted to build an understanding of the problem at hand however, once this became an excessively time-consuming activity with limited success, students opted for strategies that limited their learning, instead of re-evaluating their plan of attack, and assessing what they have learnt or whether they should continue.

Students spoke about "running out of time" and/or the pressures of having to work within a restricted timeframe. Students driven by achieving high grades prioritised their activities, making conscious decisions about what programming activities they would or would not do.

> I had little patience to spare at the moment with exams approaching like a mad dog to a piece of meat and ten billion other projects/ assignments/ pracs to complete – none of which were easy, mind you. I decided to do the best I could on this prac, but I would not devote any large amounts of time to problem solving or de-bugging – I just could not afford to at the moment. [Excerpt CS.4c]

To save time, students restricted the type of learning approach they would take. As in the case below, the student saved on time by copying slabs of code directly from the text book.

> After a good fifteen minutes, the underlying sense of it all was not sitting quite nicely, as I would have hoped. The sample code seemed to be making sense, but I was struggling to gain an overall picture. I considered spending more time looking at my notes. But time was running short and I had a prac to finish. In the hope that my actually implementing the code would concrete the concept, I dove straight into it… but hit the ground very quickly. Now I was getting impatient. "That's it", I thought, "I will do this the crude way. I will copy the notes." So that is precisely what I did. [Excerpt CS.4b]

Interviews with students revealed that students who copied slabs of code to complete their task didn't search for a deep understanding, they were happy to apply their attention superficially and succeed regardless.

Independence

A common theme emerging from the data was the tendency of students to seek assistance from friends when they ran into difficulty tackling a task. Assistance was provided by friends who were commonly in a senior year level having more programming experience. Although, friends acted as valuable resource, vital in getting students out of situations when they became stuck, students would rely and act on their friend's suggestions, regardless of whether or not they understood the consequences of their suggestion.

> When my friend told me to write a piece of code, I did it. When the code involved pointers, I leant heavily on his suggestions, making little effort to really figure out their background or implications. There was just not enough time, I told myself. The code was working and that was all I cared about. [Excerpt CS.4a]

Sometimes, students seeking assistance from friends were left feeling rather inadequate, or in an utter state of confusion.

Groupwork skills

The productivity of the team, in part, depends on students' interpersonal skills. The group dynamics can improve via their social interactions (communication, consultation and meetings) yet, there are risks involved when students lack the skills needed to contribute to the team.

> During the group activity, I was a bit shy, didn't participate a lot and I let my group members do all the work. [Excerpt VB.26]

Group members who fail to participate in group discussions inhibit the facilitation of social interactions that are necessary for successful learning.

Attitude towards programming errors

Students showed a variety of attitudes to handling programming errors. It was common for students to deal with errors by either stopping altogether and moving onto the next problem or repeatedly trying something different without reflecting on what had been done. These types of strategies are referred by Perkins et al. (1986) as *extreme stopping* and *extreme moving* respectively, with the students labelled as *stoppers* or *movers*.

As one example, the student in excerpt CS.4b illustrates the case of extreme stopping. The student did not know the answer to the problem and was unwilling to explore it any further, so promptly moved onto to the next task.

> Almost instantly, another problem hit me. How on earth was I suppose to implement that in MIPS code? "Stuff it" was my almost definite answer, and moved promptly onto the next problem of the "a[i]". [Excerpt CS.4b]

In the next case, the student never for a moment stops to reflect on what has been done. The students goes around in circles, retrying approaches that have already proven unworkable, yet instead of dealing with the mistakes and the information they might yield, ignores them and continually moves on to keep themselves busy without pausing to think about what might be causing the problem at hand.

> I was going around in circles. Maybe if I run it again it will work …? I'll just run it another time to see if it will work … I had got to the point where I wasn't doing anything constructive. [Excerpt CS.5b]

Perkins labelled students as stoppers or movers. However, this study revealed that students can behave differently in when working on different tasks. The important point is that students sometimes do not see their error messages as vehicles for insight or learning. When students adopt an *extreme stopping* type behaviour they give up all hope of progressing and are reluctant to make any further changes. This has implications for their motivation. *Extreme moving* should also be criticised as in this case students are not reflecting on the meaning of the errors generated, and/or are failing to interpret the errors correctly, thereby making changes without any systematic plan. At least these students are still motivated to try different things, although not behaving metacognitvely.

5 Discussion and Conclusion

Motivation appears labile, that is, it moves easily and is sensitive to other factors. Reasons that explain why students start with an intrinsic motivation, and then change, are offered by Jadud (2006). Throughout this paper, examples have been presented of how students moved between different states of motivation. Students who were intrinsically motivated to learn could move to an achieving state, depending on the challenges they faced. It appears that to sustain intrinsic motivation there needs to be some success achieved otherwise students may resort to a focus on marks or other signs of external achievement.

In this study there were very few students who spoke about having low motivation. Most students wanted to learn and to understand. Rather, the problem was the skills they lacked to complete the task at hand, which dampened their motivation. The lack of skills influenced motivation in a negative direction and the presence of programming skills appeared to increase motivation. It is the academic's role to exemplify, nurture and facilitate that desire and to move motivation in the "right" direction.

Deficiencies in five technical skills emerged: the inability to closely track code; ineffective tinkering; the inability to breakdown a programming task; lack of problem-solving skills; and limited debugging skills. The generic personal skills that students lacked were time management; working independently; group work skills; and a positive attitude towards dealing with programming errors.

Three of the five technical skills — tinkering, close tracking and debugging — are interdependent. If students can close track their code, localising their problems and understanding the issues they can make informed changes. As a consequence, their modification to the code would not be a tinker but an informed and well-planned action. However, where students localise the problem but do not fully understand it, a change to a code would be considered a tinker, because the effect of the change on the output could not be predicted. At least if students tinker they have not given up altogether, but in order to tinker, students have to want to make changes, that is, they need the right attitude to deal with programming errors and to be motivated to do so. Although motivation plays an important part in preventing students from giving up and feeling defeated, less complex tasks in which students experience successes could encourage them to continue until they solve the problem.

The two other technical skills; breaking down a programming task and problem-solving skills were also interdependent. Students cannot break down large and complex programming tasks if they lack problem-solving skills. Studies into problem solving in programming, which originally started in the mid 1980s, are now being revisited by researchers as this is a prominent concern.

The generic skills that students lacked have also been highlighted. In the university context, the development of generic skills has not been seen as part of the Information and Communication Technology charter. In the primary and secondary school system there have been a range of attempts to teach generic personal skills; they are now part of curriculum reform. However, recently industry and governing bodies (IEEE/ACM, ACS, DEST and employer reports) have recognised the importance of these non-programming skills in programming.

A number of issues related to the social organisation of the task were raised in the individual domain that relate to group work skills, communication, and being a team player. Finally, students' attitudes to dealing with programming errors would seem to be influenced by the challenges they face which arise from the task.

Throughout this paper various issues have been raised, but only some possible solutions have been provided. These issues need further investigation. For example, if tinkering and breaking down tasks is important, then some tasks need to be set at the undergraduate level that have less emphasis on implementation and more on practising these skills. Teaching considerations should include how students are expected to learn these skills. Are these skills to be acquired from tutors who are typically untrained in teaching or should students be expected to learn these skills for themselves? If students could improve their capabilities then learning the key features of the language would be a much faster and more self-directed process. Some of these issues may be beyond the normal curriculum or outside the awareness of educators, yet they raise implications for teaching and for both course and task design. The next phase of the research investigates external influences.

6 References

Baird, J. R., & Mitchell, I. J. (1991). Some theoretical perspectives on learning, teaching, and change. *Journal of Science and Mathematics Education in Southeast Asia, 14*(1), 7-21.

Caspersen, M. (2007) Educating Novices in The Skills of Programming. PhD Dissertation. University of Aarhus, Denmark.

Baird, J. R., & White, R. T. (1982b). Promoting self-control of learning. *Instructional Science, 11*, 227-247.

Baird, R. J., & Northfield, R. J. (1995). *Learning from the PEEL experience*. Melbourne, Australia: The Monash University Printing Services.

Bassey, M. (1999). *Case Study Research in Educational Settings*. Buckingham, United Kingdom: Open University Press.

Beaubouef, T., Lucas, R., & Howatt, J. (2001). The UNLOCK system: Enhancing problem solving skills in CS-1 students. *ACM Special Interest Group Computer Science Education (SIGCSE) Bulletin, 33*(2), 43-46.

Bergin, S., & Reilly, R. (2005). *Programming: factors that influence success.* Paper presented at the Thirty-sixth SIGCSE technical symposium on computer science education, St. Louis, Missouri, USA.

Bunderson, E., & Christensen, M. (1995). An analysis of retention problems for female students in university computer science programs. *Journal of Research on Computing in Education, 28*(1), 1-18.

Byrne, P., & Lyons, G. (2001, 25-27 June). *The effect of student attributes on success in programming.* Paper presented at the Sixth Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2001), University of Kent, Canterbury, United Kingdom.

Daigle, R. J., Doran, M. V., & Pardue, J. H. (1996). Integrating collaborative problem solving throughout the curriculum. *Proceedings of the Twenty-seventh SIGCSE Technical Symposium on Computer Science Education, 28*(1), 237-241

Denny, Luxton-Reilly and Simon, (2008). Evaluating a New Exam Question: Parsons Problems. Presented at the Fourth International Computing Education Workshop, Sydney, Australia, September 2008.

de Laet, M., Slattery, M. C., Kuffer, K., & Sweedy, K. E. (2005). *Computer games and CS education.* Paper presented at the Thirty-sixth SIGSCE Technical Symposium on Computer Science Education, St. Louis, Missouri, USA.

de Raadt, M., Watson, R., & Toleman, M. (2006). *Chicken sexing and novice programmers: Explicit instruction of problem solving strategies.* Paper presented at the Eighth Australasian Computing Conference (ACE 2006), Hobart, Tasmania, Australia.

Deci, E. L. (1975). *Intrinsic Motivation*. New York, USA: Plenum.

Entwistle, N. (1998). Motivation and approaches to learning: Motivation and conceptions of teaching. In S. Brown, S. Armstrong & G. Thompson (Eds.), *Motivating Students*. London, United Kingdom: Kogan Page.

Fredricks, A. J., Blumenfeld, C. P., & Paris, H. A. (2004). School engagement: Potential of the concept, state of the evidence. *Review of Educational Research, 74*(1), 59-109.

Jadud, M. (2006). Methods and Tools for Exploring Novice Compilation Behaviour. *ICER'06*, September 9–10, 2006, Canterbury, United Kingdom.

Glaser, B., & Strauss, A. (1967). *The Discovery of Grounded Theory: Strategies for Qualitative Research*. New York, USA: Aldine.

Goold, A., & Rimmer, R. (2000). Factors affecting performance in first-year computing. *ACM Special Interest Group Computer Science Education (SIGCSE) Bulletin, 32*(2), 39-43.

Haden, P. (2006). *The incredible rainbow spitting chicken: Teaching traditional programming skills through games programming.* Paper presented at the Eighth Australasian Computing Education Conference (ACE 2006), Hobart, Tasmania, Australia.

Hagan, D. (2004). *Employer satisfaction with ICT graduates.* Paper presented at the Sixth Australasian Computing Education Conference (ACE 2004), Dunedin, New Zealand.

Hagan, D., & Markham, S. (2000). Does it help to have some programming experience before beginning a computing degree program? *ACM Special Interest Group Computer Science Education (SIGCSE) Bulletin, 32*(3), 25-28.

Helme, S., & Clarke, D. (2001). Identifying cognitive engagement in mathematics classroom. *Mathematics Education Research Journal, 13*, 133-153.

Kassin, S. (2003). *Psychology*. USA: Prentice Hall.

Katz, S., Aronis, J., Allbritton, D., Wilson, C., Soffa, M. L., (2003). *A Study to Identify Predictors of Achievement in an Introductory Computer Science Course* . Proceedings of the 2003 SIGMIS conference on Computer personnel research., April 10-12, 2003, Philadelphia, Pennsylvania.

Lister R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, E., Sanders, K., Seppälä, O., Simon, B., Thomas, L., (2004) A Multi-National Study of Reading and Tracing Skills in Novice Programmers, *SIGCSE Bulletin*, Volume 36, Issue 4 (December), pp. 119-150.

Maslow, A. (1962). *Towards a psychology of being*. Princeton, New Jersey, USA: Van Nostrand.

McCracken, M., V. Almstrum, D. Diaz, M. Guzdial, D. Hagen, Y. Kolikant, C. Laxer, L. Thomas, I. Utting, T. Wilusz, (2001) *A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-year CS Students.* SIGCSE Bulletin, 33(4). pp 125-140.

Miles, M. B., & Huberman, M. A. (1994). *Qualitative Data Analysis: A Sourcebook of New Methods* (2nd Edition ed.). Thousand Oaks, California, USA: Sage Publications Inc.

Patton, M. Q. (1990). *Qualitative Evaluation and Research Methods (Second Edition).* Newbury Park, California, USA: Sage Publications.

Ramalingam, V., LaBelle, D., & Wiedenbeck, S. (2004). *Self-efficacy and mental models in learning to program.* Paper presented at the Ninth Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE 2004), Leeds, United Kingdom.

Ryan, R. M., & Deci, E. L. (2000). Intrinsic and extrinsic motivations: classic definitions and new directions. *Contemporary Educational Psychology, 25*(1), 54-67.

Schwarz, N. (1999). Self-reports: How the questions shape the answers. *American Psychologist, 54*(2), 93-105.

Shulman, J. (1997). Teaching Cases: New approaches to teacher education and staff development. http://www.ed.gov/pubs/triedandtrue/teach.html Accessed 1 May 2006.

Simon, Cutts, Q., Fincher, S., Haden, P., Robbins, A., Sutton, K., et al. (2006). *The ability to articulate strategy as a predictor of programming skill.* Paper presented at the Eighth Australasian Computing Education Conference (ACE 2006), Hobart, Tasmania, Australia.

Simon, Fincher, S., Robbins, A., Baker, B., Box, I., Cutts, Q., et al. (2006). *Predictors of success in a first programming course.* Paper presented at the Eighth Australasian Computing Education Conference (ACE 2006), Hobart, Tasmania, Australia.

Taylor, H., & Mounfield, L. (1994). Exploration of the relationship between prior computing experience and gender on success in college computer science. *Journal of Educational Computing Research, 11*(4), 291-306.

Tolhurst, D., Baker, B., Hamer, J., Box, I., Lister, R., Cutts, Q., et al. (2006). *Do map drawing styles of novice programmers predict success in programming? A multi-national, multi-institutional study.* Paper presented at the Eighth Australasian Computing Education Conference (ACE 2006), Hobart, Tasmania, Australia.

Wiedenbeck, S. (2005). *Factors Affecting the Success of Non-Majors in Learning to Program.* ICER'05, October 1–2, 2005, Seattle, Washington, USA.

Wilson, B. (2002) A Study of Factors Promoting Success in Computer Science Including Gender Differences. *Computer Science Education,* Vol. 12, No. 1-2, pp. 141-164.

Winslow, L. E. (1996, Sept 1996). *Programming pedagogy -- A psychological overview.* Paper presented at the Twenty-seventh SIGCSE Technical Symposium on Computer Science Education, Atlanta, Georgia, USA.