Education Research and Perspectives
An international journal

Volume 41, 2014, Pages 196-216

How Students Experience Learning to Program

Rachel Cardell-Oliver[†]

The University of Western Australia

This study seeks to understand how students experience learning computer programming, and the implications of those experiences for the quality of their learning. In order to identify the essence of the experiences, different types of artefacts produced by students during teaching are analysed including program code, programming assignment demonstration interviews, course feedback surveys, emails, and comments written on examination papers. The main contribution of this paper is the description, using narratives, of four distinct student experiences of a first programming course: thriving, surviving, drowning and lost. Each narrative shows a unique combination of effective and ineffective learning behaviours.

Introduction

Students' experiences of learning computer programming are varied and complex. Whilst some students thrive, others struggle to make progress. This problem has been observed and analysed in many research studies (see Robins [2010] for a review). A symptom of the problem is higher than usual rates of both failure grades and high grades in programming courses. However, there is still little consensus on the reasons that student outcomes differ so widely. Few studies have focussed on the different ways in which students experience learning to program nor on how specific differences affect learning. Therefore, this paper address two questions about learning to program: What are the different ways in which students experience their programming courses? and What are the implications of these experiences for student learning?

[†] Address for correspondence: Associate Professor Rachel Cardell-Oliver, School of Computer Science and Software Engineering, The University of Western Australia (M002), 35 Stirling Highway, Crawley, Western Australia, 6009. Email: Rachel.Cardell-oliver@uwa.edu.au.

Context

The context for this study is first year students learning computer programming at The University of Western Australia, one of Australia's "Group of Eight" leading research-intensive universities. Between 150 and 280 students each year take a 13-week course on introductory programming in the Java language. This first programming course uses a traditional mix of lectures and laboratories. Formal assessment is based on practical programming assignments and written exams with multiple-choice and short answer questions. In the literature, this type of course is called a CS1 (Computer Science 1) course.

I have taught and developed the introductory programming course over the last five years. One novel feature of the course is that software engineering measurement is embedded in the programming laboratory classes. The aim is to provide students with detailed feedback on the quality of all the computer programs they write, while they are developing those programs. The formative feedback provided is detailed and positive. It offers multi-faceted views of the quality of the computer programs the students are writing. Feedback is generated automatically by the same systems that are used by professional software developers. The frequency and amount of feedback is directly controlled by each individual. The properties measured by the in-lab feedback system are also used as part of the summative assessment criteria for programming assignments, giving an unambiguous specification of what is required.

In previous studies I have investigated the effects of this type of feedback on student learning. In Cardell-Oliver et al. [2010] I demonstrated that the quality of the programs written by students improves when they use the in-lab feedback. However, although performance improves on average, I also observed that individuals have different responses to the feedback. While some students benefitted, others did not improve, and some were even hindered by adopting unproductive learning strategies. In Cardell-Oliver [2011] I showed how quantitative measures of students programs could alert instructors to learning difficulties, including some of the unproductive behaviours I had observed. In Cardell-Oliver [2013] I

contrasted quantitative measures of student programs with qualitative evidence from project interviews with the students. This study identified further contradictory evidence. Some students produced reasonable programming assignments but in face-to-face interviews they demonstrated poor understanding of core concepts. Other students were motivated and worked hard but performed poorly in assessments. The quantitative methods used were not able to capture the complexity and nuances of such behaviours and outcomes. I concluded that what was needed was a view of learning that considers students' social and emotional experiences as well as the skills, knowledge and ability they bring to their study of computer programming. These observations provided the motivation for the current study, and in particular my first research question: What are the different ways in which students experience their programming courses?

Conceptual Framework

A conceptual framework for how students learn (Ambrose et al., 2010) underlies the research design introduced in this paper. Learning is viewed as a process that involves not only students' skills, knowledge and abilities, but also their social and emotional experiences. These factors work together to influence how effectively students do or do not make use of the learning support that is provided for them, and ultimately the quality of their learning. Factors that affect learning are categorised in terms of students' experience prior to learning a new subject and their experiences whilst they are learning it. Students' prior knowledge and how they organise knowledge can either support or hinder their learning. Learning is also affected by students' current level of development and the social, emotional, and intellectual climate of a course. During learning, motivation is a critical factor that shapes, directs and sustains what students learn. The quality of student learning is enhanced by goal-based practice and targeted feedback. In order to develop mastery of a subject, component skills are acquired and integrated and students also need to know when to apply those skills. Finally, students' ability to reflect on and adapt their learning strategies impacts their success. Students need to learn to monitor and adjust their approaches to learning.

This conceptual framework is chosen for several reasons that resonated with my own experience of teaching computer programming. The underlying assumption for this framework is that learning is a developmental process that takes place in students, and that learning involves changes, over time, in knowledge, beliefs, behaviours and attitudes. Learning is shaped by how students interpret and respond to their experiences. My aim is to answer questions about the complexity of students' responses to feedback by allowing for the many different factors that affect how students engage in the learning process. That is, as well as students' skills, knowledge and abilities, I consider their social and emotional experiences. The principles for learning described in Ambrose et al. [2010] are used to frame answers to the second research question of this study: What are the implications for learning of students' experiences of learning to program?

Related Work

This study draws on related work in two areas: studies of how students experience the task of learning to program; and studies of why learning computer programming, specifically, is so difficult. This section summarises previous research in these areas.

Experiencing learning to program

Students conceptualise the task of learning computer programming in a number of ways. According to Booth [1993], they may perceive it as computer-oriented, problem-oriented or product-oriented. Or, according to Bruce et al. [2004], some students perceive the task as skill-acquisition, whilst others as problem solving and still others as learning to participate in the computing profession. Each of these perceptions is associated with an internal horizon and an external horizon that represent the focus of the participants' attention, and the perceptual boundary of the category. In their study, Bruce et al. [2004] found that a student who experiences the act of learning to program as the acquisition of coding skills, for example, focuses on syntax and practising coding tasks. Their perceptual boundary is the realm of the programming language, without awareness of the broader context of programming or of the professional programming world.

Responses to learning programming are dominated by emotional experiences and reactions. Kinnunen and Simon [2010] analysed student reactions at each stage of their programming assignments, from getting started, to encountering and dealing with difficulties. Common experiences they identified included encountering unexpected problems ("hit by lightening"), not knowing what to do ("OK, what now?"), and not using feedback to guide actions ("hamster wheel"). Kinnunen and Simon [2012] analysed students' perceptions of their experiences, finding that some students had negative perceptions of self-efficacy even after positive programming experiences, whilst others had positive self-efficacy even after negative programming experiences.

Determinants of success or failure

Students conceptualise the task of learning computer programming in Many hypotheses have been proposed that aim to explain why some students fail and others succeed in learning to program. One approach to answering this question has been to use quantitative methods such as regression analysis to investigate the relationship between possible determining factors and student grades. However, such studies have failed to identify specific factors that predict success. Robins [2010, p37] reviews existing studies of this sort, arguing that:

"[T]he typical introductory programming (CS1) course has higher than usual rates of both failing and high grades ...[the] conventional explanation has been that learners naturally fall into populations of programmers and non-programmers. A review of decades of research, however, finds little or no evidence to support this account."

Another line of research starts with the assumption that various types of cognitive overload account for the problems that students experience and that this can explain high failure rates. The researchers argue that particular aspects of pedagogy (the way computer programming is taught) lead to cognitive overload, and therefore that new, research-led approaches to instructional design will address the problem. Robins [2010] proposes a model called "learning edge momentum" (LEM) to explain the phenomena of

students who struggle early in programming units and never catch up. He argues that tight integration of concepts makes learning to program harder than other subjects because failure in acquiring one concept makes learning other closely linked concepts harder. Conversely, success in understanding a concept makes learning linked concepts easier. However, Petersen et al. [2011] argue that the LEM theory does not explain student outcomes in programming courses. They review typical programming exam questions and argue that most questions examine too many different concepts, and that this lack of separation of concerns does not allow students to demonstrate what they do know. In a similar vein, Caspersen and Bennedsen [2007] argue that students have poor learning outcomes because of poor educational design of programming courses, which lack sufficient scaffolding and apprenticeship examples. They argue that poor design leads to cognitive overload and so to student failure. Kolikant and Mussai [2008] argue that students' conceptions of program correctness are flawed because of the common assessment practice of marking programming assignments as the sum of points for separate aspects of a program. This nurtures students' misconceptions that partial correctness of a program is all that is required.

In summary, researchers have shown that students differ in their conceptions of programming, their emotional responses when undertaking programming assignments and their perceptions of self-efficacy based on those experiences. Researchers have argued that cognitive overload, caused by poor design of courses and poor assessment practices, explains why some students fail. However, the question remains of why other students succeed. Little or no evidence has been found to support the hypothesis that learners naturally fall into populations of programmers and non-programmers.

Method

This study addresses the question of how students experience learning to program and how those experiences affect the quality of their learning. A qualitative research design is used to answer these questions because of limitations of existing quantitative or theory-

based research designs. Quantitative methods can measure what a student produces, or measure certain characteristics of the student. But they give no insight into why certain combinations are observed. In previous studies I found that some students were able to produce correct programs, and correctly answer multiple-choice exam questions, but they demonstrated poor understanding of the material in more open-ended assessments. Another type of research design is to use theories of learning to account for observations about students' behaviours and outcomes. But these do not capture the personal and complex nature of learning. For example, poor instructional design certainly impedes learning for some students, but others will thrive none-the-less. My aim is to gain insight into why this is the case, and so a qualitative research design is chosen since it captures the complexity and individuality of the student learning experience.

Educational research performed by teachers in their classrooms (teacher-researchers) is known as "insider" research, whilst that based on systematic observation of teaching by an external researcher is known as "outsider" research. Clearly the methodology in this paper is insider research. The strengths and weaknesses of each approach have been argued by McNamara [1980] and Hammersley [1991, 2006]. Teacher researchers often have long-term experience of the setting being studied (as in my case) and so understand the history and contextual setting. However, this knowledge can be superficial or distorted. The teacher-researcher already has relationships in their setting and can use that to collect further data. However, those relationships can exclude as well as include others, and so may not include what is necessary for research purposes. This study guards against the pitfalls by assessing the themes and narratives against a general theory of student learning, and by considering a wide range of student artefacts as evidence for the themes.

Data

Many different types of data have been used for narrative research in education (Connelly and Clandinin 1990). Whilst interviews with students and teachers are common (Haggis 2004), researchers have also used written field notes of shared experience, journal notes

made by participants, and various types of documents. Studying students' experience of learning to programming offers new opportunities from data sources that may not be available in other educational research studies. In particular, many of the learning interactions that take place in this setting are in electronic, written form: students communicate with the lecturer and each other by email or a web discussion forum, they submit all their written work to an online repository, and electronic records of participation, demonstrations and grades are maintained. Two sources of hand-written artefacts were also used in our study: student course feedback comments and their comments written on examination papers. Such data might be considered low quality in some domains of educational research. However, I argue that this data provided unique and significant insights into students' experiences. The comments written by students on their final exam papers turned out to be a rich source of "think aloud" evidence. For example, on their exam question papers students often showed their reasoning behind their answers, or showed areas of uncertainty by changing their choices several times.

The data were selected from the 2013 cohort of 280 students at The University of Western Australia studying introductory computer programming in the Java language. All data were collected as part of the normal routine of teaching this course and were used with appropriate ethical clearance.

Analysis of Themes

Data were first analysed by reading through the different data sources (learning activities, assessments, interactions and observations) and identifying recurring themes. I considered each type of data in turn, recording observed phenomena and the evidence for each. Each type of data source provided different insights into students' experiences. Student comments written on their exam scripts were particularly informative about the experience of weaker students. These comments provided a form of "think aloud" evidence about the thought process of the student. Comments from course tutors about student responses during their project demonstration interviews also provided such think aloud insights. Students'

comments on anonymous course feedback forms highlighted common experiences about the aspects of the course students did and did not enjoy and on their motivation. Email messages sent to me as the course lecturer were received from only a small proportion of the students. These interactions often concerned negative experiences of not being able to complete an assignment or other difficulties with the course. Programs and tests submitted for assessment were used to cross correlate observations about students' experiences with the course outcomes for those students.

From the thematic analysis of the data I identified a number of emotional themes: confidence, panic, satisfaction, frustration, as well as degrees of success with the course as shown by final grades. Then I selected eight individuals who were representative of these themes and studied their data in detail, mapping their emotional responses with their academic work in the unit. This identified themes such as strong skills throughout, developing skills during the course, never got started, and mistaken confidence. I could also link these with specific experiences (e.g. from an email) for individual students. The coded themes from these two stages of data analysis were then used to construct the narratives.

Generation of Narratives

A narrative aims to describe how a given person, in a given context, makes sense of a given phenomenon. Narratives are used to illustrate the experiences of students learning to program. Three dimensions of narrative inquiry are temporality, sociality and place (Clandinin and Huber, 2010). Temporality in our study concerns the past, present and future of student experiences during their semester of studying programming. Sociality concerns personal (feelings, hopes, reactions) and social (cultural, institutional) conditions. Personal conditions are expressed through students' own words in their written artefacts (e.g. emails, assessments, web forum posts). Social conditions are expressed in the narratives through the institutional and the cultural expectations for university students and their study at UWA. For example, the frank and informal exchanges between students and lecturer that were commonplace for the cohort in this study are expressed in the narratives. These interactions differ

from those I have experienced when teaching in the UK and Germany.

In the first stages of data analysis I identified clusters of ideas about the experiences of students in the course. From these clusters, I constructed short narrative accounts from the point of view of a student. Writing these narratives involved the three stages of broadening, borrowing and restorying (Connelly and Clandinin 1990). Broadening occurs when an event experienced by one student is used to generalize about their emotions. Burrowing considers the emotional quality of events and the possible origins for those feelings. In restorying the researcher reviews the event being described in the context of the larger significance of this event in the life of the individual.

While all the examples in the narratives are taken from the data, they have been combined and changed slightly so as not to identify particular individuals. Quotations by people, such as the lecturer and friend, who interacted with the student are used to reinforce the sense of being there. Anonymous student comments from course questionnaires and the online help forum were also used in the narratives, although these could not be directly matched with individuals. All other data sources were identified with an individual and so could be cross-correlated.

Narratives on Learning to Program

This section presents the four themes identified in the data analysis in the form of short narratives. Each narrative presents a student's point of view of their experience of learning to program.

Thriving. The first account is of a student who thrives on the feedback provided.

It's incredibly satisfying when your program works

Taking a Computer Science unit this semester on learning how to program was my favourite unit. The labs were interesting and well paced. Large seemingly impossible tasks were broken down into manageable chunks. The learning curve was good.

Programming can be difficult at first. It is annoying when your program doesn't work, and you spend ages trying to figure our why. First you have to work out how to translate the task to be done into computer code, and

then you have to fix any bugs (mistakes). There is software we can run that shows you the bugs in your program. Then you have to find out what is causing each one and fix it. The main thing I have learnt is how to find and fix bugs. Bugs can seem to come from nowhere, for no reason. But there is always a logical reason behind a bug. It is incredibly satisfying when your program does work. That's the feeling that motivates me to work hard.
The final project was the most enjoyable and interesting part of the unit since it was the most challenging. I was able to show that I could solve problems that I could never have tackled at the start.

Surviving. The second narrative concerns a performance-directed learner whose focus is on what has to be done to achieve good marks.

Just tell me what I need to do

At school I was in the top group and now I want good marks in my Computer Science degree at university. It is important for me to know exactly what I need to do. I check the requirements for assessments carefully and ask for clarification if I am not sure. I focus on the assessed tasks. But I do not submit the non-assessed weekly labs because I am also busy with other work.
I like the feedback provided in my programming unit to identify bugs in your assignments. It makes it clear exactly what we are expected to do ... most of the time. I don't understand why I failed the first project (24/50) when everything ran okay minus a few small parts. The project was pretty good otherwise. I put a lot of effort into that project and was quite confident for at least 60%.
I was upset about my project mark. So I contacted the lecturer and asked her why I had not passed. She said, "Bugs do matter. Focus on how to find and fix them." I am also expected to be able to explain what I have written and to make changes to it. That is focussing on minor problems, which I don't really care about. Still, if those are the rules, I'll live with them. I did better in the second project.
I focus my exam study on past papers. I did well in the mid-semester test and exam, so that seems to be a good strategy. I am happy with my final mark of 68% for the unit.

Drowning. The third narrative describes a student who is working hard and appears to be managing but who, in fact, has failed to understand fundamental concepts.

Not Waving but Drowning

I am optimistic and hardworking and I have decided to study Computer Science at university. My programming unit is going well - not brilliant but OK. I attend all lectures and lab classes. I do the weekly practical work and submit it, whether complete or not. Sometimes I can't work out how to do a step. There is software we can run that shows you the bugs in your program. Sometimes my programming assignments fail these checks. Then I leave a note for myself so I can come back to it. But I never do get time for that.
This semester has been hard because there is a lot going on for me at the moment. My wallet and phone were stolen and there were insurance problems. Then my laptop kept crashing, so I had to go into town on the bus to get it fixed and I had to wait three hours to see someone. Half the day gone when I needed to work on my assignment. I spent four hours the next day getting one small part of the assignment to work. A friend told me, "Programming is like that. You keep trying and trying and sometimes it takes a long time."
Now it is exam time. I looked over my mid-semester test before the exam. That was useful. I am disappointed that I failed this unit, because I thought it would work out in the end. I thought I understood what was required of me.

Lost. The final narrative concerns a low-achieving learner who became lost early on and never catches up.

Completely Lost

I am taking a Computer Programming unit this semester. It is not a core unit for my degree in Engineering but I thought it would be useful to know how to program.
I haven't been able to do any of the lab exercises. My first project was three days late. I had a panic attack and I was not able to finish it. There is software we can run that tells you that you've made a mistake. But I don't know what to do about mistakes when they are pointed out. I feel helpless.
Talking my programming problems through with the lab tutor showed me that I could do some things after all. Then I felt better. But although I can follow when my tutor explains, I don't know where to start when I try it for myself.
Other subjects have higher priority for my degree than this one. I tried to study to bring up my grade in programming but once you get behind it is impossible to catch up. I wish I had realised much earlier that this unit wouldn't work out for me.

Implications for Learning

The four narratives in this paper describe different ways in which students experience learning to program. The second research question of this paper is what are the implications of these experiences for student learning? That question is answered here in terms of the conceptual framework for learning that was outlined at the start of the paper.

Thriving

All students encounter problems when learning to program. The thriving student has a positive perception of their ability to deal with problems, and effective strategies for overcoming problems when they are encountered. Therefore their learning thrives. Kinnunen and Simon [2012] identified a similar phenomenon in students' perception that "even though I am struggling now, I know I can get there". The thriving student is good at monitoring their progress and adjusting their approach to learning where necessary ("the main thing I have learnt is . . . "). They have effective strategies for organising their knowledge. Motivated by getting their programs to work ("that's the feeling that motivates me to work hard" [my emphasis]), they acquire and practice component skills, and practice integrating those skills. A risk for the thriving student is that they resent over-specified tasks, and may become bored and demotivated. Therefore, it is important for courses to include open-ended activities to motivate the thriving student.

Surviving

The motivation of the surviving student is extrinsic. This student's learning is driven by the course assessment criteria. The surviving student demonstrates two behaviours that support learning. They are motivated by success ("I want good marks") and so they practice the component skills of computer programming ("I put a lot of effort into that project"). They may have poor skills in organising knowledge ("I don't understand why I failed the first project (24/50)" [my emphasis]). A risk for these students is that typical methods for assessing programming assignments may reward shallow

approaches. Kolikant and Mussai [2008, p. 135] studied this phenomena, concluding:

"We found that students conceptualized program correctness as the sum of the correctness of its constituent operations and, therefore, they rarely considered programs as incorrect. Instead, as long as they had any operations written correctly students considered the program 'partially correct'. We suggest that this conception is a faulty extension of the concept of a program's grade, which is usually calculated as the sum of points awarded for separate aspects of a program. Thus school (unintentionally) nurtures students' misconception of correctness."

The surviving student is not aware of gaps in their knowledge structures ("I was quite confident ... everything ran okay minus a few small parts"). Their analysis of their learning skills focusses on optimising performance in assessments ("I study past papers"). Further, since their prior learning has rewarded this performance-driven approach, they are reluctant to change it ("if those are the rules, I will live with them"). As a result, they miss the opportunity for deeper learning by better organising their knowledge structures and integrating the skills they have learnt.

Drowning

The title of the drowning narrative is taken from a poem by Stevie Smith called Not Waving but Drowning, in which a man swimming in the sea drowns because, although he signalled for help, onlookers thought he was just waving to them. The drowning student demonstrates two behaviours that support learning. They are well motivated and they invest time and effort to practice the component skills for computer programming ("I attend all the lectures and lab classes. I do the weekly practical work"). However, their learning is hindered by misinterpreting the feedback they receive and their inability to analyse their learning and adjust their strategies ("My programming unit is going well"). This behaviour may be reinforced by strategies that have worked well in prior learning but that break down when learning computer programming ("I leave a note for myself so I can come back to it"). Their learning strategies tend to be shallow ("I looked over my mid-semester test" [my emphasis]). They

have poor organisation of the knowledge they have, tending to piece things together ("you keep trying and trying") rather than structuring their knowledge. The drowning student experiences many distractions (fixing the laptop, insurance problems). These may be unconscious avoidance strategies. They are also hampered by poor advice they receive from other students. For example the friend who said programming just involves trying and trying until it works undermines the idea that finding bugs is a logical process. That advice reinforced the drowning student's poor learning strategies.

Lost

The lost student demonstrates two types of behaviour that could support learning. They start with the motivation to succeed ("I thought it would be useful to know how to program"), and they come to the subject with learning strategies that have served them well in the past ("I tried to study to bring up my grade in programming"). However, they also have many strategies that hinder their learning. They are able to monitor their learning to some extent ("Talking my problems through with my the lab tutor"). They recognise, from the feedback they receive, that they are not achieving the learning objectives set for them ("I don't know what to do about mistakes when they are pointed out"). However, their emotional responses when faced with difficulties prevent them from adjusting their learning strategies ("I had a panic attack" and "I feel helpless"). Their response to difficulties is one of panic and avoidance (e.g. focussing on other subjects), which leads to poor results and then lost motivation. Their approach is again emotional in face-to-face learning situations ("I felt better"), but without the ability to learn from these. Furthermore, their selective approach to completing learning activities means that they receive insufficient practice in component skills and as a result they consistently underestimate the time and effort required to complete larger tasks ("Other subjects have higher priority"). In summary, lost learners have the potential to succeed, but are hindered by their inability to identify how and when to respond to the problems they encounter.

Commonalities and Differences

Each narrative is associated with a distinctive signature of effective and ineffective learning behaviours. A common theme in all four narratives is that getting programs to "work" is the dominant motivation for students. However, given the same feedback, students differed in their responses when their programs did not work. While the thriving student is able to analyse and overcome the problems ("seemingly impossible tasks were broken down into manageable chunks") the lost student suffers from loss of motivation and feelings of helplessness ("I don't know where to start when I try it for myself"). Contrasting reactions to challenges (e.g. "Jane sees 50 compiler errors as a challenge. John sees them as defeat.") can be understood in terms of students' self-theories: whether they have a growth mindset or a fixed mindset (Simon et al. 2008).

Another common theme in the narratives is that learning experiences and strategies that have worked for students in the past can hamper their progress when they learn programming ("I thought the project was pretty good", "I do not submit the non-assessed weekly labs" and "Other subjects have higher priority"). For the drowning and lost student, the inability to identify correctly where their learning problems lay was critical ("I am disappointed that I failed this unit, because I thought it would work out in the end" and "I wish I had realised much earlier that this unit wouldn't work out for me.").

A third common theme is that emotional responses were (surprisingly) significant in students' experience of learning to program. Negative feelings of being overwhelmed by external and internal pressures dominated for the drowning and lost student ("I feel helpless" and "Half the day gone when I needed to work on my assignment"). Positive feelings dominated for the thriving and surviving student ("it is incredibly satisfying when your program does work" and "I am happy with my final mark"). Kinnunen and Simon [2012] also found that emotional experiences dominated students' experience of learning to program.

Narratives can be used to identify common themes in students' experience. But it should be remembered that the narratives are drawn from distinctive, individual experiences. Since every learner is

uniquely situated, the differences between students' accounts are also important in trying to understand learning (Haggis 2004).

Application

Narratives enable experiences to be seen in a holistic and integrated way, in the speakers' voice. The sense of being there makes the observations immediately useable by students, teachers and researchers. Research narratives should produce some useable practices and also "evoke emotions in the reader, to restore from the sediments of memory similar personal experiences and mental images or to alter the reader's prevalent mind-set" (Heikkinen, Huttunen and Syrjala 2007, p 8). Evocativeness has also been described as the ability to "have another participant read the account and to respond to such questions as 'What do you make of it for your teaching (or other) situation?'" (Connelly and Clandinin 1990, p 8). I envisage that the four narratives of this paper can be used in practice in several ways. First, the narratives can be used in training sessions with tutors and lab demonstrators to alert them to the types of student experiences that lie behind the questions they will be asked in class. Second, I plan to use these narratives to inform changes to the current course organisation to reduce the problems that overwhelm some students. I believe that these narratives will also resonate with others' experiences and can be used to inform their teaching and learning. The experience of students in the United States, as reported by Kinnunen and Simon [2012], concurs with the experiences of the Australian students in our narratives. This suggests that the identified phenomena are general ones, with cross-cultural relevance. Third, these narratives provide a new way for researchers to understand the complex question of how students learn computer programming. More generally, these narratives help others to recognise the mix of behaviours shown by succeeding or failing students.

Gender aspects of the thriving, surviving, drowning or lost experiences are beyond the scope of this paper, but would be an interesting area for future work. For engineering students, the belief that engineering aptitude as a fixed ability, which in turn is associated with a tendency to drop classes when faced with difficulty

has been found to have a gender bias, as has emphasis on instrinsic rather than extrinsic factors (Heyman et al 2001).

Conclusions

This paper addresses two questions about learning to program: What are the different ways in which students experience their programming courses? and What are the implications of these experiences for student learning? To answer the first question, I constructed narratives, based on an analysis of learning artefacts from the course. The narratives describe four types of student experience when learning to program: thriving, surviving, drowning and lost. Although three of these categories are well known from the general literature on learning, the not waving but drowning experience seems to have particular resonance for computer programming students. Analysis of the narratives in terms of a conceptual framework of how students learn provides insights into the second question of why computer programming students thrive or not. These insights were not apparent in previous quantitative and theoretical studies.

The main findings of this paper concern students' motivation and learning strategies. Getting programs to work is the dominant motivation for most students. Emotional responses are a significant part of the experience of learning to program. Students differ in the ways their prior learning experiences and strategies either help or hinder them when learning to program. Traditional ways of setting and assessing programming assignments and exams may mislead students and lecturers about students' progress. The findings reported in this paper offer insights towards better understanding of these complex problems.

Ethics Clearance

This research has been assessed and granted exemption from full Human Research Ethics review for this project (14 June 2013) RA/4/1/6236 titled Formative Feedback and Automated Assessment for Learning Computer Programming.

Acknowledgements

This work is supported by Teaching Fellowship grant F12047-2010 from the University of Western Australia. The author would like to thank H. Wildy, E. Chapman, C. Baillie, the anonymous referees, and members of the UWA Engineering Education Writing Group and the Faculty Academy for the Scholarship of Education for their valuable discussions and feedback on this paper.

References

Ambrose, S.A., Bridges, M.W., Di Pietro, M., Lovett, M.C. and Norman, M.K. (2010) *How Learning Works: Seven Research-Based Principles for Smart Teaching*. John Wiley & Sons, 2010. ISBN 978-0-470-61760-1 (epub).

Booth, S. (1993) A study of learning to program from an experiential perspective. *Computers in Human Behavior*, 9(23):185–202, 1993. ISSN 0747-5632. doi: http://dx.doi.org/10.1016/0747-5632(93)90006-E.

Bruce, C., Buckingham, L., Hynd, J., McMahon C., Roggenkamp, M., and Stoodley, I. (2004) Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university. *JITE*, 3:143–160, 2004.

Cardell-Oliver, R. (2013) Evaluating the application and understanding of elementary programming patterns. In *Proceedings of the 22nd Australasian Software Engineering Conference (ASWEC), 2013*, pages 60–67, 2013. doi: 10.1109/ASWEC.2013.17.

Cardell-Oliver, R. (2011). How can software metrics help novice programmers? In John Hamer and Michael de Raadt, editors, *Australasian Computing Education Conference (ACE 2011)*, volume 114 of CRPIT, pages 55–62, Perth, Australia, 2011. URL http://crpit.com/confpapers/CRPITV114Cardell-Oliver.pdf.

Cardell-Oliver, R., Zhang, L., Barady, R., Lim, Y.H., Naveed, A., and Woodings, T. (2010) Automated feedback for quality assurance in software engineer- ing education. In *Proceedings of the 2010 21st Australasian Software*

Engineering Conference, ASWEC '10, pages 157–164, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4006-1. doi: http://dx.doi.org/10.1109/ ASWEC.2010.24.

Caspersen, M.E. and Bennedsen, J. (2007) Instructional design of a programming course: a learning theoretic approach. In *Proceedings of the Third international Workshop on Computing Education Research, ICER '07*, pages 111–122, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-841-1. doi: 10.1145/ 1288580.1288595.

Clandinin, D. J., & Huber, J. (2010). Narrative inquiry. In B. McGaw, E. Baker, & P. P. Peterson (Eds.), *International Encyclopedia of Education* (3rd ed.). New York, NY: Elsevier, p 436-441.

Connelly, F. Michael, and Clandinin, D. Jean (1990). Stories of Experience and Narrative Enquiry, *Educational Researcher*, 19: 2, doi: 10.3102/0013189X019005002

Haggis, T. (2004). Meaning, identity and 'motivation': expanding what matters in understanding learning in higher education?. *Studies in Higher Education*,*29* (3), 335-352.

Hammersley, M. (1993) On the teacher as researcher. *Educational Action Research*, *1*(3), 425-445.

Hammersley, M. (1981), The Outsider's Advantage: a reply to McNamara. *British Educational Research Journal*, 7: 167–171. doi: 10.1080/0141192810070205

Heikkinen, H., Huttunen, R., & Syrjala, L. (2007). Action research as narrative: Five principles for validation. *Educational Action Research*, 15(1), 5–19. doi: 10.1080/ 09650790601150709

Heyman, G. D. Martyna B. and Bhatia. S. (2002) Gender and achievement-related beliefs among engineering students. *Journal of Woman and Minorities in Science and Engineering*, 8:41-52

Kinnunen, P. and Simon, B. (2010) Experiencing programming assignments in CS1: the emotional toll. In *Proceedings of the Sixth international workshop on Computing Education Research, ICER '10*, pages 77–86, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0257-9. doi: http://doi.acm.org/10.1145/1839594. 1839609.

Kinnunen, P. and Simon, B. (2012) My program is OK - am I? Computing Freshmen's Experiences of Doing Programming Assignments. *Computer Science Education*, 22(1):1–28, 2012. doi: 10.1080/08993408.2012.655091.

Kolikant, Ben-David Y. and M. Mussai, M. (2008) 'So my program doesn't run!' definition, origins, and practical expressions of students' (mis)conceptions of correctness. *Computer Science Education*, 18:135–151.

McNamara, David R. (1980) The Outsider's Arrogance: the failure of participant observers to understand classroom events, *British Educational Research Journal*, 6:113-125. doi:10.1080/0141192800060201

Petersen, A., Craig, M., and Zingaro, D. (2011) Reviewing CS1 exam question content. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education, SIGCSE '11*, pages 631–636, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0500-6. doi: 10.1145/1953163.1953340.

Robins, A. (2010) Learning edge momentum: a new account of outcomes in CS1. *Computer Science Education*, 20(1):37–71, March 2010. doi: 10.1080/08993401003612167.

Simon, B., Hanks, B., Murphy, L., Fitzgerald, S., McCauley, R., Thomas. L., and Zander, C. (2008) Saying isn't necessarily believing: influencing self-theories in computing. *Proceedings of the Fourth international Workshop on Computing Education Research (ICER '08)*. ACM, New York, NY, USA, 173-184. doi: 10.1145/1404520.1404537