Journal of Information Technology Education Volume 9, 2010

The Fear Factor: How It Affects Students Learning to Program in a Tertiary Environment

Christine Rogerson and Elsje Scott
University of Cape Town, Cape Town, South Africa

Christine.Rogerson@uct.ac.za; Elsje.Scott@uct.ac.za

Executive Summary

This paper examines how students' experiences of learning to program are affected by feelings of fear, using a phenomenological approach to elicit rich descriptions of personal experiences from the narratives of final year undergraduate students.

In the course of reviewing current work concerning learning or teaching programming, certain focal areas of research emerged. This paper attempted to group these into three main topics. These are the predictors of student success in learning programming, the barriers to learning programming. and the teaching tools or learning methodologies which could assist with learning programming. The review was conducted with particular emphasis on phenomenological research in this field. Cockburn's concepts of skills acquisition and Dreyfus' levels of adult learning are discussed and are used as a theoretical lens to examine the growth of the students.

Learning to program or code forms part of the core courses taught by the Information Systems (IS) department at the University of Cape Town (UCT). Assuming one of the goals of education is to prepare students for the working world, a strong practical component is required. Although programming skills are less central to the IS curriculum than to that of computer science, a number of IS graduates become business analysts or project managers and are required to communicate with team members and, at the very least, to have a good understanding of programming issues. To address this need, the IS department at UCT designed a curriculum that introduces students to problem-solving, coding, and testing issues through an action learning cycle and that culminates in third year by requiring students to initialize an IS project from conception to readiness to implement. It is from this background that the students for this study were selected.

This paper approached the research from an interpretative stance, and, as the main aim was to describe students' life experiences and discover the essence and meaning of programming from their perspective, a phenomenological methodology appeared to be ideally suited.

From the analysis, six themes were uncovered. These themes are the apprehension or fear associated with programming, the resulting negative perceptions, the nature of programming that gives rise to these feelings, the internal factors influencing these feelings, the external factors influencing these feelings, and finally how these feelings have affected the students' growth or skills acquisition. A model is proposed that illustrates the relationship between the six themes, guides the analysis, and helps to makes sense of the implications.

This paper offers an insight into the difficulties experienced by students learning to program, and should be of interest

Editor: Thomas Connolly

to educationalists, particularly those in the programming environment, who seek to understand the problems faced by students in order to provide more effective support through their teaching approach and student interactions.

Keywords: Programming, Information Systems, Phenomenology, Fear, Education.

Introduction

"Learning and teaching are central activities in the increasingly complex and multi-facetted educational enterprises that are a prominent feature of post-industrial societies" (Hager, 2005, p. 633). Research into educational methods and theories is an ongoing process that covers many disciplines. In the Computer Science and Information Systems fields, there has been much concern regarding both the learning and teaching of computer programming and the high attrition rates associated with these courses (Robins, Rountree & Rountree, 2003). In spite of many years of research, it would appear that there is still a global problem within the educational environment relating to computer programming (Mead et al., 2006). The focus of many studies has been on improving the quality of teaching, aiming to make it more efficient and effective. However, it appears that less attention has been paid to the experience of learning to program from the students' perspective, and understanding what, in their perception, are the difficulties presented by the nature of programming. In the words of Jenkins (2002, p. 53), "If students struggle to learn something, it follows that this thing is for some reason *difficult* to learn." It would seem that in order to teach more effectively, it is essential for educators to have an understanding of what it is about programming that makes learning it so troublesome for many students worldwide.

This paper reports on a study that was conducted to understand and describe how students personally experience fear whilst learning programming and to investigate whether this fear affects other aspects related to their studies, such as self-confidence, time management, and problem solving skills, all of which are needed in order to succeed (Scott, 2008; Simon et al., 2006). Narratives of their experiences were analyzed and relationships between the units of meaning were elicited to formulate and propose a model as a concrete tool for educators interested in studying how the fears that beset programming students affect their learning.

Fear, programming, and phenomenology were core concepts used in this study. "Fear" may be considered an overly emotional word, but for this study it should be regarded as a descriptor for denoting a lack of interest in, or lack of appreciation of, programming as a discipline. In other words, some students may be experiencing a lack of confidence or apprehension regarding their ability to code or program, rather than actual fear. "Programming", as used in this study, refers to the full systems development cycle. It includes learning the basic theoretical concepts, as well as the practical act of coding, and having a final product ready for implementation. "Phenomenology" is an interpretive research approach used to study human experiences (Bruce et al., 2004). It does not aim to categorize individuals but rather to detail the number of different ways people can experience a single phenomenon, which, in this study, is learning to program.

It is anticipated that this research into the challenges and difficulties of learning to program, as encountered and narrated by the students themselves, will contribute to a deeper understanding of this area of concern for educationalists in general and tertiary programming educators in particular. It is also hoped that the research findings will provide useful insight for students who are also experiencing anxiety as a result of their struggle with learning programming.

The Act of Learning to Program

The teaching and learning of programming appears to be a subject of worldwide, historic concern. "Learning to program is notoriously considered difficult. In spite of more than forty years of experience, teaching programming is still considered a major challenge" (Caspersen & Bennedsen, 2007, p. 111). Many researchers regard programming courses as extremely demanding and the associated attrition rates high (Bergin & Reilly, 2005a, 2005b; Boyle, Carter, & Clark 2002); Gomes & Mendes, 2007a, 2007b; Jenkins, 2002; Simon et al., 2006). Mead et al. (2006, p. 183) comment "…over the past twenty-five years national and international studies have provided empirical indicators showing that learning to program is indeed challenging for most students."

Predictors of student success in learning programming

Numerous papers have been published concerning the demographics, attitudes, and abilities of successful programming students (Bergin & Reilly, 2005a; Byrne & Lyons, 2001; Gomes & Mendes, 2007a, 2007b; Kinnunen, McCartney, Murphy, & Thomas, 2007; McCracken et al., 2001; Simon et al., 2006; Wilson & Shrock, 2001). In their study, Wilson and Shrock (2001) investigated twelve predictive factors including previous programming experience, previous non-programming experience, work style preference, math background, gender, the student's own attribution for success or failure, and the student's level of comfort. Wilson and Shrock's (2001) finding was that, of all the variables addressed in their study, comfort level was the most reliable predicator of success. Low levels of comfort often result in feelings of anxiety, which is counterproductive to learning and may result in a dislike of programming as a discipline (Simon et al., 2006).

Another factor that may particularly affect this research is attribution. Attribution theory seeks to identify how individuals credit their success or failure (Wilson & Shrock, 2001). From past research, it appears that when individuals attribute their success to their own ability or task difficulty and their failure to the amount of effort expended or luck, they will be more likely to persist and eventually succeed at the task (Perry, Stupnisky, Daniels & Haynes, 2008; Weiner, 2008, Wilson & Shrock, 2001). The literature also appears to indicate a general acceptance of the positive relationship between mathematical ability and successful student programmers (Boyle et al., 2002; Byrne & Lyons, 2001; Gomes & Mendes, 2007a, 2007b; McGettrick et al., 2005; Simon et al., 2006). However, Wilson & Shrock (2001) suggest that what is more important is a mathematical background where the focus is on logic and problem solving. Often poor problem solving ability is the foremost cause of failure amongst programming students (Gomes & Mendes, 2007a, 2007b) and it becomes necessary to investigate alternative teaching approaches in order to help students develop competence in finding solutions to problems (Muller, Ginat & Haberman, 2007). Kinnunen at al. (2007) concur that students' chances of success are influenced by their way of thinking and, consequently, how they attempt to problem solve.

Inherent factors such as gender, age, culture, and language have been assessed as contributors to success with varying outcomes that make it difficult to draw any conclusions regarding using them as reliable predictors (Byrne & Lyons, 2001; Jenkins, 2002; Kinnunen et al., 2007). Although gender issues have raised much concern and it does seem that the majority of programming students are male (Palma, 2001), latter studies, in particular, have found that "female students achieved equally high scores as their male counterparts, an outcome which contradicts some of the evidence found in previous studies" (Byrne & Lyons, 2001, p. 52). Prior experience, both computer and academic, has been extensively investigated as a factor of success (Bergin & Reilly, 2005a; Byrne & Lyons, 2001; Hagan & Markham, 2000; Simon et al., 2006). Lack thereof appears to be a significant disadvantage, with the success factor increasing systematically in relation to the number of programming languages to which students had been exposed (Hagan & Markham, 2000). On the other hand, Boyle et al. (2002) found that students with mediocre entrance level results performed as well in the programming courses as those with excellent results, making any prior academic learning of debatable value.

Psychological attitudes as predicators of success cannot be dismissed. Motivation and attitude to learning can be considered central success factors (Gomes & Mendes, 2007b; Jenkins, 2002;

McCartney, Eckerdal, Moström, Sanders & Zander, 2007; Robins et al., 2003; Simon et al., 2006). Belief in oneself and one's abilities, known as self-efficacy, is also an important consideration. Together with Gomes & Mendes (2007b), Kinnunen et al. (2007) emphasize students' study methods, motivation, and expectations as being highly relevant to their success. Students who engage with their subject and are motivated to seek the broader picture are usually more successful than those who merely memorize formulas (Simon et al., 2006). Due to its nature, learning to program thus seems to require a particular approach, and courses that offer coping skills may be more valuable to students than those focusing on mere study skills (Jenkins, 2002).

Barriers to learning programming

There are several barriers to learning programming, which include the nature or complexity of programming itself. This often involves learning a new way to think and to study; the difficulty of becoming reflective or self-aware; the newness and challenges of the tertiary teaching environment; and the negative image and reputation of programming.

If Winslow's (1996) assertions that it takes approximately ten years for a novice programmer to become an expert are reasonably accurate, it is clear that the acquisition of programming knowledge and skills is not an easy process.

One of the first barriers that programming students need to overcome during the course of their studies appears to be the mastery of a completely new way of thinking (Eckerdal, Thuné & Berglund, 2005; Robins et al., 2003). Robins et al. (2003) question the cognitive requirements of learning to program; Mayer, Dyck, and Vilberg (1986) refer to this as the connection between learning to think and learning to program, whilst Eckerdal et al. (2005, p. 135) pose the question "What does it take to learn 'Programming Thinking'?" In addition to learning to think like a programmer, some researchers have considered the importance of reflection in learning to program (Cockburn, 2002; Fekete, Kay, Kingston & Wimalaratne, 2000; Schwartzman, 2006; Scott & Sewchurran, 2008, Sewchurran, 2008). Schwartzman (2006) acknowledges that confusion and uncertainty are part of learning. He concurs that reflective practices enable students to better understand (particularly in the case of programming) what is causing their problems and become more responsible for their own learning. Despite this many students struggle to overcome this barrier and react in a defensive rather than reflexive manner, which is both unproductive and problematic (Schwartzman, 2006).

The educational environment that is rarely personalized (Gomes & Mendes, 2007b) also plays its part in the problems associated with learning to program and can affect students' motivation and cognitive abilities (Simon et al., 2006). Many students entering tertiary education lack the less obvious life skills and, as a result, face challenges in their programming education as basic concepts that are presented at this level are critical to future programming skills (Jenkins, 2002).

Due to factors like time constraints and large class sizes at tertiary institutions, students often do not receive immediate feedback or individual explanations. The situation becomes problematic if a lack of exclusive attention is combined with the failure to address the different learning styles of students (Jenkins, 2002). D'Souza et al. (2008, p. 75) observe that when students struggle with programming it "affects most facets of their study, for example: their progress through their study program, their study habits, their confidence and their time management." Programming cannot be learned out of text books; programmers need to program in order to succeed (Gomes & Mendes, 2007b; Jenkins, 2002; Lahtinen, Ala-Mutka, & Järvinen, 2005).

Another barrier to understanding programming mentioned in the literature is an inability to trace code (Thomas, Ratcliffe & Thomasson, 2004). A considerable amount of effort and time may be required to code a fairly simply program, and finding mistakes is equally time consuming (Eckerdal, McCartney, Moström, Ratcliffe, & Zander 2006). In addition programming requires high level of abstraction, and demands both practice and intensive effort. "Computers are cruel. They expect things to be exact and they just don't work… unless you are pretty exact about telling them what to do" (Kinnunen et al., 2007, p. 64).

Jenkins (2002) also mentions that the reputation and image of programmers as "nerds" or "geeks" cause a barrier that may prevent student programmers from fully engaging with the discipline. One of the grand challenges of computing education is therefore to promote a positive image of computing (McGettrick et al., 2005).

McPherson (2005, p. 711) advises that "to come through the ordeal of excessive complexity [such as being faced with programming issues], we learners need some sort of self-confidence, courage, resilience or self-esteem, and some willingness to take increasing responsibility for our learning, with the planning, implementation, self-questioning, persistence and evaluation which may be involved."

Teaching tools and learning methodologies

In order to overcome these barriers to learning the art of programming, educators need to address the issue from different perspectives and offer ways of teaching more effectively by changing their course design, teaching methods, or teaching tools.

Lahtinen, Ala-Mutka, and Järvinen (2005) investigated the problems associated with learning programming in order to develop more supportive learning materials and suggested that "since success in creating a functional program is a major positive force on students' traditional programming work, materials should have more problem-solving nature instead of representing concepts" (p.17). In their study, Gomes and Mendes (2007a) felt that educators should focus on the individual student and be aware of both their knowledge level and preferred learning style; use programming patterns to give examples of good practice; and include games in the curriculum (Von Wangenheim & Shull, 2009), with the aim of enhancing problem solving abilities. Educators themselves can provide effective support (Robins et al., 2003) through a mentoring program that can help restore students' confidence in their programming abilities (D'Souza et al., 2008). Practice, persistence, social networks, and step by step instructions, or frameworks, were suggestions given by the students themselves in a study by McCartney et al. (2007).

A recurring theme in the literature is a pattern-based approach to instruction (Caspersen & Bennedsen, 2007; Muller et al., 2007). These patterns can be used by students as building blocks for developing their own programs and provide "an expert solution to a recurring design or programming problem" (Muller et al., 2007, p. 152).

Another suggested teaching method is the use of scaffolding to provide a structure that students can use as a base on which to build (Caspersen & Bennedsen, 2007; Mead et al., 2006; Thomas et al., 2004). Scaffolding should provide clear directions, with a set purpose, using methods that allow students to complete the task whilst providing feedback and assessment (Mead et al., 2006). It seems that deeper problems underlie some students' difficulties with programming since, despite being given a framework, weaker students were still unable to make use of it (Thomas et al., 2004).

Students who adopt a deep approach to learning fare better than students using a surface approach (Simon et al., 2006). A deep approach relates the learning to other topics and personal experience; whilst a surface approach follows facts and reproduces exactly what is covered in class assignments, without any questioning or true understanding. Whilst surface learning has its place in programming, competence and tacit knowledge can only be achieved if deep learning has taken place (Jenkins, 2002; Schwartzman, 2006; Scott & Sewchurran, 2008; Simon et al., 2006).

It would appear from the various studies of learning tools and teaching methodologies discussed above, that learning to program is not a comfortable or easy process, and it is understandable why some students develop a negative attitude towards the discipline.

Learning theories

One of the most significant theories of learning appears to be the five stages of adult learning developed by Dreyfus and Dreyfus and published in 1986 in *Mind over Machine: the Power of Human Intuition and Expertise* (McPherson, 2005; Mead et al., 2006). This model proposes five different levels or stages by which adult learners acquire skills. Each of these stages has individual characteristics; after starting out as novices, learners gradually progress through the stages of advanced beginner, competence, proficiency, and being an expert. The premise of the learning stages is that individuals learn skills gradually through instruction, practice, and apprenticeship.

During the novice stage, the student learns by the application of rules and following instructions. During the stage of advanced beginner, the learner starts to deal with practical situations and to recognize where to apply the previously learnt skills in an appropriate context. At this stage, the learner begins to draw from experience, but is still following instructions and learning by example. At the competence stage of learning, the learner is aware of the different ways of applying the rules and procedures to the situation at hand and must now make choices. The learners realize that not only is the outcome dependent on their own actions but they must also accept responsibility for those actions. It is this emotional involvement and experience that elevates the competent performer to the proficiency stage (Dreyfus & Dreyfus, 1986).

To accomplish the desired outcome, the proficient learners must still decide what to do and may revert to following learned rules and procedures. To reach the fifth stage, the proficient performer must not only sense what needs to be done but also unconsciously know how to attain the desired result (Dreyfus & Dreyfus, 1986). The expert stage requires many years of practice and experience; it is unlikely that it could be achieved by undergraduates and will, therefore, not be discussed any further in this paper.

It may be interesting to compare the stages described above to Cockburn’s (2002) three stages of learning, namely, following, detaching, and fluency. Cockburn believes that when individuals are in the following stage, they are concerned with finding a method that works. Inundated with new concepts and with the goal of making the procedure work, they are unable to cope with learning more methods at this stage. They need explicit instructions that result in a successful resolution. This stage appears to relate quite closely to the Dreyfus and Dreyfus’ (1986) novice and advanced beginner levels. During Cockburn’s detaching phase, students realize that they need more than one procedure or rule, since, depending on the context, even a successful procedure has limitations. The detached learner is now open to learning the alternative concepts or rules and, additionally, learning when and where to apply them, realizing that the surrounding circumstances must be considered. This stage can be likened to the competent performer stage.

Cockburn’s (2002) final stage of fluency is very similar to Dreyfus and Dreyfus’s (1986) fifth, or expert, stage. At this level, the methodology is no longer important. The way that fluent practitioners achieve their goal is unconscious and instinctive. Mead et al. (2006, p. 183) state that “despite the best intentions and efforts of the computer science education community, our programming students may or may not have the programming mastery expected of them,” and it is critical that students should be able to transfer taught concepts to new situations. This transfer of knowledge is a gradual process and the ability to apply that knowledge when appropriate is part of the learning process. Undergraduate students are unlikely to attain the expert level of skill where they transcend from their rules-based learning to instinctive, unconscious task completion. However, with self awareness, continued practice, and learning, they can become fluent or expert programmers.

Both of the theories discussed above propose that adult learners progress from one level or stage to the next, and they can be used effectively as theoretical lenses to understand phenomena of interest during this process.

Methodology

An interpretive research philosophy was used for this study in an attempt to obtain a holistic understanding of what is experienced by programming students. The first step was to identify causes, characteristics, and consequences relating to the fear of programming, as narrated by the participants. Student perceptions were then categorized into meaning units, which could be grouped into themes that would eventually lead to the construction of a model.

A phenomenological approach was used to collect and analyze the data. Phenomenology is concerned with the study of an experience from the perspective of the participant (Eckerdal et al., 2005); exploration of the meaning of a phenomenon can only happen through the narrative, or storytelling, of the person experiencing it, which is entirely subjective (Giorgi, 1997; Saunders, Lewis & Thornhill, 2007; Wimpenny & Gass, 2000). This study thus appeared particularly suited as phenomenological research requires evidence to be "derived from first person reports of life experiences" (Moustakas, 1994, p. 84). The data collected was verbal and dependant on the meaning that the experience had for the participants at the time of the interview. This was done through in-depth, unstructured, one-on-one interviews. The findings were interpreted based on the researcher's understanding and observations. Due to the need for rich descriptions of the experience, the sample size was small, and the individual narratives remained the driving force. Great care was taken to evaluate the area of concern from the perspective of the students experiencing the phenomena and, at the same time, to provide an accurate description of this experience.

This study sought to understand and describe how students manifest their fear or apprehension of learning programming and whether the final year context had any effect on their anxiety. Based on the participants' narratives, the researcher felt that it might be possible to discern their level of learning as defined by Dreyfus and Dreyfus (1986), and to attempt to correlate this to one of Cockburn's (2002) three stages of skills acquisition.

Since this research involved describing the experience of learning to program in a tertiary institution, students registered for a course with a core programming component appeared best suited to participate in the study. In order to understand whether the additional stress of being a final year student had any bearing on the learning experience and increased the associated apprehension, a third year course was selected. Due to time and resource constraints, the study was limited to the capstone course of the Bachelor of Commerce in Information Systems degree at the University of Cape Town (UCT), for which designing and building a working software system forms a core component. Participation was entirely voluntary, and any student registered for this course who had experienced concerns regarding their programming skills was encouraged to participate, regardless of gender, background, age, or culture. A written appeal and a subsequent verbal appeal to this class resulted in eleven volunteers, all of whom were interviewed. Due to a failure of the recording device, only ten interviews were transcribed. The demographics have not been analyzed as the objective of this study is to describe the various ways students have experienced their fear whilst learning to program, rather than to categorize them statistically. It is interesting to note that at the time of the interviews, none of the participants had failed a programming course whilst registered for their current degree. In addition, all participants expressed a desire to continue on to their fourth year in 2010, if their final marks met the criteria.

Analysis procedure

The interviews were transcribed verbatim by the researcher and reviewed iteratively. Analysis began after the first interview and was continued throughout the research process using Weft QDA, an open source software package designed for qualitative data analysis. The researcher began by reading through each transcript marking significant statements, which were then added to one or more of the researcher defined categories. The categories were then individually reviewed and renamed, and significant statements were moved or deleted as necessary. The transcripts were not altered in any way. The researcher then grouped the categories or meaning units and searched for themes that described these groupings. This was an intuitive, reflective, iterative process and the themes and meaning units were frequently shifted and changed until the researcher felt that relationships were emerging that could support a framework or model, whilst not detracting from the original experience of the participants.

Data Analysis and Implications

The aim of this research was to understand and describe the apprehension associated with learning programming, as well as to gain insight into students' experience of learning this discipline, particularly those who are struggling. The ultimate aim was to identify key elements of the fear caused by the nature of programming. During the analysis of the data, six main themes emerged regarding the students' experience of learning programming: the apprehension or fear associated with programming, the resulting negative perceptions, the nature of programming that gives rise to these feelings, the internal factors influencing these feelings, the external factors influencing these feelings, and finally how these feelings have affected the students' growth or skills acquisition. Excerpts from the transcripts are used in the following sections to illustrate how these themes manifested themselves.

The model represented in Figure 1 has been devised to guide the analysis and to make sense of the implications revealed by the data. This model shows fear as the tip of the "learning to program" iceberg, with internal and external factors causing the formation of negative perceptions and raising the level of fear. Underlying everything is the nature of programming, and it is only through growth that the fear of programming will dissipate.

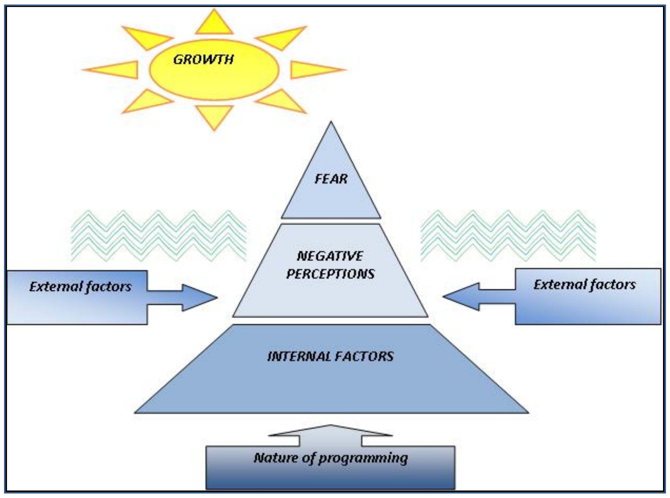

Figure 1: Iceberg Model (initial) - students' fear of programming

A brief explanation or summary of each theme is followed by an in-depth analysis of the data. As each theme is analyzed, the model is built up, and the implications of that theme are discussed. This section is followed by a discussion, or synthesis, that considers the entire model (Figure 2).

Fear of Programming

One of the initial themes that emerged from the interviews was that students do indeed perceive that there is a problem associated with learning programming. During the course of all ten interviews, participants described how the word "programming" evoked feelings of apprehension or discomfort. During the course of the interviews, words such as anxiety, panic, nightmare, and stress were used repeatedly as participants expressed their feelings regarding programming: "*it's as scary as hell… it kind of stresses me so much*." When questioned as to what they visualized when the word "programming" was mentioned, a participant responded, *"I see a lot of syntax* [nervous laugh] *symbols and words put together, that are supposed to be meaningful, but to me, I grew up speaking English, and I grew up speaking my home language, and now I have to learn this whole new different language and manipulating it is a nightmare* [strained laugh] *a little bit of a nightmare."*

Other participants expressed this a little differently but the meaning of the concern is similar, *"If I have to take programming on its own, me and him, we've got problems."* Of the ten participants, six were quite emotional whilst recounting their frustrations, *"I think it's a particular problem for me with programming, I have a huge problem with anxiety."* Even unconsciously, participants' feelings about programming continued to show throughout the interview with the words they chose to describe their learning experiences, "*…well we're forced to go in and sit in the workshop,"* with the use of the word "forced" clearly demonstrating the reluctance of the participant to attend.

Implications: Fear of programming

There appears to be little doubt that programming elicits feelings of fear. One of the causes for this feeling could be the fact that many students are first exposed to programming at the beginning of their tertiary level studies or that those with prior experience of programming may be confronted with a very different level of expectations. Gomes and Mendes (2007b) argue that the instability and change that students experience at this time probably gives rise to an initial negativity or inability to focus on a particular subject: "*You're coming from very far, having to live down here by yourself*" and "*I was really flailing around. I didn't have an idea if this is how I must manage my time, didn't know anyone, no other students to help me, just thinking back, I get emotional.*"

This feeling of fear, or rather the apprehension created by the emotional and social changes of entering a new environment, may lead to confusion or cloud understanding when the basic concepts of programming are presented (Jenkins, 2002). "*I don't know where my mind was at that time, but when they did classes, I couldn't really understand them so well.*" As initial concepts are vital to forming the strong foundations for future knowledge to be built on, it is not difficult to understand how the struggle with programming commences.

The fear factor would also lead to a lack of comfort which may inhibit the likelihood of students asking questions or initiating discussions (Bergin & Reilly, 2005b). According to Wilson and Shrock (2001), comfort level was the most reliable predictor of success and was measured by rating students' anxiety. The result of feeling fear when learning programming is an increased possibility of misunderstanding, a sense of confusion, an inability to focus and grasp key concepts, a lack of comfort, and a questioning of one's ability. All of these give rise to greater fear and negativity and the perpetuating of a vicious cycle.

Negative perceptions

With each interview, the researcher endeavored to elicit when the participant had first started to feel apprehensive about programming. In nine cases, it appeared that participants' first encounters were unclouded by negative perceptions, and the participants were open, if not keen, to learn programming: *"I was very excited about it, about learning coding."* However, as participants progressed through the first coding course, this receptiveness began to fade. *"It was towards the end of my first year that it suddenly wasn't as much fun as it used to be, second year, it's just gotten worse and worse, until now."* Apart from their own growing feeling of negativity, some of the participants experienced receiving negative feedback from others; *"For me to see an Honors student telling me how hard it is, is quite worrying."*

Implications: Negative perceptions

The change from a neutral to a negative perception of programming is an obvious cause of concern for educators, as negative perceptions are considered to be a barrier to learning (Jenkins, 2002, Simon et al., 2006). When students struggle with a particular course or have doubts about their ability to succeed, the ensuing negativity tends to permeate all their studies (D'Souza et al., 2008) and affects their confidence, time management, and study habits. Hearing negative comments about programming courses from other people, particularly those perceived to be an authority, caused participants extra anxiety and increased the fear factor.

Nature of programming

The nature of programming is often cited as a cause of learning difficulties. Although only three participants mentioned the nature of programming specifically, all participants used words such as "effort" and "frustration" when describing their experience of programming, and the failure to get a program to run was frequently highlighted. From these interviews, the emergent meaning units appear to be precision and logical thought processes, as well as patience, persistence, and practice. The time required to solve the problem and then implement the code was also an issue that appeared to be an intrinsic part of programming: *"It's this business of taking forever and not being intuitive."*

Meaning unit: Precision & logic

Whilst there are other disciplines that require precision, programming is perhaps the only discipline where a missing comma or semi-colon prevents any outcome at all. Participants were vocal about the aggravation caused by this need for absolute precision. Student: *"Then, you're like darn! Just a small little thing and you thought it was a huge error or something, and then it gets very frustrating."* Researcher: *"So you don't feel excited when you find the error?"* Student: "*I feel like I've wasted five hours. What could I have done in five hours as opposed to looking for this comma! That's what I'm thinking.* [Laugh]*"* The three participants who brought up the nature of programming in their interviews all referred to having to think or learn differently in this discipline. *"There's definitely a different way to learning it than to everything else,"* and *"It's just like my brain is not wired that way."* Another participant explained, *"I have no problems with the concepts. I grasp them quite easily. I think it's [the]transferring - trying to picture it in my head, the way it works with the theory, and then trying to translate that to code and make it work the way it's working in my head."* There appears to be a logical step by step process which enables programmers to translate their goals into code and, until this process is mastered, students are faced with a challenge which some find very hard to meet: *"but now, when I get down to do the programming, getting everything the way I want it on my blueprint, it doesn't happen - because of coding."*

Meaning unit: Patience & persistence

These two qualities are definitely not unique requirements of programming, but both are needed if students are to succeed at learning to code. Whilst effort, attitude, and other internal factors contribute to this, programming does require students to not only spend time on the tasks (patience) but also to keep trying (persistence): "*I was like, NO! I can't do this anymore because I spent hours in the labs trying to find an error in one line* [Big sigh]*.*"

Despite this effort, it appeared that participants experienced a lack of reward which contributed to elevated feelings of frustration.

Meaning unit: Practice & time

During the interviews, it was quite clear that participants viewed programming as a practical discipline that cannot be learned from theory alone: *"You can't just read up on something...; you have to actually use it to make it work"*. Eight participants mentioned that the more one practices, the easier programming or coding becomes. This is obviously time intensive, and all ten made reference to the amount of time they had invested in the practical coding part of programming: *"It's coding that takes up most of the time. I'm doing two courses this semester, and most of my time is just trying to code,"* and "*I had to give up certain things… It takes me longer to do programming,*" Most tellingly, in respect of the coding section of the third year project, "*So like we're here every single day between seven and eleven* [at night]*, trying to code. And any time any one has any free time, you sit and you try to code.*"

Implications: Nature of programming

The nature of programming is mentioned in many of the papers covered in the literature review (Kinnunen et al., 2007, Lahtinen et al., 2005, Mead et al., 2006, Robins et al., 2003, Simon et al., 2006). It appears that the precision and logic required by the nature of programming and its practical aspect, which in turn requires patience, persistence, practice, and time, all contribute to the difficulties of learning to program; this difficulty triggers the fear or apprehension, which the participants experienced. Kinnunen et al.'s (2007) reference to the cruelty of computers and Jenkins' (2002) comments on the very unforgiving nature of programming is supported by the findings of this study and directly reflects the experience of the participants. Comments on the practical nature of programming and the necessity of constant practice when applying the concepts also support the observations of Gomes & Mendes (2007b), Jenkins (2002) and Lahtinen et al. (2005). Thomas, Ratcliffe, and Thomasson (2004) mention how students struggle to trace code. This inability to discover why their program does not work and the resultant feelings of frustration was emphasized by the participants. Several participants of this study mentioned that they had to apply a different way of thinking to programming. In common with the findings of Kinnunen et al. (2007), they tended to express this as applying "logic" or thinking "logically", but found it quite difficult to explain. This new way of thinking is discussed by Eckerdal et al. (2005) and Robins et al. (2003), while Mayer et al. (1986) suggest the existence of a relationship between learning to think and learning to program.

Internal Factors

For the purpose of this study, internal factors are descriptors for psychological attributes that can only be altered by the individuals themselves: *"I've put my sweat and blood into it."* There is no immediate intervention that any educator can effect to change these factors, which seem to have a considerable impact on the participant's fear level. The sub-units identified under this theme are effort, motivation, attitude, self-efficacy, and attribution.

Meaning unit: Effort

In all ten interviews, the amount of effort that participants felt they needed to apply in order to cope with the programming course was difficult to overlook. "*It's not that I hate coding. It just fatigues you. It just drains you all the time*". Participants also felt that this considerable application of their energy was not normal practice: "*I have to work so much harder to make my systems or my applications run… It takes like five times the effort for me to make it work than I would say the average person in my class*." From the narratives of the participants, there is a clear indication that they often perceive the effort required to learn programming as a problem.

Meaning unit: Motivation

Owing to their feelings of anxiety and their negative perception of programming, the participants in this study appeared to focus on the end goal in order to motivate themselves: "*Because I knew it was my major and because I knew it was what I wanted to do*," or "*At the end, you have to code. You have to graduate and you have to pass.*" Eight participants felt that it was this need to pass the course that kept them persisting: "*If it's a must, I'll get it done.*" Three participants expressed a sense that something was missing as the motivation was coming from external rather than internal sources: "*I felt like I was just doing the course for them. I was not doing it for myself anymore*." This sense of a lack of internal motivation is expressed by a participant who stated, "*I really do love coding, but because I'm not so strong at it, it doesn't excite me as much as I would like it to.*" Lack of internal motivation was perceived to be a problem by participants.

Meaning unit: Attitude

The importance of attitude was evident in each interview. Participants mentioned how they had to keep talking to themselves in order to stay on track and meet the course deliverables: "*I still have to convince myself that it's actually worth doing… I have to drag myself there*." In spite of extremely negative feelings towards programming, a participant stated, "*I refuse to believe it's impossible* [learning to program]. *I actually just won't even let myself entertain the concept* [of failing]." All ten participants expressed strong sentiments regarding the importance of having the right attitude if they were to overcome their difficulty with programming. "*It's just a matter of changing your attitude at that moment, and then telling yourself, this is what I have to do, and you have to convince yourself you can do it.*" That this is not always easy to do was also mentioned in many of the interviews: "*And it's been trying to change my thinking in fact that I've been having a hard time with. Because I do think the way you approach* [programming] *makes a big difference.*"

Meaning unit: Self-efficacy

Considering self-efficacy to be a belief in one's ability to succeed, it became obvious from the narratives of the participants that programming was causing them to doubt their ability, with one participant musing, "*Is it really possible for you to be this smart and to not be able to figure out something that someone else can do in such a short space of time?"* and another insisting, *"But you know, I'm good at doing research and I'm good at grasping things and if you give me enough time with something then I can do it but I'm just* [pause] *I'm a bit disappointed in myself."* This self doubt directly affected the students' motivation, attitude, and readiness to apply the extra effort programming appears to require. Only one participant explicitly mentioned that he managed to overcome this obstacle: *"Although I was scared about it, I knew if I really applied myself I could do it, and I kept on telling myself that,"* but despite the professed loss of confidence, all ten participants maintained that they were determined to complete their studies.

Meaning unit: Attribution

Perseverance and determination to succeed at a difficult task may depend on how individuals credit their success or failure. Five participants mentioned that succeeding in getting a program to run gave them a great boost and encouraged them to continue: "*It's a feeling of accomplishment that I've gotten through this, I've put all this code down and it's working, there's no silly errors* *that could have been avoided,*" and *"Wow it worked! I did that! I made it work!"* In contrast, another participant appeared so discouraged with the programming struggle that any successes were attributed to luck: *"See you get to a point where you're so used to failing at something, even when you realize I did pretty good at that point, it's really hard to even acknowledge because you're thinking, ok that was probably a lucky break type set up."*

Implications: Internal factors

Understanding and addressing the internal factors that came up in this study are crucial if fear of programming is to be overcome. McPherson (2005) observed that self confidence, courage, resilience, and self-esteem were necessary if students were to overcome the complexity of learning to program. Motivation and attitude to learning are mentioned as central success factors in many papers (Gomes & Mendes, 2007b; Jenkins, 2002; Robins et al., 2003; Simon et al, 2006), and the importance of positive self-efficacy is commented on by both Wilson and Shrock (2001) and Weiner (2008). Students' attitudes to learning and the resulting behavior are also considered to be of great consequence (Gomes & Mendes, 2007b; Kinnunen et al., 2007). The participants of this study also acknowledged that their attitude was important and often expressed their battle to change it, in the face of their struggle with programming.

Motivation as a key element is illustrated by the narrative: "*When I'm working on the next exercise and it doesn't run the first time, then I get that I want it to work just as good as the other one, so I work on it to make it work and to produce the same result as the one I did before*." Suitable guidance and counseling can assist to alleviate or avoid the entrenchment of fear of programming. It may simply be enough to talk to the class generally if these symptoms are apparent and to encourage students to be self aware and to seek assistance if required.

External Factors

Whereas the internal factor theme requires that individuals are self aware and are able to initiate change themselves, the theme of external factors covers those issues requiring intervention by the institution or educators. This excludes the factor of final year, which cannot be avoided, and which is discussed separately.

Meaning unit: Lecturers

One of the first sub-units that emerged was the importance of the lecturer. All ten participants mentioned their lecturers and how their teaching style, language, enthusiasm, and availability directly impacted the students' experience. The importance of the approachability of the lecturer was a recurrent subject, and this was often perceived as the lecturer's enthusiasm for the subject: "*He's approachable and he loves what he does*". Three participants commented on their first year lecturer and how they felt this lecturer contributed to their struggle with programming. This feeling about the impact of a lecturer can be summed up by the comment, "*So it does make a difference, the lecturers, it's their ability to put across what they know*."

Meaning unit: Tutors

Although this unit could be combined with that of "lecturers" above, the participants viewed their lecturers as quite distinct from their tutors, and their experiences also appeared to be different. There were diverse perceptions of the contribution of tutors to the learning experience, with some students finding tutoring essential to getting through the course: "*But I will say definitely that the support that they give us in the form of the tutors is very helpful,*" and "*he forces you to think and learn about what you're doing.*" However others complain that, "*My colleagues are more helpful than them*" or "*the one answer that they love giving you is just Google it."* In addition, there were several comments that tutors simply did the work for the students, rather than enabling them.

Meaning unit: Peers

Peers provided another meaning unit to this category. Not only from the aspect of helping their fellow students to comprehend concepts or assist in getting programs to run, but there also appeared to be a certain degree of competition within the class which provided an impetus to succeed: "*But now everyone sees your marks, you don't want to fall behind the class and everyone*." The assistance provided by peers was a recurrent topic highlighted by all ten participants: "*Our group, because we know we're not all strong, we're helping each other,"* and "*Whenever I'm stuck, my first point of call that I make is to my colleagues*." The benefit of pair programming was also pointed out in the course of the interviews by nine of the participants.

Meaning unit: Teaching methodology

The concept of guided learning encompasses various teaching methodologies such as scaffolding, schema, constructionist, and anchor based learning. In the first semester of third year, IS students at UCT have to develop two different pilot systems in their workshops. For one system they are given a step by step guide, whilst for the other system they are given a detailed brief, but no steps are provided. By adhering to this framework, students are receiving guidance on how to proceed systematically and logically through the one system, whilst learning to apply similar programming principles and standards to the second system. This methodology appears to draw diverse observations from the participants with three explaining how this guided learning had been extremely useful: "*That works so much better because now you have something that works, and if you do have a problem you can easily reference and say where am I going wrong. So that makes it a whole lot easier*," and another three disagreeing completely with the concept: "*You just continue from the solution, so you just don't really learn much from that.*" From the discourse of the participants, it seems that the teaching methodology used in first year, for both IS and Computer Science, is inadequate. Seven participants appeared to have had discouraging introductions to programming, and most felt they only picked up the concepts taught in each year in the following year: "*Like last year, what we did in first year made sense to me. This year, what we did last year is making sense… I am building on it, but it's almost as if I'm building at the same time I'm learning*."

Meaning unit: Timing

This meaning unit was taken to represent timing in the context of teaching methodology, not time management skills. Whilst all participants recalled the tremendous effort required to program ("*Because you end up having to copy and paste almost 400 lines of code, and having to, wanting to, gain an understanding of each little piece is not exactly time friendly*"), a few pointed out that they felt there were too many concepts introduced at each session and that they would have coped better if given more time to absorb the new concepts ("*It would be better if they would give us … less steps in the same time*").

Implications: External factors

Robins et al. (2003) observed that lecturers themselves were the most effective teaching tool. All ten participants brought up their lecturers, either in a positive or negative way, but they were never neutral. Lister, Simon, Thompson, Whalley, and Prasad (2006) also investigated the effectiveness of educators and mention the importance of enthusiasm and love for the subject, which was also remarked on by participants in this study. The method of teaching programming warrants careful consideration. Format of the lectures was described as an issue by three participants: "*And in lectures, they just put things up and expect you to know it*". Tutors' teaching styles are as important as those of lecturers, and more attention should possibly be paid to recruiting tutors that are empathetic and good at human relations, rather than being excellent programmers. The role of tutors does not seem to have received much mention in the reviewed literature, however, D'Souza et al.'s (2008) coverage of mentoring suggests that the earlier students receive support, the better their learning experience will be. McCartney et al. (2007) noted that students attributed their ability to succeed with programming directly to social interaction and credited getting "un-stuck" to the assistance of friends.

Both Caspersen and Bennedsen (2007), and Muller et al. (2007) discuss pattern-based approaches to teaching programming, and a similar approach is offered by scaffolding (Caspersen & Bennedsen, 2007; Mead et al., 2006; Thomas et al., 2004). Students learn by finding similarities and differences in the patterns or using a given structure as a base to build on. These methods appear similar to the approach taken in the programming courses at UCT. Whilst Muller at al. (2007) found that the pattern-based approach helped increase students' problem solving ability, some of the participants in this study felt very negatively towards this methodology. Conversely, others found it of great assistance, suggesting that the learning level or style of the individual could be causing the difficulty, rather than the method. By taking cognizance of the participants' negative feedback, it might be possible to adapt the methodology to make it more effective for all.

The timing of the introduction of new concepts is critical according to Cognitive Load Theory (CLT) (Bannert, 2002; Kirschner, 2002; Tuovinen, 2000). The basic premise of CLT is that new ideas should be introduced when there is enough space in the working memory, so that additional blocks of knowledge can be stored in long term memory in a single unit and so become easier to access. Some participants appeared to be suffering from cognitive overload. Modifying the amount of material presented at one workshop may alleviate some of this pressure and the resultant anxiety.

Anxiety Associated With Third Year

Although anxiety caused by being in third (final) year is an external factor, it cannot be avoided as it is part of the process of completing a degree. However, it does add another dimension to the participants' experience: *"Yough! It's just hard. Third years quite intense, I must say. There's just way too much pressure."* Not unpredictably, five participants mentioned an increase in anxiety directly attributable to being in final year: *"I don't even know where to start right now and I'm supposed to be graduating and possibly getting a job next year."* All ten participants mentioned the scope of the programming required by the capstone course and how that had increased their apprehension in view of their perceived lack of skill in this area: *"…'cos there is a lot of uncertainty, especially now with this project we're going to be doing this year uhmm where it's very open ended and I think it's just the fear of …will we be able to produce something that going to be worthwhile, that's going to work out,"* and *"But there's sort of this huge project sitting there … there is a lot of pressure because at the end of the year, at the Expo, your project has to be displayed with everyone else's… and uhmm, like we all want to do a good job at it, and it's important that we understand what we're doing."*

Implications: Third year anxiety

Not only do these students have to cope with the usual difficulties of programming, they also face additional issues such as acquiring a different level of understanding, mastering more advanced programming principles, and writing or using more extensive programs or systems. A main requirement of the capstone course is the completion of a full systems development project where principles of project management are incorporated, affording students an opportunity to put these principals into practice in a real world situation. The development of life skills, such as time management, working in teams, and dealing with different stakeholders, can be uncomfortable and might add to students' fear. The imminence of graduating and beginning a career places an additional burden on these final year students, which affects their learning.

The finishing touches required to complete a full system are time consuming, and producing an implementable product demands keen attention to detail. The participants in this study repeatedly attributed the problems they were experiencing to lack of time and, also, to poor time management, with procrastination becoming an issue. This in turn added to the anxiety of completing this project: *"And the more you put it off,* [laugh] *the worse it gets. Like we've talking about this for weeks now, my group members and I. And like every day, we're okay, fine, we're going to do this tomorrow. But when we get there, we just find other things to do.* [laugh]*"* Being able to cope with the additional burdens of third year is also part of acquiring more mature life skills, the lack of which Jenkins (2002) attributes to being a cause of student attrition and failure in programming courses. Winslow (1996) emphasizes that it takes time to become an expert programmer and that it can only be achieved through experience, practice, and learning from mistakes. Schoenfeld (1999) observed that the difficulty of transition from student to worker is often underestimated, and this appears to be the view of the participants in this study as well.

There is also the concern expressed by the participants of not being able to live up to the standard set in previous years and of facing humiliation when presenting their completed projects to the public, in particular industry and learners from Western Cape schools at an annual exhibition (Scott, Sewchurran & van der Merwe, 2008). This showcase of abilities can be intimidating for those who are uncomfortable with their coding skills. Comments made by the participants clearly show that they are very aware of this aspect of third year, and it is a very real issue when considering the role of fear in students' experiences of programming. It is perhaps surprising that, in view of the constantly expressed anxiety and apprehension concerning their programming skills, all ten participants are intending to register for IS Honors, provided they make the required grade, in spite of acknowledging the major programming component involved.

Growth

Growth is needed for an individual to progress to the level of detachment or proficiency in programming. One of the objectives of this study was to discover whether the skill level or learning stage of the participant could be discerned from their narrative and to identify the coping mechanisms students employed to help them succeed. This theme encompasses reflection, responsibility, learning stage, or skill acquisition level, as well as success factors, including coping mechanisms and positive feedback.

Meaning unit: Reflection

Reflection in this context implies that students think about their learning experiences and question their grasp of the concepts: "*I don' feel confident enough to do it in a working environment*." There appears to be a general concern that they have not yet moved into the next stage of learning, and that they may only have acquired a superficial rather than deep knowledge of programming. However, the participants definitely questioned the wisdom of pursuing purely a surface learning approach and realized that more through integration was advisable: *"It's very easy to just do it without thinking what you're doing or how you are doing it. But if you step back and say, ok, what am I doing now, why am I doing it? And then try even to do it on your own, later ..., then you can remember…You have to consciously, like, not just copy and paste, without even looking at what you're copying and pasting. You have to go through it word by word, or line by line, and say ok this is what this line does, and I'm doing because if I don't this will happen, and then it will work for you.*"

Meaning unit: Responsibility

In line with the growth theme, the sub-unit of responsibility is indicative of the students' acceptance that only they are responsible for their own learning. If the lecturer is not providing adequate input, then other sources of information must be found. The participants' statements showed that they were prepared to locate their own sources and depend on their own initiative to find a solution: "*I realize that teaching is not always in the class. There are lots of other sources of information other than what you get from your lecturer,*" and, after a bad experience, "*I got the text book after that disastrous project, and I just started reading up on things and stuff like that*." Simply asking when the meaning is unclear also shows an acceptance of responsibility: "*Then I go ask people, or read about it somewhere, or asked someone to explain it in simpler terms, and that's how I sort of understood it*."

Meaning unit: Learning stage

Part of this research was also to understand and to discover from the narratives of the participants where the students were according to Cockburn's (2002) stages of skills acquisition, and Dreyfus and Dreyfus' (1986) level of learning. Three participants mentioned that they learned retroactively, feeling that they were always a year behind: "*I found my second year easier than my first year. Because in second year that's when I really understood the stuff we did in first year and why we did that stuff*." Eight were concerned that they were not at the point they felt they should be: "*I don't think my understanding is complete for a third year student, and that's the honest truth… I think there has to be a point where everything comes together and I'm still not at that point ….*" Seven participants appeared to be at what Cockburn designated the following stage: "*We're normally given, start from here, do this, do that, and then, when we have to do something on our own, we get stuck*". Some participants commented that they still found programming to be completely unintuitive: "*You almost feel like you're learning something completely new all the time, and that you don't have any internal memory about it*."

Meaning unit: Success factors & coping skills

As part of the growth theme, it seemed appropriate to explore the participants' perceptions of factors that they associated with success in programming, the mechanisms used to cope with their anxiety, and the extent of any positive feelings they had towards programming. All ten participants mentioned why they thought some of their fellow students did not appear to have any problems with programming. Enjoyment was perceived to be the number one success factor: "*I think programming, you have to love it, and be patient*." The intuitive factor was also noted: "*He just stands there and sees how it's going to go and then just sits and starts doing it. He just loves it. He enjoys doing it*." This was followed by the ability to think logically: "*I think it's to do with logic. Certain people understand things with logic and are able to apply this logic into their coding*". Another factor these participants mentioned was the ability to code without seeming to apply any effort ("*They think in code*"), whereas those experiencing difficulties stated, "*I find when I think about it I've got to take my thoughts and then translate them into code, and that's an active process*." Three participants referred to this as being a natural inborn ability ("*Something internalized. They just knew how to figure all that stuff out*"), but others believed it was the amount of practice these students put in: "*Well they probably practice all the time*" and, "*But I then also think that the more practice you do the easier it gets or the more logical it starts beginning to get*". Preferred learning style may also be a factor, with being able to picture the solution an important aid to successful programming: "*He's like "Visualize, visualize", but I just can't visualize what* [he's] *talking about*."

In order to cope with their difficulties in programming, the participants found that they needed to be proactive and take responsibility for ensuring that they understood new concepts: "*What worked for me was I started going to see one of my lecturers one-on-one,*" and "*So like just constantly asking people, if you don't understand something, it really helps a lot. I found that one of main reasons I overcame my fear of coding, or my nervousness towards coding*." As a consequence of their apprehension regarding programming, nearly all participants reported a problem with procrastination: "*He can start and finish. I have problems starting and finishing.*" When asked how they coped with their programming problem, participants detailed how they had to force themselves to just get on with it: "*You have to sit down and just do it. And the more you put it off* [laugh] *the worse it gets.*"

Finally in order to discover if there were any positive feelings associated with programming, participants were asked how they felt when a program ran successfully. This struck a note and all ten participants recalled such moments with much pleasure and satisfaction: "*It's such an exhilarating feeling,*" and "*Ah, it's just feels like success! Like you can do it! It gives you confidence as well.*" Although the participants perceive these moments to be rare, possibly due to their negativity in respect of programming, when they do happen, they carry a motivating element that seems to transcend the anxiety that has accompanied these students for three years: "*That's an amazing feeling. Like, just. It makes you feel like, Oh I did it!* [laugh] *And it just sort of wipes away all your fears and doubts for that moment*" and "*Oh gosh, it's the best feeling ever. I feel like I've won something. The few time they've run, the programs, I can't even explain how nice it is. It actually wants me to code something else.*"

Implications: Growth

From the analysis of the data, it is clear that the participants of this study had begun to question the depth of their knowledge, their grasp of the concepts, and were generally becoming more involved with programming. According to Dreyfus and Dreyfus (1986), this is a sign that the advanced beginner is moving into the competent performer stage. In spite of their professed fear, the majority of participants expressed their desire to become competent programmers ("*I WANT to be excited about it, I AM, like, I really do love coding*"), or at least have a good understanding of programming as they acknowledged the benefits this would have in their future careers ("*So I'm doing what I know I'm going to be doing in the industry, when I go out there and I'm working*").

Although surface learning has its place in programming, the participants exhibit signs of deeper learning as well, which is important for growth and competence (Jenkins, 2002; Schwartzman, 2006; Scott & Sewchurran, 2008; Simon et al., 2006). They have begun to relate their learning to their previous experience, instead of merely reproducing exactly what is covered in class, and to find that they can source the answers for themselves, whether through the world wide web, text books, or asking questions. In addition, some of the participants are finding that they are ready to adapt to using different methods to attain similar results.

This ability to use different methodologies is directly related to what Dreyfus and Dreyfus (1986) discuss as being part of becoming a competent performer. The learner is conscious that there are many different ways to apply the previously learnt skills and that there are choices to be made. The degree to which the learner accepts responsibility for these choices is indicative of whether they can be classed as a competent performer rather than advanced beginner. Almost imperceptibly, despite the fact the participants still perceive themselves to be incompetent or are very apprehensive regarding their programming capabilities, they are becoming more comfortable with the discipline: "*but it's not something that I'm terrified of anymore because of this change in the way we're being taught how to program. The shift has kind of happened gradually after first year.*" Although this participant attributes the greater ease he is feeling to the teaching method employed in final year, the underlying cause maybe that he is moving from following to detaching (Cockburn, 2002).

Even though the participants commented on their lack of internalization or intuitiveness, the assumption of responsibility and realization that something was missing appears to be indicative of their growing understanding: "*I do understand it better than I did before. I think I still need to do* *a lot of work, just in terms of really understanding what I'm doing but I'm more comfortable with it now.*" This appears particularly illustrative of moving from conscious incompetence to conscious competence (McPherson, 2005; Scott & Sewchurran, 2008). Although the learners are aware of what they do not know, they are starting to absorb and master programming. It seems that these participants are only slowly becoming aware that they have acquired a skill and have yet to acknowledge this; in fact their standard of performance is evidenced by the reality that none have failed an IS course. One of the participants even achieved the highest marks for a second year project, but still considers himself an under- skilled coder, although the fear factor has decreased.

In spite of the positive move from advanced beginner to competent performer that seems to be occurring and the subsequent increasing level of comfort or lessening of fear, when considering the theme of growth and the fact that these are final year students, it was disturbing to learn that participants in the study perceived that they lacked some important background: "*I put a break point and try to see where it's crashing but I don't understand why it crashed and I don't know what to ask because I don't understand what's happening*," and "*I don't necessarily understand which lines do what, and I won't even lie to you.*" These feelings of confusion are a recurrent complaint: "*Because even if I look at a code sometimes I not really sure what it's for or, you know, what its purpose is there, kind of thing*." This lack of mastery is a concern expressed by Mead et al. (2006) and Stamouli and Huggard (2006), who stated that it was critical that students are able to transfer taught concepts to new situations, and the importance of a stable foundation is the keystone for many of the learning theories (Caspersen & Bennedsen, 2007; Muller et al., 2007; Thomas et al., 2004). From the study, it is apparent that the participants also recognize the importance of a firm base, and part of their anxiety may have resulted from their recognition of this missing element: "*Because I think that's what held me back in second year as a coder. I had this basic knowledge that was very shaky which I got in the first year coding course. A lot of work in second year went into rebuilding and stabilizing that first year base.*"

It seems as if this growth and movement into the third stage of competent performer (Dreyfus, 2001) is happening but is not always perceived as such by the participants themselves. Their descriptions of their feelings when confronted with a successfully executed program are illustrative of how much they are actually engaging with programming, despite their stated aversion: "*Imagine if I could program, because I've seen things like* [sigh]." Additionally, although they state that the occurrences of such fulfillment are rare, it may be a matter of perception, with their negative feelings or attributions blocking out how frequently they do in fact succeed and how close they are to being competent performers by the time they graduate, with fluency and proficiency likely to occur if they continue with a fourth year of study.

Discussion/synthesis

The completed model (Figure 2) includes all meaning units. The fear of programming as evidenced by the participants of this study is what shows on the surface. This fear is triggered by the nature of programming, introduced in first year. In this study, the participants were all high performers at their secondary schools. To be accepted by the Faculty of Commerce at UCT is fairly competitive and only high scoring school leavers are selected, with mathematics being a compulsory subject. For the students who volunteered for this study, programming was the first subject where they did not easily succeed: "*I don't remember struggling in high school. I don't remember struggling in school at all. So to this day it's one of those things where if I have to fail a test, I'm like oh my goodness I can't fail a test. It's like a really serious strain,*" and similarly, "*I don't like looking bad, I like having things all together. When I feel not in control, it's part of my anxiety.*" No matter what their intelligence and prior academic excellence, programming requires effort, practice, new approaches to thinking, a deep understanding rather than surface memorization, in

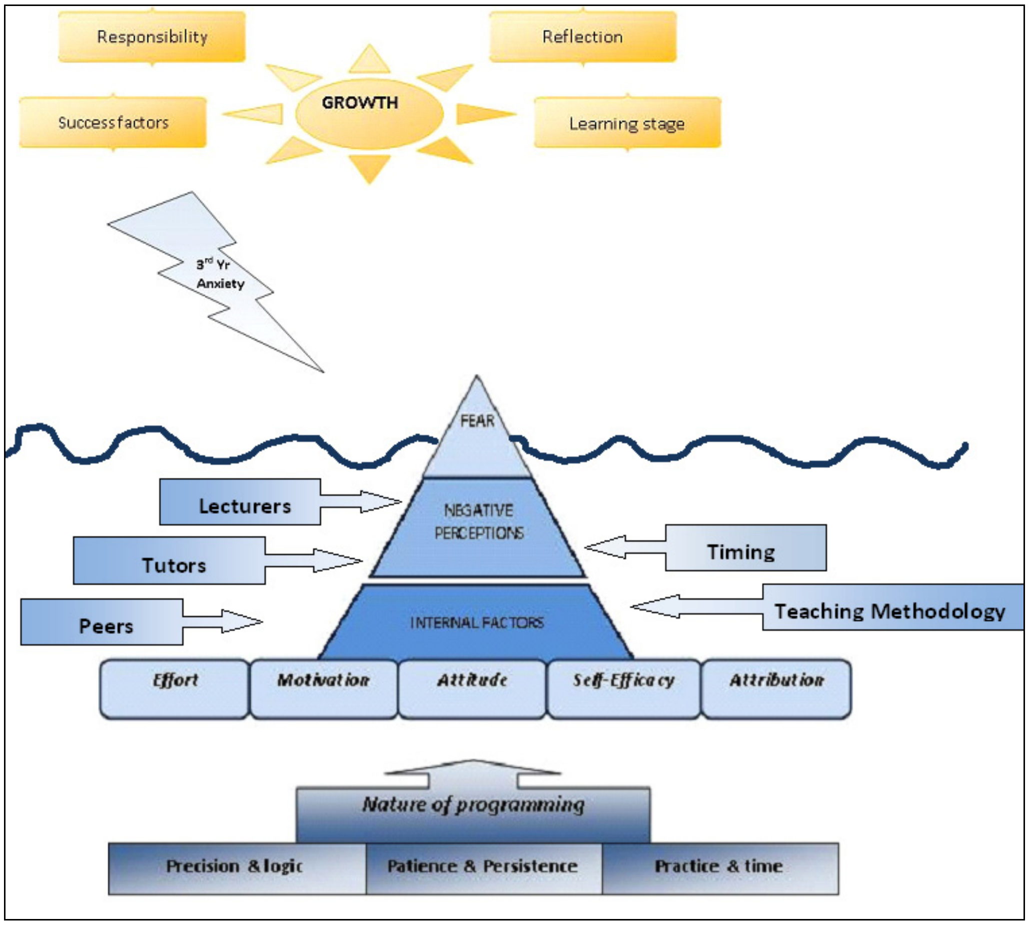

Figure 2: Iceberg Model (complete) - students' fear of programming

other words, a completely different skill set. Faced with the new environment and the new discipline, these students found themselves struggling in a subject for possibly the first time, even those who had prior exposure to programming at school: "*But I just didn't like the fact that I had failed, so I just moved to something else.*" The internal factors of effort required, motivation, attitude, self-efficacy, and attribution came into play and possibly set up the negative perceptions of programming. This in turn was affected by the various external factors, and so the fear of programming was built up. Advancing into third year was a further factor which also contributed to the fear factor and to metaphorical icebergs of various sizes floating on the sea of learning to program. It is only through growth, applying such factors as reflection, taking responsibility, looking for the positives, and building on success, that the fear factor will be reduced or disappear, and these students will be able to detach from following the rules and move onwards from advanced beginners to competent performers and, finally, to proficient practitioners.

Bearing in mind that none of these participants actually failed a programming course while registered for their current degree and all are planning to continue on to Honors if possible, it appears from this study that the fear of learning programming is not due solely, or even mainly, to its inherent nature but rather emerges from the nature of the students themselves, their personalities, and the high standards and expectations that they set themselves. Fear of failure, rather than fear of programming, would seem to be an overbearing issue, with programming acting as the trigger or the focus because this is the course where most of their energy is expended.

Conclusion

From this study, the essence of the fear experienced by these participants when learning programming at a tertiary institution is alienation, both from themselves and the discipline. The fear forms a very real, almost physical barrier that causes intense emotion and causes loss of confidence and self-doubt. It is very unsettling and gives rise to negative perceptions that affect attitude and motivation. Although these participants were motivated to succeed, the ensuing struggle resulted in classic symptoms of procrastination and avoidance, which only added to the problem.

The amount of time and effort required to overcome their difficulties intrude into other courses. Loss of confidence and a low comfort level inhibit the ability to question, resulting in more confusion and loss of self esteem. Increased levels of anxiety inhibit the appreciation of programming, and a liking for the subject is clearly seen as a success factor: *"So that could be the case for them, just because they're good at it, they enjoy doing it and that encourages them to learn more… I think that once I am completely comfortable with it and I know exactly what I'm doing, then I'll get back into enjoying it again."*

These participants have managed to reach the final year of their undergraduate degree in spite of their negative perceptions and ever present anxiety. Their narratives are inspiring and may add impetus to educators' continuing search for teaching excellence, particularly in respect to programming courses. It is the participants' stories, in their own voices, which have been described in this research, and it is hoped that their experiences may trigger points for educators involved in teaching programming. There appears to be a critical need for intervention, and some suggestions have been made. For example, formal, one-on-one consultations with the lecturer at strategic intervals may help students to overcome their fears sooner and, as a result, increase their comfort and enjoyment levels. The aim, as always, is to empower the students to reach their full potential, without the disabling factor of fear hindering the process.

This research investigated whether students experience fear when learning programming at a tertiary institution IS and attempted to describe "how" and "what" they experience. A number of significant statements were made by participants, which enabled themes to be developed and a composite model built to try and explain the relationship between fear and learning to program. It is important to note that these interviews took place at a particular time and place and are a reflection of these participants' experience at that time, but, nonetheless, it attempts to take subjective experiences of a common phenomenon and add them together to give a realistic account of a lived experience. The close correlation between the findings of this study and current literature provides further validity.

Great care was taken during the interviews to apply the principles of phenomenological research, and it is the actual words of the interviewees that provide validity for this study (Moustakas, 1994). Although the researcher is employed at UCT, it is not an academic appointment and does not involve contact with students. Additionally, the researcher is not a programmer so there was less prior socialization involved and this allowed the researcher to be relatively unbiased when searching for meaning units. However, the researcher did find remaining constantly in the present moment and beginning each interview with freshness, as well as the phenomenological process of epoche, that is the putting aside of one's own experiences and prejudgments (Moustakas, 1994), difficult to maintain at all times. This was especially true in the later interviews, where there was a tendency to stray into grounded theory methodology, which had to be consciously checked. It appears that this is recognized as an area of concern with phenomenological studies (Wimpenny & Gass, 2000). Nevertheless, as the researcher was fully conscious of this important aspect of a phenomenological study and attempted to adhere to Moustakas's (1994, p. 22) critical principal of being "completely open, receptive and naïve in listening to, and hearing research participants describe their experience of the phenomenon being investigated," it is hoped that the individual narratives remain the driving force in this study as opposed to the researcher's preconceptions.

As this is a qualitative, phenomenological study, the number of participants is small in comparison to a quantitative study, and statistical generalization of the findings to the student population as a whole would be questionable. However, according to Lee and Baskerville (2003, p. 230), "In interpretivism, a theory's pertaining only to the setting where it was developed would not detract from its validity or scientific status." Therefore, for this qualitative study, the object is not to generalize from the sample to the population but rather to generalize from the individual findings to a theory within a particular setting (Lee & Baskerville, 2003, Merriam, 1988), the setting here being an affluent African university. Phenomenology appeared to be ideally suited to this study but a possible weakness is the difficulty of maintaining the methodology throughout all interviews, as described above.

It is hoped that this research will help educators to understand the inhibiting force of fear and, through this understanding, not only enable their students to graduate as competent programmers, but also help students to realize that they are more prepared than they think, in spite of how they feel. The relationship between the fear of programming, the underlying negative perceptions, external factors, such as effective teachers and teaching strategies, internal factors, such as psychological attitude and motivation, and student growth is highlighted in this study by the voices of the students themselves.

References

Bannert, M. (2002). Managing cognitive load — Recent trends in cognitive load theory. *Learning and Instruction*, *12*(1), 139-146.

Bergin, S., & Reilly, R. (2005a). Programming: Factors that influence success. *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, February 23 – 27, St. Louis, Missouri, USA.

Bergin, S., & Reilly, R. (2005b). The influence of motivation and comfort level on learning to program. *Proceedings of the 17th Annual Workshop of the Psychology of Programming Interest Group*, June 29 – July 1, Brighton, UK.

Boyle, R., Carter, J., & Clark, M. (2002). What makes them succeed? Entry, progression and graduation in computer science. *Journal of Further & Higher Education*, *26*(1), 3-18.

Bruce, C., Buckingham, L., Hynd, J., McMahon, C., Roggenkamp, M., & Stoodley, I. (2004). Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university. *Journal of Information Technology Education*, 3, 143-160. Retrieved from http://www.jite.org/documents/Vol3/v3p143-160-121.pdf

Byrne, P., & Lyons, G. (2001). The effect of student attributes on success in programming. *SIGCSE Bulletin, 33*(3), 49-52.

Caspersen, M. E., & Bennedsen, J. (2007). Instructional design of a programming course: A learning theoretic approach. *Proceedings of the Third International Workshop on Computing Education Research*, September 15-16, Atlanta, Georgia, USA.

Cockburn, A. (2002). *Agile software development*. Boston, MA: Addison-Wesley.

Dreyfus, H. L. (2001). How far is distance learning from education? *Bulletin of Science Technology Society*, *21*(3) 165-174.

Dreyfus, H. L., & Dreyfus, S.,E. (1986). *Mind over machine: The power of human intuition and expertise in the era of the computer.* New York, NY: Free Press.

D'Souza, D., Hamilton, M., Harland, J., Muir, P., Thevathayan, C., & Walker, C. (2008). Transforming learning of programming: A mentoring project. *Proceedings of the Tenth Conference on Australasian Computing Education*, January 1, Wollongong, Australia.

Eckerdal, A., Thuné, M., & Berglund, A. (2005). What does it take to learn 'programming thinking'? *Proceedings of the First international Workshop on Computing Education Research*, Seattle, October 1-2, WA, USA.

Eckerdal, A., McCartney, R., Moström, J. E., Ratcliffe, M., & Zander, C. (2006). Can graduating students design software systems? *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, March 1 – 5, Houston, Texas, USA.

Fekete, A., Kay, J., Kingston, J., & Wimalaratne, K. (2000). Supporting reflection in introductory computer science. *Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education,* March 7 – 12, Austin, Texas, United States.

Giorgi, A. (1997). The theory, practice, and evaluation of the phenomenological method as a qualitative research procedure. *Journal of Phenomenological Psychology*, *28*(2), 235-260

Gomes, A., & Mendes, A.J. (2007a, June 14 - 15). An environment to improve programming education. *Proceedings of the 2007 international conference on Computer systems and technologies*, Bulgaria.

Gomes, A., & Mendes, A. J. (2007b) Learning to program – difficulties and solutions. *Proceedings of the 2007 international convergence on Engineering education,* September 3-7, Coimbra, Portugal.

Hagan, D., & Markham, S. (2000). Does it help to have some programming experience before beginning a computing degree program? *Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSE conference on Innovation and Technology in Computer Science Education*, July 11 – 13, Helsinki, Finland.

Hager, P. (2005). New approaches in the philosophy of learning. *Educational Philosophy and Theory*, *37*(5), 633-634.

Jenkins, T. (2002). On the difficulty of learning to program. *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sci*ences*,* August 27-29, Loughborough, UK. Retrieved April 1, 2009 from http://www.ics.heacademy.ac.uk/Events/conf2002/jenkins.html

Kinnunen, P., McCartney, R., Murphy, L., & Thomas. L. (2007). Through the eyes of instructors: a phenomenographic investigation of student success. *ICER '07: Proceedings of the third international workshop on Computing education research*, September 15-16, Georgia, USA

Kirschner, P. A. (2002). Cognitive load theory: Implications of cognitive load theory on the design of learning. *Learning and Instruction*, *12*(1), 1-10.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H.M. (2005) A study of the difficulties of novice programmers. *Annual Joint Conference Integrating Technology into Computer Science Education: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science educatio*n, Monte de Caparica, Portugal.

Lee, A. S., & Baskerville, R. L. (2003). Generalizing generalizability in information system research. *Information Systems Research*, *14*(3), 221-243.

Lister, R., Simon, B., Thompson, E., Whalley, J. L., & Prasad, C. (2006). Not seeing the forest for the trees: Novice programmers and the SOLO taxonomy. *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, June 26-28, Bologna, Italy.

Mayer, R. E., Dyck, J. L., Vilberg, W. (1986). Learning to program and learning to think: What's the connection? *Communications of the ACM*, *29*(7), 605–610.

McCartney, R., Eckerdal, A., Moström, J. E., Sanders, K., & Zander, C. (2007). Successful students' strategies for getting unstuck. *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, June 23-27, Dundee, Scotland.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., & Kolikant, Y.B.D. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *SIG-CSE Bulletin, 33*(4), 125-180.

McGettrick, A., Boyle, R., Ibbett, R., Lloyd, J., Lovegrove, G., & Mander, K. (2005). Grand challenges in computing: Education - A summary. *The Computer Journal*, *48*(1), 42-48.

McPherson, I. (2005). Reflexive Learning: Stages towards wisdom with Dreyfus. *Educational Philosophy and Theory*, *37*(5), 705-718.

Mead, J., Gray, S., Harner, J., James, R., Sorva, J., Clair, C.S., & Thomas, L. (2006). A cognitive approach to identifying measurable milestones for programming skill acquisition. *ITiCSE-WGR '06: Working group reports on ITiCSE on Innovation and technology in computer science education*, June 26 – 28, Bologna, Italy.

Merriam, S. B. (1988). *Case study research in education: A qualitative approach*. San Francisco: Jossey-Bass.

Moustakas, C. E. (1994). *Phenomenological research methods*. Thousand Oaks, California: Sage Publications, Inc.

Muller, O., Ginat, D., & Haberman, B. (2007). Pattern-oriented instruction and its influence on problem decomposition and solution construction. *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, June 23-37, Dundee, Scotland.

Palma, P. D. (2001). Viewpoint: Why women avoid computer science. *Communications of the ACM, 44*(6), 27-30.

Perry, R. P., Stupnisky, R. H., Daniels, L. M., & Haynes, T. L. (2008). Attributional (explanatory) thinking about failure in new achievement settings. *European Journal of Psychology of Education*, *23*(4), 459-475.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137-172.

Saunders, M., Lewis, P., & Thornhill, A. (2007). *Research methods for business students* (4th ed). Harlow, England: Pearson Education.

Schoenfeld, A. H. (1999). Looking toward the 2first century: Challenges of educational theory and practice. *Educational Researcher*, *28*(7), 4-14.

Schwartzman, L. (2006). A qualitative analysis of reflective and defensive student responses in a software engineering and design course. *Proceedings of the 6th Baltic Sea Conference on Computing Education Research*, February 01, Uppsala, Sweden.

Scott, E. C. (2008). From requirements to code: Issues and learning in IS students systems development projects. *Journal of Information Education Technology (JITE) - Innovations in Practice (IIP)*, 7, 1-13. Retrieved from http://www.jite.org/documents/Vol7/JITEv7IIP001-013Scott.pdf

Scott, E.C., & Sewchurran, K. (2008). Reflection-in-action: Using experience to reconstruct meaning in a learning environment. *International Conference on Information Technology in Education (CITE2008)*, December 12 – 14, Wuhan, China.

Scott, E.C., Sewchurran, K., & van der Merwe, N. (2008). Windows to the real world – A comparison of IS project management student learning. *Second European Conference on Information Management and Evaluation*, September 11 – 12, Royal Holloway, University of London, UK.

Sewchurran, K. (2008*).* Toward an approach to create self-organizing and reflexive information systems project practitioners. *International Journal of Managing Projects in Business*, *1*(3), 316-333.

Simon, S., Fincher, S., Robins, A., Baker, B, Box, I., Cutts, Q., de Raadt., M., Haden, P., Hamer, J., Hamilton, M., Lister, R., Petre, M., Sutton, K., Tolhurst, D., & Tutty, J. (2006). Predictors of success in a first programming course. *ACM International Conference Proceeding Series*; *Proceedings of the 8th Australian conference on Computing education*, January 16 – 19, Hobart, Tasmania, Australia.

Stamouli, I., & Huggard, M. (2006). Object oriented programming and program correctness: The students' perspective. *Proceedings of the Second International Workshop on Computing Education Research*, September 09 – 10, Canterbury, United Kingdom.

Thomas, L., Ratcliffe, M., & Thomasson, B. (2004). Scaffolding with object diagrams in first year programming classes: Some unexpected results. *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, March 3 – 7, Norfolk, Virginia, USA.

Tuovinen, J. E. (2000). Optimising student cognitive load in computer education*. Proceedings of the Australasian Conference on Computing Education*, December, Melbourne, Australia.

Von Wangenheim, C. G., & Shull, F. (2009). To game or not to game? *Software, IEEE*, *26*(2), 92-94.

Weiner, B. (2008). Reflections on the history of attribution theory and research: People, personalities, publications, problems. *Social Psychology*, *39*(3), 151-156

Wilson, B. C., & Shrock, S. (2001). Contributing to success in an introductory computer science course: A study of twelve factors. *Proceedings of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education*, March , Charlotte, North Carolina, USA.

Wimpenny, P., & Gass, J. (2000). Interviewing in phenomenology and grounded theory: Is there a difference? *Journal of Advanced Nursing*, *31*(6), 1485-1492.

Winslow, L. E. (1996). Programming pedagogy - A psychological overview. *SIGCSE Bulletin*, *28*(3), 17-22.

Biographies

Christine Rogerson is the system support and operations officer in the Finance department at the University of Cape Town, South Africa, and holds the SAP Finance training portfolio. She decided to venture into academia and completed her BCom Honors in Information Systems in 2009. Currently she is registered for a MCom in Information Systems. The dissertation is focused on replacing the current staff classroom-based training with online training, and the affect this has on learning.

Elsje Scott is a Senior Lecturer at the Department of Information Systems, University of Cape Town, South Africa. She has been teaching object-oriented programming for the past 20 years. Currently her main research interest is the coherent practice of the capstone course with the emphasis on as-lived project experiences and the development of project practitioners. The research is underpinned by theories and models for cognition and learning in order to develop IS competence and life-long learning.