Factors Affecting Course Outcomes in Introductory Programming

Susan Wiedenbeck
Drexel University
susan.wiedenbeck@cis.drexel.edu

Deborah LaBelle
Pennsylvania State University-Delaware County
dm19@psu.edu

Vennila N.R. Kain
Media Laboratory
Masssachusetts Institute of Technology
vennila@vennila.net

Keywords: POP-I.B, programmer education, POP-II.A, novices, POP-5.A, mental models, POP-5.A, self-efficacy

Abstract

Learning to program is difficult for many students. Although several factors that affect learning to program have been identified over the years, we are still far from a full understanding of why some students learn to program easily and quickly while others flounder. Two constructs that may affect learning to program are self-efficacy and mental models. Self-efficacy is the individual's judgment of his or her ability to perform a task in a specific domain (Bandura 1986). A mental model is a person's internal (mental) representation of real world objects and systems (Norman 1983). Separate research on self-efficacy and mental models has shown that both are important to knowledge acquisition and transfer. Using a path-analytic approach, this research investigates the joint effects of self-efficacy, mental model, and previous experience on learning to program in an introductory course. The results show that self-efficacy for programming is influenced by previous programming experience, and student self-efficacy increases substantially during an introductory programming course. Furthermore, students' mental models of programming influence their self-efficacy, and both the mental model and self-efficacy have a direct effect on overall success in an introductory course.

Introduction

The dropout and failure rates in introductory programming courses at the university level are evidence to the fact that learning to program is a difficult task. One source suggests that the dropout and failure rate is as high as 30 percent (Guzdial & Soloway, 2002). Decisions about majoring in computer science and related fields are often determined by a student's success or failure in the introductory course. If a student drops out, fails, or passes with a struggle, that student is unlikely to enroll for a follow-on course. In spite of research on factors that influence the enrolment and success of novices in introductory programming, it is still not fully understood what makes an introductory programming course positive and successful for some, but difficult and frustrating for others.

Several factors that may influence novices' success in an introductory university-level programming course have been discussed in the literature. The most frequently mentioned factor is previous computer programming experience, usually in secondary school (Bunderson & Christensen, 1995; Byrne & Lyons 2001; Hagan & Markham 2000; Sacrowitz & Parelius 1996; Taylor & Mounfield 1989; Wilson & Shrock 2001). These studies provide converging evidence that prior programming experience has a positive effect on success in an introductory university course. Other factors that may affect course success have been less well investigated. Two recent studies have shown a positive relationship of mathematics or science background to computer programming success (Byrne &

Lyons 2001; Wilson & Shrock 2001). A relationship between student learning styles and learning to program has been found by both Byrne and Lyons (2001), and Thomas, Woodbury, and Jarman (2002). Other intriguing factors that have been addressed in recent studies include student attributions of success to oneself or to outside forces (Wilson & Shrock 2001), students' course outcome expectations (Rountree, Rountree & Robins 2002), and self-efficacy (Wilson & Shrock 2001). A factor of potential interest that has been studied in basic computer training, but not to our knowledge in computer programming, is computer playfulness during training (Martocchio & Webster 1992; Potosky 2002). Finally, there is a body of research on the student's mental model of programming in relation to success in specific programming tasks (Cañas, Bajo & Gonzalvo 1994; Corritore & Wiedenbeck 1991; Soloway & Ehrlich 1984; Wiedenbeck, Ramalingam, Sarasamma & Corritore 1999).

In summary, there is a substantial literature on factors affecting success in the initial programming course. However, more research is needed to determine which are the key factors, how they interact with each other, and how they combine to affect course outcomes. We are interested in creating and testing models that incorporate factors that appear to be important on theoretical or empirical grounds. Two important constructs in cognitive and social cognitive theory of the past 20 years are mental models and self-efficacy. Our goal in this research is to study self-efficacy and mental models of beginning programmers, explore the relationship between these concepts, and investigate their combined influence on course performance.

Background on Self-Efficacy and Mental Models

Self-Efficacy and Its Role in Learning

Bandura (1986, p. 391) defines self-efficacy as "people's judgments of their capabilities to organize and execute courses of action required to attain designated types of performance." Self-efficacy beliefs are a key element in human performance over a very broad range of situations, for example, efficacy for work tasks, for physical activities, for personal relationships (Bandura 1977, 1986; Gist & Mitchell 1992). Self-efficacy is important in learning activities because learning involves more than just acquiring skills. As Bandura says, "competent functioning requires both skills and self-beliefs of efficacy to use them effectively" (1986, p. 391). In learning situations, self-efficacy influences the use of cognitive strategies while solving problems, the amount of effort expended, the type of coping strategies adopted, the level of persistence in the face of failure, and the ultimate performance outcomes (Bandura 1986; Gist & Mitchell 1992; Zimmerman 1995). According to self-efficacy theory (Bandura 1977, 1986), judgments of self-efficacy are based on four sources of information: the individual's performance attainments, experiences of observing the performance of others, verbal persuasion, and physiological reactions that people use partly to judge their capableness and vulnerabilities. Of these four sources, the most important is performance attainments, that is, the individual's evaluation of the outcomes of his or her direct attempts to perform an activity.

Educational researchers recognize that, because skills and self-beliefs are so intertwined, one way of improving student performance is to improve student self-efficacy. Interventions to improve student self-efficacy focus on specific skills or knowledge and target the four sources of information that students use to evaluate their self-efficacy, as defined above. Providing students with direct hands-on experiences in an activity is critical, since the strongest source of information is performance outcomes (Bandura 1977, 1986; Pajares 1996; Zimmerman 1995). Making hands-on experiences positive is also important, especially in the early stage of learning, when the task may seem overwhelming. Attempts have also been made, with some success, to increase self-efficacy in learning by peer modeling of tasks, verbal persuasion, or other types of social influences, such as cooperative learning environments (Bandura 1986; Compeau & Higgins 1995; Gist, Schwoerer & Rosen 1989). Several studies of self-efficacy in learning to use personal computers or computer applications have been carried out (e.g., Compeau & Higgins 1995). However, to our knowledge the only study that has directly targeted introductory computer programming students is Wilson and Shrock (2001). This study, contrary to theoretical expectations, did not find a significant effect of student self-efficacy on course outcomes.

Mental Models and Programming

Norman (1983) defines mental models as predictive representations of real world systems. People create internal representations of objects and processes in the world, and they use these mental representations to reason about, explain, and predict the behavior of external systems. Mental models are critical in debugging a process when things go wrong because the mental model supports the person in reasoning about and localizing possible faults (Allen 1997). Mental models have been studied in many domains and situations (e.g., Stevens & Gentner 1983). In recent years, the mental models concept has been popularized by practitioner magazines and web sites in areas such as human-computer interaction (e.g., McDaniel, 2003).

Programming is a cognitive activity that requires the programmer to develop abstract representations of a process and express them in the form of logic structures. In the case of creating, modifying, reusing, or debugging a program, the programmer must also translate these abstract representations into completely correct code using a formal language. Having a well-developed and accurate mental model is likely to affect the success of a novice programmer in an introductory programming course. A programmer's mental model could encompass useful knowledge about how programs work in general, stereotypical ways of solving common programming problems, and how a particular program is structured and functions, as well as knowledge about the syntax and semantics of a specific language (Cañas et al. 1994).

Mental models (also referred to as schemas or plans) have been shown to play an important role in program comprehension (Soloway & Ehrlich 1984; Littman, Pinto, Letovsky & Soloway 1986; Nanja & Cook 1987; Pennington 1987; Corritore & Wiedenbeck 1991; Wiedenbeck et al. 1999) and also in comprehension-related tasks, such as modification and debugging. For example, Littman and his colleagues (1986) found strong effects of mental model formation in a program modification task. Participants were asked to modify a program but were not given any explicit instructions about how to approach the task. The results showed that programmers who first attempted to systematically read and comprehend the program were much more successful in doing the modifications than programmers who jumped immediately into making modifications. The difference in performance between programmers who built a mental model of the program and those who did not was especially great in modifications that involved interactions with code in other parts of the program. Similar results were reported by Nanja and Cook (1987) in a comparison of novices and experts debugging a program. A conclusion that can be made from these studies is that novices' success in programming tasks may be increased by greater attention to building a good mental model of the program. These studies of mental models in programming do not deal directly with the issue of success in introductory programming courses. However, the relationship between a good mental model and success in programming tasks suggests that having a good mental model may be an important contributor to course outcomes.

Model of Factors Affecting Performance in Introductory Programming

This study proposes a model of performance of novice programmers based on their previous programming experience, self-efficacy for programming, and mental model. Our model is represented in Figure 1. The ovals represent factors, or variables, that may affect success in introductory programming. The arrows represent predicted relationships between the variables. These relationships are directional, as shown by the arrow heads. Multiple links *into* an oval indicate that an oval is affected by more than one factor. Multiple links *out of* an oval indicate that the factor represented by the oval affects more than one other factor. With respect to terminology, if variable A points to variable B, then in that relationship A is referred to either as the independent or the predictor variable, and B is referred to as the dependent or response variable.

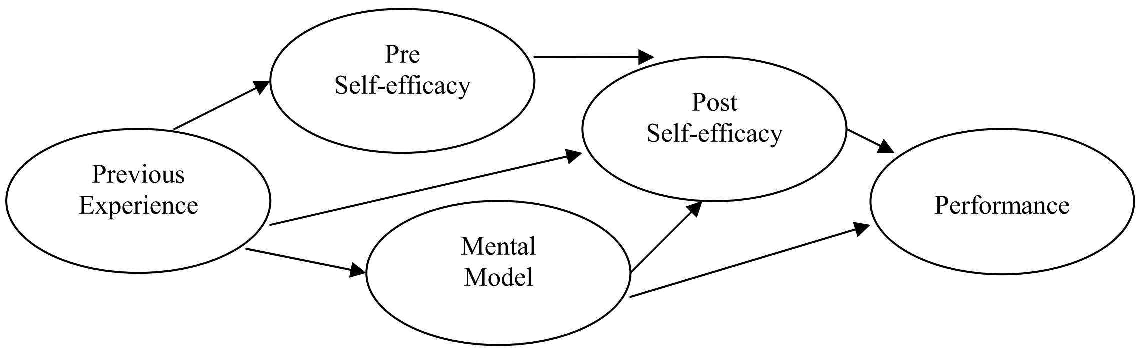

Figure 1 – Proposed model of factors affecting student performance in an introductory programming course

In line with the considerable existing research (Byrne & Lyons 2001; Bunderson & Christensen, 1995; Hagan & Markham 2000; Sacrowitz & Parelius 1996; Taylor & Mounfield 1989; Wilson & Shrock 2001), we expect previous experience to be important to success in an introductory programming course. Our hypothesis is that previous experience acts as a significant predictor of both students' self-efficacy and mental models of programming, which in turn predict course performance. Based on self-efficacy theory (Bandura 1977, 1986), we expect that students' self-efficacy will increase as a result of instruction and continued hands-on exposure to programming, so post-self-efficacy should be higher than pre-self-efficacy. We also hypothesize that students' mental models of programming will have a significant effect on their self-efficacy beliefs. That is, having a clear mental model of what programs do and how they do it will increases students' feelings of efficacy about programming. Finally, it is expected that both mental model and self-efficacy will explain a significant amount of course performance. In keeping with Bandura, students' knowledge content and organization, as represented in the mental model, and their self-beliefs will be intertwined in successful course outcomes.

Our model includes the concept of *mediation* of the effect of variables. A variable may not affect performance directly, but instead may affect it indirectly through another variable. This is seen in the pre-self-efficacy variable, which is hypothesized to affect performance indirectly through its effect on post-pre-self-efficacy. Therefore, pre-self-efficacy does not have a direct effect on performance, but its effect is mediated by the intervening variable, post-self-efficacy. Likewise, the previous experience variable's effect on performance is indirect and mediated through several other variables. The model also suggests that one variable may affect another variable both directly *and* indirectly. For example, the mental model variable is shown as having a direct effect on performance, and also an indirect effect on performance via its effect in strengthening post-self-efficacy.

Methodology

Participants

Seventy-five students took part in the study. The students were enrolled in four sections of an introductory C++ programming course at a large public university in the United States. The four sections were taught by four different instructors who coordinated with each other to cover the same material. Twenty-five of the participants were female and 75 were male. The average age of the participants was 20 years. The majority of the participants were second or third year undergraduates. The participants came from a wide variety of majors, ranging from computer science to agricultural studies. The majority were not computer science major, but were taking the course as a requirement of their major or for personal interest.

The participants had low experience in computer programming. Based on their self-reports, it was found that on average they had taken 1.27 programming courses in secondary school and had been exposed to 1.37 programming languages. Aooroximately half of the participants had not previously studied programming.

Materials

The materials included a background questionnaire, a self-efficacy scale, and two instruments to measure mental models.

The background questionnaire used five questions measuring the breadth of participants' prior computer and programming background: number of courses taken that used computer applications as a mandatory part of course work (e.g., spreadsheets, databases), number of programming courses taken, number of programming languages used, number of programs written, and length of the programs written.

Self-efficacy was measured using the Computer Programming Self-Efficacy Scale (Ramalingam & Wiedenbeck 1998). This validated instrument was used previously by Wilson and Shrock (2001) in their research on success factors in introductory computer science courses. The scale consists of thirty-three items that ask students to judge their capabilities in a wide range of programming tasks and situations. Based on self-efficacy theory, four categories of questions are included in the scale:

- Simple programming tasks (9 questions, e.g., "I would be able to write a program that computes the average of 3 numbers")
- Complex programming tasks (11 questions, e.g., "I would be able to comprehend a long, complex multi-file program")
- Independence and persistence (8 questions, e.g., "I would be able to find ways of overcoming the problem if I got stuck at a point while working on a programming project")
- Self-regulation (4 questions, "I would be able to find a way to concentrate on my program, even when there were many distractions").

Responses were marked on a 7-point Likert scale ranging from "not confident at all" to "absolutely confident."

The students' mental models were evaluated by using two measurements, program comprehension and program recall. The program comprehension booklet consisted of six short C++ programs (each 15-20 lines long). The programs consisted of a class definition, a constructor, a member function of the class, and a main function. (See Appendix A for an example.) Each of the programs was followed by a list of five true/false questions covering each of the information categories developed by Pennington (1987) to measure the mental model of programmers: elementary operations, control flow, data flow, program function, and program state. These categories have been used in more recent research on mental models (Good & Brna in press), including object-oriented programming (Wiedenbeck et al. 1999).

Program recall has been used as a measure of mental organization or mental models in past programming research, e.g., Shneiderman (1976). Recall is usually measured as the number of lines recalled correctly. Our program recall booklet, modeled on Shneiderman's, contained a C++ program that dealt with temperature conversions. The program consisted of 27 lines of code.

Procedure

The study was carried out over the course of a fifteen week semester in two parts. The first part took place in the second week of the semester, and the second part in the thirteenth week of the semester.

The first phase of the study involved collecting the student programming background information and having students complete the self-efficacy scale, which yielded the pre-self-efficacy score.

The second part of the study involved completion of the two programming tasks designed to assess the student's mental model and a repetition of the same self-efficacy scale used in the first part. The participants were given a program comprehension booklet containing the six programs and associated questions. For each of the six programs, they had 1.5 minutes to study the program and two minutes to answer the questions. The participants were not allowed to look back at the programs while answering the questions. After completing the program comprehension booklet, participants were given the recall booklet. They had five minutes to study the program, then closed the booklet and had five more minutes to recall and reproduce the program from memory, as best they could. The participants also completed the self-efficacy scale, yielding the post-self-efficacy score.

The performance measure was the student's final course grade and was obtained from the instructor at the end of the semester. For the purposes of the experiment a participant's course grade (given as a letter grade: A, B, C, etc.) was translated to a numerical scale of 0-9, where 0 represented failure in the course and 9 represented the highest level of achievement.

Results

Self-Efficacy of Novice Programmers

The alpha-reliability of the self-efficacy scale was .98, indicating a highly reliable scale. The mean pre-self-efficacy score was 94.63 and the mean post-efficacy score was 163.37 out of a maximum possible score of 231 (see Table 1). Participants' self-efficacy increased significantly over the course of a semester of instruction (t = 12.78, p<.0001). The mean increase in self-efficacy was 68.75.

	Mean	StdDev	Min	Max
Pre-SE	94.63	49.51	33	219
Post-SE	163.37	41.36	52	229

Table 1 – Self-efficacy data (N=75)

To understand better the changes in self-efficacy over the course, we divided the participants into quartiles of equal size based on their pre-self-efficacy scores (Table 2). Change in self-efficacy was calculated as the difference between post-self-efficacy and pre-self-efficacy scores. The data were analyzed with a one-way ANOVA. The ANOVA was significant, F(3,71) = 12.83, p<.0001. The results show significant effects of time of measurement (pre vs. post), quartile, and the interaction between time and quartile. The ANOVA was followed by a Tukey range test to determine specifically how the groups differed from one another. The results indicate that the group with the highest pre-self-efficacy (Q4, group mean = 165.05) experienced the least increase in self-efficacy. This group differed significantly (p<.05) from groups Q1-Q3, which experienced much larger increases in efficacy (Table 2). Groups Q1, Q2, and Q3 did not differ significantly from one another.

	Pre-SE Mean	Post-SE Mean	Change Mean	Change StdDev	N
Q1	46.47	134.42	87.95	46.68	19
Q2	66.39	159.94	93.56	32.01	18
Q3	99.11	170.05	70.95	31.98	19
Q4	165.05	188.89	23.84	40.25	19

Table 2 – Descriptive data for the four quartiles grouped on pre-self-efficacy

Analysis of the Model

The correlations of variables in the model are shown in Table 3. Significant correlations exist between a number of pairs of variables. The correlations are suggestive but do not identify causalities, nor do they identify the effects of multiple independent variables on dependent variables, including the ultimate dependent variable, course performance. A major goal of this study was to evaluate whether the data was consistent with our model as a whole. To this end, we used path analysis (Kerlinger & Pehazur 1973).

	Previous experience	Pre-self-efficacy	Post-self-efficacy	Mental model	Performance
Previous experience	1.00				
Pre-self-efficacy	.63**	1.00			
Post-self-efficacy	.61**	.51**	1.00		
Mental model	.16	.05	.32**	1.00	
Performance	.25*	.10	.36**	.48**	1.00

*Table 3 – Correlation of variables (*p<.05, **p<.01)*

Path analysis consists of a series of multiple regressions used to analyze the relationships of variables in a model (Kerlinger & Pehazur 1973). Path analysis and other similar techniques, collectively known as structural equation modeling, have been used widely in educational research to study the relationship of multiple variables to each other and to educational outcomes (e.g., Horn, Bruning, Schraw, Curry, & Katkanant 1993; Zimmerman, Bandura & Martinez-Pons 1992). In the computer domain path analysis techniques have been used to study the learning of end-user applications (Compeau & Higgins 1995), data modeling skills (Ryan, Bordoloi & Harrison 2000), and instrinsic motivation in computer training (Davis & Wiedenbeck 2001; Martocchio & Webster 1992).

In our model (Fig. 1) each arrow represents a possible relationship of a predictor (independent) variable on a response (dependent) variable. A guideline for the number of subjects needed in a path analysis is 10 subjects per path in the model (Kerlinger & Pehazur 1973). Our N of 75 falls within this guideline. Figure 2 shows the results of the analysis. The strengths of each relationship are depicted as "path coefficients," or standardized regression weights, which vary between 0 and 1 (shown on the arrows in Figure 2). A significant path coefficient indicates that there is indeed a reliable causal relationship between the predictor and response variable. All paths predicted in the model were significant ($p<.05$) *except for* the path from previous experience to mental model. The R^2 value associated with each dependent variable indicates how much of the variance in the variable is explained by the predictor variables feeding into it. As Figure 2 shows, the variance explained is moderate to high (Cohen, 1977), except for the effect of previous experience on mental model.

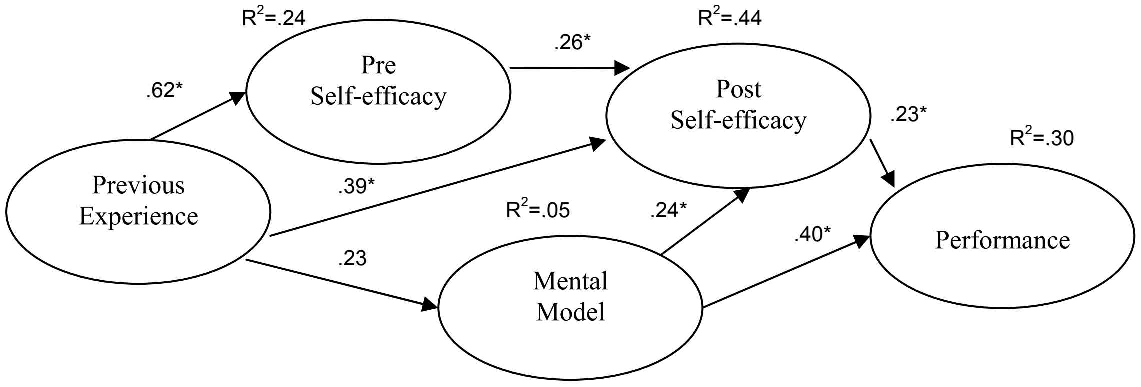

*Figure 2 – Results of the analysis of relationships in the model (*p<.05)*

In addition to analyzing the individual relationships, path analysis is concerned with analyzing the fit of the overall model with the data. This is done using a Chi-square test. The non-significant path from previous experience to mental model was eliminated for this analysis. The Chi-square test of this reduced model with four degrees of freedom yielded χ^2=1.35, p>.85. Contrary to the usual interpretation of the p-value, in this analysis the non-significant result represents an adequate fit. The non-significant p-value literally means that our model, which contains a theory-based subset of all possible paths, is as good a predictor of performance as a model containing every possible path. Consequently, this more parsimonious subset of relationships adequately represents the important influences on student performance among these variables.

Based on the results of the path analysis and the test of the fit of the model we revised the model, eliminating the path from previous experience to mental model. The revised model is shown in Figure 3 and is discussed in the following section.

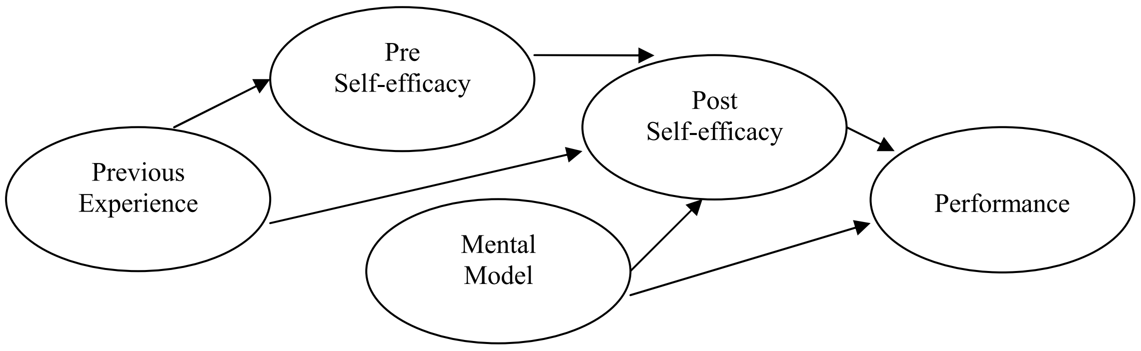

Figure 3 – Revised model of factors affecting student performance in an introductory programming course (eliminating non-significant path)

Discussion

Self-Efficacy, Mental Models, and Programming Performance

The self-efficacy of students increased significantly over the course of a semester of instruction. This is consistent with self-efficacy theory. Learners evaluate their ability to perform based on direct experience with tasks. A semester of concentrated programming instruction coupled with frequent hands-on programming tasks is likely to increase self-efficacy, unless the tasks are much too difficult for the learners. Our results also support the proposition that individuals' changes in self-efficacy are, at least in part, a function of their pre-self-efficacy. This is shown in the quartiles analysis. The three lower quartiles showed a significant increase in efficacy, with group 2 registering the greatest increase. However, the group with the highest initial self-efficacy experienced the least increase in efficacy as a result of a semester of instruction in C++ programming. This is a sensible result because a strong sense of self-efficacy is not easily changed, but weak self-efficacy is more malleable (Bandura, 1986). Thus, the result is consistent with self-efficacy theory.

The highest quartile is different from the other groups in another way as well. The standard deviation for this group was almost double its mean change in self-efficacy (Table 2). This implies that some students in the group increased in efficacy, but others decreased. Further analysis showed that slightly more than one-fifth of this group experienced a decrease in self-efficacy. This suggests that these students overestimated their ability to cope with the challenges of the introductory course. This underlines the volatility of self-efficacy when measured in an early stage of learning. Students who have low knowledge and experience in a task may not be able to accurately gauge their level of capableness. They may base self-efficacy judgments on tasks that are not particularly relevant to programming (e.g., using computer applications) or on generalized self-beliefs ("I can excel in most academic situations"), which may lead to some readjustment of self-beliefs once the student engages actively in programming.

Our results show (Fig. 2) that previous experience is a strong predictor of pre-self-efficacy, as expected from prior research. The relationship is positive: students with higher previous experience have higher pre-self-efficacy. Interestingly, previous experience also predicts post-self-efficacy. This indicates that students' prior secondary school experience continues to affect their perceptions of their capabilities even near the end of a semester of programming instruction. If a student continues to study programming, it seems clear that at some point this previous secondary school experience will lose its predictive value, i.e., students will base their self-beliefs on their more recent programming experiences. However, at this early stage, beliefs that arise from prior experience still have weight. An interesting question is how long the prior secondary school experience persists in affecting students' self-beliefs.

Contrary to our original expectations, the previous secondary school experience did not affect student's mental models of programming. Our initial prediction was that the previous programming knowledge and skills of incoming students would correlate well with the student's mental model at a later point in the course. This was not the case. An explanation of this result might be that students in secondary school do not gain too much in terms of mental models of how programs work. This may be affected by a number of factors: the intensity and rigor of the prior course, the types of programming activities, the amount of hands-on practice, and the programming language used. In terms of research follow-up, a first need is to replicate the result. If the result is replicated, a goal for future research is to pinpoint why students' previous secondary school experience does not directly affect their mental models measured later in their university course. Based on the current findings, it appears that the contribution of students' previous experience is most reflected in self-beliefs.

The results also show that having developed a strong mental model increases beliefs of self-efficacy, as seen in the post-self-efficacy measure. Students who are able to develop a stronger mental model in the programming domain express higher beliefs about their capability to carry out a range of programming tasks. This fits with self-efficacy theory. Students' ability to carry out tasks, such as comprehending or modifying a program, strengthen their self-efficacy beliefs.

Finally, both what student know, as represented by their internal mental model, and what they believe about themselves, as represented by their self-efficacy, affect their performance in the course. Both factors have a substantial influence on the course outcome. Together they account for 30 percent of the variance in the course grade, a high amount in behavioral research (Cohen, 1977). Instructors have always recognized the importance of what students know. The results of this study underline the parallel importance of students' self-beliefs on course outcomes.

Limitations of This Study

Several limitations of this study should be acknowledged and ultimately addressed in future reseach. First, the study took place in a relatively short time span of a 15 week semester. The results might be quite different in a full year course. It would be very informative to track changes in student self-efficacy and mental models, and their effects on performance at mid-year and end-year intervals. Second, the diversity of the students in the population of the current experiment raises questions that cannot be answered with the existing data. A portion of the participants were computer science majors, but the rest came from a variety of disciplines. We would like to investigate computer science students separately from other students whose interests and motivations may be more similar to end-user programmers; however, the number of students in the current study is not sufficient to analyze subgroups. Additional cohorts are needed to do this. Third, the self-reporting of previous experience is a limitation, since students may have difficulty remembering details about their previous experience accurately. They may also interpret questions differently than expected. To address this problem it would be advisable to directly inventory the skills of students at the beginning of the study by having them carry out quantifiable tasks that yield a prior experience score. Fourth, the measure of performance based only on the final course grade is not an ideal measure of success. Final grades are a rather blunt instrument, and it might be worthwhile to consider additional means of measuring student achievement. Fifth, our results do not agree with Wilson and Shrock's (2001) study that failed to find a significnt relationship of self-efficacy to course outcomes. The methodologies of the two studies were very different, and this is a possible explanation. Nevertheless, the conflicting results are a compelling reason for further study.

Finally, we believe that an important, and as yet unstudied, area of research is the study of students who fail to complete the introductory programming course. Introductory programming courses justifiably have a reputation of high drop-out rates. If students drop out before completing a longitudinal study such as ours, we fail to capture information that would help us to understand the problems they face. When designing interventions to help student succeed, we may be missing the students most at risk.

Conclusion

Pedagogical Interventions in Intorducotry Programming Courses

This research evaluated a model of factors affecting student success in an introductory programming course. A theory-based model such as ours can provide a basis for future interventions. Based on this research, we see two paths of intervention, one focused on students' mental models and the other on self-efficacy.

Computer science educators appear to be familiar with the mental models concept and its value in student learning. This study of students in an introductory programming course confirms the importance of the student's mental model in programming. The mental model affects success directly, and also strenghtens self-efficacy, which is another factor in overall success. This implies that teaching should pay close attention to development of the students' mental model. The goal of building good mental models could be approached by instruction that engages the student in hands-on learning tasks that make conceptual ideas concrete. Cañas et al. (1994) advise doing this through tracing the logic of programs. They found that students who have help creating a mental model using a tracing mechanism gain a better understanding of the semantics of a program. Important programing tasks, including program debugging and modification, often involve tracing activities. Therefore, they appear to be possible tasks to develop the student's mental model. Such tasks involving both program comprehension and writing code have the advantage of being realistic, while also strengthening the mental model through reasoning about consequences of actions.

Our study also supports the importance of student self-efficacy in introductory programming. Self-efficacy interventions have been carried out in other fields with success, and that increases the likelihood that similar interventions, adapted for programming instruction, would also be successful. A first issue of concern is supporting students' self-efficacy rather than undermining it. Self-efficacy can be undermined if instruction and practice are too far in advance of students' current feeling of capableness. Students should be challenged, but they are likely to give up if they are overwhelmed.

Interventions to support and increase self-efficacy follow from self-efficacy theory. According to Bandura (1986), these include performance successes, observation of the performance of peers, social persuasion, and the monitoring of one's physiological state. The need for performance successes indicates that students must steadily carry out tasks of increasing difficulty, until they have a history of solid attainments. Frequent but small hands-on programming activities would be likely to build the history of success more than less frequent, large assignments. For students to monitor their capableness, timely and sufficient feedback is necessary. Observing the perforamnce of peers has also been successful as a self-efficacy strengthening intervention (Bandura 1986; Compeau & Higgins 1995). Peer modeling appears to be successful because the student recognizes the model as similar to him or herself, that is, not an expert who has all the answers. In programming courses, peer modeling could be "live" in a classroom with a peer working through a problem while other students watch, or it could be done by students viewing a video of a peer successfully planning and executing a progrmming task. In peer modeling it is important that the viewers see the model confronting difficult situations and overcoming them. The modeling should not be scripted to eliminate struggle because the point is for students to see how obstacles are overcome. Social persuasion as a method of improving self-efficacy is related to work groups. Students working together, especially if they have different levels of self-efficacy, are in a position where social persuasion takes place. An instructor can facilitate social persuasion in the classroom or online by forming work groups of students with different levels of capableness and giving them tasks that promote interaction of group members. Lastly, students monitor their own physiological state to infer their capableness. Students who are in an ongoing state of anxiety about their programming course feel less efficacious. To support their self-efficacy, instructors can strive for a classroom that is calm and non-competitive. Making ample help availabe and student errors recoverable also aids in creating comfort.

Acknowledgements

This work has been funded in part by the EUSES Consortium via the National Science Foundation (ITR-0325273). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation

References

Bandura, A. (1977) Self-efficacy: Toward a unifying theory of behavioral change. *Psychological Review*, 84(2), 191-215.

Bandura, A. (1986) *Social Foundations of Thought and Action*. Prentice Hall, Englewood Cliffs, NJ.

Bunderson, E.D. & Christensen, M.E. (1995) An analysis of retention problems for female students in university computer science programs. *Journal of Research on Computing in Education*, 28(1), 1-15.

Byrne, P. & Lyons, G. (2001) The effect of student attributes on success in programming. *ITiCSE: Proceedings of the 6th annual conference on Innovation and technology in computer science education*. ACM Press, NY, 49-52.

Cañas, J.J., Bajo, M.T. & Gonzalvo, P. (1994) Mental models and computer programming. *International Journal of Human-Computer Studies*, 40(5), 795-811.

Cohen, J. (1977). *Statistical Power Analysis for the Social Sciences*. Academic Press, New York.

Compeau, D.R. & Higgins, C.A. (1995) Computer self-efficacy: development of a measure and initial test. *MIS Quarterly*, 19(2), 189-211.

Corritore, C. L. and Wiedenbeck, S. (1991) What do novices learn during program comprehension? *International Journal of Human-Computer Interaction*, 3(2) 199-222.

Gist, M.E and Mitchell, T.R. (1992) Self-efficacy: A theoretical analysis of its determinants and malleability. *Academy of Management Review*, 17, 183-211.

Good, J. And Brna, P. (in press) Progrm comprehension and authentic measurement: a scheme for analysing descriptions of programs. *International Journal of Human-Computer Studies*.

Davis, S. and Wiedenbeck, S. (2001) The mediating effects of intrinsic motivation, ease of use and usefulness perceptions on performance in first-time and subsequent computer users. *Interacting with Computers*, 13, 549-580.

Gist, M. E., Schwoerer, C. and Rosen, B. (1989) Effects of alternative training methods on self-efficacy and performance in computer software training. *Journal of Applied Psychology*, 74, 884-891.

Guzdial, M. & Soloway, E. (2002) Log on education: teaching the Nintendo generation to program. *Communications of the ACM*, 45(4), 17-21.

Hagan, D. and 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*, ACM, NY, pp. 25-28.

Hasan B. (2003) The influence of specific computer experiences on computer self-efficacy beliefs. *Computers in Human Behavior*, 19(4), 443-450.

Horn, C., Bruning, R., Schraw, G., Curry, E. and Katkanant, C. (1993) Paths to success in the college classroom. *Contemporary Educational Psychology*, 18, 464-478

Karsten R. & Roth R.M. (1998) Computer self-efficacy: a practical indicator of student computer competency in introductory IS courses. *Informing Science*, 1(3), 61-68.

Kerlinger, F.N. & Pehazur, E.J. (1973) *Multiple Regression in Behavioral Research*. Holt, Reinhart & Winston, New York.

Martocchio, J. J. Webster, J. (1992) Effects of feedback and cognitive playfulness on performance in microcomputer software training. *Personnel Psychology*, 45, 553-578.

McDaniel, S. (2003) What's your ideal of a mental model? http://www.boxesandarrows.com/ archives/ whats_your_idea_of_a_mental_model.php. Accessed Feb. 3, 2004.

Norman, D.A. (1983) Some observations on mental models. In D. Gentner and A.L. Stevens, Eds., *Mental Models*, Erlbaum, Hillsdale, NJ.

Pajares, F. (1996) Self-efficacy beliefs in academic settings. *Review of Educational Research*, 66(4), 543-578.

Pennington, N. (1987) Comprehension strategies in programming. In E. Soloway and S. Iyengar, Eds., *Empirical Studies of Programmers: Second Workshop*. Ablex, Norwood, NJ, pp. 100-113.

Potosky D. (2002) A field study of computer efficacy beliefs as an outcome of training: the role of computer playfulness, computer knowledge, and performance during training. *Computers in Human Behavior,* 18, 241-255.

Ramalingam V. & Wiedenbeck S. (1998) Development and validation of scores on a computer programming self-efficacy scale and group analyses of novice programmer self-efficacy. *Journal of Educational Computing Research,* 19(4), 365-379.

Ryan, S.D., Bordoloi, B. & Harrison, D.A. (2000) Acquiring conceptual data modeling skills : the effect of cooperative learning and self-efficacy on learning outcomes. *Data Base for Advances in Information Systems*, 31(4), 9-24.

Sacrowitz, M.G. and Parelius, A.P. (1996) An unlevel playing field: women in the introductory computer science courses. *Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education*, ACM, New York, pp. 37-41.

Shneiderman, B. (1976) Exploratory experiments in programmer behavior. *International Journal of Computer and Information Sciences*, 5 (2), 123-143.

Soloway, E. & Ehrlich, K. (1984) Empirical studies of programmer knowledge. *IEEE Transactions of Software Engineering*, SE-10 (5), 595-609.

Gentner, D. & Stevens, A.L. (1983) *Mental Models*. Lawrence Erlbaum, Hillsdale, NJ.

Taylor, H. & Mounfield, L. (1989).Exploration of the relationship between prior computing experience and gender on success in college computer science. *Journal of Educational Computing Research*, 11(4), 291-306.

Thomas, L., Woodbury, J. & Jarman, E. (2002). Learning styles and performance in the introductory programming sequence. *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education.* ACM Press, NY, 33-37.

Wiedenbeck, S., Ramalingam, V., Sarasamma, S. & Corritore, C.L. (1999) A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting with Computers*, 11, 255-282.

Wilson, B.C. & Shrock S. (2001) Contributing to success in an introductory computer science course: s study of twelve factors. *Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education, ACM Press*, NY, pp. 184-188.

Zimmerman, B. J. (1995) Self-efficacy and educational development. In A. Bandura (Ed), *Self-Efficacy in Changing Societies*, Cambridge, Cambridge University Press, pp. 203-231.

Zimmerman, B.J., Bandura, A. & Martinez-Pons, M. (1992) Self-motivation for academic attainment: the role of self-efficacy beliefs and personal goal setting. *American Educational Research Journal*, 29(3), 663-676.

Appendix A: Example Program Used in Mental Models Task

```
#include <iostream.h>
class Car
{
private:
            int Passengers, Speed;
public:
            Car(int p, int s);
            void check_speed_limit();
};
Car::Car(int, int)
{
            Passengers = p;
            if (p == 0)
                Speed = 0;
            else
                Speed = s;
}
void Car::check_speed_limit()
{
            if (Speed >= 55)
                cout << "Over the limit! Slow Down!!! \n";
}
int main()
{
            Car mycar(1, 25);
            mycar.check_speed_limit();
            return 0;
}
```

Questions

1. Is the speed of mycar set to 25? (operations)
2. Is the output statement executed before the speed is checked? (control flow)
3. Does the value of Passengers affect the value of Speed? (data flow)
4. When the cout statement is reached, is the value of Speed less than 55? (state)
5. Does the program compare the speed of two cars? (function)