Factors Affecting the Success of Non-Majors in Learning to Program

Susan Wiedenbeck
College of IST, Drexel University
3141 Chestnut Street
Philadelphia, PA 19104 USA
+1 215 895 2490
susan.wiedenbeck@cis.drexel.edu

ABSTRACT
The introductory programming course is difficult for many university students, especially students who have little prior exposure to programming. Many factors affecting student success have been identified, but there is still a dearth of knowledge about how key factors combine to affect course outcomes. In this study we develop and empirically test a model integrating three factors of importance in learning to program: previous programming experience, perceived self-efficacy, and knowledge organization. The participants were non-majors. The findings showed that perceived self-efficacy increased significantly during a semester course. Previous experience affected perceived self-efficacy but not knowledge organization. Both perceived self-efficacy and knowledge organization had an effect on the course grade, as well as on success in a specific programming task, debugging. The results on self-efficacy also suggested that the participants were overconfident about their programming capabilities. The contribution of this paper is the identification of the joint effects of an important set of factors for programming success by non-majors.

Categories and Subject Descriptors
K.3.2 [**Programming Languages**]: Computer and Information Science Education – *Computer science education.*

General Terms
Experimentation.

Keywords
Learning to program, self-efficacy, knowledge organization, non-majors, end-user programmers, debugging.

1. INTRODUCTION
Learning to program is recognized as a difficult task for students. Many students in their introductory course struggle to understand the underlying concepts and to overcome the challenges of implementing error-free programs that meet the stated requirements. Not surprisingly, there is a wide spread of course outcomes. For example, it has been estimated that 15-30 percent of students enrolled in an introductory course either drop out of the course or fail it [20]. Other reports cite even higher drop out and failure rates [27].

High drop out rates are a concern to computer science educators and the IT profession because of the potential shortage of professional developers. However, another concern is the non-professional, or end-user, programmer [29]. Today there is a proliferation of work-related and personal situations in which end users need to program, for example, creating macros, spreadsheet formulas, and dynamic web applications in the workplace [32, 36, 37, 50] or programming domestic technologies, such as electronic thermostats [6], in the home. It is estimated that the number of end-user programmers is approaching 55 million in the United States [8], so this is an important group of programmers.

We argue that no matter what kind of programming end users do, they will be much better prepared if they have successfully completed a formal course of instruction in programming. Reasoning about the logic of a conditional statement embedded in a spreadsheet formula is the same as reasoning about a conditional in Java or C++. End-user programmers increasingly need programming training for success in their future careers. This is recognized by educators and students themselves and is the reason why many students in fields such as business and sciences are required or choose to take an introductory programming course.

Because of the importance of end-user programming activities it is important for computer science educators to recognize the needs of non-majors and to support them in their introductory course. There is ample evidence of the value of previous programming experience in successfully completing introductory programming courses (see Section 2). It is likely that non-majors enter the first programming course with less previous experience in programming than CS majors. If that is the case, non-majors may be more likely to fail to complete the course or to complete it with a low grade.

To support students in the introductory programming course, whether majors or non-majors, we need to understand the cognitive and social-cognitive factors that affect their success in learning. Cognitive factors of high interest in learning to program include previous programming experience and knowledge

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICER'05, October 1–2, 2005, Seattle, Washington, USA.
Copyright 2005 ACM 1-59593-043-4/05/0010...$5.00.

organization. Social-cognitive factors include self-beliefs, specifically self-efficacy in this research.

Although there have been many studies of factors for success in programming, as discussed below, has only considered one or a few factors. Correspondingly, most prior research has not built and tested integrated models of groups of factors and how they interrelate. Our objective in this research is to develop and test a predictive model of performance that identifies key factors *and their interrelationship*s that lead to successful performance. We focus on non-majors because of the increasing importance of end-user programming activities and the known difficulties that end users face, especially in developing dependable, error-free programs [7, 31]. The contribution of this work is to create and empirically validate such a model, leading to a better understanding of pedagogical interventions that may improve performance.

In the following section we discuss the prior research on factors leading to success in introductory programming and present our research model. Section 3 describes the methodology, and Section 4 presents the results. The results are discussed in Section 5 and the conclusions are in Section 6.

2. PRIOR RESEARCH AND MODEL

2.1 Previous Programming Experience and Other Factors from Prior Research

In the CS literature the most frequently mentioned factor for success in introductory programming is previous programming experience [9, 10, 21, 34, 39, 47, 52]. These studies were carried out using different methodologies and sometimes different participant populations, but nevertheless they provide converging evidence that previous programming experience has a positive effect on success in an introductory university course.

Other factors that may affect course success have been studied more sparsely. Two recent studies have shown a positive relationship of mathematics or science background to computer programming success [10, 52]. A relationship between student learning styles and learning to program has been found in two studies [10, 48]. Other intriguing factors that have been addressed in recent studies include student attributions of success to oneself or to outside forces (e.g., luck) [52] and students' course outcome expectations [38]. A factor of potential interest that has been studied in basic computer training, but not to our knowledge in programming, is computer playfulness [25, 33]. A negative factor affecting student success is a high amount of game playing by students [52].

In our model of programmer performance we chose to use previous experience because it is the most highly studied and highly supported factor from existing studies of learning to program.

2.2 Self-Efficacy

According to Bandura, self-efficacy is a judgment by an individual of how capable one is of successfully carrying out a course of action to attain a certain outcome [3, 4, 5]. As Bandura says, "perceived self-efficacy is concerned not with the number of skills that you have, but with what you believe you can do with what you have under a variety of circumstance" [4, p. 36]. An individual's judgment of his or her ability to perform a task within a specific domain has many behavioral consequences. Bandura argues that self-efficacy beliefs influence people's choice of activities, the amount of effort they expend, their persistence in overcoming difficulties, and, not least, their performance outcomes. Individuals with high self-efficacy believe that they can succeed in challenging tasks. Accordingly, they are motivated to attempt such tasks, and they apply their best efforts to achieve them. Individuals with high self-efficacy are more likely to persist in spite of difficulties, initiating coping strategies to overcome problems. They are also likely to persist in challenging tasks in the face of competing demands.

Individuals' self-efficacy beliefs come from four sources of information [5, 17]. These include: (1) personal experiences of mastery of a task, (2) second-hand experiences, such as observing a peer carry out a task, (3) verbal persuasion, for example, encouragement by others, and (4) emotional arousal, consisting of monitoring one's levels of fatigue, stress, and anxiety in order to assess self-efficacy. Of the four sources of self-efficacy beliefs, personal experience of successfully mastering a task is the most direct and most powerful.

Bandura [5] identifies three main dimensions on which self-efficacy judgments may vary. These include generality, level, and strength. All three have important performance implications. Generality concerns the breadth of efficacy judgments. Self-efficacy is specific to a certain task or activity. It is not a character trait or generalized belief, such as general self-confidence. Because of its task-specificity, an individual may have high self-efficacy in one domain, such as driving a car, but low self-efficacy in another, such as computer programming. Nevertheless, transfer of self-efficacy beliefs across related activities may occur, for example, across different programming languages. The dimension of level is concerned with the effect of task difficulty on efficacy judgments. Simply stated, people's judgments of self-efficacy in a task depend on the level of task difficulty that they are exposed to or accustomed to performing. For example, a student in an introductory programming course may have high self-efficacy for writing a 100 line program in a certain language, but low self-efficacy for writing a 1,000 line program in the same language. Finally, the dimension of strength concerns how strongly held are people's self-efficacy beliefs, or put in another way, how much would it take to change their beliefs. Strong judgments are not easily changed, whereas weak ones are more malleable.

The theme of malleability is important in self-efficacy theory because it poses the potential of improving people's performance via increasing their self-efficacy [17]. Efficacy beliefs have been shown to be malleable, especially in the early stages of skill development [3, 4, 5, 13, 49]. Too much malleability would not be desirable, if it led to steep drops in self-efficacy based on a single poor outcome. However, malleability is not likely to lead to radical changes in self-efficacy. Instead, people employ unique combinations of rules, or heuristics, to weigh and integrate the efficacy information obtained from the various efficacy indicators that form their self-efficacy judgments.

Self-efficacy theory has been widely applied in educational research. While intellectual ability and domain knowledge are major factors in achievement in educational settings, perceived self-efficacy also plays a strong role. Those with the same level of cognitive skill development vary in their intellectual performance

depending on the strength of their self-efficacy beliefs [53]. Furthermore, since self-efficacy is malleable, it has the potential to be an additional route to higher student achievement. Numerous research studies show that a person's efficacy beliefs in a domain have a strong relationship with skill attainment, e.g., [18, 22, 28, 30, 41, 42, 43, 44, 54, 55].

In spite of its demonstrated importance in education and training, self-efficacy theory has rarely been applied to the domain of introductory computer programming. One study of introductory programming students at the university level [52] found, against expectations, that there was no effect of self-efficacy on course outcome. Another study [34] found, as predicted by theory, a significant effect of self-efficacy on course grades. Because of the importance of self-efficacy in learning and its far-reaching effects on behavior we use it as one of the predictive factors in our model.

2.3 Knowledge Organization in Programming

Psychological studies of programmers have shown that more proficient programmers are distinguished from less proficient programmers by their organization of programming knowledge [1, 26]. In program recall tasks [15, 41], using short programs, the more skilled programmers were able to recall a well-structured program accurately after very short exposure to it. Less skilled programmers, given the same amount of time to study the program, recalled much more poorly. However, given a program with the lines randomly shuffled, both groups recalled equally poorly. This result replicates the results of memorization of chess formations [11, 14], and circuit diagrams [16] by high and low skilled individuals.

In all these domains the interpretation of the results is that a person who is successful at memorizing large amounts of material can, at a glance, chunk the materials to be memorized into meaningful groups, based on their functional organization of domain knowledge in long-term memory [1, 11, 14, 15, 16, 26, 45]. By contrast, all individuals do poorly memorizing the shuffled materials, since the shuffling disrupts any patterns that one could use to link to organized knowledge in memory.

Applying this to programming, a programmer who notices that four consecutive lines of code are part of a loop that outputs data can remember them as a single unit. This reduces the burden on short-term memory and allows the programmer to more efficiently transfer the information to long-term memory. At retrieval time the programmer outputs the chunks and can, if necessary, use the context provided by the chunks to reconstruct elements that he or she has difficulty remembering precisely. This gives the programmer an advantage in memorization and recall over a less skilled individual, who treats every piece of information as an independent unit to be memorized separately. Less skilled programmers lack sufficient organized, meaningful knowledge patterns in memory [1, 15, 26, 45] and therefore do not have a basis for quick recognition of typical code chunks. Programmers who cannot chunk effectively simply focus on rote memorization of the material, usually line-by-line in sequential order. As a result, they can only memorize a small number of lines of code in the time available. At retrieval time, they are doubly handicapped; they recall few lines because they memorized few lines and they lack the context of the meaningful chunks to reconstruct partly remembered lines.

In our model we include this important factor of programming knowledge organization. Knowledge organization can be studied by asking the programmer questions about a program. However, it is difficult to devise a set of questions that is valid and reliable for measuring such knowledge-in-the-head without bias. Instead, based on the early research on knowledge organization in programming, we chose to use a program recall task of a well structured and meaningful program.

2.4 Research Model

Bandura outlines two principal ways in which causality of self-efficacy may be established experimentally [4]. One method is to carry out an intervention in which the participants' perceived self-efficacy is raised to an initial preselected level and the effects on the outcome behaviors are subsequently measured. This is followed by a second intervention to raise perceived self-efficacy to a higher level and measure the behaviors again at the new level. If there is a significant difference between the first and second measures, then the causality of self-efficacy is established. Examples of this method are reported in [5].

A second way of studying the effects of self-efficacy is to test multivariate relations using regression analysis. This method is normally used when multiple factors are hypothesized to affect performance. The objective is to study experimentally a proposed model of the strength and relationships of the factors in the model in order to determine how the factors interact and contribute to the performance outcomes. In this type of study there is no single, specific experimental intervention to raise self-efficacy or other factors to a certain level. Instead, the existing levels of the factors are measured to determine the fit of the hypothesized model. If the measures of the factors are gathered repeatedly over a period of time, events normally occurring during that time may affect the measures, so this becomes a "natural" intervention in itself, but not a specific planned intervention. For example, in an educational setting self-efficacy and other factors may be measured at different times during the semester. The ongoing learning in the course will affect measures gathered as the course proceeds.

The method of testing multivariate relations in a model is widely used in studying self-efficacy in learning because learning outcomes are influenced by multiple interacting variables that combine to affect performance. A second reason for using this method in educational settings is that it allows the researcher to study learning in its natural setting in an ongoing course without special interventions. The only difference from the normal activities in the course is collection of data (self-efficacy, etc.) during the semester. Examples of this approach in educational settings may be found in [22, 28, 41, 44, 53, 55]. The multivariate modeling approach is used in this study. We propose a model of performance of non-major programmers based on three factors: their previous experience, perceived self-efficacy, and knowledge organization. The model is shown in Figure 1. The following paragraphs explain the rationale for the model.

We expect that previous experience will be a significant predictor of non-majors' self-efficacy, both pre-self-efficacy measured near the beginning of the course and post-self-efficacy measured near the end of the course. This prediction comes from self-efficacy theory [3, 5], which posits that experience carrying out tasks is a strong source of people's self-efficacy beliefs. Furthermore,

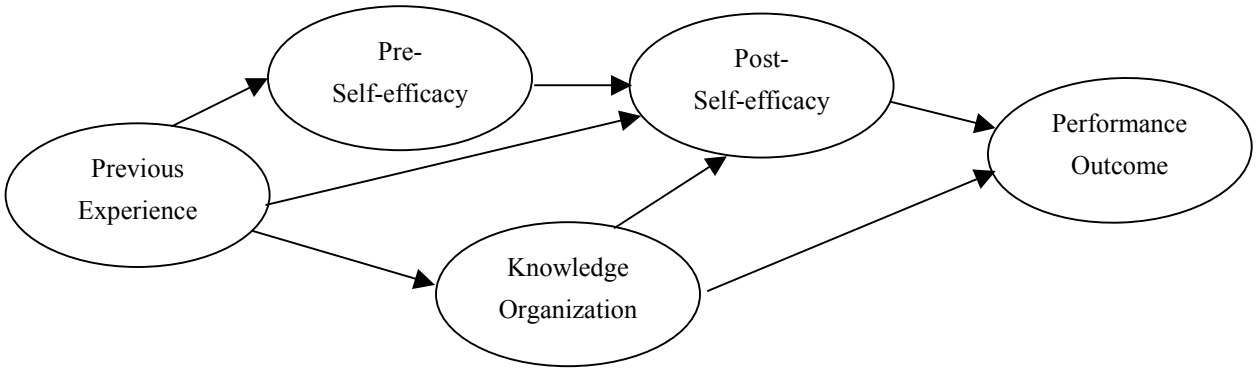

Figure 1. Research Model – ovals represent the elements of the model and arrows represent hypothesized relationships in the model.

studies of students in introductory programming have repeatedly found a relationship between prior experience and measures of performance [9, 10, 21, 34, 39, 47, 52]. Our model, however, proposes that the effect of previous experience is not direct, but is mediated by perceived self-efficacy. That is, previous experience increases a student's self-efficacy, and self-efficacy, in turn, has an effect on later variables in the model. This hypothesis of self-efficacy mediating the effects of previous experience is based on a pilot study which failed to show a direct effect and had moderate correlations of about .20 between previous experience and performance. The direct path from previous experience to post-self-efficacy is included because it is predicted that previous experience is not wholly mediated by pre-self-efficacy, i.e., previous experience continues to have a direct effect on students' post-self-efficacy, measured at a later point in the course.

The path from previous experience to knowledge organization represents our expectation that previous experience helps students in an introductory programming course to develop their knowledge organization. The psychological literature on knowledge organization [11] suggests that, beyond study and instruction, people develop their knowledge organization by extensive, ongoing experience. Thus, we expect that previous experience will lead to better knowledge organization. This relationship of previous experience to knowledge organization is also suggested in the programming domain by the differences in knowledge organization between less skilled and more skilled programmers in program recall tasks [1, 15, 25, 45].

We further expect that perceived self-efficacy measured at the beginning of a course (pre-self-efficacy) will influence self-efficacy measured later in the course (post-self-efficacy). It is expected that students' perceived efficacy will increase as a result of course instruction and continued exposure to hands-on programming (post-self-efficacy higher than pre-self-efficacy). In a programming course students have multiple experiences that may help build self-efficacy. These tend to include at least three sources of self-efficacy identified by Bandura [5]: performance of programming tasks, modeling of tasks by others (students in laboratory settings or group assignments), and encouragement by the instructor and peers.

The path from knowledge organization to post-self-efficacy represents a prediction that students' knowledge organization affects their self-efficacy measured near the end of the course. With the exception of a small study of our own [34], we do not have direct support from the literature for this path. However, we believe that well-developed knowledge organization will add to students' perceptions of their efficacy in programming. One of our objectives is to determine whether this relationship is supported by the data.

We use two distinct measures of performance outcomes: course grade and debugging performance. Final grades are often based in part on elements that are not directly related to individual programming performance, e.g., class participation. Therefore, we included a second measure of performance focused entirely on a task-oriented aspect of programming. A debugging task was chosen because of the serious problems that end users experience in creating dependable, error-free programs [7, 31]. We predict that previous experience, knowledge organization, and self-efficacy will explain a significant amount of variance in both course performance and debugging performance. Having good organization of programming knowledge is important in successful performance. Programming research has found links between knowledge organization and course examination scores [15]. In addition, empirical research on debugging [19] suggests that debugging success is influenced by knowledge organization. It is also well established in many domains (including the computer applications domain) that self-efficacy affects course and training outcomes, e.g., [13, 22, 34, 41, 53].

3. METHOD

3.1 Participants

The participants were 120 non-majors enrolled in multiple sections of an introductory university programming course using C++ over the period of five academic semesters (approximately 1.5 years). There were five different instructors who covered the same material and used the same textbook. Sixty-six percent of the participants were male and 34 percent female. The courses included both CS majors and non-majors. We analyzed only the non-majors. The non-majors came from a variety of academic majors. The most frequent majors were physical and natural sciences (39 percent) and business (28 percent). Engineering students were not included in the study because they took a

different programming course. The participants were typically third year undergraduates.

3.2 Materials

The materials included a background questionnaire, a self-efficacy scale, and a program recall measure. These materials were a subset of materials developed in a previous study [34].

A background questionnaire was used to determine previous experience. Five questions measuring the breadth of participants' prior computer and programming background were used: 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. The question on use of computer applications was included, first, because the level of computer application experience may have an effect on perceived self-efficacy for programming, and, secondly, because students may engage in some end-user programming activities while using applications. Participants were asked to take some time to think about their experience and make their best estimate. Each question was answered on a six-point scale, specific to the question.

Self-efficacy was measured using the Computer Programming Self-Efficacy Scale [35]. This validated instrument has been used previously in research on success factors in introductory computer science courses [34, 52]. The scale consists of 33 items that ask students to judge their ability to do a wide range of programming tasks. Responses are marked on a 7-point Likert scale, where 1 represents "no confidence at all" in doing the task and 7 represents "absolute confidence." The 33 items in the scale form four groups targeting different aspects of self-efficacy. One group of eleven items inquired about the participants' confidence in their ability to perform a range of programming tasks, starting from the simple and specific (e.g., understand or construct individual statements) and going to the difficult and generic (e.g., given any problem specification, write a program to solve it). The objective was to get a broad spectrum of responses on a range of tasks at various levels of difficulty. A second group of seven items focused on performing a specific task, such as writing a program under a variety of different scenarios (e.g., with someone to help the student get started, as opposed to having only a technical manual as a reference). The purpose of these questions was to capture the extent of participants' motivations and persistence in the face of difficulties. This part of the scale is modeled on Compeau and Higgins' software self-efficacy scale [13]. A third group of four questions asked the participants to rate their capabilities for self-organization, regulation, and order in the presence of distractions and competing demands, self-regulation being a key concept associated with self-efficacy [4]. Finally, a group of six questions measured the participants' confidence in performing a range of subtasks that programmers are required to perform in working with programs, such as comprehension, debugging, and modification. The Cronbach's Alpha for the scale in this study was .97, indicating that it was highly reliable.

The participants' knowledge organization was evaluated using a program recall task, based on the literature indicating that recall of a program, given a short time to memorize it, is an indicator of well-organized knowledge [45] and furthermore that program recall predicts examination grades [15]. The program recall booklet, modeled on Shneiderman's materials [45], consisted of a page of instructions followed by a single page containing the C++ program to be memorized. The program carried out temperature conversions between Fahrenheit and Celsius scales. It consisted of one class, a main function, and two additional functions. The program had 27 lines of code in total.

For each participant the final measure of course performance was the final grade assigned by the course instructor. In the grading scheme nine grade categories were used. Therefore, the final performance scores ranged from 1 to 9. The measure of debugging success was based on four questions in the final examination that required the student to find and correct errors in segments of code. Each problem was given a score of 0-3 so the cumulative score was on a scale of 0-12. The scoring was done by two independent evaluators who subsequently resolved their differences by discussion.

3.3 Procedure

The data collection was conducted over the course of a 16 week semester. The data collection was divided into two phases: the early phase in weeks 1-8 and the late phase in weeks 9-16. In the early phase data collection was carried out in weeks 1 and 8. All data were collected during class. In week 1, the data collected were the participants' self-reported previous experience and their pre-self-efficacy measured using the Computer Programming Self-Efficacy Scale [35]. Week 1 was chosen in order to obtain these data at the earliest point possible, before instruction had an effect on the participants. The other data collection in the early phase of the study took place at the beginning of week 8. At this time students carried out the program memorization/recall task to measure knowledge organization. Other educational studies using path analysis (e.g., Horn [22]) have explored the role of domain knowledge on academic success by giving a test of knowledge at the beginning of the course. However, in our case the self-reported previous experience was very low and half the students had no previous programming experience. Therefore, it seemed futile to measure knowledge organization at the start of the course. The knowledge organization task was deferred to the end of the first phase of the study when all the participants had some exposure to programming. Participants were given the program recall booklet. They had five minutes to study the program, then closed the booklet and had five more minutes to recall and exactly reproduce the program from memory, to the best of their ability.

In the late phase, self-efficacy data were collected again in week 14 using the same self-efficacy scale. (Week 14 was chosen rather than week 15 because of constraints in the course in the last week of teaching.) This yielded the post-self-efficacy score. The final examination, which included the debugging problems, was given in week 16. The participants' final course grades and debugging scores were obtained from the instructor after the end of the course.

4. RESULTS

4.1 Descriptive Statistics

The means and standard deviations of the previous experience variables are shown in Table 1. The variables were rated on a 6-point scale with higher numbers indicating greater experience. For each participant the five measures were summed to create their previous experience score. The mean previous experience of the

participants was 8.39 (StdDev 5.37) out of a maximum possible score of 30.

Table 1. Means and standard deviations of previous experience variables.

	Mean	StdDev
Number of courses using computers	2.62	1.87
Number of programming courses	1.15	1.15
Number of computer programs	1.37	1.47
Number of programming languages	1.43	1.11
Longest length of program	1.83	1.71

Table 2 shows the mean question score and standard deviation of the pre-self-efficacy and post-self-efficacy measures. Recall that questions were scored on a scale of 1-7 with 7 representing the highest confidence. The change score between the pre- and post-scores was +1.38.

Table 2. Means and standard deviations of self-efficacy measures.

	Mean	StdDev
Pre-self-efficacy	3.73	1.54
Post-self-efficacy	5.11	1.17

Program recall was evaluated by counting the number of lines that were an exact match to the program and were in the correct sequential location. An alternative scoring method is to score functionally equivalent code as correct because programmers may remember the gist of the code and reconstruct it at recall time [15]. However, judging whether code is functionally equivalent can lead to errors and inconsistencies. We preferred to use the strict procedure. There were 27 lines in the program and consequently the maximum possible score was 27. The mean of the recall task was 13.06 (StdDev 4.82).

The mean course performance measure, based on the final grade with a range from 1-9, was 6.03 (StdDev 1.99). The mean debugging score, ranging from 0-12, was 8.92 (StdDev 1.69).

4.2 Correlations

The correlations of the measures are shown in Table 3. The table includes the two alternative outcome variables, course grade and debugging score. Most of the correlations are significant. However, correlations cannot establish causality, nor can they evaluate the effect of multiple variables on a dependent variable.

4.3 Path Analysis

In the model (Figure 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 10-20 subjects per path in the model [23]. Our N of 120 and 7 paths falls within this guideline. Figure 2 shows the results of the analysis using final grade as the performance variable. The strength of each relationship is depicted by the path coefficients, or standardized regression weight, which varies between 0 and 1 (shown on the arrows in Figure 2). A significant path coefficient indicates that there is a reliable relationship between the given predictor and response variable. All paths predicted in the model were significant ($p<.01$) except for the path from previous experience to knowledge organization, which was not significant. The R^2 value associated with each dependent variable indicates how much of the variance in that variable is explained by all the predictor variables contributing to it, i.e., the paths coming into the variable. The variances explained are moderate or high, according to Cohen [12], except for knowledge organization with only 2 percent explained.

Table 3. Correlations of variables (*p<.01; other correlations are non-significant).

	Previous experience	Pre-self-efficacy	Post-self-efficacy	Knowledge organization	Course grade	Debugging grade
Previous experience	1.00					
Pre-self-efficacy	.49*	1.00				
Post-self-efficacy	.55*	.51*	1.00			
Knowledge organization	.13	.08	.28*	1.00		
Course grade	.27*	.02	.37*	.46*	1.00	
Debugging grade	.24*	.02	.39*	.42*	.81*	1.00

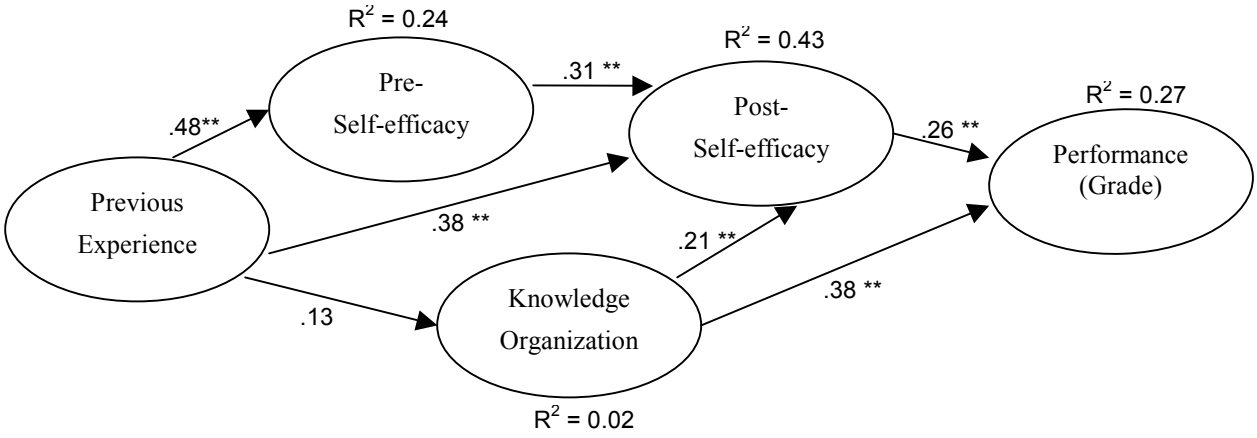

Figure 2. Results of the path analysis with the course grade as the performance outcome variable (p < .01; other paths are not significant). The numbers on the arrows are the path coefficients and the R^2 statistics are the proportion of the variance accounted for.**

The path analysis in Figure 2 shows significant causal paths among pairs of variables in the model. However, it does not give an overall evaluation of the goodness-of-fit of the data to the model. This is done using a Chi-square test and several indices of fit that have been developed for this purpose [40]. Following the usual procedure, the non-significant path from previous experience to knowledge organization was eliminated for this analysis.

A Chi-square test is normally done first. Contrary to the usual interpretation of a p-value, in this analysis a non-significant result represents an adequate fit. In model testing a non-significant p-value indicates that the model, which contains a subset of all possible paths, is as good a predictor of performance as a model containing every possible path. A goal in modeling is to find parsimonious models. In fact, some of the indices of fit reported below penalize models with many paths.

The Chi-square test of this model with 3 degrees of freedom yielded χ^2=7.90, p<.05, a significant result. In addition, a set of indices frequently used to assess model fit is shown in Table 4. The mathematical explanation of the indices is beyond the scope of this paper, but can be found in Schumacker and Lomax [40]. A significant χ^2 can often be explained by a large N. However, three of the indices also fail to conform to the adequate fit values. Furthermore, the RSMEA is quite divergent from its adequate fit value. Together these results lead to the conclusion that the overall fit of the model is not good.

Table 4. Indices of fit for initial path model in Figure 2.

Index name	Value	Adequate fit value [40]
GFI (Goodness-of-Fit Index)	.98	>.90
AGFI (Adjusted Goodness-of-Fit Index)	.87	>.90
NFI (Normed Fit Index)	.95	>.90
TLI (Tucker-Lewis Index)	.89	>.90
CFI (Comparative Fit Index	.97	>.90
RSMEA (Root-Mean-Square Error of Approximation)	.18	<.08 or <.05

In response to the poor fit, we reviewed the data again. One thing that stood out was the very low correlation of pre-self-efficacy to course performance. Self-efficacy is usually a strong predictor of outcomes, so this suggested that the poor fit might be related to the relationship between pre-self-efficacy and performance. Therefore, we tested an alternate model which added a direct path from pre-self-efficacy to performance. The results are shown in Figure 3. They indicate that there is a significant path between

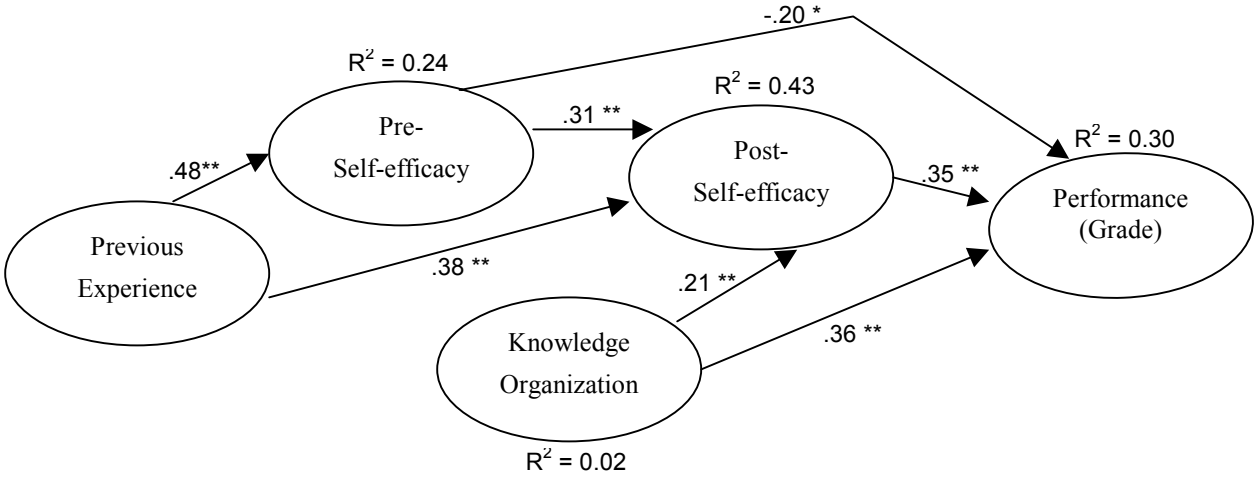

Figure 3. Results of the revised path analysis with the course grade as the performance outcome variable (*p<.05, **p < .01). This model adds a direct path from pre-self-efficacy to performance and eliminates the non-significant path from previous experience to knowledge organization.

pre-self-efficacy and performance, but the path coefficient, surprisingly, is negative, i.e. higher pre-self-efficacy is associated with lower performance. We discuss possible reasons for the negative path from pre-self-efficacy to performance in the next section.

Going on to the fit of this model, the Chi-square test with 2 degrees of freedom resulted in χ^2=2.91, p>.23. The non-significant statistic is an indicator of good fit, as discussed above. Table 5 shows the indices of fit for this model. In this revised model all of the indices of fit indicate a good model fit. Note that the RSMEA in Table 5 gives two adequate fit values because experts differ on this value.

Table 5. Indices of fit for revised model shown in Figure 3.

Index name	Value	Adequate fit value [40]
GFI (Goodness-of-Fit Index)	.99	>.90
AGFI (Adjusted Goodness-of-Fit Index)	.93	>.90
NFI (Normed Fit Index)	.98	>.90
TLI (Tucker-Lewis Index)	.97	>.90
CFI (Comparative Fit Index	.99	>.90
RSMEA (Root-Mean-Square Error of Approximation)	.06	<.08 or <.05

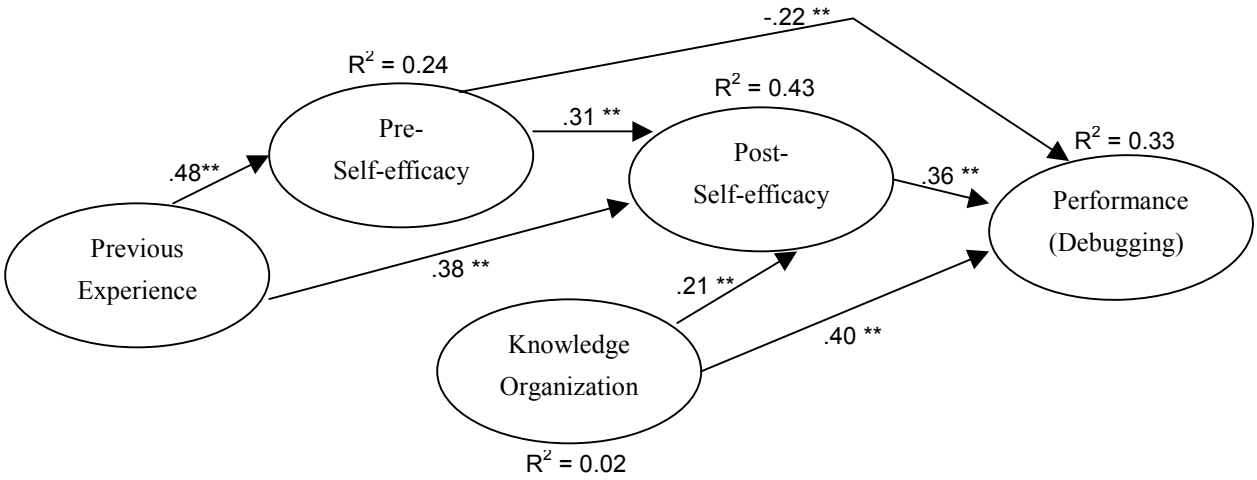

Figure 4. Results of the revised path analysis with the debugging score as the performance outcome variable (p<.01).**

The results of the model using the debugging score as the performance outcome is shown in Figure 4. The results were very similar to the model using course grade. The amount of variance explained in the performance variable was slightly higher than in the model with course grade. The Chi-square was non-significant (χ^2=3.02, p>.25) and the indices of fit were all within the accepted bounds.

5. DISCUSSION

The change between pre- and post-self-efficacy indicates that non-majors' perceived self-efficacy increased substantially over the course of a semester. This is consistent with self-efficacy theory. Learners evaluate their capabilities to perform based on direct experience with tasks, as well as other influences that may take place during learning, such as behavior modeling, persuasion by others, and self-monitoring. A semester of concentrated programming instruction coupled with frequent hands-on programming is likely to increase self-efficacy, unless the tasks are much too difficult for the learners. Students with high experience are likely to have a strong sense of self-efficacy that is resistant to change. However, students starting with low previous experience, as in this study, normally have initial beliefs that are weaker and more subject to change. The increase from pre- to post-self-efficacy reflects the malleability of their weakly held beliefs [5].

Our results of the path analysis (Figure 2) show that previous experience is a strong predictor of pre-self-efficacy, as expected from previous research. The relationship is positive: students with higher previous experience have higher pre-self-efficacy. Furthermore, previous experience also predicts post-self-efficacy. This indicates that the students' previous 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 beyond the introductory course, it seems likely that soon this previous experience will lose its predictive value. Instead students will increasingly base their self-beliefs on their more recent programming experiences. However, at this early stage in programming, beliefs that arise from their prior experience still have weight in their judgments. This implies that, for non-majors who take only a single programming course, the influence of their low previous experience may affect their decisions about later engaging in end-user programming.

Contrary to the original prediction, the non-majors' previous experience did not affect their knowledge organization in programming. The initial expectation was that the previous programming experience of non-majors would predict knowledge organization measured near the middle of the course. This was not the case: there was no significant relationship and an extremely low amount of variance accounted for (2 percent). An explanation of this result might be that the non-majors' very low previous experience did little to build knowledge organization. They gained their knowledge organization during the course itself. The previous experience of the students, whether relatively lower or higher, was not correlated with their knowledge organization after several weeks in the introductory programming course. Based on the current findings, it appears that the contribution of non-majors' previous experience is strongly reflected in their self-efficacy beliefs but little reflected in their knowledge organization. In a student population with higher previous experience, for example a group consisting of CS majors, the results might be different.

The results also show that having developed good knowledge organization increases beliefs of self-efficacy, as seen in the post-self-efficacy measure. Students who in the early part of the course were able to develop strong knowledge organization had higher beliefs about their capability to carry out a range of programming tasks under a variety of conditions. This fits with self-efficacy theory.

In our initial model (Figure 1) we predicted that only post-self-efficacy and knowledge organization would directly affect performance and that the effects of other variables in the model would be mediated by those two variables. There were indeed strong relationships of both post-self-efficacy and knowledge organization to performance (Figure 2). Together they explained 27 percent of the variance in the course grade, a moderate amount according to Cohen [12]. However, the goodness-of-fit of the model was not adequate, indicating that other relationships in the model had not been accounted for.

Revisiting the correlations, we found that post-self-efficacy had a high correlation with performance, pre-self-efficacy had a very low correlation performance, and pre-self-efficacy had a high correlation with post-self-efficacy. The assumption that pre-self-efficacy was fully mediated by post-self-efficacy appeared to be faulty. Consequently, the model was rerun adding a direct path from pre-self-efficacy to performance. The results (Figure 3) showed that there was indeed a significant relationship of pre-self-efficacy to performance; the assumption of full mediation was wrong. Overall, the model fit improved with the additional path, and the percentage of variance accounted for in performance increased. However, surprisingly, the relationship between pre-self-efficacy and performance was negative. Higher pre-self-efficacy was associated lower performance.

The negative path from pre-self-efficacy to performance seems counter-intuitive, since the non-majors' mean perceived self-efficacy increased from the pre- to post-measure. A possible explanation is that some students were actually *overconfident* about their capabilities at the beginning of the course. According to Bandura [5], individuals with low previous experience are subject to misjudging their capabilities because of lack of indicators. They may base self-efficacy judgments on tasks that are not particularly relevant to programming (e.g., game playing) or on generalized self-beliefs ("I can excel in most academic situations"). Later their active programming experiences and feedback on performance may have allowed them to more accurately perceive their capabilities. About 20 percent of the participants had lower perceived post-self-efficacy scores than pre-self-efficacy scores, which makes this explanation plausible. In addition, overconfidence has been reported in end-user debugging of spreadsheets [51]. We hope to carry out another study to see if this effect can be replicated.

The model using debugging scores as the performance measure (Figure 4) held no surprises. The results were highly congruent with the model based on course grade. Our initial thought had been that relationships between predictor variables and overall course grade might be weak because the grade encompasses many elements, some of which may not be directly focused on programming performance. The debugging performance measure

was meant to determine whether this was the case. As it turned out, using the same predictor variables with debugging success as the performance variable, a somewhat larger amount of variance was accounted for in performance. However, the differences were slight. The similarity of the results can be accounted for by the very high correlation of the course grade and the debugging scores. This result, while not yielding a great deal of new information, does allay concerns about using course grade as the performance variable.

There are several limitations of this study that should be addressed in future research. First, the study took place in the short time span of a 16 week semester. Non-majors do not normally take a follow-on programming course, but it would be very informative to track them as they take other courses involving end-user programming, such as courses involving problem solving with spreadsheets. Logistically, this would be difficult and may not be feasible, but it would be useful to see how non-majors fare in end-user programming tasks, after taking a formal programming course. Second, the self-reporting of previous experience is a limitation, since students may have difficulty remembering details about their previous experience accurately. However, we think that in this study the risk of memory inaccuracies in participants' self-reports was not high. The non-majors' previous experience was scant, so most students probably did not have to think hard to answer the questions. Third, our results do not agree with Wilson and Shrock's study [52] that failed to find a significant relationship between self-efficacy and course outcome. The methodologies of the two studies were different in numerous ways, and this is a possible explanation. However, we would like to understand why the results were different. Finally, and most importantly, achieving adequate model fit does not assure that the model is true [24]. Model fit can be influenced by several things, for example, a model with too many paths or an incorrect model that is mathematically equivalent to the correct model [24]. For that reason, replication studies are needed as well as investigation of alternative models.

We believe that an important, and largely unstudied, area of research is the study of non-majors who drop out of the introductory programming course. If students drop out before completing the course, we fail to capture data that might help us to understand the challenges they face.

6. CONCLUSIONS

This research evaluated a model of factors affecting non-majors' success in their introductory programming course. This theory-based model provides a basis for future interventions to help students succeed. Based on this research, there appear to be two general paths of intervention, one focused on non-majors' knowledge organization in programming and the other on their self-efficacy.

This study of non-majors confirms the importance of their knowledge organization in course outcomes. Their knowledge organization affected success directly and also strengthened post-self-efficacy. This implies that teaching should pay close attention to development of the students' organization of knowledge. Students may learn a great deal of poorly organized programming knowledge that does not well support actual tasks of programming [1, 2]. Soloway and Ehrlich [46] have shown empirically that skilled programmers possess plans that consist of organized, meaningful, and goal-oriented programming knowledge. They argue that students need gradually to develop a large stock of typical, recurring plans to apply in writing or comprehending programs. Today many textbooks for introductory programming courses reinforce knowledge of typical ways to achieve recurring goals. This approach supports the development of organized, goal-oriented programming knowledge and should be encouraged.

Our study also demonstrates the importance of non-majors' perceived self-efficacy in introductory programming. Research in several fields reports that self-efficacy interventions can be carried out successfully in the context of education and training. That suggests the likelihood that self-efficacy interventions, adapted for programming instruction, could be successful. According to Bandura [3, 5], interventions to improve self-efficacy include performance success, observation of the performance of peers, social persuasion, and monitoring of one's physiological state.

Performance success consists of steadily carrying out tasks of increasing difficulty, until the student has a history of solid attainments. Frequent, small hands-on programming activities seem likely to build the history of success more than infrequent, large assignments. For students to assess their capability while carrying out hands-on programming tasks, they need feedback that is specific, frequent, and timely. Observing the performance of peers has also been used as a self-efficacy strengthening intervention [5, 13]. Peer modeling is more successful than modeling by an expert because the student recognizes that the model does not know all the answers. This makes the successes of the model more compelling to the student. Peer modeling may be difficult to arrange in a course setting. The modeling could be done live in a classroom with a peer working through a problem while other students watch, but that can be embarrassing and stressful to the model. It may be more appropriate for students to view a video of a peer successfully planning and executing a programming task, including showing the model struggling with and finally overcoming difficulties. Social persuasion can contribute to improving self-efficacy. The persuasion may come from an instructor but, like peer modeling, it appears to be more effective if it comes from other students. In many CS classrooms students work together on problems, and this is a good setting for social persuasion, especially if the students have different levels of self-efficacy. That puts the group in a position where social persuasion can take place, provided that there is on-going and lively interaction in the group. In order to achieve different levels of self-efficacy in a work group, it is better for an instructor to form the groups using information about the students' capabilities than to allow students to choose their own groups. Finally, students monitor their own physiological states, particularly anxiety and stress, to infer their capableness. Low anxiety makes students feel more capable. If the anxiety comes from stress and competition in the classroom, then the intervention should analyze classroom behaviors to attempt to make the classroom more comfortable to students.

7. ACKNOWLEDGMENTS

This work was supported in part by the EUSES Consortium via NSF grant CCR-0324844. Grateful acknowledgment is made to Vennila Ramalingam for her inspiration and support, as well as for the use of materials she developed. Also thanks to Thomas Green and anonymous reviewers of an earlier PPIG submission.

8. REFERENCES

[1] Adelson, B. Problem solving and the development of abstract categories in progrmming languages. *Memory and Cognition*, 9, 4 (1981), 422-433.

[2] Adelson, B. When novices surpass experts: The difficulty of a task may increase with expertise. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 10, 3 (1984), 483-495.

[3] Bandura, A. Self-efficacy. In V.S. Ramachaudran (Ed.), *Encyclopedia of Human Behavior*, Vol. 4, Academic Press, NY, 1994, 71-81.

[4] Bandura, A. *Self-efficacy: The Exercise of Control*. W.H Freeman, NY, 1997.

[5] Bandura, A. *Social Foundations of Thought and Action*. Prentice Hall, Englewood Cliffs, NJ, 1986.

[6] Blackwell, A.F. End user developers at home. *Communications of the ACM,* 4, 9 (2004), 65-66.

[7] Boehm, B. and Basili, V. Software defect reduction top 10 list. *Computer*, 34, 1 (2000), 135-137.

[8] Boehm , B.W., Horowitz, E., Madachy, R., Reifer , D., Clark, B.K., Steece, B., Brown, A.W., Chulani, S., Abts, C. *Software Cost Estimation with COCOMO II*. Prentice Hall PRT, Upper Saddle River, NJ, 2000.

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

[10] Byrne, P. and Lyons, G. The effect of student attributes on success in programming. *Proceedings of the 6th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, (Canterbury, U.K.,2001). ACM Press, NY, 2001, 49-52.

[11] Chase, W.G. and Simon, H.A. The mind's eye in chess. In W.G. Chase (Ed.), *Visual Information Processing*. Academic Press, NY, 1973, 215-281.

[12] Cohen, J. *Statistical Power Analysis for the Social Sciences*. Academic Press, NY, 1977.

[13] Compeau, D.R. and Higgins, C.A. Application of social cognitive theory to training for computer skills. *Information Systems Research*, 6, 2 (1995), 118-143.

[14] De Groot, A.D. *Thought and Choice in Chess*. Mouton, The Hague, 1965.

[15] Di Persio, T., Isbister, D. and Shneiderman, B. An experiment using memorization/reconstruction as a measure of programmer ability. *International Journal of Human-Computer Studies*, 13 (1980), 339-353.

[16] Egan, D.E. and Schwartz, B.J. Chunking in recall of symbolic drawings. *Memory and Cognition*, 7, 2 (1979), 149-158.

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

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

[19] Gugerty, L. and Olson, G.M. Comprehension differences in debugging by skilled and novice programmers. In E. Soloway and S. Iyengar (Eds.), *Empirical Studies of Programmers*. Ablex, Norwood, NJ, 1986, 80-98.

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

[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 Conference on Innovation and Technology in Computer Science Education*, (Helsinki, Finland, July 11-13, 2000), ACM, NY, 2000, 25-28.

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

[23] Kerlinger, F.N. and Pehazur, E.J. *Multiple Regression in Behavioral Research*. Holt, Reinhart and Winston, NY, 1973.

[24] Kline, R.B. *Principles and Practice of Structural Equation Modeling*, 2nd Ed. Guilford Press, NY, 2004.

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

[26] McKeithen, K.B., Reitman, J.S., Rueter, H.H.,and Hirtle, S.C. Knowledge organization and skill differences in computer programmers. *Cognitive Psychology*, 13 (1981), 307-325.

[27] McKinney, S. and Denton, L.F. Houston, we have a problem : there's a leak in the CS1 affective oxygen tank. *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, (Norfolk, VA, March 3-7, 2004), ACM Press, NY, 2004, 236-239.

[28] Mitchell, T.R., Hopper, H., Daniels, D., George-Falvy, J. and James, L.R. Predicting self-efficacy and performance during skill acquisition. *Journal of Applied Psychology*, 79, 4 (1994), 506-517.

[29] Nardi, B. *A Small Matter of Programming.* MIT Press, Boston, 1993.

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

[31] Panko, R. What we know about spreadsheet errors. *Journal of End User Computing*, (Spring 1998), 15-21.

[32] Peyton-Jones, S., Blackwell, A. Burnett, M. A user-centered approach to functions in Excel. *Proceedings of the 8th ACM SIGPLAN International Conference on Functional Programming*, (Uppsula, Sweden, August 25-29, 2003), ACM, NY, 2003, 165-176.

[33] Potosky, D. 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 (2002), 241-255.

[34] Ramalingam, V., LaBelle, D., and Wiedenbeck, S. Self-efficacy and mental models in learning to program. *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, (Leeds, U.K., June 28-30, 2004), ACM, NY, 2004, 171-175.

[35] Ramalingam V. and Wiedenbeck S. 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 (1998), 365-379.

[36] Rosson, M.B., Ballin, J., and Nash, H. Everyday programming: challenges and opportunities for informal web development. *Proceedings of the IEEE Symposia on Visual Languages and Human-Centric Computing*, (Rome, Italy, September 26-29, 2004), IEEE Press, 2004, 123-130.

[37] Rothermel, G., Burnett, M., Li, L., Dupuis, C., and Sheretov, A. A methodology for testing spreadsheets. *ACM Transactions on Software Engineering and Methodology*, 10, 1 (2001), 110-147.

[38] Rountree, N., Rountree, J., and Robins, A. Predictors of success and failure in a CS1 course. *SIGCSE Bulletin*, 34, 4 (2002), 121-124.

[39] 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*, (Philadelphia, PA, February 15-17, 1996), ACM, NY, 1996, 37-41.

[40] Schumacker, R.E. and Lomax, R.G. *A Beginner's Guide to Structural Equation Modeling*. Lawrence Erlbaum, Hillsdale, NJ, 1996.

[41] Schunk, D.H. Modeling and attributional effects on children's achievement: a self-efficacy analysis. *Journal of Educational Psychology*, 73, 1 (1981), 93-105.

[42] Schunk, D.H. and Hanson, A.R. Peer models: influence on children's self-efficacy and achievement behaviors. *Journal of Educational Psychology*, 77 (1985), 313-322.

[43] Schunk, D.H., Hanson, A.R. and Cox, P.D. Peer model attributes and children's achievement behaviors. *Journal of Educational Psychology*, 79 (1987), 54-61.

[44] Shell, D.F., Murphy, C.C. and Bruning, R. Self-efficacy, and outcome expectancy mechanisms in reading and writing achievement. *Journal of Educational Psychology*, 81, 1 (1989), 91-100.

[45] Shneiderman, B. Exploratory experiments in programmer behavior. *International Journal of Computer and Information Sciences*, 5, 2 (1976), 123-142.

[46] Soloway, E. and Ehrlich, K. Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*, SE-10, 5 (1984), 595-609.

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

[48] Thomas, L., Woodbury, J. and Jarman, E. Learning styles and performance in the introductory programming sequence. *Proceedings of the 33th SIGCSE Technical Symposium on Computer Science Education*, (Covington, KY, February 27-March 3, 2002), ACM Press, NY, 2002, 33-37.

[49] Torkzadeh, G. and Koufteros, X. Factorial validity of a computer self-efficacy scale and the impact of computer training. *Educational and Psychological Measurement*, 54, 3 (1994), 813-821.

[50] Wiedenbeck, S. and Engebretson, A. Comprehension strategies of end-user programmers in an event driven application. *Proceedings of the IEEE Symposia on Visual Languages and Human-Centric Computing*, (Rome, Italy, September 26-29, 2004), IEEE Press, 2004, 207-214.

[51] Wilcox, E., Atwood, J., Burnett, M., Cadiz, J., and Cook, C. Does continuous visual feedback aid debugging in direct-manipulation programming systems? *CHI'97* (1997). ACM Press, NY, 22-27.

[52] Wilson, B.C. and Shrock S. Contributing to success in an introductory computer science course: s study of twelve factors. *Proceedings of the 32th SIGCSE Technical Symposium on Computer Science Education*, (Charlotte, NC, 2002), ACM Press, NY, 2002, 184-188.

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

[54] Zimmerman, B.J. and Bandura, A. Impact of self-regulatory influences on writing course attainment. *American Educational Research Journal*, 31, 4 (1994), 845-862.

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