ISSN: 2473-4901
v9 n5997

Predicting Perceived Programming Self-Efficacy for Information System Students

Ramadan Abdunabi
Ramadan.Abdunabi@colostate.edu
Computer Information Systems

Ilham Hbaci
ilham.hbaci@colostate.edu
CSU Stem Center

Teddy Nyambe
Teddy.Nyambe@colostate.edu
Computer Information Systems

Colorado State University,
Fort Collins, CO 80523,

Abstract

Programming is a major subject in various Information Systems (IS) programs, with students often finding it a challenging skill to acquire. While there is extensive literature on factors helping students learn to program, most of which focuses on non-IS students. Due to the increasing demand for professionals with programming skills, there is a pressing need for further research on factors that could enhance learning programming skills at the higher education level. One promising approach to address this issue involves examining students' internal characteristics, their programming self-efficacy, and its connection to instructional methods that can enhance it. This study adopted a quantitative research design to evaluate students' programming self-efficacy. A survey was conducted to measure students' beliefs in their programming competence and engagement in various instructional activities, including the value they attributed to learning programming, the time spent practicing, and the frequency they sought guidance from teaching assistants (TA). Through a hierarchical multiple regression analysis, this work investigated how these mentioned variables could predict student-programming self-efficacy. The results indicated that the value students placed on learning programming emerged as the most significant predictor for programming self-efficacy. On the other hand, there was no substantial relationship between programming self-efficacy and the practice time or consulting TA. These findings suggest that educators and instructional designers need to emphasize the practicality and importance of learning how to program to enhance students' perceived value.

Keywords: Information Systems, Predicting Programming Self-Efficacy, Pedagogy, Fragmented Learning, Mobile Learning.

1. INTRODUCTION

Information Systems (IS) students envision themselves as programmers in the future, and they believe programming is a significant and valuable skill they need to acquire (authors). Additionally, many of the IS-related jobs, such as business analyst, data analyst, data mining, project manager, and others, require programming skills. As technology advances and companies and government agencies seek efficiency and cost savings, demand for information system specialists should continue to grow at a projected rate of 23% from 2021 to 2031, which would be higher than the projected growth for all other occupations (U.S. Bureau of Labor Statistics, 2023).

Although programming is a required outcome of IS graduates and core competency for employment in many IT industries (Bashir & Hoque, 2016), difficulty with computer programming has been shown to contribute to well-documented dropout rates in introductory programming courses in the United States (Kori, Pedaste, Leijen, & Tõnisson, 2016; Zhang, Zhang, Stafford, & Zhang, 2014). Programming has been considered a difficult task because it involves skills that go well beyond how to write error-free programs (Loksa, Jernigan, Oleson, Mendez, & Burnett, 2016; Forte, & Guzdial, 2005).

Learning to program requires effective instruction on syntax, data structures, and abstraction but additionally requires investigation and evaluation of physiological traits of the individuals, such as self-efficacy (Bandura, 1977; Gupta and Bostrom, 2019; Metcalfe, & Shimamura, 1994). Self-efficacy is an individual's judgment of their capabilities to organize and execute courses of action required to attain designated types of performance (Metcalfe, & Shimamura, 1994). Further, it has been observed by Schunk & Pajares (2005) that how people behave can often be better predicted by the beliefs they hold about their capabilities than by what they are capable of accomplishing, for these self-efficacy perceptions help determine what individuals do with their knowledge and skills. Based on Bandura's (1977) theory, individuals who have a strong sense of self-efficacy in a specific situation would devote their attention and effort to the demands of this situation and, when faced with difficulties, these individuals would try harder and persist longer than individuals who have had low perceived self-efficacy. Programming self-efficacy is defined as individuals' evaluation of their ability to solve computational problems by employing their programming skills and experiences (Kong, 2017; Marakas et al., 2022). Students with high computer programming self-efficacy tend to utilize their skills to solve computational problems and persist in solving challenging ones (Latifah & Nugraha, 2023).

Since the nineties of the last century, researchers have started to examine the possible instructional factors within educational contexts affecting students' self-efficacy within the higher educational level (Van Dinther, Dochy, & Segers, ,2011). Based on the literature reviewed (Sheokand, 2022; Gumelar et al., 2022; Van Dinther, Dochy & Segers, 2011) concluded that it is possible to influence student self-efficacy by instructional factors, but these factors are more effective if they are based on social learning theory by Bandura (1977). While widespread research has been conducted to investigate factors that are related to students' programming self-efficacy and how it is influenced by various pedagogies and demographic variables (Askar & Davenport, 2009; Bashir & Hoque, 2016; Cigdem & Yildirim, 2014; Konecki, 2014; Korkmaz & Altun, 2014; Nurhikmah et al., 2021; Özmen & Altun, 2014; Rogerson & Scott, 2010; Tsai, 2019; Tsai, Wang, & Hsu, 2019; Wiggins, Grafsgaard, Boyer, Weigold & Weigold, 2021; Wiebe, & Lester, 2017), most of these studies are focused on the population of computer science and engineering students (non-IS students), and their varying outcomes make it difficult to draw any conclusions regarding reliable predictors for students' programming self-efficacy, particularly among IS students.

An instructor of two introductory programming classes at the CIS department, College of Business, Colorado State University, has explored various instructional approaches in teaching programming. This research elected three instructional approaches based on Bandura's (1977) theory, as Van Dinther, Dochy, and Segers, (2011) recommended. This study aimed to explore how three instructional approaches—(a) perceived value of programming, (b) weekly time spent on programming practice, and (c) frequency of TA consultations—contributed to the programming self-efficacy scores of IS students in two introductory programming courses. The research question related to the body of this study is defined as follows: Do undergraduate IS students' perceived value of learning programming, the number of hours they spend per week practicing weekly assigned programs, and the number of times they consult TA predict their levels of programming self-efficacy? The findings should help educators and instructional designers to develop, update, or improve instructional approaches for their classes that, in turn, facilitate high levels of programming self-efficacy among students.

2. RELATED WORK

The key point of computing self-efficacy attributed to individuals with high self-efficacy would competently write programs and utilize different software systems. Nevertheless, those with low computer self-efficacy would perceive their capabilities as limited to software or computer systems (Gupta & Bostrom, 2019; Malaquias et al., 2021). The principle of self-efficacy is further emphasized by Zimmerman (2000) that self-officious students participate more readily, work harder, persist longer, and have fewer adverse emotional reactions when they encounter difficulties than do those who so doubt their capabilities.

Although Bandura's theory has been widely used in the literature and has demonstrated validity, there was a growing recognition that additional explanatory variables that rely on this theory were needed (Gupta & Bostrom 2019; Metcalfe, & Shimamura, 1994), particularly for a unique population like IS students. This work combined the variables: value of learning programming (students' perceptions of the interest, usefulness, and importance of a task), practice programming, and consulting TAs, and investigate how these three key variables could predict student perceptions of programming self-efficacy.

Value of Learning Programming
It has been argued by Simpkins, Davis-Kean, and Eccles (2006) that individuals' values of learning a particular aspect influence educational and career choices and course success in many fields. Beyer (2014) also found that if students believed that careers in computer science (CS), for example, did not reflect their interpersonal values, they did not desire to pursue a CS major, even if it could lead to lucrative careers. In a review research of 64 articles highlighting the potential utility of self-efficacy to maximize student learning outcomes, Bartimote-Aufflick, Bridgeman, Walker, Sharma, and Smith (2016) found that self-efficacy is repeatedly highly correlated with the value of learning programming. Further, Wigfield et al. (2000) explain that individuals' expectancies for success and achievement values predict their overall achievement outcomes, including their performance, persistence, and choices of activities. In this regard, students who recognize the significance of technology and coding in computer-based products, which are now more accessible in society than ever before, typically possess strong programming self-efficacy. Powell et al. (2015) conducted two studies to explore the impact of screencast creation and group participation on student learning outcomes in programming. The findings suggest that both screencast creation and group participation have a positive influence on learning success in programming. However, both studies show a limited effect of Self-Efficacy development while creating screencasts and group participation.

Programming Practice Time
In addition to the importance of the variable value for learning to program, it was necessary to practice how to code on a regular basis to acquire, improve, or even maintain it. The amount of practice required would depend on the nature of the activity and on each individual. For example, a key determinant of success for novice programmers would be the extent to which they practiced writing code (Denny, Cukierman, & Bhaskar, 2015). Practically, the time spent practicing programming during the semester was found to be a significant predictor of students' academic performance, with students who spent more time coding having better performance (Niitsoo, Paales, Pedaste, Siiman, & Tõnisson, 2014). Referring to the conducted literature review, research effort regarding how programming self-efficacy is related to the amount of time practicing programming out of the class time is limited, specifically, for IS majors.

Teaching Assistant
Graduate student TAs have played a vital role in undergraduate teaching in higher education through their work as graders, tutorial leaders (tutoring), and lab demonstrators. The teaching abilities and preparedness of TAs could directly influence undergraduate instruction in different fields. For instance, in numerous fields, undergraduates reported greater gains in content knowledge when TAs were perceived as supportive of their learning (Wheeler, Maeng, Chiu, & Bell, 2017). Regarding the relation between tutoring and programming self-efficacy, Wiggins, Grafsgaard, Boyer, Wiebe, and Lester (2017) found that tutoring tends to be associated with student programming self-efficacy among CS students. Students who acknowledged the tutor's feedback and had dialogs with tutors were found as highly self-efficacious students. Students who engaged in fewer interactions with tutors, compared to those with high programming self-efficacy, exhibited lower levels of programming self-efficacy (Wiggins, Grafsgaard, Boyer, Wiebe, & Lester, 2017).To the best of our knowledge, the research conducted regarding the role of TAs and its relation to programming self-efficacy, specifically in IS schools, is still limited; hence, this study shed light on the question of "Do TAs Matter?" by exploring if students' communications with TAs could predict their programming self-efficacy.

3. CONCEPTUAL FRAMEWORK

This study was based on Bandura's self-efficacy theory (1977). Concerning the factors that build, assess, and interpret individuals' self-efficacy,

ISSN: 2473-4901
v9 n5997

Bandura (1977) highlighted that users' value of learning a skill (such as a coding skill) was a critical factor and it formed an individual skill self-efficacy. Therefore, when considering the importance of learning programming, it is essential to focus on enhancing users' programming self-efficacy. Moreover, Gist and Mitchell (1992) presented processes that assess self-efficacy such as the analysis of task requirements (an individual's determination of what it takes to perform a task) and attributional analysis of experience (an individual's judgment about why a performance level occurred). Practically, applying Gist and Mitchell (1992) processes in an educational setting demonstrated that the analysis of task requirements included the time dedicated to the course work, which indicated that the time a student spent learning programming skills could be considered one of the factors related to a student's programming self-efficacy. Additionally, the attributional analysis of experience provided students' personal perception and understanding regarding why they reached a specific performance level, as Redmond (2016) stated. One of those attributes that influenced students' learning at a specific level was the availability of communication between students and authorized assistants such as professors/TAs (Redmond,2016). This clearly indicated that students' call for support from their TAs might contribute to their programming self-efficacy. Figure 1 summarizes the conceptual framework that addresses the research questions.

4. METHODOLOGY

This section describes the research methods, reliability, measurements, and results that were conducted and obtained. This study used a reliable measurement of programming self-efficacy for students where individuals operationally have been asked whether they could perform specific levels of programming tasks through the degree of that endorsement (such as from total uncertainty to total certainty).

Participants
The research was conducted at a large state university in the United States after obtaining the approval of the Institutional Review Board. A nonprobability convenience sampling method allowed data collection within time and place restrictions. A total of 140 students completed the survey and the norms for participants' selection were: the target population was undergraduate CIS students, the accessible population was undergraduate students from the Colorado State University, College of Business, Computer Information Systems. Participants were 18 years or older, and each participant was taking either a junior-level programming course (CIS240) or a senior-level programming course (CIS340). Table 1 contains the participants' demographics.

Participant and Class Info	N (%)
Gender	
Male	99 (70.7%)
Female	41 (29.3%)
Class	
Application Design and Development course	74 (52.9%)
Advanced Application Design and Development course	66 (47.1%)

Note. Ages ranged from 19 to 61 years old. N=140.
Table 1: Participants' Characteristics

The two-course activities included in-class exercises where the TAs helped students, viewed interviews with influential people in the sector of programming, lectures by guest speakers who discussed their programming experience, and advised students on how to find appropriate jobs. There was one TA for each class, and these were master's students in Computer Information Systems. Each one had at least two (2) years of programming experience with Java. Their work experience as TAs ranged from 2 to 4 semesters. Their responsibilities included working on multi-step programming problems during in-class

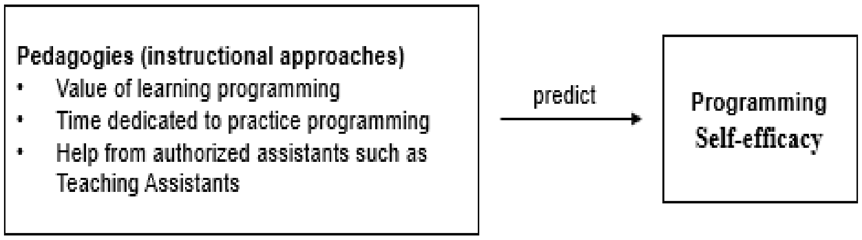
Pedagogies (instructional approaches)
• Value of learning programming
• Time dedicated to practice programming
• Help from authorized assistants such as Teaching Assistants
predict
Programming Self-efficacy

Figure 1: Instructional approaches to predict programming self-efficacy.

exercises, tutoring students during lab hours, responding to students' questions via emails, and grading that included descriptive feedback. The assignments are due every two weeks, and the first week is designated for finishing practice examples related to the covered topic. These practice examples are not graded, but students are advised to practice them before completing the assignments.

Research Process and Measurements

This study used a survey research design. Data was collected during two spring semesters in two consecutive academic calendar years. The survey link was created with Qualtrics and distributed to the students of the two programming classes at the end of the Spring semesters. Participants completed a set of questions in three sections: student programming self-efficacy, the perceived value of learning programming, and demographic characteristics.

To measure students' programming self-efficacy, this work used 32 items from Askar and Davenport's (2009) Java Programming Self-Efficacy scale, where students rated their perceived self-efficacy in doing various programming-related tasks on a Likert-type scale. Askar and Davenport's (2009) administered the instrument with a sample from the similar target population as the current study consisting of English speakers who are undergraduate non-CS students using Java programming language. Their scale for Java Programming consisted of 32 items (as one construct), and the reliability of the scores from their sample was 0.99. In the current study, all 32 items in their original format were utilized. However, modifications were made because of the limited number of students who attended both classes. Hence, the scale from a 7-point Likert-type was reduced to scale to a 5-point Likert-type scale, ranging from 1 (not confident at all) to 5 (confident). A five-point scale rather than a seven-point scale was chosen based on the literature suggestion that five-point scale increases response rate and response quality along with reducing respondents' frustration level (Babakus & Mangold, 1992).

The questions created for the scale measuring the perceived value of learning programming were largely based on Baser's attitude survey (2013). This scale indicated to what degree students agreed with statements related to their perceived value of program learning. This scale consisted of five items on a 5-point Likert scale ranging from 1 (strongly disagree) to 5 (strongly agree). The last section of the survey contained questions about the age and gender of the participants as well as the name of the course they were taking. In addition, this section included two additional questions: (a) How many hours do you spend practicing the weekly assigned programs by the instructor? and (b) How many times this semester did you consult the TA for help in your assignments?

Validity and Reliability

The construct validity of the computer programming self-efficacy scale was examined via Exploratory Factor Analysis. The scale's allocation to the factors was specified through principal component analysis with oblique rotation (Promax). After an iterative process to examine scree plots and the eigenvalue, the scree plot clearly showed inflexions that would justify retaining two to three factors to extract. For meaningful interpretation, two factors were extracted out of 32 items, 20 items with load values over 0.3 were retained and included in the analysis, and 12 items with loads separated into two factors were excluded. The Kaiser-Meyer-Olkin (KMO) measure verified the sampling adequacy for the analysis, KMO = 0.90, and all KMO values for individual items were greater than 0.79, which was above the acceptable limit of 0.5 [20]. Bartlett's test of Sphericity, $\chi2$ (190) = 1620.100, $p< 0.001$, showed that there were patterned relationships between the items so that the factor analysis could be used (Field, 2009). The two factors explained a cumulative variance of 53.07% and were labeled as (a) independence and persistence in programming tasks and (b) scaffolding for programming. Table A in Appendix A presents the 20 retained items with their factor loadings and eigenvalues. Cronbach's alpha for the 20 retained programming self-efficacy items was 0.93. Individually, the reliability of independence and persistence scores were similarly high (α = 0.93, 16 items), and the reliability for scaffolding scores was slightly lower (α = 0.79, 4 items) but was still an acceptable value (Field, 2009).

In addition, convergent validity and discriminant validity were run to establish the construct validity for the five items measuring perceived value of learning. The output showed that two items needed to be dropped. These two items had Pearson correlation (r) < 0.30 with related variables, and Pearson correlation (r) > 0.20 with unrelated variables (Robinson, 2018). For the remaining three items that measure the perceived value of learning programming, Cronbach's alpha was high (α=.81), and the corrected item-total correlation was all above

0.30, which was encouraging (Field, 2009).

Results
This work employed IBM SPSS Statistics 21 to administer the survey and complete the data analyses. The data revealed that the self-efficacy perceptions levels of CIS students ranged from 37 to 99, with an overall mean of 3.55 (SD = 0.63). In terms of percentage distribution, 22.2% of students had a high level of self-efficacy perceptions (*M* > 4.00), 76.4% had a medium level of self-efficacy perceptions (*M* > 2.00 but < 4.00), and 1.4% of students had a low level of self-efficacy perceptions (*M* < 2.00). Relatively, students' perceived value of learning to program ranged from 7 to 15, with an over-all mean of 4.45 (*SD* = 0.63) and fell into medium and high level. In regard to percentage distribution, 72.1% of students perceived high value of programming, while 27.9% of students perceived medium value of programming.

Results from a G*Power analysis (a statistical tool used to estimate needed sample sizes based on the selected statistical test) indicated the sample size required to answer this study's research question (with medium effect size = 0.15, *α* err prob = 0.05, power (1-β err prob) = 0.90) was 108. The actual sample size used for the regression analysis was very suitable (*N* = 121).

Prior to conducting a hierarchical multiple regression, the relevant assumptions of this statistical analysis were tested. Firstly, a sample size of 121 out of 140 was deemed adequate, given four independent variables (course, value, practice time, and consulting TA) to be included in the analysis. The assumption of collinearity was also met as VIF scores were well below 10, and tolerance scores above 0.2. There were no influential cases biasing the model as the values of Cook's Distance were all under 1. Residual and scatter plots indicated the assumptions of normality, linearity, and homoscedasticity were all satisfied.

Two hierarchical multiple regression with four stages were conducted with Independence and Persistence and scaffolding (overall self-efficacy) as the dependent variables. Course was entered at stage one of the regressions as an extraneous variable to control for the variable course (control the difference between the two course levels). The Attachment variable consulting TA was entered at stage two; practice time was entered at stage three; and value was entered at stage four.

The first hierarchical multiple regression was ran with Independence and Persistence of Programming Self-Efficacy. Regression statistics are presented in Table 2. The analysis revealed that, at stage one, Course did not contribute significantly to the regression model, *F* (1, 119) = 0.188, *p* > 0.05, and accounted for 0.2 % (0.002) of the variation in Independence and Persistence of Programming Self-Efficacy (R^2 change = 0.2%). Introducing the attachment variable Consulting TA increased the value to 0.04, which meant Consulting TA accounted of 4.3% of the variation in Independence and Persistence of Programming Self-Efficacy, and

Variable	β	T	Sig.	R	R^2	ΔR^2
Step1				*.04*	*.00*	*.00*
Course	*.04*	*.43*	*.67*			
Step 2				*.20*	*.04*	*.04*
Course	*.02*	*.20*	*.84*			
Consult TA	*-.21*	*-2.27**	*.03*			
Step 3				*.21*	*.05*	*.00*
Course	*.03*	*.26*	*.79*			
Consult TA	*-2.26*	*-2.33**	*.02*			
Practice Time	*.06*	*.59*	*.55*			
Step 4				*.36*	*.13*	*.09*
Course	*.04*	*.45*	*.66*			
Consult TA	*-.18*	*-1.88*	*.06*			
Practice Time	*.03*	*.30*	*.77*			
Value	*.30*	*3.42***	*.00*			

Note: N=121, *p<.05, p**<.01

Table 2: Summary of hierarchical multiple regression Analysis for Variables Predicting Independence and Persistence of Programming Self-Efficacy.

this change in R^2 was not significant, F (2, 118) = 2.68, p > 0.05), (R^2 change = 4.2%).

Adding Practice Time variable to the regression model explained 4.6% (0.046) of the variation in Independence and Persistence of Programming Self-Efficacy, and this change in R^2 was not significant, F (3, 117) = 1.89, p > 0.05), (R^2 change = 0.3%). Finally, the addition of the variable Value explained 13.3% (0.13) of the variation in Independence and Persistence of Programming Self-Efficacy, and this change in R^2 was statistically significant, F (4, 116) = 4.47, p < 0.01), (R^2 change = 8.7%). When the four independent variables were included in stage four of the regression model, they explained 13.4% of the variance in Independence and Persistence of Programming Self-Efficacy, and the most important predictor in Independence and Persistence of Programming Self-Efficacy was the variable Value. The prediction power of the variable Value was moderate (.299) suggesting that approximately 29.9% of the variation in the variable Independence and Persistence of Programming Self-Efficacy (Dependent variable) can be explained by the variable Value (Independent variable).

The second hierarchical multiple regression was run with Scaffolding of Programming self-efficacy. Regression statistics are presented in Table 3. The analysis revealed that, at stage one, Course did not contribute significantly to the regression model, F (1, 119) = 2.90, p > 0.05, and accounted for 2.4 % (0.024) of the variation in Scaffolding of Programming Self-Efficacy (R2 change = 2.4%). Introducing the attachment variable consulting TA increased slightly the value to 0.025, which meant consulting TA accounted for 2.5% of the variation in Scaffolding of Programming Self-Efficacy, and this change in R^2 was not significant, F (2, 118) = 1.53, p > 0.05), (R^2 change = 0.0.1%). Adding Practice Time variable to the regression model explained 2.8% (0.028) of the variation in Scaffolding of Programming Self-Efficacy, and this change in R^2 was not significant, F (3, 117) = 1.11, p > 0.05), (R^2 change = 0.2%). Finally, the addition of the variable value explained 13% (0.13) of the variation in Scaffolding of Programming Self-Efficacy, and this change in R^2 was statistically significant, F (4, 116) = 4.33, p < 0.01, (R^2 change = 10.3%). When the four independent variables were included in stage four of the regression model, they explained 13% of the variance in Scaffolding of Programming Self-Efficacy, and the most important predictor in Scaffolding of Programming Self-Efficacy was the variable value.

In summary, the findings concluded that the variable value was the most important predictor for programming self-efficacy. All three variables (consult TA, practice time, and value) explained approximately 13% of the variance in programming self-efficacy.

5. DISCUSSION AND CONCLUSIONS

The findings in this study confirmed that the value of learning programming was the most predictable value among the three examined

Variable	β	T	Sig.	R	R^2	ΔR^2
Step1				*.15*	*.02*	*.02*
Course	*-.15*	*-1.70*	*.09*			
Step 2				*.15*	*.03*	*.00*
Course	*-.16*	*-1.73*	*.07*			
Consult TA	*-.04*	*-.41*	*.09*			
Step 3				*.16*	*.03*	*.00*
Course	*-.15*	*-1.66*	*.10*			
Consult TA	*-.06*	*-5.57*	*.57*			
Practice Time	*.05*	*.54*	*.59*			
Step 4				*.36*	*.13*	*.10*
Course	*-.14*	*-1.56*	*.12*			
Consult TA	*-.00*	*-.02*	*.98*			
Practice Time	*.02*	*.22*	*.83*			
Value	*.33*	*3.70**	*.00*			

Note: N=121, *$p<.001$

Table 3: Summary of hierarchical multiple regression Analysis for Variables Predicting scaffolding of Programming Self-Efficacy.

variables for both independence and persistence of programming self-efficacy and scaffolding. This indicated that the more students valued learning programming, the more they became independent and persisted in solving challenging programming problems. Relatively, students who valued programming received appropriate support or scaffolding. Therefore, the value of learning programming was an important factor that should be a part of educational interventions that seek enhancing IS students programming self-efficacy.

IS educators and instructional designers could clarify the utility of programming skills through various instructional methods. First, not only through explicit verbalization of course goals and usefulness but also through less direct means (Neuville, Frenay, & Bourgeois, 2007). For instance, educators could utilize professionals' stories of successful people in programming from around the world and meeting with guest speakers with IS degrees. Since one of the class activities provided to the participants in our study was utilizing this method and most of these students perceived high value of programming, this method could be supported.

Second, educators would need to stress how learning programming would be a "relevant and authentic" skill that has meaning in their career. Third, educators would activate students' personal interest through opportunities for choice and control over some academic activities. For example, they could constrain the general framework of an oral or written exercise (e.g., to have recourse to the theories developed in the course), while giving students the freedom to choose their own specific subject (Neuville, Frenay, & Bourgeois, 2007). This technique demonstrated its effectiveness when Denny, Cukierman, and Bhaskar (2015) in their experiment allowed students (*n* > 180) in an introductory programming course to invent numerous programming exercises. This technique helped students not only to be more exposed to real world problems but also develop confidence and skills which, in turn, assisted them to practically develop and assess their own value of learning programming.

The lack of any significant predictive relationship between programming self-efficacy of IS students, practice time, and TAs consultations from this study was unexpected. This finding contradicted Özmen and Altun (2014), who concluded that more practice time led to high programming self-efficacy among non-CS students. Moreover, the findings of this study conflicted with Wiggins, Grafsgaard, Boyer, Wiebe, and Lester's (2017), who concluded that tutoring tend to be associated with increased programming self-efficacy among CS students. These results did not lead us to ignore the influence of these variables on programming self-efficacy. Instead, it led us to think deeper and provide some interpretations that might justify our findings and improve our future research.

In this work, due to a lack of literature that presents the number of hours should students practice programming and consulting TAs, it was difficult to draw a conclusion or measure in the survey questions. Thus, this was an unavoidable step and it was considered subjective to students; it could be regarded as a limitation of this research.

As a result, there is one possible interpretation for finding a lack of any significant predictive relationship between programming self-efficacy of IS students, practice time, and TAs consultations. The subjective survey question format might have made the students unable to estimate the realistic number of hours practicing programming and consulting the TAs. It was hard for students to think and remember the number of practice hours while taking the survey. For example, they needed to estimate how much time it took them to download the practice examples from the course shell, think and manipulate the code, and then run them to solve problems. It was also realizable that it was difficult for students to provide realistic number of hours in case they think the practice examples are a waste of time since they are optional, not graded, not tracked, and solve a problem that is similar to the assignment; hence, they preferred to skip the practice examples and work immediately on the assignments instead. Furthermore, students might have thought that their need to contact the TA depended on the topic difficulty level; hence, estimating the consultation number of hours was not easy.

Since this study was exploratory among the population of IS students, more data is also needed to analyze in greater depth to increase our understanding of the relations between programming self-efficacy with both programming practice times and the number of consulting hours with TAs. For instance, students practice time should be tracked using one of the online labs such as Pearson MyLab where educators can monitor and track the amount of time students spend. Future studies should also ask students from the beginning of the semester to record the number of hours spent consulting

TAs and the reasons for consulting them. Students must be asked to turn in all this information for each assignment. Also, future data should provide more evidence regarding whether the students find consulting the TAs helpful and conducting an in-depth qualitative approach which gives access to richer, contextualized, and holistic descriptions.

This study led us to further validate a tool created to measure programming self-efficacy for our sample population of IS students. This allowed us to utilize the instrument to its full potential for this study and present evidence for future use in the IS population. Educators can use this instrument to identify their student's programming self-efficacy level and for more in-depth future studies looking at other factors that might enhance it. These research findings could contribute to the body of the literature, which, in turn, could promote accomplishing longitudinal evidence to prove or demonstrate the nature of the relationships between programming self-efficacy and factors that might be necessary to enhance it, specifically for IS students.

6. REFERENCES

Askar, P., & Davenport, D. (2009). "An investigation of factors related to self-efficacy for Java programming among engineering students." Online Submission 8(1).

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

Babakus, E., & Mangold, W. G. (1992). Adapting the SERVQUAL scale to hospital services: an empirical investigation. Health services research, 26(6), 767.

Baser, M. (2013). "Attitude, gender and achievement in computer programming." Online Submission 14(2):248–55.

Beyer, S. (2014). "Why are women underrepresented in Computer Science? Gender differences in stereotypes, self-efficacy, values, and interests and predictors of future CS course-taking and grades." Computer Science Education 24(2–3):153–92.

Bashir, G. M. M., & Hoque, A., S., M., L. (2016). "An effective learning and teaching model for programming languages." Journal of Computers in Education 3(4):413–37.

Bartimote-Aufflick, K., Bridgeman, A., Walker, R., Sharma, M., & Smith, L. (2016). The study, evaluation, and improvement of university student self-efficacy. Studies in Higher Education, 41(11), 1918-1942. https://doi.org/10.1080/03075079.2014.999319

Brooks, D. C., & Pomerantz, J. (2017). ECAR study of undergraduate students and information technology 2017 4(3): 3.

Bureau of Labor Statistics, U.S. Department of Labor, Occupational Outlook Handbook, Operations Research Analysts, at https://www.bls.gov/ooh/math/operations-research-analysts.htm (visited June 06, 2023).

Cigdem, H., & Yildirim, O. G. (2014). Predictors of C# programming language self-efficacy among vocational college students. International Journal on New Trends in Education and Their Implications, 5(3), 145-153. Retrieved from: www.ijonte.org

Denny, P., Cukierman, D., & Bhaskar, J. (2015, November). Measuring the effect of inventing practice exercises on learning in an introductory programming course. In Proceedings of the 15th Koli Calling Conference on Computing Education Research (pp. 13-22).

Field, A. (2009). Discovering statistics using SPSS statistics (3rd ed.). Thousand Oaks, CA: Sage publications.

Fong, C. J., Gilmore, J., Pinder-Grover, T., & Hatcher, M. (2019). "Examining the impact of four teaching development programmes for engineering teaching assistants." Journal of Further and Higher Education 43(3):363–80.

Forte, A., & Guzdial, M. (2005). Motivation and nonmajors in computer science: identifying discrete audiences for introductory courses." IEEE Transactions on Education 48(2):248–53.

Gist, M. E., & Mitchell, T. R. (1992). Self-efficacy: A theoretical analysis of its determinants and malleability. Academy of Management Review 17(2):183–211.

Gumelar, G., Martadi, M., Rosalinda, I., Yudhaningrum, L., & Warju, W. (2022, April). Computer Self-Efficacy, Task Value, Digital Literacy, Online Learning Perceptions on Indonesian University Students' Learning Satisfaction. In 1st World Conference on Social and Humanities Research (W-SHARE 2021) (pp. 94-99). Atlantis Press.

Gupta, S., & Bostrom, R. P. (2019). A revision of computer self-efficacy conceptualizations in information systems. ACM SIGMIS Database: The DATABASE for Advances in Information Systems, 50(2), 71-93.

Kalish, A., Rohdieck, S., Border, L., Schram, L. N., von Hoene, L., Palmer, M., ... & Horii, C. V. (2009). Structured Professional Development for Graduate and Professional Students: A Taxonomy. In Professional and Organizational Development Conference. Houston, TX.

Kajfez, R. L., & Matusovich, H. M. (2013). The future possible selves of graduate teaching assistants in first-year engineering programs. 5th First Year Engineering Education (FYEE).\

Konecki, M. (2014). Problems in programming education and means of their improvement. DAAAM International Scientific Book, 459-470. Retrieved from:

http://www.daaam.info/Downloads/Pdfs/science_books_pdfs/2014/Sc_Book_2014-037.pdf

Korkmaz, Ö., & Altun, H. (2014). Adapting computer programming self-efficacy scale and engineering students' self-efficacy perceptions. Online Submission, 1(1), 20-31.

Kori, K., Pedaste, M., Leijen, Ä., & Tõnisson, E. (2016). The role of programming experience in ICT students' learning motivation and academic achievement. International Journal of Information and Education Technology, 6(5), 331.

Kong, S. C. (2017). Development and validation of a programming self-efficacy scale for senior primary school learners. In S. C. Kong, J. Sheldon, & K. Y. Li (Eds.), Proceedings of the International Conference on Computational Thinking Education (pp. 97–102). The Education University of Hong Kong.

Latifah, A., & Nugraha, J. (2023). The influence of relevance and computer self-efficacy on students' behavioral intention in using the digital library. Jurnal Inovasi dan Teknologi Pembelajaran: Kajian dan Riset Dalam Teknologi Pembelajaran, 10(1), 92-105.

Lee, C., & Bobko, P. (1994). Self-efficacy beliefs: Comparison of five measures. Journal of Applied Psychology, 79(3), 364.

Malaquias, R. F., de Oliveira Malaquias, F. F., Ha, Y. M., & Hwang, Y. (2021). A cross-country study on intention to use mobile banking: Does computer self-efficacy matter? Journal of Global Information Management (JGIM), 29(2), 102-117.

Marakas, G. M., Aguirre-Urreta, M., Shoja, A., Kim, E., & Wang, S. (2022). The Computer Self-Efficacy Construct: A History of Application in Information Systems Research. Foundations and Trends® in Information Systems, 6(2), 94-170.

Ma, Y., Zhao, L., Li, N., & Wang, S. (2016). A new type of mobile learning resources—a probe into the development model of education APP [J]. China Educational Technology, 64–70.

Metcalfe, J., & Shimamura, A. P. (Eds.) (1994). Metacognition: Knowing about Knowing. MIT press.

Neuville, S., Frenay, M., & Bourgeois, E. (2007). Task value, self-efficacy and goal orientations: Impact on self-regulated learning, choice and performance among university students. Psychologica Belgica, 47(1).

Niitsoo, M., Paales, M., Pedaste, M., Siiman, L., & Tõnisson, E. (2014). Predictors of Informatics Students Progress and Graduation in University Studies. In International Technology, Education and Development Conference (pp. 380-392).

Nurhikmah, H., Farida, F., & Ervianti, E. (2021). The Impact of Computer-based Test and Students' Ability in Computer Self-Efficacy on Mathematics Learning Outcomes. Journal of Education Technology, 5(4).

O'neal, C., Wright, M., Cook, C., Perorazio, T., & Purkiss, J. (2007). The impact of teaching assistants on student retention in the sciences: Lessons for TA training. Journal of College Science Teaching, 36(5), 24.

Özmen, B., & Altun, A. (2014). Undergraduate students' experiences in programming: Difficulties and obstacles. Turkish Online Journal of Qualitative Inquiry, 5(3), 1-27.

Powell et al (2015). Evaluating the Effectiveness of Self-Created Student Screencasts as a Tool to Increase Student Learning Outcomes in a Hands-On Computer Programming Course. Information Systems Education Journal (ISEDJ) 13(5). and Powell, L. M., Wimmer, H. (2016. Information Systems Education Journal, 14(3) pp 85-95

Redmond, B. F. (2016). Self-efficacy and social cognitive case study. Overview of Social

Cognitive and Self-Efficacy Theories. Retrieved October 28, 2019 from: https://wikispaces.psu.edu/display/PSYCH484/7.+Self-Efficacy+and+Social+Cognitive+Theoris

Robinson, M. A. (2018). Using multi-item psychometric scales for research and practice in human resource management. Human Resource Management, 57(3), 739-750.

Rodgers, K. J., Marbouti, F., Shafaat, A., Jung, H., & Diefes-Dux, H. A. (2014). Influence of teaching assistants' motivation on student learning. In 2014 IEEE Frontiers in Education Conference (FIE) Proceedings (pp. 1-8). IEEE.

Rogerson, C., & Scott, E. (2010). The fear factor: How it affects students learning to program in a tertiary environment. Journal of Information Technology Education: Research, 9(1), 147-171.

Sheokand, J. (2022). COMPUTER SELF-EFFICACY CORELATION TO PERSONALITY OF UNDERGRADUATES IN HARYANA. Towards Excellence, 14(1).

Simpkins, S. D., Davis-Kean, P. E., & Eccles, J. S. (2006). Math and science motivation: A longitudinal examination of the links between choices and beliefs. Developmental psychology, 42(1), 70.

Schunk, D. H., & Pajares, F. (2005). Competence perceptions and academic functioning. Guilford Pres.

Tsai, C. Y. (2019). Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy. Computers in Human Behavior, 95, 224-232.

Tsai, M. J., Wang, C. Y., & Hsu, P. F. (2019). Developing the computer programming self-efficacy scale for computer literacy education. Journal of Educational Computing Research, 56(8), 1345-1360.

U.S. Department of Labor Bureau of Labor Statistics. (2018). Occupational outlook handbook, 2018-2019 edition. U.S. Department of Labor, Washington, D. C. Retrieved from: https://www.bls.gov/ooh/computer-and-information technology/home.htm

Van Dinther, M., Dochy, F., & Segers, M. (2011). Factors affecting students' self-efficacy in higher education. Educational research review, 6(2), 95-108.

Weigold, A., & Weigold, I. K. (2021). Measuring confidence engaging in computer activities at different skill levels: Development and validation of the Brief Inventory of Technology Self-Efficacy (BITS). Computers & Education, 169, 104210.

Wheeler, L. B., Maeng, J. L., Chiu, J. L., & Bell, R. L. (2017). Do teaching assistants matter? Investigating relationships between teaching assistants and student outcomes in undergraduate science laboratory classes. Journal of Research in Science Teaching, 54(4), 463-492.

Wigfield, A., Tonks, S., & Klauda, S. L. (2000). Expectancy–Value Theory of Achievement Motivation. Contemporary Educational Psychology.

Wiggins, J. B., Grafsgaard, J. F., Boyer, K. E., Wiebe, E. N., & Lester, J. C. (2017). Do you think you can? the influence of student self-efficacy on the effectiveness of tutorial dialogue for computer science. International Journal of Artificial Intelligence in Education, 27(1), 130-153.

Zhang, L., Li, B., Zhou, Y., & Chen, L. (2019). Can Fragmentation Learning Promote Students' Deep Learning in C Programming?. In Foundations and Trends in Smart Learning (pp. 51-60). Springer, Singapore.

Zimmerman, B. J. (2000). Self-Efficacy: An Essential Motive to Learn. Academic Press, Article 0361-476X/00.

APPENDIX A

	Independence and persistence in Programming Tasks	Scaffolding for Programming
26. I could come up with a suitable strategy for a given programming project in a short time.	*.78*	
3. I could write logically correct blocks of code using Java.	*.77*	
17. I could debug (correct all the errors) a long and complex program that I had written and make it work.	*.76x§*	
18. I could comprehend a long, complex multi-file program.	*.74*	
6. I could write a Java program that computes the average of any given number of numbers.	*.74*	
28. I could mentally trace through the execution of a long, complex multi-file program given to me.	*.73*	
8. I could build my own Java swing GUIs.	*.72*	
13. I could understand the object-oriented paradigm.	*.71*	
5. I could write a Java program that computes the average of three numbers.	*.71*	
11. I could write a long and complex Java program to solve any given problem as long as the specifications are clearly defined.	*.70*	
29. I could rewrite lengthy and confusing portions of code to be more readable and clear.	*.65*	
12. I could organize and design my program in a modular manner.	*.63*	
10. I could write a reasonably sized Java program that can solve a problem this is only vaguely familiar to me.	*.62*	
14. I could identify the objects in the problem domain and could declare, define, and use them.	*.62*	
27. I could manage my time efficiently if I had a pressing deadline on a programming project	*.58*	
9. I could write a small Java program given a small problem that is familiar to me	*.51*	
21. I could complete a programming project if I could call someone for help if I got stuck.		*.91*
22. I could complete a programming project once someone else helped me get started.		*.90*
19. I could complete a programming project if someone showed me how to solve the problem first.		*.74*
24. I could complete a programming project if I had just the built-in help facility for assistance.		*.61*
Eigen value	*8.75*	*1.86*
% of variance	*43.77*	*9.30*
α	*.93*	*.79*

Note: N=140

Table A: Items Loadings for the Two Factors of Self-efficacy