



Volume 18, 2019

TOWARDS ENHANCING PROGRAMMING SELF-EFFICACY PERCEPTIONS AMONG UNDERGRADUATE INFORMATION SYSTEMS STUDENTS

Ramadan Abdunabi*	Colorado State University, Fort Collins, USA	ramadan.abdunabi@colostate.edu
Ilham Hbaci	University of Northern Colorado, Greeley, USA	hbac3324@bears.unco.edu
Heng-Yu Ku	University of Northern Colorado, Greeley, USA	HengYu.Ku@unco.edu

* Corresponding author

ABSTRACT

Aim/Purpose	Currently, Information Systems (IS) departments in business schools are moving towards integrating learning to program or code in their undergraduate core courses. Many factors affecting IS student success in learning to program have been identified, but there is still a dearth of knowledge about student perceptions on their own competence. The purpose of this study was to investigate factors that may affect the success of IS students in learning to program.
Background	Students' perceptions about the value and difficulties to learn programming can affect their skills acquisition. IS educators need to understand the student perception related to difficulties of learning to program in order to offer more effective support during their teaching process and interactions with students. To address this need, this study examines two critical elements to improve teaching IS programming courses: (a) Programming Self-Efficacy—students' beliefs on their own programming competence, combined with (b) levels of programming skills which IS students initially thought to learn for their future profession.
Methodology	This study uses quantitative data drawn from undergraduate students in a Computer Information Systems classes at Colorado State University in U.S.A. and supported by qualitative data.

Accepting Editor Keith A. Willoughby | Received: January 27, 2019 | Revised: March 31, 2019 | Accepted: April 9, 2019.

Cite as: Abdunabi, R., Hbaci, I., & Ku, H-Y. (2019). Towards enhancing programming self-efficacy perceptions among undergraduate information systems students. *Journal of Information Technology Education: Research*, 18, 185-206. <https://doi.org/10.28945/4308>

(CC BY-NC 4.0) This article is licensed to you under a [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/). When you copy and redistribute this paper in full or in part, you need to provide proper attribution to it to ensure that others can later locate this work (and to ensure that others do not accuse you of plagiarism). You may (and we encourage you to) adapt, remix, transform, and build upon the material for any non-commercial purposes. This license does not permit you to use this material for commercial purposes.

Contribution	Quantitative data measures the correlation between students' programming self-efficacy, their perceived value of programming, their own practice time, and the frequency of teaching assistant (TA) consultations. The qualitative data was utilized to understand students' thoughts of the programming skills they need in their future profession that may influence their programming self-efficacy during the learning process. The importance of this study lies in the potential that the findings of this study are critical to investigate the most influential factors that are likely to be a vehicle through which educators can either improve self-efficacy of their students and/or understand it more fully. Furthermore, these findings may influence pedagogical practices for teaching programming languages in higher education contexts more successfully. For instance, applying a contextual learning approach may assist in identifying the most effective approach to teach programming courses, and in turn, will lead to increased learning outcomes as encountered and narrated by IS students.
Findings	The correlation analysis indicated a significant positive correlation between IS students' programming self-efficacy and their perceived value of learning programming. However, the practice time and frequency of TA consultations had no significant relationship with programming self-efficacy. In addition, the qualitative data revealed a clear placement of IS students' vision of their future coding level into five categorical programming skills: novice, communicator, end-user, and professional, with a new category of "competent" emerging.
Recommendations for Practitioners	The study suggests that IS Educators need to embed interventions for increasing IS students' perceived value of learning programming and practice time. It is also very effective to associate the class activities with real life projects. Furthermore, we suggest to educators to apply the contextual learning approach that would support higher levels of value for programming and programming self-efficacy among IS students. Also, coordination between educators and employers can aid in developing effective programming classes, improving IS students' job marketability.
Recommendations for Researchers	Investigating other factors that potentially contribute to IS students' programming self-efficacy, such as previous computer programming and math exposure, motivation, and economic status.
Impact on Society	Realizing the importance of the programming self-efficacy could help IS educators to teach effective and efficient programming courses that ultimately result in students learning programming with high acquisition and less distress. Highlighting the importance of linking what the market needs with the course content would increase students' programming self-efficacy and their chances of obtaining jobs.
Future Research	An interactive programming tool is a suggested supplement for IS educators to increase student enthusiasm about practice time that would support students work on their own and enjoy the class, and educators would be capable to accurately track and assess students' participations.
Keywords	information systems, programming self-efficacy, IS student classification, competent programmers

INTRODUCTION

Recent years have witnessed contemporary business and jobs in demand throughout the United States, which require Information Systems (IS) majors to possess competent programming skills

(Konecki, 2014). Examples of IS jobs that require coding skills include business analyst, data analyst, data mining, project management, software engineer, software developer, software tester, systems analyst, systems designer, database administrators, and network administrators. Employment in these areas is projected to grow 13% from 2016 to 2026, which is higher than the projected growth for all other occupations. This growth includes a 15% increase for web developers, a 11% increase for software developers, a 9% increase for computer system analysts, and an 11% increase for database administrators (U.S. Department of Labor, 2018). Because of the importance of programming activities at workplace for IS majors, IS departments require students to complete programming courses that are tailored to them in order to perform effectively in their future careers. For IS student success, and to prepare them for the working world, it is important for IS educators to recognize the needs of IS students and provide effective support in their programming courses.

Despite the demand for programming skills in today's workplaces, undergraduate IS students, who have little or no prior exposure to programming, express difficulties learning computer programming in core IS courses, and they struggle to complete introductory programming courses with successful performance (Bashir & Hoque, 2016). IS students encounter difficulties understanding the underlying concepts and struggle writing error-free programs that meet the stated requirements in their introductory programming course (Forte & Guzdial, 2005; Wiedenbeck, 2005). Several empirical studies have established as many as one-third of students who take introductory programming courses either fail or withdraw from these courses (X. Zhang, Zhang, Stafford, & Zhang, 2013). Furthermore, several studies have shown that the dropout and failure rates in introductory programming courses at the university level are evidence to the fact that learning to program is a difficult task (Dasuki & Quaye, 2016; Vihavainen, Paksula, & Luukkainen, 2011; Wiedenbeck, Labelle, & Kain, 2004). Numerous factors influencing student success regarding programming have been identified (Dasuki, & Quaye, 2016; Wiedenbeck, 2005), but there is still a dearth of knowledge about how student perceptions affect course outcomes. To tackle this challenge, the goal of this study was to explore some factors related to student perceptions that might contribute to IS student success with learning programming.

The sources of students' apprehension about and difficulty with learning computer programming are varied, and some research indicates that personal characteristics and internal factors such as self-efficacy—students' belief in their own programming competence—are likely to play a crucial role in levels of struggle with programming and a student's ability to overcome them (Askar & Davenport, 2009; Pajares, 1996). The programming self-efficacy influences students' choices of activities, the amount of effort they spend in accomplishing a task, and how long they will persist in a stressful situation to perform that task (Bandura, 1977).

Although self-efficacy is an internal factor that describes a psychological attribute that can be altered by students themselves and an educator has no direct way to change it (Rogerson & Scott, 2010), there are various studies that have attempted to investigate factors that assist educators to influence students' programming self-efficacy in an effort to reduce their struggle with learning programming (Rogerson & Scott, 2010). Substantial research has investigated various factors involved in the potential relationship between learning computer programming and perceptions of self-efficacy because programming self-efficacy has been shown to be associated with greater persistence and willingness to approach new and challenging programming tasks (Cigdem & Yildirim, 2014; Dang, Zhang, Ravindran, & Osmonbekov, 2016; Korkmaz & Altun, 2014; Rogerson & Scott, 2010; Y. G. Zhang & Dang, 2015). As observed by Cigdem and Yildirim (2014), Özmen and Altun (2014), Rogerson and Scott (2010), and Wiggins, Grafsgaard, Boyer, Wiebe, and Lester (2016), the time students spent practicing, use of assistance, and how they value learning programming influence learning programming skills.

The literature indicates that most studies identifying obstacles to learning computer programming, sources of struggle, and ways to help only focus on Computer Sciences (CS) students who are required to design and implement efficient computer programs, software, and applications (Askar &

Davenport, 2009; Bashir & Hoque, 2016; Cigdem & Yildirim, 2014; Konecki, 2014; Özmen & Altun, 2014; Rogerson & Scott, 2010). On the other hand, IS students' role is to use, modify, and troubleshoot those programs, software, and applications. Therefore, CS students may have a motivational advantage over their non-major peers in learning programming. CS majors have chosen to learn programming, whereas non-majors are often required to code regardless of their personal opinions of its value or utility. Such studies vary in research results which make it difficult to draw any conclusions regarding reliable predictors for students' programming self-efficacy, particularly among IS majors in business schools.

In addition to students' challenge in learning program, it is also crucial for the IS educators to ensure they teach students the appropriate amount of computer programming skills conforming to their students' expectation of what programming level to learn, which may influence student perceptions. IS students might need programming skills that are beyond those of an end-user programmer (Barker, 2002), but they do not need extensive professional programming skills. Non-professional programmers, including IS students, need to code, such as creating macros, spreadsheet, formulas, task automation scripts, and/or dynamic web applications (Boehm et al., 2000). Teaching IS students is a challenge because their major does not solely focus on learning programming and computing that allows them to obtain professional foundational skills that can be applied towards any career in coding. Instead of having students solely focus on coding and programming, IS students are required to learn basic programming yet give insight into other facets such as project management, managerial communications, and/or business administration. Therefore, the goal of this study was also an attempt to find out how much focus should be given to teaching IS students about programming and coding, and what they need to be successful in learning programming in order to be well prepared for their future professions.

In an effort to help IS educators enhance their students' programming competency, this study researched into a psychological aspect that potentially reduces the challenges and difficulties of learning to program, as encountered and narrated by IS students. We used correlational analysis to investigate whether the three variables, 1) participants' perceived value of programming, 2) how much time per week participants spent practicing programming assignments, and 3) the number of times participants consulted teaching assistants, are related to IS students' programming self-efficacy scores of two IS programming courses. Understanding these variables should guide IS educators to develop class activities that could increase the programming self-efficacy among IS students. In addition, we used the open-ended question portion of the instrument to investigate what level of programming is suitable for IS students to satisfy student expectations of their level of coding to learn. This data will provide IS educators a better understanding of the population they teach and whether the amount of programming is suitable for them.

The paper continues with Literature Review and Hypotheses Development, which presents a review and summary of the discussions reported in literature pertained to this study, research hypothesizes, objectives, and questions. Then, we present processes performed in our research method emphasizing objective measurements and the statistical analysis of data collected through cross-sectional surveys. The paper ends with a discussion of the research findings and offer suggestions of interventions for IS educators, and then concludes with directions to future work.

LITERATURE REVIEW AND HYPOTHESES DEVELOPMENT

Difficulties faced by students while learning programming contribute directly to different types and different levels of perceptions of programming (Tan, Ting, & Ling, 2009). For example, students' perceptions of their own competence in learning (self-efficacy) is one of the reliable indicators that predicts a person's performance (Askar & Davenport, 2009). Self-efficacy has been defined as "learners' judgements of their capabilities to organize and execute courses of action required to attain designated types of performances" (Bandura, 1986, p. 391). It is potentially critical in an educational setting because students with high self-efficacy are more likely to accept challenging tasks and put

forth more effort to accomplish them than students who have low self-efficacy. Therefore, students with high self-efficacy tend to succeed in their future professions when they are compared with students with low-self- efficacy, who are less likely to pursue new learning opportunities (Askar & Davenport, 2009).

Furthermore, beliefs held by individuals about their capabilities and about the outcomes of their efforts strongly influence the way they will behave (Pajares, 1996). Consequently, self-efficacy potentially has a strong relationship with students' choices of what to learn and the effort they think they will need to exert to meet their learning goals (Rogerson & Scott, 2010). Previous research found that self-efficacy is not only associated with learning a subject and willingness to learn (Askar & Davenport, 2009; Fasogbon, Jegede, Adetan, & Aderbigbe, 2016; Y. G. Zhang & Dang, 2015), but also is related to students' orientation toward jobs in computer and information systems related areas, as students with high self-efficacy are more likely willing to choose Computer Information Systems as the area of their profession (Rosson, Carroll, & Sinha, 2011).

Towards gaining better background related to our first research question, we found in existing research that 1) value of learning programming, 2) practice programming, and 3) consulting TAs are the three key factors that could influence student perceptions of self-efficacy. For example, it has been argued that the value of learning programming is a catalyst to learning and achievement (X. Zhang et al., 2013), and students who value the subject matter are more likely to apply deep-level of learning techniques (Blumenfeld, Kempler, & Krajcik, 2006). Wu, Tennyson, and Hsia (2010), articulated that students' computer self-efficacy is related to how they value the way of learning. Research that seeks to address the needs of computer programming for students has also highlighted that the perceived value of programming skills among non-CS students is likely to have important implications for developing suitable programming courses for students with varying needs (Korkmaz & Altun, 2014; X. Zhang et al., 2013). As stated by Bartimote-Aufflick, Bridgeman, Walker, Sharma, and Smith (2016), the value of learning among undergraduate students is one of the variables that are repeatedly highly correlated with self-efficacy, and it is likely to be a vehicle through which educators can either improve self-efficacy of their students and/or understand it more fully. Based on that, we stated our first hypothesis as the following:

Hypothesis 1: There is an association between the value of learning programming and student programming self-efficacy.

Lack of practice in programming is considered as another reason for failure in programming (Özmen & Altun, 2014) because programming is all about solving problems via various lines of coding. Students might visualize various ways to solve a problem, but practically, they need to write the code to obtain tangible output. Niiitsoo, Paales, Pedaste, Siiman, and Tõnisson (2014) stated that time spent practicing programming during the semester was a significant predictor of Information and Computer Technology students' academic performance, with students who spent more time programming having better performance. In our research, we adopted the factor of time spent to practice programming that allow students to make mistakes in an ungraded environment, but we decided to investigate how it is related to student programming self-efficacy, and test the hypothesis:

Hypothesis 2: There is an association between programming practice and student programming self-efficacy.

Wiggins et al. (2016) investigated the relationship between 66 undergraduate students' programming self-efficacy and human-to-human computer-mediated tutorial dialogue. Wiggins et al. (2016) found that the social dialogue between participants who were novices and had no substantial prior programming experience and tutoring tends to be associated with increased self-efficacy. Particularly, tutoring was helpful for students with high self-efficacy. The students with high self-efficacy demonstrated more advanced self-regulated learning strategies, and they were receptive to feedback. For students with low self-efficacy, guidance may provide greater learning, but it may also increase frustration. Relatively, Kinnunen and Malmi (2006) found tutoring or TAs' pedagogical inability to help students was an issue related to students dropping out of Introductory of Computer Science among

non-CS majors. As discussed, the idea of having a teaching assistant to support students' learning is not new, but current research that focuses on how tutoring relates to programming self-efficacy is limited and has not been investigated with IS students. Hence, we hypothesize:

Hypothesis 3: There is an association between consulting TA(s) and student programming self-efficacy.

Finally, to help answer the second research question of our study, we focused on Chilana et al.'s (2015) study that completed research with non-CS undergraduate students. Chilana et al. identified programmer categories for students with various career expectations based on their level of valuing programming. Chilana et al. identified four programming skill categories: 1) non-programmer (who does not know how to write a code), 2) conversational programmer (who is more programming literate than non-programmer but not necessarily enough to be an end-user), 3) end-user programmer (who is able to use software or applications for data entry purposes and manipulate some lines of codes of these programs to meet specific needs), and 4) professional programmer (who is able to solve problems and produce products via programming). Chilana et al. found that some non-CS majors value learning programming because they want to be programmers or end-user programmers (e.g., data analysts or project managers). Other students expressed different reasons for valuing learning programming. Those students who were more programming literate than non-programmers but were not end-user programmers were classified as conversational programmers. Conversational programmers are not learning programming to be professional or end-user programmers; rather, they need to learn programming "for the pragmatic reason of being able to converse in the programmer's languages and improving their perceived marketability in the software industry" (Chilana et al., 2015, p. 252).

Consequently, it is worthwhile to find a category for IS students as programmers because they have not been classified in the literature as a specific type of programmer relative to Chilana et al.'s (2015) programming categories. According to Dreyfus and Dreyfus (1986), IS students do not need to be professional programmers, but they can become experts after many years of experience and practice. Furthermore, Barker (2002) considered IS students as end-users that were defined by Lieberman, Paternò, Klann, and Wulf (2006) as users "who are acting as non-professional software developers, at some point to create, modify or extend a software artifact" (p. 2). Barker articulated that IS students experience problems with being end-user programmers "due to incomplete information, incorrect design procedures, and inadequate software knowledge" (p. 62). This problem indicates that IS students might stand in between the end-user and professional programmer categories. Therefore, our research investigations could assist IS educators to have better understandings of the students they teach.

THEORETICAL FRAMEWORK

This study is guided by the self-efficacy theory of Bandura (1977), which states that the more individuals believe they can accomplish a task using their skills under certain circumstances, the more they can complete it and succeed. The basic principle behind self-efficacy theory is that individuals are more likely to engage in activities for which they have high self-efficacy and less likely to engage in those they do not (Bandura, 1977). Hence, in educational settings, efforts should focus on increasing students' self-efficacy to improve their academic performance and achievement. Individuals gain satisfaction when they achieve goals they value, and when they achieve these valued goals, they are more likely to continue to extend a high level of effort (Bandura, 1988). In addition, Bandura (1988) also emphasizes that setting valued goals is a significant resource to build self-efficacy. As a result, the value of learning a skill (such as programming) and identifying explicit goals for learning this skill are both important factors that need to be considered to build this skill of self-efficacy.

Furthermore, relying on Bandura's self-efficacy theory, Gist and Mitchell (1992) articulated that there are some processes need to be highlighted to assess and interpret individuals' self-efficacy. The three assessment processes for self-efficacy are the analysis of task requirements (an individual's determi-

nation of what it takes to perform a task), attributional analysis of experience (an individual's judgment about why a performance level occurred), and assessment of personal and situational resources/ constraints (an individual's consideration of personal and situational factors).

Those three assessment processes of self-efficacy play a critical role in academic success (Redmond, 2010). For instance, the analysis of task requirements includes the time dedicated to the course work (Redmond, 2010), which indicates that the time a student spends to learn programming skill can be considered one of the factors that relates to a student programming self-efficacy. In addition, the attributional analysis of experience provides personal perception and understanding that a student has in regard to why he/she reached a specific performance level (Redmond, 2010). One of those attributes that influence students' learning at a specific level is the availability of communication between students and professor/teaching assistant (TA) (Redmond, 2010). This explicitly indicates that students' call for support from their TAs might contribute to their programming self-efficacy. Finally, personal and situational resources include a confirmation for the students taking courses at an appropriate level in which they can succeed (Redmond, 2010). We believe this attempt can help IS students to express their goals of learning to program and to help educators to estimate the amount of programming provided for them in the material.

Finally, drawing on this framework, we highlighted the factors that are essential to build and assess programming self-efficacy (see Figure 1) and identify two objectives with two research questions for this study.

- **Objective 1:** Assisting IS educators to make a decision about whether they need to consider the factors (the value of programming, their own practice time, as well as the frequency of TA consultations) with high or low priority in creating their class activities.
- **RQ1:** Do IS students' perceived value of learning programming, their own practice time, and frequency of TA consultations associate with their programming self-efficacy?
- **Objective 2:** Increasing IS educators' understanding of the population they teach by specifying students' expectations about their belief on programming levels conform to their learning goals.
- **RQ2:** What are IS students' expectations about their belief on programming levels conform to their learning goals?

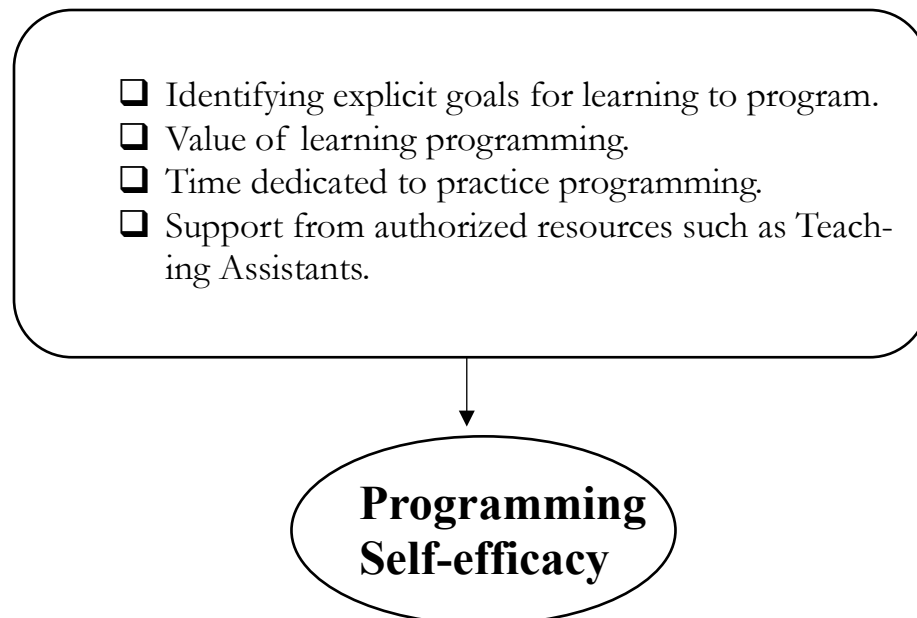


Figure 1. Theoretical framework: External factors essential to build and assess programming self-efficacy.

METHODOLOGY

RESEARCH PROCESS AND MEASUREMENTS

A survey research design was used in this study. The survey link created using Qualtrics was distributed to the students of the two Java courses at the end of the Spring 2017 and Spring 2018 semesters. An extra credit assignment worth 5 points was provided as an incentive for voluntary student participation. Each participant completed a set of questions in four sections: student Java programming self-efficacy, perceived value of learning programming, future use of programming, and demographic characteristics.

To measure students' Java programming self-efficacy, we used 32 items from Askar and Davenport's (2009) Java Programming Self-Efficacy scale, where students rated their perceived self-efficacy with various Java programming-related tasks on a Likert-type scale. The survey was modified slightly for use in the current study by reducing the scale from a 7-point Likert-type scale to a 5-point Likert-type scale, ranging from 1 (not confident at all) to 5 (confident). This modification was made in an effort to maximize the completion rate obtained by reducing the cognitive load required to complete the survey (Driscoll, 2005), thus making the scale less time consuming for participants.

The questions created for the scale measuring the perceived value of learning programming section were largely based on Baser's (2013) attitude survey. This scale indicates to what degree students agree with statements related to their perceived value of program learning. This scale consists of five items on a 5-point Likert scale ranging from 1 (strongly disagree) to 5 (strongly agree). The future use of programming section contains only one open-ended question (e.g., explain how you view programming as being a part of your future profession) related to the future use of programming from the participants' perspectives.

The last section of the survey contains questions about age and gender of the participants as well as the name of the course they were taking. In addition, this section includes additional two questions (a) How many hours do you spend to practice the weekly assigned programs by the instructor? and (b) How many times in this semester did you consult the teaching assistant for help in your assignments?

The appendix presents the questionnaire of the study.

COURSE INFORMATION AND DESIGN

The first junior-level programming course was titled as "Application Design and Development" and the second senior-level programming course was titled as "Advanced Application Design and Development." These two courses teach the general-purpose programming language, Java. The Java programming language is of specific interest to IS majors because of its prominent, standard industry programming language, so that students will be well-prepared for their future professions.

The activities that are accompanied with the two courses include weekly self-practice examples uploaded along with the weekly material of the courses on Canvas (a Learning Management System). The practice examples were resource of code reuse that would help students to write their actual programming assignments. Almost all the code fractions, blocks, and/or statements that students need for their assignments are in those practice programs. This would help students to self-practice through a non-graded environment, and they would be able to make changes in the code and observe the outputs. In addition, students are provided with a graded activity for participation in an in-class-exercise, where they are observed and helped by the TAs to solve some multi-step problems related to the programming actual assignments. To obtain additional support from the TAs, students are provided lab hours to communicate with TAs and can also use Canvas to send their questions to them as well.

Other activities that have been added in these two courses are viewing interviews with influential people in the sector of computer programming, such as the founders of Microsoft, Facebook, Twitter, and Dropbox, and inviting guest speakers who graduated with an IS degree and shared with student their programming experience that helped them to obtain IS jobs. Finally, students are supported with a general help center for all IS courses provided at the College of Business, which provides another source for them to understand concepts, complete homework assignments, and ask questions.

PARTICIPANTS

The study was conducted under the approval of the Institutional Review Board at a large state university in the Western region of the United States using a nonprobability convenience sampling method. Convenience sampling allowed data collection within time and place constraints. The criteria for selection were the following: (a) the target population was undergraduate business students, (b) the accessible population was undergraduate IS students from the College of Business, (c) participants were 18 years of age or older, and (d) each participant was taking either a junior-level programming course or a senior-level programming course. A total of 140 students completed the survey, and their demographic information is presented in Table 1.

Table 1. Participants' Characteristics

PARTICIPANT 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.

DATA VALIDITY, RELIABILITY, AND ANALYSIS

VALIDITY AND RELIABILITY

Computer programming self-efficacy scale

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 principle component analysis with oblique rotation (Promax). After an iterative process to examine scree plots and eigenvalues, the scree plot clearly showed inflexions that would justify retaining two factors to extract. Out of 32 items, 20 items that had load values over .3 were retained and included in the analysis, and 12 items with loads separated into two factors were excluded (Büyüköztürk, 2002). The Kaiser-Meyer-Olkin (KMO) measure verified the sampling adequacy for the analysis, KMO = .90, and all KMO values for individual items were greater than .79, which is above the acceptable limit of .5 (Field, 2013). Bartlett's test of Sphericity, $\chi^2(190) = 1620.100, p < .001$, showed that there were patterned relationships between the items, so the factor analysis can be used (Field, 2009). The two factors explained a cumulative variance of 53.07% and are labeled as (1) independence and persistence in programming tasks (16 items), and (2) scaffolding for programming (4 items). Table 2 presents the retained 20 items with their factor loadings and Eigen values. The Cronbach's alpha for the 20 retained

Java programming self-efficacy items was .93. Individually, the reliability of independence and persistence in programming tasks scores was similarly high ($\alpha = .93$), and the reliability for scaffolding for programming scores was slightly lower ($\alpha = .79$), but it is still an acceptable value (Field, 2013).

Table 2. Items Loadings for the Computer Programming Self-Efficacy Scale

	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.	.76	
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 clearer.	.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	

	INDEPENDENCE AND PERSISTENCE IN PROGRAMMING TASKS	SCAFFOLDING FOR PROGRAMMING
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

Perceived value of learning programming scale

In addition, convergent validity and discriminant validity were run to establish the construct validity to 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) < .30 with related variables, and Pearson correlation (r) > .20 with unrelated variables (Robinson, 2018). For the remaining three items measuring perceived value of learning programming, Cronbach's alpha was high ($\alpha = .81$), and the corrected item-total correlation are all above .3, which is encouraging (Field, 2016).

DATA ANALYSIS

We used IBM SPSS Statistics 21 to administer the survey and complete the data analyses. The statistical partial correlation that does not make the distinction between independent and dependent variables was chosen to analyze the data because we attempted to measure relationships between variables (independent and dependent variables) whilst controlling the effect of a third variable (course). All assumptions to test the compatibility between the obtained data and the statistical partial correlation were tested including test of outliers, normality, and linearity. When deviation from the assumption of a normal distribution is presented, parametric tests should not be used; equivalent non-parametric

tests should be used instead (Gall, Borg, & Gall, 1996). Non-parametric tests are tests of statistical significance, distribution free tests, and yield the same level of statistical significance as parametric tests when the sample size is large, i.e., 30+ (Gall et al., 1996; Pallant, 2007). Although non-parametric partial correlation test was conducted instead of the parametric partial correlation due to assumptions violation (e.g. normality), the parametric and non-parametric partial correlation tests provided the same conclusion.

FINDINGS

The data revealed that the self-efficacy perceptions levels of IS students ranged from 37 to 99, with an overall mean of 3.55 (SD = .63). In terms of percentage distribution, 22.2% of students have high level of self-efficacy perceptions ($M > 4.00$), 76.4% have medium level of self-efficacy perceptions ($M \geq 2.00$ but ≤ 4.00), and 1.4% of students have low level of self-efficacy perceptions ($M < 2.00$). Relatively, students' perceived value of learning to program ranged from 7 to 15, with an overall mean of 4.45 (SD = .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.

Towards meeting our goal, which is measuring the correlation between programming self-efficacy and three related factors, the output of running the first non-parametric parametric partial correlation test showed that there was a statistically significant, moderate, positive correlation between student independence and persistence in programming tasks and perceived value of learning programming whilst controlling for course ($r(118) = .306$, $N = 121$, $p = .001$), supporting *Hypothesis 1*. In addition, there was a statistically significant, low, negative correlation between student independence and persistence in programming tasks and times of consulting TA for help whilst controlling for course ($r(118) = -.199$, $N = 121$, $p = .030$), supporting *Hypothesis 3*. However, the analysis showed that there was no significant correlation between independence and persistence in programming tasks and number of practicing programming hours whilst controlling for course ($r(118) = .004$, $N = 121$, $p = .965$). Therefore, *Hypothesis 2* was not supported.

The output of running the second non-parametric parametric partial correlation test showed that there was a statistically significant, moderate positive correlation between student scaffolding for programming and perceived value of learning programming whilst controlling for course ($r(118) = .397$, $N = 121$, $p < .001$), supporting *Hypothesis 1* as well. On the other hand, there was no significant correlation between scaffolding for programming and times of consulting TA for help ($r(118) = -.113$, $N = 121$, $p = .219$), and between scaffolding for programming and number of practicing programming hours ($r(118) = .031$, $N = 121$, $p = .740$) whilst controlling for course. Table 3 presents a summary of how these findings support (or contradict) with the study hypotheses.

Table 3. Summary of How Findings Support (or Contradict) with the Hypotheses.

HYPOTHESES	FINDINGS
<i>Hypothesis 1:</i> There is an association between the value of learning programming and student programming self-efficacy.	There was a statistically significant correlation between student overall self-efficacy (independence and persistence in programming tasks and student scaffolding for programming) and perceived value of learning programming whilst controlling for course; hence, <i>Hypothesis 1 is supported.</i>

HYPOTHESES	FINDINGS
<i>Hypothesis 2:</i> There is an association between programming practice and student programming self-efficacy.	There was no significant correlation between overall self-efficacy (independence and persistence in programming tasks and student scaffolding for programming) and number of practicing programming hours whilst controlling for course; hence, <i>Hypothesis 2 is not supported.</i>
<i>Hypothesis 3:</i> There is an association between consulting TA(s) and student programming self-efficacy.	There was a statistically significant correlation between student independence and persistence in programming tasks (a factor of Programming self-efficacy) and times of consulting TA for help whilst controlling for course, <i>supporting Hypothesis 3.</i> However, there was no significant correlation between scaffolding for programming (a factor of Programming self-efficacy) and times of consulting TA for help whilst controlling for course which <i>contradicts with Hypothesis 3.</i>

Finally, towards meeting our second goal, which is identifying business students' future profession as programmers, we analyzed the open-ended question on the survey using frequency of participants' responses and completing qualitative coding of participants' responses. A total of 127 out of 140 participants responded to the prompt of "Explain how you view programming as being a part in your future profession". The data revealed that categories representing IS undergraduates are similar to Chilana et al.'s (2015) four categories for skill classification (novice, communicator, end-user, and professional), with a new category of "competent" emerging.

The results revealed that only seven students (5%) expected to be novice programmers in their future professions. This group showed limited willingness to expand their knowledge beyond what they have acquired from the general technical environment, and they expect to hold jobs with basic technology use. A total of 20 participants (16%) viewed themselves as future communicators—those who can read and write basic lines of code and work effectively with expert programmers in their future workplaces. In addition, 32 students (25%) anticipated being end-user programmers—those who are able to use software and applications for data entry and analysis purposes and to develop pieces of code for these applications that are tailored to specific needs (such as data analyst).

Furthermore, 54 participants (43%) viewed themselves as being competent programmers—those who are able to write a complete code (e.g., complete program or website) but not as advanced as CS programmers. Some expected positions for this group would be software, applications, and/or website developers. This group of participants is categorized as a new category of programmers that is not found in Chilana et al.'s (2015) four categories of programmers. Finally, only 14 participants (11%) viewed themselves as future professional programmers—those who are able to write a complete and advanced code as professionally as CS majors. These individuals will have job positions higher than the competent group, such as senior level of software specialists, applications programmers, and/or website developers. Figure 2 illustrates the distribution of participants' responses.

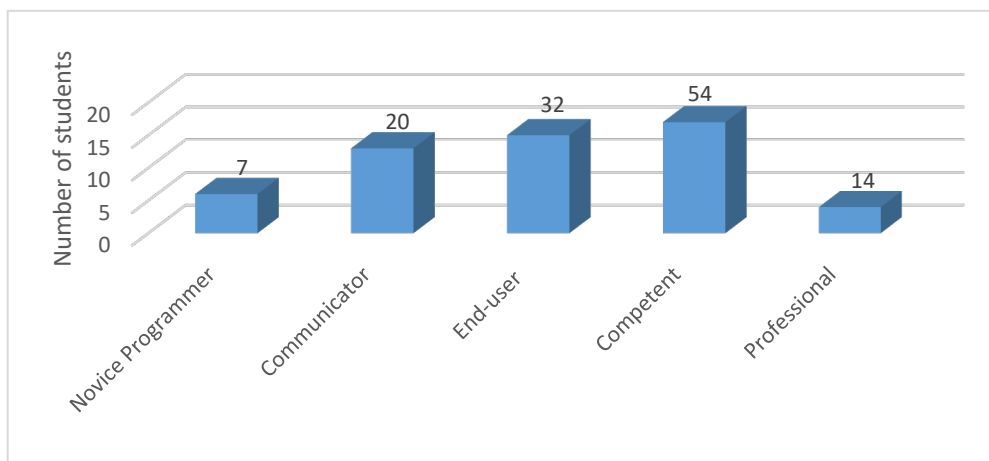


Figure 2. Undergraduate Information Systems students’ view of their programming skills needed for their future profession.

As a result of the distribution of IS students’ vision of their future coding career, we developed Figure 3, which represents undergraduate IS students’ programming categories. None of the participants viewed themselves as being experts, as this category is unlikely to be achieved by undergraduate students.







Novice	Communicator	End-User	Competent	Professional	Expert
					
Very basic programming knowledge	Read and write basic lines of coding	Use software for data entry and analysis, able to write, and manipulate pieces of code	Responsible to write a complete code of a program, but not sufficient	Write a complete sufficient code of program or website	After many years of experience since graduation

Figure 3. Categories of programming skills for undergraduate Information Systems students.

DISCUSSION AND CONCLUSIONS

This research focused on the aspect of students’ programming self-efficacy in order to assist educators construct class activities that support class materials. The factor analysis results showed that the latent variables of the construct Java programming self-efficacy were loaded in two factors and labeled as (1) independence and persistence in programming tasks, and (2) scaffolding for programming. The first research question addressed whether IS students’ perceived value of learning programming, their own practice time, and frequency of TA consultations associate with their programming self-efficacy. Both independence and persistence in programming tasks and scaffolding for programming (overall programming self-efficacy) showed that they are positively correlated with student perceived value of learning to program. This indicates that the more students value the learning to program, the more they become independent and persist in solving challenging programming problems. Relatively, once students receive appropriate support or scaffolding to learn programming, the more they value programming. This finding confirms what other studies found regarding how the value of learning programming is an important factor to learn programming (Blumenfeld et al., 2006; Wu et al., 2010; X. Zhang, 2013). Although the data as presented in the findings indicated that the participants of this study do value programming, the sources that assist them to gain this value

cannot be confirmed. We anticipate as a part of this study that students' value of learning programming can be obtained using a combination of various resources, such as meeting with guest speakers who graduated with IS degrees, watching interviews of successful people in programming from around the world, and linking course activities with IS projects funded by IT companies. Such activities will expose students to real world experience and competition to win competent programming jobs.

The finding of having a significant negative correlation between independence and persistence in learning programming and the amount of time of consulting TAs for help seemed logical. It indicates that the more students become independent in solving programming tasks, the less they need to consult TAs for assistance. However, the data also showed that there is no significant correlation between scaffolding and the amount of time consulting TAs for help. This finding does not indicate diminishing the importance of support that TAs provide while learning programming, but the IS students participated in this study might not consider TAs as the only source for help. The participants might receive help from their peers, from the help center provided by their college, and/or from any other individuals outside the class; therefore, we are looking forward to considering these external support factors in our continuous work.

The data results also showed that time spent to practice programming does not correlate with independence and persistence in programming tasks and scaffolding for programming (overall programming self-efficacy). The lack of the correlation was unexpected, which contradicts with what Özmen and Altun (2014) found in their study. Özmen and Altun concluded that there is a relationship between practice time and student programming self-efficacy, which in turn reduces the difficulty to learn programming. One possible explanation is that practice assignments were not restricted throughout this research, and students were not earnest in practice programming since practice examples were ungraded and un-trackable by the instructor or the TAs. In the current study, students were given these practice examples to work on outside the class on their own time; henceforth, we advise IS educators to make the practice time outside the class trackable and gradable task using an IS technology. Research showed that adopting interactive textbooks into introductory STEM (science, technology, engineering, and math) courses improves student coding skills (Edgcomb & Vahid, 2014, 2015; Edgcomb, Vahid, Lysecky, & Lysecky, 2017). With the help of the interactive programming textbook, we believe that IS educators will be able to track students' programming practice time and grade students' activities.

An investigation of the overall programming self-efficacy showed that IS students' programming self-efficacy perception levels are generally at medium level as presented in the findings. This finding is consistent with a similar conclusion drew from Korkmaz and Altun's study (2014). As programming skill is a critical part of IS careers (Konecki, 2014), we can confirm that the IS students' perceived level of programming self-efficacy is not appropriate to be competent programmers. This level is an indicator that students generally are not confident in designing and implementing a complete software solution. This requires collecting more data to investigate other factors that potentially contribute to IS students' programming self-efficacy, such as computer programming background and previous math exposure, motivation, and economic status.

The second research question addressed IS students' expectations about how their belief on programming levels conforms to their learning goals. Based on the qualitative findings of this study, the results showed that there is a new category that emerged for IS students beside Chilana et al.'s (2015) non-Computer Science programmers' four categories. Data distribution of IS students participated in this research was grouped into five programmer categories: novice, communicator, end-user, competent, and professional. A high percentage ($N = 54$ or 43%) of the participants' responses showed that IS students think that they need to be competent programmers in order to succeed in their future profession. The new category of competent programmers supports what Barker (2002) articulated. Barker called for the necessity to have programming skills higher than skills required for being end-user programmers; therefore, IS students need to be competent when they will need to write a

complete code (e.g., complete a program or website) but not as advanced as CS programmers. According to Heinlein (1987), a competent IS student programmer should be able to architect, implement, debug, and customize software applications, manage, market, and document software projects; and ultimately possess a dual role with technical and business orientation.

Participants' responses showed that three groups of students ($N = 59$ or 46%) see themselves in the future as novice, communicator, or end-user programmers. We suspect that these three categories are the groups who struggle with learning programming because, as Rogerson and Scott (2010) described, the difference between the students' expectation of what to learn and the class requirements is most likely the source of struggle in learning programming. Moreover, based on the current projection by the U.S. Department of Labor (2018), IS students in these three categories do not possess needed competent programming skills to meet the employment specification in the IS job market.

Interestingly, we did not expect to have some IS students ($N = 11\%$) who desired to be professional in programming, because to be a professional in programming, CS is the main major that could provide such skill. The CS degree major is math heavy at the undergraduate level that trains students to be professional in programming. Therefore, this group of students might have a misunderstanding of what would take to be professional programmers, which necessitates the need to focus not only on programming, but also to focus on the underlying algorithms and data structure that make codes work; nevertheless, this not compatible with what IS students need.

IS students' programming self-efficacy perception at medium level is inconsistent with students' programming expectations. IS students who possess medium self-efficacy levels are often programming at the competent level (neither at the end-user or professional levels). We anticipate that this finding necessitates IS educators to increase students' programming self-efficacy by providing the class activities that support the programming skill level consistent with students' expectations. Thus, we urge IS educators, as the ones who know their students the most, need to clearly state verbally and in the class syllabus about the IS course objectives and learning outcomes as well as how much time and effort are expected from students to spend to learn programming.

It would be much easier for educators if IS students programming skills fall into a specific category, which in turn would assist in creating uniform programming courses across a variety of IS specialties but based on our finding of having different programming categories of IS students, the variety of skills needed among these technology users poses a significant challenge to instructional planning. As a result of this variation in learning needs, teaching programming in a realistic context might be a useful catalyst to the needed changes in student attitudes and perceptions of learning to program and programming instruction, or a contextual teaching approach (Hudson & Whisler, 2007). Contextual teaching would provide IS students with opportunities to learn and practice the skills they will be expected to use in their future careers, which supports both academic and career success.

Based on the contextual learning approach, IS educators may create programming courses that differ across specialties so that programming or coding classes are specific to each specialty based on students' future respective domains. Therefore, coordination between IS educators and career-specific decision makers that identify necessary programming knowledge and skills for various career domains is critical to producing instructional programs that are tailored to meet actual career demands. This type of coordination will help educators reduce programming content that students are unlikely to need in their future careers. Establishing greater career relevance in programming courses is likely to increase the value that students place on learning programming skills. To the degree that valuing programming is related to higher programming self-efficacy, contextual learning may encourage students to learn programming and indirectly reduce fears about learning to program because contextual learning offers high levels of relevance and actual skill practice.

LIMITATIONS

Relative to generalizing the findings, the sample size could be considered relatively small. This can be referred to the normal setting in many universities, which is in our case a limited number of undergraduate IS students who were attending the two programming courses offered by their departments. However, this sample size was statistically appropriate to run the designated statistical and factor analysis for this study. Additionally, a self-report survey was used to obtain the data; therefore, the results could be influenced by the students' social willingness to provide desirable information rather than their honest responses. Nevertheless, surveys are one of the most appropriate quantitative research methods, as they tend to identify "trends in attitudes, opinions, behaviors, or characteristics of a large group of people" (Creswell, 2012, p. 21).

RECOMMENDATIONS

We conclude by offering some recommendations for future research. First, an interactive programming tool is a suggested supplement for IS educators to increase student enthusiasm about practice time of programming that would influence their programming self-efficacy. Second, this research can be replicated by using the same methodology with participants from other universities and use various high-level programming languages. This would contribute in making the findings more generalizable. Third, this study mainly focused on self-efficacy and its related variables discussed in the literature that assist IS educators growing their undergraduate IS students programming competency. Future research could enrich the use of the survey by incorporating individual interviews, focus group interviews, and/or classroom observations to provide a more comprehensive understanding of IS students' programming acquisition.

REFERENCES

- Askar, P., & Davenport, D. (2009). An investigation of factors related to self-efficacy for java programming among engineering students. *TOJET: The Turkish Online Journal of Educational Technology*, 8(1), 26-32. Retrieved from <http://files.eric.ed.gov/fulltext/ED503900.pdf>
- Bandura, A. (1977). Self-efficacy: Toward a unifying theory of behavioral change. *Psychological Review*, 84(2), 191-215. <https://doi.org/10.1037//0033-295x.84.2.191>
- Bandura, A. (1986). *Social foundations of thought and action: A social cognitive theory*. Englewood Cliffs, NJ: Prentice Hall. <https://doi.org/10.5465/amr.1987.4306538>
- Bandura, A. (1988). Organizational applications of social cognitive theory. *Australian Journal of Management*, 13(2), 275-302.
- Barker, S. K. (2002). Training business students to be end-used developers: Are case studies the best option? *Issues & Trends of Information Technology Management in Contemporary Organizations*, 1, 62-69.
- 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>
- Baser, M. (2013). Attitude, gender and achievement in computer programming. *Online Submission*, 14(2), 248-255.
- 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), 413-437. <https://doi.org/10.1007/s40692-016-0073-2>
- Blumenfeld, P., Kempler, T., & Krajcik, J. (2006). Motivation and cognitive engagement in learning environments. In K. Sawyer (Ed.), *Cambridge handbook of the learning science* (pp. 475-488). New York: Cambridge University Press. <https://doi.org/10.1017/cbo9780511816833.029>
- Boehm, B. W., Horowitz, E., Madachy, R., Reifer, D., Clark, B. K., Steece, B., Brown, A. W., Chulani, S., & Abts, C. (2000). *Software cost estimation with COCOMO II*. Upper Saddle River, NJ: Prentice Hall PRT.

Enhancing Programming Self-efficacy

- Büyüköztürk, Ş. (2002). *Sosyal bilimler için veri analizi el kitabı* [Data analysis handbook for social sciences]. Ankara: PegemA Press. <https://doi.org/10.14527/9789756802748>
- Chilana, P. K., Alcock, C., Dembla, S., Ho, A., Hurst, A., Armstrong, B., & Guo, P. J. (2015, October). Perceptions of non-CS majors in intro programming: The rise of the conversational programmer. *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'15)*, 251-259. <https://doi.org/10.1109/vlhcc.2015.7357224>
- 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.
- Creswell, J. W. (2012). *Educational research: Planning, conducting, and evaluating quantitative and qualitative research* (4th ed.). Upper Saddle River, NJ: Pearson/Prentice Hall.
- Dang, Y. M., Zhang, Y. G., Ravindran, S., & Osmonbekov, T. (2016). Examining student satisfaction and gender differences in technology-supported, blended learning. *Journal of Information Systems Education*, 27(2), 119-130.
- Dasuki, S., & Quaye, A. (2016). Undergraduate students' failure in programming courses in institutions of higher education in developing countries: A Nigerian perspective. *The Electronic Journal of Information Systems in Developing Countries*, 76(1), 1-18. <https://doi.org/10.1002/j.1681-4835.2016.tb00559.x>
- Dreyfus, H. L., & Dreyfus, S. E. (1986). *Mind over machine: The power of human intuition and expertise in the era of the computer*. New York, NY: Free Press.
- Driscoll, M. P. (2005). *Psychology of learning for instruction*. Boston: Pearson Allyn and Bacon.
- Edgcomb, A., & Vahid, F. (2014). Effectiveness of online textbooks vs. interactive web-native content. In *2014 ASEE Annual Conference*.
- Edgcomb, A., & Vahid, F. (2015). How many points should be awarded for interactive textbook reading assignments? In *Frontiers in Education Conference (FIE), 2015 IEEE* (pp. 1-4). IEEE. <https://doi.org/10.1109/fie.2015.7344350>
- Edgcomb, A., Vahid, F., Lysecky, R., & Lysecky, S. (2017, March). Getting students to earnestly do reading, studying, and homework in an introductory programming class. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 171-176). ACM. <https://doi.org/10.1145/3017680.3017732>
- Fasogbon, S. K., Jegede, P. O., Adetan, D. A., & Aderbigbe, A. A. (2016). Assessment of Java programming self-efficacy among engineering students in a typical Nigerian university. *African Journal of Sustainable Development*, 6(2), 173-187.
- Field, A. (2009). *Discovering statistics using SPSS statistics* (3rd ed.). Thousand Oaks, CA: Sage publications.
- Field, A. (2013). *Discovering statistics using IBM SPSS statistics* (4th ed.). Thousand Oaks, CA: Sage publications.
- Forte, A., & Guzdial, M. (2005). Motivation and non-majors in CS1: Identifying discrete audiences for introductory computer science. *IEEE Transactions on Education*, 48(2), 248-253. <https://doi.org/10.1109/te.2004.842924>
- Gall, M. D., Borg, W. R., & Gall, J. P. (1996). *Educational research: An introduction* (6th ed.). United Kingdom: Longman Publishing.
- 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. <https://doi.org/10.5465/amr.1992.4279530>
- Hudson, C. C. & Whisler, V. R. (2007). Contextual teaching and learning for practitioners. *Journal of Systems, Cybernetics and Informatics*, 6(4) 54-58. Retrieved from [http://www.iijsci.org/journal/cv\\$/sci/pdfs/e668ps.pdf](http://www.iijsci.org/journal/cv$/sci/pdfs/e668ps.pdf)
- Heinlein, R. A. (1987). *Time enough for love*. Penguin.
- Kinnunen, P., & Malmi, L. (2006). Why students drop out CS1 course? In *Proceedings of the Second International Workshop on Computing Education Research* (pp. 97-108). ACM. <https://doi.org/10.1145/1151588.1151604>

- Konecki, M. (2014). Problems in programming education and means of their improvement. In B. Katalinic (Ed.), *DAAAM international scientific book 2104* (pp. 459-470). <https://doi.org/10.2507/daaam.scibook.2014.37>
- Korkmaz, Ö., & Altun, H. (2014). Adapting computer programming self-efficacy scale and engineering students' self-efficacy perceptions. *Online Submission*, 1(1), 20-31. <https://doi.org/10.17275/per.14.02.1.1>
- Lieberman, H., Paterno, F., Klann, M., & Wulf, V. (2006). End-user development: An emerging paradigm. In H. Lieberman, F. Paterno, & V. Wulf (Eds.), *End-user development* (pp. 1-8), Dordrecht: Springer. https://doi.org/10.1007/1-4020-5386-x_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*. Valencia, Spain.
- Özmen, B., & Altun, A. (2014). Undergraduate students' experiences in programming: Difficulties and obstacles. *Turkish Online Journal of Qualitative Inquiry*, 5(3), 1-27. <https://doi.org/10.17569/tojqi.20328>
- Pajares, F. (1996). Self-efficacy beliefs in academic settings. *Review of Educational Research*, 66(4), 543-578. <https://doi.org/10.3102/00346543066004543>
- Pallant, J. (2007). *SPSS survival manual* (3rd ed.). New York: Open University Press.
- Redmond, B. F. (2010, December 6). Self-efficacy and social cognitive case study [Blog post]. Retrieved from: <https://wikispaces.psu.edu/display/PSYCH484/7.+Self-Efficacy+and+Social+Cognitive+Theories#id-7.Self-EfficacyandSocialCognitiveTheories-MeasuringSelf-Efficacy>
- Robinson, M. A. (2018). Using multi-item psychometric scales for research and practice in human resource management. *Human Resource Management*, 57(3), 739-750. <https://doi.org/10.1002/hrm.21852>
- Rogerson, C., & Scott, E. (2010). The fear factor: How it affects students learning to program in a tertiary environment. *Journal of Information Technology Education*, 9(1), 147-171. <https://doi.org/10.28945/1183>
- Rosson, M. B., Carroll, J. M., & Sinha, H. (2011). Orientation of undergraduates toward careers in the computer and information sciences: Gender, self-efficacy and social support. *ACM Transactions on Computing Education (TOCE)*, 11(3), 14. <https://doi.org/10.1145/2037276.2037278>
- Tan, P. H., Ting, C. Y., & Ling, S. W. (2009). Learning difficulties in programming courses: undergraduates' perspective and perception. *International Conference on Computer Technology and Development*, 42-46. <https://doi.org/10.1109/icctd.2009.188>
- U.S. Department of Labor Bureau of Labor Statistics. (2018). *Occupational outlook handbook, 2017-2018 edition*. U.S. Department of Labor, Washington, D. C. Retrieved from <https://www.bls.gov/ooh/computer-and-information-technology/home.htm>
- Vihavainen, A., Paksula, M., & Luukkainen, M. (2011, March). Extreme apprenticeship method in teaching programming for beginners. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (pp. 93-98). ACM. <https://doi.org/10.1145/1953163.1953196>
- Wiedenbeck, S. (2005, October). Factors affecting the success of non-majors in learning to program. In *Proceedings of the First International Workshop on Computing Education Research* (pp. 13-24). ACM. <https://doi.org/10.1145/1089786.1089788>
- Wiedenbeck, S., Labelle, D., & Kain, V. N. (2004, April). Factors affecting course outcomes in introductory programming. In *16th Annual Workshop of the Psychology of Programming Interest Group* (pp. 97-109).
- Wiggins, J. B., Grafsgaard, J. F., Boyer, K. E., Wiebe, E. N., & Lester, J. C. (2016). 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. <https://doi.org/10.1007/s40593-015-0091-7>
- Wu, J. H., Tennyson, R. D., & Hsia, T. L. (2010). A study of student satisfaction in a blended e-learning system environment. *Computers & Education*, 55(1), 155-164. <https://doi.org/10.1016/j.compedu.2009.12.012>

- Zhang, X., Zhang, C., Stafford, T. F., & Zhang, P. (2013). Teaching introductory programming to IS students: The impact of teaching approaches on learning performance. *Journal of Information Systems Education*, 24(2), 147-155.
- Zhang, Y. G., & Dang, Y. M. (2015). Investigating essential factors on students' perceived accomplishment and enjoyment and intention to learn in web development. *ACM Transactions on Computing Education (TOCE)*, 15(1), 3:1-3:21. <https://doi.org/10.1145/2700515>

APPENDIX

PART I: JAVA PROGRAMMING TASKS

Please rate your confidence in doing the following Java programming related tasks using a scale of (1-not at all confident; 2-mostly not confident; 3- Neither confident nor unconfident; 4-mostly confident;5-absolutely confident)

1. I could write syntactically correct Java statements.
2. I could understand the language structure of Java and the usage of the reserved words.
3. I could write logically correct blocks of code using Java.
4. I could write a Java program that displays a greeting message.
5. I could write a Java program that computes the average of three numbers.
6. I could write a Java program that computes the average of any given number of numbers.
7. I could use built-in functions that are available in the various Java swing GUIs.
8. I could build my own Java swing GUIs.
9. I could write a small Java program given a small problem that is familiar to me.
10. I could write a reasonably sized Java program that can solve a problem this is only vaguely familiar to me.
11. I could write a long and complex Java program to solve any given problem as long as the specifications are clearly defined.
12. I could organize and design my program in a modular manner.
13. I could understand the object-oriented paradigm.
14. I could identify the objects in the problem domain and could declare, define, and use them.
15. I could make use of a pre-written function, given a clearly labeled declaration of the function.
16. I could make use of a class that is already defined, given a clearly labeled declaration of the class.
17. I could debug (correct all the errors) a long and complex program that I had written and make it work.
18. I could comprehend a long, complex multi-file program.
19. I could complete a programming project if someone showed me how to solve the problem first.
20. I could complete a programming project if I had only the language reference manual for help.
21. I could complete a programming project if I could call someone for help if I got stuck.
22. I could complete a programming project once someone else helped me get started.
23. I could complete a programming project if I had a lot of time to complete the program.
24. I could complete a programming project if I had just the built-in help facility for assistance.
25. While working on a programming project, if I got stuck at a point I could find ways of overcoming the problem.

26. I could come up with a suitable strategy for a given programming project in a short time.
27. I could manage my time efficiently if I had a pressing deadline on a programming project.
28. I could mentally trace through the execution of a long, complex multi-file program given to me.
29. I could rewrite lengthy and confusing portions of code to be more readable and clearer.
30. I could find a way to concentrate on my program, even when there were many distractors around me.
31. I could find ways of motivating myself to program, even if the problem area was of no interest to me.
32. I could write a program that someone else could comprehend and add features to at a later date.

PART II: PERCEIVED VALUE OF LEARNING PROGRAMMING

Please indicate to what degree you agree with each of the following statements using a scale of (1 - strongly disagree; 2 – somewhat disagree; 3 - Neither agree nor disagree; 4 – somewhat agree; 5 - strongly agree)

1. Learning programming helps me not only in my academia, but also to solve problems in my daily life.
2. Learning programming in one class increases my productivity in other programming classes.
3. Learning programming improves the value of my degree.
4. Learning programming provides me a high chance to get a job quickly.
5. Learning programming increases the chance of getting high payment in my future career.

PART III: FUTURE USE OF PROGRAMMING

1. Explain how you view programming as being a part in your future profession (e.g., end-user programmer, professional programmer, or anything else)?

PART IV: DEMOGRAPHIC INFORMATION

1. What is your age?
2. What is your gender? (Male/Female)
3. Which class are you in? (Application Design and Development (CIS 240)/
4. Advanced Application Design and Development (CIS 340)
5. How many hours do you spend per week to practice the weekly assigned
6. programs by the instructor?
7. How many times in this semester did you consult the teaching assistant for
8. help in your assignments?

BIOGRAPHIES



Ramadan Abdunabi is a Clinical professor of Computer Information Systems in the College of Business at Colorado State University. He received his MCS and Ph.D. in Computer Science from Colorado State University, USA. His research interests include software design and development, computer networks and security, Information Systems education: pedagogy, curriculum design & implementation, distance education challenges, and technology impact.



Ilham Hbaci has recently earned her Ph.D. in Educational Technology with a minor in Applied Statistics and Research Methods from University of Northern Colorado, Greeley, USA. She earned her Master of Business Administration (MSBA)-Computer Information System Department-Business School at Colorado State University, Fort Collins, USA. Her research interest is technology integration, teaching and learning strategies, educational qualitative and quantitative research, in general.



Heng-Yu Ku is a Professor in the College of Education and Behavioral Sciences at the University of Northern Colorado, Greeley, CO, 80639, USA. His research interests include technology integration, teaching and learning strategies, and distance education.